THE DEVELOPMENT OF A MATHEMATICAL PROGRAMMING TECHNIQUE AS A DESIGN TOOL FOR TRAFFIC MANAGEMENT

PATRICIA ANNE WACKRILL

A thesis submitted in partial fulfilment of the requirements of the Council for National Academic Awards for the degree of Doctor of Philosophy

June 1990

Middlesex Polytechnic in collaboration with
MVA Systematica

ACKNOWLEDGEMENTS

I acknowledge with gratitude both help and support from the following people and organisations. The project was conceived by Prof. C. Wright and Dr. G. Appa but without this support it might never have been undertaken.

1. A research grant from the Science and Engineering Research Council for two years.

2. Financial support from the Faculty of Engineering, Science, and Mathematics at Middlesex Polytechnic and help from several members of staff.

3. Software from the collaborating establishment MVA Systematica and advice from Mr. M. Logie.

4. The use of software, and the invitation to attend research seminars, from the London School of Economics.

5. Data on the Hazel Grove network from the Institute of Transport Studies in the University of Leeds and help from Dr. T. van Vuren with its interpretation. Permission to quote the analysis of results with this data from the Director of Works at Stockport M.B.C.

6. Discussions with Dr. M.Maher and colleagues at the Transport and Road Research Laboratory.

7. Discussions with Dr. M.Bell and colleagues at Nottingham University Transport Research Group.

8. Continual guidance and encouragement from the Director of Studies Dr. G. Appa.

9. The inspiration necessary to persevere in completing the thesis came largely from the Internal Supervisor Prof. C. Wright.

10. Prof. A Land first introduced me to the Out-of-Kilter algorithm and its applications. She also suggested the approach using solutions to a series of ILP problems. Her comments, as External Supervisor, on the preparation of the thesis were most helpful.

11. Unfailing support and encouragement, especially in debugging programs, from my husband Bruce Wackrill.

THE DEVELOPMENT OF A MATHEMATICAL PROGRAMMING TECHNIQUE AS A DESIGN TOOL FOR TRAFFIC MANAGEMENT.

P. A. WACKRILL

ABSTRACT

In urban areas, competition for road space at junctions is one of the major causes of congestion and accidents. Routes chosen to avoid conflict at junctions have a mutually beneficial effect which should improve circulation and reduce accidents. A prototype design tool has been developed to provide for traffic management based on such routes.

The mathematical model behind the design tool works with a given road network and a given O-D demand matrix to produce feasible routes for all drivers in such a way that the weighted sum of potential conflicts is minimised. The result is a route selection in which all journeys from origin i to destination j follow the same route.

The method which works best splits the problem into single commodity problems and solves these repeatedly by the Out-of-Kilter algorithm. Good locally optimal solutions can be produced by this method, even though global optimality cannot be guaranteed. Software for a microcomputer presented here as part of the design tool is capable of solving problems on realistic networks in a reasonable time.

This method is embedded in a suite of computer programs which makes the input and output straightforward. Used as a design tool in the early stages of network design it gives a network-wide view of the possibilities for reducing conflict and indicates a coherent set of traffic management measures. The ideal measure would be automatic route guidance, such as the pilot scheme currently being developed for London. Other measures include a set of one-way streets and banned turns. The resulting turning flows could be used as input to the signal optimiser TRANSYT to determine signal settings favouring the routeing pattern.

The project was funded by the S.E.R.C. and carried out at Middlesex Polytechnic in collaboration with MVA Systematica.

CONTENTS

CHAPTER 1

INTRODUCTION

"Where there is no vision the people perish."

Proverbs 29:18.

1.1 THE VISION

Through the centuries man has had visions of how he could improve the way he organized his life. Usually these visions have been concerned with increased efficiency in the use of scarce or expensive resources of energy or materials. Recently the emphasis has shifted to combine considerations of increased efficiency with limited damage to the environment.

The particular vision which inspired this project concerns the improvement of traffic circulation in urban areas. Traffic is unable to circulate freely because the amount of road space available has not kept up with the demand for it. The aim of traffic management is to facilitate circulation while paying due attention to safety and environmental considerations. This aim can be achieved by several means.

First, the capacity of the road network to accommodate the flow of traffic can be increased by building new roads. The capacity of existing roads where on-street parking occurs can be increased by restricting that parking. Bottlenecks often occur at junctions; traffic signals regulate the flow through a junction, so

that it can be used more efficiently. The signal settings can be optimised either for isolated junctions, or for a whole set of co-ordinated junctions, so that the green time allotted to each stream of traffic is used more efficiently.

Second, demand can be curbed by the legal enforcement of traffic restrictions. Various methods have been tried; in Britain access is denied to heavy goods vehicles at certain times and places. One might expect congestion, which pushes up the cost of travel, to curb demand, but demand continues to grow. Road pricing can be used to deter drivers from using the network at the most congested times and places. A scheme to implement road pricing is at an advanced stage in the Netherlands (Stoelhorst and Zandbergen 1990).

Third, the driver can be encouraged to satisfy his demand for a route to his destination in such a way that traffic circulates more efficiently. At present this is done both directly and indirectly, but in rather a piecemeal fashion. Direct guidance is given by signposts to bypasses and ring roads, and by one-way streets and restricted access. In this way the traffic manager succeeds in diverting some of the traffic away from congestion blackspots. The efficiency of the one-way gyratory system is, however, dependent on the distribution of traffic between the various routes through it. An

example, where efficiency was improved by reverting to two-way circulation is given by Wright and Semmens (1984). Indirect guidance is given by changed traffic conditions in parts of the network; it has been found that drivers respond to changes in signal settings which favour certain streams of traffic (Allsop and Charlesworth 1977), and to other changes affecting road capacity. The decisions to guide traffic in these ways are, however, made on an ad hoc basis, what one might call a 'bottom-up' approach to traffic management.

The design tool developed in this project starts with the whole network and the demand for routes through it, and finds a coherent routeing pattern to facilitate circulation. It will be referred to as the CROWN design tool because the method used achieves 'Conflict Reduction Over a Wide Network'. The word crown is associated with the top; the CROWN design tool provides for a top-down approach to the design of traffic management measures. This is a new approach to facilitating circulation, by means of a more efficient routeing pattern.

Circulation would be improved because, in the urban situation, the delays caused by streams of traffic competing for road space at junctions are the primary symptoms of congestion. A routeing pattern chosen for its relatively low level of conflict will therefore reduce that competition. If the engineer actually knows the

routeing pattern he wishes to encourage, he can design his local controlling measures to encourage that pattern. The CROWN design tool identifies one-way streets and banned turns consistent with the routeing pattern. It also shows the flows arising from that pattern; these flows could be used as input to a signal setting optimisation program such as TRANSYT. If signals were set on this basis, road users would discover that green time was relatively longer for the manoeuvres favoured by the routeing pattern. Automatic route guidance (AUTOGUIDE) is being developed as a pilot project for the London area; the guidance is to be in the form of in-vehicle advice to the driver rather than physical guidance. AUTOGUIDE would provide the ideal means for encouraging the use of the routeing pattern devised by the CROWN design tool. The idea of routes chosen to minimise conflict is not new; its development is traced in the next section.

1.2 THE VISION DEVELOPED

When paths have to be laid out between fixed points, it may be desirable that these paths cross as little as possible. This was the case in Turan's brick factory, where the bricks were transported between various kilns and storage yards by rail. Where the rails crossed the trucks were likely to be derailed (Turan 1977).

Holroyd and Miller (1966) considered the desirability of finding routes through a town so that the number of path crossings was minimised. They analysed the statistical properties of the number of path crossings in idealised grid networks. They pointed out the contrast between the effects of choosing a route to minimise journey time and a route to minimise the number of path crossings encountered. In the first case, the total journey time for the group of drivers as a whole may actually be increased by the choice. In the second case, the total number of path crossings in the system will be reduced by the action of each driver avoiding such path crossings. They also developed methods for laying out paths so as to minimise crossings in simple regular networks.

Wright (1978) took up this theme in his vision of alleviating congestion by imposing route choice on drivers. Wright, Appa and Jarrett (1989) explored ways of tracing paths through idealised networks to minimise the number of crossings. Further exploration was deemed to require a computer program to find such routes in any given network. Appa set the development of such a program as a student project in the final year of the Mathematics for Business degree course at Middlesex Polytechnic in 1986. This thesis describes the project which grew directly out of that student project carried out by Large.

1.3 THE VISION REFINED

An algorithm for finding routes to minimise crossings will require some means of recognising path crossings. The road network has to be modelled so that one can tell whether the paths of the vehicles using a pair of routes have to cross or not. There are two separate cases to consider. The two paths may enter a junction by different approaches, cross each other and leave by different exits. This case is easy to detect. Alternatively the two paths may enter a junction by different approaches and leave by the same exit, staying together until they diverge at a subsequent junction. A very intricate model would be required to distinguish between such paths which diverged without actually crossing and those which did cross. Practical considerations come to the rescue here.

It is the necessity for merging in order to leave the junction that is significant as far as competition for road space is concerned. Even if there are two lanes, the choice of lane will not be governed by whether the vehicles are following paths which must cross. Any lane changing necessary for the junction at which crossing, or indeed non-crossing, paths diverge depends on the driver's preferred lane, rather than his entry to it in the first place.

The vision is therefore refined to one in which the number of crossings and mergings at junctions is minimised. Paths which cross and use the same road between junctions will be counted in these mergings. Paths which merge on leaving a junction without actually having to cross will also be counted in these mergings. This refined vision is defended in the next section.

1.4 THE VISION DEFENDED

The criterion for route choice to be defended is that the number of crossings and mergings at junctions should be minimised. If one defines conflicting streams as those which cross or merge, this number will be the sum of the products of the flows in each pair of conflicting streams. It will be referred to as the amount of conflict. The product of flows features in the first order approximations of the calculations in queueing theory; the expected frequency with which two vehicles arrive at a junction so as to compete for road space is proportional to the product of flows in their streams. Reduction in the expected number of such competitions should be beneficial on three counts.

Such a competition is usually resolved by one competitor giving way to the other. However the giving way is regulated, it is a cause of delay. This delay adds to the journey time; compared with the rural situation, it has a larger effect than the distance travelled. When

a particular stream features in several pairs of conflicting streams, realism may be served by weighting the different types of conflict to reflect the likely delay. If routes are chosen to minimise the amount of conflict, one should reap the benefit of a reduction in the total delay at junctions.

Alternatively, the competition is not resolved and therefore results in an accident. This rare result is also likely to vary with the number of competitions. Reduction in this number should be beneficial in reducing the likelihood of accidents. Recent studies, made at the Transport and Road Research Laboratory, have resulted in the development of accident predictive relations in terms of products involving powers of the flows in conflicting streams (Summersgill 1988). The CROWN design tool could be fairly easily modified to incorporate these relations so as to reflect the potential for accidents more accurately (Wackrill 1990).

The third beneficial effect concerns the environmental consideration of air and noise pollution. Bell (1990) asserts that the level of noise and air pollution would be reduced if the amount of conflict at junctions was reduced.

This criterion also has merit in comparison to two others commonly used for route choice. The criteria with

which it will be compared are Wardrop's criteria (Wardrop 1952). His first criterion, that the journey times on all the routes actually used are equal, and less than those which would be experienced by a single vehicle on any unused route, leads to a user-equilibrium. The effects of congestion on journey time have to be taken into account and this makes algorithms to assign routes according to this criterion quite complicated. The time for each link of the journey may be dependent not only on the volume of traffic using that link but also, through the delay caused by queueing at junctions, on the volumes of traffic passing through junctions downstream of the direction of travel. The conventional way of dealing with such a minimisation problem is to use approximate link times to prime an iterative process in which assignment to minimum time routes alternates with the computation of the time for each link. It can easily happen that a particular origin to destination route oscillates between paths; each becomes more congested, and therefore less attractive in the subsequent assignment, so this conventional solution method may run into difficulties. These difficulties can be overcome by such methods as incremental assignment, in which link times are recomputed after successive proportions of the traffic have been loaded onto the network.

Wardrop's second criterion, that the average journey time is a minimum, leads to a system optimum rather than a user-optimum. Holroyd and Miller (1966)

illustrate the possible difference between the two with the following example. Consider two roads between an origin and a destination where the journey time on each road is partly constant and varies partly as the traffic flow on that road. Suppose the demand for trips between the origin and destination is 12 and the flow is divided into Q1 drivers using Road 1 and Q2 drivers using Road 2 so that

$$Q1 + Q2 = 12.$$

Suppose also that the journey times are respectively,

$$T1 = 6 + Q1 \text{ and } T2 = 12 + 2Q2.$$

Then the total journey time of all the drivers, Q1*T1 + Q2*T2, is a minimum, 189, when Q1 is 9 and Q2 is 3; whereas if individual drivers minimise their own journey time, Q1 is 10 and Q2 is 2, giving a total journey time of 192 for all the drivers.

The traffic engineer prefers the second criterion while the driver prefers the first.

The criterion of minimum conflict has the property that the system optimum will also be a user-optimum. If it were not a user-optimum, then a route with less conflict could be found which would reduce the total number of

conflicts from the hypothetical minimum. In the urban situation, this criterion is closely correlated to journey time. Drivers might well find routes chosen according to this criterion an acceptable alternative to the routes they would choose for themselves, and thus find their aims coinciding with those of the traffic engineer.

Compared with Wardrop's criteria, the criterion of minimum conflict is relatively easy to apply. It takes account of the interaction between streams of traffic at junctions, using the details of the topology of the road network, without the need for details of the geometry of the junctions. Engineers would like to address the problem of junctions but do not always have the resources for collecting the data required for a conventional model. The CROWN design tool would enable them to take some account of junction conflicts, but with relatively simple input. Such a tool has been lacking from the traffic manager's toolkit (Boyce 1988).

1.5 THE VISION IN DETAIL

The routeing patterns chosen so that the total amount of weighted conflict at junctions is minimised are found by means of a computer program. The CROWN design tool is a suite of three computer programs. The input consists of three data files. One contains the data which specifies the road network for which an efficient routeing pattern is desired. The network could be an existing one,

or an existing one modified by proposed changes, or an idealised one for a green field site. A second data file is a trip matrix indicating the demand for paths between various origins and destinations on the road network. The third data file, which is optional, specifies the weighting factors to reflect the relative danger and delay arising from different pairs of conflicting manoeuvres at different types of junction; this latter file need not be specific to the network. To use the tool, one runs the suite of programs with the appropriate input files. The output specifies the routeing pattern.

A secondary function of the suite of programs is to show the effect, in terms of volumes of traffic, of the resulting routeing pattern on the network. Some links will be used in one direction only; these indicate streets, which, if they were made into one-way streets, would reinforce the routeing pattern. Some permitted junction manoeuvres will not be used at all; these manoeuvres indicate turns which could be banned to reinforce the routeing pattern.

1.6 THE OBJECTIVE: TO REALISE THE VISION

The objective of this project was to realise the vision in the form of a computer program developed to the point where its use could be demonstrated. Refinement of the code to professional standards would be left to the

software house benefiting from the commercial exploitation of the tool. The objective has been achieved in nine stages.

1) A way of modelling the road network in which conflicting manoeuvres at junctions could be identified was developed.

2) The objective function, to be minimised, subject to flow conservation constraints, was defined in terms of conflicting flows on this model network.

3) Mathematical programming techniques for solving this kind of constrained minimisation problem were investigated.

4) A satisfactory technique was selected and adapted for use in solving the problem.

5) Preliminary tests on small networks were carried out to validate the procedure.

6) In order to test the procedure with larger networks, an algorithm was designed to create the details of the model network, to identify conflicting manoeuvres and to match them to appropriate weights.

7) An algorithm was then designed to translate results, obtained in terms of the model network, into terms relating to the road network.

8) These two algorithms and the selected solution technique were embedded in a suite of computer programs written in the FORTRAN 77 language.

9) The suite was tested with various networks to assess performance.

The description of the completed project begins with the model network, because the problem to find a routeing pattern is defined in terms of it. This model network is an elaboration of the road network; algorithms were designed to translate from one to the other. The model and the algorithms are described in Chapter 2. The problem is defined and solution methods are investigated in Chapter 3. The fine details of the solution method chosen are given in Chapter 4. The Out-of-Kilter algorithm, adapted and used as a subroutine in our solution method, is described in detail in Chapter 5, so that the adaptations can be explained. The suite of programs which constitute our design tool is described in Chapter 6. The performance of the tool is assessed in terms of results with test networks in Chapter 7. Conclusions are drawn in Chapter 8.

CHAPTER 2

A NETWORK MODEL WHICH SHOWS CONFLICTING MANOEUVRES

The purpose of this chapter is to show how conflict between streams of traffic at a junction is quantified so that it can be used as a criterion for route selection. The network has to be specified in a particular way to make this possible. This particular way of specifying the network will be referred to as the 'Circulation System', the term used by Wright, Appa and Jarrett (1989), to contrast it with references to the road network. The specification of the Circulation System will be described in Section 2.1. The industrial collaborators MVA Systematica see the essential difference between the road network and the Circulation System in terms of trees and what they call 'vines'. The significance of this difference is explained in Section 2.2. We need to be able to identify the streams which conflict with a given stream at a junction; the way we use a junction model to make this possible is described in Section 2.3. A driver in a particular stream at a junction may experience conflict with traffic in the conflicting streams. The way this potential conflict is quantified so that it can be used as the cost to the driver of making each manoeuvre will be explained in Section 2.4. An algorithm was devised to synthesise the details of the Circulation System from details of the road network. A program POLYARCS was written incorporating

this algorithm but also performing other relevant functions. The program's functions are described in Section 2.5. In order to explain how the algorithm to synthesise the Circulation System works, the way the details of the road network are specified has to be defined; this is done in Section 2.6. The steps of the algorithm can then be described in Section 2.7.

2.1 SPECIFICATION OF THE CIRCULATION SYSTEM

Consider what the proverbial man in the street might say when asked for directions to some nearby destination. He might say "First left then second right". He is specifying the route in terms of the required movements at the next three junctions. This is just the way we need to express each route, if we are to identify the streams of traffic which conflict with each other on a network.

In graph theory this way of expressing a route would be described as a path in a directed graph or "digraph". A digraph is specified as a list of vertices and a list of ordered pairs of distinct vertices. The ordered pairs are called "arcs" to distinguish them from the unordered pairs which are the edges in a ordinary graph. A path would then be specified by an ordered list of arcs in the form uv, vw, wx, xy, yz.

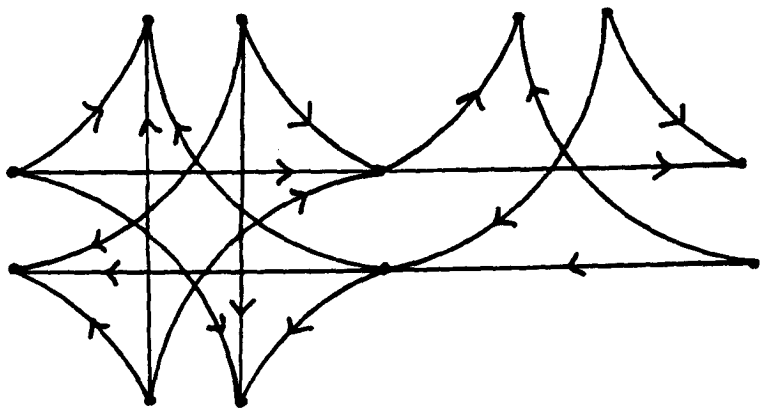
The image used by that man in the street so foreshortens the road between junctions that leaving one junction is practically identified with arriving at the next. Thus for our purposes, the first vertex in the ordered pair of vertices, defining an arc, corresponds to a particular approach to a junction, and the second vertex corresponds to a particular exit, so that movement along an arc represents a particular manoeuvre at a junction. Once one has left a junction, using a particular exit, both the next junction that one will encounter, and the approach on which one will arrive, are already determined, so the same vertex can be used to represent both the exit from one junction and the approach to the next junction.

We illustrate the correspondence between the road network and the Circulation System in Figure 1. Arrows on the arcs assume that road users drive on the left; we will assume driving on the left throughout this thesis. The CROWN design tool can be used by traffic managers in countries where road users drive on the right; junction details would have to be entered in a different order which will be described in Section 2.6.4.

It is obvious that the Circulation System is much more complicated than the road network. One can sketch the Circulation System and then specify it in a form suitable for input to a computer. However, for all but very simple road networks, the sketches become so extensive that the effort required would be prohibitive.

It seemed essential therefore, to design an algorithm to perform this feat of technical sketching in computer terms. The program POLYARCS was written to accept details of the road network as input, and synthesise the specification of the corresponding Circulation System. Its functions are described in Section 2.5.

Road Network



Circulation System

Fig. 1 Road network and corresponding Circulation System

2.2 TREES AND VINES

MVA Systematica describe the Circulation System as an expansion of the road network which allows the building of vines as opposed to trees; these terms need

explanation. The plane drawing of a road network consists of links and nodes. A crossroads is represented by a node. Links would only cross if there was a bridge or flyover. The activity of an assignment program in finding minimum cost routes between origins and destinations is described as path building. The paths from a single origin form a tree. In a tree there is only one path between any pair of nodes. However, the traffic engineer may want to allow more than one path between a pair of nodes. This is where vines come in.

Consider traffic approaching a crossroads where the right turn is banned. One often caters for right-turning traffic by indicating a sequence of left, right, right, turns starting at the junction before the crossroads. See Figure 2 below.

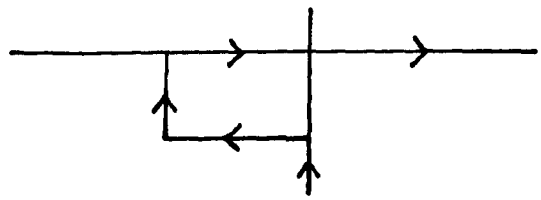


Fig. 2 Right-turning traffic

One would expect straight-ahead traffic to proceed straight ahead. This requires the links in the tree to cross at a node; something with which the minimum cost path building algorithm cannot cope. The way to get round this difficulty is to expand the junctions as in the Circulation System. Figure 1. Paths can then be built in which the two streams do cross but not at a vertex of this

expanded network. The paths from a single origin in this expanded network are said to form 'vines'. There will only be one path between each pair of vertices in the vine but the arcs between two different pairs of vertices may cross.

2.3 INDENTIFICATION OF CONFLICTING STREAMS

To show how conflicting streams are identified we consider the manoeuvres as represented in the digraph of the T-junction shown in Figure 3.

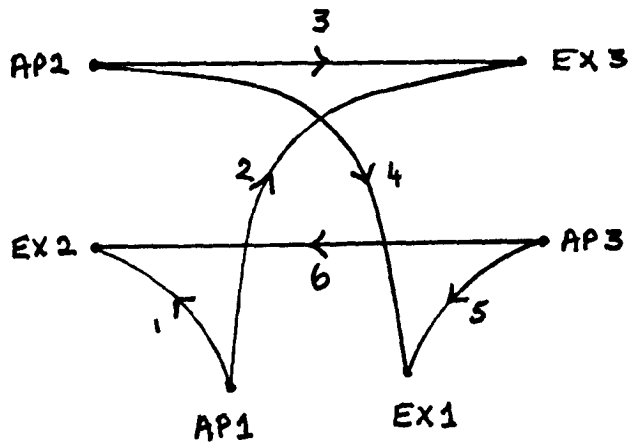


Fig. 3 A T-junction

First we will consider the left-turning manoeuvre from Approach 1 to Exit 2, represented by Arc 1. It merges with the manoeuvre from Approach 3 to Exit 2, represented by Arc 6, so we identify Arc 6 as conflicting with Arc 1. Similarly, we see that Arc 6 crosses Arcs 2 and 4 as well as merging with Arc 1, so the list of arcs conflicting with Arc 6 would consist of Arcs 1, 2 and 4.

The program POLYARCS incorporates digraph models of various different types of junction. It uses these models to create the part of the Circulation System corresponding to each junction. For each arc which it creates, it compiles the list of conflicting arcs from the model. The models are shown in Appendix 1.

2.4 CONFLICT QUANTIFIED AND USED AS A COST

Referring again to Figure 3, the potential conflict encountered by a driver using Arc 1 is quantified as the volume of traffic using Arc 6, the conflicting arc, because that driver is potentially in conflict with any of the users of Arc 6. The 'cost' to a driver of using Arc 1 is therefore equated to the volume of traffic using Arc 6. The way in which this definition of cost has to be revised is explained in Subsection 3.5.3. The cost, to the whole system, of having many drivers using Arc 1 will be the volume of traffic assigned to Arc 1 multiplied by the cost, to one driver, of using Arc 1. This definition, as it stands, is reflexive in that, for instance, the cost of assigning 3 drivers to Arc 1 and 4 drivers to Arc 6 (in the absence of any other traffic at that junction) is 12 for each of these two assignments.

In order to find a cost for each arc we need both the list of arcs conflicting with it, and the volume of traffic using each of those conflicting arcs. What is called an "incumbent assignment" of traffic to the

network is used to provide this figure for volume of traffic using each arc. The way a first incumbent assignment is found is described in Section 4.1. The assignment program POLYSEND then finds a succession of new assignments using each one as the incumbent from which to compute the costs for the next.

Some conflicting manoeuvres would seem to be more dangerous, or to cause more delay than others. In order to reflect these differences, each term in the sum referred to above can be given an appropriate weight. The program POLYARCS has provision for a list of weights to correspond to each list of conflicting arcs.

2.5 THE FUNCTION OF THE PROGRAM POLYARCS

The Circulation System is an elaboration of the road network. It is defined in terms of vertices and arcs, hence the name POLYARCS. When the program was first being tested, with very simple networks, deducing the specification of the Circulation System from the details of the road network was easy. However, as soon as tests were extended to a real network this deduction became tedious and prone to error. Compiling the lists of conflicting arcs was also tedious. It was therefore decided to write a program to -

1) accept details of the road network,

2) synthesise the specification of the corresponding Circulation System while drawing the user's attention to any inconsistencies in the input data,

3) compile lists of conflicting arcs with appropriate weights,

4) prepare data to set up the correspondence between flows on the arcs of the Circulation System and flows on the links of the road network.

Stage 2 proved to be a fairly complicated exercise in algorithm design. A similar network synthesis is performed within the TRIPS suite of programs developed by the industrial collaborators, MVA Systematica. However, in this application we need to identify pairs of arcs which conflict. By using a purpose built algorithm, Stages 3 and 4 could be anticipated from the outset.

2.6 SPECIFICATION OF THE ROAD NETWORK

The first function of the program POLYARCS is to accept details of the road network. Much of that detail is common to traffic modelling packages available commercially. Those details which pertain to the CROWN design tool in particular will be clearly indicated. The

road network is specified by giving each junction a distinct number, and then making a list of the links between junctions, giving each link a consecutive number as it is listed. For the CROWN design tool we need to specify the order of the links round each junction as well.

Traffic management is concerned with the traffic as well as the road network. Traffic is generated by the need to transport people or goods between certain origins and destinations. For the purposes of a traffic study, the area covered by the road network is divided into a manageable number of zones. Trips are deemed to originate or terminate in these zones. The points where traffic can enter the network from outside the study area, or leave the area, are also designated as zones; these are often referred to as external zones. Each zone has a notional zone centroid which is connected to the network by a notional zone connector, functioning like a link. An extra junction can be defined, if necessary, to be the point at which the zone connector meets the road network.

The first record of the input file which specifies the road network contains entries which enable the computer to interpret further records correctly. These entries are the numbers of zones, nodes and links. The number of zones indicates how many zone records are to be read. The number of link records and of junction records to be read are indicated similarly. The precise details

of the specification are given in the subsections which follow.

2.6.1 Zone records

In the CROWN design tool, the number of zone connectors for Zone M is indicated by a single integer in the Mth zone record.

2.6.2 Links specified by A nodes and B nodes

The topology of the road network is specified by a list of links. The CROWN design tool requires the zone connectors, defined in Section 2.6, to be listed first, and in zone order. The relevance of this is set out in Section 2.7.2. Each link is specified by the numbers of the nodes at its ends. One of these nodes is referred to as the A node and the other as the B node. If the road is two-way the link will be two-way and it will not matter which node is designated as the A node. If, however, the road is a one-way street, the link will be one-way, and the nodes should be chosen so that the permitted direction is from the A node to the B node. For links which are zone connectors, the A node should have the same number as the zone; one-way outbound zone connectors are allowed in the current version of the program, but not one-way inbound connectors.

2.6.3 Link records

An example of a link record is shown below.

5 12 1

If this is the Mth such record, it means that link number M joins node 5 to node 12. The 1 indicates that it is a two-way link. A zero in this position would indicate that the link was one-way. These three entries are sufficient for the CROWN design tool. Commercial modelling packages would include many more entries: the distances, times, type, capacity etc. of the link.

2.6.4 Junction records

We need to record the order of links round a junction so that we can distinguish left-turning, right-turning and straight ahead movements. Some commercial packages make this distinction too. What follows is specific to the CROWN design tool but mimics the widely used SATURN program input to some extent. Each link has a link number, corresponding to its order in the list of links, so a junction can be specified by the node number of the junction and a list of the link numbers of those links terminating at it. For countries where road users drive on the left, the list should be in clockwise order. Conversely, where road users drive on the right, the list should be in anti-clockwise order. Further comments on the application of the program suite in countries where road

users drive on the right will be found in Appendix 2. Provision has been made for priorities at junctions to be specified by attaching significance to the link which is first in the list of links for input to the CROWN design tool. For a priority T-junction, the convention is that the link which is the minor road is listed first in this list. At a crossroads, we make the simplifying assumption that the minor road will cross the major road, and list one of the minor links first.

The possible turning movements and, in particular, the way those turning movements conflict with each other will vary according to the type of junction. Each type is specified by a type number so that the program uses the appropriate digraph to model the junction. Provision has been made in the prototype program version of the CROWN design tool for the following types of junction: no specified priority, priority, mini-roundabout, conventional roundabout, signalised, and grade-separated. Different subroutines will be called depending on how many links meet at a junction, so this number is also recorded in the junction record. A junction record might be as shown below.

25 2 3 14 13 32

This means that Node 25 is a junction of Type 2, a priority junction, that it has three arms, Links 14, 13 and 32 in that clockwise order, and with Link 14 being the

minor road. Three-arm junctions are referred to as T-junctions throughout.

2.7 SYNTHESIS OF THE CIRCULATION SYSTEM

The second function of the program POLYARCS is to specify the arcs and vertices of the Circulation System. As explained in Section 2.1, one vertex represents both an exit from one junction and the approach to the next. This means that a vertex corresponds to a particular side of the road which is the link joining the two junctions. An arc corresponds to a manoeuvre through a junction, so its start and end vertices correspond to the links and the side of the road in which one starts and finishes this manoeuvre. The first task then, is to set up the correspondence between links and vertices. The way this this is done is shown in Subsection 2.7.1.

Some vertices will be used for origins and destinations of flow. The correspondence between these vertices and the zones for the trip matrix is described in Subsection 2.7.2. The next step is to create ordered pairs of vertices to correspond to the arcs in the appropriate digraph models of the junctions. If there is inconsistency in the input data, the creation of these ordered pairs will be halted. Detection of such inconsistencies is described in Appendix 3. The creation of arcs is described in Subsection 2.7.3. The creation of the lists of conflicting arcs and their corresponding

weights is described in Subsection 2.7.4. Finally, the translation from flow on arcs in the Circulation System, to flows on links in the road network is described in Subsection 2.7.5.

2.7.1 Vertices created to correspond to links

The reader has been introduced to the digraph model for one type of junction; the model for a T-junction was shown in Figure 3. As explained in Section 2.6, a vertex corresponds to a particular side of the road. A path through such a vertex will therefore be along a link in a particular direction. If it is in the A to B direction, that vertex is described as being 'Upstream of the B node' so we use element L of an array UB to record the number of the vertex upstream of the B node for link number L. Conversely, paths in the B to A direction pass 'Downstream' through vertices with numbers recorded in the array DB.

Vertices in the Circulation System are given numbers in the order in which they are created. When link number L is being processed, the element UB(L) will be set equal to the next vertex number, to indicate that it is the vertex upstream of the junction represented by the B node of link L. If the link numbered L is two-way, the element DB(L) will be set equal to the following vertex number, to show that it is downstream of that junction and that traffic is permitted to leave the junction in the B

to A direction. Otherwise this element will remain zero, and no such vertex will be created. Consequently a one-way link will correspond to a single vertex.

The vertex downstream of the B node would clearly be upstream of the junction represented by the A node and vice versa. The decision to consider vertices in relation to the B node is arbitrary, but has the result that it is the upstream array that contains no zeros.

2.7.2 Vertices corresponding to origins and destinations

In the road network, the origins and destinations of traffic coincide with zone centroids. The trip matrix will define the demand for trips between pairs of zones. A route assignment program needs to find routes for the trips between each pair of distinct zones. Routes are defined as a succession of arcs, so they start and finish at vertices. The origin corresponding to a zone centroid will therefore be the vertex corresponding to the side of the zone connector used for outbound traffic. The destination will be the vertex corresponding to the side of the zone connector used for inbound traffic. These vertices have to be identified correctly with the corresponding zone. This is accomplished as follows.

In the link records, the zone connectors are assumed to be listed first, and in zone order. The A node is assumed to have the same number as the zone. Links are

processed in order, starting with Link 1, and a pair of vertices are created, and numbered consecutively, for each link that is processed. The origin vertex for Zone 1 will therefore be vertex number 1, and the destination vertex will be vertex number 2. If there is more than one zone connector for any particular zone, the same vertex is designated as upstream of all the B nodes for those zone connectors and similarly for the vertex downstream of the corresponding B nodes. This designation enables a route to be found which uses the most suitable zone connector both for leaving the zone as an origin and for arriving at it as a destination.

Proceeding in this way, the origin vertex for Zone r will be vertex number 2r-1 and the destination vertex will be vertex number 2r. The demand for trips between Zone p and Zone q is then interpreted as a demand between vertex number 2p - 1 and vertex number 2q.

2.7.3 Creation of arcs

When all the vertices have been created, the program proceeds to create the arcs using a digraph model of the appropriate type for each junction. A manoeuvre corresponds to movement from one link to another, so the arc corresponding to it will start at a vertex in the pair corresponding to the one link and finish at a vertex in the pair corresponding to the other link. The vertices

have to be paired up in the correct order to create the appropriate arcs. This process can be compared to a jigsaw puzzle consisting of a junction piece and one piece for each arm. We know which arm piece is to be fitted to each hole in the junction piece but we have to decide which end of the arm piece to fit in that hole. The explanation of how this is done follows.

The vertices in the model are designated as approach vertices, e.g. AP(1), for the first link, and as exit vertices, e.g. EX(2), for the second link. They are matched to the vertices already created for the Circulation System. The program checks whether the B node or the A node of each link matches the node number of the junction. This matching process is illustrated with an example of a T-junction. Consider an example in which the junction record for the T-junction and the link records for the three links are as shown below.

Junction record	25	1	3	14	13	32
Link record for link 13	13	25	26	1		
Link record for link 14	14	24	25	1		
Link record for link 32	32	27	25	1		

Further suppose that the vertices in the Circulation System corresponding to these three links have numbers as shown below.

UB(13) = 101, DB(13) = 102,
UB(14) = 103, DB(14) = 104,
UB(32) = 139, DB(32) = 140.

The completed matching is shown in Figure 4.

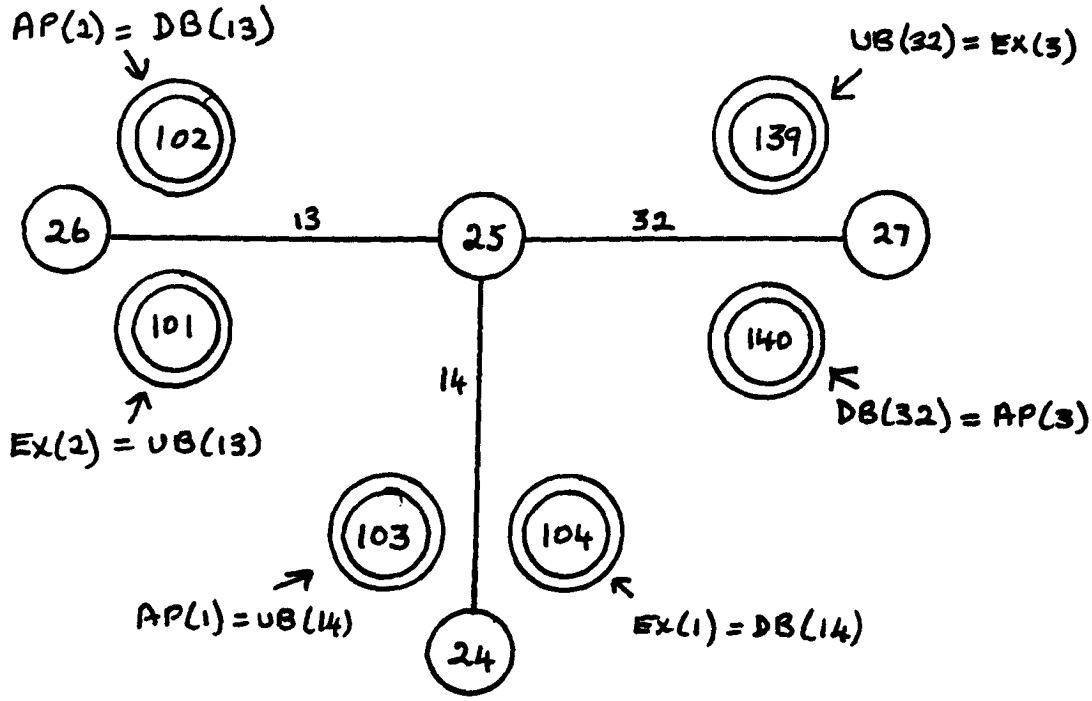


Fig. 4 Matching vertices

For the first link (Link 14), traffic approaches the junction upstream of the B node (25), so AP(1) is set equal to UB(14). Similarly traffic leaving by the first exit is downstream of the B node so, EX(1) is set equal to DB(14).

For the second link (Link 13), traffic approaches the junction downstream of the B node (26), so AP(2) is set equal to DB(13). Similarly traffic leaving by the second exit is upstream of the B node so, EX(2) is set equal to UB(13).

Once the elements of the AP and EX arrays have been matched to vertex numbers, the arcs can be created. The digraph models of each type of junction are stored in the computer memory as arrays and rules which

1) relate ordered pairs of elements from the AP and EX arrays respectively, which will represent vertices, to elements of an array KJ which represent arcs,

2) list for each arc the numbers of those arcs conflicting with it, and

3) list the first and last arc number of arcs starting at each approach vertex.

If non-zero vertex numbers have been matched to both elements in the pair corresponding to the first element in the array KJ, then an arc will be defined as being bounded by that pair of vertices. It will be allocated the next available arc number, and this number entered, as the first element, in the temporary array KJ. This procedure is repeated for all subsequent pairs of elements in the AP and EX arrays. In this way arcs are only created for permitted manoeuvres. In Figure 5, we show the arcs which would be created for the T-junction example above if we suppose that we are starting with arc number 201.

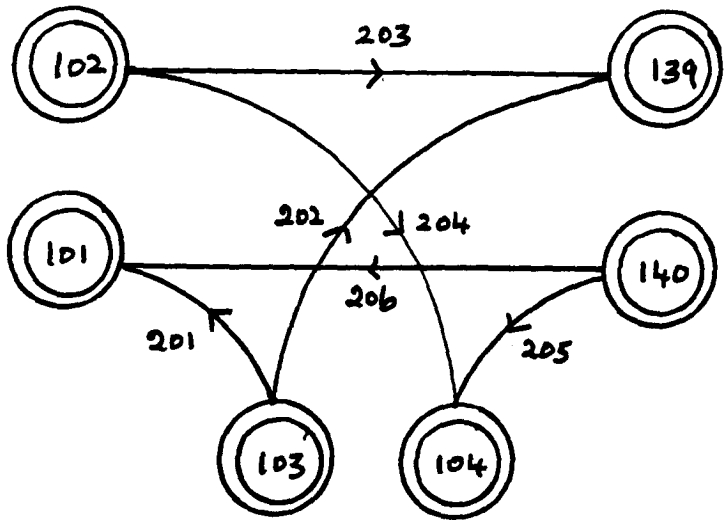


Fig. 5 Creation of arcs

From Figure 5, the reader can observe that arcs from a vertex upstream of the junction are numbered consecutively, starting with the left turning arc, and finishing with the right turning arc. All traffic proceeding along this link towards that junction has to pass through this vertex, so if we add up the flows on these arcs, the sum will give the total flow in this direction on this link. These arcs are numbered consecutively to make that process simpler.

2.7.4 Creation of list of conflicting arcs, with weights

The third function of the program POLYARCS is to compile a list, for each arc, of the arc numbers of those arcs which conflict with it, and matching weights for each of these conflicting arcs. This list is needed for the computation of costs in the assignment program POLYSEND. This program POLYARCS includes a one-to-many mapping of

the elements of the array KJ onto themselves which constitutes the ordered set of conflicting arcs for each arc. For example, at the T-junction, the set for Arc KJ(2) will consist of KJ(3), KJ(4) and KJ(6). Corresponding to the set of conflicting arcs for Arc KJ(2), there will be a set of appropriate weights. These are indicated by arguments in an array of weights. Thus for each set of conflicting arcs appropriate weights are also recorded, as described in Appendix 1. The list of conflicting arcs and appropriate weights is added to, after the creation of arcs is completed for each junction.

2.7.5 Flows on arcs converted to flows on links

The fourth function of the program POLYARCS is to prepare data to enable results, expressed in terms of flows in the Circulation System, to be re-expressed in terms of flows on links of the road network. When arcs are created, starting at the vertex UB(L), they will have consecutive numbers. The first and last of these numbers are stored in arrays as FIRSTAB(L) and LASTAB(L). These arcs carry flow from the A node to the B node. Addition of the flows on arcs numbered FIRSTAB(L) to LASTAB(L) will thus give total flow along Link L in the direction A to B. Similarly when arcs are created starting at DB(L) their first and last numbers are stored in arrays as FIRSTBA(L) and LASTBA(L). The arcs numbered from FIRSTBA(L) to LASTBA(L) are used to obtain total flow in the B to A direction of Link L.

By this means, flows in each direction on each link can be computed, with the exception of flows along zone connectors into destinations. The exception arises because each such flow is carried by arcs having a common end vertex rather than a common start vertex. An arbitrary choice was made to obtain flows on links by summing the flows on arcs carrying flow leaving the link and entering another one rather than those carrying flow entering the link; no flow leaves a zone connector into a destination to enter another link. The effect is that flows into destinations appear as zeros in the output. An extra subroutine to list the arcs terminating at each destination would be needed to remove this exception.

The contents of these arrays are recorded in a file ARCLINK.DAT. Once the main program POLYSEND has determined the assignments, the following program POLYLINK uses this file to convert flows on arcs of the Circulation System into flows on the links of the original road network.

2.8 CONCLUSION

The Circulation System has to be created so that the problem to find minimum cost routes can be formulated in terms of flows on arcs in it. The problem is formulated in the next chapter.

CHAPTER 3

THE MATHEMATICAL PROBLEM AND ITS SOLUTION

The aim of this project is to develop a mathematical programming technique as a design tool for traffic management. In Chapter 2, the way that the road network is modelled in order to make it possible to quantify conflict for each arc of the model, was explained. This particular model will be referred to as the Circulation System. The problem is defined in terms of the Circulation System. The inputs to the problem are described in Section 3.1. Traffic flow in a network with n zones can be modelled as an n-commodity problem. This is explained in Section 3.2. Some notation is introduced in Section 3.3 so that the problem can be formulated in Section 3.4. Several possible methods of solution are discussed in Section 3.5.

3.1 THE INPUTS TO THE PROBLEM

There are four categories of inputs to the problem. They are:

1) the road network with its zones, and rules governing permitted traffic movements,
2) the weights for different types of conflict,
3) the trip matrix,
4) the objective desired in the solution.

These are described in detail in the following

subsections.

3.1.1 The road network and the weights

Both the details of the road network and the weights for different types of conflict are prepared by the program POLYARCS. It produces a file, ARCS.DAT, which lists, for each arc of the Circulation System, the numbers of its start and end vertices. For each zone r, it will have designated vertex number 2r - 1 to be the origin vertex and vertex number 2r to be the destination vertex.

It also produces a file, CONFLICT.DAT, which lists for each arc, the numbers of those arcs conflicting with it, and the weight to be applied to each of those conflicting arcs when the cost of using the arc is computed. It is these two sets of lists of prepared input that will be used when the variables are defined in Section 3.3, and when the problem is formulated in Section 3.4. Each pair of conflicting arcs appears twice in the CONFLICT.DAT file, once in a list pertaining to one of the pair and again in a list pertaining to the other of the pair. This double entry format is not specifically required in order to define the objective function, but it is convenient for the solution method eventually chosen.

There are other details of the road network which appear in traffic models but which do not have to be included for the CROWN design tool. These are the length of each link, its capacity, and a speed-flow curve. The signal settings for signalised junctions are not included either. The intention is to minimise a measure of the amount of conflict between streams of traffic at junctions, so link times, which could be computed from link distance and a speed-flow curve, are not relevant to the main objective, although they may be relevant for comparing the various properties of different traffic assignments. An assignment made without capacity restraint can show where extra capacity would be advantageous. It may happen to use some links in one direction only in which case more capacity is actually available than would be specified for two-way operation. Although the facility for capacity restraint is available in the Out-of-Kilter algorithm it is not being used both because its use introduces considerably more complexity and because it may inhibit desirable possibilities at a design stage. Signal settings affect the delays at junctions and therefore the link times but we are not primarily concerned with link times.

3.1.2 The trip matrix

In reality, the demands for trips fluctuate both with the time of day, and from day to day. Traffic engineers usually model demand by assuming a steady state.

They may use different trip matrices to model steady states for the morning peak, the evening peak and off-peak travel demand. Recent advances in traffic modelling include what is called dynamic traffic assignment to distinguish it from the static, steady state assignment, which is adequate for the purpose of the CROWN design tool.

The elements of the trip matrix are used directly in formulating the problem. They are used to specify the amount of flow emanating from each origin to each destination and the amount of flow into each destination from each origin.

3.1.3 The objective

The objective is that the total weighted sum of the conflicts at junctions should be minimised. The input required for this is a list of pairs of conflicting arcs and a weight to be applied to each pair. This is the CONFLICT.DAT file produced by the program POLYARCS.

3.2 AN N-COMMODITY FLOW PROBLEM

Consider first a traffic flow problem in which traffic from various different origins all goes to the same destination. The network would consist of origin vertices, intermediate vertices and a destination vertex,

these vertices being connected by one-way arcs. If one variable is used for the flow on each arc, paths can be found for all this traffic, by specifying the amount of flow out of each origin, the total flow into the destination, and conservation of flow for all intermediate vertices. This implies that such a flow problem is in fact a single commodity flow problem.

A very similar argument can be used to show that a traffic flow problem in which all the traffic starts from the same origin and goes to different destinations is also a single commodity flow problem.

Either of these arguments can be used to explain that the general traffic flow problem, in which there are n zones functioning as both origins and destinations, and in which the demand for trips between pairs of zones is specified by a trip matrix, is an n-commodity flow problem. The commodities are distinguished from each other either by origin or by destination. An n-commodity problem will therefore need n variables for the flow on each arc.

In the definition of variables which follows, we define our commodities by their origin. The relative merits of this way of defining the variables, as opposed to the alternative way, are discussed in Subsection 4.3.2.

3.3 NOTATION USED TO DEFINE THE PROBLEM

A set of variables is defined for each arc of the Circulation System. Each set consists of the flows currently assigned from the different zones. A variable can therefore be identified by two subscripts; one is the zone number for the zone where the flow originates and the other is the arc number. However, for reasons that will become clear, when we come to specify the constraints, it is more convenient to identify an arc by the numbers of the ordered pair of terminal vertices. Thus the variable for flow from Zone p on arc (i,j) is denoted by FLOW(p,i,j).

A particular instance of the traffic flow problem is specified by the elements of the trip matrix and the weights. The demand for trips from Zone p to Zone q will be denoted by T(p,q). For a particular pair of conflicting arcs denoted by x, the weight to be applied to that pair will be denoted by W(x).

3.4 THE FORMULATION OF THE PROBLEM.

The problem is formulated in terms of the variables FLOW(p,i,j), and the constants T(p,q) and W(x). Before formulating the constraints it will be convenient to define further variables in terms of the variables FLOW(p,i,j). The total flow from Zone p, into Vertex j, is defined by:

$$FLOWIN(p,j) = \sum_{i} FLOW(p,i,j);$$

where the summation is taken over every i for which there is an arc from i to j,

The total flow from Zone p out of Vertex i is defined by:

$$FLOWOUT(p,i) = \sum_{j} FLOW(p,i,j)$$

where the summation is taken over every j for which there is an arc from i to j.

The total flow on arc (i,j) is defined by:

$$TOTFLOW(i,j) = \sum_{p} FLOW(p,i,j)$$

where the summation is taken over all zones p.

We now formulate the constraints of the problem. It will be convenient to be able to refer to the number of zones in the network, so we denote this number by n. The origin vertex for Zone p is vertex number 2p -1 (see Subsection 2.7.2). We have n constraints for flow out of each zone.

For each Zone p they take the form:

$$FLOWOUT(p,2p-1) = \sum_{q} T(p,q) \qquad [1]$$

where the summation is taken over all zones, q.

