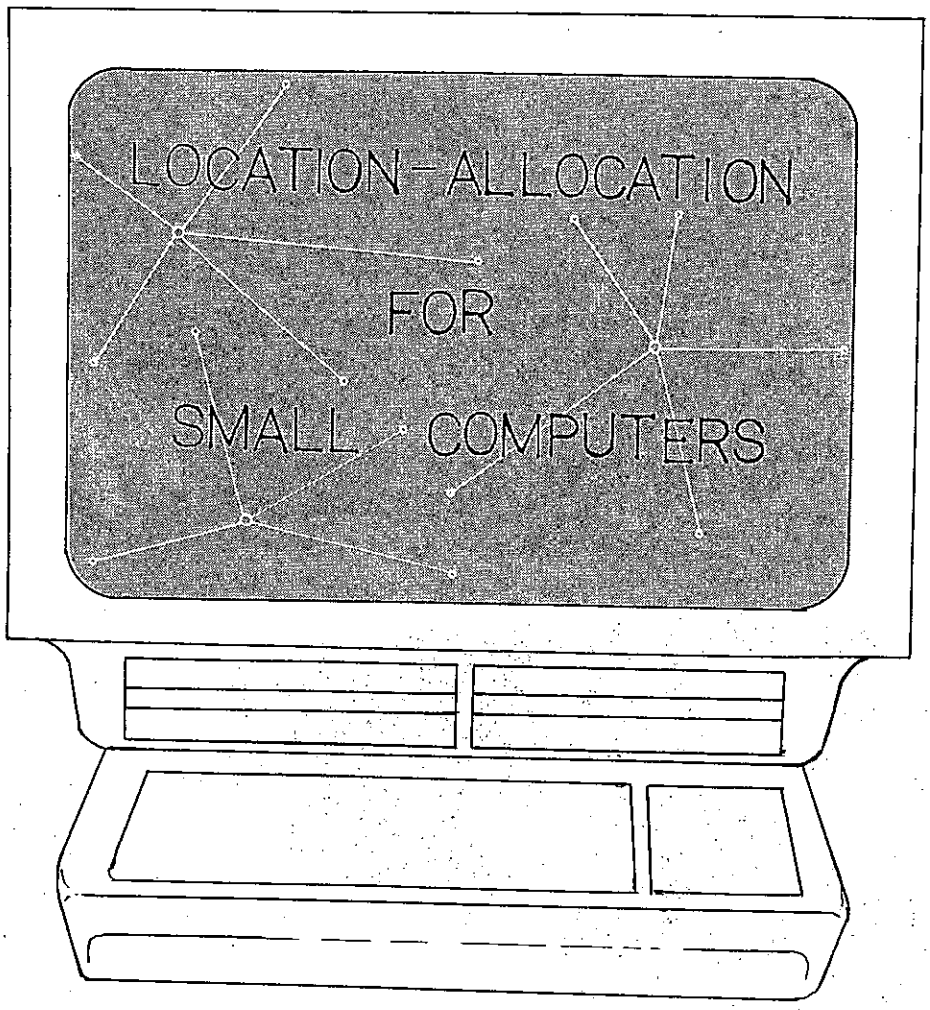


BY

MICHAEL F. GOODCHILD
VALERIAN T. NORONHA

MONOGRAPH 8
DEPARTMENT OF GEOGRAPHY
THE UNIVERSITY OF IOWA
IOWA CITY, IOWA 52242

SEPTEMBER, 1983

LOCATION-ALLOCATION FOR SMALL COMPUTERS

BY
MICHAEL F. GOODCHILD
AND
VALERIAN T. NORONHA

MONOGRAPH 8
DEPARTMENT OF GEOGRAPHY
THE UNIVERSITY OF IOWA
IOWA CITY, IOWA 52242

OCTOBER 1983

Other University of Iowa Monographs

No. 6 Rushton, Gerard, Michael F. Goodchild and Lawrence M. Ostresh, Jr., (eds.). Computer Programs for Location-Allocation Problems, 1973, 321 pp. Price: $20.00.

Available from:

CONDUIT
The University of Iowa
P.O. Box C
Oakdale, Iowa 52319 U.S.A.

Also available from the same address is the package "Optimal Location of Facilities." The package includes:

1. Monograph No. 6;
2. Technical documentation of the programs in Monograph 6;
3. The book: G. Rushton, Optimal Location of Facilities, (Wentworth, N.H.: COMPress, Inc., 1978);
4. A tape containing 14 source programs in ANSI Fortran (approximately 1600 lines of code). Price: $80.00.

No. 7 Hillsman, Edward L., Heuristic Solutions to Location-Allocation Problems: A User's Guide to ALLOC IV, V and VI, 1980, 169 pp. Price: $6.00.

Available from:

Department of Geography
The University of Iowa
Iowa City, Iowa 52242 U.S.A.

Also available from the same address are:

1. Source program listings in ANSI Fortran for Monograph 7; Price: $8.00.
2. A tape containing 6 source programs in ANSI Fortran (approximately 6,000 lines of code). The 6 source programs are written as separate unlabelled files on tape. The tape is recorded at 1600 bpi, in 80-character logical records on a 9-track tape in EBCDIC code. Price: $35.00.

Availability of programs for this monograph:

The generic source programs listed in the text are available in machine-readable form on tape at a cost of $30 including tape. The tape includes the nine programs and the example data sets described in the text. They are recorded as separate files at 1600 bpi, in 80-character logical records blocked to 1600 characters per block on unlabeled 9-track tape, in EBCDIC code.

Write to: Departmental Programmer
Department of Geography
The University of Iowa
Iowa City, Iowa 52242
U.S.A.

Implementations of the codes for various microcomputer systems are under development, beginning with APPLE. For details of availability of diskettes write to

Michael F. Goodchild
Department of Geography
The University of Western Ontario
London, Ontario N6A 5C2
Canada

TABLE OF CONTENTS

LIST OF FIGURES

PREFACE

The origins of this work go back to the 1973 NSF Summer Institute at the University of Iowa, perhaps the first international meetings on location-allocation. One of its more tangible results was a monograph (Rushton et al., 1973) of computer codes and documentation, together with a source code tape. This format proved highly successful, and over the past ten years the programs have been applied to real planning problems in a number of countries, and used in many undergraduate and graduate courses.

The 1973 package was designed for mainframe computers, and for batch operation. Perhaps the most significant conclusion from the recent popularity of personal computers is their power to attract users who had never previously had any contact with digital hardware. This is especially true in the planning professions, and in the third world. Cost, and the personal control the user has over a microcomputer, are enormously persuasive factors, and led us to the conclusion that there was an immediate need for small computer versions of the location-allocation codes. The result is this package of nine interactive programs, which are intended to be the nucleus of any microcomputer implementation. The documentation describes the algorithms and coding in considerable detail, since it is likely that the programs would be 'fine-tuned' to any specific system. We intend to develop implemented versions for various systems, beginning with Apple II, and to distribute the code on diskette together with implementation notes. The code included in this monograph is 'generic', or as system-independent as it is possible to be in this field.

The development of the package was supported by the National Science Foundation* and The University of Western Ontario. We wish to thank

Drs. Gerard Rushton, Michael McNulty and Vinod Tewari for initiating the project and for many helpful suggestions.

Parts of an earlier draft of sections of this monograph, specifically those on algorithms, appear in a monograph titled "Spatial Algorithms for Processing Land Data with a Microcomputer" published by the Lincoln Institute of Land Policy, Cambridge, Massachusetts.

* NSF Grant INT-8017879, "Planning Techniques for the Efficient Location of Public Service Facilities in India."

1. INTRODUCTION

1.1 Microcomputers

It seems clear that the microcomputers introduced in the late 1970s have now produced what might be termed a revolution in human technology. Why this should be so is not at all obvious at first sight. Microcomputers differ from their ancestors, the mainframes and microcomputers of the 60s and 70s, in more than one dimension, and offer advantages, and at the same time disadvantages, at several different levels. It might be useful to begin this monograph by identifying them.

First and most obviously, microcomputers are physically small, as the term suggests. This is the outcome of research during the 1950s and 60s on miniaturization of electronic components for the space program, primarily in the U.S. Some success had been achieved by reducing the size of individual components, helped by the replacement in the 1950s of the cumbersome vacuum tube by the much smaller and more reliable transistor. But the real breakthrough came with the development of large-scale integration of components at the manufacturing stage. Electronic circuitry in digital computers is highly repetitive, and lends itself well to the use of standard integrated units, or 'chips', containing many thousands of components in standard arrangements.

Second, and perhaps surprisingly, chips proved to be extremely easy and cheap to mass produce. By avoiding all of the highly labour-intensive work of wiring individually the large numbers of components of a modern electronic computer, it became possible to produce an integrated circuit or chip at a similar cost to that of a single transistor. A part of the

electronics market which had been lost by the U.S. because of high labour costs in the 60s and early 70s was recaptured as a result.

Third, development of microprocessors coincided with similar developments in appropriate peripherals. Early work in interactive computing in the 60s had made use of the teletype, printing at some 10 characters per second using the 'golf-ball' technology more fully developed later in electric typewriters. By the late 70s it had become possible to print at speeds as high as 120 characters per second on plain paper using a dot matrix, in which characters are formed by hammering a selection from an array of pins onto a fabric ribbon. This technology is readily applicable to the use of various sizes and styles of lettering and to the production of simple graphics. Other examples of small peripherals suitable for microcomputers include casette tapes, floppy disks, and colour graphic display screens in the form of standard TV sets. Each of these is a cheap, physically compact alternative to mainframe peripherals costing tens of thousands of dollars.

Fourth, microprocessors have extended significantly the range of human activities able to make use of digital technology. They have been incorporated into household appliances such as microwave ovens and washing machines, and into automobile carburetors and dashboards. Electronic calculators have replaced slide rules, log tables and mechanical calculating machines, and to a large extent hand calculation and mental arithmetic. And manufacturers have been extremely successful at convincing office managers of the advantages of word processing over the conventional typewriter.

Finally, microcomputers have captured the popular imagination in a way that mainframes, with their air of remote and inaccessible authority never could. A large market has developed for the personal, household

computer, helped particularly among parents of young children by a conviction that the computer industry will continue to be a strong source of employment. Microcomputers are seen as a cost-effective alternative to mainframes in high schools and even elementary schools, and in universities as a means of avoiding the congestion that often surrounds centralized facilities. And video games reinforce the commonly held view that microcomputers are a symbol of the future.

The dividing line between microcomputers and minis is not at all clear. The earliest micros were distinguished by having all processing circuitry concentrated on a single chip, but this sort of large-scale integration is now common for minis and even mainframes. Early micros were 8-bit machines with a mere 8K (8 x 1024) bytes of storage, but micros are now available with 16-bit and even 32-bit words. Speed is a distinguishing factor, since a microprocessor takes orders of magnitude longer to execute basic instructions. Microcomputers are usually single-user machines, whereas minis and mainframes usually serve many users simultaneously. But the most useful definition is probably simply one of size and cost. The term microcomputer is most often associated with a small, table-top machine costing less than $10,000 including peripherals, and built around a microprocessor.

1.2 Location-allocation

Location is an important factor in the success or failure of many human activities. A retailer must choose a location which is accessible to his customers, in order to attract sufficient business. On the other hand a school must be accessible in order for it to provide the level of service expected of a public facility. The geographer's historical interest in

location has been to explain existing patterns of human activities. Why, for example, do some kinds of retail activity tend to cluster together while others form a dispersed pattern? And how can the size and growth of certain towns or cities be explained when others stagnate? The same knowledge can be applied to the planning of future locations, in other words to prescription rather than explanation, and location-allocation is the term given to a set of techniques developed for this purpose.

The aim of location-allocation is to determine the best, or optimal locations for one or more facilities from which some service is to be provided to a spatially dispersed population. The location problem is the question of where to locate, given knowledge of which people are to be served from each facility: the allocation problem is to decide which people should be served from which facility. In most applications both problems must be solved simultaneously, in a 'chicken and egg' fashion, although there are unlimited variations on the basic theme.

More formally, suppose that the population requiring the service is located at a series of points $(x_i, y_i, i=1, n)$ with weights $(w_i, i=1, n)$. The p facilities are to be located at $(u_j, v_j, j=1, p)$, and of the w_i people at place i, a number t_{ij} make use of the facility at j. To do so they each travel a distance d_{ij}, or in some cases the service will be transported that distance to them.

A large number of applications have been modelled in this way since the earliest work on location-allocation in the 1960s, including retail stores, schools, recreation facilities, emergency services such as fire and ambulance stations, government offices, gas stations and restaurants.