Each destination may receive flow from all the other zones. Traffic from one part of a zone to another part, for example in a car park or on a housing estate, is not modelled in this formulation or in the MICROTRIPS suite of programs. If such trips, which are called intra-zonal trips, are important, the zone should be split so that movements between its parts can be modelled. Therefore each destination may receive (n - 1) commodities and we have n(n - 1) constraints for flow into the destinations. The destination vertex for Zone q is vertex number 2q (see Subsection 2.7.2). We will often find it convenient to refer to the flow from origin, Zone p, to destination, Zone q, as being between O-D pair (p,q).

For each O-D pair (p,q), with $p \neq q$, these constraints take the form:

$$FLOWIN(p,2q) = T(p,q) \qquad [2]$$

If the Circulation System has m vertices, there will be n(m - 2n) constraints for flow through the m - 2n intermediate vertices.

For each Zone p, and for each intermediate Vertex t, they will take the form:

$$FLOWIN(p,t) = FLOWOUT(p,t) \qquad [3]$$

This gives a total of n(m - n) constraints. As a guide to the likely size of m, the number of vertices, for a road network with n zones and k links (excluding zone connectors) m equals 2k (or less if there are one-way links), so there would be 2kn - n^2 constraints. Typically the number of zones into which a network would be divided would be chosen so that n would be approximately k/4. This gives us about $7n^2$ constraints. For a problem with 50 zones we would have about 20,000 constraints. The use of a network algorithm enables such large numbers of constraints to be handled with relative ease. Capacity constraints have not been included at this stage of development of the CROWN design tool as explained in Subsection 3.1.1.

The objective function is defined by:

$$C = \sum_{x} W(x)*TOTFLOW(a,b)*TOTFLOW(c,d)$$

where arcs (a,b) and (c,d) are the conflicting pair x and the summation is over all pairs x of conflicting arcs.

C is a quadratic function of the variables FLOW(p,i,j). The problem can now be formulated as:

Minimise C subject to the constraints detailed at [1], [2] and [3].

3.5 POSSIBLE SOLUTION METHODS

Although one intuitively feels that the number of trips demanded between each O-D pair should be an integer, the demand is expressed per unit time, so this is not necessarily the case.

The problem is to determine how the traffic should be allocated between the possible routes in order to minimise the value of the objective function. A solution is said to be in equilibrium when no individual change of route will reduce the value of the objective function. In our case, there will be no O-D pair for which a change of route will reduce the conflict for that O-D pair. If there were, then such a change would also reduce the value of the objective function. This implies that our system equilibrium solutions consist of user-equilibria. We make use of this fact in one of our solution methods. In general, solutions to minimisation problems can be in equilibrium without being globally optimal. Such solutions are referred to as local optima. The solution we seek is a global optimum.

Another property of an equilibrium solution to our problem is that all vehicles between a particular O-D pair will use the same route. If there were two routes with an equal amount of conflict, the total amount of conflict would only be increased by assigning some vehicles to each route; they would have to merge at some point. This property is known as the group travel property. It is

made use of in some of the solution methods which follow. Traffic managers refer to assignments where this property holds as all-or-nothing assignments.

The problem has been formulated as a constrained minimisation problem with a quadratic objective function. A mathematical programming technique for solving Quadratic Programming (QP) problems is considered in Subsection 3.5.1. Two different approaches using Integer Linear Programming (ILP) methods are considered in Subsection 3.5.2. Finally a heuristic method for improving a solution is outlined in Subsection 3.5.3.

3.5.1 Quadratic Programming

The solution requires the minimisation of a quadratic objective function. The first question to consider is whether the algorithms available for solving what are called quadratic (as opposed to linear) programming problems would be appropriate. Each term in the objective function represents the amount of flow on one arc multiplied by the amount of flow on a conflicting, and therefore different, arc, so there are no squared terms. This means that the function is not convex and the matrix of the quadratic form is not positive definite. For such functions Wolfe's method and Beale's method may only find a local rather than a global optimum (Sheffi 1985). A small test problem is formulated for quadratic

programming and solved using Beale's method in Appendix 4. The value obtained for the objective function is higher than the values obtained by other methods that were tried. The actual mechanism which prevented the solution process progressing to a better optimum is identified using this example. There is, however, a standard method for converting the quadratic function into a linear function using zero-one integer variables. This method is explained in the next section.

3.5.2 Integer linear programming

Two different approaches using integer programming were tried. The first method involved the conversion of the quadratic function into a linear function by the introduction of many extra zero-one variables. The other involved making a list of all plausible routes for each O-D pair, and using zero-one decision variables corresponding to the use or non-use of these routes. The formulation of a trivially small example by these two methods is given in Appendix 4. A description of the two methods follows.

FIRST METHOD

For the conversion of a quadratic function to a linear function the original variables have to be zero-one variables too. Fortunately the original variables can be split up to satisfy this condition. Each variable represents the use or non-use of each arc by the flow from

each O-D pair. The volume of flow is taken into the coefficients of a new but still quadratic objective function. For each pair of variables x and y occurring in the quadratic objective function, a new zero-one variable, z, is defined by

$$x + y \leq 1 + z$$

so that the product xy will only contribute to the quadratic function when z = 1. When each such product xy is replaced by z, minimisation of the quadratic function can be taken care of by minimising the weighted sum of the z's which is a linear function. This is an integer linear programming problem (ILP). As the number of variables, n, increases, the number of computations necessary to solve such problems increases exponentially with n. There is no known algorithm which solves the problem with computations whose number is a polynomial function of n. The problem is said to be N-P complete. The phenomenon is known as the combinatorial explosion: the number of possible solutions, which have to be tested to see if the objective function is a minimum, rises explosively with the size of the problem. This method can be used to find the global optimum for a small test problem so that the result can be compared with the results obtained by other methods.

The small test problem solved by QP in Appendix 4 is also solved by this method in that appendix. The road

network only has 6 links and 3 zones. The Circulation System has 12 vertices and 18 arcs but 52 zero-one variables were needed in the formulation.

A slightly bigger problem was also formulated by this method. The network is taken from WRIGHT (1979) and it has a street plan with 5 zones, 8 junctions and 15 links, 4 of which were one-way. Its corresponding Circulation System had 25 vertices and 37 one-way arcs. Eighteen pairs of arcs were involved in crossings and nineteen pairs were involved in mergings. This network would have had 740 variables just for the different flows on the different arcs. With very careful consideration of each flow and each arc one can establish that certain flows would not use certain arcs in any sensible solution, so the number of variables can be reduced from 740 to 81. However, by the time a zero-one variable has been defined for every pair of these variables occurring in the quadratic function, the total number of variables has reached 575. These extra variables require 494 constraints to define them. In addition there are 55 flow conservation constraints making a total of 549 constraints.

SECOND METHOD

The second method uses decision variables for the possible routes assuming that the group travel property holds. Considerable pre-processing of the data is required. The pre-processing brings to light some

interesting aspects of conflict between routes on a network. For each O-D pair there will be a finite number of alternative routes, none of which pass through the same vertex twice. Let us suppose we have a list of these routes for each O-D pair. The conflicts that vehicles using these routes might encounter can be shown in a square matrix M; each row and each column represents a route so that element M(i,j) can be used to show the number of conflicts between route i and route j. For simplicity, unit weights will be assumed so that the matrix will be symmetric, with zeros in the leading diagonal. This implies that the number of conflicts between route i and route j is entered as both M(i,j) and M(j,i). When this matrix is used to cost out a particular assignment in terms of conflict, each conflict is accounted for twice because the number of conflicts between each pair of routes has been entered twice. It turns out to be quite convenient to retain this double count of the conflicts.

Different categories of conflict can be identified in a process which starts by partitioning the matrix. Each part is a rectangular sub-matrix for the conflicts between the routes for one O-D pair and the routes for another O-D pair. Consider that part with rows corresponding to all the routes between O-D pair s and with columns corresponding to routes between a different O-D pair t. This submatrix may have some interesting properties. These are discussed as two cases.

CASE 1 All the elements in this part are strictly positive, with each one greater than or equal to some number k. This implies that whatever route is chosen for O-D pair s and whatever route is chosen for O-D pair t the two chosen routes will have at least k conflicts. O-D pairs s and t are said to have a topologically essential cost of k. If T(s) trips are assigned to O-D pair s and T(t) trips are assigned to O-D pair t, then for each such part, k*T(s)*T(t) cost units will be topologically essential in any solution.

Furthermore, if the globally optimum solution consists entirely of such topologically essential conflicts, then the routeing pattern for that solution will be globally optimal for any trip matrix. This property, which conforms with common sense, was observed when the program was tested with a small network. This small test network evidently has the special property that there is a globally optimum solution consisting only of topologically essential conflicts.

This may be a fairly rare property. The test network happened to include a zone inside a ring which was not a two-way ring. When the one-way link was made into a two-way link, then situations

occurred where, for example, O-D pair r could either use a route clockwise round the ring to avoid conflict with O-D pair s, or use a route anticlockwise round the ring to avoid conflict with O-D pair t. The globally optimal solution had to include conflict with one of these pairs, but neither of these conflicts was topologically essential on its own. It should be fairly easy to spot an O-D pair for which this situation existed.

Even where the network does not have this special property, the topologically essential conflicts can be used to obtain a lower bound for the solution to a conflict-minimising problem. Each part of the conflict matrix can be examined for the occurrence of topologically essential conflicts and all such conflicts added to obtain a lower bound for the total number of conflicts.

CASE 2 becomes relevant once any parts to which Case 1 applied have been 'reduced' by carrying out Step 1 below. It is also relevant for any parts to which Case 1 does not apply.

STEP 1 Reduce the conflict matrix to remove topologically essential conflicts in the following way. For each part of the matrix with smallest positive element k, subtract k from every element and record that

O-D pairs s and t contribute k*T(s)*T(t) essential conflicts.

When the cost matrix has been reduced by performing Step 1, there may be parts where, for a particular row i, all the elements M(i,j) in that part are strictly positive with each one greater than or equal to h. Consider such a part and suppose that the rows represent routes between O-D pair s, with row i representing route I, and that the columns represent routes between O-D pair t. In this case, if the solution is constrained so that route I is used, then the topologically essential conflicts will be increased by h*T(s)*T(t), no matter what route is selected for O-D pair t. The sum of such increases in conflicts for all parts of the matrix in which row i features, provides a figure for the increase in cost that must occur if traffic is constrained to use route I. This will be called the route cost of route I. One can assign traffic purely on the basis of these route costs. The solution to such a relaxed problem is obtained as a first solution for this second method of solution by ILP. The route costs are obtained by performing Step 2.

STEP 2 Reduce each row in each part by subtracting h the smallest positive entry from each element. For example suppose there are three routes A, B, and C

between O-D pair s and four routes W, X, Y, and Z between O-D pair t. Suppose that the part corresponding to conflict between O-D pairs s and t, and its transpose are shown below. For simplicity, also suppose that only one trip is demanded between each of these O-D pairs.

$$\begin{array}{c} \\ A \\ B \\ C \end{array}\begin{array}{c} \begin{array}{cccc} W & X & Y & Z \end{array} \\ \begin{bmatrix} 1 & 2 & 2 & 1 \\ 1 & 0 & 3 & 1 \\ 2 & 2 & 2 & 3 \end{bmatrix} \end{array} \qquad \begin{array}{c} \\ W \\ X \\ Y \\ Z \end{array}\begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 1 & 1 & 2 \\ 2 & 0 & 2 \\ 2 & 3 & 2 \\ 1 & 1 & 3 \end{bmatrix} \end{array}$$

This part will attribute route costs of

1 to A	1 to W
2 to C	2 to Y
	1 to Z.

Sub-matrices reduced as in Step 2, are shown below.

$$\begin{array}{c} \\ A \\ B \\ C \end{array}\begin{array}{c} \begin{array}{cccc} W & X & Y & Z \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 3 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \qquad \begin{array}{c} \\ W \\ X \\ Y \\ Z \end{array}\begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 0 & 1 \\ 2 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix} \end{array}$$

The two matrices are no longer transposes of each other, but the two together have significance. Refer back to the matrix they come from, and consider, for example, the cost in extra conflict of using route C with route Z. The CZ element is 3; it occurs twice in the conflict matrix, so with double counting, this combination contributes 6 conflicts. The 6 is the sum of

the route cost of using C = 2
the route cost of using Z = 1
the cost of using C with Z = 3.

The cost of using C with Z can be obtained by adding the CZ element remaining in the one reduced part to the CZ element remaining in the other reduced part. Using the pairs of reduced parts in this way, a submatrix of what will be called pair costs, can be obtained. For conflict between the example O-D pairs s and t above it is:

$$\begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{cccc} W & X & Y & Z \end{array} \\ \begin{bmatrix} 0 & 3 & 1 & 0 \\ 1 & 0 & 4 & 1 \\ 1 & 2 & 0 & 3 \end{bmatrix} \end{array}$$

To summarise, possible conflict between O-D pairs s and t has been split into:

a) topologically essential conflict,
b) route conflict for each route between O-D pair s,
c) route conflict for each route between O-D pair t,
d) pair conflict for each particular pair of routes.

The second ILP method of solution involves solving a series of ILP problems.

For the first problem, the variables correspond to the plausible routes between each O-D pair. The ith route between O-D pair s is denoted by Si, and the variable corresponding to it by X(Si). The route cost of using route Si is denoted by R(Si). The objective function to be minimised is:

Σ R(Si)*X(Si) where summation is taken over all routes and all O-D pairs.

Each O-D pair is constrained to use only one route so, for each O-D pair s:

$$\sum_{i} X(Si) = 1.$$

For the second problem, more variables are introduced, one for each pair of routes used in the solution to the first problem. For the pair of routes Si and Tj, the variable X(SiTj) is introduced and the pair cost, as defined above, is denoted by P(SiTj). For each new variable, there is a constraint:

$$X(Si) + X(Tj) \leq 1 + X(SiTj).$$

The objective function is augmented with an extra term:

$$P(SiTj)*X(SiTj).$$

Pair costs are introduced in this way, only as the pairs are used in the solution to the preceding ILP problems. Each successive ILP problems is a tightening of the previous problem. In this series of problems, one will arise where the solution contains no pairs of routes for which the pair cost has not been included in the objective function. This may, of course, involve quite a long series of ILP problems. The final solution in the series will be optimal. This statement is justified in Appendix 4.

This method is used to solve the small example also solved by other methods in Appendix 4. Full details of the way the solution progressed are given in that appendix. In summary, the series consisted of seven problems, only 7 out of the 26 pair costs had to be introduced to the objective function before the optimal solution was found. This implies that the first in the series of ILP problems had 12 variables, and the last and biggest had 19 variables. In contrast, the same problem solved by the first method had 52 variables.

This method capitalises on the network nature of the problem by taking paths as the fundamental variables. It recognises that the topological conflicts fall into three classes: those essential for the given network and trip matrix, those essential for each path, the route costs, and those pertaining to the use of a particular pair of paths, the pair costs. By starting with a solution which minimises the set of route conflicts it concentrates subsequent effort in a sensible direction.

In any realistic problem a large number of paths would have to be considered. The introduction of pair costs in the way described might turn out to involve the solution of very many ILP's. However, the pair costs which are introduced are restricted to pairs involving routes which appeared in the previous solutions. The large number of paths, which would have to be considered

in a realistic problem, would make even these reduced ILP problems rather big.

3.5.3 A heuristic method involving improvement.

An alternative method, which also makes use of the network nature of the problem, involves splitting the n-commodity problem into n single commodity problems and using a network algorithm to solve each of these subproblems in turn. It requires a starting solution to be improved. Methods for finding a starting solution are described in Section 4.1. Arc costs have to be computed for each subproblem. Consider the subproblem for the commodity defined as originating in Zone r. The values of the variables FLOW(p,i,j) can be fixed at their values in the incumbent solutions to all the subproblems and used in the formula for arc cost. This subproblem is to find values for FLOW(r,i,j) which minimise the cost of the assignment. This is the formula for cost used in Beale's method. However, with this formula a better solution may have a higher cost; this higher cost is not the true cost as demonstrated with an example, in the next two paragraphs.

The example, the same as the one used in Appendix 4, consists of a ring road with Zones 1 and 2 outside it, and Zone 3 inside it. The trip matrix is shown in shown below.

$$
\begin{array}{c} \\ 1 \\ 3 \\ 5 \end{array}
\begin{array}{c}
\begin{array}{ccc} 2 & 4 & 6 \end{array} \\
\begin{bmatrix} - & 1 & 3 \\ 1 & - & 1 \\ 1 & 4 & - \end{bmatrix}
\end{array}
$$

The Circulation System for this network with its three T-junctions is shown in Figure 6. The numbers by the arrows on the arcs show the total flow in an assignment of these trips and the numbers in brackets show the costs. For simplicity, we use unit weights for all pairs of conflicts. The value of the objective function for this assignment is 20.

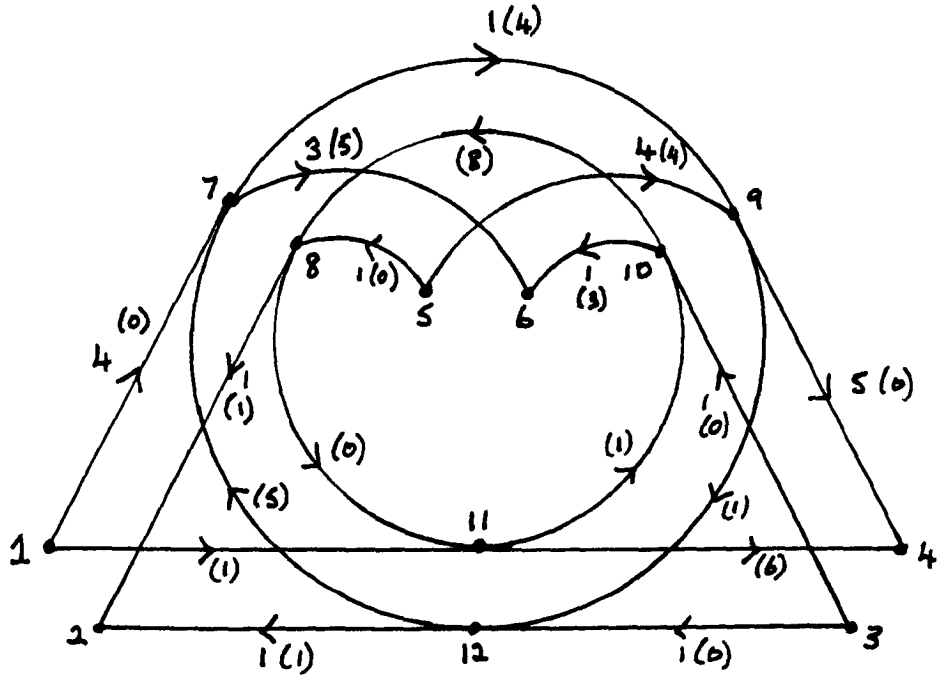


Fig. 6 Circulation System with assigned flows

Compare the costs of the two possible paths from Vertex 5 to Vertex 4. The clockwise path has a cost of 4, and the anticlockwise path has a cost of 6. Because the clockwise path has a lower cost, it would appear that the

total cost of the assignment would not be improved by assigning the flow, from Vertex 5 to Vertex 4, to the anticlockwise path. However, one element of the cost of the anticlockwise path is the cost of merging with 4 units flowing from 5 to 4 along arc (9,4). If this flow were transferred to the anticlockwise path it would no longer be on arc (9,4). This sort of cost will appropriately be called a 'ghost' cost. If it is omitted from the computations, the cost of the anticlockwise path from Vertex 5 to Vertex 4 will fall to 2, which is its true cost. When the assignment is changed by switching the flow of 4 units from Vertex 5 to Vertex 4 to the anticlockwise path, the reader can confirm, from Figure 7, that the number of conflicts has been reduced from 20 to 12.

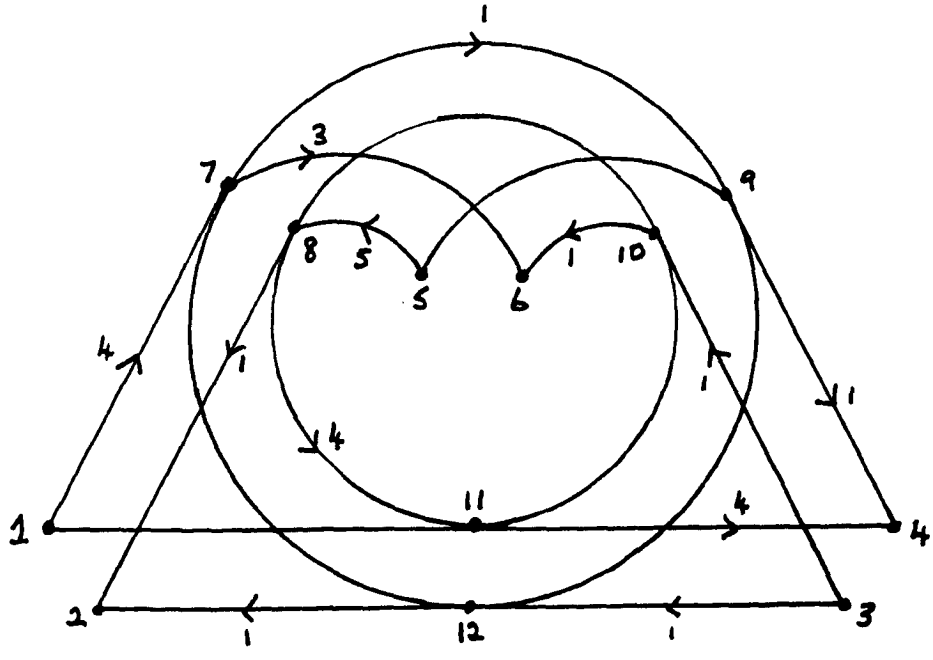


Fig. 7 A better assignment

However, omitting the value of FLOW(r,i,j) from the cost computation means that no account can be taken of any conflict between vehicles setting out from Zone r. Common sense would suggest that routes from the same origin fan out without crossing one another or merging with each other. The network algorithm has been modified so that it finds routes which form vines whose branches do not merge, although they may cross.

Crossing conflicts between traffic from the same origin can be accounted for if, instead of splitting the problem into n subproblems corresponding to the n commodities, it is split it into n(n - 1) subproblems corresponding to the demand for trips between each O-D pair. This involves (n -1) times as many variables, with the r in FLOW(r,i,j) referring to O-D pair r. Omitting the value of FLOW(r,i,j) from the cost computation in the subproblem involving O-D pair r, will remove the ghost costs and allow crossing conflicts with traffic from the same origin to be accounted for. Unfortunately this move towards accuracy multiplies the number of computations, that have to be done for each cycle of subproblems, by approximately (n - 1). As explained in Section 5.5, a subproblem for one O-D pair involves nearly as many computations as a subproblem for one commodity. This quest for extreme accuracy does not seem to justify the extra computations required, when the demand for trips is modelled rather crudely by a steady state. The heuristic method chosen for the CROWN design tool was therefore

based on subproblems corresponding to the n commodities.

The heuristic method will improve the solution, or leave it unchanged, with every subproblem that is solved. This improvement can be continued by solving each subproblem again and again. The solution of each subproblem once will be described as a complete cycle of iteration, to distinguish it from the solution of one subproblem, which will be referred to as a subcycle. Proceeding from one subcycle to the next, trips may be reassigned to different routes. Each reassignment will reduce the conflict at some junctions but it may increase it at others. Although the reductions will exceed the increases, the total cost of some of the routes used in the current assignments of trips from other origins may well show an increase over the total cost in the previous assignment. However, each reassignment made in this way will either reduce the value of the objective function or leave it unchanged. This is in contrast to what may happen when the time taken to traverse each arc, suitably modified to take account of congestion effects, is used as the arc cost; the value of the objective function may increase. This property of such time costs implies that, when they are used, the system optimum obtained may not be a user-optimum. With conflict costs, the system optimum will always be a user-optimum. The iteration process can, in principle, be carried on until no changes have taken place for a complete cycle of iterations. However, the equilibrium solution may be only a local optimum.

During the study, one condition which inhibited further improvement was identified. It involved the merging of trips from different origins to the same destination. Any improvement involves switching the route for one component. This may lead to the use of a route which diverges from the original route and then has to merge with it again to reach the same destination. This second merging involves an unnecessary conflict which would not occur if both the flows were simultaneously reassigned to a new common route. This phenomenon is referred to as 'mutually beneficial sightseeing' to indicate that the assignment is sub-optimal, but that change is inhibited by mutual benefit. The phenomenon is demonstrated with two examples in Section 4.1.1. A way to avoid its occurrence is suggested in Section 4.3.2.

It is regrettable that a global optimum cannot be guaranteed by the heuristic method. The solution may get trapped at a local optimum. Recent research has focused on ways of getting the solution out of the trap. Simulated annealing and what is called "Taboo search" are examples of approaches to this problem. It arises because the globally optimal solution may not be reachable by stepwise improvement from a particular incumbent solution. This problem is addressed in the CROWN design tool by offering the user different ways for finding a start-up solution with which to prime the iterative process, and by offering a choice in the order in which he solves and resolves the subproblems. The significance of the

differences in the values objective function obtained with these various options is illustrated in Sections 7.3 and 7.4.

3.6 CONCLUSION

In this chapter the problem has been formulated in terms of the Circulation System. Four methods of solving the problem of finding an assignment which minimises the total number of weighted conflicts at junctions have been reviewed. Although the goal of finding a global optimum appears not to be a practical proposition, a return to Beale's method is not recommended. This is because his technique involves taking account of the ghost costs defined in Section 3.5.3. Two possible formulations by Integer Linear Programming have been explored. Both these formulations increased rapidly in complexity as soon as the network ceased to be of only trivial size. The issues addressed in developing a heuristic method involving improvement have been summarised. As this is the method incorporated in the CROWN design tool, full details of this method appear in the next chapter.

CHAPTER 4

THE HEURISTIC METHOD OF SOLUTION

The method chosen to solve the problem is a heuristic method, involving improvement in the value of the objective function, by an iterative procedure. The purpose of this chapter is to describe that iterative process in more detail. In Section 4.1 various methods for finding a start-up solution with which to prime the iterative process are described. The iterative process may be restarted with a preferred interim solution. The provision for this is described in Section 4.2. The choice of the definition of subproblem to be solved in each subcycle is discussed in Section 4.3 Certain terms of the objective function are selected for improvement by the solution of each subproblem. These terms are identified, and the way in which the value of the quadratic objective function is improved by the minimisation of a series of linear objective functions is explained in Section 4.4.

4.1 PRIMING THE ITERATIVE PROCESS

The objective is to minimise conflict, which has been quantified as a cost dependent on existing traffic flows. A set of flows is therefore needed for each arc of the Circulation System, in order to compute the costs. These flow values should be plausible so that the first iteration uses plausible costs. The four methods that

were considered for finding a plausible solution with which to prime the iterative process are described in this section.

4.1.1 LOADFLOW

One sensible way to build up a starting solution is to load traffic from the first origin onto an empty network with zero costs on the arcs. Traffic from subsequent origins is then loaded so as to avoid as much conflict as possible with traffic already loaded. This method of obtaining a solution is called 'LOADFLOW'. It requires costs to be recomputed before the traffic from each successive origin is loaded. In the other methods, the same arc costs are used for finding minimum cost routes for all the traffic. However, although this method involves an extravagant use of computer time, it has the appeal that the ultimate objective of minimising conflict is being applied as the traffic is being loaded on to the network. The start-up solution obtained this way should have a lower amount of conflict than solutions obtained without taking any account of conflict. However one can see that it might be very sensitive to the order in which origins are selected for the flows from them to be assigned. This sensitivity is demonstrated with the network and trip matrices shown below.

A simple network in which there are only two routes from each origin to each destination, clockwise and

anticlockwise round a block, is used. Different orders of loading for two fairly sparse trip matrices were tried. In the first one, the heavier flows were loaded first and in the second one they were loaded last. The two trip matrices we use are:

Trip matrix 1

		to 5	6	7
from	1	0	0	2
	2	0	0	2
	3	4	0	0
	4	0	3	0

Trip matrix 2

		to 5	6	7
from	1	0	0	4
	2	0	0	4
	3	3	0	0
	4	0	2	0

The road network is shown in Figure 8.

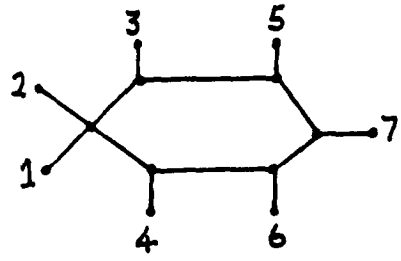


Fig. 8 The road network for LOADFLOW

Two routeing patterns were obtained for Trip Matrix 1, first by loading the heavier flows first and then by loading the lighter flows first: Cases A and B respectively.

Case A

Assignment in descending order of number of trips required; i.e. origins 3, 4, 1 then 2.

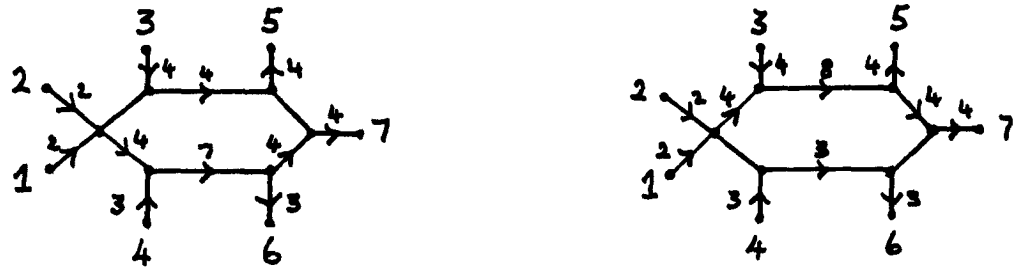
Case B

Assignment in ascending order of number of trips required; i.e. origins 1,2 4 then 3.

The routeing patterns obtained in these two cases are:

Case A	Case B
From 3 to 5: clockwise	From 1 to 7: clockwise
From 4 to 6: anticlockwise	From 2 to 7: clockwise
From 1 to 7: anticlockwise	From 4 to 6: anticlockwise
From 2 to 7: anticlockwise	From 3 to 5: clockwise

They are shown in Figure 9.



Total conflicts = 16 Total conflicts = 20

Fig. 9 Routeing patterns with Trip Matrix 1

In Case A, the assignment of the trips from origins 1 and 2 follows the assignment of the largest flow (4 units from 3 to 5), so they are assigned to avoid a merge with those 4 trips. In Case B, the trips from origins 1 and 2 were already assigned to the route which the flow from origin 3 could not avoid.

Case B demonstrates mutually beneficial sightseeing for the traffic from origins 1 and 2. Further assignments cannot improve the situation, Case A offers a better solution.

Two routeing patterns were obtained for Trip Matrix 2, first by loading the heavier flows first and

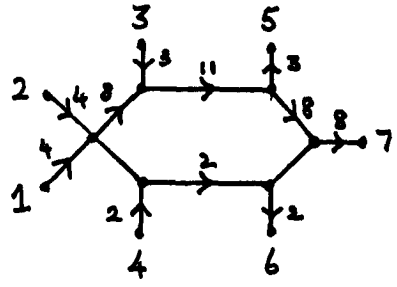
then by loading the lighter flows first: Cases C and D respectively.

Case C	Case D
Assignment in descending order of number of trips required; i.e. origins 1, 2, 3 then 4.	Assignment in ascending order of number of trips required; i.e. origins 4, 3, 1 then 2.

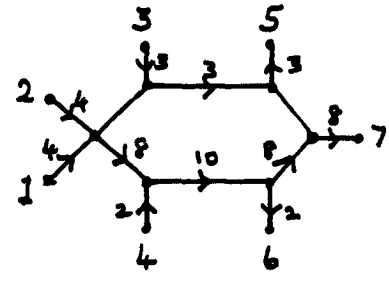
The routeing patterns obtained in these two cases are:

From 1 to 7: clockwise	From 4 to 6: anticlockwise
From 2 to 7: clockwise	From 3 to 5: clockwise
From 3 to 5: clockwise	From 1 to 7: anticlockwise
From 4 to 6: anticlockwise	From 2 to 7: anticlockwise

They are shown in Figure 10.



Total conflicts = 40



Total conflicts = 32

Fig. 10 Routeing patterns with Trip Matrix 2

In Case C, the flows from origins 1 and 2 are the largest flows and the flow from origin 3 cannot be assigned to avoid conflict with them. In Case D, the flows from origins 1 and 2 can once again be assigned to avoid conflict with either the trips from origin 3 or the trips from origin 4. It is the conflict with the greater number of trips, from origin 3, which is avoided.

Case C provides another demonstration of mutually beneficial sightseeing for traffic from origins 1 and 2.

In the face of this sensitivity to the order of loading and the large number of computations involved, simpler methods for finding a starting solution were designed.

4.1.2 DARTFLOW

The simplest method of all is to set all arc costs at unity and find minimum cost paths to assign the traffic to. This method is called 'DARTFLOW'. Routes are chosen to minimise the number of junctions used.

4.1.3 DASHFLOW

An elaboration of the DARTFLOW method goes some way towards taking account of conflict; the cost on each arc is set equal to the number of arcs it conflicts with. For example, turning left at a T-junction would have a cost of 1, whereas turning right would have a cost of 3; going straight ahead at a crossroads would have a cost of 6. This method is called 'DASHFLOW'. It is recommended for starting up, with DARTFLOW as an alternative which might lead to a better local optimum.

4.1.4 FASTFLOW

At an early stage in the development of the program, a method was used which was a crude imitation of a minimum journey time assignment. The real network, for which details were available, also included details of free flow times for the links; free flow time means time computed, from distance, using expected average speed in uncongested conditions. As an arc in the Circulation System corresponds to movement from one link to another, the average of the times for these two links was taken as the time for the arc and used as the cost.

The incentive to develop such a method was to provide some comparison between routes chosen to minimise journey time and those chosen to minimise conflict. The question of the trade-off between conflicts and journey time is of some interest. However, the only way to achieve a proper comparison is to use a conventional assignment package which takes the effects of congestion on journey time into account, so this poor imitation of conventional assignment was abandoned as a method for finding a start-up solution.

In principle comparisons can be made between the times for an assignment which minimises time and for one which minimises conflict, and between the number of conflicts occurring in these two assignments, without having to use a conventional assignment as the start-up solution. To make this comparison straight forward, input

and output files would need to be compatible with a conventional assignment package such as MICROTRIPS.

4.2 RESTARTING THE ITERATIVE PROCESS

As the user is offered various options for running the assignment program, he may want to try with various options for a few iterations each, and then choose the best solution to date to restart the iterative process. To the computer, re-starting is merely another method of priming the iterative process. The option of starting up with a previously obtained solution is called 'OLDFLOW'.

4.3 THE SERIAL SOLUTION OF SINGLE COMMODITY PROBLEMS

The term serial implies that the subproblems are solved one after the other. The user may choose the order in which the subproblems are solved; the various options are detailed in Subsection 6.2.2. The order chosen may affect the routeing pattern obtained and the value of the objective function at each stage of the iterative process. Such effects are assessed in Section 7.4. The user has another option which controls the maximum number of times each subproblem is solved before the program stops. This option is explained in Subsection 6.2.2 and the rate of convergence is demonstrated with various test problems in Section 7.2.

The two alternative ways of defining the commodities for an n-commodity problem are explained below. It is also possible to split the problem into n(n - 1) subproblems.

4.3.1 Commodity defined by origin

In Section 3.4, the problem was formulated in terms of commodities defined by their origin. This is the way the problem was formulated from the very start of the project, and this definition is used in the CROWN design tool. This choice was made for several reasons. The simple reason is that this was the way it was first thought of. The MICROTRIPS programs also sort routes by origin zone first and destination zone second.

The choice also lends itself to a natural visual image, that of the tree. Routes with a common origin vertex form a tree (the term is retained here although the word vine is more appropriate, see Section 2.2). In graph theory, a tree need not be directed, but a tree in a directed graph is described as being rooted at a particular vertex. The arcs in it will be directed away from the root towards what are called the tips of the branches.

More significantly, when one is concentrating on crossing and merging conflicts, and, as explained in Subsection 3.5.3, no account can be taken of conflicts

between vehicles being assigned in the same subproblem, it was attractive to define the subproblems so that such conflicts seemed unlikely to occur; routes from the same origin seemed likely to fan out in such a way that there would be no crossing or merging conflicts between them.

However, these conflicts can and do occur. Merging conflicts have been prevented by modifying the algorithm used to find minimum cost routes so that the group travel property will hold; routes from the same origin will be common until they diverge for their different destinations. Crossing conflicts can occur but no account will be taken of them. The problem being solved by the CROWN design tool is therefore a relaxation of the original problem. The relaxation involves neglecting crossing conflicts between vehicles starting out from the same origin.

The iterative process can be illustrated in terms of a fictional situation. Let us suppose that the number of trips required between each O-D pair is fixed. Let us further suppose that all the trips from the same origin are made in a fleet of vehicles, and that the fleet manager dictates the routes to be used. A set of routeing plans, one for each manager, is what we call a starting solution. Each day we hope to improve the solution. At dawn on the first day, the manager of the first fleet uses the plans from all the other managers to plot the numbers

of vehicles using each arc of the Circulation System. He wants his vehicles to encounter the minimum amount of conflict with all the other vehicles. He has a handy computer program to devise a routeing plan which achieves his aim. He dictates his new plan to his drivers. All the other drivers use the plan they have already. During the day, his drivers avoid conflict with the other drivers, so when the other drivers get together that evening they conclude that, between them, they experienced less conflict than the day before. That night the manager of the second fleet feeds the details of the routes all the drivers in the other fleets used that day into the computer program, and comes up with a new plan for his drivers by dawn. As they start out on the second day, his drivers are using a new plan but all the other drivers are using their plan of the day before. As the days progress, the manager of each fleet has a turn at finding a better plan; sometimes he does not succeed but at least he will not find a worse plan. If all the managers are given a second turn, and then further turns, at finding a new plan, there will come a time when no better plans have been found for a whole cycle of searches. The plans in operation at that time constitute the set of plans which we call the final solution.

4.3.2 Commodity defined by destination

If commodity is to be defined by destination, the variable FLOW(p,i,j) has to be redefined as the flow into Zone p on arc (i,j). This gives rise to a new set

of constraints:

$$\text{FLOWOUT}(p,2q-1) = T(q,p) \text{ for each } q \neq p,$$

$$\text{FLOWIN}((p,2p) = \sum_{q} T(q,p),$$

$$\text{FLOWIN}(p,t) = \text{FLOWOUT}(p,t) \text{ for each intermediate vertex } t.$$

These constraints will ensure the assignment of the required trips between all the other zones and Zone p. The single commodity is flow into Zone p.

The routes will form a bundle of rays converging at the destination vertex. Unfortunately, the term ray suggests a straight route, and the term converge suggests that the routes all merge at the same point; the overtones of science detract from the suitability of this image. The image of a basin of tributaries converging at the mouth of a river has the right overtones but it is not in general use; the set of routes to a common destination will be referred to as a bundle of rays.

There will certainly be conflict between flows into the same destination because the routes will merge with each other. However the amount of topologically necessary merging, for a given trip matrix, will be invariant; it will be

$$\sum_{r,s} \{T(r,p)*T(s,p)\}$$

where the summation is taken over all pairs r and s with r not equal to s. It will be desirable that the bundles of routes merge with each other only once. The algorithm can be adapted to build routes by working backwards from the destination in such a way that the group travel property holds, in that once routes have merged they do not diverge again.

This formulation may actually inhibit the occurrence of 'mutually beneficial sightseeing' mentioned in Subsection 3.5.3. Looking back at Figure 9 in Subsection 4.1.1, if no account is taken of the mergings with flows into the same destination, the clockwise route from either Zone 1 or Zone 2 to Zone 7 has a cost of 4, and the anticlockwise route has a cost of 3. The solution would therefore switch both these flows to the anticlockwise route. With the other definition of subproblems they get trapped on the clockwise route. This would be a good reason for considering changing the program to solve subproblems with commodities defined by destination.

4.3.3 Commodity defined by O-D pair

As explained in Subsection 3.5.3, crossing conflicts between flows from the same origin could be accounted for if the problem were defined by O-D pair. This involves splitting it into n(n - 1) subproblems. This would increase the amount of computation required.

It might be desirable to find a good solution by solving subproblems defined by either of the two methods already considered and then separating the flow variables to correspond to O-D pairs. An enlarged iterative procedure could then be used to solve subproblems defined by O-D pair.

4.4 THE QUADRATIC FUNCTION AS A SUM OF LINEAR FUNCTIONS

The linear functions which the Out-of-Kilter algorithm minimises are related to the original quadratic objective function. If the commodities are defined by origin, the format of the objective function which uses the variables FLOW(p,k) for flow from Zone p on arc k, where arc number k is the arc (i,j), is more compact for the purposes of this section.

To see how the terms are related, consider the terms of the quadratic function in detail. Consider the terms in the product of flows for just one pair of conflicting arcs m and n. Suppose the problem concerns flows from 4 origins: W,X,Y and Z. Denote flows from these origins on arc m by FLOW(W,m), FLOW(X,m), FLOW(Y,m), and FLOW(Z,m) and on arc n by FLOW(W,n), FLOW(X,n), FLOW(Y,n), and FLOW(Z,n) respectively. The product of flows on this pair of arcs can be shown in a table. All terms of the sort FLOW(W,m)*FLOW(W,n) are being neglected because crossing conflicts between flows from the same origin are being neglected. When these

terms are set out in a tabular form, there will be no diagonal terms. Those terms which will appear in the linear objective function when flows from origin W are assigned are indicated by a 'W' in Table 1. The linear terms are:

```
    [FLOW(X,m) + FLOW(Y,m) + FLOW(Z,m)]*FLOW(W,n)
  + [FLOW(X,n) + FLOW(Y,n) + FLOW(Z,n)]*FLOW(W,m);
```

The variables in these terms are FLOW(W,n) and FLOW(W,m). The parts in square brackets are the coefficients; they take on the values corresponding to the last assignment of flows from X, Y and Z.

TABLE 1

	FLOW(W,m)	FLOW(X,m)	FLOW(Y,m)	FLOW(Z,m)
FLOW(W,n)	0	W	W	W
FLOW(X,n)	W	0		
FLOW(Y,n)	W		0	
FLOW(Z,n)	W			0

Denoting the terms that will appear in the linear function when the trips from X, Y and Z are reassigned by X, Y and Z respectively, the complete table would appear as shown in Table 2.

TABLE 2

	FLOW(W,m)	FLOW(X,m)	FLOW(Y,m)	FLOW(Z,m)
FLOW(W,n)	0	W X	W Y	W Z
FLOW(X,n)	W X	0	X Y	X Z
FLOW(Y,n)	W Y	X Y	0	Y Z
FLOW(Z,n)	W Z	X Z	Y Z	0

It will be observed that each non-zero cell has exactly two letters, indicating that the terms of the quadratic function will appear in exactly two of the linear functions which are minimised as the flow from each origin is assigned. Looked at the other way round, once no changes have occurred for a complete cycle of reassigning trips from W, X, Y and Z, the sum of the values of the linear functions which have been minimised equals twice the value of the quadratic function.

Now the minimum of a sum of parts is not necessarily the sum of the minima of the parts, particularly if those parts are interdependent. However, the way these parts depend on each other means that there is no see-saw effect; when the value of one part is reduced the sum of the values of the others will be reduced by the same amount. In conflict terms, the flows from one origin have been reassigned to remove certain conflicts from the system and the other flows encounter correspondingly fewer conflicts. Hence the CROWN design tool produces flow patterns which progressively decrease

the value of the quadratic function.

4.5 CONCLUSION

The way the CROWN design tool solves the route allocation problem is not perfect, but it has been refined to overcome some of the difficulties encountered in the other possible methods which were considered. It makes use of a network algorithm, the Out-of-Kilter algorithm. This algorithm is described in some detail, in the next chapter, so that modifications made to it can be explained.

CHAPTER 5

THE ALGORITHM FOR FINDING MINIMUM CONFLICT ROUTES

The iterative process, described in the last chapter, requires the use of an algorithm to find minimum cost routes. There are several such algorithms available. The Out-of-Kilter algorithm was chosen in the first place because it allows for capacity limitations on the arcs, which are relevant to traffic assignment. Although this facility is being by-passed in the prototype tool, it is intended that capacity restraint should be re-instated as an option when the CROWN design tool is developed further. This algorithm may well not be the most efficient one, but any software house interested in developing the fruits of this research for commercial purposes would probably use their own favoured algorithm at the time. The quest for the most efficient algorithm was not specified as part of the research project, but the question of efficiency has not been entirely neglected.

There are those modifications which have been developed by others to improve the efficiency of the algorithm as applied to minimum cost assignment problems. These are documented in the paper by Barr et al. (1974). They are not incorporated in the CROWN design tool.

Then there are those modifications which improve its efficiency in solving this particular problem. These

are described in Sections 5.5 and 5.6. In a personal conversation, Dr. Glover (joint author of Barr et al. (1974)), suggested that the efficiency of the Out-of-Kilter algorithm, when fine tuned to this particular application, might well be comparable with the efficiency of other, more general purpose, algorithms.

So that these and other modifications can be described, this chapter starts with a brief general description of the algorithm in Section 5.1, before the mathematical meaning attached to the term 'out-of-kilter' is explained, in Section 5.2. The way in which the algorithm searches for a flow augmenting circuit is described in Section 5.3. A trip matrix does not come into the general description of the algorithm; the way the demand for routes can be incorporated is explained in Section 5.4. The group travel property does not always hold for solutions found by the Out-of-Kilter algorithm. The modification to ensure it does hold is described in Section 5.5. The way in which one of the search routines is speeded up is described in Section 5.6. The implications of re-instating capacity restraint in the design tool are explained in Section 5.7.

5.1 THE OUT-OF-KILTER ALGORITHM

This algorithm finds a minimum cost loading of a single commodity onto a network, where each arc has a lower and an upper bound on capacity; flow is conserved

through the vertices. It can be used to find a cost minimising assignment of a single commodity demanded between certain sources and destinations. The term, single commodity, implies that the origin of the commodity is immaterial. In traffic assignment, however, origin matters, so traffic consists of many commodities. As already pointed out, in Subsection 4.2.3, all the traffic either from a common origin or into a common destination can be treated as a single commodity. The algorithm is used for trips from one origin at a time, thus solving the multi-commodity problem by solving a series of single commodity problems.

5.2 THE MEANING OF 'OUT-OF-KILTER'

The algorithm functions on a network of one-way arcs, in which every arc is part of at least one circuit of connected one-way arcs. Each arc is specified by a start vertex I, and an end vertex J. Associated with each arc three further items of data must be supplied. These are an upper and a lower bound on capacity, and a cost. A variable flow of a single commodity is associated with each arc, and it is the sum, taken over all arcs, of the product flow*cost which the algorithm minimises. In the more usual network, with sources and sinks, artificial arcs, and possibly vertices, will have to be specified which connect up the sinks to the relevant sources in any network flow problem.

Associated with each vertex, is another variable called the dual value or shadow cost. In mathematical programming terms the dual value is associated with the constraint to conserve flow through the vertex. Both these sets of variables are initially zero. From the cost on an arc, and the dual values of its start and end vertices, a net cost can be computed for it. The net cost is defined by

net cost = dual value at start vertex
+ cost
- dual value at end vertex.

It may be helpful to think of the dual value at a start vertex as a buying price, the cost as a transportation cost and the dual value at an end vertex as a selling price.

Then

net cost = buying price at start vertex
+ transportation cost
- selling price at end vertex.

This makes economic sense of the 'Out-of-Kilter' idea; if the net cost on an arc is positive, then only enough flow to satisfy the lower bound (LB) of flow should be assigned to it. But if the net cost is negative, implying that it is profitable to use that arc, then the maximum amount of flow, the upper bound (UB), should be assigned to it. If the net cost is zero then it does not matter as long as the flow is within the bounds on capacity. These

conditions are shown in what is known as the Kilter Diagram in Figure 11. At all stages of the algorithm, the values of the two variables, flow and net cost, can be plotted as points on a graph for each arc. A line showing points fulfilling these conditions is called the Kilter line. If the point is NOT on the Kilter line the arc is described as 'Out-of-Kilter'.

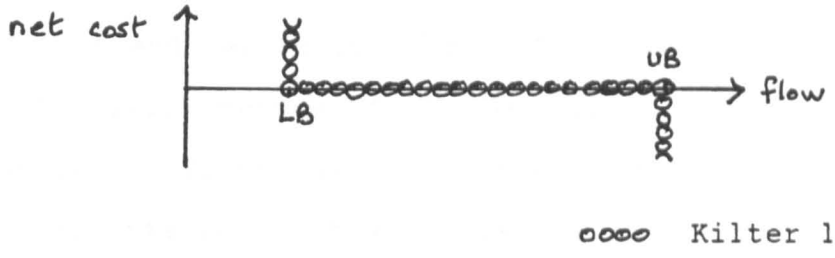


Fig. 11 The Kilter Diagram

The algorithm starts with zero flows on all arcs, so strictly positive lower bounds on some arcs are what drives it. The first step of the algorithm is a search of the list of arcs until one is found which is 'Out-of-Kilter'. This first arc will trigger off the next step, to find a flow-augmenting circuit. The value of flow for this arc can only be increased if two conditions are met. The first is that flow will be conserved through all vertices of the network. The second is that the proposed increase in flow in the circuit does not put any 'In Kilter' arcs 'Out-of-Kilter'. All the dual values are initialised at zero with the result that the first net costs used are merely transportation costs.

5.3 THE SEARCH FOR A FLOW AUGMENTING CIRCUIT

The algorithm starts to build a vine of arcs, rooted at the end vertex of the 'Out-of-Kilter' arc, and consisting of arcs which would not be put 'Out-of-Kilter' by an increase in flow. When an arc has been added to the vine its end vertex is labelled with the number of its start vertex. This facilitates the tracing back through the vine to identify the arcs of a flow-augmenting circuit when one has been found. As each new arc is added to the vine and its end vertex labelled, the algorithm tests whether the start vertex of the 'Out-of-Kilter' arc has been labelled. If it has, then there is a path through the vine from its root, the end vertex of the 'Out-of-Kilter' arc, to the start vertex of the 'Out-of-Kilter' arc. This path together with the 'Out-of-Kilter' arc forms a flow-augmenting circuit. The algorithm then augments the flow on the arcs of this circuit by as much as is required or at least as much as is permitted by the upper bounds.

If the algorithm fails to find a flow-augmenting circuit, the dual values of all vertices not in the vine are increased by the smallest amount which will enable at least one arc to be added to the vine. In order to progress with finding a flow augmenting circuit at this juncture, this smallest amount, the cost of using this extra arc, has to be accepted in the objective function. The increase in dual values has the effect of increasing the selling price but not the buying price on arcs with

start vertices, but not end vertices, in the vine. For at least one of these, the increased differential between selling price and buying price will reduce the net cost to zero, and then it can be added to the vine. The search continues, keeping the cost to reach the tips of the vine to a minimum, until a flow augmenting circuit is found. Thus the algorithm ensures that the total cost on the flow augmenting circuit has been kept to a minimum. The vine-building process is illustrated in Appendix 5.

5.4 USING THE ALGORITHM FOR TRAFFIC ASSIGNMENT.

As explained in Section 3.2, traffic assignment is an n-commodity problem. As explained in Section 3.5.3, this algorithm is being used to solve a series of single commodity problems, the commodity being defined by origin. In the Circulation System, origins of flow have one-way arcs out of them but not into them, and conversely for destinations, so circuits connecting the origin with each destination node do not exist in this network. To enable the algorithm to function, an artificial arc must be added to the network to connect each destination directly to the origin. The number of trips required between the origin and that destination is then set as the lower bound of flow on that artificial arc.

When that arc has been brought 'into Kilter', a flow satisfying the the demand for trips between that

origin and that destination will have been assigned to the network. Costs for the arcs are based on an incumbent assignment. By excluding the flows from the origin being considered, the 'ghost' costs, described in Subsection 4.2.1, can be removed.

5.5 ADAPTATION TO ENSURE GROUP TRAVEL PROPERTY

Because the trips from only one origin are being assigned, all the artificial arcs from the destination vertices will end at that origin vertex. These are the only 'Out-of-Kilter' arcs, so the vines built to find flow augmenting circuits will all be rooted at the same vertex. However, the algorithm builds a fresh vine rooted at the end of each 'Out-of-Kilter' arc, in order to find a flow augmenting circuit to bring it into Kilter, but it will retain dual values between finding one flow augmenting circuit and the next. This can result in the violation of the group travel property. In conflict terms this implies routes from the same origin diverging and then merging again before diverging to their separate destinations. This would involve a conflict which is not only unnecessary but also not taken into account in the arc costs. The small example of vine-building shown in Appendix 5 is extended to provide an illustration of how this can happen, in Appendix 6.

The algorithm is therefore modified to retain vines between finding one flow augmenting circuit and the

next. This is possible because the vines required would all be rooted at the same vertex, this vertex being the origin for trips being assigned. At each stage, all tips of the branches which are not destination vertices will represent the ends of equal cost routes from the root. When the next 'Out-of Kilter' arc is considered, its start vertex will either be in the existing vine or not. If it is in the vine, a minimum cost path can be traced back to the root; no further search will be needed to add more branches, and the time needed to build a new vine from scratch will be saved. If it is not in the vine, further branches will be needed, but the addition of further branches also takes less time than building from scratch. Retention of vines saves computing time as well as avoiding unnecessary merging of paths from the same origin.

5.6 A TIME-SAVING ADAPTATION

Details of the arcs in the circulation system are supplied to the algorithm as an ordered set. Those in the Circulation System will be the same for each run but the artificial arcs will be different. The loop to identify the 'Out-of-Kilter' arcs is adapted to test only those arcs which can ever be 'Out-of-Kilter'. These are the artifical arcs, and they are put at the end of the list of arcs so that the loop can be restricted to testing these. This saves computing time.

5.7 CAPACITY RESTRAINT

As explained in Subsection 3.1.1, capacity constraints are not applied in the prototype tool. Without them, arcs do not have to be tested for sufficient spare capacity. The effect of including them is, therefore, a considerable increase in the time needed for computation. One can no longer assume that minimum cost paths already identified have sufficient spare capacity. Fresh vines would have to be built for each Out-of-Kilter arc. With the building of fresh vines vehicles from the same origin might merge with each other. Although this might be necessary to keep flows within capacity, one would not want it to happen just because the cost of merges between vehicles from the same origin did not enter the cost calculations. In order to include these costs the problem would have to be split into a subproblem for each O-D pair. This would multiply the computation time approximately by the number of zones. The extra time would not only enable capacity restraint to operate, but also provide for the cost of crossings between vehicles from the same origin to be included in the cost calculations.

5.8 CONCLUSION

With this chapter the description of how solutions are obtained is complete. The next two chapters concern the user's interaction with the design tool; firstly his input and output, secondly his appraisal of the output.

CHAPTER 6

THE DESIGN TOOL

The aim of this project is to create the prototype of a design tool for traffic managers. M.V.A. Systematica, a software house specializing in transport modelling, accepted the invitation to collaborate on the project. Compatibility with their software was borne in mind during development of the design tool. For a start, the tool was programmed in the language they use, FORTRAN 77. It was not intended to produce a commercial package at this prototype stage, but to investigate possible solution algorithms, and to use the most effective one. The structure reflects the natural divisions of the computation required with the use of subroutines. To make the task of refining the code to professional standards straightforward, each subroutine is described in comment lines; further comment lines, charting the progress of the computations, are included in the longer subroutines. The structure is built up in layers so that intermediate results can be easily checked for correctness. Once the main program had been validated with small networks two subsidiary programs were written, one to accept relatively simple input and the other to produce relatively simple output.

This chapter starts with a brief description of the structure of the programs in Section 6.1. The input

required from the user is intended to be simple in format; it is described in Section 6.2. The design tool consists of a suite of three programs; the function of each one, and the way that they are designed to be run one after the other is explained in Section 6.3. The interpretation of the output is given in Section 6.4. These three sections are summarised with a flow chart in Section 6.5. The CROWN design tool could be adapted to assess the effects of proposed traffic management measures on conflict at junctions and on accidents. The necessary adaptations are detailed in Section 6.6.

6.1 THE STRUCTURE OF THE PROGRAMS

Chapter 5 was devoted to a description of the Out-of-Kilter algorithm which is at the heart of the solution method. It is used repeatedly with different data sets. The structure of the main program POLYSEND was therefore built up by stages, starting with a FORTRAN program, the subroutine KILTER. The different data sets are prepared for input to KILTER. The output from KILTER is processed both to provide interim solutions to the problem and to modify the input for the next call to KILTER. It is therefore the data structures for the input to and the output from KILTER which dictate the form of the surrounding structures. The design is actually an 'inside-out' design approach.

During the course of the program, KILTER is called on successive occasions for use with different inputs. Some of these inputs depend on the changes that KILTER itself has made to the values of some of the variables at previous calls. An iterative procedure ITERATE manages these inputs.

6.1.1 The subroutine ITERATE

As explained in Section 5.1, KILTER handles flows of a single commodity. As explained in Section 4.3, the prpblem is solved as a series of subproblems. The subroutine KILTARCS sets up the input for the artificial arcs, defined in Section 5.4, appropriate to the single commodity in the current subproblem. The subroutine GETREADY computes the costs to be used for the arcs in the current subproblem. It calls a subroutine ARCCOSTS to compute the cost of using each arc. The solution of each subproblem therefore involves calls to KILTARCS, GETREADY and KILTER.

When each subproblem has been solved once, the value of the objective function is reviewed. As explained in Section 4.4, this value is the sum of the values of the linear objective functions minimised in the solution of each subproblem. The subroutine ZONECOST evaluates the linear objective functions for each subproblem. It also calls ARCCOSTS. The subroutine SUMCRASH adds these values together.

6.1.2 Priming the iterative process

The iterative process has to be primed with a start-up solution. Some of the subroutines called by ITERATE are also called to obtain the start-up solution. The first option the user has is the order in which the subproblems are solved; a call to a subroutine LOADING reads this option from the screen. The various options will be described in Subsection 6.3.2. The second call is to a subroutine NETWORK which reads the file ARCS.DAT, produced by the program POLYARCS and described in Subsection 3.1.1. The option for the order in which the subproblems are solved is taken up by the subroutine ASSIGN which then calls LOADFLOW, DARTFLOW, DASHFLOW or OLDFLOW as appropriate. The subroutines called are tabulated in Table 3 below.

TABLE 3

SUBROUTINES CALLED BY ASSIGN

LOADFLOW	DARTFLOW	DASHFLOW	OLDFLOW	
*	*	*		KILTARCS
*				BUILDUP
*	*	*		KILTER
*	*	*		VINES(0)
*	*	*	*	SUMMARY
	*	*		TOTALFLO
*	*	*	*	ZONECOST
*	*	*	*	SUMCRASH

LOADFLOW is the only subroutine that uses the output from previous calls to KILTER; the subroutine BUILDUP processes this output. The description of VINES and SUMMARY is deferred until the next section, on output reports.

6.1.3 Output reports

Once iteration has stopped, either because tests have shown that there will be no further changes in the flow variables, or because the maximum number of iterations set by the user have been carried out, the main program calls a subroutine FINAL. FINAL calls VINES to record flows from each origin on each arc in the file FARCFLOW.DAT. VINES(0), called by those subroutines which find a start-up solution rather than restart with an old solution, records flows in the file STARCFLO.DAT. A list of unused arcs is recorded in the file SUMMARY.RPT by calling the subroutine SUMMARY. The value of the objective function at the end of each cycle of iterations is also recorded in this file so that the user can see how the process of reassignment converged.

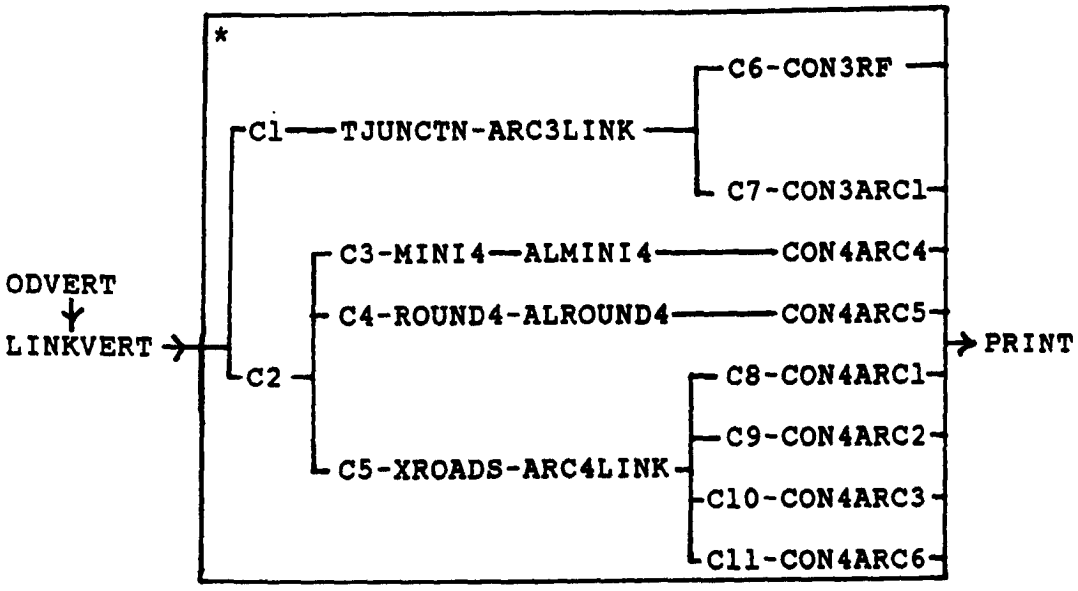
6.1.4 The program POLYARCS

The structure of this program follows the structure of the network synthesis algorithm as described in Section 2.6. It is a top-down tree structure. The

subroutine ODVERT creates vertices corresponding to those links which are zone connectors. The call to ODVERT is followed by a call to REMVERT which creates vertices for the remaining links. The program spends most of its time iterating inside the subroutine MAKEARCS which creates the arcs of the Circulation System. Finally it calls a subroutine PRINT to record output to be used for the program POLYSEND in the file ARCS.DAT, and for the program POLYLINK in the file ARCLINK.DAT.

MAKEARCS processes the junction records and calls the subroutine TJUNCTN if the junction has three arms, and one of the subroutines MINI4, ROUND4 or XROADS if the junction has four arms and conditional on junction type. The prototype design tool only processes three-arm and four-arm junctions.

The branching structure of the program is shown in Figure 12. The second level of branching prepares data to be recorded in the file ARCLINK.DAT for use by the program POLYLINK. The third level, its iterative nature indicated by the *, prepares data on the pairs of arcs which conflict, and on the weights to be attached to those conflicts. This data is recorded in the file CROSSFLO.DAT for use by the program POLYSEND.



Conditions for branching.

C1: 3 arms

C2: 4 arms

C3: mini-roundabout

C4: roundabout

C5: not C3 or C4

C6: mini-roundabout

C7: not C6

C8: free-for-all

C9: priority junction

C10: signalised junction

C11: grade-separated

Fig 12 The structure of POLYARCS

6.1.5 The program POLYLINK

This data processing program was designed with a top-down approach. It reads the files ARCLINK.DAT, STARCFLO.DAT and FARCFLOW.DAT. It adds up consecutive

items of data from the latter two files in DO-loops where the number of times the loop is executed is determined by the data in the first file. These sums represent flows on links of the road network. The results for total flows are recorded in the files SLINKFLO.DAT and FLINKFLO.DAT, for start-up flows and final flows respectively. The same results are sorted into bands according to volume of flow and recorded in the files SLINKSUM.DAT and FLINKSUM.DAT respectively. The results for flows segregated by origin are recorded in the files SLINKTRE.DAT and FLINKTRE.DAT, respectively. As the only iterations were within single DO-loops subroutines did not seem appropriate in this program.

6.2 INPUTTING THE DATA.

The data pertaining to the road network is entered in the form of a file called LINKS.DAT. The matrix of trips demanded on that network is entered in a file called TRIPS.DAT. A set of weights, to reflect the relative importance of the different types of conflict, is provided as a default in the program, but the intention is to give the user the option of using his own set of weights in the form of a file WEIGHTS.DAT, when the prototype design tool is developed further. All the record types used in these input files will now be specified.

6.2.1 The LINKS.DAT file.

This file has been described in Section 2.5; it is described again here in the context of using the design tool. The road network consists of nodes and links. The number of zones, which are origins or destinations of trips, is denoted by the parameter ZONES. Zones are connected to the network by connectors either to a node or to a node created in the middle of a link to terminate the connector. The connectors function like links, and the total number of connectors and links is denoted by the parameter LINKS. The number of nodes, including zones and any nodes required to terminate connectors, is denoted by the parameter NODES. The nodes should have distinct numbers but the order or the existence of gaps in the numbers is of no significance.

The first record in the LINKS.DAT file is

ZONES,NODES,LINKS.

This will control the DO-loops to read the other records. The user may wish to postpone entering NODES and LINKS until the file is complete, when he can deduce from the line numbers how many links and junctions there are.

The zone records state the number of connectors from each zone, these numbers being entered in zone order; each record consists of an integer, the number of zone connectors for that zone.

The link records have a record for each link starting with the zone connectors in zone order. This ordering ensures that in the Circulation System the origin vertex for the rth zone will have vertex number 2r-1 and the destination vertex for that zone the vertex number 2r. The connectors and links will acquire link numbers in the order in which they appear in these records. The user is advised to write in the numbers on a copy of the road network as a means not only of ticking them off, but also to assist him in identifying the numbers of the links which meet at each junction. The format of these records is -

A-NODE, B-NODE, TW(L).

TW(L) = 0 implies link L is one-way from A to B.
TW(L) = 1 implies link L is two-way.

In the junction records provision is made for seven different junction types. The diagrams for these appear in Appendix 1. Three of the types require comment here: free-for-all, grade separated, and user to be asked for weights. The "free-for-all" junction has no link given priority; this allows the user to experiment with the network before deciding on any priorities or other junction types. The grade separated junction has right-turning arcs which fly over or pass under the traffic on the main links so as to avoid crossing conflicts with it.

Traffic on the minor links in a four-arm grade separated junction also flies over or passes under. When the user designates a junction as "user to be asked for weights" the intention is that the tool will be developed further to prompt him for weights to be entered in accordance with the diagrams given in Appendix 1. The form of these records is free format -

XION(X), JT(X), NL(X), INJ(X,1),....INJ(X,NL(X))

where XION(X) is the node number of the Xth junction in the road network.

JT(X) is the junction type for that junction with:

JT(X) = 1 for a free-for-all junction
JT(X) = 2 for a priority junction
JT(X) = 3 for a signalised junction
JT(X) = 4 for a mini-roundabout
JT(X) = 5 for a roundabout
JT(X) = 6 for a grade separated junction
JT(X) = 7 FOR USER TO BE ASKED FOR WEIGHTS

NL(X) is the number of links at that junction.

INJ(X,1) is the link number of a minor road meeting at a priority junction or otherwise an arbitrary first link meeting at that junction.

Further links meeting at that junction are listed in clockwise order after the first, so INJ(X,NL(X)) is the link number of the last link.

6.2.2 The TRIPS.DAT file.

This is a standard character matrix file such as would be created by the MICROTRIPS transport modelling suite when 'dumping' a matrix in a file. The format is as

follows:

Columns	Contents
1 - 6	Zone number
7 - 12	Cell value for first column
13 - 18	Cell value for second column
19 - 24	Cell value for third column
.	.
.	.
73 - 78	Cell value for twelfth column.

If there are more than twelve zones in the matrix, then each matrix row will spill over onto additional records. The first record for each matrix row will contain values for columns 1 to 12 of the matrix; the second record will contain values for columns 13 to 24 etc. Each zone will start on a new output record.

6.2.3 The WEIGHTS.DAT file.

This file will be optional; it will allow the traffic manager to apply his own set of standard weights to the various conflicts in different types of junctions instead of the set of default weights provided in the program. Appendix 1 shows which arguments in the appropriate arrays apply to the different types of conflict.

6.3 RUNNING THE PROGRAMS

6.3.1 The program POLYARCS.

The first program in the suite, POLYARCS, synthesises the specification of the Circulation System from the details of the road network supplied by the user in a file LINKS.DAT. It may happen that the records in the LINKS.DAT file are inconsistent. If this is the case, the program cannot continue; it will stop with a a message on the screen:

"Mistake in LINKS.DAT file for junction ."
"Check your LINKS.DAT file and start again."

The user will have been directed to the scene of the inconsistency. The way such errors are detected has been explained in Appendix 3. If there are no inconsistencies, the program will complete the specification of the Circulation System and output the details to a file ARCS.DAT.

The program will proceed to prepare a list, for each arc, of those arcs conflicting with it, and a list of matching weights from the arrays WT3 and WT4 for each of those conflicting arcs. These lists are output to a file CONFLICT.DAT. These two files, ARCS.DAT and CONFLICT.DAT, will be used by the next program POLYSEND.

The program also identifies the first and last numbers of those arcs which between them will carry all the flow in one direction along each link of the road network. These details, which are needed for translating results in terms of flow on the arcs of the Circulation System into flows on the links of the road network, are output in a file ARCLINK.DAT. This file will be needed by the final program POLYLINK.

6.3.2 The program POLYSEND.

The second program, POLYSEND, is the route assignment program. It uses three files: ARCS.DAT, CONFLICT.DAT, created by POLYARCS, and TRIPS.DAT which has been prepared by the user. The user will be asked to input certain parameters as follows:

The parameter INIT controls which of the four methods of obtaining an initial assignment to prime the iterative process is to be used. These different methods have been described in Section 4.1.

INIT = 0 if LOADFLOW is to be the initial assignment
INIT = 1 if DARTFLOW is to be the initial assignment
INIT = 2 if DASHFLOW is to be the initial assignment
INIT = 3 if OLDFLOW is to be the initial assignment.

The parameter MAXITRN is the maximum number of iterations the user wants the program to run for. (It has been found that beyond about the third iteration the improvements, per complete iteration, in the objective function amount to less than 1% of its value for the initial assignment.)

The user will then be asked to if he wants to specify the order in which he wants the subproblems to be solved by giving answers as below:

1 for numeric order of zones,
2 for an order he will be asked to input
3 for a random order, for which he will be asked to input an integer seed to prime the random selection.

His selection will set up a correspondence between problem number and zone number. The order in which the subproblems are solved can have an effect on the outcome of the program.

POLYSEND prepares two files for the third program, POLYLINK, to give the user the opportunity to analyse results both for the initial assignment and for the final assignment. The flows assigned in the initial assignment are output in the file STARCFLO.DAT; those for the final assignment are output in the file FARCFLOW.DAT.

POLYSEND summarises the run, with details of the order of assignment and of the values of the objective function after each complete iteration. This summary is written to the file SUMMARY.RPT.

6.3.3 The program POLYLINK

When the user runs POLYSEND for the first time, with a particular combination of network and trip matrix, he will have no idea whether the options he has chosen are

giving rise to one of the more efficient solutions or to one of the less efficient solutions. Several runs with different orders of assignment and perhaps different values of INIT are recommended. He could then select the best solution and either use it as input to POLYLINK straight away or use it as OLDFLOW.DAT, with INIT = 4, to improve it for a few more iterations of POLYSEND.

POLYLINK converts the results obtained by POLYSEND, which are in terms of volumes of flow on the arcs in the Circulation System, to volumes of flow on the links of the road network. It produces two sets of files, one for the initial assignment and the other for the final assignment. Two files in each set give total flows on links, separately in each direction. One has the links in link order and the other has them sorted into bands according to the level of total flow. A third file shows the assignment of trips from each origin separately, listing volumes of flow from each origin in link order. The set of files with filenames beginning with S (for Start-up) give the results for the initial assignment. These may serve as a benchmark for comparison with the final assignment, which has been made with the aim of reducing potential conflict between streams of traffic. The set with filenames beginning with F (for Final) gives the results for the final assignment. The finer details of their contents are described in the next section.

6.4 INTERPRETING THE OUTPUT.

POLYARCS produces three output files for use by the other programs. Although they are formatted files they are not intended to be read by the user; a readable format was useful for checking the program during development. POLYSEND produces three output files only one of which is intended to be read by the user. POLYLINK produces six output files.

6.4.1 The file SUMMARY.RPT

The file SUMMARY.RPT summarises a run of the program POLYSEND. First it gives the order in which the subproblems were solved. Second it gives the value of the objective function after each complete cycle of iteration. Third it lists unused arcs. These correspond to turning movements which are not used in the efficient routeing pattern. The program could be enhanced to make it easier to identify these by expressing them as a series of three junction numbers, rather than as their start and end vertex numbers. An analysis has been made of the unused turning movements in Section 7.7. The banning of unused turning movements would be consistent with the efficient routeing pattern, and it would push traffic towards that pattern.

6.4.2 The files SLINKSUM.DAT and FLINKSUM.DAT.

These files, produced by POLYLINK, summarise the results in terms of total flows on the links, with the flows sorted into bands to show those links with bigger

volumes of flow distinctly from those with lighter flow.

The first table is of unused links. The flows into destinations will be shown as zero; the reason for this and a way of overcoming this anomaly were given in Subsection 2.5.7.

Those links used only in one direction are either one-way streets already or would make good candidates for one-way streets which are consistent with the efficient routeing pattern. To some extent they would push the traffic towards that pattern.

Subsequent tables have headings giving the bands and then start node, end node, and the volume of flow. A plot of these link flows onto a diagram of the network or displayed on the screen, if the programs were to be run in a graphics environment, would give the traffic manager a first impression of where the traffic was concentrated.

6.4.3 The files SLINKFLO.DAT and FLINKFLO.DAT.

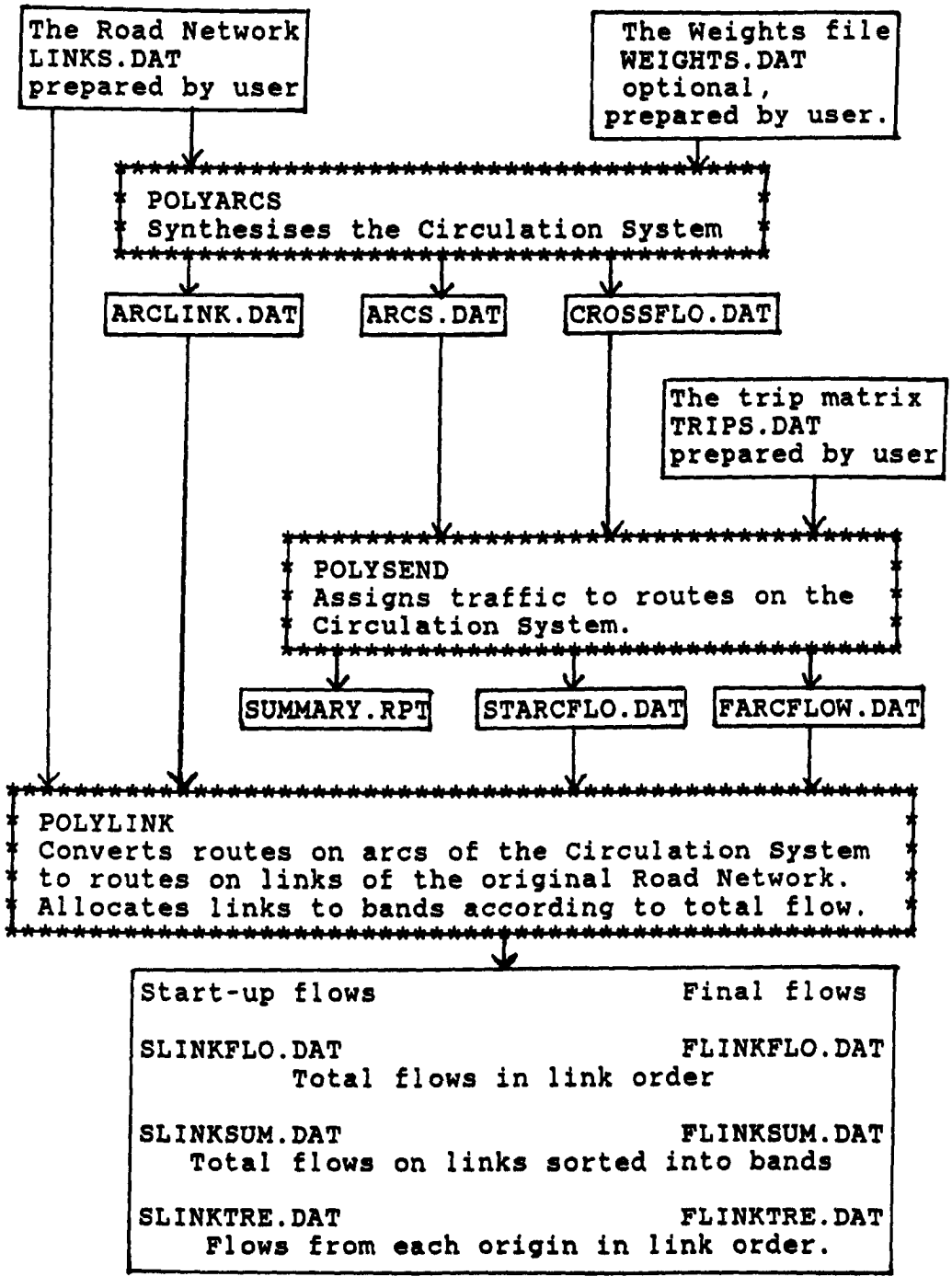
These files list in link order the total flows in each direction on each link. They contain exactly the same information as SLINKSUM.DAT and FLINKSUM.DAT but in link order rather than sorted into bands. These files would be more appropriate than SLINKSUM.DAT and FLINKSUM.DAT if one wanted to look at a particular link,

to see how near to capacity it was, for instance.

6.4.4 The files SLINKTRE.DAT and FLINKTRE.DAT.

These files, produced by POLYLINK, decompose the volumes of flow according to their origin. They consist of tables which are lists, for every link used and in each direction, of the flow from the origin named in the heading to the table. Again flows into destinations will be shown as zero by default. Plotted onto a diagram of the network these routes form a vine. Certain features of the routes may stand out, and comparison of the vines for the initial flows and for the final flows may give the traffic manager insight into desirable changes to encourage. The move to a graphics environment would be much more rewarding than any attempt to give these vines in terms of consecutive nodes along the paths from the origin to each destination as an aid to plotting.

6.5 FLOW DIAGRAM FOR THE SUITE OF PROGRAMS



6.6 ASSESSING TRAFFIC MANAGEMENT MEASURES.

Drivers react, over a period of time, to changes in traffic management measures. The routes they choose are forecast using conventional route assignment programs which imitate the criteria they use for route choice. From such programs, the redistribution of traffic vis-a-vis capacity and the effect on journey times can be forecast. The CROWN design tool could be adapted to assess the likely effect of these changes on the amount of conflict at junctions, and the potential for accident.

To assess the effect on the amount of conflict, total turning flows would have to be accessed from a conventional assignment package. A new subroutine would be required to read in these turning flows and convert them to flows on the arcs of the Circulation System. This subroutine could then call ZONECOST and SUMCRASH to compute the total amount of conflict which is a simple proxy for both the potential for accidents and for noise and air pollution.

The design tool also has the potential for adaptation so that the costs on arcs of the Circulation System are computed using accident predictive relations, such as those recently developed at the Transport and Road Research Laboratory. Adapted in this way, it could then be used for assessing the predicted number of accidents for a given flow pattern (Wackrill 1990).

6.7 CONCLUSION.

The CROWN design tool has relatively simple input. In its prototype version, it can be used to guide the traffic manager with respect to some of the management measures he uses. It has the potential for enhancement so that it could be a useful adjunct to existing traffic modelling packages. Care has been taken with programming so that distinct functions are performed by separate subroutines. The source code is given in Appendix 7. This general description of how it might be used will be followed, in the next chapter, with some results to demonstrate its performance.

CHAPTER 7

SOME RESULTS TO DEMONSTRATE PERFORMANCE

Experiments with the design tool to show both how well it performs, and how it can help with traffic management problems are described in this chapter. These experiments were carried out during the course of the project, some before the facility for weighting conflicts was installed and some with weighted conflicts. The simple default weighting of 1 for a merge, 2 for a crossing, and 3 for interlocking right turns, is used when weighted conflicts are referred to. The test networks used are described in Section 7.1. During the iterative process, the program progressively disentangles the routeing pattern to reduce the total number of conflicts; examples to show how quickly this process converges are given in Section 7.2. The user is offered various options for finding a start-up solution, to prime the iterative process. As shown in Section 4.1, the LOADFLOW method is very sensitive to the order in which trips are assigned from the various zones, so comparisons of performance, in Section 7.3, are restricted to the other two methods, which are not affected by the order of assignment. The DARTFLOW method and several test networks are used to investigate the effect of order of assignment on the reduction in conflict that is achieved at each iteration; the results are given in Section 7.4. This is followed with a study of the effect on traffic distribution, of some of the routeing patterns obtained,

in Section 7.5. Then, in Section 7.6 some spatial features of the routeing patterns obtained are analysed. These are the patterns the CROWN design tool identifies as efficient in reducing conflict. Traffic engineers may not be in a position to impose such patterns on the traffic, but they can take measures which would reinforce such patterns; the designation of one-way streets and the banning of particular turning movements are such measures. In Section 7.7, the extent to which the patterns indicate the places in the network where such measures would be appropriate is demonstrated.

7.1 TEST NETWORKS.

Some cities are actually built on a square grid pattern, so such a pattern is one obvious choice for idealised networks. These networks also have four-fold symmetry, so for every zone, except the centre zone, there are three others like it. When one attempts to quantify some characteristics of the routeing patterns there will be four which in theory should correspond to each other.

The three idealised road networks used are described first. They are extensions of one another. The first is a 4 by 4 square grid with 16 zones inside the 16 blocks, each being connected to the midpoints of the sides of its block. The number of trips required for each O-D pair was set as inversely proportional to the

road distance between them. This network and the corresponding trip matrix were extended to a 5 by 5 square grid. In reality, traffic enters and leaves a city across what is effectively an orbital ring road, so this network was extended further by the addition of external zones on continuations of the grid lines beyond what had been the boundary road. To increase the realism further, the junctions on the outer ring were modelled to allow right turning traffic to avoid crossing conflicts with the traffic on this ring road. This extended network is called "M25" in acknowledgement of its faint similarity to the orbital road around London. Two trip matrices were devised for it: one to represent the morning peak and the other the evening peak. The demand for trips between internal and external zones was assumed to be independent of distance between them and similarly for a small demand between external zones. The relative sizes of the non-diagonal elements in these trip matrices are indicated in Table 4.

TABLE 4

TRIP MATRICES FOR THE M25 NETWORK

	Morning peak		Evening peak	
To	External zones	Internal zones	External zones	Internal zones
From External zones	10	40	10	10
Internal zones	10	Average = 40	40	Average = 40

The real network and trip matrix used are for Hazel Grove near Manchester. The road network has 68 junctions and 127 links and the trip matrix is for 33 zones. Only 17 of these junctions are cross-roads and the rest are T-junctions, a very different structure to the grid networks with their predominance of crossroads. The relative sizes of the elements in the trip matrix are indicated in Table 5. Most of the traffic is between external zones and very little indeed between internal zones for the particular trip matrix available for this network. This, again, is rather different from the matrices created for the M25 network.

TABLE 5

TRIP MATRIX FOR HAZEL GROVE

From \ To	External zones	Internal zones
External zones	Average = 36 Range 0 - 360	Average = 10 Range 0 - 120
Internal zones	Average = 12 Range 0 - 230	Average = 1 Range 0 - 10

7.2 CHANGES IN THE VALUE OF THE OBJECTIVE FUNCTION.

To demonstrate how fast the value of the objective function changes, the DARTFLOW method was used to obtain a start-up solution and experiments made with three networks. Changes in the value of the objective function for unweighted conflicts, occurring in each complete cycle of iterations, are shown as a percentage of its value at the start-up assignment. These are given in Table 6.

TABLE 6

PERCENTAGE CHANGES IN THE VALUE OF THE OBJECTIVE FUNCTION

After	4 by 4 grid	5 by 5 grid	Hazel Grove
1 cycle	43	49	17
2 cycles	5	4	3
3 cycles	0.6	0.5	2.5
4 cycles	0.2	0.3	1.5
5 cycles	0.05	0.08	0.5

This table would seem to indicate that it is only appropriate to do about five complete cycles of re-assignments.

7.3 THE EFFECTS OF THE STARTING ASSIGNMENT

The effects of the two start-up methods which are not affected by the order of assigning trips are compared. The methods are DARTFLOW, which minimises the number of junctions used on a route, and DASHFLOW which minimises the number of conflicting streams of traffic encountered on a route regardless of the amount of traffic in those streams. The number of conflicting arcs is an attribute of each arc which is recorded for another purpose and therefore available for this purpose. Results with these two methods, with weighted conflicts, on the Hazel Grove network, are shown in Table 7.

TABLE 7

THE EFFECTS OF START-UP SOLUTION WITH HAZEL GROVE

	DARTFLOW	DASHFLOW
Start-up	60.5	59.6
After 1 cycle	49.2	49.1
2 cycles	46.8	47.3
3 cycles	45.9	45.9

Weighted conflicts in millions correct to 3 sig. fig.

DASHFLOW is better to start with, but gives rise to a worse value than DARTFLOW by the end of the second cycle and is then equally good by the end of the third cycle. It is also noticeable that at all stages the differences only showed in the third significant figure.

7.4 THE EFFECTS OF ORDER OF ASSIGNMENT

Two experiments, in which the order of re-assignment is varied, are carried out using DARTFLOW as the start-up method. In the first one with the smaller, 4 by 4 grid network, the facility for generating random orders of re-assignment is used to obtain five such random orders. The program is run until there would be no further changes in the objective function for unweighted conflicts. Different final values are obtained for the objective function with the different orders of assignment; one might be a global optimum but the others must be local optima. These random orders are given numbers in descending order of eventual merit; the results are shown in Table 8. There is not much difference between the values of the objective function at the

various local optima.

TABLE 8

RANDOM ORDERS OF RE-ASSIGNMENT WITH THE 4 BY 4 GRID

Order no.	1	2	3	4	5
Start-up	3192288	3192288	3192288	3192288	3192288
After					
1 cycle	1783282*	1794012	1806858	1804978	1834255
2 cycles	1619471	1619109*	1629345	1667849	1624815
3 cycles	1603598	1602653	1602355*	1621859	1603765
4 cycles	1593551*	1599153	1599005	1599819	1601927
5 cycles	1589246*	1596234	1598135	1597521	
6 cycles	1587654*	1591325	1597115		
7 cycles	1587474*	1589450	1596115		
8 cycles	1587492*	1589250			

* best result at this stage.

For the second experiment, the bigger 5 by 5 network was used and trips were assigned in an order related to the spatial layout of the zones at which they originated. These orders are described in words here, and their meaning shown in Figure 13. The simplest order starts in the top left-hand corner and works across the rows of the grid as one reads a page of English script; this is called order No. 1. Order No. 2 starts in the top left-hand corner and works down the columns of the grid. Order No. 3 starts off like order No. 1 but then spirals in clockwise to the central zone. Order No. 4 starts off like order No. 2 but then spirals in anti-clockwise to the central zone. Order No 5 starts at the central zone and spirals out clockwise to finish in the top right-hand corner. Order No 6 starts at the central zone and spirals out anticlockwise to finish in the top left-hand corner.

Order No. 1
down rows from left.

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Order No. 2
along columns from top.

1	6	11	16	21
2	7	12	17	22
3	8	13	18	23
4	9	14	19	24
5	10	15	20	25

Order No. 3
clockwise spiral in.

1	2	3	4	5
16	17	18	19	6
15	24	25	20	7
14	23	22	21	8
13	12	11	10	9

Order No. 4
anti-clockwise spiral in.

1	16	15	14	13
2	17	24	23	12
3	18	25	22	11
4	19	20	21	10
5	6	7	8	9

Order No. 5
clockwise spiral out.

21	22	23	24	25
20	7	8	9	10
19	6	1	2	11
18	5	4	3	12
17	16	15	14	13

Order No. 6
anti-clockwise spiral out.

25	24	23	22	21
10	9	8	7	20
11	2	1	6	19
12	3	4	5	18
13	14	15	16	17

Fig. 13 Geometric orders of assignment for the 5 by 5 grid

The program was run for three complete cycles of iteration; the results, with numbers of unweighted conflicts, after each cycle of iterations, are given in Table 9.

TABLE 9

GEOMETRIC ORDER OF ASSIGNMENT WITH THE 5 BY 5 GRID

Order	Start-up	after - 1 cycle	2 cycles	3 cycles
1	39968016	23173468	20550060	20187722
2	39968016	22730166	20455650	20170602
3	39968016	22877552	20506156	20207414
4	39968016	21928642*	20268430*	20077082*
5	39968016	25601850	20661382	20225182
6	39968016	24959956	20605034	20152354

* best result at this stage.

Anti-clockwise spiralling in gives the best result, and anti-clockwise spiralling out the second best result. Order No. 2, which starts off anti-clockwise is third best. One wonders whether, if the Circulation System had been created for driving on the right, the corresponding clockwise order of assignments would have been better.

7.5 THE WAY THAT TRAFFIC IS DISPERSED

The total flows on links show where the traffic is concentrated in the routeing patterns. The patterns found for three networks are analysed.

Assignment in the 'anti-clockwise spiralling in' order gave the best result with the 5 by 5 grid, so the program was run with this order for 10 cycles of iteration, and with weighted conflicts, in an attempt to achieve a really good solution. In the course of the tenth cycle the objective function only improved by 0.00015 % of its value at the start of the iteration. When the total flows were analysed, it was noticed not only that traffic is concentrated on the outer ring, but also that on all links in the outer ring the clockwise flow is about twice as heavy as the anti-clockwise flow.

For the M25 network, the traffic is again concentrated on the outer ring but it is fairly evenly distributed between clockwise and anticlockwise flow.

For the Hazel Grove network, most of the traffic is concentrated on a central spine but much is also concentrated on the outer ring.

7.6 SPATIAL PROPERTIES OF THE ROUTEING PATTERNS

The tendency of the routeing patterns to favour clockwise or anticlockwise orbital routes is illustrated with the M25 network and weighted conflicts. Just four of the 90 diagrams of trees of paths, which were drawn from each of the 45 zones and for morning and evening trip matrices, are selected for this purpose. They are shown in Figures 14 to 17. The convention, common in electrical diagrams, that paths which do not interfere with each other are shown with a 'bridge' or a curve, is used. The reader may recall that the junctions on the outer ring road were modelled with flyovers, so that there are no crossing conflicts for traffic on this outer ring. The tree and not the underlying road network is shown for the sake of clarity. The origin zone is shown by a circle and the destination zones as arrowheads.

This tendency, to use clockwise or anticlockwise orbital routes is analysed for all 90 of the diagrams referred to above. The way paths were selected to contribute to the totals in this analysis is illustrated with reference to Figures 14 to 17. In the analysis it was noted that the path used along the outer ring is not

always the shorter of the two possible paths. For zones diametrically opposite each other, it is also interesting to observe whether the orbital path used is clockwise or anticlockwise. Further details of this analysis, which is shown in Table 10, are given following the figures showing the routeing patterns.

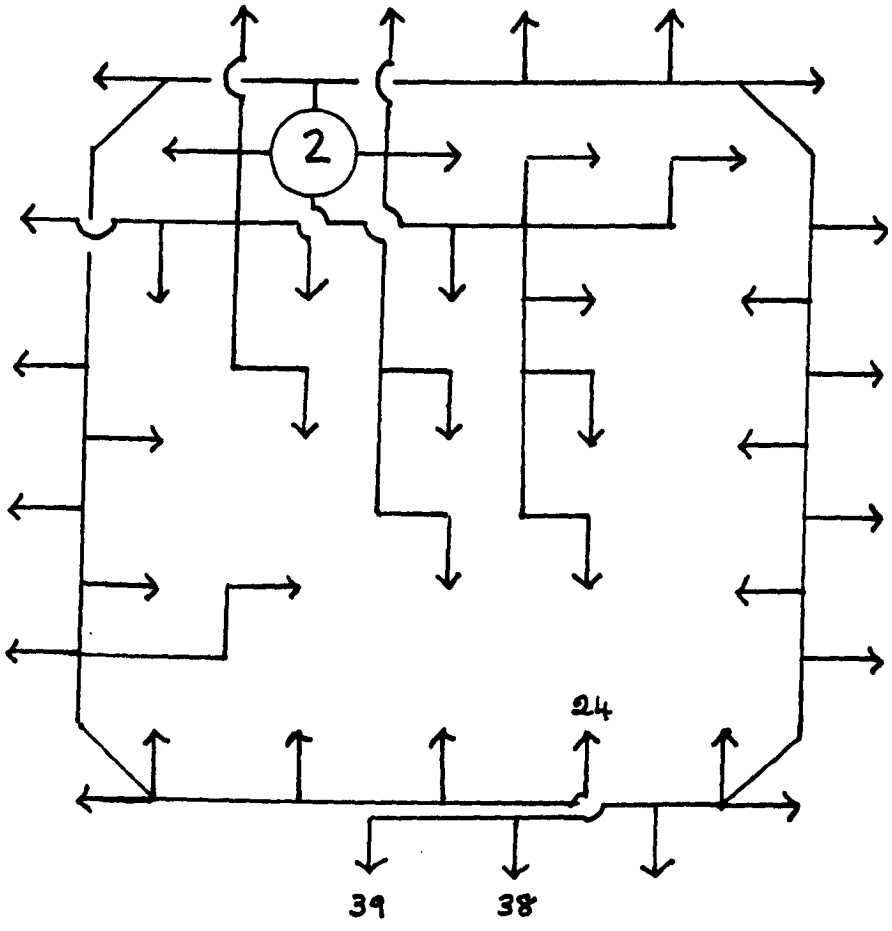


Fig. 14 Paths from internal Zone 2: morning trips

The routeing pattern in Figure 14 contributes 1 (the path to zone 24) to the count, in Table 10, of 11

paths from internal zones to internal zones which use the outer ring anticlockwise. It contributes 2 (the paths to zones 38 and 39) to the count of 19 paths from internal zones to external zones which use the outer ring clockwise.

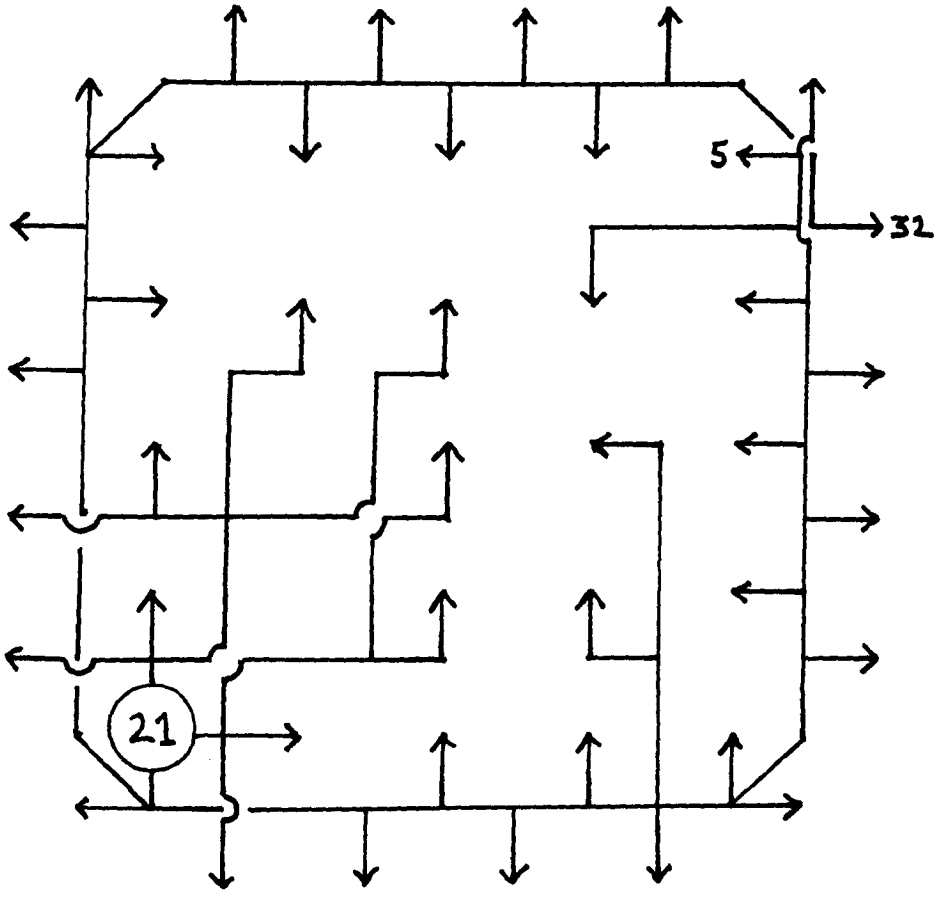


Fig. 15 Paths from internal Zone 21: evening trips

The routeing pattern in Figure 15 contributes 1 (the path to zone 5) to the count, in Table 10, of 11 paths from internal zones to internal zones which use the outer ring anticlockwise. It contributes 1 (the path to

zone 32) to the count of 7 paths from internal zones to external zones which use the outer ring clockwise.