The simplest models make the following assumptions:

1) All demand must be allocated to a facility,

$$\sum_j t_{ij} = w_i$$

2) Some applications imply control of allocation patterns by the central planner. For example, in many systems elementary school students are assigned to a school according to their place of residence. In other applications, including the majority of private sector examples, allocation is by consumer choice. In either case, it is often assumed that allocation is to the nearest facility, i.e.

$$t_{ij} = w_i \quad \text{if} \quad d_{ij} < d_{ik},\ k \neq j$$
$$t_{ij} = 0 \quad \text{otherwise}$$

We can now write $t_{ij} = w_i\ x_{ij}$ where $x_{ij} = 1$ if demand at i is allocated to j, 0 otherwise.

3) In the simplest models the objective to be optimized in seeking the best solution is a function of distance alone. One example is to locate so as to minimize total distance,

$$\text{Min} \sum_i \sum_j w_i\ x_{ij}\ d_{ij}$$

In other cases it may be preferable to consider the worst level of service provided, rather than the average level, suggesting that one should minimize the maximum distance over which service is supplied,

$$\text{Min} \max_{i,j} x_{ij}\ d_{ij}$$

Another is to assume that some standard distance S exists, and to maximize the number of people within this distance of a facility,

$$\text{Max} \sum_i \sum_j w_i x_{ij} \qquad \text{where } x_{ij} = 1 \text{ if } d_{ij} \leq S \text{ and } d_{ij} < d_{ik},\ k \neq j;\ x_{ij} = 0 \text{ otherwise}$$

Finally, one might wish to minimize total distance subject to the constraint that no person be further than a specified distance S from a service,

$$\text{Min} \sum_{i,j} w_i x_{ij} d_{ij} \qquad x_{ij} d_{ij} \leq S$$

All four of these objective functions can be handled as examples of a general form

$$\text{Min} \sum_{i,j} x_{ij} c_{ij}$$

by appropriate definition of c_{ij}, an approach which allows one central algorithm to solve all four problems. This strategy has been termed "distance editing". The specific edits for the four problems above are:

Min. total distance	$c_{ij} = w_i d_{ij}$
Min. maximum distance	$c_{ij} = w_i d_{ij}$ if $d_{ij} \leq S$, L otherwise where L is a large number. Solve repeatedly reducing S until solution is infeasible
Max. coverage	$c_{ij} = -w_i$ if $d_{ij} > S$, 0 otherwise
Min. total distance subject to distance constraint	$c_{ij} = w_i d_{ij}$ if $d_{ij} \leq S$, L otherwise

Two largely independent literatures have developed in location-allocation, taking different approaches to the treatment of space and thus to strategies for problem solution. The continuous space approach assumes that all points in the plane are feasible locations for facilities, that

travel is possible in all directions and that distance is measured by some simple rule, usually along straight lines. The discrete space approach assumes that travel is limited to a network, that distances are measured on the network, and that locations are feasible only at a limited set of locations on the network, usually at the nodes. Clearly this is more realistic. However, there are many examples for which the scale of the problem makes continuous space an acceptable approximation. Continuous space is cheaper in terms of both data collection and computer time, and the cost of discrete space will often not be justified by its increased accuracy. The concern in this monograph is primarily with discrete space methods, although some continuous-space options are discussed.

1.3 Problems in adapting location-allocation algorithms to small computers

From the point of view of speed and core memory capacity, microcomputers offer similar resources to the early mainframes of the 1960s. Because they are most commonly single-user systems, the issue of speed is not particularly important: it is quite reasonable to consider problems requiring several hours of continuous processing. However the issue of core memory capacity is much more critical and requires a more or less complete restructuring of the familiar location-allocation algorithms.

Consider a typical microcomputer with 64K bytes (K denotes 1024) of random-access central memory. The operating system is likely to use perhaps 16K, leaving 48K for the user. A real number will require 4, or perhaps even 8 bytes of storage, which means that the system has the capacity to store at most 12,000 real numbers. But the program itself must also be stored in core, either as source code in the case of interpreters, or as compiled code. In either case this is liable to use up several more K of

storage leaving very little for data. Yet in a location-allocation problem with 1000 nodes there are 10^6 distances in the matrix of d_{ij}s.

Microcomputer source languages are notoriously variable. The commonest language, BASIC, has as many dialects as there are brands of microcomputers, particularly in the input and output statements for secondary memory, which are additions to the original language. Some manufacturers offer compiler languages such as FORTRAN, COBOL and PASCAL. And there is no standardization in graphics features. With these points in mind, the subsequent sections describe only the core algorithms: it is assumed that the user will write his own peripheral programs for additional functions, such as graphic display, which would be highly system-dependent. The core programs are written in a dialect of BASIC which allows self-explanatory variable names, and remarks have been inserted in the code. Again, it is assumed that the user will shorten variable names and delete remarks when adapting the code for systems with BASIC interpreters, in order to reduce unnecessary use of core memory. The input and output statements for secondary memory are those used by BASIC-PLUS on the PDP11/70 under RSTS-E, and their functions are explained in a separate section. Also detailed are the domains of each array so that use can be made of variable word length arithmetic, particularly integers (often denoted by %) where they are supported.

2. THE PROGRAM PACKAGE

2.1 Structure

The package consists of 9 independent routines. In terms of the typical application their logical interdependence is summarized in Figure 1. The user will normally begin by identifying a set of demand points. Two

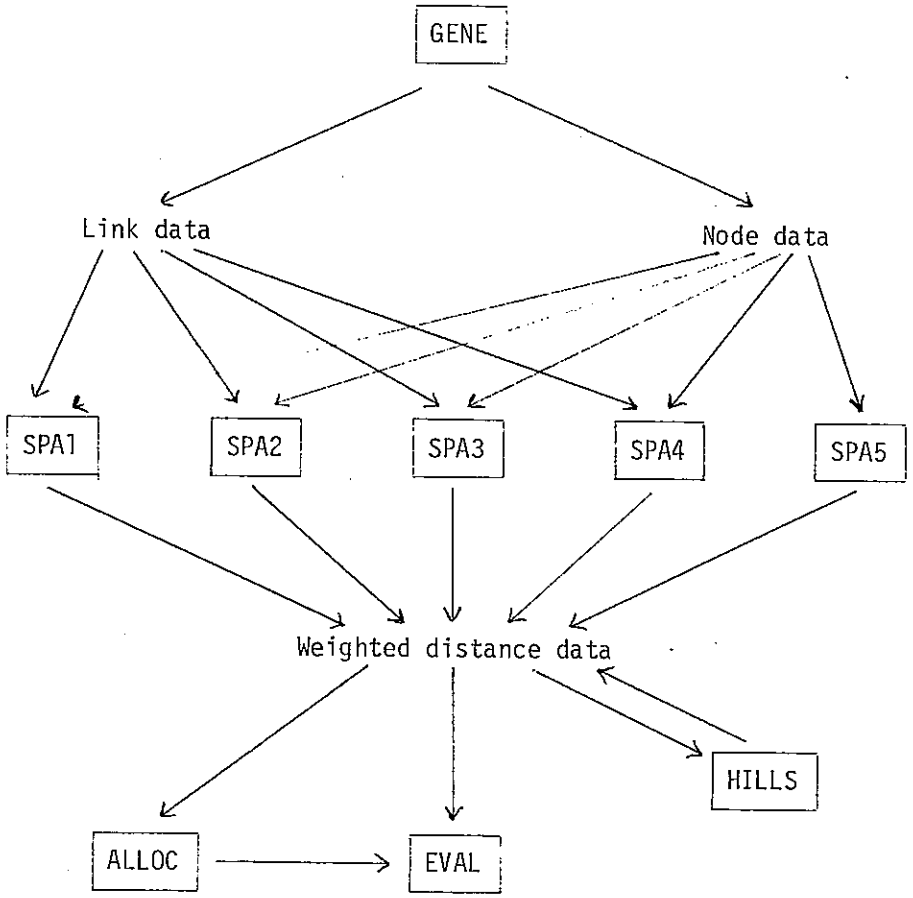


Figure 1: Structure of the package

approaches are then possible. If the transportation network between these points is sufficiently dense, it may be possible to assume straight line travel, and to evaluate distances by reference to coordinates. This is the function of SPA5, which produces weighted distance data in the common data format.

The alternative is to identify all links in the transport network between demand points, noting the origin, destination and length of each link. The four shortest path routines, SPA1 to SPA4, provide alternate versions of the Dijkstra (1959) shortest path algorithm in order to generate the shortest distances between all pairs of demand points. The four routines differ in the demands placed on central memory, SPA1 being the most demanding and SPA4 the least.

The function of GENE is to generate random test data for any of the SPA routines, in a format compatible with their input requirements.

The location-allocation algorithm is contained in ALLOC, and is a modification of the vertex substitution class of algorithms for the p-median problem first defined by Teitz and Bart (1968). ALLOC has options to fix certain sites in the solution. HILLS can be used to edit the data prior to ALLOC in order that the same basic routine can solve a wider class of problems. Finally EVAL produces a detailed evaluation of a particular solution, optimal or non-optimal, for a given data set.

The next sections describe the necessary terms, and then the common data formats in detail.

2.2 Definitions

A node is a point at which demand exists. When using shortest path algorithms SPA1 to SPA4 it is necessary that nodes also be identified at

junctions in the transport network, even if these may have no weight, in order to describe all of the links. Also any feasible sites for central facilities which do not lie either at demand points or at junctions in the network must be identified as nodes, because the search for feasible sites will be limited to the set of nodes.

A link is a path of feasible direct transportation between two nodes, with no intervening nodes. Links and nodes are therefore the basic elements comprising the transport network. Where more than one feasible link exists between two nodes the shortest one should be used.

Weight is the amount of demand present at a node. It will usually be a population count.

A candidate node is a node which may be selected as a facility site. All nodes may be candidates, but it is preferable in the interests of minimizing computing time and data storage to be as restrictive as possible.

Two of the versions of SPA require the study area to be divided into a set of regions. This is purely to improve computational efficiency. In practice the number of regions should be determined by test runs and should be chosen to optimize computing time. The regions are delimited by assigning a region number to each node. GENE creates regions by overlaying a simple grid, for which the user is asked to specify the number of rows and the number of columns. The number of regions is then simply the product of these.

The term string denotes a list of the candidates reachable from a given node within a distance of ZLIMIT, and their associated weighted distances. A string is the logical record from which a weighted distance file is constructed.

2.3 Data formats

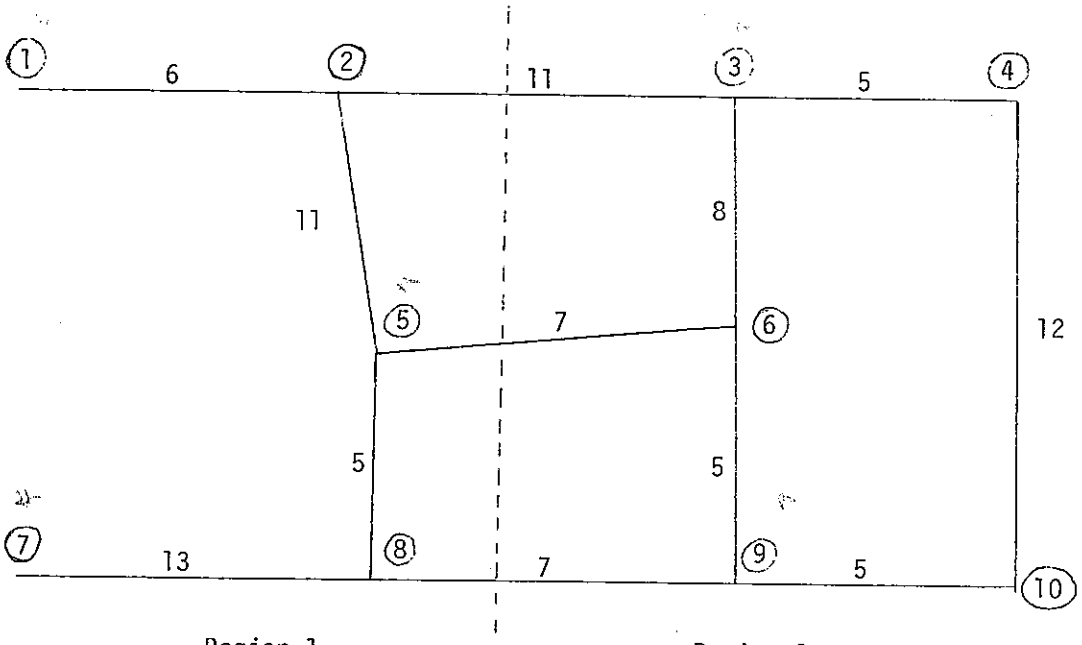
2.3.1 Link data

Link data is used by SPA1, SPA2, SPA3 and SPA4. There are two versions of the format, depending on whether links are arranged in order by origin or randomly. In the ordered form, used by SPA1 and SPA3, the links originating in each demand node are placed together and all links therefore occur twice in the file. The random form is used by SPA2 and SPA4, and in this version of the format each link occurs only once in the file. Note that in this version there is the implicit assumption that links are undirected, that is, that the distance from A to B along a link is identical to the distance from B to A. This need not be true for the ordered link format.