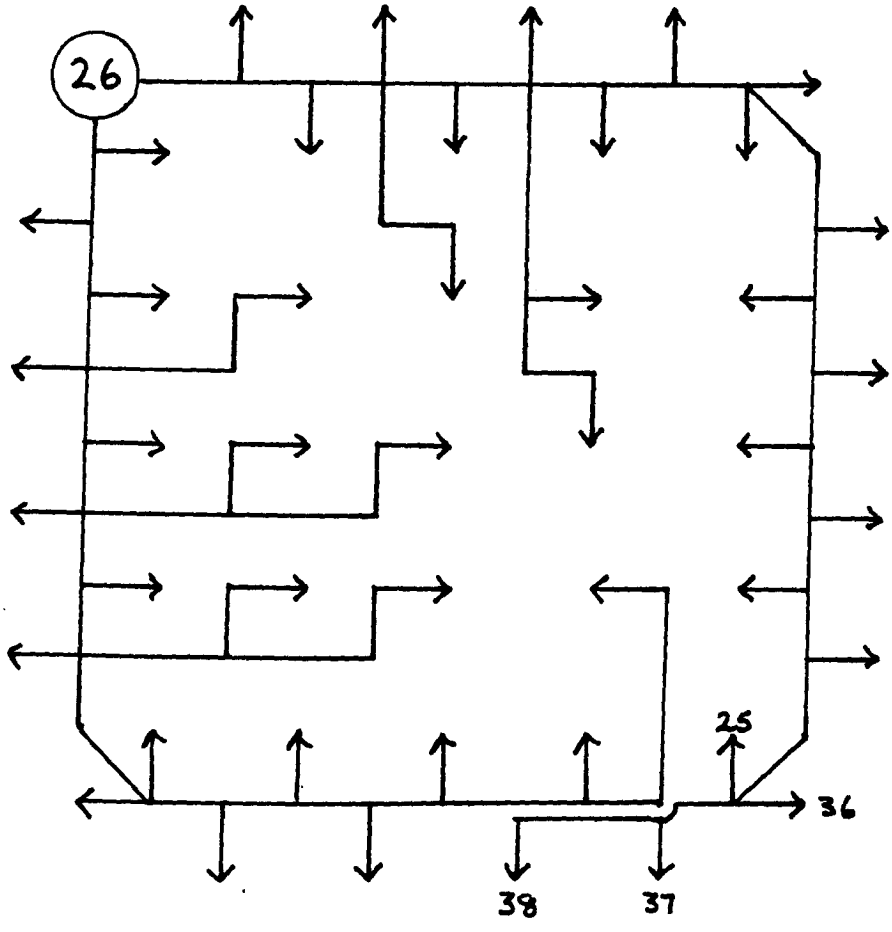


Fig. 16 Paths from external Zone 26: morning trips

The routeing pattern in Figure 16 contributes 1 (the path to zone 25) to the count, in Table 10, of 14 paths from external zones to internal zones which use the outer ring clockwise. It contributes 3 (the paths to zones 36, 37 and 38) to the count of 56 paths from external zones to external zones which use the outer ring

clockwise.

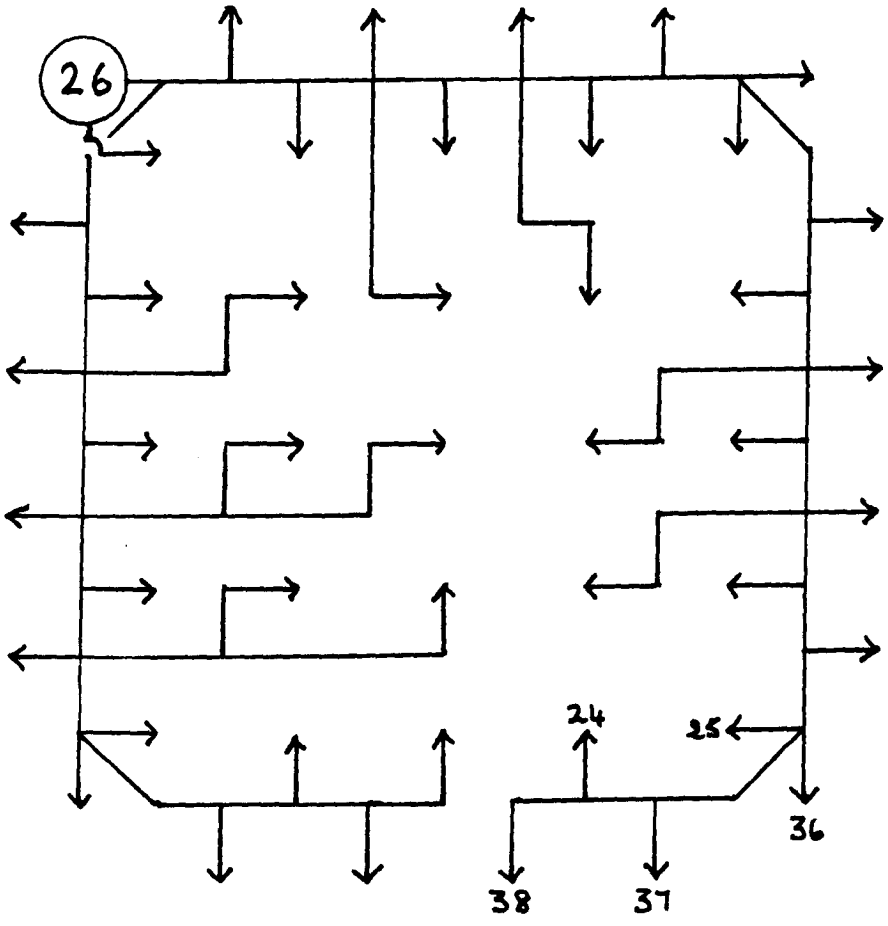


Fig. 17 Paths from external Zone 26: evening trips

The routeing pattern in Figure 17 contributes 2 (the paths to zones 24 and 25) to the count, in Table 10, of 24 paths from external zones to internal zones which use the outer ring clockwise. It contributes 3 (the paths to zones 36, 37 and 38) to the count of 56 paths from external zones to external zones which use the outer ring clockwise.

In the analysis which follows only four categories of paths are considered. First those paths between pairs of zones just inside the outer ring, referred to simply as internal zones, will be considered. Paths to or from zones further inside the outer ring were not considered because they tend to be direct, and paths using the outer ring seem more interesting. Secondly, paths from internal to external zones are considered; thirdly, paths from external to internal zones; and lastly paths between pairs of external zones.

When these paths use the outer ring, only those using the longer orbital path are analysed. In the case of paths between diametrically opposite zones, all those using the outer ring are analysed. The analysis is to find how many of these paths use the orbital route clockwise and anticlockwise. This analysis is made separately for morning and evening peak trip matrices. The analysis is shown in Table 10.

TABLE 10

ANALYSIS OF PATHS FOR THE M25 NETWORK

	Morning peak		Evening peak	
Paths for zone types	cw.	acw.	cw.	acw.
Internal to internal	6	11	5	11
Internal to external	19	1	7	3
External to internal	14	2	24	0
External to external	56	0	56	0

cw. = clockwise. acw. = anticlockwise.

For paths between pairs of internal zones, about twice as many use an anticlockwise route as use a clockwise route for both morning and evening peak matrices. In contrast, paths between pairs of external zones are all clockwise.

The paths from internal zones to external zones tend to use clockwise orbital routes more than anticlockwise. This tendency is much more marked in the morning than in the evening. In the morning the flow from internal to external zones is lighter than it is in the evening.

The paths from external zones to internal zones also tend to use clockwise orbital routes more than anticlockwise. This tendency is much more marked in the evening than in the morning. In the evening the flow from

external to internal zones is lighter than it is in the morning.

There does not seem to be a simple explanation of why these tendencies are affected by the difference in the trip matrices in the way they are. Two changes have been introduced between the morning and evening trip matrices; the internal to external demand has been increased from 10 to 40 trips per O-D pair and the external to internal demand has been decreased from 40 to 10 trips per O-D pair. The next step to establishing an explanation would be to make these changes one at a time. The intermediate step might represent a mid-day trip matrix. Further steps would then involve making the changes more gradually.

For the Hazel Grove network a similar analysis of the tendency of paths to use the outer ring in the clockwise or anticlockwise direction is made. Only one trip matrix was available and it was very skewed in that very many of the trips were destined for one part of the network. The lack of symmetry in the network and the skew in the matrix preclude much deduction from the analysis. In this assignment, paths make significant use of the central spine so these paths are given a place in the analysis as well as those which use the outer ring. The results of the analysis are shown in Table 11.

TABLE 11

ANALYSIS OF PATHS HAZEL GROVE

		Along outer ring	
Paths for zone types	Spine	cw.	acw.
Internal to internal	48	11	30
Internal to external	46	52	73
External to internal	22	2	8
External to external	55	31	45

cw. = clockwise. acw. = anticlockwise.

The Hazel Grove trip matrix is dominated by trips between pairs of external zones; just the opposite to the model trip matrix created for the M25 network. Another major difference is that junctions on the outer ring of the M25 network were modelled with flyovers whereas no junctions in the Hazel Grove network were modelled that way. These differences are too great for much weight to be attached to any consistency in tendencies to use the outer ring clockwise or anticlockwise. It was however the fairly marked tendency to use the outer ring anticlockwise in the Hazel Grove network which prompted the analysis of the M25 paths. In the latter case this tendency was only evident for paths between pairs of internal zones.

There are many spatial properties of the vine of routes from a common origin. In nature, trees can be classified according to the way their branches spread out. In fruit culture, certain formations are actually

encouraged. The vines which the CROWN design tool creates are two-dimensional and the ones analysed here have two main branches curving round the outside. These main branches sometimes extend over the top. The magnitude and direction of such extensions is the property analysed above.

To analyse a property one has to be able to define it and count its occurrence. In the course of the project other spatial properties relevant to traffic were analysed. One was the tendency of the tips of the vines to terminate in a left turn or a right turn. Another was the tendency of the branches to pass clockwise or anticlockwise round blocks. As no clear pattern was emerging the analysis is not recorded here.

The effect of choosing minimum conflict routes is to avoid some turning movements altogether; an analysis of the unused turning movements is included in the next section.

7.7 TRAFFIC CONTROL MEASURES

The interest in the unused turning movements centres on the fact that they indicate streets whose capacity could be increased by being designated one-way, and turns which could be banned. These traffic control measures would reduce conflict locally and might encourage drivers into more efficient routeing patterns. They are

consistent with the pattern found by the design tool. The simple weighting system used did not reflect any priority at junctions, so the routeing patterns suggest where the priorities should be rather than vice versa.

7.7.1 Analysis of proportions of unused elements

The unused elements are analysed both in terms of numbers of links which could be made one-way, and in terms of the numbers of unused turning movements segregated by T-junction and crossroads. The data are given in Tables 12 and 13.

TABLE 12

M25 NETWORK WITH EVENING PEAK TRIP MATRIX

Number of links used only one way: 2 out of 240.

Unused turning movements: 166 out of 1032,

at crossroads:	37	left turns
	43	straight on
	86	right turns
at T-junctions:	none.	

TABLE 13

HAZEL GROVE

Number of links used only one way: 16 out of 127.

Unused turning movements: 142 out of 501,

at crossroads:	20	left turns
	27	straight on
	32	right turns
at T-junctions:	12	left turns
	51	right turns.

In the Hazel Grove network about a quarter of the possible turning movements are not used compared with only about a sixth in the M25 network.

7.7.2 The locations of unused elements

The actual location of these unused movements is more interesting for the Hazel Grove network; the streets which could be made one-way are shown in Figure 18. These streets are marked with a single arrow to distinguish them from the two streets which the local authority has already designated as one-way; the latter are marked with a double arrow. The design tool has effectively 'created' three one-way circulatory systems and six other one-way streets.

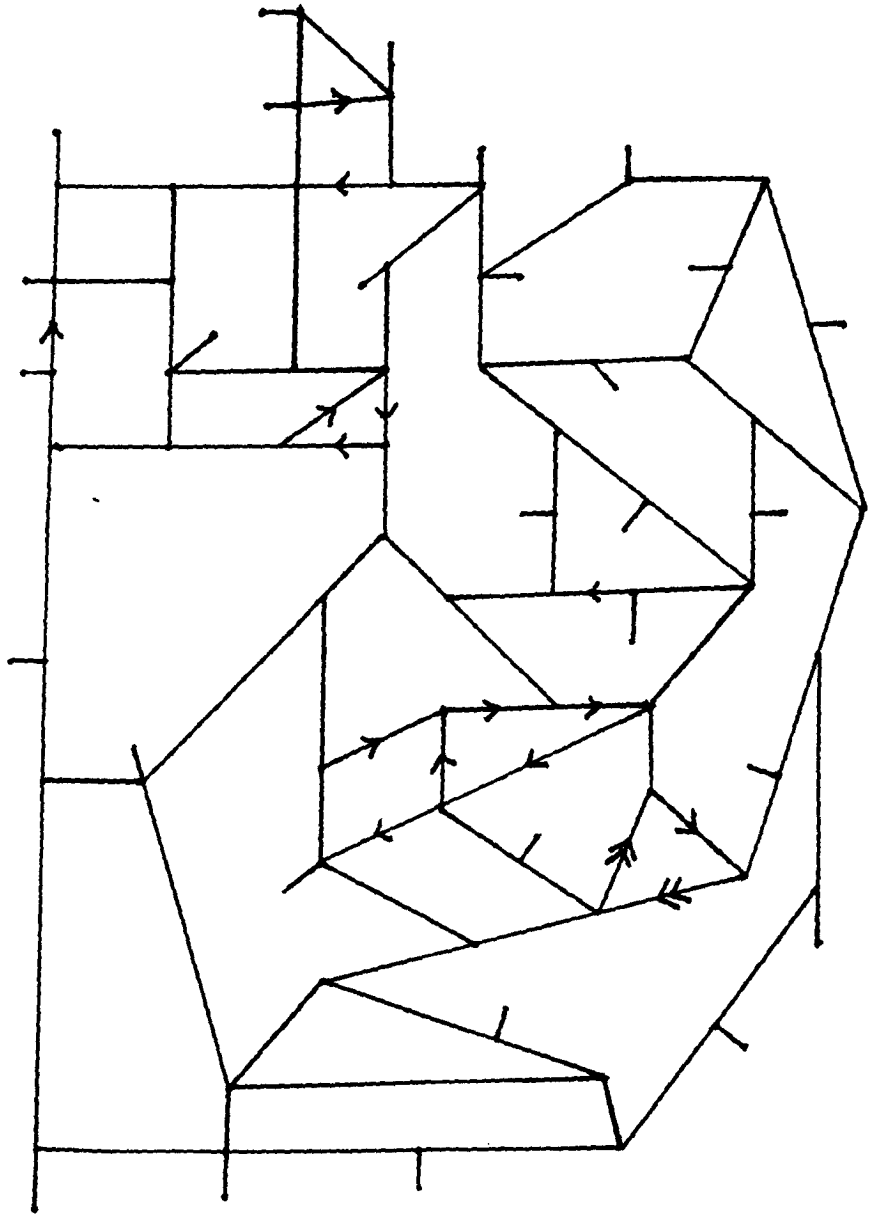


Fig. 18 One-way streets for Hazel Grove

The location of unused turning movements in the Hazel Grove network shows how conflicts are avoided. The road joining two of the one-way circulation systems exhibits an interesting feature. This part of the road network is shown in Figure 19. The four T-junctions, shown by a circle, are examined in detail. The Circulation System for these four junctions is shown in Figure 20, with the unused movements shown distinctly. There are no crossing conflicts at these junctions. The design tool has effectively turned this road into a dual carriage-way.

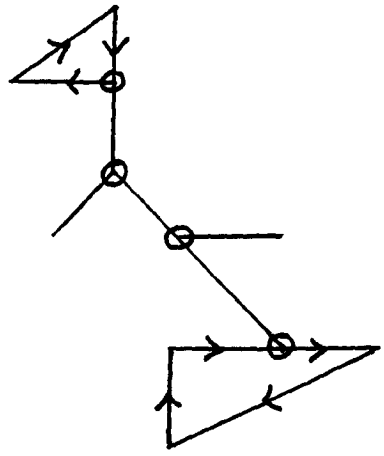


Fig. 19 Two one-way circulatory systems for Hazel Grove

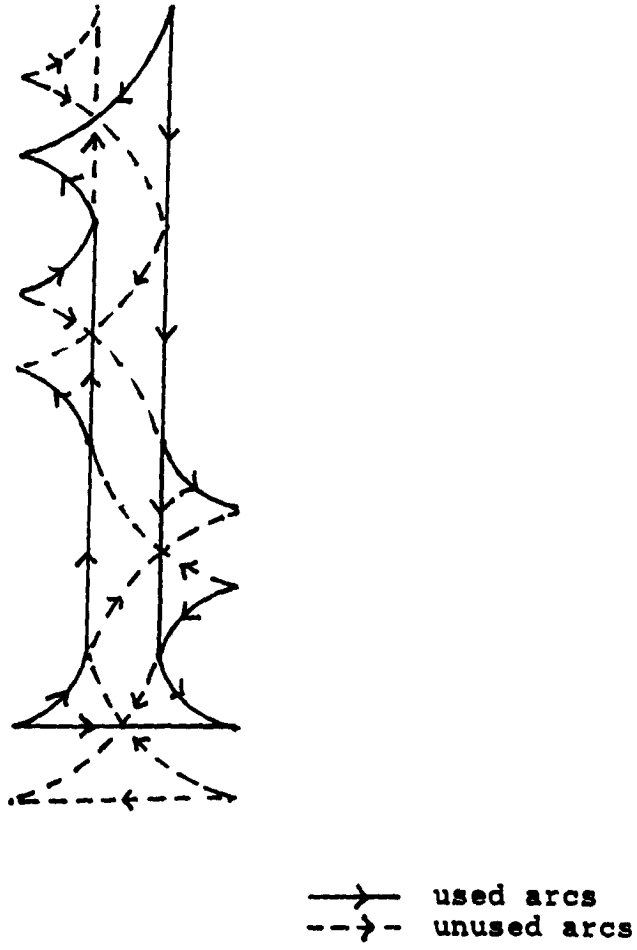


Fig. 20 The Circulation System for four T-junctions

The spatial distribution of the total flows was described in Section 7.5, but one aspect of that distribution is more appropriately described in the context of the one-way streets 'created' by the CROWN design tool. There are several blocks of streets where most of the traffic is flowing clockwise round the block. There is one block, sharing a link with one of these clockwise blocks, where most of the traffic is flowing anticlockwise. These features may remind one of eddies in a fluid. They are shown in Figure 21.

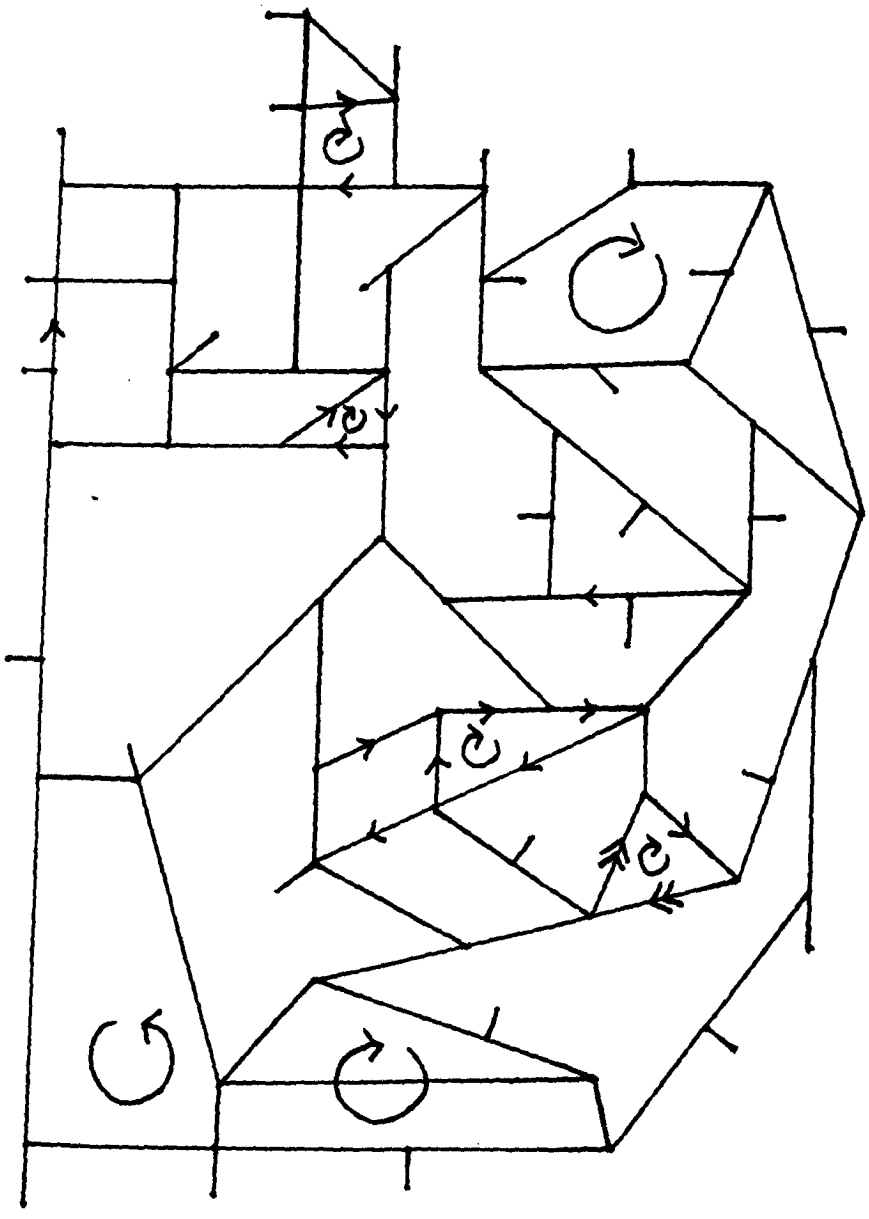


Fig. 21 'Eddies' of traffic in Hazel Grove

7.7.3 Designing traffic control measures

The design tool gives the traffic manager the details of a routeing pattern which reduces conflict for the given road network and trip matrix. The results could be used to design traffic management measures which would encourage drivers to use those routes. That encouragement could range from automatic route guidance to local measures which force or encourage drivers to use one street rather than another.

The presently available method of direct route guidance is by signposts. The diagrams one can draw from the output in the file FLINKTRE.DAT show the vines of routes from each origin and thus the route which should be taken to each destination. Hierarchical signing on hierarchical networks could go some way to directing traffic onto these routes; it would tend to compress traffic onto single routes rather than multiple routes.

Automatic on-board route guidance is being developed for trial in the London area. The most obvious criterion for the choice of route would be minimum journey time, especially in view of the fact that current link times reflecting the effect of any incident or roadworks could be fed to the on-line computer, and the advised routes updated accordingly. This obvious choice may not be the best either for reducing the level of congestion occurring in the system or for safety at junctions. The routes found by the CROWN design tool have much to commend

them as an alternative to those found to mimimise journey time.

The one-way streets found by the design tool would force drivers towards the desired routeing pattern. Their likely effect on traffic movements could be tested using a conventional traffic assignment program. The unused turning movements could be banned, again forcing drives towards the desired routing pattern. Their likely effect could be tested by heavy turn penalties in a conventional assignment program.

Drivers could also be encouraged towards the desired routeing pattern by various means. When changes are made, those who regularly use the network are expected to discover the improvements and switch their routes to make use of them over a period of time. The capacity of particular roads and junctions can sometimes be increased.

Those roads where an increase in capacity could be expected to provide indirect guidance could be identified from the plot of the results in the file FLINKSUM.DAT. Capacity can be increased by widening the road or restricting on-street parking.

The capacity of junctions can be increased both by channelling traffic appropriately and by preferential signal timings. Shifts in signal offsets and cycle

timings which reduce the delay for existing traffic flows are identified by using programs, such as TRANSYT and SCOOT. Those involved in managing traffic where there are either co-ordinated or isolated traffic signals already realise that shifts which affirm the existing flows may not be most beneficial in reducing congestion. However, the flows used as input to TRANSYT do not have to be existing flows; there is no reason why an alternative set of flows, such as those provided by the CROWN design tool, could not be used. TRANSYT would then optimise the performance index for these flows. If the resulting shifts were implemented, drivers might well find shorter queues where conflict minimising routes produced the heavier flows. They might even respond by trying out an alternative route. In this way the traffic manager could break away from his present passive response to existing flows.

7.8 CONCLUSION

The design tool has been tested with several networks. It finds routes to reduce the amount of conflict between streams of traffic at junctions by using less direct routes for some of the traffic. The iterative process converges rapidly. Single cycles of iteration with these networks take between 20 and 40 minutes each. In further development of the design tool, an attempt will be made to reduce these times. The resulting values of the objective function are only slightly sensitive to the

options chosen for a particular run.

Examples have been given of the kinds of insights into solutions to the traffic management problem which have been made possible with the development of this tool. The tool has been demonstrated with a real network and with idealised networks. The tool can be used, in a general way, to help us explore the routeing potential for different types of network.

CHAPTER 8

CONCLUSION

8.1 THE VISION REALISED

Various mathematical programming methods for solving the problem to find conflict minimising routes were investigated. The objective was to develop a method which would work reasonably well under realistic conditions. As a result of the investigation, the heuristic method for improving a start-up assignment of traffic was chosen. Two particular hindrances to this improvement were identified. One is the effect of what were called 'ghost costs'; these were easy to eliminate. The other is what was called 'mutually beneficial sightseeing'. The opportunity for this to occur might be reduced by redefining the subproblems, which are solved during the iterative process; one could try defining each subproblem as involving traffic with a common destination rather than a common origin.

The second formulation for solution by ILP showed up the way that the topology of the network restricts paths whose end points are fixed so that there is a certain amount of conflict between them; the conflict between a pair of paths can be split into topologically essential conflict, a path conflict and a path pair conflict.

Once the heuristic method had proved satisfactory for small test networks, it was tried out with bigger and realistic networks. The effort involved in preparing a complete specification of the Circulation System from a road network diagram might deter practising traffic engineers from using the design tool. An algorithm was designed to perform this task automatically. The design of these algorithms proved to be a quite a challenge. The design tool thus consists of three programs. POLYARCS prepares certain input files for the other two programs. The main program POLYSEND assigns traffic to routes chosen to reduce the amount of conflict at junctions. POLYLINK processes the output files from POLYSEND to express the results in terms of links in the original road network.

The operation of the program suite was validated with test networks. Its use was demonstrated with various networks of a more realistic size. The resulting routeing patterns were plotted on diagrams of the networks and their features studied. Conflict was sometimes reduced by not making use of all possible manoeuvres at junctions. These unused manoeuvres point to suitable one-way streets and banned turns. The effects of designating these streets for one-way operation and banning these turns could be assessed using a conventional assignment.

8.2 THE VISION AMENDED

The ideal of finding a globally optimal solution to the problem had to be abandoned. Any practitioner realises that the mathematical model of a problem is only an approximation to the real world, so when the best solution to his problem proves impossible to find in a reasonable time, he has to be content with finding a good solution. Provision is made for him to find several reasonably good locally optimal solutions.

No provision is made in the tool for taking account of crossing conflicts between traffic from the same origin. To do so would involve redefining the subproblems to involve the traffic between one O-D pair at a time. This would increase the number of subproblems by a factor of (n - 1). As common sense would suggest that such paths would usually fan out without crossing, the large increase in the number of computations required did not seem worthwhile.

8.3 THE VISION EXTENDED

Throughout the thesis, refinements to the design tool have been suggested. It is intended that these should be made during further development of the design tool. Perhaps the most obvious one is the inclusion of capacity restraint. The network algorithm has provision for this. The test for spare capacity would have to be applied instead of being bypassed. Residual capacity

would also have to be recomputed after the solution of each subproblem. However, it would involve the building of a fresh vine for each O-D pair because some branches already in the vine might not have enough spare capacity to accommodate the flow required for the next O-D pair. This would increase the number of computations required by a factor of (n -1).

Details of the volume of flow, in each direction through each junction, is accessible from the output from the main program POLYSEND, but it would be more convenient to have these turning volumes expressed in terms of the links in the road network rather than in their present form, which refers to the more complex Circulation System.

The data files which are output by the program POLYLINK can never be an entirely satisfactory medium for the solution to a spatial problem. A graphics environment is the proper one in which to display the solution to a traffic problem. Such an environment requires that the nodes (junctions) of the road network be given co-ordinates in co-ordinate file. It is expected that the next stage of development will include making output files from the CROWN design tool compatible with MVGRAF the graphics program produced by the industrial collaborators MVA Systematica.

The single criterion, of reducing conflict between streams of traffic, could be combined with the criterion of reducing journey distance, if movements along links as well as junction manoeuvres were represented in the Circulation System. The network synthesis program could easily be adapted to represent these movements as well.

Diverging conflicts were not included because they were deemed to be of secondary importance compared with merging and crossing conflicts. The lists of conflicting arcs could easily be extended to include pairs of diverging arcs.

8.4 FURTHER VISIONS

The CROWN design tool could be adapted to find routes which specifically reduced the potential for accidents. The adaptation would involve replacing the cost functions with the accident predictive relations developed at the Transport and Road Research Laboratory. A paper in which this possibility was explored was presented at the Universities Transport Study Group Annual Conference in January 1990 (Wackrill 1990).

The concept of minimising conflict between streams of traffic can be applied to streams of pedestrians. A map has recently appeared at Kings Cross Station in London showing the paths into which rush hour pedestrians are guided to reduce conflict. A sketch of this map is shown

in Figure 22.

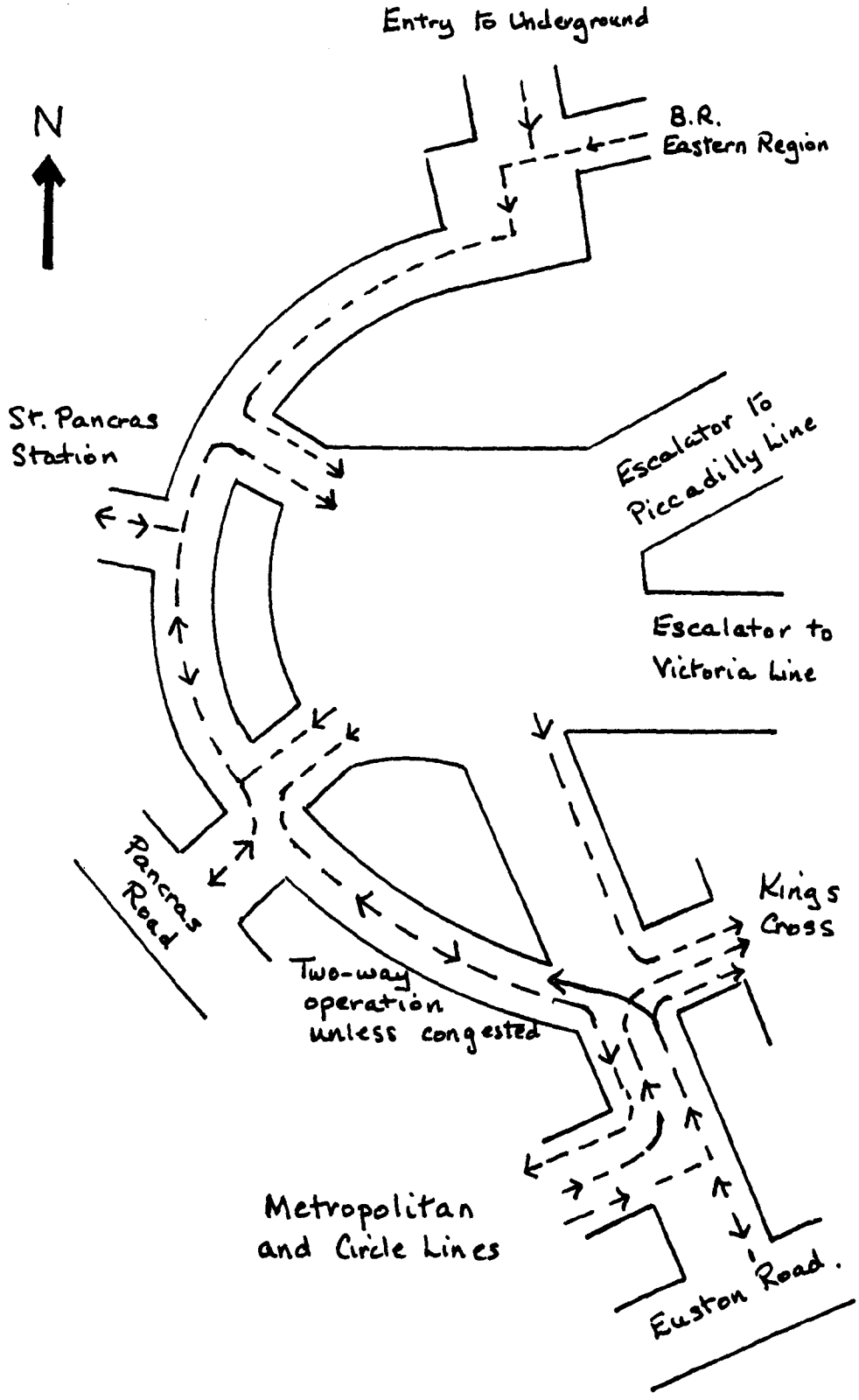


Fig. 22 Sketch of pedestrian movements at Kings Cross.

In principle, the interaction between vehicle and pedestrian flows could be modelled in the CROWN design tool. This is relevant to the siting of bus stops and pedestrian crossings.

8.5 THE HEAVENLY VISION

"Each of you should look not only to his own interests, but also to the interests of others." Philippians 2:4.

Traffic management is a social responsibility. This approach to traffic management looks not only to the interests of the individual driver but also to those of the group of drivers as a whole. It looks beyond the driver to include his passengers, in reducing the potential for accidents. It looks beyond road users to urban inhabitants in reducing the amount of noise and air pollution. May this heavenly vision do some earthly good.

REFERENCES

ALLSOP, R. E. and CHARLESWORTH, J.A. 1977. Traffic in a signal-controlled network: an example of different signal timings inducing different routeings. Traffic Engng & Control 18 (5), 262-264.

BARR, R.S., GLOVER, F. and KLINGMAN, D. 1974. An improved version of the Out-of-Kilter mathod and a comparative study of computer codes. Math. Prog. 7(1974), 60-86.

BELL, M.G.H. 1990. Environmental impact of traffic. Presented at the 22nd Universities Transport Studies Group Annual Conference. (Unpublished).

BOYCE, D.E. 1988 Route guidance systems for improving urban travel and location choices. Transpn. Res. A Vol. 22A, No. 4, 274-281.

HOLROYD, E.M., and MILLER, A.J. 1966 Route Crossings in Urban Areas. Australian Road Research Proceedings 237, 394-419.

SHEFFI, Y. 1985. Urban transportation networks: equilibrium analysis with mathematical programming methods. Prentice-Hall 203-229.

STOELHURST, H.J., and ZANDBERGEN, A.J. 1990. The development of a road pricing system in The Netherlands. Traffic Engng & Control 31 (2), 66-71.

SUMMERSGILL, I. 1988 Accident Predictive Relations for some Junction Types in Great Britain. P.T.R.C. Conference Proceedings 1988 163 - 178.

TURAN, P. 1977. A note of welcome. J. Graph Theory 1 (1979) 7-9.

WACKRILL, P.A. 1990. Routes chosen to reduce the potential for urban accidents. Presented at the 22nd Universities Transport Studies Group Annual Conference. (Unpublished).

WARDROP, J.G., 1952. Some theoretical aspects of road traffic research. Proceedings. Institution of Civil Engineers II(1), 325-378.

WRIGHT, C.C., 1978 Control of drivers' route choice: pipe dream or panacea? Transportation 7(1978) 193-210.

WRIGHT, C.C. 1979. Arcs and cars: an approach to road traffic management based on graph theory. Graph theory and combinatorics (R.J.Wilson, editor). London: Pitman, 133-146.

WRIGHT, C.C., APPA, G.M., and JARRETT,D.F. 1989 Conflict-minimising traffic patterns: a graph-theoretic approach to efficient traffic circulation in urban areas. Transportation Research 23A(2), 115-127, 1989.

WRIGHT, P.T., and SEMMENS, M.C. 1984. An assessment of the Denham roundabout conversion. Traffic Engng & Control 25 (9), 422-426.

APPENDIX 1

THE DIGRAPH MODELS

The purpose of this appendix is to relate the weights to the particular pairs of conflicting movements.

Diagrams show the arcs in each junction type. Approach vertices are labelled with A, exit vertices with E, and vertices on a roundabout with R.

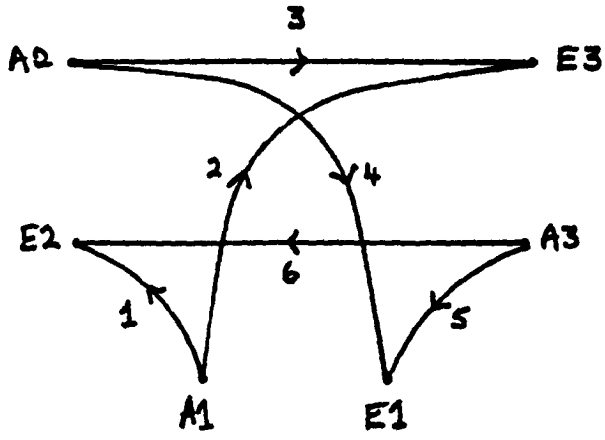
There follows a table containing a row for each arc, and an ordered list in that row of the arcs conflicting with that arc. The format of this table is then used to show the corresponding weights.

Under the heading 'Arguments in the WT3 (or WT4) array', a number n in row r and column c will imply that the nth element in the WT3 (or WT4) array is to be used for weighting the conflict between the arc in row r and the cth arc with which it conflicts. Where the pattern of conflicts and weights repeats itself, only one repeat is used to indicate the corresponding arguments.

Under the heading 'Default weights', the same table, or part of it, is used to show the actual default weights available in the program. These are 1 for a merge, 2 for a crossing and 3 for interlocking turns.

Where several junction types have the same diagram, the corresponding tables appear one after the other under these two headings. Each junction type has a name and a type number, the value of JT(X) for the type.

3-ARM JUNCTIONS

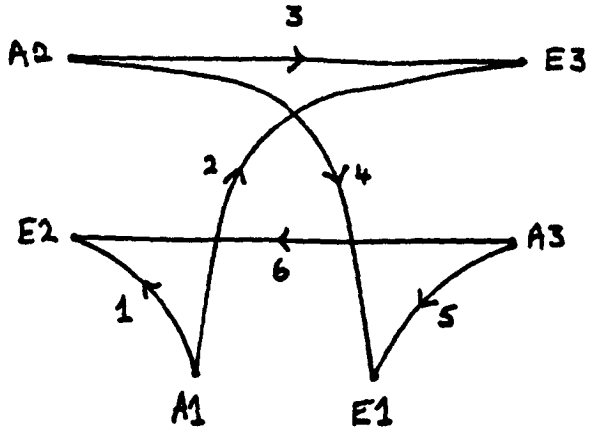


Arcs	Conflicting arcs 1st	2nd	3rd
1	6		
2	6	4	3
3	2		
4	2	6	5
5	4		
6	4	2	1

CORRESPONDING WEIGHTS

Arguments in WT3 array			Default weights		
FREE-FOR-ALL: JT(X) = 1			Repeat with three-fold symmetry.		
1			1		
2	3	4	2	2	1
PRIORITY: JT(X) = 2					
5			1		
6	7	8	2	2	1
9			1		
10	11	12	2	2	1
13			1		
14	15	16	2	2	1
SIGNALS: JT(X) = 3					
17			1		
18	19	20	2	2	1
21			1		
22	23	24	2	2	1
25			1		
26	27	28	2	2	1

3-ARM JUNCTIONS

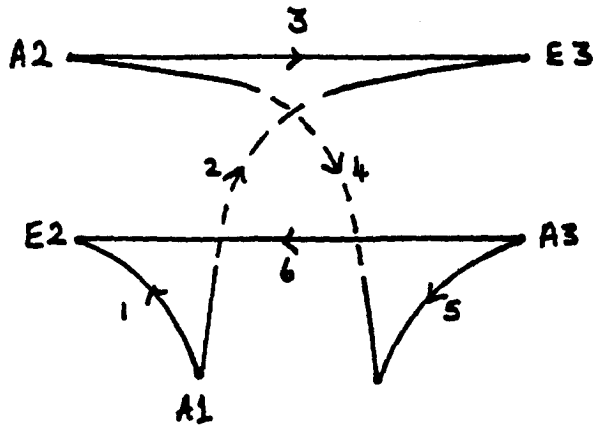


Arcs	Conflicting arcs 1st	2nd	3rd
1	6		
2	6	4	3
3	2		
4	2	6	5
5	4		
6	4	2	1

CORRESPONDING WEIGHTS

Arguments in WT3 array			Default weights		
MINI-ROUNDABOUT JT(X) = 4			Repeat with 3-fold symmetry.		
29			1		
30	31	32	2	2	1
ROUNDABOUT JT(X) = 5			Repeat with 3-fold symmetry.		
33			1		
34	35	36	2	2	1

3-ARM JUNCTION: GRADE SEPARATED

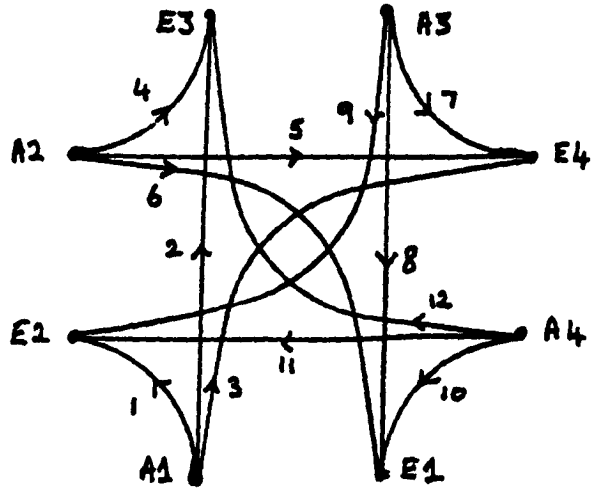


Arcs	Conflicting arcs 1st	2nd	3rd
1	6		
2	3		
3	2		
4	5		
5	4		
6	1		

CORRESPONDING WEIGHTS

Arguments in WT3 array	Default weights
GRADE-SEPARTATED JT(X) = 6	
37	1
38	1
39	1
40	1
41	1
42	1

4-ARM JUNCTION: FREE-FOR-ALL



Arcs	Conflicting arcs 1st	2nd	3rd	4th	5th	6th	7th
1	11	9					
2	11	9	6	5	12	4	
3	11	9	12	6	8	5	7
4	2	12					
5	2	12	9	8	3	7	
6	2	12	3	9	11	8	10
7	5	3					
8	5	3	12	11	6	10	
9	5	3	6	12	2	11	1
10	8	6					
11	8	6	3	2	6	1	
12	8	6	9	3	5	2	4

CORRESPONDING WEIGHTS

FREE-FOR-ALL JT(X) = 1 Repeat with 4-fold symmetry.

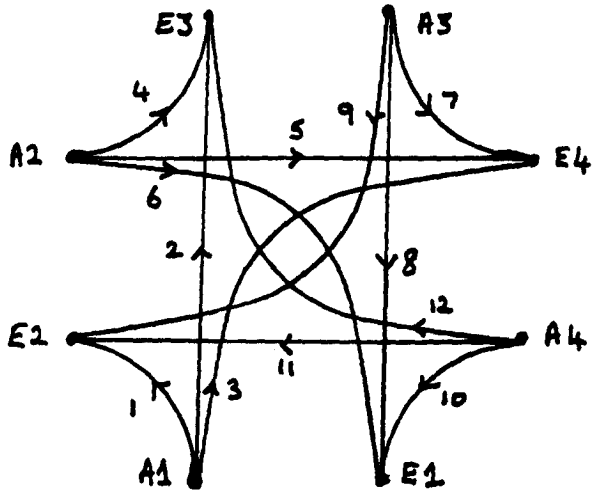
Arguments in WT4 array

1	2					
3	4	5	6	7	8	
9	10	11	12	13	14	15

Default weights

1	1					
2	2	2	2	1	1	
2	3	2	2	2	1	1

4-ARM JUNCTION: PRIORITY



Arcs	Conflicting arcs 1st	2nd	3rd	4th	5th	6th	7th
1	11	9					
2	11	9	6	5	12	4	
3	11	9	12	6	8	5	7
4	2	12					
5	2	12	9	8	3	7	
6	2	12	3	9	11	8	10
7	5	3					
8	5	3	12	11	6	10	
9	5	3	6	12	2	11	1
10	8	6					
11	8	6	3	2	6	1	
12	8	6	9	3	5	2	4

CORRESPONDING WEIGHTS

PRIORITY JT(X) = 2 Repeat with 2-fold symmetry.

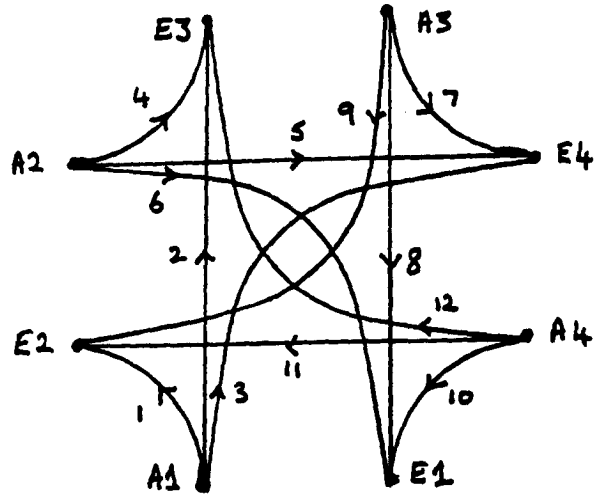
Arguments in WT4 array

16	17					
18	19	20	21	22	23	
24	25	26	27	28	29	30
31	32					
33	34	35	36	37	38	
39	40	41	42	43	44	45

Default weights

1	1					
2	2	2	2	1	1	
2	3	2	2	2	1	1
1	1					
2	2	2	2	1	1	
2	3	2	2	2	1	1

4-ARM JUNCTION: SIGNALS



Arcs	Conflicting arcs 1st	2nd	3rd	4th	5th	6th	7th
1	11	9					
2	11	9	6	5	12	4	
3	11	9	12	6	8	5	7
4	2	12					
5	2	12	9	8	3	7	
6	2	12	3	9	11	8	10
7	5	3					
8	5	3	12	11	6	10	
9	5	3	6	12	2	11	1
10	8	6					
11	8	6	3	2	6	1	
12	8	6	9	3	5	2	4

CORRESPONDING WEIGHTS

SIGNALS JT(X) = 3 Repeat with 4-fold symmetry.

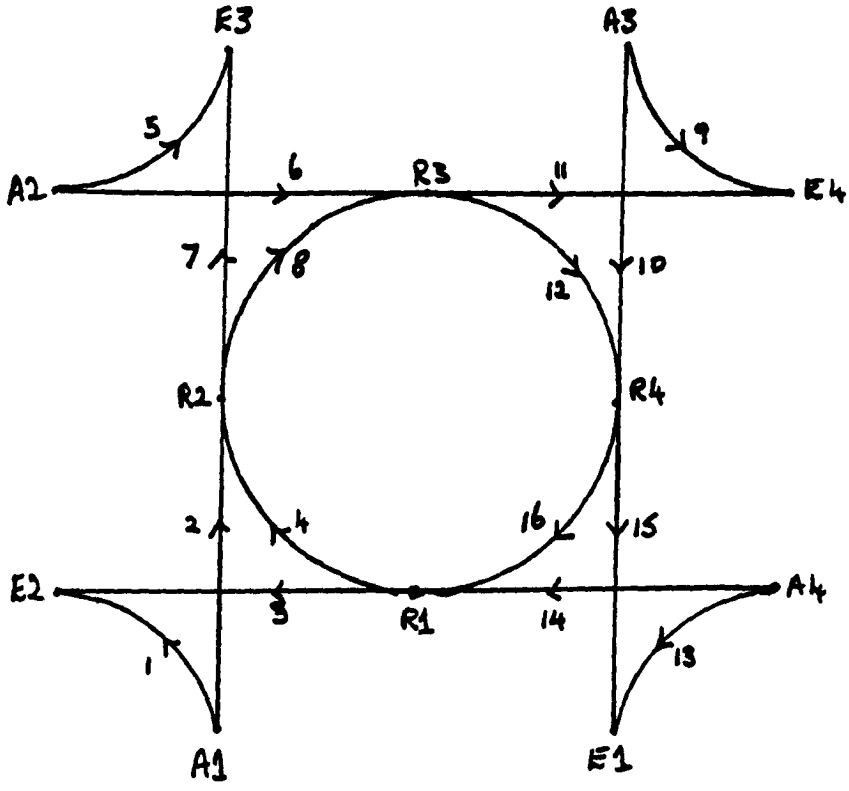
Arguments in WT4 array

46	47					
48	49	50	51	52	53	
54	55	56	57	58	59	60

Default weights

1	1					
2	2	2	2	1	1	
2	3	2	2	2	1	1

4-ARM MINI-ROUNDABOUT



Conflicting arcs

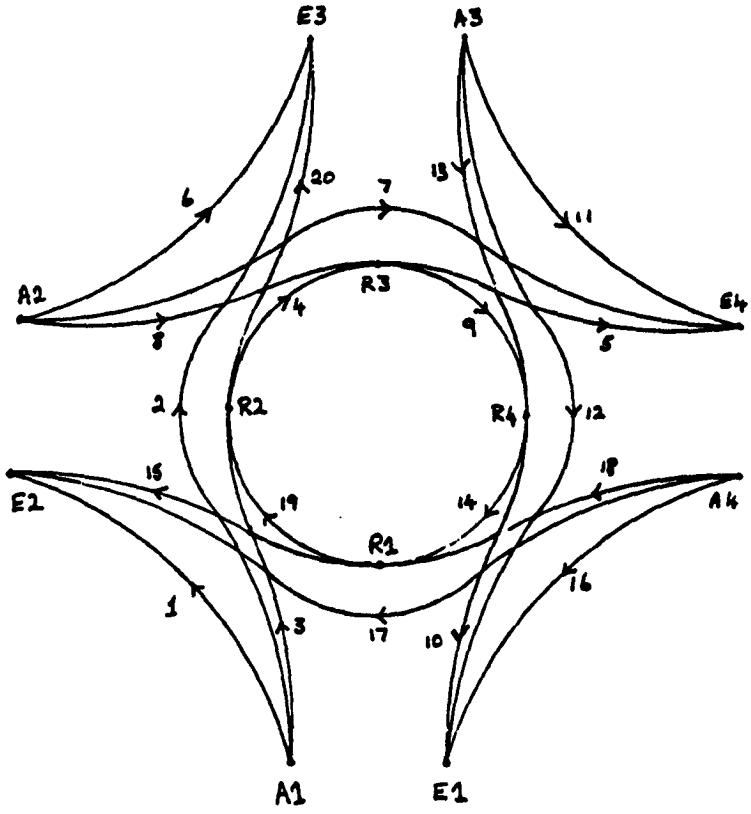
Arcs	1st	2nd
1	3	
2	3	4
3	2	1
4	2	
5	7	
6	7	8
7	6	5
8	6	
9	11	
10	11	12
11	10	9
12	10	
13	15	
14	15	16
15	14	13
16	14	

CORRESPONDING WEIGHTS

MINI-ROUNDABOUT JT(X) = 4 Repeat with 4-fold symmetry.

Arguments in WT4 array.		Default weights.	
61		1	
62	63	2	1
64	65	2	1
66		1	

4-ARM ROUNDABOUT



Conflicting arcs repeat with 4-fold symmetry.

Arcs	1st	2nd	3rd	4th	5th	6th
1	17	15				
2	17	15	8	7	6	20
3	17	15	19			
4	8					
5	13	12	7	11		

CORRESPONDING WEIGHTS

ROUNDABOUT JT(X) = 5 Repeat with 4-fold symmetry.

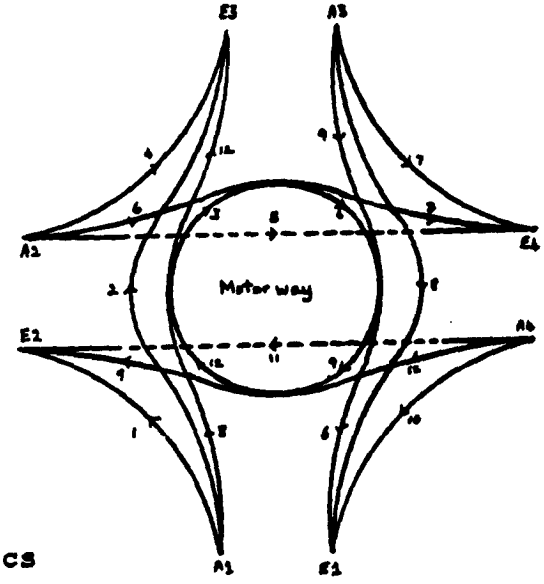
Arguments in WT4 array

67	68				
69	70	71	72	73	74
75	76	77			
78					
79	80	81	82		

Default weights

1	1				
2	2	2	2	1	1
2	2	1			
1					
2	2	1	1		

4-ARM GRADE SEPARATED JUNCTION



Conflicting arcs

Arcs	1st	2nd	3rd	4th	5th	6th
1	11	9				
2	9	6	4	12		
3	9	12	6	8	5	7
4	2	12				
5	7	3				
6	2	12	3	9	8	10
7	5	3				
8	3	12	10	6		
9	3	6	12	2	11	1
10	8	6				
11	1	9				
12	8	6	9	4	2	4

CORRESPONDING WEIGHTS

GRADE-SEPARATED JT(X) = 6 Repeat with 4-fold symmetry.

Arguments in WT4 array

83	84				
85	86	87	88		
89	90	91	92	93	94
95	96				
97	98				
99	100	101	102	103	104

Default weights

1	1				
2	2	1	1		
3	2	2	2	1	1
1	1				
1	1				
2	3	2	2	1	1

APPENDIX 2

USE WITH RIGHT HAND DRIVING

When the digraphs are reflected, or viewed from below 'the plane of the paper', they represent driving on the right. It follows that the mirror image of the Circulation System for left-hand driving on a road network is the Circulation System for right-hand driving on the mirror image of the road network.

This property can be exploited when using overhead projection for the digraph models of junctions, by placing the foil the other way up on the projector in countries where road users drive on the right. If the road network is symmetric, appropriate routeing patterns can also be shown in this way.

APPENDIX 3

DETECTION OF INCONSISTENCIES

Inconsistency in the input data is detected and reported in the following way. When all the vertices have been created, the program proceeds to create the arcs using a digraph model of the appropriate type for each junction. For each link meeting at the junction, the program checks whether the B node of the first link matches the node number of the junction; if it does the program proceeds as described in Chapter 2.

If the B node does not match the node number of the junction, the program goes on to check whether the A node matches, and proceeds as described in Chapter 2.

If, however, neither the A node nor the B node matches the junction node then, there must be some inconsistency. The program stops with a message indicating inconsistency at the junction specified by its node number. Either the link is correctly recorded in the link records but should not appear in this particular junction record, or the link record is faulty. In either case the user is alerted, and directed to the likely sources of his error.

APPENDIX 4

EXAMPLE TO ILLUSTRATE QP AND ILP SOLUTIONS

The road network used for this example is shown in Figure 1. The Circulation System is shown in Figure 2.

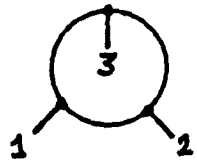


Fig. 1. The road network

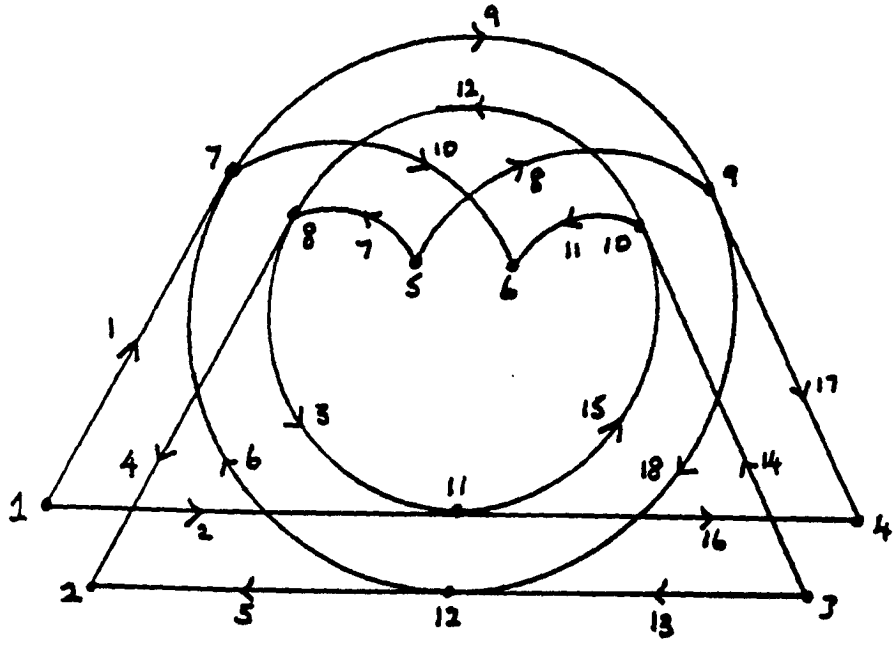


Fig. 2 The Circulation System

The trip matrix used is given below.

	1	2	3
1	-	1	3
2	1	-	1
3	1	4	-

4.1 THE VARIABLES

Three variables are defined for each arc of the Circulation System. They correspond to flows from the three Origin Vertices 1, 3, and 5. Consideration of the flow conservation constraints shows that, for example, there will be no flow from Origin Vertices 3 or 5 on Arcs 1 or 2. Similarly there will be no flow from Origin Vertex 1 on Arcs 4 or 5. Arc 3 will not carry flow from Origin Vertices 1 or 3. Arc 6 will not carry flow from Origin Vertices 1 or 5. This cuts down the number of variables required for the first 6 arcs from 18 to 8. The 24 variables are tabulated in terms of arcs and the origin vertices, in Table 1.

TABLE 1

Arcs	1	2	3	4	5	6
Flow from Vertex 1	x(1)	x(2)	-	-	-	-
Flow from Vertex 3	-	-	-	x(3)	x(4)	x(5)
Flow from Vertex 5	-	-	x(6)	x(7)	x(8)	-

Arcs	7	8	9	10	11	12
Flow from Vertex 1	-	-	x(9)	x(10)	x(11)	-
Flow from Vertex 3	-	-	-	x(12)	x(13)	x(14)
Flow from Vertex 5	x(15)	x(16)	-	-	-	-

Arcs	13	14	15	16	17	18
Flow from Vertex 1	-	-	x(17)	x(18)	x(19)	-
Flow from Vertex 2	x(20)	x(21)	-	-	-	-
Flow from Vertex 3	-	-	-	x(22)	x(23)	x(24)

The constraints for flow through intermediate vertices show that the number of variables can be reduced further. For instance, conservation of flow through Vertex 7, involves flow arriving on Arcs 1 and 6 and leaving on Arcs 9 and 10. For the flow from Vertex 3, this implies that x(5) = x(12). There are five similar equations for flow through the five other intermediate vertices. This reduces the number of variables required to 18. In the case of flow from Vertex 1 through Vertex 7, the equation obtained is x(1) = x(9) + x(10). For this particularly simple network, the eighteen variables are split into two sets, one being those single variables on the left sides of the six similar equations, and the other being the six pairs of variables on the right hand sides. Each of the six variables on the left hand sides can therefore be replaced by the sum of the pair of variables on the right sides. This reduces the number of variables required to twelve. A closer inspection shows that these variables correspond to the two arc-disjoint paths between each origin to destination pair of vertices (O-D pair). This problem involves six O-D pairs and just two possible paths between each pair. The problem is formulated in terms of these variables. The paths are referred to as clockwise (cw) or anti-clockwise (acw). The variables are defined in Table 2 below, and the path corresponding to variable x(n) will be referred to as Path n.

TABLE 2

From vertex	to vertex	flow cw	flow acw
1	4	x(1)	x(2)
1	6	x(3)	x(4)
3	2	x(5)	x(6)
3	6	x(7)	x(8)
5	2	x(9)	x(10)
5	4	x(11)	x(12)

This leaves a very simple and convenient set of constraints for the flow into each destination vertex. There is one constraint for each non-zero element of the trip matrix, with its right hand side equal to that element.

For the objective function each pair of conflicting arcs has to be identified and the variables representing flow on the one multiplied by those representing flow on the other.

4.2 SOLUTION BY QUADRATIC PROGRAMMING

The solution is obtained using the MPCODE software, developed at the London School of Economics, and based on Beale's method. The solution is used to demonstrate that what were called 'ghost costs' in Chapter 3, Subsection 3.5.3, are not only included but also inhibit further improvement of the solution. The data file for the formulation in Section 3.1 is shown below. It consists of twelve records as follows:

Record 1 - some words to control the output, NONE asks for the minimal amount of output.

Record 2 - M (no. of constraints), N (no. of variables), and NUMQ (the dimension of the D matrix).

Record 3 - triples of 2 integers and a real or integer number, for the row, column and coefficient of each non-zero element in the D matrix.

Record 4 - 'MAX' or 'MIN', M, N,

Record 5 - pairs of an integer for the variable, and a real or integer for its coefficient for the non-zero terms in the linear part of the objective function.

Record 6 - triples of three elements for each constraint, an integer for the constraint number, a letter 'L','G', or 'E' for its sign, and a real or integer number for the right hand side.

Records 7 onwards - Separate records for each row of the A matrix, starting with the row identifier followed by two commas, then for each non-zero element a pair consisting of an integer to identify the variable and a real or integer number for its coefficient.

The problem can be solved with different trip matrices merely by making changes to Record 6.

The data file for this problem

```
'NONE',/
6,12,12,/
1,2,1, 1,7,1, 1,9,1, 1,11,1, 1,12,1,
2,6,2, 2,7,1, 2,8,1, 2,9,1, 2,10,1, 2,11,1, 2,12,1,
3,4,1, 3,6,1, 3,7,1, 3,8,1, 3,9,1, 3,11,1,
4,6,2, 4,7,2, 4,8,1, 4,10,1, 4,12,1,
5,6,1, 5,9,1, 5,10,1,
6,7,2, 6,9,3, 6,10,1, 6,11,1, 6,12,2,
7,8,1, 7,9,2, 7,10,1, 7,11,1,
8,9,1, 8,12,1,
9,10,1, 9,12,1,
11,12,1,/
'MIN',6,12,/
/
1,'E',1, 2,'E',3, 3,'E',1, 4,'E',1, 5,'E', 1, 6,'E',4,/
1,,1,1, 2,1,/
2,,3,1, 4,1,/
3,,5,1, 6,1,/
4,,7,1, 8,1,/
5,,9,1, 10,1,/
6,,11,1, 12,1,/
```

The relevant parts of the output file are shown below. The program has found a local optimum which is not the global optimum. The way the variables are defined provides for a very relevant interpretation of the partial derivatives of the quadratic function. The values are the costs, in terms of conflict, of using each of the twelve paths. In each case the cost of the alternative path for any O-D pair is greater than or equal to the cost of the path used; the partial derivative is at a minimum so the program terminated with this solution.

Part of the output file for the solution

IN THE FOLLOWING OUTPUT, THE C VECTOR CONTAINS THE PARTIAL DERIVATIVES OF THE QUADRATIC FUNCTION.

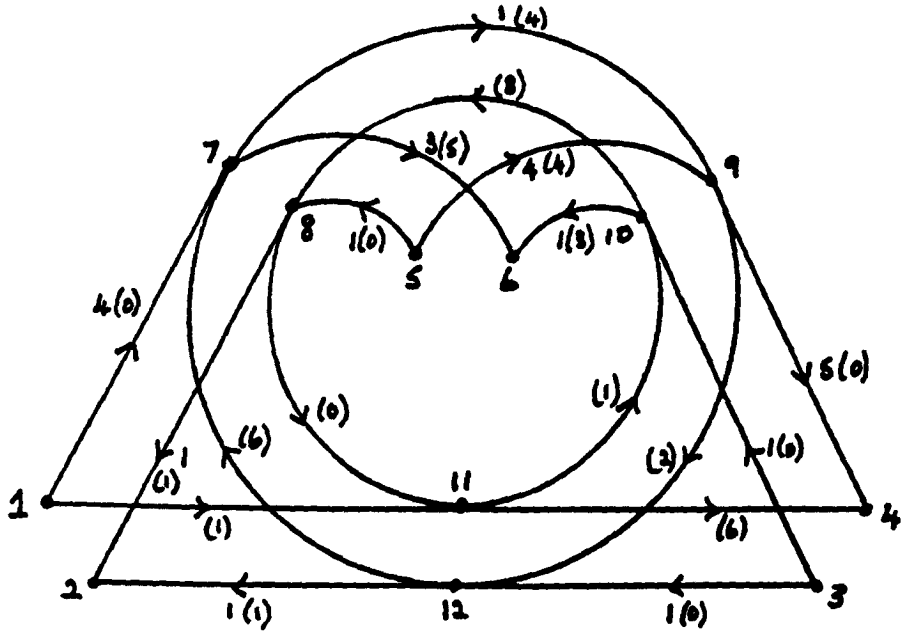
****** QUADRATIC PROGRAM OPTIMUM ******

Value of Minimand = 20.00000000

PRIMAL SOLUTION

Variable j	Name	c(j)	x(j)	yA-c	Status
1		4.000	1.000	0.000	BS
2		7.000	0.000	-3.000	DN
3		5.000	3.000	0.000	BS
4		5.000	0.000	0.000	DN
5		1.000	1.000	0.000	BS
6		9.000	0.000	-8.000	DN
7		9.000	0.000	-7.000	DN
8		3.000	1.000	0.000	BS
9		7.000	0.000	-6.000	DN
10		1.000	1.000	0.000	BS
11		4.000	4.000	0.000	BS
12		6.000	0.000	-2.000	DN

The total flows, with the costs in brackets, are shown in Figure 3. The cost of the unused paths all incorporate what were called 'ghost costs'. For each such path, this is the cost due to the flow on the other of the pair of paths between the same O-D pair. If that path was actually used, the flow would no longer be on the other path; so the cost of using it goes down as soon as it is used. When the ghost costs are removed, the costs are reduced to the values shown below. The variables have been defined in such a way that the terms which give rise to ghost costs can be identified and removed from the objective function.



Notation flow(cost)

Fig. 3 The first solution

Costs reduced by the removal of ghost costs

Variable j	c(j)	x(j)	yA-c	Status
1	4.000	1.000		
2	6.000	0.000		
3	5.000	3.000		
4	2.000	0.000		
5	1.000	1.000		
6	8.000	0.000		
7	9.000	0.000		
8	3.000	1.000		
9	6.000	0.000		
10	1.000	1.000		
11	4.000	4.000		
12	2.000	0.000		

With these reduced costs the value of the partial derivative could be reduced by making use of some of the unused paths. Path 4, anticlockwise from Vertex 1 to Vertex 6, is cheaper than the used Path 3. Path 12, anticlockwise from Vertex 5 to Vertex 4, is also cheaper

than the used Path 11.

When the ghost costs are removed from the objective function, a better solution is obtained. The relevant part of the output file is shown below, and the solution is shown in Figure 4.

Part of the output file for the better solution

IN THE FOLLOWING OUTPUT, THE C VECTOR CONTAINS THE PARTIAL DERIVATIVES OF THE QUADRATIC FUNCTION.

****** QUADRATIC PROGRAM OPTIMUM ******

Value of Minimand = 10.00000000

PRIMAL SOLUTION

Variable j	Name	c(j)	x(j)	yA-c	Status
1		5.000	1.000	0.000	BS
2		6.000	0.000	-1.000	DN
3		6.000	0.000	-5.000	DN
4		1.000	3.000	0.000	BS
5		1.000	1.000	0.000	BS
6		13.000	0.000	-12.000	DN
7		13.000	0.000	-9.000	DN
8		4.000	1.000	0.000	BS
9		3.000	1.000	0.000	BS
10		4.000	0.000	-1.000	DN
11		1.000	4.000	0.000	BS
12		6.000	0.000	-5.000	DN

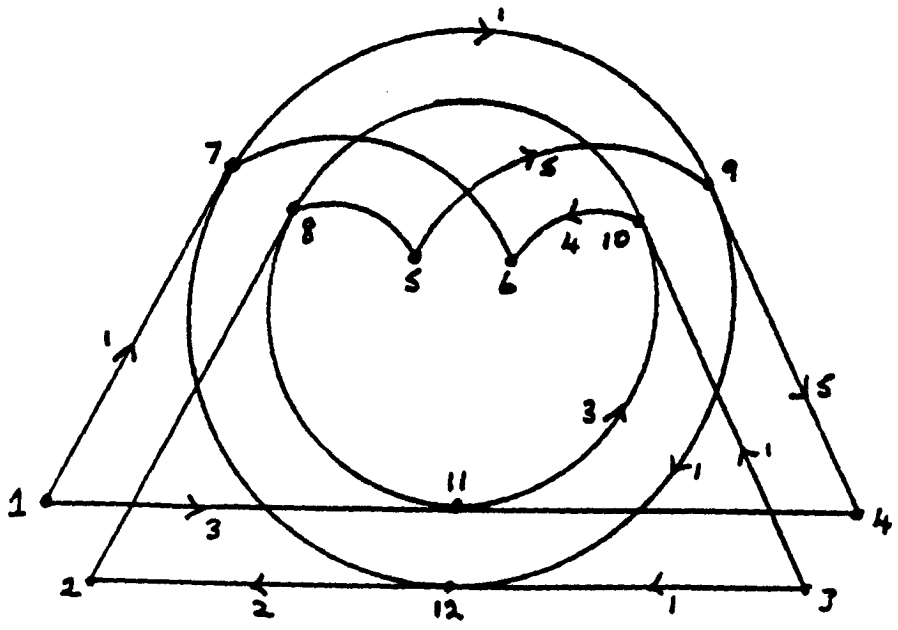


Fig. 4 The better solution

This example demonstrates a case in which the ghost costs, incorporated in the partial derivative which Beale uses to determine the point at which to terminate his program, inhibited the improvement of the solution to a better optimum. In this small example, where it was feasible to relate variables to paths between O-D pairs, the removal of the ghost costs allowed the program to proceed to find the global optimum as demonstrated by the Integer Linear Programming method in the next section.

4.3 FIRST ILP METHOD

The purpose of using ILP is to guarantee a global optimum by the branch and bound method. The MPCODE, developed at the Londion School of Economics, was used.

The variables defined at the end of Section 3.1 are redefined as zero-one variables signifying the non-use or use of each path respectively. The constraints of the QP formulation, are changed by replacing all the right hand sides by 1, to ensure that only one of each pair of paths connecting an O-D pair is used; this is jusified by the fact that the group travel property will hold in the optimal solution. The terms of the quadratic function, which signified the number of times a pair of paths conflicted, therefore acquire increased coefficients in proportion as the two paths involved have to carry more than one unit of flow. For example, the term x(4)*x(12) becomes 12x(4)*x(12) because Path 4, if it is used, will carry 3 units of flow, and Path 12, if it is used, will carry 4 units of flow.

In enable the use of ILP, the quadratic objective function is replaced with a linear function. This involves the definition of new variables, one for each term in the quadratic function, for example,

$$x(1) + x(2) \leq 1 + x(13)$$

$$\text{and} \quad x(1) + x(7) \leq 1 + x(14).$$

For these examples the terms are then replaced with linear terms as follows:

$$x(1)*x(2) \Longrightarrow x(13)$$

$$x(1)*x(7) \Longrightarrow x(14).$$

The data file for this formulation is shown below. In this case, Record 2 consists of M, N, and NUMD, with NUMD being -1 to show that all the variables take discrete values. Record 3 is like Record 4 for QP and consists of 'MIN', M, N, and ISBND with ISBND being -1 to show that all the variables are zero or one. Record 4, like Record 5 in QP, shows the elements of the linear objective function. Records 5 and 6 and following are like Record 6 and 7 and following for QP.

```
'NONE',/
46,52,-1,/
'MIN',46,52,-1,/
13,1, 14,1, 15,1, 16,4, 17,4,
18,2, 19,1, 20,1, 21,1, 22,1, 23,4, 24,4,
25,9, 26,3, 27,3, 28,3, 29,3, 30,12,
31,6, 32,6, 33,3, 34,3, 35,12,
36,1, 37,1, 38,1,
39,2, 40,3, 41,1, 42,4, 43,8,
44,1, 45,2, 46,1, 47,4,
48,1, 49,4,
50,1, 51,4,
52,16,/
1,'E',1, 2,'E',1, 3,'E',1, 4,'E',1, 5,'E',1, 6,'E',1,
7,'L',1, 8,'L',1, 9,'L',1, 10,'L',1, 11,'L',1, 12,'L',1,
13,'L',1, 14,'L',1, 15,'L',1, 16,'L',1, 17,'L',1,
18,'L',1, 19,'L',1, 20,'L',1, 21,'L',1, 22,'L',1,
23,'L',1, 24,'L',1, 25,'L',1, 26,'L',1, 27,'L',1,
28,'L',1, 29,'L',1, 30,'L',1, 31,'L',1, 32,'L',1,
33,'L',1, 34,'L',1, 35,'L',1, 36,'L',1, 37,'L',1,
38,'L',1, 39,'L',1, 40,'L',1, 41,'L',1, 42,'L',1,
43,'L',1, 44,'L',1, 45,'L',1, 46,'L',1,/
1,,1,1, 2,1,/
2,,3,1, 4,1,/
3,,5,1, 6,1,/
4,,7,1, 8,1,/
5,,9,1, 10,1,/
6,,11,1, 12,1,/
7,,1,1, 2,1, 13,-1,/
8,,1,1, 7,1, 14,-1,/
9,,1,1, 9,1, 15,-1,/
10,,1,1, 11,1, 16,-1,/
11,,1,1, 12,1, 17,-1,/
12,,2,1, 6,1, 18,-1,/
13,,2,1, 7,1, 19,-1,/
14,,2,1, 8,1, 20,-1,/
```

continued overleaf

```
15,,2,1, 9,1, 21,-1,/
16,,2,1, 10,1, 22,-1,/
17,,2,1, 11,1, 23,-1,/
18,,2,1, 12,1, 24,-1,/
19,,3,1, 4,1, 25,-1,/
20,,3,1, 6,1, 26,-1,/
21,,3,1, 7,1, 27,-1,/
22,,3,1, 8,1, 28,-1,/
23,,3,1, 9,1, 29,-1,/
24,,3,1, 11,1, 30,-1,/
25,,4,1, 6,1, 31,-1,/
26,,4,1, 7,1, 32,-1,/
27,,4,1, 8,1, 33,-1,/
28,,4,1, 10,1, 34,-1,/
29,,4,1, 12,1, 35,-1,/
30,,5,1, 6,1, 36,-1,/
31,,5,1, 9,1, 37,-1,/
32,,5,1, 10,1, 38,-1,/
33,,6,1, 7,1, 39,-1,/
34,,6,1, 9,1, 40,-1,/
35,,6,1, 10,1, 41,-1,/
36,,6,1, 11,1, 42,-1,/
37,,6,1, 12,1, 43,-1,/
38,,7,1, 8,1, 44,-1,/
39,,7,1, 9,1, 45,-1,/
40,,7,1, 10,1, 46,-1,/
41,,7,1, 11,1, 47,-1,/
42,,8,1, 9,1, 48,-1,/
43,,8,1, 12,1, 49,-1,/
44,,9,1, 10,1, 50,-1,/
45,,9,1, 12,1, 51,-1,/
46,,11,1, 12,1, 52,-1,/
```

The output file is too extensive to reproduce here. The solution is the same as that obtained by QP when the ghost costs were removed.

4.4 SECOND ILP METHOD

For the second ILP method, a matrix of costs is compiled in terms of conflicts between all possible pairs of paths, suitably weighted for those paths which carry more than one unit of flow. The matrix, partitioned as described in Chapter 3, Subsection 3.5.2, is shown below.

Path	1	2	3	4	5	6	7	8	9	10	11	12
1	-	1	-	-	-	-	1	-	1	-	4	4
2	1	-	-	-	-	1	1	1	1	1	4	4
3	-	-	-	9	-	3	3	3	3	-	12	-
4	-	-	9	-	-	6	6	3	-	3	-	12
5	-	-	-	-	-	1	-	-	1	1	-	-
6	-	1	3	6	1	-	2	-	2	1	4	4
7	1	1	3	6	-	2	-	1	2	1	4	-
8	-	1	3	3	-	-	1	-	1	-	-	4
9	1	1	3	-	1	2	2	1	-	1	-	4
10	-	1	-	3	1	1	1	-	1	-	-	-
11	4	4	12	-	-	4	4	-	-	-	-	16
12	4	4	-	12	-	4	-	4	4	-	16	-

This matrix is reduced as described in Case 1 in Chapter 3, Subsection 3.5.2; for each part whose elements are all greater than some number k, k is subtracted from each of those elements. The sum of all the k's for each part reduced in this way is twice the number of essential conflicts in the problem; which ever pairs of paths are chosen these conflicts are unavoidable. The number of essential conflicts is 8. To highlight places where these reductions were made zeros are entered when the reduction resulted in a zero. The reduced matrix is shown below.

Path	1	2	3	4	5	6	7	8	9	10	11	12
1	-	1	-	-	-	-	1	-	1	-	0	0
2	1	-	-	-	-	1	1	1	1	1	0	0
3	-	-	-	9	-	3	0	0	3	-	12	-
4	-	-	9	-	-	6	3	0	-	3	-	12
5	-	-	-	-	-	1	-	-	0	0	-	-
6	-	1	3	6	1	-	2	-	1	0	4	4
7	1	1	0	3	-	2	-	1	2	1	4	-
8	-	1	0	0	-	-	1	-	1	-	-	4
9	1	1	3	-	0	1	2	1	-	1	-	-
10	-	1	-	3	0	0	1	-	1	-	-	-
11	0	0	12	-	-	4	4	-	-	-	-	16
12	0	0	-	12	-	4	-	4	-	-	16	-

Next each row in each part is examined, and a similar reduction to that described for Case 2 in Chapter 3, Subsection 3.5.2, is carried out. Route conflicts obtained are:

2 for Path 2,

7 for Path 6,

2 for Path 7,

and 2 for Path 9.

The reduced matrix is shown below.

Path	1	2	3	4	5	6	7	8	9	10	11	12
1	-	1	-	-	-	-	1	-	1	-	0	0
2	1	-	-	-	-	1	0	0	0	0	0	0
3	-	-	-	9	-	3	0	0	3	-	12	-
4	-	-	9	-	-	6	3	0	-	3	-	12
5	-	-	-	-	-	1	-	-	0	0	-	-
6	-	1	0	3	1	-	2	-	1	0	0	0
7	0	0	0	3	-	2	-	1	1	0	4	-
8	-	-	0	0	-	-	1	-	1	-	-	4
9	0	0	3	-	0	1	1	0	-	1	-	-
10	-	1	-	3	0	0	1	-	1	-	-	-
11	0	0	12	-	-	4	4	-	-	-	-	16
12	0	0	-	12	-	4	-	4	-	-	16	-

The pair conflicts are computed by adding the elements (i,j) and (j,i) to find the pair conflict for the pair of paths i and j. The 26 non-zero pair conflicts are listed below.

P(1,2) = 2, P(1,7) = 1, P(1,9) = 1,

P(2,6) = 2, P(2,10) = 1,

P(3,4) = 18, P(3,6) = 3, P(3,9) = 6, P(3,11) = 24,

P(4,6) = 9, P(4,7) = 6, P(4,10) = 6, P(4,12) = 24,

P(5,6) = 2,

P(6,7) = 4, P(6,9) = 2, P(6,11) = 4, P(6,12) = 4,

P(7,8) = 2, P(7,9) = 2, P(7,10) = 1, P(7,11) = 8,

P(8,9) = 1, P(8,12) = 8,

P(9,10) = 2,

P(11,12) = 32.

The method involves solving a sequence of ILP problems.

FIRST PROBLEM

Twelve (0,1) variables are defined to correspond to non-use and use of the 12 paths. Six constraints specify that exactly one of the pair of paths between each O-D pair is used. The objective function to be minimised involves only the route costs for each path.

The data file is shown below.

```
'NONE',/
6,12,-1,/
'MIN',6,12,-1,/
2,2, 6,7, 7,2, 9,2,/
1,'E',1, 2,'E',1, 3,'E',1, 4,'E',1, 5,'E',1, 6,'E',1,/
1,,1,1, 2,1,/
2,,3,1, 4,1,/
3,,5,1, 6,1,/
4,,7,1, 8,1,/
5,,9,1, 10,1,/
6,,11,1, 12,1,/
```

It will be observed that only Paths 2, 6, 7, and 9 have non-zero route costs. The solution to this problem is to use:

Paths 1, 3, 5, 8, 10, and 11.

The value of the minimand is zero and the LP solution did, in fact, satisfy the discrete constraints.

SECOND PROBLEM

Of the path pairs used in the first solution only the pair

(3,11) has a positive cost, 24, so a variable x(13) defined by:

x(3) + x(11) LE 1 + x(13),

is introduced and a term 24*x(13) is added to the objective function.
The solution to this problem is to use:

Paths 1, 4, 5, 8, 10, and 11.

Path 4 is used instead of Path 3 to avoid the pair conflict whose cost has been introduced. The value of the minimand is zero and the LP solution did, again, satisfy the discrete constraints.

THIRD PROBLEM
Of the path pairs used in the second solution only the new pair (4,10) has a positive cost, 6, so a variable x(14) defined by:

x(4) + x(10) LE 1 + x(14),

is introduced and a term 6*x(14) is added to the objective function.

The solution to this problem is to use:

Paths 1, 3, 5, 8, 10, and 12.

The solution has gone back to using Path 3, but uses Path 12 instead of Path 11 to avoid the pair costs of both (3,11) and (4,10). The value of the minimand is zero and the LP solution did, again, satisfy the discrete constraints.

FOURTH PROBLEM

Of the path pairs used in the third solution only the new pair (8,12) has a positive cost, 8, so a variable x(15) defined by:

x(8) + x(12) LE 1 + x(15),

is introduced and a term 8*x(15) is added to the objective function.

The solution to this problem is to use:

Paths 1, 4, 5, 8, 9, and 11.

The solution has gone back to using Path 4, but uses Path 9 instead of Path 10 to avoid the pair cost of (4,10). It also uses Path 11, now that Path 3 is not being used, instead of Path 12 to avoid the pair cost of (8,12). The value of the minimand is 2, the route cost of Path 9, and the LP solution did, once again, satisfy the discrete constraints.

FIFTH PROBLEM

Of the path pairs used in the fourth solution only the new

pair (8,9) has a positive cost, 1, so a variable x(16) defined by:

x(8) + x(9) LE 1 + x(16),

is introduced and a term 1*x(16) is added to the objective function.

The solution to this problem is to use:

Paths 1, 3, 5, 7, 10, and 12.

The solution has gone back to using Path 3, but uses Path 12 instead of Path 11 to avoid the pair cost of (3,11). It also uses Path 7 instead of Path 8 to avoid the pair cost of (8,12), which exceeds the route of Path 7. It uses Path 10, now that Path 4 is not being used, instead of Path 9 to avoid the route cost of Path 9. The value of the minimand is 2, the route cost of Path 7. This time the LP solution did not satisfy the discrete constraints.

SIXTH PROBLEM

Of the path pairs used in the fifth solution only the new pairs (1,7) and (7,10) have positive costs, both being 1, so two new variables x(17) and x(18) defined by:

x(1) + x(7) LE 1 + x(17),

and x(7) + x(10) LE 1 + x(18),

are introduecd and two terms 1*x(17) and 1*x(18) are added to the objective function.

The solution to this problem is to use:

Paths 1, 4, 5, 8, 9, and 11.

The solution has gone back to using Path 4, but uses Path 9 instead of Path 10 to avoid the pair cost of (4,10), which exceeds the route cost of Path 9. It also uses Path 8 instead of Path 7 to avoid the route cost of Path 7 and the pair cost of (1,7) which together exceed the pair cost of (8,9). It uses Path 11 instead of path 12 to avoid the pair cost of (8,12). The value of the minimand is 3, being the sum of the route cost of Path 9 and the pair cost of (8,9). The LP solution does not satisfy the discrete constraints.

SEVENTH PROBLEM

Of the path pairs used in the sixth solution the new pair (1,9) has a positive cost of 1, so a new variable x(19) defined by:

x(1) + x(9) LE 1 + x(19),

is introduced and a term 1*x(19) added to the objective function.

The solution to this problem is to use:

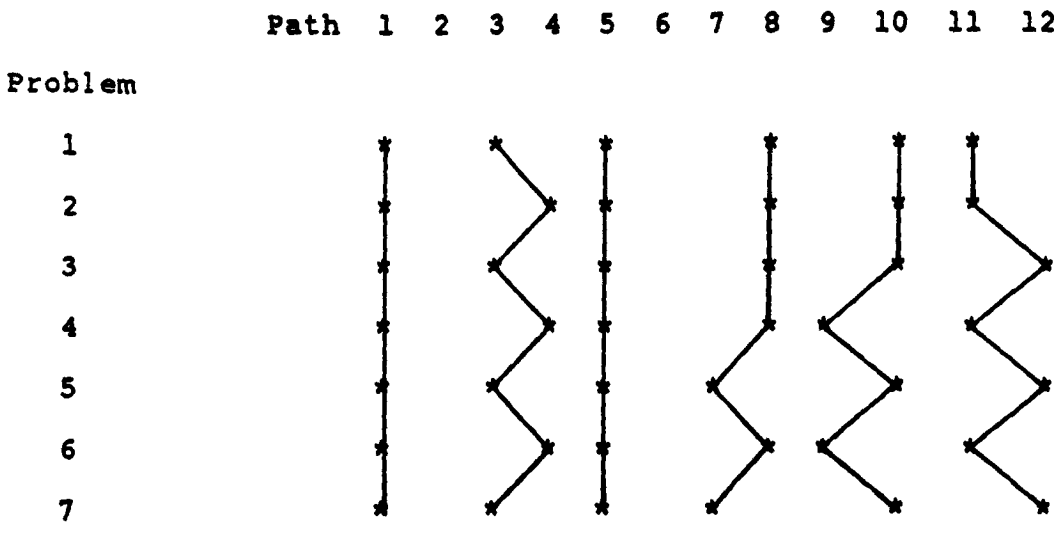
Paths 1, 3, 5, 7, 10, and 12.

The solution has gone back to using Path 10 to avoid both the pair cost of (1,9) and the route cost of Path 9, but uses Path 3 instead of Path 4 to avoid the pair cost of (4,10). It therefore uses Path 12 instead of Path 11 to avoid the pair cost of (3,11) at 24, and it uses Path 7 to avoid the pair cost of (8,12) at 8; these avoidance measures incur the lower total costs of the route cost of Path 7, and the pair costs of (1,7) and (7,10). The value of the minimand is 4, being the sum of the route cost of Path 7 and the pair costs of (1,7) and (7,10). The LP solution does not satisfy the discrete constraints.

For this solution all the costs of used pairs of paths feature in the objective function. Although there may be other solutions with the same value of the objective function, there will not be one with a lower value of the objective function for the following reason. If we assume that this is not the case we arrive at a contradiction as follows. Any change of path used to give a better solution will either involve a zero or a non-zero pair cost. If it involves a zero pair cost it would be the solution to this last problem - a contradiction. If it involves a positive pair cost, either that cost already features in the objective function or it does not. If it does, then it is equally as good as the current solution -

a contradiction. If it does not, then the cost of the current solution would be higher than the cost of the alternative feasible solution neglecting this pair cost. However, the current problem is the first to have a solution with all the pair costs already featuring in the objective function so no chaeper alternative solution can exist. We therefore conclude that we have an optimal solution. It is different to the one found by the first ILP method. The objective functions used in this series of problems solved by the second method have taken no account of the 8 essential conflicts but have counted the remaining conflicts twice. To obtain the true value, we divide 4 by 2 and add on 8 to obtain the value of 10 obtained by the first method.

The way these seven solutions have involved switches between pairs of paths is shown in a diagram in Figure 5.



* used path.

Fig. 5 Successive solutions to the seven problems

APPENDIX 5

THE VINE BUILDING PROCESS

A very simple network is used to demonstrate the tree building process. It consists of an origin vertex number 1, a destination vertex number 2, 5 arcs as shown in Figure 1, together with the artificial arc added to connect Vertex 2 to Vertex 1. Arc numbers, lower bounds and costs are as shown. The problem is to find a minimum cost route for 1 trip from Origin 1 to Destination 2. The flow on each arc is set to zero and the dual value at each vertex is set to zero at the start-up. The steps are somewhat abbreviated.

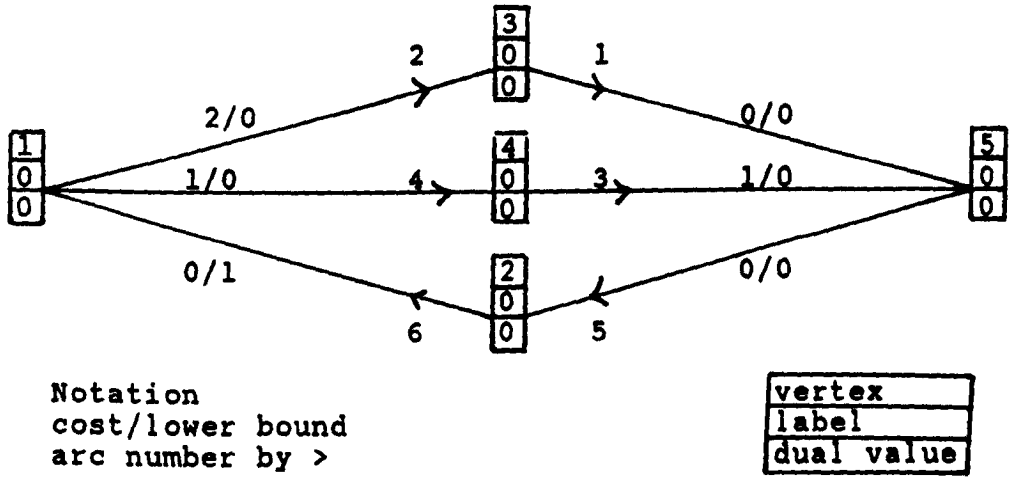


Figure 1

Step 1

Look for Out-of-Kilter arc.

Find Arc number 6

Set SRC = 1, the number of the vertex at the end of Arc 6.

Set SNK = 2, the number of the vertex at the start of Arc 6.

Label Vertex 1 with 6. See Figure 2.

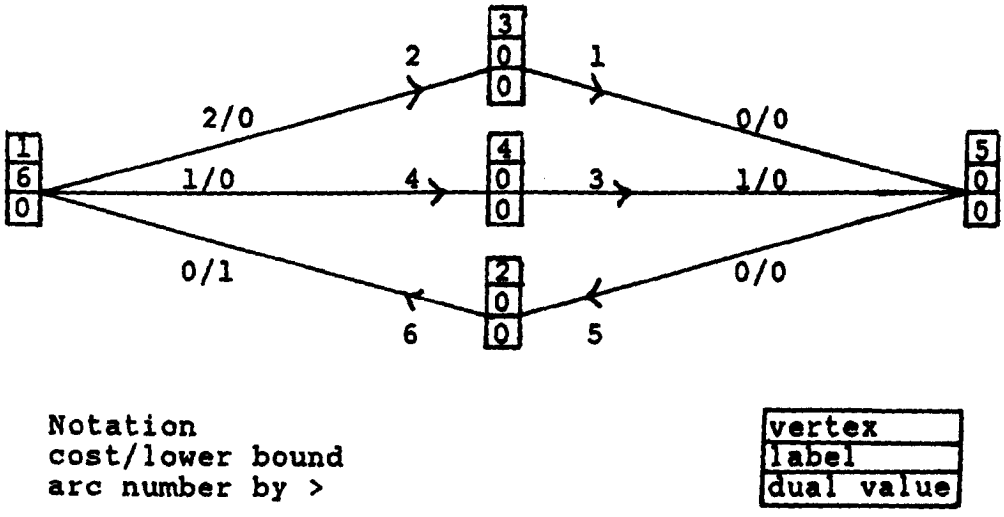


Figure 2

Step 2

Look for arcs with 1) start vertex with non-zero label,

find Arcs 2 and 4;

and 2) end vertex with zero label,

Arcs 2 or 4 have this property;

and 3) net cost negative or zero

BUT net cost(2) = 2

and net cost(4) = 1

so neither of these arcs can be added to the vine.

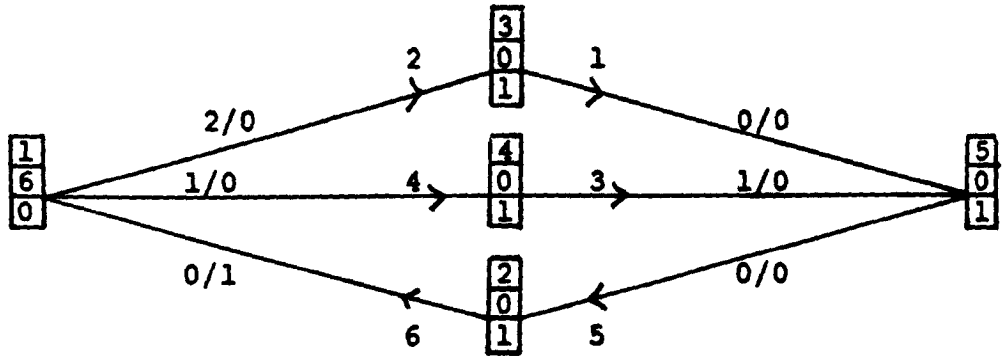
Step 3

Look for smallest differential in dual values which will bring the net cost of one of these arcs to zero.

Look at Arc 2 and set DEL = 2.

Look at Arc 4 and reset DEL = 1

Increase the dual value, of all vertices with zero label by DEL. See Figure 3.



Notation
cost/lower bound
arc number by >

vertex
label
dual value

Figure 3

Step 4

Go back to Step 2

Look for arcs with 1) start vertex with non-zero label,

find Arcs 2 and 4;

and 2) end vertex with zero label,

Arcs 2 or 4 have this property;

and 3) net cost negative or zero

net cost(2) = 1

and net cost(4) = 0.

Arc 4 can be added to the vine and Vertex 4 labelled with 4.

Check whether the label of Vertex 2 (because SNK = 2) is non-zero.

It is not non-zero so LAB is set equal to 1 to indicate that some labelling has happened, and that it is worthwhile to repeat step 2. See Figure 4.

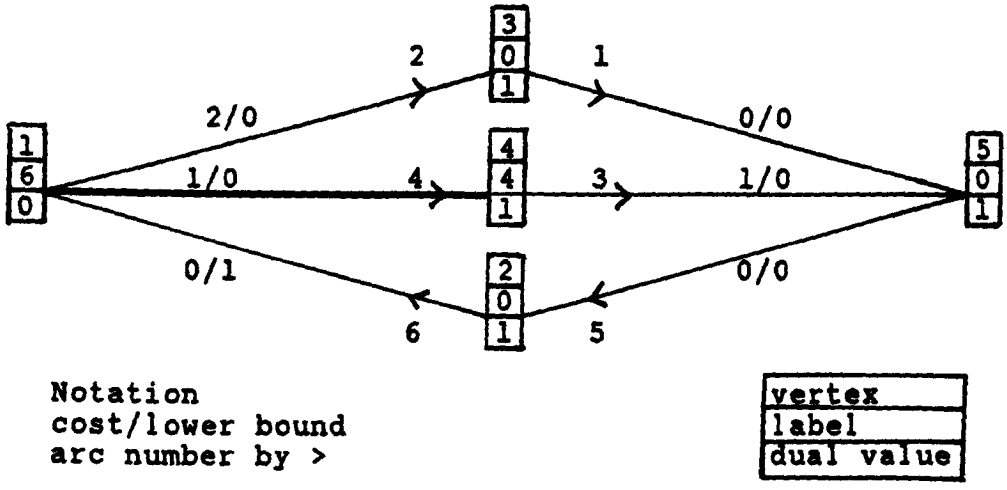


Figure 4

Return to Step 2.

Look for arcs with 1) start vertex with non-zero label,

find Arcs 2, 3 and 4;

and 2) end vertex with zero label,

Arcs 2 and 3 have this property;

and 3) net cost negative or zero

BUT net cost(2) = 1

and net cost(3) = 1.

so neither of these arcs can be added to the vine.

Return to Step 3.

Look for smallest differential in dual values which will bring the net cost of one of these arcs to zero.

Look at Arc 2 and set DEL = 1.

Look at Arc 3 and keep DEL = 1.

Increase the dual value, of all vertices with zero label by DEL. See Figure 5.

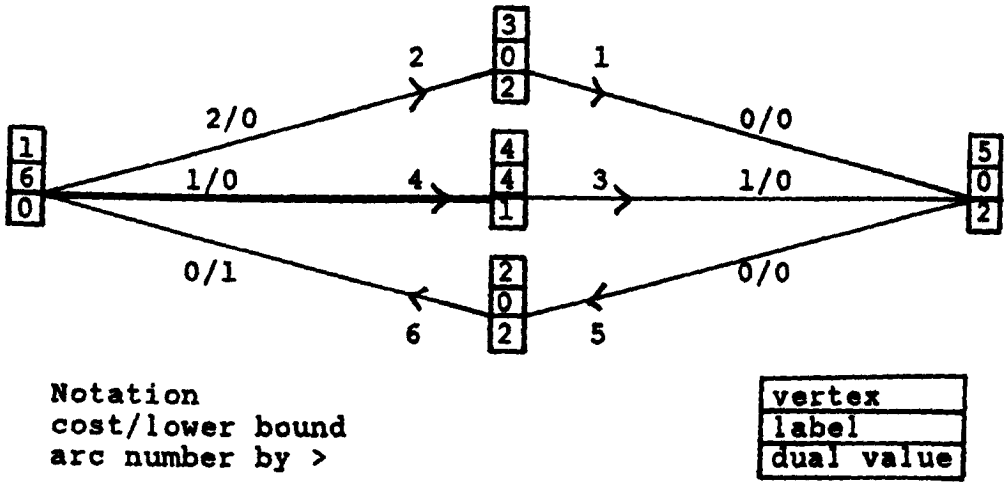


Figure 5

Go back to Step 2.

Look for arcs with 1) start vertex with non-zero label,

find Arcs 2, 3 and 4;

and 2) end vertex with zero label,

Arcs 2 and 3 have this property;

and 3) net cost negative or zero

net cost(2) = 0.

Add Arc 2 to the vine and label Vertex 3 with 2.

Check whether the label of Vertex 2 (because SNK = 2) is non-zero.

It is not non-zero so LAB is set equal to 1 to indicate that some labelling has happened, and that it is worthwhile to continue with Step 2. See Figure 6.