2.3.1.1 Random link format (SPA2, SPA4)

Each record contains three items: origin node number, destination node number and link length. The example network shown in Figure 2 would give the following data (compare section 5.1):

Origin	Destination	Distance
1	2	6
2	3	11
2	5	11
3	4	5
3	6	8
4	10	12
5	6	7
5	8	5
6	9	5
7	8	13
8	9	7
9	10	5



Node	Weight
1*	7
2	3
3*	10
4	51
5*	1
6	5
7*	3
8	0
9*	2
10	18
	100

*denotes candidate node

Figure 2: Example network

Note that the order of any origin and destination pair could be reversed and the records could be shuffled without changing the meaning of the data.

2.3.1.2 Ordered link format (SPA1, SPA3)

In this format all links originating or terminating at a given node are listed together. The valency of a node is the number of links listed for the node. The data must be ordered with the nodes in sequence, and for each node there must be three records, as follows:

1) Identifying node number and valency.

2) Identifying numbers of nodes reached by each link.

3) Length of each link.

The network in Figure 2 would give the following data (compare section 5.1):

node 1 records	(1	1	
	(2		
	(6		
node 2 records	(2	3	
	(1	3	5
	(6	11	11
node 3 records	(3	3	
	(2	4	6
	(11	5	8
node 4 records	(4	2	
	(3	10	
	(5	12	
	.		
	.		
	.		
	etc.		

The sequence of records must correspond to the sequence of nodes, but the order in which links are specified for each node could be changed without affecting the meaning of the data.

2.3.2 Node data

A node data file contains a record for each node, and the records must be arranged in order of node identifying numbers. Each record contains six items of information, as follows:

1) node identifying number.

2) region number. If no system of regions is defined, these can all be set to 1.

3) node weight.

4) candidacy, 1 if the node is a candidate for a site and 0 if not.

5) x coordinate of node.

6) y coordinate of node.

Note that coordinates are processed only by SPA5 and dummy numbers may be substituted if this routine is not used.

The data in Figure 2 would give the following file (compare section 5.1):

Node	Region	Weight	Candidacy	x	y
1	1	7	1	0	0
2	1	3	0	0	0
3	2	10	1	0	0
4	2	51	0	0	0
5	1	1	1	0	0
6	2	5	0	0	0
7	1	3	1	0	0
8	1	0	0	0	0
9	2	2	1	0	0
10	2	18	0	0	0

2.3.3 Weighted distance data

Weighted distance data is produced by each of the SPA programs and HILLS, and read by HILLS, ALLOC and EVAL. It is organized by node, but nodes with no weight are omitted except as possible candidates. Distances are given from each node to each candidate node in order of increasing distance, up to a specified limiting distance ZLIMIT, in the form of the product of distance and weight, as a string. A weighted distance of zero between a node and itself occurs whenever a node is also a candidate.

The three records for each node are organized as follows:

Record 1	1) node identifying number. 2) number of weighted distances given for this node, in other words the string length. 3) weight. 4) candidacy, 1 if node is a candidate, else 0.
Record 2	List of candidate nodes in order of ascending distance from this node, up to a distance of ZLIMIT.
Record 3	Weighted distances to candidate nodes in record 2.

Note that with a very restrictive ZLIMIT it is possible that a node with non-zero weight may have no weighted distances listed, or a string length of zero. Nodes with zero weight may appear only as candidate nodes.

The example of Figure 2 gives the following weighted distance data, for a distance constraint (ZLIMIT) of 20 units (compare section 5.1):

node 1 records	(	1	3	7	1
	(	1	3	5	
	(	0	119	119	
node 2 records	(	2	3	3	0
	(	1	3	5	
	(	18	33	33	
node 3 records	(	3	4	10	1
	(	3	9	5	1
	(	0	130	150	170

node 4 records	(4	2	51	0	
	(3	9			
	(255	867			
node 5 records	(5	5	1	1	
	(5	9	3	1	7
	(0	12	15	17	18
node 6 records	(6	3	5	0	
	(9	5	3		
	(25	35	40		
node 7 records	(7	2	3	1	
	(7	5			
	(0	54			
node 9 records	(9	3	2	1	
	(9	5	3		
	(0	24	26		
node 10 records	(10	3	18	0	
	(9	5	3		
	(90	306	306		

Note the absence of node 8, which has no weight and is not a candidate.

The next sections describe the algorithms of the SPA programs and ALLOC.

3. ALGORITHMS

3.1 Shortest Paths

Shortest path algorithms find the shortest path or shortest distance between one or more nodes in a network. The input consists of a list of links, giving the length of each link by one or more modes of transport, and the output is part or all of the matrix of distances between the nodes. A shortest path algorithm is normally used as the first step in the solution of a discrete-space location-allocation problem, in order to find the distances between each node at which service is needed and each node identified as a possible site for the service (candidate node). SPA (Rushton et al.,

1973) is an example of such a program designed to run on a large computer. It stores each link in core memory.

In a problem with N demand nodes, a typical highly connected transport network would have on the order of 3N links. Each link length would be a real number, occupying probably 4 bytes or 32 bits in an 8-bit microcomputer. The origin and destination nodes could be identified by sequential integers in a smaller number of bits, if that is permitted by the programming language of the microcomputer, but at minimum 8 bytes would be necessary for each link. In total 24N bytes would be required to store the link data set. Clearly, then, it is difficult to handle shortest-path problems of a realistic size ($N \sim 10^3$) with algorithms which require every link to be in core memory, using microcomputers with only 32K or 64K bytes of core.

With a problem involving M candidate nodes the matrix of distances between all demand nodes and all possible candidates contains MN entries, increasing linearly with M. However as M increases, the distance separating demand nodes and the locations from which services are provided decreases, and many of the MN distances become redundant. The user may be able to supply a value ZLIMIT such that distances between candidate nodes and demand nodes which exceed this value can be ignored. The shortest path algorithm need not normally calculate the shortest distance for all MN node pairs, but only for those with shortest-distance less than ZLIMIT.

The algorithm on which each of SPA1 through SPA4 is based is that of Dijkstra (1959). This algorithm is most efficient at finding the distances from a given origin node to all possible destination nodes. It uses the following steps:

1) Label the origin noc

2) Examine all nodes cu
by traversing a sing
to it by adding the
at the 'reached' nod

3) Find the shortest of
associated node 'reac
shortest distance.

4) If any nodes remain '

In each of the SPA routin
the shortest distance foul
order in which distances
order in which informatior

The differences be
a different approach to th
described in the following

3.1.1 SPA1

SPA1 stores all li
Each link is stored twice,
vectors COREDE and COREDI s
link, and the vectors ADDRE
first link and the valency,
Figure 2 would be stored as

1) Label the origin node 'reached', with distance from the origin zero.

2) Examine all nodes currently 'unreached'. If the node can be reached by traversing a single link from a 'reached' node, compute the distance to it by adding the length of this link to the distance from the origin at the 'reached' node. Such nodes will be referred to as 'reachable'.

3) Find the shortest of the distances computed in step 2. Label the associated node 'reached' with a distance from the origin equal to this shortest distance.

4) If any nodes remain 'unreached', return to step 2.

In each of the SPA routines, execution will return to step 2 only as long as the shortest distance found in step 3 is less than ZLIMIT. Note that the order in which distances are found and nodes are reached in step 3 is the order in which information is found in the distance file.

The differences between the four programs occur because each uses a different approach to the storage of the link data. The approaches are described in the following four sections.

3.1.1 SPA1

SPA1 stores all links in core in sequential order by node number. Each link is stored twice, once for each of the nodes it connects. The vectors COREDE and COREDI store the destination node and length of each link, and the vectors ADDRESS and LENG keep track of the position of the first link and the valency, for each origin. For example the network in Figure 2 would be stored as follows:

element	COREDE	COREDI
1	2	6
2	1	6
3	3	11
4	5	11
5	2	11
6	4	5
7	6	8
8	3	5
9	10	12

etc.

element	ADDRESS	LENG
1	0	1
2	1	3
3	4	3
4	7	2

etc.

For example, the links for node 2 start at ADDRESS(2)+1 and there are LENG(2) of them (compare the ordered link data given in section 2.3.1.2). This method of storage is highly efficient for fast computation but this is achieved only by making heavy use of central memory.

SPA1 builds a table of all 'reachable' nodes, that is, 'unreached' nodes reachable by one link from 'reached' nodes, so step 2 of the algorithm need only scan this table in searching for the shortest distance, rather than scan the entire set of nodes. Again this results in computational efficiency at the expense of central memory. The set of reachable nodes is stored in vectors RTO (the node identifying number) and RDIST (the distance from the origin) and is of length RNUM.

3.1.2 SPA2

SPA2 stores all links in central memory like SPA1 but in random order and without duplication. Links are stored in three vectors, OG for the origin node, DE for the destination node and DI for the link length. With N nodes and bN unique links SPA2 will store 3bN numbers in its link data whereas SPA1 must store 4bN numbers plus 2N address pointers and valencies.

In general b can be e
highly connected tran
numbers, which are int
bytes than distances,
b = 2 and that the sys
number the requirement
for SPA1.

Because the li
gained by building a t
links in each executio
connect a 'reached' no
efficient than SPA1, bu

3.1.3 SPA3

SPA3 and SPA4 a
achieved by defining a
in the nodes file. On
in central memory at a
nodes in the same regi
As soon as a node is re
reread to append the li
be computed to all desi
in central memory. Its
restrict the set of dis

SPA3 is the most
links and builds a tabl
store the starting loca

In general b can be expected to take a value of roughly 2 for a typical highly connected transport network. Note however that in many systems node numbers, which are integers with limited domains, may be stored in fewer bytes than distances, which are likely to be real numbers. Assuming that b = 2 and that the system requires 4 bytes per real number and 2 per node number the requirements in bytes of central memory are 16N for SPA2 and 26N for SPA1.

Because the links are stored randomly there would be no advantage gained by building a table of reachable nodes. So SPA2 must examine all links in each execution of step 2 of the algorithm. finding those which connect a 'reached' node with an 'unreached' one. SPA2 is inevitably less efficient than SPA1, but is also less demanding on central memory.

3.1.3 SPA3

SPA3 and SPA4 avoid storing all links in central memory. This is achieved by defining a set of regions and allocating each node to its region in the nodes file. Only those regions which contain reachable nodes need be in central memory at any one time. At the outset all links with one or both nodes in the same region as the origin node are brought into central memory. As soon as a node is reached which is in a new region the links file is reread to append the links of this new region. Of course, if distances must be computed to all destinations this method will eventually require all links in central memory. Its potential efficiency lies in the use of ZLIMIT to restrict the set of distances.

SPA3 is the most complex of the four routines because it uses ordered links and builds a table of reachable nodes. The vector POINT is used to store the starting location in central memory of the links for a given node,

when the node is in one of the regions currently in core. ADDRESS and LENG are used in SPA3 for the starting location and length of the first, second etc. node in central memory.

3.1.4 SPA4

SPA4 takes the same approach of dividing the study area into regions, but assumes the links to be in random order. Like SPA2 there would be no advantage in building a reachable node table so every link in central memory is examined in each execution of step 2 of the basic algorithm.

3.2 Location-allocation

The Teitz and Bart algorithm (Teitz and Bart, 1968) is one of the most commonly used procedures for discrete space location-allocation. Given a network of N demand nodes, M of which are identified as candidates for selection as centres, and given the M x N natrix of shortest paths between demand nodes and candidates, it makes a systematic search for the subset of P candidate nodes which optimize an objective function. The objective function is the traditional one of location-allocation, the total weighted distance separating each demand node from its nearest solution node. The data set of shortest distances is assumed to be organized by demand node, and to include for each demand node the weight and a list of weighted distances to candidate nodes in ascending order, usually truncated at some upper bound. This is the form of distance data generated by the shortest path algorithms described in section 3.1.

The algorithm requires a starting solution, which is often a random subset of P candidate nodes or an intuitive guess. The major cycle of the algorithm then examines each of these solution nodes in turn. A swap is

made by dropping this s
nodes not currently in

Consider the ma
candidate nodes and each
In all, M x P swaps are
the M candidates, althou
solution node by itself

Candidate nodes

M

Figure 3:

There are several po
of swaps. In the original Te
was selected, and swapped in
the best swap produced an imp
permanent. The next candidat
the swap matrix this correspor
Alternatively one could take t
ment by each candidate node, i
matrix.

made by dropping this solution node and replacing it by one of the candidate nodes not currently in the solution.