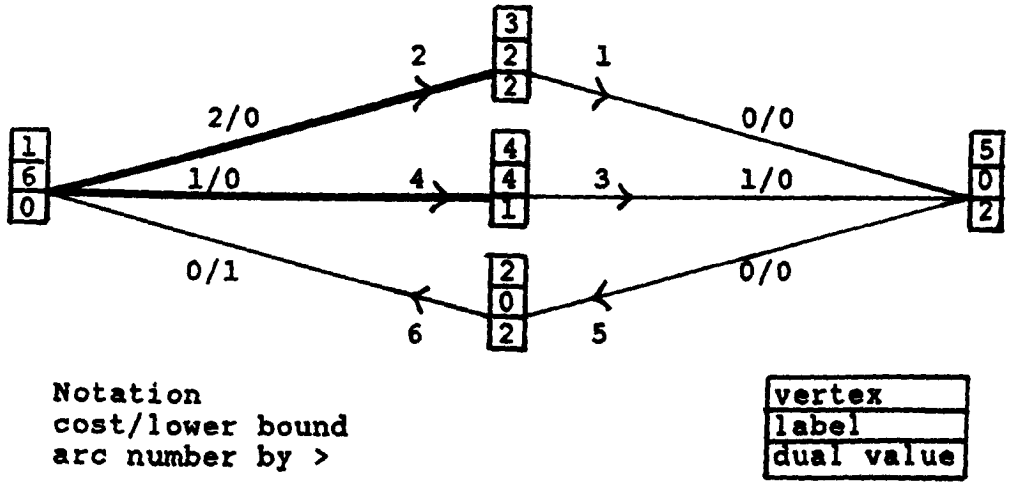


Figure 6

Continue with Step 2

Look for arcs with 1) start vertex with non-zero label,

find Arc 3;

and 2) end vertex with zero label,

Arc 3 has this property;

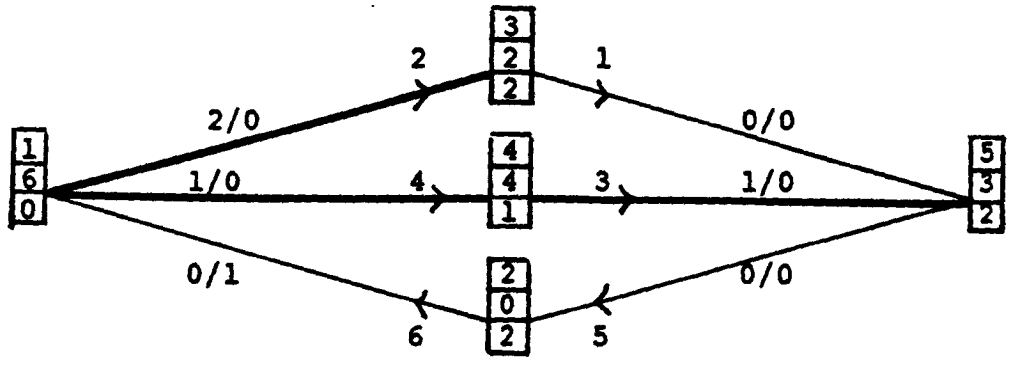
and 3) net cost negative or zero

net cost(3) = 0.

Arc 3 can be added to the vine and Vertex 5 labelled with 3.

Check whether the label of Vertex 2 (because SNK = 2) is non-zero.

It is not non-zero, so LAB is set equal to 1 to indicate that some labelling has happened, and that it is worthwhile to continue with Step 2. See Figure 7.



Notation
cost/lower bound
arc number by >

vertex
label
dual value

Figure 7

Continue with Step 2

Look for arcs with 1) start vertex with non-zero label,

find Arcs 4 and 5;

and 2) end vertex with zero label,

Arc 5 has this property;

and 3) net cost negative or zero

net cost(5) = 0.

Arc 5 can be added to the vine and Vertex 2 labelled with 5.

Check whether the label of Vertex 2 (because SNK = 2) is non-zero.

It is non-zero, so flow is augmented on the flow augmenting circuit from vertex 1 to 4 to 5 to 2 to 1, by 1 unit, and a return made to Step 1.

Step 1.

Look for Out-of-Kilter arcs.

Find none. Stop.

APPENDIX 6

ENSURING THE GROUP TRAVEL PROPERTY

The small example in Appendix 5 is extended with the addition of a second destination at Vertex 6, as shown below, and demand for an extra trip from the origin, Vertex 1, to Vertex 6. After having augmented the flow to cater for 1 trip from 1 to 2 the problem is to find a minimum cost route for one trip from 1 to 6. Labels would be reset at zero but dual values would be retained. The start-up position is as in Figure 1.

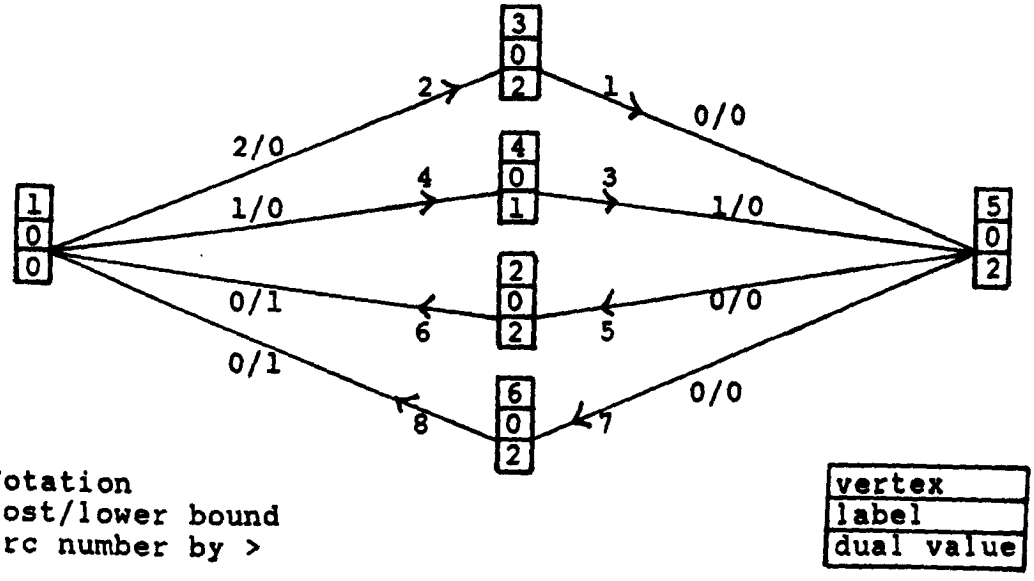


Figure 1

Step 1

Look for Out-of-Kilter arc.

Find Arc number 8.

Set SRC = 1, the number of the end vertex for Arc 8.

Set SNK = 6, the number of the start vertex for Arc 8.

Label vertex 1 with 8.

See Figure 2.

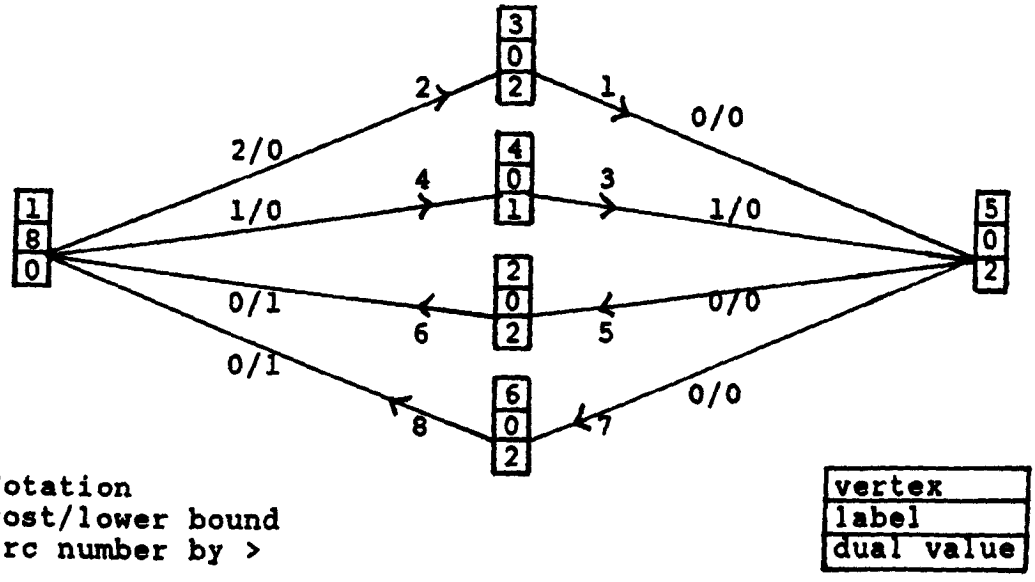


Figure 2

Step 2

Look for arcs with 1) start vertex with non-zero label,

find Arcs 2 and 4;

and 2) end vertex with zero label,

Arcs 2 and 4 have this property;

and 3) net cost negative or zero

net cost(2) = 0

Arc 2 can be added to the vine and Vertex 3 labelled with 2.

Check whether the label of Vertex 6 (because SNK = 6) is non-zero.

It is not non-zero, so LAB is set equal to 1 to indicate that some labelling has happened, and that it may be worthwhile repeating Step 2.

See Figure 3 and continue with Step 2.

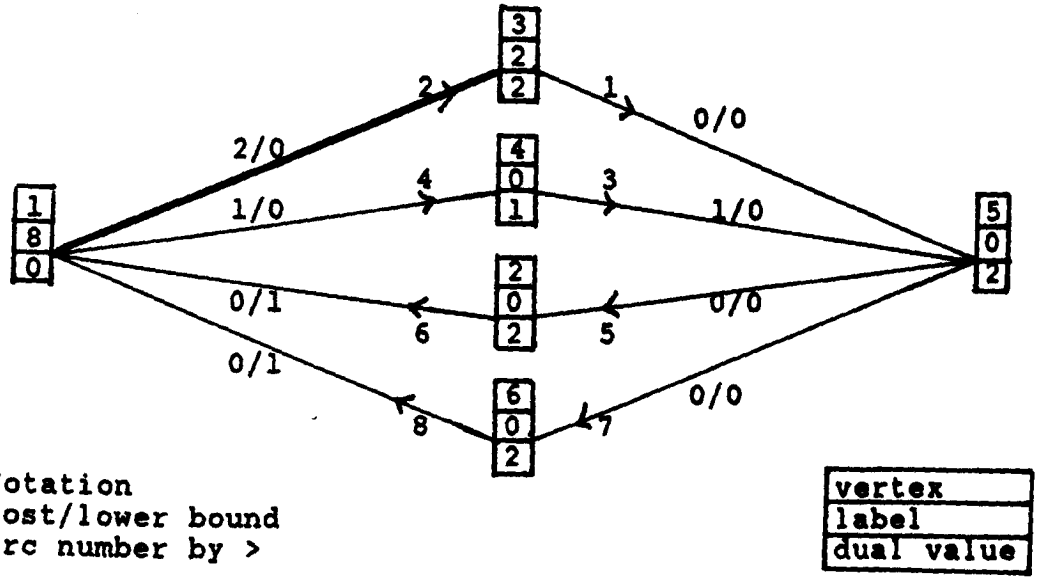


Figure 3

Continue with Step 2

Look for arcs with 1) start vertex with non-zero label,

find Arc 4;

and 2) end vertex with zero label,

Arc 4 has this property;

and 3) net cost negative or zero,

net cost(4) = 0

Arc 4 can be added to the vine and Vertex 4 labelled with 4.

Check whether the label of Vertex 6 (because SNK = 6) is non-zero.

It is not non-zero, so continue with Step 2.

See Figure 4.

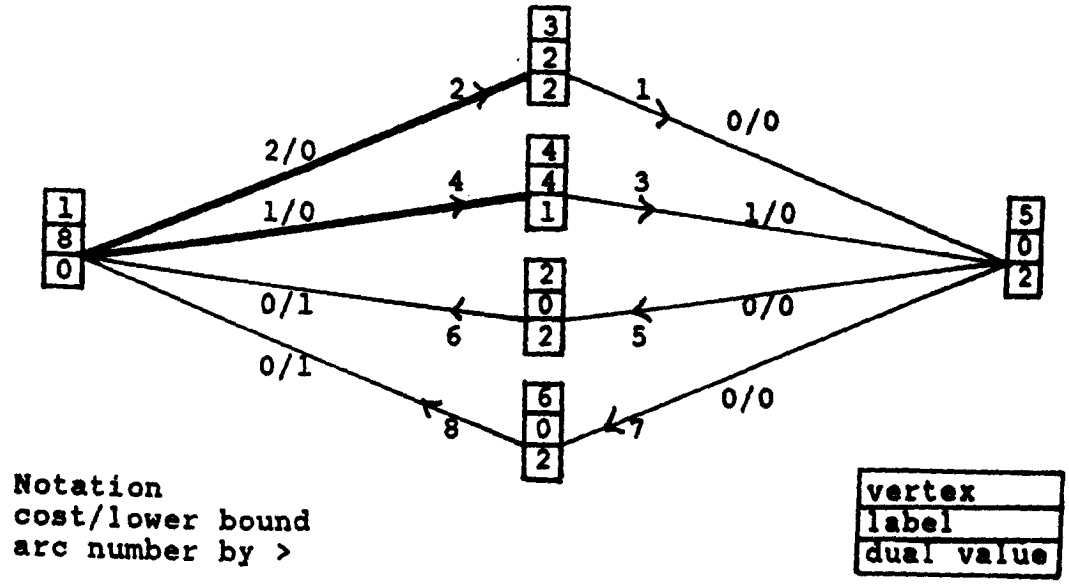


Figure 4

Continue with Step 2

Look for arcs with 1) start vertex with non-zero label,

find no more arcs.

But some labelling has happened, so it is worthwhile to repeat Step 2 starting with Arc 1.

Look for arcs with 1) start vertex with non-zero label,

find Arcs 1, 2, 3, 4;

and 2) end vertex with zero label,

Arcs 1 and 3 have this property;

and 3) net cost negative or zero,

net cost(1) = 0.

Arc 1 can be added to the vine and Vertex 5 labelled with 1.

Check whether the label of Vertex 6 (because SNK = 6) is non-zero.

It is not non-zero, so continue with Step 2.

See Figure 5.

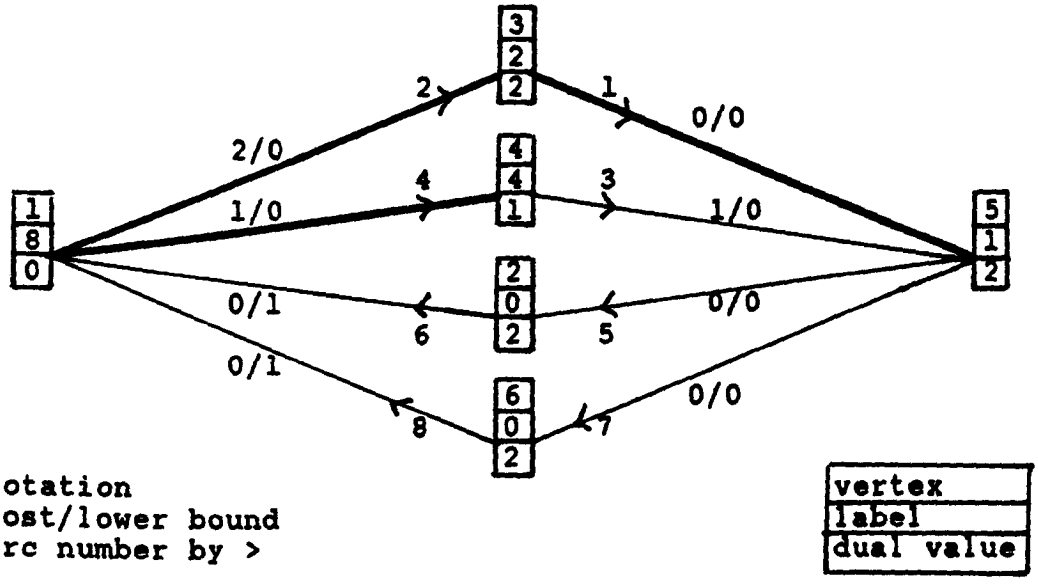


Figure 5

Continue with Step 2

Look for arcs with 1) start vertex with non-zero label,

find Arcs 2, 3, 4, 5 and 7.

and 2) end vertex with zero label,

Arcs 5 and 7 have this property;

and 3) net cost negative or zero,

net cost(5) = 0.

Arc 5 can be added to the vine and Vertex 2 labelled with 5.

Check whether the label of Vertex 6 (because SNK = 6) is non-zero.

It is not non-zero, so continue with Step 2.

See Figure 6.

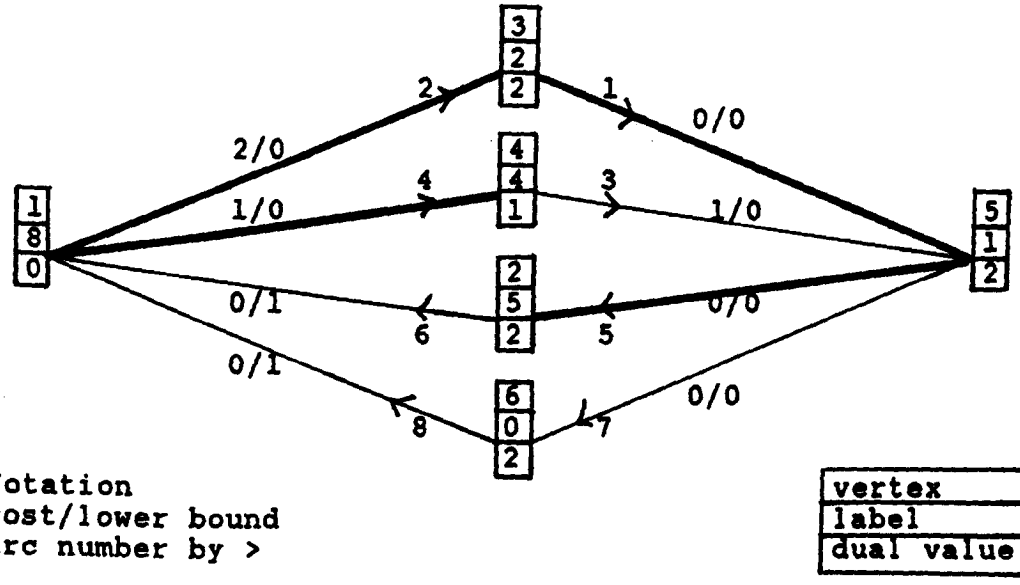


Figure 6

Continue with Step 2

Look for arcs with 1) start vertex with non-zero label,

find Arc 6 and 7;

and 2) end vertex with zero label,

Arc 7 has this property;

and 3) net cost negative or zero,

net cost(7) = 0.

Arc 7 can be added to the vine and Vertex 6 labelled with 7.

Check whether the label of Vertex 6 (because SNK = 6) is non-zero.

It is, so proceed to Step 5.

Step 5

Augment flow on flow augmenting circuit, Vertex 1 to 3 to 5 to 6 by 1 unit.

Return to Step 1

Step 1

Look for Out-of-Kilter arcs. Find none. Stop.

The result of this assignment is that the trip from 1 to 2 is assigned to the path through vertices 1, 4, 5 and 2 and the trip from 1 to 6 is assigned to the path through vertices 1, 3, 5 and 6. These paths diverge at 1 and merge again at 5. This is violating the group travel property. The routes would have had the same costs if both trips had been routed via 4 and they would have avoided the merge with each other.

To avoid the occurrence of this situation, the following change is made to the algorithm.

The labels as well as the dual values are retained between finding flow augmenting paths.

With each new Out-of-Kilter arc which is discovered, a check is made as to whether the vertex whose number matches the new value of SNK is labelled. If it is, then the new destination for which a trip is required, is already in the vine and a flow augmenting path can be found straight away. If it is not, then one starts adding branches to the tips of the vine.

This has the effect of ensuring that all the routes from an origin form a vine of paths which do not merge; the group travel property is ensured. It also has the effect of saving the computing time that would be involved in building a new vine from scratch for each destination.

APPENDIX 7

THE SOURCE CODE FOR THE PROGRAMS

This appendix contains the source code for the programs, written in the FORTRAN 77 language. The listings are in the order in which the suite of programs is to be used, namely POLYARCS the program to synthesise the Circulation System, POLYSEND the assignment program which finds the minimum cost routes, and POLYLINK the program which translates the results from POLYSEND into flows on the links of the road network. Subroutines appear in alphabetical order following each main program. These programs were compiled using the RMFORT FORTRAN compiler. They have been run on a Victor 286 micro-computer.

```
      PROGRAM POLYARCS
c     12 June 89
C     This program takes details of nodes and links in an urban road
c     network and creates a pair of vertices (one if the link is one-
c     way) at the midpoint of each link.
C     It creates arcs to join up the vertices according to the allowed
C     traffic movements at the junction represented by the node.
C     If there are inconsistencies in details of the links said to be
c     incident at a junction, e.g. neither the A-node nor the B-node
c     is the junction node, the program stops with a report of the
c     junction node where this happened.
C     The program can model movements at t-junctions, t-junctions onto
c     motorways,
c     crossroads, mini-4-arm roundabouts, conventional 4-arm roundabouts
c     and flyovers over a roundabout or underpasses below a roundabout.
c     It compiles a list of those arcs which conflict with each arc.
c     IT COMPUTES A CONFLICT WEIGHT TO BE APPLIED TO EACH ARC IN THE
C     LIST ABOVE.
c     It creates the files, ARCS.DAT, and CROSSFLO.DAT which can be
c     read by the program, POLYSEND.
c     It creates the file, ARCLINK.DAT, to be read by the program,
c     POLYLINK
C     It can synthesise a traffic circulation network with
c     up to 1200 arcs from a road network with
c     up to 50 zones, 300 junctions and 300 links.
      INTEGER ZONES,XIONS,LINKS,A,B,TW,UB,DB,X,
     1 DA,UA,INJ,XION,I,J,NC,CR,CW,ARCS,FIRSTAB,
     1 FIRSTBA,AP,EX,R,FEED,ZONESIN
C
      COMMON INJ(300,8),XION(300),UB(300),DB(300),I(1200),J(1200),
     1 NC(1200),CR(1200,8),NODES,ARCS,ZONES,A(300),B(300),FIRSTAB(300),
     1 LASTAB(300),FIRSTBA(300),LASTBA(300),KJ(20),AP(4),EX(4),NL(300),
     1 CW(1200,8),JT(300),TW(300),DA(200),UA(200),R(5),
     1 FEED(50),LINKS,XIONS,ZONESIN
C
C     ZONES is the number of nodes which can be origins or
c     destinations of traffic.
C     FEED(M) is the number of connectors from zone M into the
c     network.
c     XIONS is the number of junctions in the urban road network.
c     LINKS is the number of links in the urban road network.
C     A(L) and B(L) are the end nodes of link, L, A being the start
c     node if link, L, is one-way. TW(L) is 0 if link, L, is one-way
c     and 1 if link, L, is two-way.
c     UB(L) is the vertex created on link, L, upstream of node, B.
c     DB(L) is the vertex created on link, L, downstream of node, B.
C     DA(M) is the vertex number of the vertex from which flow
c     from the origin zone, M, will emanate.
c     UA(M) is the vertex number of the vertex at which flow
c     to the destination zone, M, will arrive.
c     NL(J) is the number of links incident to junction, J.
c     INJ(J,NL(J)) is a list for each junction, J, of the link numbers
c     of those links which are incident to junction,J.
c     AP(q) is the vertex number of the vertex upstream of J
c     on the qth link incident to junction, J.
c     EX(q) is the vertex number of the vertex downstream of J
c     on the qth link incident to junction, J.
```

```
C       R(q) the extra vertex at the centre of a roundabout opposite
c       the qth link.
c       ZONESIN is the total number of zone connectors.
c       XION(J) is the number of the Jth junction, allowing for the
c       possibility that the junction numbers need not be consecutive.
c       I(K) and J(K) are the numbers of the start and finish vertices
c       of the directed arc, K.
c       NC(K) is the number of arcs conflicting with arc, K.
c       CR(K,NC(K)) is a list for each arc, K, of the numbers of the
c       NC(K) arcs conflicting with it.
c       JT(X) is the junction type
c       JT(X) = 1 for a free-for-all junction
c       JT(X) = 2 for a priority junction
c       JT(X) = 3 for a signalised junction
c       JT(X) = 4 for a mini-roundabout
c       JT(X) = 5 for a roundabout
c       JT(X) = 6 for a flyover
c       JT(X) = 7 FOR USER TO BE ASKED FOR WEIGHTS
C       CW(K,NC(K)) is the weight for the NC(K)th arc conflicting with
c       arc K.
c
        OPEN(UNIT=6,FILE ='ARCS.DAT',STATUS='NEW')
        OPEN(UNIT=7,FILE ='LINKS.DAT',STATUS='OLD')
        OPEN(UNIT=8,FILE ='CROSSFLO.DAT',STATUS='NEW')
        OPEN(UNIT=9,FILE ='ARCLINK.DAT',STATUS='NEW')
        READ(7,*)ZONES,XIONS,LINKS
        ZONESIN = 0
        DO 10 M=1,ZONES
        READ(7,*)FEED(M)
        ZONESIN = ZONESIN + FEED(M)
   10   CONTINUE
C       We have got past 1st record and next ZONES records
        DO 40 L=1,ZONESIN
          READ(7,*,END=40)A(L),B(L),TW(L)
   40   CONTINUE
C       So L will be number of links so far + 1
C       All link records for zone connectors have been read so there are
c       LINKS-ZONESIN links left.
        DO 50 ML=L,LINKS
          READ(7,*,END=50)A(ML),B(ML),TW(ML)
   50   CONTINUE
c       All link records have now been read
        DO 60 X=1,XIONS
          READ(7,*,END=60)XION(X),JT(X),NL(X),(INJ(X,JL),JL=1,NL(X))
   60   CONTINUE
        CLOSE(UNIT=7,STATUS='KEEP')
C       All road network records have been read.
        CALL ODVERT
        CALL LINKVERT(L)
        CALL MAKEARCS
        CALL PRINT
        WRITE(9,*)ZONES,XIONS,LINKS,ARCS
        DO 70 L=1,LINKS
          WRITE(9,*)L,FIRSTAB(L),LASTAB(L),TW(L),FIRSTBA(L),LASTBA(L)
   70   CONTINUE
        STOP 'Output files are ARCS.DAT, CROSSFLO.DAT and ARCLINK.DAT'
        END
```

```
      BLOCK DATA W3
      INTEGER CON3,CONRF,WT3
      COMMON/C3/CON3(6,3),CONRF(6),WT3(42)
      DATA (CON3(I,1),I=1,2) / 2*6 / (CON3(2,I),I=2,3) / 4,3 /
     1 (CON3(I,1),I=3,4) / 2*2 / (CON3(4,I),I=2,3) / 6,5 /
     1 (CON3(I,1),I=5,6) / 2*4 / (CON3(6,I),I=2,3) / 2,1 /
     1 CONRF(1) / 6 / CONRF(2) / 3 / CONRF(3) / 2 /
     1 CONRF(4) / 5 / CONRF(5) / 4 / CONRF(6) / 1 /
     1 WT3(1) / 1 / (WT3(I),I=2,3) / 2*2 / WT3(4) / 1 /
     1 WT3(5) / 1 / (WT3(I),I=6,7) / 2*2 / WT3(8) / 1 /
     1 WT3(9) / 1 / (WT3(I),I=10,11) / 2*2 / WT3(12) / 1 /
     1 WT3(13) / 1 / (WT3(I),I=14,15) / 2*2 / WT3(16) / 1 /
     1 WT3(17) / 1 / (WT3(I),I=18,19) / 2*2 / WT3(20) / 1 /
     1 WT3(21) / 1 / (WT3(I),I=22,23) / 2*2 / WT3(24) / 1 /
     1 WT3(25) / 1 / (WT3(I),I=26,27) / 2*2 / WT3(28) / 1 /
     1 WT3(29) / 1 / (WT3(I),I=30,31) / 2*2 / WT3(32) / 1 /
     1 WT3(33) / 1 / (WT3(I),I=34,35) / 2*2 / WT3(36) / 1 /
     1 (WT3(I),I=37,42) / 6*1 /
      END
      BLOCK DATA W4
      INTEGER CON4,WT4,CON4MR,CONRO,CONFO
      COMMON/C4/CON4(12,7),WT4(104),CON4MR(16,2),CONRO(20,6),
     1 CONFO(12,6)
      DATA (CON4(I,1),I=1,3) / 3*11 / (CON4(I,1),I=4,6) / 3*2 /
     1 (CON4(I,1),I=7,9) / 3*5 / (CON4(I,1),I=10,12) / 3*8 /
     1 (CON4(I,2),I=1,3) / 3*9 / (CON4(I,2),I=4,6)   / 3*12 /
     1 (CON4(I,2),I=7,9) / 3*3 / (CON4(I,2),I=10,12) / 3*6 /
     1 (CON4(I,3),I=2,3) / 6,12  / (CON4(I,3),I=5,6)   / 9,3 /
     1 (CON4(I,3),I=8,9) / 12,6  / (CON4(I,3),I=11,12) / 3,9 /
     1 (CON4(I,4),I=2,3) / 5,6   / (CON4(I,4),I=5,6)   / 8,9 /
     1 (CON4(I,4),I=8,9) / 11,12 / (CON4(I,4),I=11,12) / 2,3 /
     1 (CON4(I,5),I=2,3) / 12,8  / (CON4(I,5),I=5,6)   / 3,11/
     1 (CON4(I,5),I=8,9) / 6,2   / (CON4(I,5),I=11,12) / 9,5 /
     1 (CON4(I,6),I=2,3) / 4,5   / (CON4(I,6),I=5,6)   / 7,8 /
     1 (CON4(I,6),I=8,9) / 10,11 / (CON4(I,6),I=11,12) / 1,2 /
     1 CON4(3,7) / 7 / CON4(6,7) / 10 / CON4(9,7) / 1 / CON4(12,7)/4/
     1 (WT4(I),I=1,2) / 2*1 / (WT4(I),I=3,6)   / 4*2 /
     1 (WT4(I),I=7,8) / 2*1 /  WT4(9) / 2 /  WT4(10) / 3 /
     1 (WT4(I),I=11,13) / 3*2 / (WT4(I),I=14,15)   / 2*1 /
     1 (WT4(I),I=16,17) / 2*1 / (WT4(I),I=18,21)   / 4*2 /
     1 (WT4(I),I=22,23) / 2*1 /  WT4(24) / 2 /  WT4(25) / 3 /
     1 (WT4(I),I=26,28) / 3*2 / (WT4(I),I=29,30)   / 2*1 /
     1 (WT4(I),I=31,32) / 2*1 / (WT4(I),I=33,36)   / 4*2 /
     1 (WT4(I),I=37,38) / 2*1 /  WT4(39) / 2 /  WT4(40) / 3 /
     1 (WT4(I),I=41,43) / 3*2 / (WT4(I),I=44,45)   / 2*1 /
     1 (WT4(I),I=46,47) / 2*1 / (WT4(I),I=48,51)   / 4*2 /
     1 (WT4(I),I=52,53) / 2*1 /  WT4(54) / 2 /  WT4(55) / 3 /
     1 (WT4(I),I=56,58) / 3*2 / (WT4(I),I=59,60)   / 2*1 /
     1 WT4(61) / 1 / WT4(62) / 2 / WT(63) / 1 /
     1 WT4(64) / 2 / WT4(65) / 1 / WT(66) / 1 /
     1 (WT4(I),I=67,68) / 2*1 /  (WT4(I),I=69,72 / 4*2 /
     1 (WT4(I),I=73,74) / 2*1 / (WT4(I),I=75,76)   / 2*2 /
     1 (WT4(I),I=77,78) / 2*1 / (WT4(I),I=79,80)   / 2*2 /
     1 (WT4(I),I=81,82) / 2*1 /
     1 (WT4(I),I=83,84) / 2*1 / (WT4(I),I=85,86) / 2*2 /
     1 (WT4(I),I=87,88) / 2*1 /  WT4(89) / 3 /
```

```
     1  (WT4(I),I=90,92) / 3*2 / (WT4(I),I=93,94)   / 2*1 /
     1  (WT4(I),I=95,96) / 2*1 / (WT4(I),I=97,98)   / 2*1 /
     1  WT4(99) / 2 /  WT4(100) / 3 /
     1  (WT4(I),I=101,102) / 2*2 / (WT4(I),I=103,104)   / 2*1 /
     1  (CON4MR(I,1),I=1,2) / 2*3 / (CON4MR(I,1),I=3,4) / 2*2 /
     1  (CON4MR(I,1),I=5,6) / 2*7 / (CON4MR(I,1),I=7,8) / 2*6 /
     1  (CON4MR(I,1),I=9,10) / 2*11 / (CON4MR(I,1),I=11,12) / 2*10 /
     1  (CON4MR(I,1),I=13,14) / 2*15 / (CON4MR(I,1),I=15,16) / 2*14 /
     1  (CON4MR(I,2),I=2,3) / 4,1 / (CON4MR(I,2),I=6,7) / 8,5 /
     1  (CON4MR(I,2),I=10,11) / 12,9 / (CON4MR(I,2),I=14,15) / 16,13 /
     1  (CONRO(I,1),I=1,3) / 3*17 / (CONRO(I,2),I=1,3) / 3*15 /
     1  (CONRO(I,1),I=6,8) / 3*2 / (CONRO(I,2),I=6,8) / 3*20 /
     1  (CONRO(I,1),I=11,13) / 3*7 / (CONRO(I,2),I=11,13) / 3*5 /
     1  (CONRO(I,1),I=16,18) / 3*12 / (CONRO(I,2),I=16,18) / 3*10 /
     1  (CONRO(I,1),I=4,5) / 8,13 / (CONRO(I,1),I=9,10) / 13,18 /
     1  (CONRO(I,1),I=14,15) /18,3 / (CONRO(I,1),I=19,20) / 3,8 /
     1  (CONRO(2,I),I=3,6) / 8,7,6,20 / (CONRO(7,I),I=3,6) / 13,12,11,5/
     1  (CONRO(12,I),I=3,6) /18,17,16,10/(CONRO(17,I),I=3,6) / 3,2,1,15/
     1  CONRO(3,3) / 19 / CONRO(8,3) / 4 /
     1  CONRO(13,3) / 9 / CONRO(18,3) / 14 /
     1  (CONRO(5,I),I=2,4) / 12,7,11 / (CONRO(10,I),I=2,4) / 17,12,16 /
     1  (CONRO(15,I),I=2,4) / 2,17,1 / (CONRO(20,I),I=2,4) / 7,2,6 /
     1  (CONFO(1,I),I=1,2) / 11,9 / (CONFO(4,I),I=1,2) / 2,12 /
     1  (CONFO(7,I),I=1,2) / 5,3 / (CONFO(10,I),I=1,2) / 8,6 /
     1  (CONFO(2,I),I=1,4) / 9,6,4,12 / (CONFO(5,I),I=1,2) / 7,3 /
     1  (CONFO(8,I),I=1,4) / 3,12,10,6 / (CONFO(11,I),I=1,2) / 1,9 /
     1  (CONFO(3,I),I=1,6) / 9,12,6,8,5,7 /
     1  (CONFO(6,I),I=1,6) / 2,12,3,9,8,10 /
     1  (CONFO(9,I),I=1,6) / 3,6,12,2,11,1 /
     1  (CONFO(12,I),I=1,6) / 8,6,9,3,2,4 /
        END
C
        SUBROUTINE ARC3LINK(X,L)
C       For each link, L, feeding into a T-junction, X, this subroutine
c       determines whether the B node or the A node is at the junction
c       and computes FIRSTAB(L) and LASTAB(L) or FIRSTBA(L) and LASTBA(L)
C       respectively. This is to enable the total flow on link, L, to be
c       computed separately for the direction A to B and B to A from the
c       array TOTEFLOW(K) as output to the file, FARCFLO.DAT, by the program
c       POLYSEND. If those links from zones have their A to B direction
c       coded away from the zone no arcs will be created from the
c       destination vertex so FIRSTBA andd LASTBA will be 0 for links from
c       from zones and no means will be provided for computing the total
c       flow along a link into a destination. It can be assumed to equal
c       the demand.
c
        INTEGER AP,EX,R,FEED,X,A,B,INJ,XION,UB,DB,I,J,NC,CR,ARCS,ZONES,
     1  FIRSTAB,FIRSTBA,TW,DA,UA,LINKS,XIONS,ZONESIN
        COMMON INJ(300,8),XION(300),UB(300),DB(300),I(1200),J(1200),
     1  NC(1200),CR(1200,8),NODES,ARCS,ZONES,A(300),B(300),FIRSTAB(300),
     1  LASTAB(300),FIRSTBA(300),LASTBA(300),KJ(20),AP(4),EX(4),NL(300),
     1  CW(1200,8),JT(300),TW(300),DA(200),UA(200),R(5),
     1  FEED(50),LINKS,XIONS,ZONESIN
          IF(AP(L).EQ.UB(INJ(X,L)))THEN
            IF(KJ(2*L-1).NE.0)THEN
              FIRSTAB(INJ(X,L))=KJ(2*L-1)
```

```
              IF(KJ(2*L).NE.0)THEN
                LASTAB(INJ(X,L))=KJ(2*L)
              ELSE
                LASTAB(INJ(X,L))=KJ(2*L-1)
              END IF
            ELSE
              IF(KJ(2*L).NE.0)THEN
                FIRSTAB(INJ(X,L))=KJ(2*L)
                LASTAB(INJ(X,L))=KJ(2*L)
              END IF
            END IF
          ELSE
            IF(KJ(2*L-1).NE.0)THEN
              FIRSTBA(INJ(X,L))=KJ(2*L-1)
              IF(KJ(2*L).NE.0)THEN
                LASTBA(INJ(X,L))=KJ(2*L)
              ELSE
                LASTBA(INJ(X,L))=KJ(2*L-1)
              END IF
            ELSE
              IF(KJ(2*L).NE.0)THEN
                FIRSTBA(INJ(X,L))=KJ(2*L)
                LASTBA(INJ(X,L))=KJ(2*L)
              END IF
            END IF
          END IF
        RETURN
        END
C       Of subroutine ARC3LINK(X,L)
C
        SUBROUTINE ALMINI4(X,L)
c       Relates arc numbers to links in a mini-roundabout.
        INTEGER AP,EX,R,FEED,X,A,B,INJ,XION,UB,DB,I,J,NC,CR,CW,WT,ARCS,
     1  FIRSTAB,FIRSTBA,TW,DA,UA,LINKS,XIONS,ZONESIN,ZONES
        COMMON INJ(300,8),XION(300),UB(300),DB(300),I(1200),J(1200),
     1  NC(1200),CR(1200,8),NODES,ARCS,ZONES,A(300),B(300),FIRSTAB(300),
     1  LASTAB(300),FIRSTBA(300),LASTBA(300),KJ(20),AP(4),EX(4),NL(300),
     1  CW(1200,8),JT(300),TW(300),DA(200),UA(200),R(5),
     1  FEED(50),LINKS,XIONS,ZONESIN
          IF(AP(L).EQ.UB(INJ(X,L)))THEN
            IF(KJ(4*L-3).NE.0)THEN
              FIRSTAB(INJ(X,L))=KJ(4*L-3)
              IF(KJ(4*L-2).NE.0)THEN
                LASTAB(INJ(X,L))=KJ(4*L-2)
              ELSE
                LASTAB(INJ(X,L))=KJ(4*L-3)
              END IF
            ELSE
              FIRSTAB(INJ(X,L))=KJ(4*L-2)
              LASTAB(INJ(X,L)) =KJ(4*L-2)
            END IF
          ELSE
            IF(KJ(4*L-3).NE.0)THEN
              FIRSTBA(INJ(X,L))=KJ(4*L-3)
              IF(KJ(4*L-2).NE.0)THEN
                LASTBA(INJ(X,L))=KJ(4*L-2)
```

```
            ELSE
              LASTBA(INJ(X,L))=KJ(4*L-3)
            END IF
          ELSE
            FIRSTBA(INJ(X,L))=KJ(4*L-2)
            LASTBA(INJ(X,L)) =KJ(4*L-2)
          END IF
        END IF
      RETURN
      END
C     Of subroutine ALMINI4(X,L)
C
      SUBROUTINE ALROUND4(X,L)
C     Relates arcs to links in a conventional 4-arm roundabout.
c
      INTEGER AP,EX,R,FEED,X,A,B,INJ,XION,UB,DB,I,J,NC,CR,CW,WT,ARCS,
     1 FIRSTAB,FIRSTBA,TW,DA,UA,LINKS,XIONS,ZONESIN,ZONES
      COMMON INJ(300,8),XION(300),UB(300),DB(300),I(1200),J(1200),
     1 NC(1200),CR(1200,8),NODES,ARCS,ZONES,A(300),B(300),FIRSTAB(300),
     1 LASTAB(300),FIRSTBA(300),LASTBA(300),KJ(20),AP(4),EX(4),NL(300),
     1 CW(1200,8),JT(300),TW(300),DA(200),UA(200),R(5),
     1 FEED(50),LINKS,XIONS,ZONESIN
        IF(AP(L).EQ.UB(INJ(X,L)))THEN
          IF(KJ(5*L-4).NE.0)THEN
            FIRSTAB(INJ(X,L))=KJ(5*L-4)
           IF(KJ(5*L-2).NE.0)THEN
             LASTAB(INJ(X,L))=KJ(5*L-2)
           ELSE
             IF(KJ(5*L-3).NE.0)THEN
               LASTAB(INJ(X,L))=KJ(5*L-3)
             ELSE
               LASTAB(INJ(X,L))=KJ(5*L-4)
             END IF
           END IF
         ELSE
           IF(KJ(5*L-3).NE.0)THEN
             FIRSTAB(INJ(X,L))=KJ(5*L-3)
             IF(KJ(5*L-2).NE.0)THEN
               LASTAB(INJ(X,L))=KJ(5*L-2)
             ELSE
               LASTAB(INJ(X,L))=KJ(5*L-3)
             END IF
           ELSE
             IF(KJ(5*L-2).NE.0)THEN
               FIRSTAB(INJ(X,L))=KJ(5*L-2)
               LASTAB(INJ(X,L))=KJ(5*L-2)
             END IF
           END IF
         END IF
       ELSE
         IF(KJ(5*L-4).NE.0)THEN
           FIRSTBA(INJ(X,L))=KJ(5*L-4)
           IF(KJ(5*L-2).NE.0)THEN
             LASTBA(INJ(X,L))=KJ(5*L-2)
           ELSE
             IF(KJ(5*L-3).NE.0)THEN
               LASTBA(INJ(X,L))=KJ(5*L-3)
```

```
                ELSE
                  LASTBA(INJ(X,L))=KJ(5*L-4)
                END IF
              END IF
            ELSE
              IF(KJ(5*L-3).NE.0)THEN
                FIRSTBA(INJ(X,L))=KJ(5*L-3)
                IF(KJ(5*L-2).NE.0)THEN
                  LASTBA(INJ(X,L))=KJ(5*L-2)
                ELSE
                  LASTBA(INJ(X,L))=KJ(5*L-3)
                END IF
              ELSE
                IF(KJ(5*L-2).NE.0)THEN
                  FIRSTBA(INJ(X,L))=KJ(5*L-2)
                  LASTBA(INJ(X,L))=KJ(5*L-2)
                END IF
              END IF
            END IF
          END IF
        RETURN
        END
C       Of subroutine ALROUND4(X,L)
c
        SUBROUTINE ARC4LINK(X,L)
C       For each link, L, feeding into a crossroads, X, this subroutine
c       determines whether the B node or the A node is at the junction
c       and computes FIRSTAB(L) and LASTAB(L) or FIRSTBA(L) and LASTBA(L)
C       respectively. This is to enable the total flow on link, L, to be
c       computed separately for the direction A to B and B to A from the
c       array TOTEFLOW(K) as output to the file, FLOW.DAT, by the program
c       POLYREAL. If those links from zones have their A to B direction
c       coded away from the zone no arcs will be created from the
c       destination vertex so FIRSTBA andd LASTBA will be 0 for links from
c       from zones and no means will be provided for computing the total
c       flow along a link into a destination. It can be assumed to equal
c       the demand.
c
        INTEGER AP,EK,R,FEED,X,A,B,INJ,XION,UB,DB,I,J,NC,CR,CW,WT,ARCS,
     1  FIRSTAB,FIRSTBA,TW,DA,UA,LINKS,XIONS,ZONESIN,ZONES
        COMMON INJ(300,8),XION(300),UB(300),DB(300),I(1200),J(1200),
     1  NC(1200),CR(1200,8),NODES,ARCS,ZONES,A(300),B(300),FIRSTAB(300),
     1  LASTAB(300),FIRSTBA(300),LASTBA(300),KJ(20),AP(4),EK(4),NL(300),
     1  CW(1200,8),JT(300),TW(300),DA(200),UA(200),R(5),
     1  FEED(50),LINKS,XIONS,ZONESIN
          IF(AP(L).EQ.UB(INJ(X,L)))THEN
            IF(KJ(3*L-2).NE.0)THEN
              FIRSTAB(INJ(X,L))=KJ(3*L-2)
              IF(KJ(3*L).NE.0)THEN
                LASTAB(INJ(X,L))=KJ(3*L)
              ELSE
                IF(KJ(3*L-1).NE.0)THEN
                  LASTAB(INJ(X,L))=KJ(3*L-1)
                ELSE
                  LASTAB(INJ(X,L))=KJ(3*L-2)
                END IF
              END IF
```

```
              ELSE
                IF(KJ(3*L-1).NE.0)THEN
                  FIRSTAB(INJ(X,L))=KJ(3*L-1)
                  IF(KJ(3*L).NE.0)THEN
                    LASTAB(INJ(X,L))=KJ(3*L)
                  ELSE
                    LASTAB(INJ(X,L))=KJ(3*L-1)
                  END IF
                ELSE
                  IF(KJ(3*L).NE.0)THEN
                    FIRSTAB(INJ(X,L))=KJ(3*L)
                    LASTAB(INJ(X,L))=KJ(3*L)
                  END IF
                END IF
              END IF
            ELSE
              IF(KJ(3*L-2).NE.0)THEN
                FIRSTBA(INJ(X,L))=KJ(3*L-2)
                IF(KJ(3*L).NE.0)THEN
                  LASTBA(INJ(X,L))=KJ(3*L)
                ELSE
                  IF(KJ(3*L-1).NE.0)THEN
                    LASTBA(INJ(X,L))=KJ(3*L-1)
                  ELSE
                    LASTBA(INJ(X,L))=KJ(3*L-2)
                  END IF
                END IF
              ELSE
                IF(KJ(3*L-1).NE.0)THEN
                  FIRSTBA(INJ(X,L))=KJ(3*L-1)
                  IF(KJ(3*L).NE.0)THEN
                    LASTBA(INJ(X,L))=KJ(3*L)
                  ELSE
                    LASTBA(INJ(X,L))=KJ(3*L-1)
                  END IF
                ELSE
                  IF(KJ(3*L).NE.0)THEN
                    FIRSTBA(INJ(X,L))=KJ(3*L)
                    LASTBA(INJ(X,L))=KJ(3*L)
                  END IF
                END IF
              END IF
            END IF
          RETURN
          END
C         Of subroutine ARC4LINK(X,L)
C
          SUBROUTINE CON3ARC1
C         Specify conflicts
C         Free-for-all 3-way junction
          INTEGER AP,EX,R,FEED,X,A,B,INJ,XION,UB,DB,I,J,NC,CR,CW,ARCS,
     1    FIRSTAB,FIRSTBA,TW,DA,UA,LINKS,XIONS,ZONESIN,ZONES,WT3,CON3,
     1    CONRF
          COMMON / C3 / CON3(6,3),CONRF(6),WT3(42)
```

```
      COMMON INJ(300,8),XION(300),UB(300),DB(300),I(1200),J(1200),
     1 NC(1200),CR(1200,8),NODES,ARCS,ZONES,A(300),B(300),FIRSTAB(300),
     1 LASTAB(300),FIRSTBA(300),LASTBA(300),KJ(20),AP(4),EX(4),NL(300),
     1 CW(1200,8),JT(300),TW(300),DA(200),UA(200),R(5),
     1 FEED(50),LINKS,XIONS,ZONESIN
      SAVE /C3/
      DO 2 KP=1,5,2
        IF(KJ(KP).NE.0)THEN
          KR = 1
          IF(KJ(CON3(KP,1)).NE.0)THEN
            CR(KJ(KP),KR) = KJ(CON3(KP,1))
            CW(KJ(KP),KR) = WT3(1)
            KR = KR + 1
          END IF
          NC(KJ(KP)) = KR - 1
        END IF
    2 CONTINUE
      DO 4 KP=2,6,2
        IF(KJ(KP).NE.0)THEN
          KR = 1
          DO 6 KO=1,3
            IF(KJ(CON3(KP,KO)).NE.0)THEN
              CR(KJ(KP),KR) = KJ(CON3(KP,KO))
              CW(KJ(KP),KR) = WT3(1+KO)
              KR = KR + 1
            END IF
    6     CONTINUE
          NC(KJ(KP)) = KR - 1
        END IF
    4 CONTINUE
      RETURN
      END
C     Of subroutine CON3ARC1
C
      SUBROUTINE CON3RF
C     Specify conflicts
C     Right-turning flyover to and from minor road
      INTEGER AP,EX,R,FEED,X,A,B,INJ,XION,UB,DB,I,J,NC,CR,CW,ARCS,
     1 FIRSTAB,FIRSTBA,TW,DA,UA,LINKS,XIONS,ZONESIN,ZONES,WT3,CON3,
     1 CONRF
      COMMON / C3 / CON3(6,3),CONRF(6),WT3(42)
      COMMON INJ(300,8),XION(300),UB(300),DB(300),I(1200),J(1200),
     1 NC(1200),CR(1200,8),NODES,ARCS,ZONES,A(300),B(300),FIRSTAB(300),
     1 LASTAB(300),FIRSTBA(300),LASTBA(300),KJ(20),AP(4),EX(4),NL(300),
     1 CW(1200,8),JT(300),TW(300),DA(200),UA(200),R(5),
     1 FEED(50),LINKS,XIONS,ZONESIN
      SAVE /C3/
      DO 2 KP=1,6
        IF(KJ(KP).NE.0)THEN
          KR = 1
          IF(KJ(CONRF(KP)).NE.0)THEN
            CR(KJ(KP),KR) = KJ(CONRF(KP))
            CW(KJ(KP),KR) = WT3(36+KP)
            KR = KR + 1
          END IF
          NC(KJ(KP)) = KR - 1
        END IF
```

```
    2   CONTINUE
        RETURN
        END
C       Of subroutine CON3RF
C
        SUBROUTINE CON4ARC1
C       12 JUNE 89
c       Free-for-all 4-way junction
c
        INTEGER XION,DB,UB,CR,CW,ARCS,ZONES,A,B,FIRSTAB,FIRSTBA,
     1  AP,EX,R,FEED,TW,DA,UA,XIONS,ZONESIN,CON4,WT4,CON4MR,CONRO,CONFO
        COMMON / C4 / CON4(12,7),WT4(104),CON4MR(16,2),CONRO(20,6),
     1  CONFO(12,6)
        COMMON INJ(300,8),XION(300),UB(300),DB(300),I(1200),J(1200),
     1  NC(1200),CR(1200,8),NODES,ARCS,ZONES,A(300),B(300),FIRSTAB(300),
     1  LASTAB(300),FIRSTBA(300),LASTBA(300),KJ(20),AP(4),EX(4),NL(300),
     1  CW(1200,8),JT(300),TW(300),DA(200),UA(200),R(5),
     1  FEED(50),LINKS,XIONS,ZONESIN
        SAVE /C4/
          DO 2 KP=1,10,3
            IF(KJ(KP).NE.0)THEN
              KR = 1
              DO 4 KO=1,2
                IF(KJ(CON4(KP,KO)).NE.0)THEN
                  CR(KJ(KP),KR) = KJ(CON4(KP,KO))
                  CW(KJ(KP),KR) = WT4(KO)
                  KR = KR + 1
                END IF
    4         CONTINUE
              NC(KJ(KP)) = KR - 1
            END IF
    2     CONTINUE
          DO 6 KP=2,11,3
            IF(KJ(KP).NE.0)THEN
              KR = 1
              DO 8 KO=1,6
                IF(KJ(CON4(KP,KO)).NE.0)THEN
                  CR(KJ(KP),KR) = KJ(CON4(KP,KO))
                  CW(KJ(KP),KR) = WT4(2+KO)
                  KR = KR + 1
                END IF
    8         CONTINUE
              NC(KJ(KP)) = KR - 1
            END IF
    6     CONTINUE
          DO 10 KP=3,12,3
            IF(KJ(KP).NE.0)THEN
              KR = 1
              DO 12 KO=1,7
                IF(KJ(CON4(KP,KO)).NE.0)THEN
                  CR(KJ(KP),KR) = KJ(CON4(KP,KO))
                  CW(KJ(KP),KR) = WT4(8+KO)
                  KR = KR + 1
                END IF
   12         CONTINUE
              NC(KJ(KP)) = KR - 1
            END IF
```

```
   10     CONTINUE
          RETURN
          END
C         of subroutine CON4ARC1
C
          SUBROUTINE CON4ARC2
C         13 JUNE 89
c         Priority 4-way junction
          INTEGER XION,DB,UB,CR,CW,ARCS,ZONES,A,B,FIRSTAB,FIRSTBA,
     1    AP,EX,R,FEED,TW,DA,UA,XIONS,ZONESIN,CON4,WT4,CON4MR,CONRO,CONFO
          COMMON / C4 / CON4(12,7),WT4(104),CON4MR(16,2),CONRO(20,6),
     1    CONFO(12,6)
          COMMON INJ(300,8),XION(300),UB(300),DB(300),I(1200),J(1200),
     1    NC(1200),CR(1200,8),NODES,ARCS,ZONES,A(300),B(300),FIRSTAB(300),
     1    LASTAB(300),FIRSTBA(300),LASTBA(300),KJ(20),AP(4),EX(4),NL(300),
     1    CW(1200,8),JT(300),TW(300),DA(200),UA(200),R(5),
     1    FEED(50),LINKS,XIONS,ZONESIN
          SAVE /C4/
            DO 2 KP=1,7,6
              IF(KJ(KP).NE.0)THEN
                KR = 1
                DO 4 KO=1,2
                  IF(KJ(CON4(KP,KO)).NE.0)THEN
                    CR(KJ(KP),KR) = KJ(CON4(KP,KO))
                    CW(KJ(KP),KR) = WT4(15+KO)
                    KR = KR + 1
                  END IF
    4           CONTINUE
                NC(KJ(KP)) = KR - 1
              END IF
    2       CONTINUE
            DO 6 KP=2,8,6
              IF(KJ(KP).NE.0)THEN
                KR = 1
                DO 8 KO=1,6
                  IF(KJ(CON4(KP,KO)).NE.0)THEN
                    CR(KJ(KP),KR) = KJ(CON4(KP,KO))
                    CW(KJ(KP),KR) = WT4(17+KO)
                    KR = KR + 1
                  END IF
    8           CONTINUE
                NC(KJ(KP)) = KR - 1
              END IF
    6       CONTINUE
            DO 10 KP=3,9,6
              IF(KJ(KP).NE.0)THEN
                KR = 1
                DO 12 KO=1,7
                  IF(KJ(CON4(KP,KO)).NE.0)THEN
                    CR(KJ(KP),KR) = KJ(CON4(KP,KO))
                    CW(KJ(KP),KR) = WT4(23+KO)
                    KR = KR + 1
                  END IF
   12           CONTINUE
                NC(KJ(KP)) = KR - 1
              END IF
   10       CONTINUE
```

```
            DO 14 KP=4,10,6
               IF(KJ(KP).NE.0)THEN
                 KR = 1
                 DO 16 KO=1,2
                    IF(KJ(CON4(KP,KO)).NE.0)THEN
                      CR(KJ(KP),KR) = KJ(CON4(KP,KO))
                      CW(KJ(KP),KR) = WT4(30+KO)
                      KR = KR + 1
                    END IF
   16            CONTINUE
                 NC(KJ(KP)) = KR - 1
               END IF
   14       CONTINUE
            DO 18 KP=5,11,6
               IF(KJ(KP).NE.0)THEN
                 KR = 1
                 DO 20 KO=1,6
                    IF(KJ(CON4(KP,KO)).NE.0)THEN
                      CR(KJ(KP),KR) = KJ(CON4(KP,KO))
                      CW(KJ(KP),KR) = WT4(32+KO)
                      KR = KR + 1
                    END IF
   20            CONTINUE
                 NC(KJ(KP)) = KR - 1
               END IF
   18       CONTINUE
            DO 22 KP=6,12,6
               IF(KJ(KP).NE.0)THEN
                 KR = 1
                 DO 24 KO=1,7
                    IF(KJ(CON4(KP,KO)).NE.0)THEN
                      CR(KJ(KP),KR) = KJ(CON4(KP,KO))
                      CW(KJ(KP),KR) = WT4(38+KO)
                      KR = KR + 1
                    END IF
   24            CONTINUE
                 NC(KJ(KP)) = KR - 1
               END IF
   22       CONTINUE
            RETURN
            END
C           of subroutine CON4ARC2
C
            SUBROUTINE CON4ARC3
C           13 JUNE 89
c           4-way signalised junction
            INTEGER XION,DB,UB,CR,CW,ARCS,ZONES,A,B,FIRSTAB,FIRSTBA,
     1      AP,EX,R,FEED,TW,DA,UA,XIONS,ZONESIN,CON4,WT4,CON4MR,CONRO,CONFO
            COMMON / C4 / CON4(12,7),WT4(104),CON4MR(16,2),CONRO(20,6),
     1      CONFO(12,6)
            COMMON INJ(300,8),XION(300),UB(300),DB(300),I(1200),J(1200),
     1      NC(1200),CR(1200,8),NODES,ARCS,ZONES,A(300),B(300),FIRSTAB(300),
     1      LASTAB(300),FIRSTBA(300),LASTBA(300),KJ(20),AP(4),EX(4),NL(300),
     1      CW(1200,8),JT(300),TW(300),DA(200),UA(200),R(5),
     1      FEED(50),LINKS,XIONS,ZONESIN
            SAVE /C4/
```

```
      DO 2 KP=1,10,3
         IF(KJ(KP).NE.0)THEN
           KR = 1
           DO 4 KO=1,2
              IF(KJ(CON4(KP,KO)).NE.0)THEN
                CR(KJ(KP),KR) = KJ(CON4(KP,KO))
                CW(KJ(KP),KR) = WT4(45+KO)
                KR = KR + 1
              END IF
    4       CONTINUE
            NC(KJ(KP)) = KR - 1
         END IF
    2  CONTINUE
       DO 6 KP=2,11,3
         IF(KJ(KP).NE.0)THEN
           KR = 1
           DO 8 KO=1,6
              IF(KJ(CON4(KP,KO)).NE.0)THEN
                CR(KJ(KP),KR) = KJ(CON4(KP,KO))
                CW(KJ(KP),KR) = WT4(47+KO)
                KR = KR + 1
              END IF
    8       CONTINUE
            NC(KJ(KP)) = KR - 1
         END IF
    6  CONTINUE
       DO 10 KP=3,12,3
         IF(KJ(KP).NE.0)THEN
           KR = 1
           DO 12 KO=1,7
              IF(KJ(CON4(KP,KO)).NE.0)THEN
                CR(KJ(KP),KR) = KJ(CON4(KP,KO))
                CW(KJ(KP),KR) = WT4(53+KO)
                KR = KR + 1
              END IF
   12       CONTINUE
            NC(KJ(KP)) = KR - 1
         END IF
   10  CONTINUE
       RETURN
       END
C      of subroutine CON4ARC3
C
       SUBROUTINE CON4ARC4
C      Mini-roundabout
C      13 JUNE 89
c
       INTEGER XION,DB,UB,CR,CW,ARCS,ZONES,A,B,FIRSTAB,FIRSTBA,
     1 AP,EX,R,FEED,TW,DA,UA,XIONS,ZONESIN,CON4,WT4,CON4MR,CONRO,CONFO
       COMMON / C4 / CON4(12,7),WT4(104),CON4MR(16,2),CONRO(20,6),
     1 CONFO(12,6)
       COMMON INJ(300,8),XION(300),UB(300),DB(300),I(1200),J(1200),
     1 NC(1200),CR(1200,8),NODES,ARCS,ZONES,A(300),B(300),FIRSTAB(300),
     1 LASTAB(300),FIRSTBA(300),LASTBA(300),KJ(20),AP(4),EX(4),NL(300),
     1 CW(1200,8),JT(300),TW(300),DA(200),UA(200),R(5),
     1 FEED(50),LINKS,XIONS,ZONESIN
       SAVE /C4/
```

```
      DO 2 KP=1,13,4
         IF(KJ(KP).NE.0)THEN
           KR = 1
              IF(KJ(CON4MR(KP,1)).NE.0)THEN
                CR(KJ(KP),KR) = KJ(CON4MR(KP,1))
                CW(KJ(KP),KR) = WT4(61)
                KR = KR + 1
              END IF
           NC(KJ(KP)) = KR - 1
         END IF
 2    CONTINUE
      DO 6 KP=2,14,4
         IF(KJ(KP).NE.0)THEN
           KR = 1
           DO 8 KO=1,2
              IF(KJ(CON4MR(KP,KO)).NE.0)THEN
                CR(KJ(KP),KR) = KJ(CON4MR(KP,KO))
                CW(KJ(KP),KR) = WT4(61+KO)
                KR = KR + 1
              END IF
 8         CONTINUE
           NC(KJ(KP)) = KR - 1
         END IF
 6    CONTINUE
      DO 10 KP=3,15,4
         IF(KJ(KP).NE.0)THEN
           KR = 1
           DO 12 KO=1,2
              IF(KJ(CON4MR(KP,KO)).NE.0)THEN
                CR(KJ(KP),KR) = KJ(CON4MR(KP,KO))
                CW(KJ(KP),KR) = WT4(63+KO)
                KR = KR + 1
              END IF
12         CONTINUE
           NC(KJ(KP)) = KR - 1
         END IF
10    CONTINUE
      DO 14 KP=4,16,4
         IF(KJ(KP).NE.0)THEN
           KR = 1
              IF(KJ(CON4MR(KP,1)).NE.0)THEN
                CR(KJ(KP),KR) = KJ(CON4MR(KP,1))
                CW(KJ(KP),KR) = WT4(66)
                KR = KR + 1
              END IF
           NC(KJ(KP)) = KR - 1
         END IF
14    CONTINUE
      RETURN
      END
C     of subroutine CON4ARC4
```

```
      SUBROUTINE CON4ARC5
C     Roundabout
C     13 JUNE 89
c
      INTEGER XION,DB,UB,CR,CW,ARCS,ZONES,A,B,FIRSTAB,FIRSTBA,
     1 AP,EX,R,FEED,TW,DA,UA,XIONS,ZONESIN,CON4,WT4,CON4MR,CONRO,CONFO
      COMMON / C4 / CON4(12,7),WT4(104),CON4MR(16,2),CONRO(20,6),
     1 CONFO(12,6)
      COMMON INJ(300,8),XION(300),UB(300),DB(300),I(1200),J(1200),
     1 NC(1200),CR(1200,8),NODES,ARCS,ZONES,A(300),B(300),FIRSTAB(300),
     1 LASTAB(300),FIRSTBA(300),LASTBA(300),KJ(20),AP(4),EX(4),NL(300),
     1 CW(1200,8),JT(300),TW(300),DA(200),UA(200),R(5),
     1 FEED(50),LINKS,XIONS,ZONESIN
      SAVE /C4/
        DO 2 KP=1,16,5
          IF(KJ(KP).NE.0)THEN
            KR = 1
            DO 4 KO=1,2
              IF(KJ(CONRO(KP,KO)).NE.0)THEN
                CR(KJ(KP),KR) = KJ(CONRO(KP,KO))
                CW(KJ(KP),KR) = WT4(66+KO)
                KR = KR + 1
              END IF
 4          CONTINUE
            NC(KJ(KP)) = KR - 1
          END IF
 2      CONTINUE
        DO 6 KP=2,17,5
          IF(KJ(KP).NE.0)THEN
            KR = 1
            DO 8 KO=1,6
              IF(KJ(CONRO(KP,KO)).NE.0)THEN
                CR(KJ(KP),KR) = KJ(CONRO(KP,KO))
                CW(KJ(KP),KR) = WT4(68+KO)
                KR = KR + 1
              END IF
 8          CONTINUE
            NC(KJ(KP)) = KR - 1
          END IF
 6      CONTINUE
        DO 10 KP=3,18,5
          IF(KJ(KP).NE.0)THEN
            KR = 1
            DO 12 KO=1,3
              IF(KJ(CONRO(KP,KO)).NE.0)THEN
                CR(KJ(KP),KR) = KJ(CONRO(KP,KO))
                CW(KJ(KP),KR) = WT4(74+KO)
                KR = KR + 1
              END IF
 12         CONTINUE
            NC(KJ(KP)) = KR - 1
          END IF
 10     CONTINUE
        DO 14 KP=4,19,5
          IF(KJ(KP).NE.0)THEN
            KR = 1
```

```
                IF(KJ(CONRO(KP,1)).NE.0)THEN
                  CR(KJ(KP),KR) = KJ(CONRO(KP,1))
                  CW(KJ(KP),KR) = WT4(78)
                  KR = KR + 1
                END IF
              NC(KJ(KP)) = KR - 1
            END IF
   14     CONTINUE
          DO 16 KP=5,20,5
            IF(KJ(KP).NE.0)THEN
              KR = 1
              DO 18 KO=1,4
                IF(KJ(CONRO(KP,KO)).NE.0)THEN
                  CR(KJ(KP),KR) = KJ(CONRO(KP,KO))
                  CW(KJ(KP),KR) = WT4(78+KO)
                  KR = KR + 1
                END IF
   18         CONTINUE
              NC(KJ(KP)) = KR - 1
            END IF
   16     CONTINUE
        RETURN
        END
C       of subroutine CON4ARC5
C
        SUBROUTINE CON4ARC6
C       13 JUNE 89
c       Fly-over (links 2 and 4 fly over links 1 and 3)
C
        INTEGER XION,DB,UB,CR,CW,ARCS,ZONES,A,B,FIRSTAB,FIRSTBA,
     1  AP,EX,R,FEED,TW,DA,UA,XIONS,ZONESIN,CON4,WT4,CON4MR,CONRO,CONFO
        COMMON / C4 / CON4(12,7),WT4(104),CON4MR(16,2),CONRO(20,6),
     1  CONFO(12,6)
        COMMON INJ(300,8),XION(300),UB(300),DB(300),I(1200),J(1200),
     1  NC(1200),CR(1200,8),NODES,ARCS,ZONES,A(300),B(300),FIRSTAB(300),
     1  LASTAB(300),FIRSTBA(300),LASTBA(300),KJ(20),AP(4),EX(4),NL(300),
     1  CW(1200,8),JT(300),TW(300),DA(200),UA(200),R(5),
     1  FEED(50),LINKS,XIONS,ZONESIN
        SAVE /C4/
          DO 2 KP=1,7,6
            IF(KJ(KP).NE.0)THEN
              KR = 1
              DO 4 KO=1,2
                IF(KJ(CONFO(KP,KO)).NE.0)THEN
                  CR(KJ(KP),KR) = KJ(CONFO(KP,KO))
                  CW(KJ(KP),KR) = WT4(82+KO)
                  KR = KR + 1
                END IF
    4         CONTINUE
              NC(KJ(KP)) = KR - 1
            END IF
    2     CONTINUE
          DO 6 KP=2,8,6
            IF(KJ(KP).NE.0)THEN
              KR = 1
              DO 8 KO=1,4
```

```
            IF(KJ(CONFO(KP,KO)).NE.0)THEN
              CR(KJ(KP),KR) = KJ(CONFO(KP,KO))
              CW(KJ(KP),KR) = WT4(84+KO)
              KR = KR + 1
            END IF
 8        CONTINUE
          NC(KJ(KP)) = KR - 1
        END IF
 6    CONTINUE
      DO 10 KP=3,9,6
        IF(KJ(KP).NE.0)THEN
          KR = 1
          DO 12 KO=1,6
            IF(KJ(CONFO(KP,KO)).NE.0)THEN
              CR(KJ(KP),KR) = KJ(CONFO(KP,KO))
              CW(KJ(KP),KR) = WT4(88+KO)
              KR = KR + 1
            END IF
12        CONTINUE
          NC(KJ(KP)) = KR - 1
        END IF
10    CONTINUE
      DO 14 KP=4,10,6
        IF(KJ(KP).NE.0)THEN
          KR = 1
          DO 16 KO=1,2
            IF(KJ(CONFO(KP,KO)).NE.0)THEN
              CR(KJ(KP),KR) = KJ(CONFO(KP,KO))
              CW(KJ(KP),KR) = WT4(94+KO)
              KR = KR + 1
            END IF
16        CONTINUE
          NC(KJ(KP)) = KR - 1
        END IF
14    CONTINUE
      DO 18 KP=5,11,6
        IF(KJ(KP).NE.0)THEN
          KR = 1
          DO 20 KO=1,2
            IF(KJ(CONFO(KP,KO)).NE.0)THEN
              CR(KJ(KP),KR) = KJ(CONFO(KP,KO))
              CW(KJ(KP),KR) = WT4(96+KO)
              KR = KR + 1
            END IF
20        CONTINUE
          NC(KJ(KP)) = KR - 1
        END IF
18    CONTINUE
      DO 22 KP=6,12,6
        IF(KJ(KP).NE.0)THEN
          KR = 1
          DO 24 KO=1,6
            IF(KJ(CONFO(KP,KO)).NE.0)THEN
              CR(KJ(KP),KR) = KJ(CONFO(KP,KO))
              CW(KJ(KP),KR) = WT4(98+KO)
              KR = KR + 1
            END IF
```

```
   24          CONTINUE
               NC(KJ(KP)) = KR - 1
             END IF
   22      CONTINUE
           RETURN
           END
C          of subroutine CON4ARC6
C
           SUBROUTINE MAKEARCS
           INTEGER XION,DB,UB,CR,CW,WT,ARCS,ZONES,A,B,FIRSTAB,FIRSTBA,
     1     AP,EX,R,FEED,TW,DA,UA,X,XIONS,ZONESIN
           COMMON INJ(300,8),XION(300),UB(300),DB(300),I(1200),J(1200),
     1     NC(1200),CR(1200,8),NODES,ARCS,ZONES,A(300),B(300),FIRSTAB(300),
     1     LASTAB(300),FIRSTBA(300),LASTBA(300),KJ(20),AP(4),EX(4),NL(300),
     1     CW(1200,8),JT(300),TW(300),DA(200),UA(200),R(5),
     1     FEED(50),LINKS,XIONS,ZONESIN
           ARCS = 0
           DO 60 X=1,XIONS
              IF(NL(X).EQ.3)THEN
                CALL TJUNCTN(X)
              ELSE IF(NL(X).EQ.4)THEN
                IF(JT(X).EQ.4)THEN
                  CALL MINI4(X)
                ELSE IF(JT(X).EQ.5)THEN
                  CALL ROUND4(X)
                ELSE
                  CALL XROADS(X)
                END IF
              END IF
   60      CONTINUE
           RETURN
           END
C          Of subroutine MAKEARCS
C
           SUBROUTINE MINI4(X)
C
           INTEGER AP,EX,R,FEED,X,A,B,INJ,XION,UB,DB,I,J,NC,CR,CW,WT,ARCS,
     1     FIRSTAB,FIRSTBA,TW,DA,UA,XIONS,ZONESIN,ZONES
           COMMON INJ(300,8),XION(300),UB(300),DB(300),I(1200),J(1200),
     1     NC(1200),CR(1200,8),NODES,ARCS,ZONES,A(300),B(300),FIRSTAB(300),
     1     LASTAB(300),FIRSTBA(300),LASTBA(300),KJ(20),AP(4),EX(4),NL(300),
     1     CW(1200,8),JT(300),TW(300),DA(200),UA(200),R(5),
     1     FEED(50),LINKS,XIONS,ZONESIN
C          Create approach and exit vertices
           DO 2 N=1,4
              AP(N)=0
              EX(N)=0
   2       CONTINUE
           DO 10 JL=1,4
           IF(B(INJ(X,JL)).EQ.XION(X))THEN
             AP(JL)=UB(INJ(X,JL))
             EX(JL)=DB(INJ(X,JL))
           ELSE IF(A(INJ(X,JL)).EQ.XION(X))THEN
             AP(JL)=DB(INJ(X,JL))
             EX(JL)=UB(INJ(X,JL))
           ELSE
           PRINT*,'Mistake in LINKS.DAT file for junction  ',XION(X)
```

```
      STOP 'Check your LINKS.DAT file and start again'
      END IF
   10 CONTINUE
C       create roundabout nodes at each entry
        N = NODES + 1
        DO 12 IR=1,4
           R(IR) = N
           N = N + 1
   12   CONTINUE
        NODES = N - 1
      DO 20 K=1,16
         KJ(K)=0
   20 CONTINUE
      K=ARCS+1
c     Create arcs from AP(1)
      IF(AP(1).NE.0)THEN
         IF(EX(2).NE.0)THEN
           I(K)=AP(1)
           J(K)=EX(2)
           KJ(1)=K
           K=K+1
         END IF
         I(K)=AP(1)
         J(K)=R(2)
         KJ(2)=K
         K=K + 1
      END IF
      IF(EX(2).NE.0)THEN
        I(K)=R(1)
        J(K)=EX(2)
        KJ(3)=K
        K=K+1
      END IF
      I(K)=R(1)
      J(K)=R(2)
      KJ(4)=K
      K = K + 1
         CALL ALMINI4(X,1)
c     Create arcs from AP(2)
      IF(AP(2).NE.0)THEN
        IF(EX(3).NE.0)THEN
          I(K)=AP(2)
          J(K)=EX(3)
          KJ(5)=K
          K=K+1
        END IF
        I(K)=AP(2)
        J(K)=R(3)
        KJ(6)=K
        K=K+1
      END IF
      IF(EX(3).NE.0)THEN
        I(K)=R(2)
        J(K)=EX(3)
        KJ(7)=K
        K=K+1
      END IF
```

```
      I(K)=R(2)
      J(K)=R(3)
      KJ(8)=K
      K = K + 1
        CALL ALMINI4(X,2)
C     Create arcs from AP(3)
      IF(AP(3).NE.0)THEN
        IF(EX(4).NE.0)THEN
          I(K)=AP(3)
          J(K)=EX(4)
          KJ(9)=K
          K=K+1
        END IF
        I(K)=AP(3)
        J(K)=R(4)
        KJ(10)=K
        K=K+1
      END IF
      IF(EX(4).NE.0)THEN
        I(K)=R(3)
        J(K)=EX(4)
        KJ(11)=K
        K=K+1
      END IF
      I(K)=R(3)
      J(K)=R(4)
      KJ(12)=K
      K = K + 1
        CALL ALMINI4(X,3)
c     Create arcs from AP(4)
      IF(AP(4).NE.0)THEN
        IF(EX(1).NE.0)THEN
          I(K)=AP(4)
          J(K)=EX(1)
          KJ(13)=K
          K=K+1
        END IF
        I(K)=AP(4)
        J(K)=R(1)
        KJ(14)=K
        K=K+1
      END IF
      IF(EX(1).NE.0)THEN
        I(K)=R(4)
        J(K)=EX(1)
        KJ(15)=K
        K=K+1
      END IF
      I(K)=R(4)
      J(K)=R(1)
      KJ(16)=K
      K = K + 1
        CALL ALMINI4(X,4)
      ARCS=K-1
      CALL CON4ARC4
      RETURN
      END
```

```
      SUBROUTINE ODVERT
C     Creates origin and destination nodes
      INTEGER XION,DB,UB,CR,CW,WT,ARCS,ZONES,A,B,FIRSTAB,FIRSTBA,
     1 AP,EX,R,FEED,TW,DA,UA,XIONS,ZONESIN
      COMMON INJ(300,8),XION(300),UB(300),DB(300),I(1200),J(1200),
     1 NC(1200),CR(1200,8),NODES,ARCS,ZONES,A(300),B(300),FIRSTAB(300),
     1 LASTAB(300),FIRSTBA(300),LASTBA(300),KJ(20),AP(4),EX(4),NL(300),
     1 CW(1200,8),JT(300),TW(300),DA(200),UA(200),R(5),
     1 FEED(50),LINKS,XIONS,ZONESIN
      NOL = 0
      DO 20 MZ=1,ZONES
         DO 30 MF=1,FEED(MZ)
            L = NOL + MF
            DA(L) = 2*MZ - 1
            UA(L) = 2*MZ
C     The values of DA(L) will be the same for all links from zone MZ
30       CONTINUE
C     All links from zone MZ have the same node downstream of their
c     A node which is the zone. etc.
         NOL=L
20    CONTINUE
      DO 40 L=1,ZONESIN
         UB(L) = DA(L)
         IF(TW(L).EQ.1)THEN
            DB(L)= UA(L)
         ELSE
            DB(L)=0
         END IF
40    CONTINUE
C     All  zone connectors have been processed so there are
c     LINKS-ZONESIN links left and 2*ZONES nodes have been created.
      END
C      of subroutine ODVERT
C
      SUBROUTINE PRINT
C
C     Prints arc numbers, start vertices, finish vertices, number of
c     conflicting arcs in the file ARCS.OUT
c     Prints the arcs conflicting with each arc in the file,CROSSFLO.DAT.
      INTEGER A,B,INJ,XION,UB,DB,I,J,NC,CR,CW,WT,ARCS,FIRSTAB,FIRSTBA,
     1 AP,EX,R,FEED,TW,DA,UA,XIONS,ZONESIN,ZONES
      COMMON INJ(300,8),XION(300),UB(300),DB(300),I(1200),J(1200),
     1 NC(1200),CR(1200,8),NODES,ARCS,ZONES,A(300),B(300),FIRSTAB(300),
     1 LASTAB(300),FIRSTBA(300),LASTBA(300),KJ(20),AP(4),EX(4),NL(300),
     1 CW(1200,8),JT(300),TW(300),DA(200),UA(200),R(5),
     1 FEED(50),LINKS,XIONS,ZONESIN
      WRITE(6,*)NODES,ARCS,ZONES
      DO 10 K=1,ARCS
         WRITE(6,*)K,I(K),J(K)
10    CONTINUE
9010  FORMAT(9I5)
9020  FORMAT(5X,8I5)
      DO 20 K=1,ARCS
      WRITE(8,9010)NC(K),(CR(K,NCF),NCF=1,8)
      WRITE(8,9020)(CW(K,NCF),NCF=1,8)
20    CONTINUE
      RETURN
```

```
      END
C     Of subroutine PRINT
C
      SUBROUTINE LINKVERT(L)
C     Creates remaining nodes corresponding to ordinary links
      INTEGER XION,DB,UB,CR,CW,WT,ARCS,ZONES,A,B,FIRSTAB,FIRSTBA,
     1 X,AP,EX,R,FEED,TW,DA,UA,XIONS,ZONESIN
      COMMON INJ(300,8),XION(300),UB(300),DB(300),I(1200),J(1200),
     1 NC(1200),CR(1200,8),NODES,ARCS,ZONES,A(300),B(300),FIRSTAB(300),
     1 LASTAB(300),FIRSTBA(300),LASTBA(300),KJ(20),AP(4),EX(4),NL(300),
     1 CW(1200,8),JT(300),TW(300),DA(200),UA(200),R(5),
     1 FEED(50),LINKS,XIONS,ZONESIN
      N = 2*ZONES + 1
      DO 50 ML=L,LINKS
        UB(ML) = N
        N = N + 1
        IF(TW(ML).EQ.1)THEN
          DB(ML) = N
          N = N + 1
        ELSE
          DB(ML)=0
        END IF
   50 CONTINUE
      NODES = N - 1
c     Otherwise it would be 1 more than the number of nodes
      RETURN
      END
C      of subroutine LINKVERT
C
      SUBROUTINE ROUND4(X)
C     Conventional 4-arm roundabout
c
      INTEGER AP,EX,R,FEED,X,A,B,INJ,XION,UB,DB,I,J,NC,CR,CW,WT,ARCS,
     1 FIRSTAB,FIRSTBA,TW,DA,UA,XIONS,ZONESIN,ZONES
      COMMON INJ(300,8),XION(300),UB(300),DB(300),I(1200),J(1200),
     1 NC(1200),CR(1200,8),NODES,ARCS,ZONES,A(300),B(300),FIRSTAB(300),
     1 LASTAB(300),FIRSTBA(300),LASTBA(300),KJ(20),AP(4),EX(4),NL(300),
     1 CW(1200,8),JT(300),TW(300),DA(200),UA(200),R(5),
     1 FEED(50),LINKS,XIONS,ZONESIN
C     Create approach and exit vertices
      DO 2 N=1,4
        AP(N)=0
        EX(N)=0
   2  CONTINUE
      DO 10 JL=1,4
      IF(B(INJ(X,JL)).EQ.XION(X))THEN
        AP(JL)=UB(INJ(X,JL))
        EX(JL)=DB(INJ(X,JL))
      ELSE IF(A(INJ(X,JL)).EQ.XION(X))THEN
        AP(JL)=DB(INJ(X,JL))
        EX(JL)=UB(INJ(X,JL))
      ELSE
      PRINT*,'Mistake in LINKS.DAT file for junction  ',XION(X)
      STOP 'Check your LINKS.DAT file and start again'
      END IF
  10  CONTINUE
C       create roundabout nodes at each entry
```

```
          N = NODES + 1
          DO 12 IR=1,4
             R(IR) = N
             N = N + 1
   12     CONTINUE
          NODES = N - 1
         DO 20 K=1,20
            KJ(K)=0
   20    CONTINUE
         K=ARCS+1
c        Create arcs from AP(1)
         IF(AP(1).NE.0)THEN
            IF(EX(2).NE.0)THEN
              I(K)=AP(1)
              J(K)=EX(2)
              KJ(1)=K
              K=K+1
            END IF
            IF(EX(3).NE.0)THEN
              I(K)=AP(1)
              J(K)=EX(3)
              KJ(2)=K
              K = K + 1
            END IF
            I(K)=AP(1)
            J(K)=R(2)
            KJ(3)=K
            K=K + 1
         END IF
         I(K)=R(2)
         J(K)=R(3)
         KJ(4)=K
         K = K + 1
         IF(EX(4).NE.0)THEN
           I(K)=R(3)
           J(K)=EX(4)
           KJ(5)=K
           K=K+1
         END IF
            CALL ALROUND4(X,1)
c        Create arcs from AP(2)
         IF(AP(2).NE.0)THEN
           IF(EX(3).NE.0)THEN
             I(K)=AP(2)
             J(K)=EX(3)
             KJ(6)=K
             K=K+1
           END IF
           IF(EX(4).NE.0)THEN
             I(K)=AP(2)
             J(K)=EX(4)
             KJ(7)=K
             K=K+1
           END IF
           I(K)=AP(2)
           J(K)=R(3)
           KJ(8)=K
```

```
          K=K+1
        END IF
        I(K)=R(3)
        J(K)=R(4)
        KJ(9)=K
        K = K + 1