Consider the matrix shown in Figure 3, in which the rows represent candidate nodes and each column represents one of the current solution nodes. In all, M x P swaps are possible between the P current solution nodes and the M candidates, although P^2 of these are swaps which would replace a solution node by itself or by a candidate which is already in the solution.

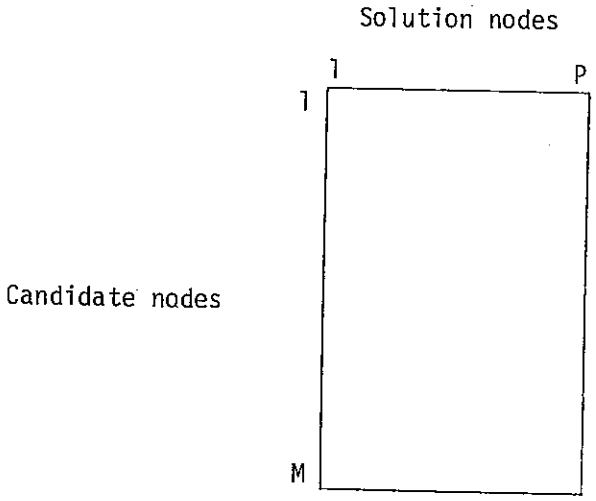


Figure 3: The matrix of possible swaps

There are several possible approaches to the systematic examination of swaps. In the original Teitz and Bart algorithm the first candidate node was selected, and swapped in as a replacement for each solution node. If the best swap produced an improvement in the objective function, it was made permanent. The next candidate node was then examined in the same way. In the swap matrix this corresponds to the examination of one row at a time. Alternatively one could take the first solution node, and consider its replacement by each candidate node, in other words examine one column in the swap matrix.

With the distance matrix in core there is little difference between the two methods. But with the distances stored in secondary memory it is essential that the number of passes through the weighted distance data be kept as low as possible, by examining the greatest number of swaps at once. The approach used is to calculate the value of the objective function for every possible swap during one pass of the weighted distance data.

The program assumes that it may not be possible to store all of the swap matrix in core. The parameter MAXSIZ, which is the number of elements in SUM, is the maximum number of entries of the matrix which can be held in core. If M x P exceeds MAXSIZ the program automatically makes additional passes in each cycle; NUMBER is the number of entries being processed in the current pass. At the end of each pass the value of each element of SUM is the value of the objective function if a specific candidate node is swapped in as replacement for a specific solution node. Solution nodes can be defined as permanent by setting the appropriate element in IN to 2, in which case the objective function will still be evaluated for the swap, but the swap will never be selected as best at the end of a cycle.

In each pass of the data set the program reads sequentially through the weighted distance data, in each case reading the weight and the list of weighted distances to candidate nodes in ascending order. A number of steps are executed for each demand node. First the signs of all elements of SUM are made negative. They will be made positive when a solution node is found to serve that demand node. Since the set of solution nodes is different for each element of SUM because of swaps, the resetting of signs is complex and depends on several conditions. The candidate nodes and associated weighted distances are processed in the order in which they are read for each

demand node, i.e., in ascending order of distance. For each one three conditions are possible:

1) The candidate node is in the solution and not permanent, and is the first such node encountered. It is possible that this solution node will become the one providing service to this demand node when a current solution node is swapped out. Thus all elements of SUM other than those in the column occupied by this node must be incremented. This is shown in Figure 4a.
2) The candidate node is not in the solution. In this case the elements affected are those in the appropriate row of SUM, since in each of these cases this candidate node will enter the solution and may be closer than the solution node currently providing service. This is shown in Figure 4b.
3) The candidate node is the second solution node encountered. In this case the affected elements of SUM are those not affected by the first condition above, i.e., the column occupied by the first solution node encountered (Figure 4a).

In each case the affected elements are examined in turn. If an element is negative, its sign is reversed and it is incremented by the weighted distance of the candidate node from the demand node. If the element is already positive then it is already being served by a closer candidate node identified earlier and can be skipped. All elements of SUM will be positive when two solution nodes have been encountered. However it is possible that the list of candidates for a demand node does not contain two solution nodes. In this case the remaining negative elements are made positive and incremented by a large constant PENALTY, flagging them as infeasible swaps.

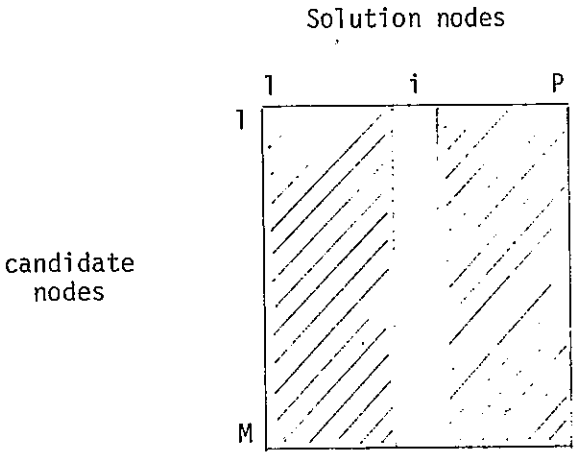


Figure 4a: Shaded elements of SUM are affected when a candidate is encountered which is the ith current solution node.

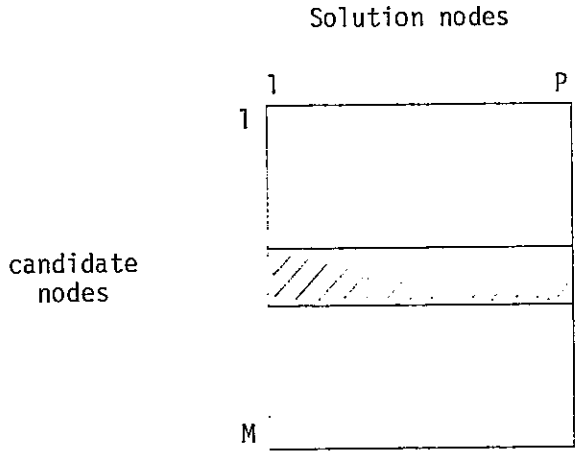


Figure 4b: Shaded elements of SUM are affected when a candidate is encountered which is not currently a solution node.

3.3 Hillsman editing

The function of Hillsman editing is to modify the set of weighted distances $w_i\ d_{ij}$. When input to ALLOC, weighted distances result in a solution of the p-median problem, in which the objective is to minimize the total distance travelled. By editing the input it is possible to use the same ALLOC code to solve a variety of different problems with different objectives. HILLS implements six of these options, as follows:

1) p-median with maximum distance constraint (Khumawala, 1973)

The problem is to locate centres to minimize total distance from each node to the nearest centre, while ensuring that no one travel further than a distance S. Let the weighted distances be denoted by c_{ij},

$$c_{ij} = w_i\ d_{ij}$$

Then this problem can be solved by substituting a large penalty whenever d_{ij} would be greater than S, that is,

$$c_{ij} = w_i\ d_{ij} \quad \text{if} \quad d_{ij} \leqslant S$$

$$c_{ij} = \text{PENALTY} \quad \text{if} \quad d_{ij} > S$$

In HILLS the value of PENALTY is set to 10^{20}

2) Minimize centres with maximum distance constraint (Toregas and Revelle, 1972)

This problem seeks to find the minimum number of centres needed to cover a set of nodes with a minimal level of accessibility. The editing is as follows:

$$c_{ij} = 1 \quad \text{if} \quad i = j$$

$$c_{ij} = 0 \quad \text{if} \quad i \neq j \quad \text{and} \quad d_{ij} \leqslant S$$

$$c_{ij} = \text{PENALTY} \quad \text{if} \quad i \neq j \quad \text{and} \quad d_{ij} > S$$

3) Maximal covering problem (Church and Revelle, 1974)

This problem seeks to maximize the number of persons within a distance S of at least one centre, or alternatively to minimize the number who are more than a distance S away. The editing is as follows:

$$c_{ij} = 0 \quad \text{if} \quad d_{ij} \leqslant S$$
$$c_{ij} = w_i \quad \text{if} \quad d_{ij} > S$$

4) Maximal covering with maximum distance constraint (Church and Revelle, 1976)

This problem locates centres so as to maximize the number of individuals within distance S of a centre while ensuring that all individuals are within distance T of a centre. The editing is as follows:

$$c_{ij} = 0 \quad \text{if} \quad d_{ij} \leqslant S$$
$$c_{ij} = w_i \quad \text{if} \quad S < d_{ij} \leqslant T$$
$$c_{ij} = \text{PENALTY} \quad \text{if} \quad d_{ij} > T$$

5) Attendance maximizing (Holmes et al., 1972)

Centres are located to maximize attendance, on the assumption that attendance falls off linearly with distance. The editing is as follows:

$$c_{ij} = A - w_i (1 - bd_{ij})$$

where b is a constant, and A is a large number. In each row of the distance matrix A is set equal to the largest value of $w_i (1 - bd_{ij})$, ensuring that all values of c_{ij} are positive. If bd_{ij} is greater than 1, $w_i (1 - bd_{ij})$ is set to zero.

6) Minimize total powered distance (Morrill, 1974)

This editing allows ALLOC to locate centres to minimize aggregate squared or cubed distance, or any other power of distance. The required transformation is:

$$c_{ij} = w_i \, d_{ij}^{\,b}$$

The general approach used by HILLS is to read a weighted distance data set, and make the necessary edits to create a new data set with the same format. The program prompts for all necessary parameters.

3.4 Data generation

The purpose of GENE is to provide easy access to random test data for the package. It places a prescribed number of nodes randomly on an area 100 by 100, overlays a grid of regions, and randomly assigns weights to each node. Candidacy is determined randomly using a probability per node prescribed by the user. Links are generated systematically, each node being joined to a number of its nearest neighbours.

4. THE PROGRAMS

4.1 General

As noted in the introduction, the programs have been written in an expanded BASIC which permits self-explanatory variable names, and have been commented throughout. It is assumed that 'fine-tuning' will be necessary to adapt and optimize the code for specific machines, and this section is intended to provide additional information with that in mind.

Variable names are used consistently throughout the package. The next section gives the domain of each variable as a guide to users of machines on which the number of bytes used to store a number can be varied, either by the use of integer arithmetic or by declaring variable precision. None of these features were present in the earliest BASIC, which assumed all variables to be real.

The subsequent section describes the use of input and output commands with respect to secondary memory, since these are not standardized among the various dialects of BASIC.

Each of the programs has checks which compare input parameters to the available memory. The dimension statements and associated limits should be adjusted to the capacity of the system being used. Once this is done, it will be possible to determine which of SPA1 through SPA4 should be used in any particular application. It can be assumed with some confidence that SPA1 will always be faster than SPA2, SPA2 than SPA3, and SPA3 than SPA4, so the choice between them should be based simply on central memory capacity. The optimal number of regions to be used in SPA3 and SPA4 depends on the characteristics of secondary memory input and output for each system, and will have to be established by experiment.

4.2 Variables

4.2.1 Vectors and Arrays

Name	Programs	Dimension	Domain	Meaning
ADDRESS	SPA1	MAXN(SPA1)	Integers 1 to MAXN(SPA1)	address of first link of each origin whose links are in core
	SPA3	MAXIN(SPA3)	Integers 0 to MAXIN(SPA3)	
CANDIDATE	SPA	MAXN	0,1	1 if a node is a candidate (scalar in ALLOC,GENE)
CID	ALLOC EVAL	MAXP	integers 1 to N	identifying node number of each site
COREDE	SPA1 SPA3	MAXCORE	integers 1 to N	destination node for each link in core
COREDI	SPA1 SPA3	MAXCORE	real positive numbers	length for each link in core
DE	SPA2 SPA4	MAXLINKS	integers 1 to N	destination node of each link in core
DI	SPA2 SPA4	MAXLINKS	positive real numbers	length of each link in core
FLAG	GENE SPA5	MAXN	0,1	1 if distance already output

Name	Programs	Dimension	Domain	Meaning
IN	ALLOC EVAL	MAXN	0,1,2	1 if node is currently a site. 2 if a site is fixed at node
INDEX	ALLOC	MAXN	integers 1 to M	position of node in candidate list
INDIST	SPA1 SPA3	MAXVCY	positive real numbers	lengths of links for each origin
INNODE	SPA1 SPA3	MAXVCY	integers 1 to N	destinations of links for each origin
INREG	SPA4	MAXREG	0,1	1 if region's links are in core
LEGEND$	EVAL	2	string	legends for output
LENG	SPA1 SPA3	MAXN(SPA1) MAXIN(SPA3)	integers 1 to MAXN(SPA1) integers 0 to MAXIN(SPA3)	valency of each origin whose links are in core
OG	SPA2	MAXLINKS	integers 1 to N	origin node of each link in core
ORDER	GENE SPA	MAXLEN (MAXN by MAXVCY in GENE)	integers 1 to N	candidates reached in ascending order of distance
POINT	SPA3	MAXN	integers 0 to MAXIN	if node's links are in core, pointer to first link, else 0
RDIST	SPA1 SPA3	MAXLIST	positive real numbers	distance to each reachable node
REGION	SPA3 SPA4	MAXN	integers 1 to R	region number for each node (scalar in GENE)
RTO	SPA1 SPA3	MAXLIST	integer 1 to N	table of reachable nodes
STRL	GENE	MAXN (scalar in other programs)	integers 0 to M	number of candidates within ZLIMIT of node, or string length
SUM	ALLOC	MAXSIZ	positive and negative real numbers	value of objective function following swap

Name	Programs	Dimension	Domain	Meaning
TONODE	ALLOC HILLS EVAL	MAXLEN	integers 1 to N	candidates reached in order of ascending distance from a node
WEIGHT	SPA	MAXN	positive real numbers and zero	node weight (scalar in ALLOC,GENE)
WTDIST	ALLOC HILLS EVAL	MAXLEN	positive real numbers and zero	weighted distances to candidate from a node
X	GENE SPA5	MAXN	positive real numbers	x coordinate of node
Y	GENE SPA5	MAXN	positive real numbers	y coordinate of node
Z	GENE SPA	MAXN	positive and negative real numbers	distance to each node
TOTDEM TOTDIST MAXDIST FAR TOTDIST2 MAXDIST2 FAR2	EVAL	MAXP	positive real numbers and zero	sums used in computing statistics (see comments)

4.2.2 Scalars

Name	Programs	Domain	Meaning
A	HILLS	positive real numbers	offset to option 5
ADD	ALLOC	positive real number	weighted distance
B	HILLS	positive real number	distance power
BEST	ALLOC	positive real number	objective function for best swap found
CANDIDATE	GENE ALLOC HILLS EVAL	0,1	1 if node is a candidate
CENTERS	ALLOC EVAL	0,1,2	number of sites already processed for this node

Name
DEST
DIST
DIST1 DIST2
DUMMY
FILE1:
FILE2$
FILE3$
FIRST
FORM$
I
ID
INCORE
IR
J
JR

Name	Programs	Domain	Meaning
DEST	GENE SPA4	integer 1 to N	a destination
DIST	GENE SPA4 HILLS	positive real number	a distance
DIST1 DIST2	EVAL	positive real numbers	distances to nearest and second nearest centres
DUMMY	SPA	positive real numbers	an input parameter read but not processed
FILE1$	GENE ALLOC SPA HILLS EVAL	string	name of input file (weighted distances for ALLOC, HILLS, EVAL, nodes data for SPA, GENE)
FILE2$	GENE SPA1 SPA2 SPA3 SPA4	string	name of links file
FILE3$	SPA HILLS	string	name of file for weighted distances
FIRST	ALLOC EVAL	integer 1 to N	node of first site found
FORM$	EVAL	string	format for printed output
I	ALLOC SPA5	integer 1 to M	a candidate node
ID	ALLOC EVAL	integer 1 to N	candidate selected as a site in initial solution
INCORE	SPA3	integer	number of nodes whose links are in core
IR	SPA3 SPA4	integer 1 to R	a region
J	ALLOC EVAL	integer 1 to P	a site
JR	SPA4	integer, 1 to R	a region

Name	Programs	Domain	Meaning
K	ALLOC	integer 1 to NUMBER	a swap
KHIGH	ALLOC	integer 1 to M*P	index of last swap in a column
KK	GENE ALLOC SPA HILLS EVAL	integer 1 to N	a node
KL	GENE SPA	integer, 1 to N	a node
KLOW	ALLOC	integer 1 to M*P	index of first swap in a column
KREAD	SPA	integer	node number as read from input
KSAVE	ALLOC	integer 1 to M*P	index of best swap
L	ALLOC SPA HILLS EVAL	integer 1 to STRL	a candidate in a string
LAST	ALLOC	integer 0 to M*P	number of swaps processed in previous passes
LC	SPA2 SPA4	integer 1 to NLINKS	a link in core
LINK	GENE SPA1 SPA3	integer 1 to LPN or VALENCY	a link
LPN	GENE	integer 1 to MAXLEN	number of links to be generated per node
M	ALLOC	integer 1 to N	number of candidate nodes
MAXCORE	SPA1 SPA3	integer	maximum number of links in core in a run
MAXIN	SPA3	integer	maximum number of nodes whose links are in core
MAXLEN	ALLOC SPA HILLS EVAL	integer	maximum number of candidates within ZLIMIT of any node

Name	Programs	Domain	Meaning
MAXLINKS	SPA2 SPA4	integer	maximum number of links to be stored in core
MAXLIST	SPA1 SPA3	integer	maximum number of reachable nodes in a run
MAXN	GENE ALLOC SPA EVAL	integer	maximum number of nodes
MAXP	ALLOC EVAL	integer	maximum number of sites in solution
MAXREG	SPA4	integer	maximum number of regions
MAXSIZ	ALLOC	integer	maximum number of swaps considered in one pass
MAXVCY	GENE SPA1 SPA3	integer	maximum valency of a node
MOST1 MOST2	EVAL	integers, 1 to P	most disposable centres
N	GENE ALLOC SPA	integer, 1 to MAXN	number of nodes in this run
N1	SPA1 SPA3	integer, 0 to MAXIN	position in core of first link for this node
N2	SPA1 SPA3	integer, 0 to MAXIN	position in core of last link for this node
NEWREG	SPA3 SPA4	integer, 1 to R	new region to append
NEWSTRING	GENE	integer 1 to MAXLEN	length of new string
NH	GENE	integer	number of columns in the grid of regions
NLINKS	SPA1 SPA2 SPA3 SPA4	integer, 1 to MAXLINKS	number of links in core in this run

Name	Programs	Domain	Meaning
NUMBER	ALLOC	integer 1 to M*P	number of swaps being processed in this pass
NV	GENE	integer	number of rows in the grid of regions
OLDOBJ	ALLOC	positive real number	objective function of current solution
OPTION	GENE HILLS EVAL	integer	option number
ORIG	GENE SPA4	integer 1 to N	an origin
P	ALLOC EVAL	integer, 1 to MAXP	number of sites in this run
PENALTY	ALLOC HILLS	positive real number	penalty when demand cannot be served
PROP	GENE	real number between 0 and 1	proportion of nodes which are candidates
R	GENE SPA3 SPA4	integer	number of regions
REACH	GENE SPA	integer 1 to N	node with shortest distance
REGION	GENE	integer 1 to R	a region
RM	SPA1 SPA3	integer 1 to RNUM	an entry in the reachable node table
RNUM	SPA1 SPA3	integer, 1 to N	number of reachable nodes
RNUM1	SPA1 SPA3	integer, 1 to N	number of reachable nodes after compression of list
S	HILLS	positive real number	distance constraint
SECOND	EVAL	integer, 1 to N	node number of second nearest centre

Name	Programs	Domain	Meaning
STRL	ALLOC SPA HILLS EVAL	integer 1 to MAXLEN	number of candidates within ZLIMIT of a node, or length of string
T	HILLS	positive real number	distance constraint
VALENCY	SPA1 SPA3	integer, 1 to MAXVCY	valency of a node
WEIGHT	GENE ALLOC HILLS EVAL	positive real number	weight of a node
ZLIMIT	SPA	positive real number	maximum relevant distance
ZMIN	GENE SPA	positive real number	shortest distance found
ZTOT	SPA2 SPA4	positive real number	total distance
NOSERVE NOSERVE2 SFAR DISPTOT DISPMAX STOTDEM STOTDIST SMAXDIST	EVAL	positive real numbers	sums: see comments

4.3 Input and output statements

OPEN string FOR INPUT / OUTPUT AS FILE expression

The file named in the specified character string is opened for input or output on the channel given by the value of the expression. Channel 1 is used for input by all programs. Channel 2 is used by GENE and SPA1 through SPA4 to input link data, and channel 3 is used by SPA and HILLS to output weighted distances

ON ERROR GO TO line number

This statement is used to detect and recover from an end-of-file condition in ALLOC, SPA2 and SPA4, HILLS and EVAL. The value of ERR is set to 11 and control transfers to the specified line number. These programs check that control has not been transferred because of some other error on the input channel.

RESUME line number

This statement is used to continue execution after an ON ERROR statement has transferred control following an end-of-file condition.

CLOSE expression

The file on the specified channel is closed.

INPUT # expression, list

The values of the variables given in the list are input from the channel given by the expression. Each INPUT statement causes a move to a new record. Data is free-formatted, delimited by commas.

PRINT # expression, list

The value of the given variables are output on the specified channel. Note that each PRINT statement must specifically include the output of delimiters (commas) in order for the data to be read successfully by an INPUT statement in a subsequent program. This is an awkward feature of the PDP11/70.

PRINT USING string, list

This form of the PRINT statement is used in EVAL to produce formatted output, in order to organize the printed output of the program in columns with headings.

4.4 Functions

Four system functions are used at various points in the package. INT, SQR and RND are likely to be available in every version of BASIC, but ERR may not have an equivalent.

INT(X) truncates the value of X to an integer (GENE, SPA5).

SQR(X) takes the square root of X (GENE, SPA5).

RND returns a random number in the range 0 to 1 (GENE).

ERR returns an integer indicating the condition which has caused an ON ERROR statement to transfer control. An end-of-file gives a value of 11 (ALLOC, HILLS, EVAL, SPA2, SPA4).

5. EXAMPLE APPLICATIONS

5.1 Simple problem

The simple network shown in Figure 2 was used for the first set of examples. The shortest path algorithm produced a 10 x 10 distance matrix, which was processed by HILLS and ALLOC to obtain solutions to a number of different two-site problems, as follows.

5.1.1 p-median

The two-site p-median solution is nodes 3 and 9, with a total distance of 594. The maximum distance travelled is 20, from node 7 to node 9.

5.1.2 p-median with maximum distance constraints

A distance constraint of 20 or more will not affect the p-median solution. Editing using HILLS for a maximum of 17 through 19 produces a

solution of nodes 3 and 7, with a maximum distance of 17. This is the minimum feasible distance constraint for 2 sites, so (3,7) is the solution to the 2-centre problem.

5.1.3 Minimize centres with a maximum distance constraint

Two centres are necessary to cover all nodes with a distance constraint of 17. At 16, 4 centres are necessary: adding a third centre alone does not reduce the distance limit below 17.

5.1.4 Maximize coverage

The shortest link in the network is of length 5, so the solution for coverage problems with distances less than 5 is to place the two sites at the candidates with the highest weight, that is, nodes 1 and 3. For distances between 5 and 16 the best sites are 3 and 9. At a distance of 17 the maximal covering problem becomes identical to the m-centre problem as all of the demand can be covered from nodes 3 and 7.

5.1.5 Maximal covering with maximal distance constraint

Suppose it is required to cover as much demand as possible within 10 units, but with the constraint that noone be more than 18 distance units from a site. The solution is nodes 3 and 7, leaving a total demand of 31 uncovered. Note that without the distance constraint the solution would have been (3,9) as above, with only 14 uncovered.

5.1.6 Attendance maximizing

Since the maximum distance travelled in the p-median solution is 20, any value of the friction of distance b less than .05 will produce the p-median solution, (3,9). Raising b to 0.1 implies that many of the demand

points are too far away from a site to produce any demand, but the solution remains (3,9).

5.1.7 Minimizing total powered distance

A distance power of 1.0 will produce the p-median solution. Integer powers as high as 10 were tried, but each produced the same solution (3,9). In general, higher powers will force the solution sites closer to the points of greatest weight, but the p-median solution already occupies the candidate nodes closest to the nodes of highest demand.

5.2 London problem

This section describes a problem of more realistic size, with 150 nodes and 9 sites to be located. London, Ontario is a city of approximately 250,000 population and in 1976 was served by 9 fire stations, each equipped with a minimum of one pumper truck. The pattern of fire alarms in the city and the suitability of the existing nine locations to deal with them were the subject of a Ph.D. dissertation by Waters (1977) from which this example is taken.

All 2,459 alarms occurring in 1973 were recorded and geocoded. The city street network was reduced to a set of some 250 major arterial links and 150 associated intersections (Fig. 5). These are the links followed by fire trucks from the responding station to the neighbourhood of the fire. Each alarm was then allocated to the nearest of the 150 intersections to create a set of demand weights.