        IF(EX(1).NE.0)THEN
          I(K)=R(4)
          J(K)=EX(1)
          KJ(10)=K
          K = K + 1
        END IF
          CALL ALROUND4(X,2)
C       Create arcs from AP(3)
        IF(AP(3).NE.0)THEN
          IF(EX(4).NE.0)THEN
            I(K)=AP(3)
            J(K)=EX(4)
            KJ(11)=K
            K=K+1
          END IF
          IF(EX(1).NE.0)THEN
            I(K)=AP(3)
            J(K)=EX(1)
            KJ(12)=K
            K=K+1
          END IF
          I(K)=AP(3)
          J(K)=R(4)
          KJ(13)=K
          K=K+1
        END IF
        I(K)=R(4)
        J(K)=R(1)
        KJ(14)=K
        K = K + 1
        IF(EX(2).NE.0)THEN
          I(K)=R(1)
          J(K)=EX(2)
          KJ(15)= K
          K = K + 1
        END IF
          CALL ALROUND4(X,3)
c       Create arcs from AP(4)
        IF(AP(4).NE.0)THEN
          IF(EX(1).NE.0)THEN
            I(K)=AP(4)
            J(K)=EX(1)
            KJ(16)=K
            K=K+1
          END IF
          IF(EX(2).NE.0)THEN
            I(K)=AP(4)
            J(K)=EX(2)
            KJ(17)=K
            K=K+1
          END IF
```

```
          I(K)=AP(4)
          J(K)=R(1)
          KJ(18)=K
          K = K + 1
        END IF
        I(K)=R(1)
        J(K)=R(2)
        KJ(19)=K
        K = K + 1
        IF(EX(3).NE.0)THEN
          I(K)=R(2)
          J(K)=EX(3)
          KJ(20)=K
          K=K+1
        END IF
          CALL ALROUND4(X,4)
        ARCS=K-1
        CALL CON4ARC5
        RETURN
        END
C       Of subroutine ROUND4(X)
C
        SUBROUTINE TJUNCTN(X)
C
C       Creates up to six arcs for possible movements at T-junction, X.
c
        INTEGER AP,EX,R,FEED,X,A,B,INJ,XION,UB,DB,I,J,NC,CR,CW,WT,ARCS,
     1  FIRSTAB,FIRSTBA,TW,DA,UA,XIONS,ZONESIN,ZONES
        COMMON INJ(300,8),XION(300),UB(300),DB(300),I(1200),J(1200),
     1  NC(1200),CR(1200,8),NODES,ARCS,ZONES,A(300),B(300),FIRSTAB(300),
     1  LASTAB(300),FIRSTBA(300),LASTBA(300),KJ(20),AP(4),EX(4),NL(300),
     1  CW(1200,8),JT(300),TW(300),DA(200),UA(200),R(5),
     1  FEED(50),LINKS,XIONS,ZONESIN
C       Create approach and exit vertices
        DO 2 N=1,3
           AP(N)=0
           EX(N)=0
    2   CONTINUE
        DO 10 JL=1,3
        IF(B(INJ(X,JL)).EQ.XION(X))THEN
          AP(JL)=UB(INJ(X,JL))
          EX(JL)=DB(INJ(X,JL))
        ELSE IF(A(INJ(X,JL)).EQ.XION(X))THEN
          AP(JL)=DB(INJ(X,JL))
          EX(JL)=UB(INJ(X,JL))
        ELSE
        PRINT*,'Mistake in LINKS.DAT file for junction  ',XION(X)
        STOP 'Check your LINKS.DAT file and start again'
        END IF
   10   CONTINUE
        DO 20 K=1,6
           KJ(K)=0
   20   CONTINUE
        K = ARCS + 1
c       Create arcs from AP(1)
        IF(AP(1).NE.0)THEN
          IF(EX(2).NE.0)THEN
```

```
            I(K)=AP(1)
            J(K)=EX(2)
            KJ(1)=K
            K=K+1
          ELSE
            KJ(1)=0
          END IF
          IF(EX(3).NE.0)THEN
            I(K)=AP(1)
            J(K)=EX(3)
            KJ(2)=K
            K=K+1
          ELSE
            KJ(2)=0
          END IF
          CALL ARC3LINK(X,1)
        END IF
c       Create arcs from AP(2)
        IF(AP(2).NE.0)THEN
          IF(EX(3).NE.0)THEN
            I(K)=AP(2)
            J(K)=EX(3)
            KJ(3)=K
            K=K+1
          ELSE
            KJ(3)=0
          END IF
          IF(EX(1).NE.0)THEN
            I(K)=AP(2)
            J(K)=EX(1)
            KJ(4)=K
            K=K+1
          ELSE
            KJ(4)=0
          END IF
          CALL ARC3LINK(X,2)
        END IF
c       Create arcs from AP(3)
        IF(AP(3).NE.0)THEN
          IF(EX(1).NE.0)THEN
            I(K)=AP(3)
            J(K)=EX(1)
            KJ(5)=K
            K=K+1
          ELSE
            KJ(5)=0
          END IF
          IF(EX(2).NE.0)THEN
            I(K)=AP(3)
            J(K)=EX(2)
            KJ(6)=K
            K=K+1
          ELSE
            KJ(6)=0
          END IF
          CALL ARC3LINK(X,3)
        END IF
```

```
      ARCS=K-1
      IF(JT(X).EQ.1) THEN
        CALL CON3ARC1
      ELSE IF(JT(X).EQ.6)THEN
        CALL CON3RF
      END IF
      RETURN
      END
C     Of subroutine TJUNCTN(X)
C
      SUBROUTINE XROADS(X)
C
C     Creates up to twelve arcs for possible movements at crossroads, X.
c
      INTEGER AP,EX,R,FEED,X,A,B,INJ,XION,UB,DB,I,J,NC,CR,CW,WT,ARCS,
     1 FIRSTAB,FIRSTBA,TW,DA,UA,XIONS,ZONESIN,ZONES
      COMMON INJ(300,8),XION(300),UB(300),DB(300),I(1200),J(1200),
     1 NC(1200),CR(1200,8),NODES,ARCS,ZONES,A(300),B(300),FIRSTAB(300),
     1 LASTAB(300),FIRSTBA(300),LASTBA(300),KJ(20),AP(4),EX(4),NL(300),
     1 CW(1200,8),JT(300),TW(300),DA(200),UA(200),R(5),
     1 FEED(50),LINKS,XIONS,ZONESIN
C     Create approach and exit vertices
      DO 2 N=1,4
         AP(N)=0
         EX(N)=0
   2  CONTINUE
      DO 10 JL=1,4
      IF(B(INJ(X,JL)).EQ.XION(X))THEN
        AP(JL)=UB(INJ(X,JL))
        EX(JL)=DB(INJ(X,JL))
      ELSE IF(A(INJ(X,JL)).EQ.XION(X))THEN
        AP(JL)=DB(INJ(X,JL))
        EX(JL)=UB(INJ(X,JL))
      ELSE
      PRINT*,'Mistake in LINKS.DAT file for junction  ',XION(X)
      STOP 'Check your LINKS.DAT file and start again'
      END IF
  10  CONTINUE
      DO 20 K=1,12
         KJ(K)=0
  20  CONTINUE
      K=ARCS+1
c     Create arcs from AP(1)
      IF(AP(1).NE.0)THEN
         IF(EX(2).NE.0)THEN
           I(K)=AP(1)
           J(K)=EX(2)
           KJ(1)=K
           K=K+1
         ELSE
           KJ(1)=0
         END IF
         IF(EX(3).NE.0)THEN
           I(K)=AP(1)
           J(K)=EX(3)
           KJ(2)=K
           K=K+1
```

```
            ELSE
              KJ(2)=0
            END IF
            IF(EX(4).NE.0)THEN
              I(K)=AP(1)
              J(K)=EX(4)
              KJ(3)=K
              K=K+1
            ELSE
              KJ(3)=0
            END IF
            CALL ARC4LINK(X,1)
          END IF
c         Create arcs from AP(2)
          IF(AP(2).NE.0)THEN
            IF(EX(3).NE.0)THEN
              I(K)=AP(2)
              J(K)=EX(3)
              KJ(4)=K
              K=K+1
            ELSE
              KJ(4)=0
            END IF
            IF(EX(4).NE.0)THEN
              I(K)=AP(2)
              J(K)=EX(4)
              KJ(5)=K
              K=K+1
            ELSE
              KJ(5)=0
            END IF
            IF(EX(1).NE.0)THEN
              I(K)=AP(2)
              J(K)=EX(1)
              KJ(6)=K
              K=K+1
            ELSE
              KJ(6)=0
            END IF
            KJ(4)=KJ(4)
            KJ(5)=KJ(5)
            KJ(6)=KJ(6)
            CALL ARC4LINK(X,2)
          END IF
C         Create arcs from AP(3)
          IF(AP(3).NE.0)THEN
            IF(EX(4).NE.0)THEN
              I(K)=AP(3)
              J(K)=EX(4)
              KJ(7)=K
              K=K+1
            ELSE
              KJ(7)=0
            END IF
            IF(EX(1).NE.0)THEN
              I(K)=AP(3)
              J(K)=EX(1)
```

```
            KJ(8)=K
            K=K+1
          ELSE
            KJ(8)=0
          END IF
          IF(EX(2).NE.0)THEN
            I(K)=AP(3)
            J(K)=EX(2)
            KJ(9)=K
            K=K+1
          ELSE
            KJ(9)=0
          END IF
          CALL ARC4LINK(X,3)
        END IF
C       Create arcs from AP(4)
        IF(AP(4).NE.0)THEN
          IF(EX(1).NE.0)THEN
            I(K)=AP(4)
            J(K)=EX(1)
            KJ(10)=K
            K=K+1
          ELSE
            KJ(10)=0
          END IF
          IF(EX(2).NE.0)THEN
            I(K)=AP(4)
            J(K)=EX(2)
            KJ(11)=K
            K=K+1
          ELSE
            KJ(11)=0
          END IF
          IF(EX(3).NE.0)THEN
            I(K)=AP(4)
            J(K)=EX(3)
            KJ(12)=K
            K=K+1
          ELSE
            KJ(12)=0
          END IF
          CALL ARC4LINK(X,4)
        END IF
        ARCS=K-1
        IF(JT(X).EQ.1) THEN
          CALL CON4ARC1
        ELSE IF(JT(X).EQ.2) THEN
          CALL CON4ARC2
        ELSE IF(JT(X).EQ.3) THEN
          CALL CON4ARC3
        ELSE IF(JT(X).EQ.6) THEN
          CALL CON4ARC6
        END IF
        RETURN
        END
C       Of subroutine XROADS(X)
```

```
      PROGRAM POLYSEND
C
C     1 September 89 Enhanced POLYSEND with OLDFLOW assignment.
c     Takes account of weights for
c     conflicts as given in every second record of CROSSFLO.DAT.
C     Also enhanced to allow user to specify a different order of
c     loading from numeric order of zones either by entering an order
c     from the keyboard or giving a seed to find a random order.
C     Four start-up assignments offered -
c     INIT = 0 Start with an empty network and load
c     it so as to avoid as much conflict as possible with trips
c     already assigned to obtain a LOADFLOW assignment
c     INIT = 1 Assign routes which use a minimum number of arcs ,
c     and therefore pass a minimum number of junctions to obtain
c     the DARTFLOW assignment.
c     INIT = 2 Set costs on arcs to equal the number of conflicting
c     arcs and find minimum cost routes for DASHFLOW assignment.
c     INIT = 3 Reads arc flows from file STARTFLO.DAT which is in the
c     same format as FARCFLO.DAT. This is the OLDFLOW assignment.
c     The program reassigns the flows from up to 50 different origins,
c     on a traffic circulation network consisting of up to 1100 arcs
c     and up to 400 nodes.
c     It does MAXITRN complete iterations or terminates when there
c     will be no further changes.
c     It reports changes after each reassignment (run of Kilter).
c
      INTEGER ARCS,ZONES,ORIGIN,FKA,CR,TRIPS,CW
      REAL LO
C     KA is the number of artificial arcs (kiltarcs) which have to be
c     added to the network to carry flow away from the destinations
c     to the origin. FKA is the arc no.of the first
c     kiltarc and LKA is that of the last kiltarc.
      COMMON I(1100),J(1100),LO(1100),FLOW(50,1100),
     1 TFLOW(1100),TCOST(1100),PI(400),INFEAS,NA(400),NODES,ARCS,ZONES,
     1 INIT,MAXITRN,ITRP,ITRD,TOTFLOW(1100),TC,
     1 OTHERFLO(1100),KA,FKA,LKA,TOTCRASH,BASECOST(50),NCHANGE(50),
     1 NC(1100),CR(1100,7),TRIPS(50,50),
     1 ORIGIN(50),MCHANGE,CW(1100,7)
 8001 FORMAT (2I5,3F10.0)
 9003 FORMAT('INFEASIBLE PROBLEM')
      WRITE(*,*)' Enter method for start-up assignment'
      WRITE(*,*)' 1 for LOADFLOW'
      WRITE(*,*)' 2 for DARTFLOW'
      WRITE(*,*)' 3 for DASHFLOW'
      WRITE(*,*)' 4 for OLDFLOW'
      READ(*,*)INIT
      WRITE(*,*)' How many iterations do you want?'
      READ(*,*)MAXITRN
      OPEN(UNIT=8, FILE = 'ARCS.DAT',STATUS='OLD')
      READ(8,*) NODES,ARCS,ZONES
      CLOSE(UNIT=8,STATUS='KEEP')
      CALL LOADING
c     Reads order of loading from screen
      OPEN(UNIT=8, FILE = 'ARCS.DAT',STATUS='OLD')
      OPEN(UNIT=9, FILE = 'TRIPS.DAT',STATUS='OLD')
      OPEN(UNIT=10,FILE = 'CROSSFLO.DAT',STATUS='OLD')
      OPEN(UNIT=11,FILE = 'STARCFLO.DAT',STATUS='NEW')
```

```
      OPEN(UNIT=12,FILE = 'FARCFLOW.DAT',STATUS='NEW')
      READ(8,*) NODES,ARCS,ZONES
c     File reopened so repeat reading of first record.
c     These set limits for do loops etc.
c     Now read CROSSFLO.DAT
      DO 2 K=1,ARCS
      READ(10,*,END=2)NC(K),(CR(K,KC),KC=1,NC(K))
      READ(10,*,END=2)(CW(K,KC),KC=1,NC(K))
    2 CONTINUE
      CLOSE(UNIT=10,STATUS='KEEP')
      IF(NODES.LE.0) GOTO 100
      KA = ZONES-1
      FKA = ARCS + 1
      LKA = ARCS + KA
      DO 3 KAR=FKA,LKA
C     Cost on the kilter arcs will always be zero
        TCOST(KAR)=0.0
    3 CONTINUE
      CALL NETWORK
c     Reads rest of ARCS.DAT
      CALL ASSIGN
c     Reads TRIPS.DAT
      DO 4 K=1,LKA
        DO 6 NO=1,ZONES
          FLOW(NO,K)=0.0
    6   CONTINUE
        TOTFLOW(K)=0.0
    4 CONTINUE
C     Now proceed according to value of INIT
      IF(INIT.EQ.0)THEN
        CALL LOADFLOW
      ELSE IF(INIT.EQ.1)THEN
        CALL DARTFLOW
      ELSE IF(INIT.EQ.2)THEN
        CALL DASHFLOW
      ELSE IF(INIT.EQ.3)THEN
        CALL OLDFLOW
      ELSE
        STOP' INIT not given as 1,2,3, or 4 - start again'
      END IF
      DO 10 NI=1,MAXITRN
c     The program spends most of its time in this loop
      CALL ITERATE(NI)
   10 CONTINUE
      CALL FINAL(NI)
c     To print the final assignment
      IF(MCHANGE.EQ.0)THEN
      STOP 'Program terminated by no further changes'
      END IF
      IF(INFEAS.EQ.1) GOTO 999
  100 STOP 'Maximum number of iterations now complete'
  999 STOP 'Infeasible problem'
      END
C
      SUBROUTINE ARCCOSTS
C     Uses data of conflicting flows from the array OTHERFLO(K)
C     to calculate costs on arcs and creates the array TCOST(K).
```

```
c
      INTEGER ARCS,ZONES,ORIGIN,FKA,CR,TRIPS,CW
      REAL LO
      COMMON I(1100),J(1100),LO(1100),FLOW(50,1100),
     1 TFLOW(1100),TCOST(1100),PI(400),INFEAS,NA(400),NODES,ARCS,ZONES,
     1 INIT,MAXITRN,ITRP,ITRD,TOTFLOW(1100),TC,
     1 OTHERFLO(1100),KA,FKA,LKA,TOTCRASH,BASECOST(50),NCHANGE(50),
     1 NC(1100),CR(1100,7),TRIPS(50,50),
     1 ORIGIN(50),MCHANGE,CW(1100,7)
      DO 4 K=1,ARCS
         TCOST(K)=0.0
         DO 6 KC=1,NC(K)
            TCOST(K)=TCOST(K) + OTHERFLO(CR(K,KC))*CW(K,KC)
    6    CONTINUE
    4 CONTINUE
      RETURN
      END
C     Of subroutine ARCCOSTS
c
      SUBROUTINE ASSIGN
C     Reads data from TRIPS.DAT
      INTEGER ARCS,ZONES,ORIGIN,FKA,CR,CW,TRIPS
      REAL LO
      COMMON I(1100),J(1100),LO(1100),FLOW(50,1100),
     1 TFLOW(1100),TCOST(1100),PI(400),INFEAS,NA(400),NODES,ARCS,ZONES,
     1 INIT,MAXITRN,ITRP,ITRD,TOTFLOW(1100),TC,
     1 OTHERFLO(1100),KA,FKA,LKA,TOTCRASH,BASECOST(50),NCHANGE(50),
     1 NC(1100),CR(1100,7),TRIPS(50,50),
     1 ORIGIN(50),MCHANGE,CW(1100,7)
 9005 FORMAT(13I6)
      DO 10 NO=1,ZONES
         DO 20 ND=1,ZONES,12
            READ(9,9005)NOR,(TRIPS(NO,NT),NT=ND,ND+11)
   20    CONTINUE
   10 CONTINUE
      CLOSE(UNIT=9,STATUS='KEEP')
      RETURN
      END
C     of subroutine ASSIGN
C
      SUBROUTINE BUILDUP
C
C     Builds up the costs on arcs
      INTEGER ARCS,ZONES,ORIGIN,FKA,CR,CW,TRIPS
      REAL LO
      COMMON I(1100),J(1100),LO(1100),FLOW(50,1100),
     1 TFLOW(1100),TCOST(1100),PI(400),INFEAS,NA(400),NODES,ARCS,ZONES,
     1 INIT,MAXITRN,ITRP,ITRD,TOTFLOW(1100),TC,
     1 OTHERFLO(1100),KA,FKA,LKA,TOTCRASH,BASECOST(50),NCHANGE(50),
     1 NC(1100),CR(1100,7),TRIPS(50,50),
     1 ORIGIN(50),MCHANGE,CW(1100,7)
      DO 4 K=1,ARCS
         TCOST(K)=0.0
         DO 6 KC=1,NC(K)
            TCOST(K)=TCOST(K) + TOTFLOW(CR(K,KC))
    6    CONTINUE
    4 CONTINUE
```

```
      RETURN
      END
C     Of subroutine BUILDUP
c
      SUBROUTINE DARTFLOW
C
C     Assigns flows to minimise the total
c     number of arcs used, computes BASECOST(NO), the total number
c     of conflicts encountered by each initial flow, and TOTCRASH,
c     the total number of conflicts in the initial flow pattern.
c
      INTEGER ARCS,ZONES,ORIGIN,FKA,CR,CW,TRIPS
      COMMON I(1100),J(1100),LO(1100),FLOW(50,1100),
     1 TFLOW(1100),TCOST(1100),PI(400),INFEAS,NA(400),NODES,ARCS,ZONES,
     1 INIT,MAXITRN,ITRP,ITRD,TOTFLOW(1100),TC,
     1 OTHERFLO(1100),KA,FKA,LKA,TOTCRASH,BASECOST(50),NCHANGE(50),
     1 NC(1100),CR(1100,7),TRIPS(50,50),
     1 ORIGIN(50),MCHANGE,CW(1100,7)
      DO 2 K=1,ARCS
      TCOST(K)=1.0
   2  CONTINUE
      DO 36 NORIG=1,ZONES
        NOD=ORIGIN(NORIG)
        CALL KILTARCS(NOD)
C     Sets up Kilter arcs
        CALL KILTER
        DO 32 K=1,ARCS
          FLOW(NOD,K)=TFLOW(K)
  32    CONTINUE
  36  CONTINUE
      CALL TOTALFLO
      DO 38 NOD=1,ZONES
      CALL ZONECOST(NOD)
  38  CONTINUE
      CALL SUMCRASH
C     Above is the means of computing TOTCRASH
 9016 FORMAT(/'THE DARTFLOW NUMBER OF POSSIBLE CRASHES IS ',F18.2)
      WRITE(7,9016)TOTCRASH
      CALL VINES(0)
C     To print DARTFLOW assignment
      CALL SUMMARY
  30  CONTINUE
      RETURN
      END
C     Of subroutine DARTFLOW
c
      SUBROUTINE DASHFLOW
C
C     Assigns flows to minimise the total number of arcs crossed
c     or merged into, computes BASECOST(NO), the total number
c     of conflicts encountered by each initial flow, and TOTCRASH,
c     the total number of conflicts in the initial flow pattern.
c
      INTEGER ARCS,ZONES,ORIGIN,FKA,CR,CW,TRIPS
      COMMON I(1100),J(1100),LO(1100),FLOW(50,1100),
     1 TFLOW(1100),TCOST(1100),PI(400),INFEAS,NA(400),NODES,ARCS,ZONES,
     1 INIT,MAXITRN,ITRP,ITRD,TOTFLOW(1100),TC,
```

```
     1  OTHERFLO(1100),KA,FKA,LKA,TOTCRASH,BASECOST(50),NCHANGE(50),
     1  NC(1100),CR(1100,7),TRIPS(50,50),
     1  ORIGIN(50),MCHANGE,CW(1100,7)
        DO 2 K=1,ARCS
        TCOST(K) = NC(K)
    2   CONTINUE
        DO 36 NORIG=1,ZONES
           NOD=ORIGIN(NORIG)
           CALL KILTARCS(NOD)
C       Sets up Kilter arcs
           CALL KILTER
           DO 32 K=1,ARCS
              FLOW(NOD,K)=TFLOW(K)
   32      CONTINUE
   36   CONTINUE
        CALL TOTALFLO
        DO 38 NOD=1,ZONES
        CALL ZONECOST(NOD)
   38   CONTINUE
        CALL SUMCRASH
C       Above is the means of computing TOTCRASH
 9016   FORMAT(/'THE DASHFLOW NUMBER OF POSSIBLE CRASHES IS ',F18.2)
        WRITE(7,9016)TOTCRASH
        CALL VINES(0)
C       To print DASHFLOW assignment
        CALL SUMMARY
   30   CONTINUE
        RETURN
        END
C       Of subroutine DASHFLOW
c
        SUBROUTINE FINAL(NI)
C
C       Arranges printing of final assignment
        INTEGER ARCS,ZONES,ORIGIN,FKA,CR,CW,TRIPS
        REAL LO
        COMMON I(1100),J(1100),LO(1100),FLOW(50,1100),
     1  TFLOW(1100),TCOST(1100),PI(400),INFEAS,NA(400),NODES,ARCS,ZONES,
     1  INIT,MAXITRN,ITRP,ITRD,TOTFLOW(1100),TC,
     1  OTHERFLO(1100),KA,FKA,LKA,TOTCRASH,BASECOST(50),NCHANGE(50),
     1  NC(1100),CR(1100,7),TRIPS(50,50),
     1  ORIGIN(50),MCHANGE,CW(1100,7)
        CALL VINES(NI)
C       To print final flow
        CALL SUMMARY
        DO 42 NOD=1,ZONES
           CALL ZONECOST(NOD)
   42   CONTINUE
C       BASECOST(NOD) is now uptodate
        CALL SUMCRASH
        RETURN
        END
C       Of subroutine FINAL
C
        SUBROUTINE GETREADY(NOD)
C
C       Computes OTHERFLO(K) appropiate to zone NOD
```

```
      INTEGER ARCS,ZONES,ORIGIN,FKA,CR,CW,TRIPS
      REAL LO
      COMMON I(1100),J(1100),LO(1100),FLOW(50,1100),
     1 TFLOW(1100),TCOST(1100),PI(400),INFEAS,NA(400),NODES,ARCS,ZONES,
     1 INIT,MAXITRN,ITRP,ITRD,TOTFLOW(1100),TC,
     1 OTHERFLO(1100),KA,FKA,LKA,TOTCRASH,BASECOST(50),NCHANGE(50),
     1 NC(1100),CR(1100,7),TRIPS(50,50),
     1 ORIGIN(50),MCHANGE,CW(1100,7)
      DO 2 K=1,ARCS
        OTHERFLO(K) = TOTFLOW(K) - FLOW(NOD,K)
    2 CONTINUE
      CALL ARCCOSTS(NOD)
      RETURN
      END
C     Of subroutine GETREADY(NOD)
C
      SUBROUTINE ITERATE(NI)
c     Reassigns successive flows with an update of TOTFLOW(K)
C     and TCOST(K) between each reassignment.
      INTEGER ARCS,ZONES,ORIGIN,FKA,CR,CW,TRIPS
      REAL LO
      COMMON I(1100),J(1100),LO(1100),FLOW(50,1100),
     1 TFLOW(1100),TCOST(1100),PI(400),INFEAS,NA(400),NODES,ARCS,ZONES,
     1 INIT,MAXITRN,ITRP,ITRD,TOTFLOW(1100),TC,
     1 OTHERFLO(1100),KA,FKA,LKA,TOTCRASH,BASECOST(50),NCHANGE(50),
     1 NC(1100),CR(1100,7),TRIPS(50,50),
     1 ORIGIN(50),MCHANGE,CW(1100,7)
      DO 30 NORIG = 1,ZONES
        NOD = ORIGIN(NORIG)
C     So if the 1st origin is zone 3 NOD = 3
        IF(NI.NE.1)THEN
          MCHANGE=MCHANGE - NCHANGE(NOD)
C         MCHANGE now refers to the last (ZONES-1) changes
        END IF
        CALL KILTARCS(NOD)
        CALL GETREADY(NOD)
c     TCOST(K) is now appropriate to flow from zone NOD
        CALL KILTER
        NCHANGE(NOD)=0
        DO 32 K=1,ARCS
          IF(TFLOW(K).NE.FLOW(NOD,K))NCHANGE(NOD)=NCHANGE(NOD)+1
            FLOW(NOD,K)=TFLOW(K)
            TOTFLOW(K)=OTHERFLO(K) + TFLOW(K)
   32   CONTINUE
C       TOTFLOW(K), FLOW(NOD,K) and NCHANGE(NOD) are now uptodate
        MCHANGE = MCHANGE + NCHANGE(NOD)
c         MCHANGE now uptodate
          IF((MCHANGE.EQ.0).AND.(NI.NE.1))GOTO 37
c       Gets out of this loop and ITERATE
        MX=NI-1
 9004   FORMAT (/' NEW VINE HAD',I5,' CHANGES IN FLOW VALUES,')
        WRITE (5,9004) NCHANGE(NOD)
 9032   FORMAT('AFTER ',I3,' COMPLETE ITERATIONS')
 9033   FORMAT('PLUS ',I3,' RUNS OF KILTER')
        WRITE(5,9032)MX
        WRITE(5,9033)NORIG
   39   CONTINUE
```

```
   30   CONTINUE
        IF(NI.NE.1)GOTO 40
        MCHANGE=0
        DO 28 N=1,ZONES
           MCHANGE=MCHANGE + NCHANGE(N)
   28   CONTINUE
        IF(MCHANGE.NE.0) GOTO 40
 9040   FORMAT(/'THERE WILL BE NO FURTHER CHANGES')
   37   WRITE(7,9040)
        WRITE(5,9040)
        GOTO 44
   40   CONTINUE
C       After each iteration give total of crashes
        DO 42 NOD=1,ZONES
           CALL ZONECOST(NOD)
   42   CONTINUE
C       BASECOST(NOD) is now uptodate
        CALL SUMCRASH
   44   RETURN
        END
C       Of subroutine ITERATE(NI)
C
        SUBROUTINE KILTARCS(NOD)
C       Sets up the artificial arcs with positive lower bounds equal
c       to the trips demanded.
        INTEGER ARCS,ZONES,ORIGIN,FKA,CR,CW,TRIPS
        REAL LO
        COMMON I(1100),J(1100),LO(1100),FLOW(50,1100),
     1  TFLOW(1100),TCOST(1100),PI(400),INFEAS,NA(400),NODES,ARCS,ZONES,
     1  INIT,MAXITRN,ITRP,ITRD,TOTFLOW(1100),TC,
     1  OTHERFLO(1100),KA,FKA,LKA,TOTCRASH,BASECOST(50),NCHANGE(50),
     1  NC(1100),CR(1100,7),TRIPS(50,50),
     1  ORIGIN(50),MCHANGE,CW(1100,7)
           DO 31 KAR=FKA,LKA
              J(KAR)=2*NOD-1
   31      CONTINUE
           DO 33 KR=1,NOD-1
              I(FKA+KR-1)=2*KR
              LO(FKA+KR-1)=TRIPS(NOD,KR)
   33      CONTINUE
           DO 34 KR=NOD+1,ZONES
              I(FKA+KR-2)=2*KR
              LO(FKA+KR-2)=TRIPS(NOD,KR)
   34      CONTINUE
        RETURN
        END
C       Of subroutine KILTARCS(NOD)
C
        SUBROUTINE KILTER
C       Assigns flows which conserves flow through nodes while
c       minimising the total cost of those flows.
c
        INTEGER ARCS,A,AOK,AF,AD,SRC,SNK,ZONES,ORIGIN,FKA,CR,CW,TRIPS
        REAL INF,B,LO
        COMMON I(1100),J(1100),LO(1100),FLOW(50,1100),
     1  TFLOW(1100),TCOST(1100),PI(400),INFEAS,NA(400),NODES,ARCS,ZONES,
     1  INIT,MAXITRN,ITRP,ITRD,TOTFLOW(1100),TC,
```

```
     1  OTHERFLO(1100),KA,FKA,LKA,TOTCRASH,BASECOST(50),NCHANGE(50),
     1  NC(1100),CR(1100,7),TRIPS(50,50),
     1  ORIGIN(50),MCHANGE,CW(1100,7)
  8000   FORMAT (/' DUAL VALUES OF UNLABELLED VERTICES MUST MOVE BY '
     1  ,F10.5)
  8001   FORMAT (/'BREAKTHROUGH WITH FLOW ',F10.5)
         ITRP = 0
         ITRD = 0
         DO 9 K=1,LKA
         TFLOW(K) = 0.0
         IK = I(K)
         JK = J(K)
         PI(IK) =0.0
         NA(IK) = 0
    9    CONTINUE
         INF=-1.0
         DO 220 AOK=FKA,LKA
C        Looking for an O-O-K arc
         IK=I(AOK)
         JK=J(AOK)
         COK=TCOST(AOK)+PI(IK)-PI(JK)
   20    IF((TFLOW(AOK).LT.LO(AOK)).OR.(COK.LT.0.0)) GOTO 30
         GOTO 220
   30    SRC=JK
         SNK=IK
         NA(SRC)=AOK
        IF(NA(SNK).NE.0)GOTO 150
C        We already have node SNK in the vine of paths.
   70    LAB = 0
         DO 100 AF = 1,LKA
         IA=I(AF)
         JA=J(AF)
         IF(((NA(IA).EQ.0).AND.(NA(JA).EQ.0)).OR.((NA(IA).NE.0).AND.
     1  (NA(JA).NE.0))) GOTO 100
c        This arc,AF, is not eligible for the vine.
         C=TCOST(AF)+PI(IA)-PI(JA)
         IF(NA(IA).EQ.0) GOTO 80
         IF(C.GT.0.0)GOTO 100
         NA(JA)=AF
c        The start node is labelled now label the finish node.
         GO TO 90
   80    IF((TFLOW(AF).LE.LO(AF)).OR.(C.LT.0.0)) GO TO 100
   90    LAB = 1
c        Some labelling has happened
         IF(NA(SNK).NE.0) GO TO 150
c        We have a flow augmenting circuit
  100    CONTINUE
         IF(LAB.NE.0.) GOTO 70
c        We might be able to do some more labelling
         DEL=INF
c        Because we havn't got a flow augmenting path
         DO 110 AD = 1,LKA
         IA=I(AD)
         JA=J(AD)
         IF(((NA(IA).EQ.0).AND.(NA(JA).EQ.0)).OR.((NA(IA).NE.0).AND.
     1  (NA(JA).NE.0))) GO TO 110
        C=TCOST(AD)+PI(IA)-PI(JA)
```

```
          IF(NA(JA).EQ.0) DEL=RMIN(DEL,C)
          IF(NA(JA).NE.0.AND.TFLOW(AD).GT.LO(AD)) DEL=RMIN(DEL,-C)
  110     CONTINUE
          IF(DEL.NE.INF) GO TO 130
          IF(TFLOW(AOK).EQ.LO(AOK)) GO TO 120
          GO TO 230
  120     DEL=ABS(COK)
  130     ITRD = ITRD + 1
          GO TO 135
  135     DO 140 N = 1,NODES
C         Increasing the dual values of unlabelled nodes
          IF(NA(N).EQ.0) PI(N)=PI(N)+DEL
  140     CONTINUE
          IF((DEL.EQ.ABS(COK)).AND.(TFLOW(AOK).GE.LO(AOK)))
     1    GO TO 220
          IK=I(AOK)
          JK=J(AOK)
          COK=TCOST(AOK)+PI(IK)-PI(JK)
          GO TO 70
C         Now try labelling again
  150     EPS=LO(AOK)
          N=SRC
C         Starting with arc,AOK, we increase the flow on it
C         and trace back increasing all flows until we reach,SRC,
C         the finish node of arc ,AOK.
  190     A=NA(N)
          M=I(A)
          TFLOW(A) = TFLOW(A) + EPS
  210     N = M
          IF(N.NE.SRC) GOTO 190
          ITRP = ITRP + 1
          GOTO 20
  220     CONTINUE
          INFEAS=-1
          T = 10000.0
          DO 250 K = 1,NODES
          IF (PI(K) .LT. T) T = PI(K)
  250     CONTINUE
          DO 260 K = 1,NODES
  260     PI(K) = PI(K) - T
  260     CONTINUE
          RETURN
  230     INFEAS =1
          RETURN
          END
C         Of subroutine KILTER
C
          SUBROUTINE LOADFLOW
C         Assigns flows, starting with empty network, & avoiding conflict.
          INTEGER ARCS,ZONES,ORIGIN,FKA,CR,CW,TRIPS
          COMMON I(1100),J(1100),LO(1100),FLOW(50,1100),
     1    TFLOW(1100),TCOST(1100),PI(400),INFEAS,NA(400),NODES,ARCS,ZONES,
     1    INIT,MAXITRN,ITRP,ITRD,TOTFLOW(1100),TC,
     1    OTHERFLO(1100),KA,FKA,LKA,TOTCRASH,BASECOST(50),NCHANGE(50),
     1    NC(1100),CR(1100,7),TRIPS(50,50),
     1    ORIGIN(50),MCHANGE,CW(1100,7)
          DO 30 NORIG = 1,ZONES
```

```
            NOD = ORIGIN(NORIG)
C           So if e.g. the 1st origin is zone 3 NOD = 3
            CALL KILTARCS(NOD)
            IF(NORIG.NE.1)THEN
c             Some updating is called for.
              DO 32 K=1,ARCS
                 TOTFLOW(K)=TOTFLOW(K) + TFLOW(K)
C                This updates TOTFLOW(K)
   32         CONTINUE
              CALL BUILDUP
c             Builds up the costs on arcs using the increased TOTFLOW(K)
            ELSE
              DO 34 K=1,ARCS
                 TCOST(K)=0.0
   34         CONTINUE
            END IF
c           TCOST(K) has been updated using the current TOTFLOW(K)
            CALL KILTER
            DO 36 K=1,ARCS
               FLOW(NOD,K)=TFLOW(K)
   36       CONTINUE
C           This updates FLOW(NOD,K)
   30     CONTINUE
          CALL VINES(0)
C         To print LOADFLOW assignment
          CALL SUMMARY
C         After each iteration give total of crashes
          DO 42 NOD=1,ZONES
          CALL ZONECOST(NOD)
   42     CONTINUE
C         BASECOST(NOD) is now uptodate
          CALL SUMCRASH
          RETURN
          END
C         Of subroutine LOADFLOW
C
          SUBROUTINE LOADING
C         Asks user if he wants an order of loading different from the
c         numeric order of zones.
          INTEGER ARCS,ZONES,FKA,CR,CW,ORIGIN,ORDER
          REAL LO
          COMMON I(1100),J(1100),LO(1100),FLOW(50,1100),
     1    TFLOW(1100),TCOST(1100),PI(400),INFEAS,NA(400),NODES,ARCS,ZONES,
     1    INIT,MAXITRN,ITRP,ITRD,TOTFLOW(1100),TC,
     1    OTHERFLO(1100),KA,FKA,LKA,TOTCRASH,BASECOST(50),NCHANGE(50),
     1    NC(1100),CR(1100,7),TRIPS(50,50),
     1    ORIGIN(50),MCHANGE,CW(1100,7)
          WRITE (*,*) 'Do you want to load other than in numeric order?'
          WRITE (*,*) 'Enter'
          WRITE (*,*) '0 for numeric order,'
          WRITE (*,*) '1 for user specified order, '
          WRITE (*,*) '2 for random, user to supply seed. '
          READ (*,*)ORDER
          IF(ORDER.EQ.0)THEN
          DO 10 N=1,ZONES
             ORIGIN(N)=N
    10    CONTINUE
```

```
      OPEN(UNIT=7, FILE = 'SUMMARY.RPT',STATUS='NEW')
      ELSE IF(ORDER.EQ.1)THEN
      WRITE (*,*) 'Enter order of loading on separate lines'
      DO 20 N=1,ZONES
         READ(*,*)ORIGIN(N)
   20 CONTINUE
      OPEN(UNIT=7, FILE = 'SUMMARY.RPT',STATUS='NEW')
      ELSE IF(ORDER.EQ.2)THEN
      WRITE (*,*) 'Enter an integer seed '
      READ(*,*)ISEED
      OPEN(UNIT=7, FILE = 'SUMMARY.RPT',STATUS='NEW')
 9005   FORMAT(/' This random order of loading was started with seed')
 9007   FORMAT(I5)
        WRITE(7,9005)
        WRITE(7,9007)ISEED
      DO 30 N=1,ZONES
         ORIGIN(N)=N
   30 CONTINUE
      DO 40 N=1,ZONES
         R = URAND(ISEED)
C     URAND is listed in 'Problem solving with Fortran77'
c     by B.D.Hahn publ Arnold 1987 page 142.
         NUM = INT(ZONES*R) + 1
         ITEMP = ORIGIN(NUM)
         ORIGIN(NUM) = ORIGIN(N)
         ORIGIN(N) = ITEMP
   40 CONTINUE
      ELSE
      WRITE (*,*) 'Not a valid entry - start again'
      END IF
 9000 FORMAT(20I3)
 9010 FORMAT(' The order of loading in this assignment is')
      WRITE(7,9010)
      DO 42 K=1,ZONES,20
         WRITE(7,9000)(ORIGIN(KL),KL=K,K+19)
   42 CONTINUE
      RETURN
      END
C     Of subroutine LOADING
C
      FUNCTION URAND(IY)
      INTEGER IA,IC,ITWO,M2,M,MIC
      DOUBLE PRECISION HALFM
      REAL S
      DATA M2/ 0/, ITWO/ 2/
      IF(M2.EQ.0)THEN
        M=1
        M2 = 16384
        HALFM = M2
        IA = 8*INT(HALFM*ATAN(1.D0)/8.D0) + 5
        IC = 2*INT(HALFM*(0.5D0 - SQRT(3.D0)/6.D0)) + 1
        MIC = (M2 -IC) + M2
        S = 0.5/HALFM
      END IF
      IY = IY*IA
      IY = IY + IC
      IY = MOD(IY,32768)
```

```
      URAND = FLOAT(IY)*S
      RETURN
      END
c     Of function URAND
C
      SUBROUTINE NETWORK
C     Reads rest of ARCS.DAT
      INTEGER ARCS,ZONES,ORIGIN,FKA,CR,CW,TRIPS
      REAL LO
      COMMON I(1100),J(1100),LO(1100),FLOW(50,1100),
     1 TFLOW(1100),TCOST(1100),PI(400),INFEAS,NA(400),NODES,ARCS,ZONES,
     1 INIT,MAXITRN,ITRP,ITRD,TOTFLOW(1100),TC,
     1 OTHERFLO(1100),KA,FKA,LKA,TOTCRASH,BASECOST(50),NCHANGE(50),
     1 NC(1100),CR(1100,7),TRIPS(50,50),
     1 ORIGIN(50),MCHANGE,CW(1100,7)
        DO 28 K=1,ARCS
          READ(8,*,END=28)KK,I(K),J(K)
          LO(K)=0.0
   28   CONTINUE
      CLOSE(UNIT=8,STATUS='KEEP')
      END
C     Of subroutine NETWORK
C
      SUBROUTINE OLDFLOW
C
C     Reads file STARTFLO.DAT to fill arrays FLOW(NOD,K) and
C     computes BASECOST(NO), the total costs of each initial
c     flow, and TOTCRASH, the total number of conflicts in the
c     initial flow pattern.
c
      INTEGER ARCS,ZONES,ORIGIN,FKA,CR,CW,TRIPS
      REAL LO
      COMMON I(1100),J(1100),LO(1100),FLOW(50,1100),
     1 TFLOW(1100),TCOST(1100),PI(400),INFEAS,NA(400),NODES,ARCS,ZONES,
     1 INIT,MAXITRN,ITRP,ITRD,TOTFLOW(1100),TC,
     1 OTHERFLO(1100),KA,FKA,LKA,TOTCRASH,BASECOST(50),NCHANGE(50),
     1 NC(1100),CR(1100,7),TRIPS(50,50),
     1 ORIGIN(50),MCHANGE,CW(1100,7)
      OPEN(UNIT=8, FILE = 'STARTFLO.DAT',STATUS='OLD')
9010  FORMAT(1X,2I4,5F10.2)
9020  FORMAT(1X,I4,5F10.2)
        DO 40 NO=1,ZONES
          DO 50 K=1,ARCS,5
            READ(8,9010)NN,KK,(FLOW(NO,KT),KT=K,K+4)
   50     CONTINUE
   40   CONTINUE
        DO 60 K=1,ARCS,5
        READ(8,9020)KK,(TOTFLOW(KT),KT=K,K+4)
   60   CONTINUE
      DO 38 NOD=1,ZONES
      CALL ZONECOST(NOD)
   38 CONTINUE
      CALL SUMCRASH
9016  FORMAT(/'THE OLDFLOW NUMBER OF POSSIBLE CRASHES IS ',F18.2)
      WRITE(7,9016)TOTCRASH
      CALL SUMMARY
      RETURN
```

```
      END
C     Of subroutine OLDFLOW
C
      SUBROUTINE SUMCRASH
c
      INTEGER ARCS,ZONES,ORIGIN,FKA,CR,CW,TRIPS
      REAL LO
      COMMON I(1100),J(1100),LO(1100),FLOW(50,1100),
     1 TFLOW(1100),TCOST(1100),PI(400),INFEAS,NA(400),NODES,ARCS,ZONES,
     1 INIT,MAXITRN,ITRP,ITRD,TOTFLOW(1100),TC,
     1 OTHERFLO(1100),KA,FKA,LKA,TOTCRASH,BASECOST(50),NCHANGE(50),
     1 NC(1100),CR(1100,7),TRIPS(50,50),
     1 ORIGIN(50),MCHANGE,CW(1100,7)
      BASHES = 0.0
      DO 20 NO = 1,ZONES
      BASHES = BASHES + BASECOST(NO)
   20 CONTINUE
      TOTCRASH = 0.5*BASHES
 9007 FORMAT(/' THE TOTAL NUMBER OF POSSIBLE CRASHES IS ',F18.1)
      WRITE(7,9007)TOTCRASH
      WRITE(5,9007)TOTCRASH
      RETURN
      END
C     Of subroutine SUMCRASH
C
      SUBROUTINE SUMMARY
      INTEGER ARCS,ZONES,ORIGIN,FKA,CR,CW,TRIPS
      COMMON I(1100),J(1100),LO(1100),FLOW(50,1100),
     1 TFLOW(1100),TCOST(1100),PI(400),INFEAS,NA(400),NODES,ARCS,ZONES,
     1 INIT,MAXITRN,ITRP,ITRD,TOTFLOW(1100),TC,
     1 OTHERFLO(1100),KA,FKA,LKA,TOTCRASH,BASECOST(50),NCHANGE(50),
     1 NC(1100),CR(1100,7),TRIPS(50,50),
     1 ORIGIN(50),MCHANGE,CW(1100,7)
 9015 FORMAT(/'THE FOLLOWING ARCS HAVE NOT BEEN USED')
      WRITE(7,9015)
 9016 FORMAT(1X,' ARC   I   J')
      WRITE(7,9016)
 9017 FORMAT(1X,3I4)
      DO 60 K=1,ARCS
      IF(TOTFLOW(K).EQ.0.0)WRITE(7,9017)K,I(K),J(K)
   60 CONTINUE
      RETURN
      END
C     Of subroutine SUMMARY
C
      SUBROUTINE TOTALFLO
C     Uses the 2-dimensional array FLOW(NO,K)
C     to create a 1-dimensional array TOTFLOW(K)
c
      INTEGER ARCS,ZONES,ORIGIN,FKA,CR,CW,TRIPS
      REAL LO
      COMMON I(1100),J(1100),LO(1100),FLOW(50,1100),
     1 TFLOW(1100),TCOST(1100),PI(400),INFEAS,NA(400),NODES,ARCS,ZONES,
     1 INIT,MAXITRN,ITRP,ITRD,TOTFLOW(1100),TC,
     1 OTHERFLO(1100),KA,FKA,LKA,TOTCRASH,BASECOST(50),NCHANGE(50),
     1 NC(1100),CR(1100,7),TRIPS(50,50),
     1 ORIGIN(50),MCHANGE,CW(1100,7)
```

```
      DO 12 K=1,ARCS
      TOTFLOW(K) =0.0
         DO 16 NO=1,ZONES
      TOTFLOW(K) = TOTFLOW(K) + FLOW(NO,K)
   16 CONTINUE
   12 CONTINUE
      RETURN
      END
C     Of subroutine TOTALFLO
c
      SUBROUTINE VINES(NI)
      INTEGER ARCS,ZONES,ORIGIN,FKA,CR,CW,TRIPS
      COMMON I(1100),J(1100),LO(1100),FLOW(50,1100),
     1 TFLOW(1100),TCOST(1100),PI(400),INFEAS,NA(400),NODES,ARCS,ZONES,
     1 INIT,MAXITRN,ITRP,ITRD,TOTFLOW(1100),TC,
     1 OTHERFLO(1100),KA,FKA,LKA,TOTCRASH,BASECOST(50),NCHANGE(50),
     1 NC(1100),CR(1100,7),TRIPS(50,50),
     1 ORIGIN(50),MCHANGE,CW(1100,7)
 9010 FORMAT(1X,2I4,5F10.2)
 9020 FORMAT(1X,I4,5F10.2)
      IF(NI.LE.1)THEN
        DO 40 NO=1,ZONES
          DO 50 K=1,ARCS,5
            WRITE (11,9010)NO,K,(FLOW(NO,KT),KT=K,K+4)
   50     CONTINUE
   40   CONTINUE
        DO 60 K=1,ARCS,5
        WRITE(11,9020)K,(TOTFLOW(KT),KT=K,K+4)
   60   CONTINUE
      ELSE
        DO 70 NO=1,ZONES
          DO 80 K=1,ARCS,5
            WRITE(12,9010)NO,K,(FLOW(NO,KT),KT=K,K+4)
   80     CONTINUE
   70   CONTINUE
        DO 90 K=1,ARCS,5
        WRITE(12,9020)K,(TOTFLOW(KT),KT=K,K+4)
   90   CONTINUE
      END IF
      RETURN
      END
C     Of subroutine VINES(NI)
C
      SUBROUTINE ZONECOST(NOD)
C     Calculates BASECOST(NOD), the total costs of flows
c     from zone NOD using FLOW(NOD,K) and TCOST(K).
C
      INTEGER ARCS,ZONES,ORIGIN,FKA,CR,CW,TRIPS
      REAL LO
      COMMON I(1100),J(1100),LO(1100),FLOW(50,1100),
     1 TFLOW(1100),TCOST(1100),PI(400),INFEAS,NA(400),NODES,ARCS,ZONES,
     1 INIT,MAXITRN,ITRP,ITRD,TOTFLOW(1100),TC,
     1 OTHERFLO(1100),KA,FKA,LKA,TOTCRASH,BASECOST(50),NCHANGE(50),
     1 NC(1100),CR(1100,7),TRIPS(50,50),
     1 ORIGIN(50),MCHANGE,CW(1100,7)
c     Now compute the total cost to the flow from zone NOD
      DO 20 K=1,ARCS
```

```
          OTHERFLO(K)=TOTFLOW(K)-FLOW(NOD,K)
   20     CONTINUE
          CALL ARCCOSTS(NOD)
C         Cost is now appropriate to flow from zone NOD
           BTC = 0.0
            DO 22 K = 1,ARCS
                 BTC = BTC +TCOST(K)*FLOW(NOD,K)
   22       CONTINUE
            BASECOST(NOD) = BTC
   21     CONTINUE
          RETURN
          END
C         Of subroutine ZONECOST(NOD)
C
          FUNCTION RMIN(X,Y)
          R = Y
          IF((X .LT. Y) .AND. (X .GE.0.0)) R = X
          RMIN = R
          RETURN
          END
C         Of function RMIN
C
          END
```

```
      PROGRAM POLYLINK
C     Starting with files, STARCFLO.DAT and FARCFLOW.DAT, which are
c     the output files from the program, POLYSEND, and the file,
c     ARCLINK.DAT, which is an output file from the program, POLYARCS,
C     POLYLINK creates files, SLINKFLO.DAT and FLINKFLO.DAT, which
c     give the total flows in each direction of the road network
c     according to the  START-UP and FINAL assignments produced by the
c     program, POLYSEND, the files, SLINKTRE.DAT and FLINKTRE.DAT,
C     which give the trees of flow from each origin in terms of links
c     on the road network and the files SLINKSUM.DAT and FLINKSUM.DAT
C     which table the links in bands according to amount of total flow.
C     The road network can have up to 300 links and 50 zones and the
c     traffic circulation network up to 1100 arcs.
      INTEGER ZONES,XIONS,ARCS,FIRSTAB,TW,FIRSTBA,A,B,FEED
      COMMON ZONES,XIONS,LINKS,ARCS,FIRSTAB(300),LASTAB(300),TW(300),
     1 FIRSTBA(300),LASTBA(300),TOTFLOW(1100),ABFLOW(300),BAFLOW(300),
     1 A(300),B(300),FLOW(50,1100)
      OPEN(UNIT=6,FILE ='LINKS.DAT',   STATUS='OLD')
      OPEN(UNIT=9,FILE ='ARCLINK.DAT', STATUS='OLD')
      OPEN(UNIT=12,FILE='STARCFLO.DAT',STATUS='OLD')
      OPEN(UNIT=13,FILE='FARCFLOW.DAT',STATUS='OLD')
      OPEN(UNIT=14,FILE='SLINKFLO.DAT',STATUS='NEW')
      OPEN(UNIT=15,FILE='FLINKFLO.DAT',STATUS='NEW')
      OPEN(UNIT=16,FILE='SLINKTRE.DAT',STATUS='NEW')
      OPEN(UNIT=17,FILE='FLINKTRE.DAT',STATUS='NEW')
      OPEN(UNIT=18,FILE='SLINKSUM.DAT',STATUS='NEW')
      OPEN(UNIT=19,FILE='FLINKSUM.DAT',STATUS='NEW')
C     Read ARCLINK.DAT and LINKS.DAT to set up arrays.
      READ(9,*)ZONES,XIONS,LINKS,ARCS
      DO 10 L=1,LINKS
      READ(9,*,END=10)LL,FIRSTAB(L),LASTAB(L),TW(L),FIRSTBA(L),
     1 LASTBA(L)
   10 CONTINUE
      READ(6,*)ZONES,XIONS,LINKS
      DO 12 M=1,ZONES
      READ(6,*)FEED
   12 CONTINUE
      DO 20 L=1,LINKS
        READ(6,*,END=20)A(L),B(L),TW(L)
   20 CONTINUE
c     Compute flows on links for LOADFLOW, DARTFLOW or FASTFLOW.
C     Start by reading STARCFLO.DAT for flows from origins.
 9005 FORMAT(1X,2I4,5F10.2)
      DO 30 NO=1,ZONES
        DO 40 K=1,ARCS,5
         READ(12,9005)NN,KK,(FLOW(NO,KT),KT=K,K+4)
   40   CONTINUE
        DO 50 L=1,LINKS
          ABFLOW(L)=0.0
          DO 60 K=FIRSTAB(L),LASTAB(L)
            ABFLOW(L)=ABFLOW(L) + FLOW(NO,K)
   60     CONTINUE
   50   CONTINUE
        DO 70 L=1,LINKS
          BAFLOW(L)=0.0
          DO 80 K=FIRSTBA(L),LASTBA(L)
           BAFLOW(L)=BAFLOW(L) + FLOW(NO,K)
```

```
   80        CONTINUE
   70      CONTINUE
 9015      FORMAT(/'THE START-UP FLOWS FROM ORIGIN ',I4,' ARE')
           WRITE(16,9015)NO
 9025      FORMAT('(But flows into destinations are 0 by default)')
           WRITE(16,9025)
 9035      FORMAT(/'  From A  to B      Flow    From B  to A      Flow')
           WRITE(16,9035)
 9045      FORMAT(3X,2I6,F10.2,5X,2I5,F10.2)
           DO 90 L=1,LINKS
             IF((ABFLOW(L).EQ.0.0).AND.(BAFLOW(L).EQ.0.0))GOTO 430
             WRITE(16,9045)A(L),B(L),ABFLOW(L),B(L),A(L),BAFLOW(L)
  430        CONTINUE
   90      CONTINUE
c        When the loop is re-entered ABFLOW and BAFLOW will be re-
c        assigned.
   30    CONTINUE
c        Compute total flows on links for LOADFLOW, DARTFLOW or FASTFLOW.
C        Read the last part of STARCFLO.DAT
 9055    FORMAT(1X,I4,5F10.2)
         DO 100 K=1,ARCS,5
           READ(12,9055)KK,(TOTFLOW(KT),KT=K,K+4)
  100    CONTINUE
C        ABFLOW and BAFLOW will now be total flows.
         DO 110 L=1,LINKS
            ABFLOW(L)=0.0
            DO 120 K=FIRSTAB(L),LASTAB(L)
               ABFLOW(L)=ABFLOW(L) + TOTFLOW(K)
  120       CONTINUE
  110    CONTINUE
         DO 130 L=1,LINKS
            BAFLOW(L)=0.0
            DO 140 K=FIRSTBA(L),LASTBA(L)
               BAFLOW(L)=BAFLOW(L) + TOTFLOW(K)
  140       CONTINUE
  130    CONTINUE
c        Write the table for SLINKFLO
 9065    FORMAT(/'THE START-UP TOTAL FLOWS ON THE LINKS ARE')
         WRITE(14,9065)
         WRITE(14,9025)
         WRITE(14,9035)
         DO 150 L=1,LINKS
           IF((ABFLOW(L).EQ.0.0).AND.(BAFLOW(L).EQ.0.0))GOTO 440
           WRITE(14,9045)A(L),B(L),ABFLOW(L),B(L),A(L),BAFLOW(L)
  440      CONTINUE
  150    CONTINUE
C        Write the links in bands for SLINKSUM
 9075    FORMAT(/'The following links have not been used')
         WRITE(18,9075)
 9080    FORMAT('     From    to')
         WRITE(18,9080)
 9085    FORMAT(3X,2I6)
         DO 160 L=1,LINKS
            IF(ABFLOW(L).EQ.0.0)WRITE(18,9085)A(L),B(L)
            IF(BAFLOW(L).EQ.0.0)WRITE(18,9085)B(L),A(L)
  160    CONTINUE
 9095    FORMAT(/'The following links have flow between 1 and 10')
```

```
      WRITE(18,9095)
 9105 FORMAT('     From    to     Flow ')
      WRITE(18,9105)
 9115 FORMAT(3X,2I6,F10.2)
      DO 170 L=1,LINKS
      IF((ABFLOW(L).GT.0.0).AND.(ABFLOW(L).LE.10.0))WRITE(18,9115)
     1 A(L),B(L),ABFLOW(L)
      IF((BAFLOW(L).GT.0.0).AND.(BAFLOW(L).LE.10.0))WRITE(18,9115)
     1 B(L),A(L),BAFLOW(L)
  170 CONTINUE
 9125 FORMAT(/'The following links have flow between 11 and 100')
      WRITE(18,9125)
      WRITE(18,9105)
      DO 180 L=1,LINKS
      IF((ABFLOW(L).GT.10.0).AND.(ABFLOW(L).LE.100.0))WRITE(18,9115)
     1 A(L),B(L),ABFLOW(L)
      IF((BAFLOW(L).GT.10.0).AND.(BAFLOW(L).LE.100.0))WRITE(18,9115)
     1 B(L),A(L),BAFLOW(L)
  180 CONTINUE
 9135 FORMAT(/'The following links have flow between 101 and 500')
      WRITE(18,9135)
      WRITE(18,9105)
      DO 190 L=1,LINKS
      IF((ABFLOW(L).GT.100.0).AND.(ABFLOW(L).LE.500.0))WRITE(18,9115)
     1 A(L),B(L),ABFLOW(L)
      IF((BAFLOW(L).GT.100.0).AND.(BAFLOW(L).LE.500.0))WRITE(18,9115)
     1 B(L),A(L),BAFLOW(L)
  190 CONTINUE
 9145 FORMAT(/'The following links have flow between 501 and 1000')
      WRITE(18,9145)
      WRITE(18,9105)
      DO 200 L=1,LINKS
      IF((ABFLOW(L).GT.500.0).AND.(ABFLOW(L).LE.1000.0))WRITE(18,9115)
     1 A(L),B(L),ABFLOW(L)
      IF((BAFLOW(L).GT.500.0).AND.(BAFLOW(L).LE.1000.0))WRITE(18,9115)
     1 B(L),A(L),BAFLOW(L)
  200 CONTINUE
 9155 FORMAT(/'The following links have flow between 1001 and 5000')
      WRITE(18,9155)
      WRITE(18,9105)
      DO 210 L=1,LINKS
      IF((ABFLOW(L).GT.1000.0).AND.(ABFLOW(L).LE.5000.0))
     1 WRITE(18,9115)A(L),B(L),ABFLOW(L)
      IF((BAFLOW(L).GT.1000.0).AND.(BAFLOW(L).LE.5000.0))
     1 WRITE(18,9115)B(L),A(L),BAFLOW(L)
  210 CONTINUE
 9165 FORMAT(/'The following links have flow greater than 5000')
      WRITE(18,9165)
      WRITE(18,9105)
      DO 220 L=1,LINKS
      IF(ABFLOW(L).GT.5000.0)WRITE(18,9115)A(L),B(L),ABFLOW(L)
      IF(BAFLOW(L).GT.5000.0)WRITE(18,9115)B(L),A(L),BAFLOW(L)
  220 CONTINUE
c     Compute flows on links from the final flow pattern.
C     Start by reading FARCFLOW.DAT for flows from origins.
      DO 230 NO=1,ZONES
        DO 240 K=1,ARCS,5
```

```
            READ(13,9005)NN,KK,(FLOW(NO,KT),KT=K,K+4)
  240     CONTINUE
          DO 250 L=1,LINKS
            ABFLOW(L)=0.0
            DO 260 K=FIRSTAB(L),LASTAB(L)
               ABFLOW(L)=ABFLOW(L) + FLOW(NO,K)
  260       CONTINUE
  250     CONTINUE
          DO 270 L=1,LINKS
             BAFLOW(L)=0.0
             DO 280 K=FIRSTBA(L),LASTBA(L)
              BAFLOW(L)=BAFLOW(L) + FLOW(NO,K)
  280        CONTINUE
  270     CONTINUE
C       Write FLINKTRE.DAT
 9175   FORMAT(/'THE FINAL FLOWS FROM ORIGIN ',I4,' ARE')
          WRITE(17,9175)NO
          WRITE(17,9025)
          WRITE(17,9035)
          DO 290 L=1,LINKS
            IF((ABFLOW(L).EQ.0.0).AND.(BAFLOW(L).EQ.0.0))GOTO 450
            WRITE(17,9045)A(L),B(L),ABFLOW(L),B(L),A(L),BAFLOW(L)
  450       CONTINUE
  290     CONTINUE
  230   CONTINUE
C       Compute the total flows on links in the final assignment.
c       Start by reading the last part, total flows, of FARCFLOW.DAT.
        DO 300 K=1,ARCS,5
          READ(13,9055)KK,(TOTFLOW(KT),KT=K,K+4)
  300   CONTINUE
        DO 310 L=1,LINKS
           ABFLOW(L)=0.0
           DO 320 K=FIRSTAB(L),LASTAB(L)
              ABFLOW(L)=ABFLOW(L) + TOTFLOW(K)
  320      CONTINUE
  310   CONTINUE
        DO 330 L=1,LINKS
           BAFLOW(L)=0.0
           DO 340 K=FIRSTBA(L),LASTBA(L)
              BAFLOW(L)=BAFLOW(L) + TOTFLOW(K)
  340      CONTINUE
  330   CONTINUE
C       Write table for FLINKFLO.DAT
        WRITE(15,9085)
        WRITE(15,9025)
        WRITE(15,9035)
        DO 350 L=1,LINKS
          IF((ABFLOW(L).EQ.0.0).AND.(BAFLOW(L).EQ.0.0))GOTO 460
          WRITE(15,9045)A(L),B(L),ABFLOW(L),B(L),A(L),BAFLOW(L)
  460     CONTINUE
  350   CONTINUE
C       Write links in bands for FLINKSUM.DAT
        WRITE(19,9075)
        WRITE(19,9080)
        DO 360 L=1,LINKS
           IF(ABFLOW(L).EQ.0.0)WRITE(19,9085)A(L),B(L)
           IF(BAFLOW(L).EQ.0.0)WRITE(19,9085)B(L),A(L)
```

```
360   CONTINUE
      WRITE(19,9095)
      WRITE(19,9105)
      DO 370 L=1,LINKS
      IF((ABFLOW(L).GT.0.0).AND.(ABFLOW(L).LE.10.0))WRITE(19,9115)
     1  A(L),B(L),ABFLOW(L)
      IF((BAFLOW(L).GT.0.0).AND.(BAFLOW(L).LE.10.0))WRITE(19,9115)
     1  B(L),A(L),BAFLOW(L)
370   CONTINUE
      WRITE(19,9125)
      WRITE(19,9105)
      DO 380 L=1,LINKS
      IF((ABFLOW(L).GT.10.0).AND.(ABFLOW(L).LE.100.0))WRITE(19,9115)
     1  A(L),B(L),ABFLOW(L)
      IF((BAFLOW(L).GT.10.0).AND.(BAFLOW(L).LE.100.0))WRITE(19,9115)
     1  B(L),A(L),BAFLOW(L)
380   CONTINUE
      WRITE(19,9135)
      WRITE(19,9105)
      DO 390 L=1,LINKS
      IF((ABFLOW(L).GT.100.0).AND.(ABFLOW(L).LE.500.0))WRITE(19,9115)
     1  A(L),B(L),ABFLOW(L)
      IF((BAFLOW(L).GT.100.0).AND.(BAFLOW(L).LE.500.0))WRITE(19,9115)
     1  B(L),A(L),BAFLOW(L)
390   CONTINUE
      WRITE(19,9145)
      WRITE(19,9105)
      DO 400 L=1,LINKS
      IF((ABFLOW(L).GT.500.0).AND.(ABFLOW(L).LE.1000.0))
     1  WRITE(19,9115)A(L),B(L),ABFLOW(L)
      IF((BAFLOW(L).GT.500.0).AND.(BAFLOW(L).LE.1000.0))
     1  WRITE(19,9115)B(L),A(L),BAFLOW(L)
400   CONTINUE
      WRITE(19,9155)
      WRITE(19,9105)
      DO 410 L=1,LINKS
      IF((ABFLOW(L).GT.1000.0).AND.(ABFLOW(L).LE.5000.0))
     1  WRITE(19,9115)A(L),B(L),ABFLOW(L)
      IF((BAFLOW(L).GT.1000.0).AND.(BAFLOW(L).LE.5000.0))
     1  WRITE(19,9115)B(L),A(L),BAFLOW(L)
410   CONTINUE
      WRITE(19,9165)
      WRITE(19,9105)
      DO 420 L=1,LINKS
      IF(ABFLOW(L).GT.5000.0)WRITE(19,9115)A(L),B(L),ABFLOW(L)
      IF(BAFLOW(L).GT.5000.0)WRITE(19,9115)B(L),A(L),BAFLOW(L)
420   CONTINUE
      STOP 'Output - SLINKTRE, SLINKFLO, SLINKSUM and
     1               FLINKTRE, FLINKFLO, FLINKSUM'
      END
```