The lengths of each link were coded in units of 50 feet, and processed by SPAI in ordered format. Studies of the London area have shown that the speed of a responding truck is almost constant over the city, so

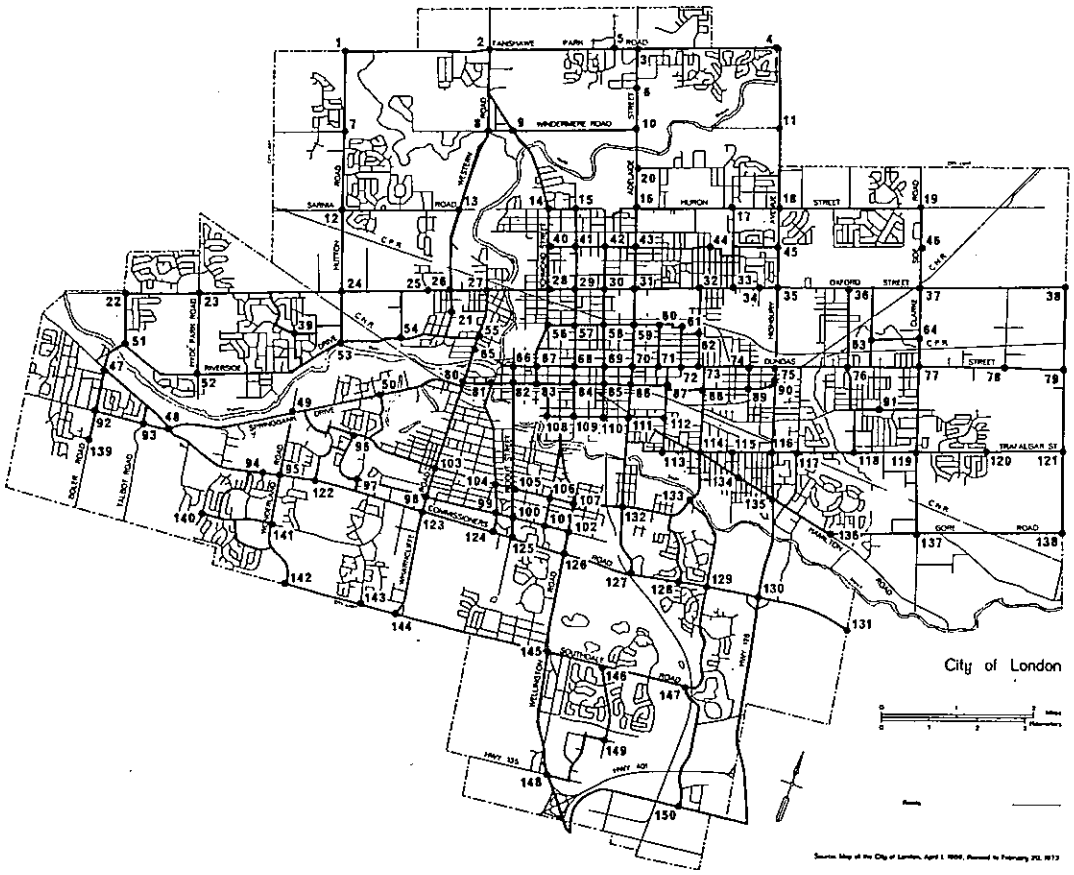


Figure 5: Major streets, London, Ontario

distance can be taken as a surrogate for time. Thus solving the p-median problem will result in a solution which minimizes expected response time to a randomly chose fire.

Three solutions are described below:

1) relocate all 9 existing stations to minimize expected response time,
2) locate a tenth station, holding the existing 9 fixed, so as to reduce expected response time as much as possible,
3) as 2, but with maximum rather than expected response time as the objective.

Each solution will now be described in turn.

5.2.1 Relocating 9 fire stations

Using the existing locations as a starting solution, ALLOC required 8 cycles to obtain a solution to the p-median problem (Figure 6). The total distance travelled was reduced from 268901 units to 228242. Note that two sites, at nodes 95 and 145, were not changed and that one site was moved twice (from 111 to 119 to 77).

The existing sites and the final solution were compared using EVAL. Part of the output of EVAL for the solution is shown in Figure 7.

5.2.2 Locating fire station 10 using the p-median solution

ALLOC was run for 10 sites, using the existing 9 as fixed locations and introducing node 120 as a possible tenth site. Only one iteration is necessary for this problem, and the solution found was node 119. This reduced the total distance travelled from 268901 with 9 sites to 241157 with 10. The maximum distance was reduced from 469 with 9 sites to 361 with 10.

Figure 6: Example run of ALLOC

```
Ready

RUN NALLOC
NUMBER OF NODES?  150
NUMBER OF SITES TO BE LOCATED?  9
SOLUTION NODE 1?  8
ENTER '1' IF MOBILE, '2' IF FIXED?  1
SOLUTION NODE 2?  24
ENTER '1' IF MOBILE, '2' IF FIXED?  1
SOLUTION NODE 3?  30
ENTER '1' IF MOBILE, '2' IF FIXED?  1
SOLUTION NODE 4?  45
ENTER '1' IF MOBILE, '2' IF FIXED?  1
SOLUTION NODE 5?  84
ENTER '1' IF MOBILE, '2' IF FIXED?  1
SOLUTION NODE 6?  88
ENTER '1' IF MOBILE, '2' IF FIXED?  1
SOLUTION NODE 7?  95
ENTER '1' IF MOBILE, '2' IF FIXED?  1
SOLUTION NODE 8?  111
ENTER '1' IF MOBILE, '2' IF FIXED?  1
SOLUTION NODE 9?  145
ENTER '1' IF MOBILE, '2' IF FIXED?  1
NAME OF INPUT FILE?  DIST. 4
   150 CANDIDATES FOUND
OJBECTIVE FUNCTION IS 268901
BEST SWAP IS 119 TH CANDIDATE FOR 111
   119 TH CANDIDATE IS NODE 119
```

Figure 6: continued

```
OBJECTIVE FUNCTION IS 249973
BEST SWAP IS 83 TH CANDIDATE FOR 84
   83 TH CANDIDATE IS NODE 83
OBJECTIVE FUNCTION IS 244824
BEST SWAP IS 114 TH CANDIDATE FOR 88
   114 TH CANDIDATE IS NODE 114
OBJECTIVE FUNCTION IS 241115
BEST SWAP IS 47 TH CANDIDATE FOR 24
   47 TH CANDIDATE IS NODE 47
OBJECTIVE FUNCTION IS 237656
BEST SWAP IS 13 TH CANDIDATE FOR 8
   13 TH CANDIDATE IS NODE 13
OBJECTIVE FUNCTION IS 233014
BEST SWAP IS 43 TH CANDIDATE FOR 30
   43 TH CANDIDATE IS NODE 43
OBJECTIVE FUNCTION IS 230151
BEST SWAP IS 77 TH CANDIDATE FOR 119
   77 TH CANDIDATE IS NODE 77
OBJECTIVE FUNCTION IS  228966
BEST SWAP IS 35 TH CANDIDATE FOR 45
   35 TH CANDIDATE IS NODE 35
OBJECTIVE FUNCTION IS 228242
NO BETTER SOLUTION EXISTS
SOLUTION IS:  13  47  43  35  83  114  95  77  145
Stop at line 2910

Ready
```

Figure 7: Example run of EVAL

```
Ready

RUN
NEVAL  17:33      31-JUL-83
NAME OF INPUT FILE?  DIST. 4
NUMBER OF SITES, OR 0 TO NEXT EXIT?  9
SOLUTION NODE 1 ? 13
SOLUTION NODE 2 ? 47
SOLUTION NODE 3 ? 43
SOLUTION NODE 4 ? 35
SOLUTION NODE 5 ? 83
SOLUTION NODE 6 ? 114
SOLUTION NODE 7 ? 95
SOLUTION NODE 8 ? 77
SOLUTION NODE 9 ? 145
OUTPUT OPTIONS:
       1  SUMMARY ONLY
       2  ALLOCATION AND SUMMARY
OPTION?  2
```

NODE	WEIGHT	CANDIDATE	NEAREST	DISTANCE	SECOND	DISTANCE
1	5	YES	13	289	43	461
2	6	YES	13	175	43	313
3	8	YES	43	212	13	324
4	17	YES	35	260	43	357
5	4	YES	43	235	13	302
6	11	YES	43	169	13	281
7	16	YES	13	203	83	462
8	22	YES	13	88	43	244
9	13	YES	13	112	43	220
10	4	YES	43	126	13	238
11	6	YES	35	170	43	274

Figure 7: continued

NODE	WEIGHT	CANDIDATE	NEAREST	DISTANCE	SECOND	DISTANCE
12	16	YES	13	120	83	379
13	49	YES	13	0	83	269
14	19	YES	43	127	83	187
15	8	YES	43	101	83	213
16	27	YES	43	42	35	230
17	19	YES	35	131	43	142
18	37	YES	35	85	43	189
19	29	YES	77	168	35	233
20	30	YES	43	84	35	272
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
141	30	YES	95	48	47	271
142	17	YES	95	119	145	280
143	7	YES	95	199	145	200
144	27	YES	145	163	95	236
145	22	YES	145	0	83	310
146	12	YES	145	58	114	360
147	17	YES	145	144	114	274
148	15	YES	145	143	83	453
149	27	YES	145	135	114	437
150	16	YES	145	280	114	410

UNSERVED DEMAND 0

DEMAND WITH NO ALTERNATE SITE 0

SITE 1 IS AT NODE 13

DEMAND ALLOCATED 203

DISTANCE TRAVELLED 20083

MEAN DISTANCE 98.931

MAXIMUM DISTANCE 289

FURTHEST NODE ALLOCATED 1

DISTANCE TRAVELLED TO NEAREST ALTERNATE 52730

Figure 7: continued

MEAN DISTANCE TO NEAREST ALTERNATE 259.754
MAXIMUM DISTANCE TO NEAREST ALTERNATE 462
FURTHEST NODE FROM AN ALTERNATE 7

SITE 2 IS AT NODE 47
DEMAND ALLOCATED 132
DISTANCE TRAVELLED 12223
MEAN DISTANCE 92.5985
MAXIMUM DISTANCE 271
FURTHEST NODE ALLOCATED 39
DISTANCE TRAVELLED TO NEAREST ALTERNATE 32708
MEAN DISTANCE TO NEAREST ALTERNATE 247.788
MAXIMUM DISTANCE TO NEAREST ALTERNATE 350
FURTHEST NODE FROM AN ALTERNATE 23

SITE 3 AT NODE 43
DEMAND ALLOCATED 280
DISTANCE TRAVELLED 23720
MEAN DISTANCE 84.7143
MAXIMUM DISTANCE 235
FURTHEST NODE ALLOCATED 5
DISTANCE TRAVELLED TO NEAREST ALTERNATE 53120
MEAN DISTANCE TO NEAREST ALTERNATE 189.714
MAXIMUM DISTANCE TO NEAREST ALTERNATE 324
FURTHEST NODE FROM AN ALTERNATE 3

SITE 4 IS AT NODE 35
DEMAND ALLOCATED 210

Figure 7: continued

DISTANCE TRAVELLED 19811
MEAN DISTANCE 94.3381
MAXIMUM DISTANCE 260
FURTHEST NODE ALLOCATED 4
DISTANCE TRAVELLED TO NEAREST ALTERNATE 37217
MEAN DISTANCE TO NEAREST ALTERNATE 177.224
MAXIMUM DISTANCE TO NEAREST ALTERNATE 357
FURTHEST NODE FROM AN ALTERNATE 4

SITE 5 IS AT NODE 83
DEMAND ALLOCATED 700
DISTANCE TRAVELLED 53807
MEAN DISTANCE 76.8671
MAXIMUM DISTANCE 240
FURTHEST NODE ALLOCATED 53
DISTANCE TRAVELLED TO NEAREST ALTERNATE 129076
MEAN DISTANCE TO NEAREST ALTERNATE 184.394
MAXIMUM DISTANCE TO NEAREST ALTERNATE 249
FURTHEST NODE FROM AN ALTERNATE 81

SITE 6 IS AT NODE 114
DEMAND ALLOCATED 400
DISTANCE TRAVELLED 37221
MEAN DISTANCE 93.0525
MAXIMUM DISTANCE 296
FURTHEST NODE ALLOCATED 131
DISTANCE TRAVELLED TO NEAREST ALTERNATE 78707
MEAN DISTANCE TO NEAREST ALTERNATE 196.768
MAXIMUM DISTANCE TO NEAREST ALTERNATE 409

Figure 7: continued

FURTHEST NODE FROM AN ALTERNATE 131

SITE 7 IS AT NODE 95
DEMAND ALLOCATED 166
DISTANCE TRAVELLED 16064
MEAN DISTANCE 96.7711
MAXIMUM DISTANCE 199
FURTHEST NODE ALLOCATED 143
DISTANCE TRAVELLED TO NEAREST ALTERNATE 39529
MEAN DISTANCE TO NEAREST ALTERNATE 238.127
MAXIMUM DISTANCE TO NEAREST ALTERNATE 304
FURTHEST NODE FROM AN ALTERNATE 140

SITE 8 IS AT NODE 77
DEMAND ALLOCATED 185
DISTANCE TRAVELLED 20515
MEAN DISTANCE 110.892
MAXIMUM DISTANCE 321
FURTHEST NODE ALLOCATED 138
DISTANCE TRAVELLED TO NEAREST ALTERNATE 41682
MEAN DISTANCE TO NEAREST ALTERNATE 225.308
MAXIMUM DISTANCE TO NEAREST ALTERNATE 404
FURTHEST NODE FROM AN ALTERNATE 138

SITE 9 IS AT NODE 145
DEMAND ALLOCATED 183
DISTANCE TRAVELLED 24798
MEAN DISTANCE 135.508

Figure 7: continued

```
MAXIMUM DISTANCE 280
FURTHEST NODE ALLOCATED 150
DISTANCE TRAVELLED TO NEAREST ALTERNATE 56778
MEAN DISTANCE TO NEAREST ALTERNATE 310.262
MAXIMUM DISTANCE TO NEAREST ALTERNATE 453
FURTHEST NODE FROM AN ALTERNATE 148

TOTAL DEMAND SERVED 2459
TOTAL DISTANCE TRAVELLED 228242
MEAN DISTANCE TRAVELLED 92.819
MAXIMUM DISTANCE TRAVELLED 321
FURTHEST NODE FROM ALLOCATED CENTRE 138

MOST DISPOSABLE CENTRES
     TOTAL DISTANCE CRITERION - CENTRE 4 AT NODE 35
     MAXIMUM DISTANCE         - CENTRE 3 AT NODE 43

NUMBER OF SITES, OR 0 TO EXIT?  0
Stop at line 1150

Ready
```

5.2.3 Locating fire station 10 using the p-centre solution

The p-centre problem was solved by starting with the solution of the previous problem, holding 9 sites fixed and with one at 119, and successively reducing the maximum distance constraint from an initial value of 360. Each step involved the use of HILLS to edit the distance data, followed by ALLOC. The lowest possible maximum distance constraint is 340, with the 10th site located at node 118. Total distance is correspondingly increased, to 241437 from 241157. The city's tenth fire station is now located between nodes 118 and 119.

6. REFERENCES

Church, R.L. and C.S. Revelle, 1974. The maximal covering location problem. Papers, Regional Science Association, 32, 101-118.

Dijkstra, E.W., 1959. A note on two problems in connexion with graphs. Numerische Mathematik, 1, 269-271.

Holmes, J., Williams, F.B. and L.A. Brown, 1972. Facility location under a maximum travel restriction: an example using day care facilities. Geographical Analysis, 4, 258-266.

Khumawala, B.M., 1973. An efficient algorithm for the p-median problem with maximum distance constraints. Geographical Analysis, 5, 309-321.

Morrill, R.L., 1974. Efficiency and equity of optimal location models. Antipode, 6, 41-46.

Rushton, Gerard, Goodchild, Michael F. and Laurence M. Ostresh Jr., eds., 1973. Computer programs for location-allocation problems. Department of Geography, University of Iowa, Monograph Number 6.

Teitz, M.B. and P. Bart, 1968. Heuristic methods for estimating the generalised vertex median of a weighted graph. Operations Research, 16, 955-961.

Toregas,C. and C. Revelle, 1972. Optimum location under time and distance constraints. Papers and Proceedings, Regional Science Association, 28, 133-144.

Waters, Nigel M., 1977. Methodology for Servicing the Geography of Urban Fire. Unpublished Ph.D. dissertation, Department of Geography, University of Western Ontario.

7. THE CODES

7.1 GENE

```
1000  ! GENE - GENERATES RANDOM NODES AND LINKS
1010  DIM Z(200)
1020  DIM X(200),Y(200),FLAG(200),ORDER(200,10),STRL(200)
1030  MAXN=200
1040  MAXVCY=10
1050  INPUT "NUMBER OF NODES";N
1060  IF N<=MAXN THEN 1090
1070  PRINT "TOO MANY NODES - LIMIT IS";MAXN
1080  GO TO 1050
1090  PRINT "NODES LOCATED RANDOMLY IN A 100 BY 100 AREA"
1100  INPUT "WHAT PROPORTION ARE CANDIDATES";PROP
1110  PRINT "WEIGHTS ASSIGNED RANDOMLY BETWEEN 0 AND 100"
1120  INPUT "NUMBER OF LINKS PER NODE";LPN
1130  IF LPN<=MAXVCY THEN 1170
1140  PRINT "TOO MANY LINKS - LIMIT IS";MAXVCY
1150  GO TO 1090
1160  ! GET REGIONS
1170  PRINT "DEFINE THE REGIONS AS A SUPERIMPOSED GRID"
1180  INPUT "NUMBER OF DIVISIONS OF X AXIS";NH
1190  INPUT "NUMBER OF DIVISIONS OF Y AXIS";NV
1200  R=NH*NV
1210  PRINT "THERE WILL BE";R;"REGIONS"
1220  ! DEFINE NODES FILE
1230  INPUT "NAME OF NODES FILE";FILE1$
1240  OPEN FILE1$ FOR OUTPUT AS FILE 1
1250  ! DEFINE LINKS FILE
1260  INPUT "NAME OF LINKS FILE";FILE2$
1270  OPEN FILE2$ FOR OUTPUT AS FILE 2
1280  PRINT "ENTER 1 FOR RANDOM LINKS (SPA2/SPA4)"
1290  INPUT "OR 2 FOR ORDERED LINKS (SPA1/SPA3)";OPTION
1300  ! GENERATE COORDINATES AND ZERO STRING LENGTHS
1310    FOR KK=1 TO N
1320    STRL(KK)=0
1330    X(KK)=RND*100
1340    Y(KK)=RND*100
1350    NEXT KK
1360  ! LOOP FOR EACH ORIGIN
1370    FOR KK=1 TO N
1380  ! FIND DISTANCES TO EACH DESTINATION
1390      FOR KL=1 TO N
1400      Z(KL)=SQR((X(KK)-X(KL))**2+(Y(KK)-Y(KL))**2)
1410    ! ROUND DISTANCE TO INTEGER
1420      Z(KL)=INT(Z(KL)+0.5)
1430    ! SET NODE UNREACHED
1440      FLAG(KL)=0
1450      NEXT KL
1460  ! LOOP FOR EACH LINK ASSIGNED
1470      FOR LINK=1 TO LPN
1480      ZMIN=1.0E20
1490    ! FIND NEXT CLOSEST NODE
1500        FOR KL=1 TO N
1510      ! IS NODE ITSELF?
1520        IF KK=KL THEN 1600
1530      ! IS NODE ALREADY REACHED?
1540        IF FLAG(KL)=1 THEN 1600
1550      ! IS NODE CLOSER THAN CURRENT CLOSEST?
1560        IF Z(KL)>=ZMIN THEN 1600
1570      ! NODE IS CLOSEST
1580        ZMIN=Z(KL)
1590        REACH=KL
1600        NEXT KL
1610    ! FLAG NODE AS REACHED
1620      FLAG(REACH)=1
1630    ! INCREMENT STRING LENGTH FOR ORIGIN
1640      STRL(KK)=STRL(KK)+1
1650    ! CHECK STRING LENGTH
1660      IF STRL(KK)<=MAXVCY THEN 1700
1670      PRINT "TOO MANY NODES IN OUTPUT STRING - LIMIT IS";MAXVCY
1680      STOP
1690    ! STORE LINK WITH ORIGIN
1700      ORDER(KK,STRL(KK))=REACH
1710    ! INCREMENT STRING LENGTH FOR DESTINATION
1720      STRL(REACH)=STRL(REACH)+1
1730    ! CHECK STRING LENGTH
1740      IF STRL(REACH)<=MAXVCY THEN 1780
1750      PRINT "TOO MANY NODES IN OUTPUT STRING - LIMIT IS";MAXVCY
1760      STOP
1770    ! STORE LINK WITH DESTINATION
1780      ORDER(REACH,STRL(REACH))=KK
1790      NEXT LINK
1800    NEXT KK
1810  ! BEGIN OUTPUT OF NODES
1820    FOR KK=1 TO N
1830  ! ASSIGN REGION - NUMBER REGIONS FROM TOP LEFT IN ROWS
1840    REGION=INT(X(KK)/100*NH)+1+INT((100-Y(KK))/100*NV)*NH
1850  ! ASSIGN CANDIDACY
1860    CANDIDATE=0
1870    IF RND>PROP THEN 1890
1880    CANDIDATE=1
1890  ! ASSIGN WEIGHT
1900    WEIGHT=INT(RND*100)
1910    PRINT #1,KK;",";REGION;",";WEIGHT;",";CANDIDATE;",";X(KK);",";Y(KK)
1920    NEXT KK
1930  CLOSE 1
1940  ! BEGIN OUTPUT OF LINKS
1950  IF OPTION=2 THEN 2180
1960    FOR KK=1 TO N
1970  ! SET ALL FLAGS TO ZERO
1980      FOR KL=1 TO N
```

```
1990      FLAG(KL)=0
2000      NEXT KL
2010      FOR L=1 TO STRL(K)
2020      ORIG=KK
2030      DEST=ORDER(KK,L)
2040    ! SKIP IF ALREADY OUTPUT
2050      IF FLAG(DEST)=1 THEN 2130
2060    ! REMOVE DOUBLE COUNTING
2070      IF DEST<=ORIG THEN 2130
2080      DIST=SQR((X(ORIG)-X(DEST))**2+(Y(ORIG)-Y(DEST))**2)
2090    ! ROUND DISTANCE TO INTEGER
2100      DIST=INT(DIST+0.5)
2110      PRINT #2,ORIG;",";DEST;",";DIST
2120      FLAG(DEST)=1
2130      NEXT L
2140    NEXT KK
2150  ! FINISHED
2160  CLOSE 2
2170  STOP
2180  ! COME HERE FOR ORDERED OUTPUT
2190    FOR KK=1 TO N
2200  ! SET ALL FLAGS TO ZERO
2210      FOR KL=1 TO N
2220      FLAG(KL)=0
2230      NEXT KL
2240  ! LOOP FOR EACH ENTRY IN STRING
2250    NEWSTRING=0
2260      FOR L=1 TO STRL(KK)
2270      DEST=ORDER(KK,L)
2280    ! SKIP IF ALREADY OUTPUT
2290      IF FLAG(DEST)=1 THEN 2340
2300    ! COMPRESS LIST
2310      NEWSTRING=NEWSTRING+1
2320      ORDER(KK,NEWSTRING)=DEST
2330      FLAG(DEST)=1
2340      NEXT L
2350  ! OUTPUT STRING LENGTH
2360    PRINT #2,KK;",";NEWSTRING
2370    IF NEWSTRING=0 THEN 2360
2380    PRINT #2,ORDER(KK,1);
2390    IF NEWSTRING=1 THEN 2430
2400      FOR L=2 TO NEWSTRING
2410      PRINT #2,",";ORDER(KK,L);
2420      NEXT L
2430    PRINT #2
2440    DIST=SQR((X(KK)-X(ORDER(KK,1)))**2+(Y(KK)-Y(ORDER(KK,1)))**2)
2450  ! ROUND DISTANCE TO INTEGER
2460    DIST=INT(DIST+0.5)
2470    PRINT #2,DIST;
2480    IF NEWSTRING=1 THEN 2550
2490      FOR L=2 TO NEWSTRING
2500      DIST=SQR((X(KK)-X(ORDER(KK,L)))**2+(Y(KK)-Y(ORDER(KK,L)))**2)
2510    ! ROUND DISTANCE TO INTEGER
2520      DIST=INT(DIST+0.5)
2530      PRINT #2,",";DIST;
2540      NEXT L
2550    PRINT #2
2560    NEXT KK
2570  ! FINISHED
2580  CLOSE 2
2590  STOP
2600  END

Task?
```

7.2 SPAT

```
1000  ! SPA1 - SPA WITH ORDERED LINKS IN CORE
1010  DIM COREDE(1000),COREDI(1000),Z(500),RTO(200)
1020  DIM RDIST(200),WEIGHT(500),CANDIDATE(500),INNODE(25)
1030  DIM INDIST(25),ADDRESS(500),LENG(500),ORDER(100)
1040  MAXN=500
1050  MAXCORE=1000
1060  MAXLIST=200
1070  MAXLEN=100
1080  MAXVCY=25
1090  INPUT "NUMBER OF NODES";N
1100  IF N<=MAXN THEN 1130
1110  PRINT "TOO MANY NODES - LIMIT IS";MAXN
1120  GO TO 1090
1130  ! GET FILE OF NODES
1140  INPUT "NAME OF NODES FILE";FILE1$
1150  OPEN FILE1$ FOR INPUT AS FILE 1
1160    FOR KK=1 TO N
1170    INPUT #1,KREAD,DUMMY,WEIGHT(KK),CANDIDATE(KK),DUMMY,DUMMY
1180    IF KK=KREAD THEN 1200
1190    PRINT "FILE ";FILE1$;" IS OUT OF SEQUENCE AT RECORD ";KK
1200    NEXT KK
1210  CLOSE 1
1220  ! GET DISTANCE LIMIT
1230  INPUT "VALUE FOR ZLIMIT";ZLIMIT
1240  ! GET FILE OF LINKS
1250  INPUT "NAME OF LINKS FILE";FILE2$
1260  OPEN FILE2$ FOR INPUT AS FILE 2
1270  NLINKS=0
1280  ! LOOP FOR EACH NODE
1290    FOR KK=1 TO N
1300    INPUT #2,KREAD,VALENCY
1310    IF KK=KREAD THEN 1340
1320    PRINT "FILE ";FILE2$;" IS OUT OF SEQUENCE AT RECORD";KK
1330  ! ARE NODES WITHIN VALENCY LIMIT?
1340    IF VALENCY<=MAXVCY THEN 1370
1350    PRINT "VALENCY FOR RECORD";KK;"EXCEEDS MAXIMUM OF";MAXVCY
1360    STOP
1370  ! IS THERE ROOM IN CORE?
1380    IF NLINKS+VALENCY<=MAXCORE THEN 1410
1390    PRINT "CORE LIMIT OF";MAXCORE;"EXCEEDED AT NODE";KK
1400    STOP
1410    MAT INPUT #2,INNODE(VALENCY)
1420    MAT INPUT #2,INDIST(VALENCY)
1430  ! PUT DATA IN CORE FOR THIS NODE
1440      FOR LINK=1 TO VALENCY
1450      NLINKS=NLINKS+1
1460      COREDE(NLINKS)=INNODE(LINK)
1470    ! CHECK DOMAIN OF NODE NUMBERS
1480      IF COREDE(NLINKS)>0 AND COREDE(NLINKS)<=MAXN THEN 1510
1490      PRINT "NODE";COREDE(NLINKS);"IN LINK RECORD";KK;"IS OUTSIDE DOMAIN"
1500      STOP
1510      COREDI(NLINKS)=INDIST(LINK)
1520      NEXT LINK
1530  ! SET POINTERS FOR THIS NODE'S LINKS
1540    ADDRESS(KK)=NLINKS-VALENCY
1550    LENG(KK)=VALENCY
1560    NEXT KK
1570  CLOSE 2
1580  PRINT NLINKS;"LINKS READ"
1590  INPUT "NAME OF OUTPUT FILE";FILE3$
1600  OPEN FILE3$ FOR OUTPUT AS FILE 3
1610  ! LOOP FOR ALL ORIGINS
1620    FOR KK=1 TO N
1630  ! IGNORE IF NO WEIGHT
1640    IF WEIGHT(KK)=0 THEN 2500
1650  ! ZERO ALL DISTANCES
1660      FOR KL=1 TO N
1670      Z(KL)=-1
1680      NEXT KL
1690  ! ORIGIN IS REACHED
1700    Z(KK)=0
1710    REACH=KK
1720    RNUM=0
1730  ! ZERO STRING LENGTH
1740    STRL=0
1750  ! IS ORIGIN A CANDIDATE?
1760    IF CANDIDATE(KK)=0 THEN 1810
1770    STRL=1
1780    ORDER(1)=KK
1790  ! BEGIN MAJOR LOOP
1800  ! COMPRESS LIST OF REACHABLE NODES
1810    IF RNUM=0 THEN 1960
1820  ! SET NEW COMPRESSED LIST LENGTH TO ZERO
1830    RNUM1=0
1840      FOR RM=1 TO RNUM
1850    ! HAS ENTRY RM IN LIST BEEN REACHED?
1860      IF Z(RTO(RM))>=0 THEN 1920
1870    ! IF NOT INCREMENT NEW LIST
1880      RNUM1=RNUM1+1
1890    ! MAKE ENTRIES IN NEW LIST
1900      RTO(RNUM1)=RTO(RM)
1910      RDIST(RNUM1)=RDIST(RM)
1920      NEXT RM
1930  ! NEW LIST IS NOW OLD
1940    RNUM=RNUM1
1950  ! APPEND NEW REACHABLE NODES FROM CORE
1960    N1=ADDRESS(REACH)+1
1970    N2=N1-1+LENG(REACH)
1980  ! LOOP ALL LINKS INCIDENT AT REACH
```

```
1970      FOR LINK=N1 TO N2
2000    ! ALREADY REACHED?
2010      IF Z(COREDE(LINK))>=0 THEN 2090
2020      RNUM=RNUM+1
2030      IF RNUM<=MAXLIST THEN 2070
2040      PRINT 'TOO MANY REACHABLE NODES - LIMIT IS';MAXLIST
2050      STOP
2060    ! NEW DESTINATION NODE AND TOTAL COST
2070      RTO(RNUM)=COREDE(LINK)
2080      RDIST(RNUM)=COREDI(LINK)+Z(REACH)
2090      NEXT LINK
2100  ! IF EMPTY THEN FINISHED
2110    IF RNUM=0 THEN 2360
2120  ! SEARCH FOR NEAREST REACHABLE NODE
2130    ZMIN=10E30
2140  ! LOOP THROUGH EACH ENTRY IN LIST
2150      FOR RM=1 TO RNUM
2160      IF RDIST(RM)>=ZMIN THEN 2200
2170    ! NODE IS LEAST TOTAL SO FAR
2180      REACH=RTO(RM)
2190      ZMIN=RDIST(RM)
2200      NEXT RM
2210  ! IF DISTANCE > ZLIMIT THEN FINISHED
2220    IF ZMIN>ZLIMIT THEN 2360
2230  ! STORE TOTAL DISTANCE
2240    Z(REACH)=ZMIN
2250  ! IS REACHED NODE A CANDIDATE?
2260    IF CANDIDATE(REACH)=0 THEN 1810
2270    STRL=STRL+1
2280    IF STRL<=MAXLEN THEN 2320
2290    PRINT 'TOO MANY NODES IN OUTPUT STRING - LIMIT IS';MAXLEN
2300    STOP
2310  ! SAVE ORDER IN STRING
2320    ORDER(STRL)=REACH
2330  ! RETURN FOR NEW CYCLE
2340    GOTO 1810
2350  ! BEGIN OUTPUT OF STRING
2360    PRINT #3,KK;',';STRL;',';WEIGHT(KK);',';CANDIDATE(KK)
2370    IF STRL=0 THEN 2500
2380    PRINT #3,ORDER(1);
2390    IF STRL=1 THEN 2430
2400      FOR L=2 TO STRL
2410    . PRINT #3,',';ORDER(L);
2420      NEXT L
2430    PRINT #3
2440    PRINT #3,Z(ORDER(1))*WEIGHT(KK);
2450    IF STRL=1 THEN 2490
2460      FOR L=2 TO STRL
2470      PRINT #3,',';Z(ORDER(L))*WEIGHT(KK);
2480      NEXT L
2490    PRINT #3
2500    NEXT KK
2510  !FINISHED
2520  CLOSE 3
2530  STOP
2540  END

Task?
```

7.3 SPA2

```
1000  ! SPA2 - SPA WITH RANDOM LINKS IN CORE
1010  DIM WEIGHT(500),CANDIDATE(500),OG(1500),DE(1500),DI(1500)
1020  DIM ORDER(200),Z(500)
1030  MAXN=500
1040  MAXLEN=200
1050  MAXLINKS=1500
1060  INPUT "NUMBER OF NODES";N
1070  IF N<=MAXN THEN 1110
1080  PRINT "TOO MANY NODES - LIMIT IS";MAXN
1090  GO TO 1060
1100  ! GET FILE OF NODES
1110  INPUT "NAME OF NODES FILE";FILE1$
1120  OPEN FILE1$ FOR INPUT AS FILE 1
1130  ! READ RECORD FOR EACH NODE
1140    FOR KK=1 TO N
1150    INPUT #1,KREAD,DUMMY,WEIGHT(KK),CANDIDATE(KK),DUMMY,DUMMY
1160  ! CHECK SEQUENCE OF NODES
1170    IF KK=KREAD THEN 1190
1180    PRINT "FILE ";FILE1$;" IS OUT OF SEQUENCE AT RECORD";KK
1190    NEXT KK
1200  CLOSE 1
1210  ! GET DISTANCE LIMIT
1220  INPUT "VALUE FOR ZLIMIT";ZLIMIT
1230  ! GET FILE OF LINKS
1240  INPUT "NAME OF LINKS FILE";FILE2$
1250  OPEN FILE2$ FOR INPUT AS FILE 2
1260  NLINKS=0
1270  ! END OF FILE CHECK
1280  ON ERROR GO TO 2230
1290  NLINKS=NLINKS+1
1300  ! CHECK NUMBER OF LINKS
1310  IF NLINKS<=MAXLINKS THEN 1340
1320  PRINT "TOO MANY LINKS - LIMIT IS";MAXLINKS
1330  STOP
1340  INPUT #2,OG(NLINKS),DE(NLINKS),DI(NLINKS)
1350  ! CHECK DOMAIN OF NODE NUMBERS
1360  IF OG(NLINKS)>0 AND OG(NLINKS)<=MAXN THEN 1390
1370  PRINT "NODE";OG(NLINKS);"IN LINK RECORD";NLINKS;"IS OUTSIDE DOMAIN
1380  STOP
1390  IF DE(NLINKS)>0 AND DE(NLINKS)<=MAXN THEN 1290
1400  PRINT "NODE";DE(NLINKS);"IN LINK RECORD";NLINKS;"IS OUTSIDE DOMAIN
1410  STOP
1420  NLINKS=NLINKS-1
1430  PRINT NLINKS;"LINKS READ"
1440  ! FILE FOR OUTPUT STRINGS
1450  INPUT "NAME OF OUTPUT FILE";FILE3$
1460  OPEN FILE3$ FOR OUTPUT AS FILE 3
1470  ! FIND DISTANCES FOR EACH ORIGIN NODE
1480    FOR KK=1 TO N
1490  ! IGNORE IF NO WEIGHT
1500    IF WEIGHT(KK)=0 THEN 2180
1510  ! ZERO DISTANCES FOR EACH DESTINATION
1520      FOR KL=1 TO N
1530      Z(KL)=-1
1540      NEXT KL
1550  ! DISTANCE TO ORIGIN IS ZERO
1560    Z(KK)=0
1570  ! ZERO STRING LENGTH
1580    STRL=0
1590  ! IS ORIGIN A CANDIDATE?
1600    IF CANDIDATE(KK)=0 THEN 1630
1610    STRL=1
1620    ORDER(1)=KK
1630    ZMIN=1.0E20
1640  ! LOOP THROUGH EACH LINK IN CORE
1650      FOR LC=1 TO NLINKS
1660    ! IS ORIGIN ALREADY REACHED?
1670      IF Z(OG(LC))>=0 THEN 1730
1680    ! IS DESTINATION ALREADY REACHED?
1690      IF Z(DE(LC))>=0 THEN 1820
1700    ! BOTH ARE REACHED
1710      GO TO 1890
1720    ! COME HERE IF ORIGIN IS REACHED
1730      IF Z(DE(LC))>=0 THEN 1890
1740    ! COMPUTE TOTAL DISTANCE TO DESTINATION
1750      ZTOT=Z(OG(LC))+DI(LC)
1760      IF ZTOT>=ZMIN THEN 1890
1770    ! NODE IS CLOSEST SO FAR
1780      ZMIN=ZTOT
1790      REACH=DE(LC)
1800      GO TO 1890
1810    ! COME HERE IF DESTINATION IS REACHED
1820      IF Z(OG(LC))>=0 THEN 1890
1830    ! COMPUTE TOTAL DISTANCE TO ORIGIN OF THIS LINK
1840      ZTOT=Z(DE(LC))+DI(LC)
1850      IF ZTOT>=ZMIN THEN 1890
1860    ! NODE IS CLOSEST SO FAR
1870      ZMIN=ZTOT
1880      REACH=OG(LC)
1890      NEXT LC
1900  ! IF CLOSEST IS FURTHER THAN ZLIMIT FINISH THIS ORIGIN
1910    IF ZMIN>ZLIMIT THEN 2040
1920    Z(REACH)=ZMIN
1930  ! OUTPUT ONLY IF NODE IS CANDIDATE
1940    IF CANDIDATE(REACH)=0 THEN 1630
1950    STRL=STRL+1
1960  ! CHECK STRING LENGTH
1970    IF STRL<=MAXLEN THEN 2010
1980    PRINT "TOO MANY NODES IN OUTPUT STRING - LIMIT IS";MAXLEN
```

```
1990    STOP
2000  ! SAVE ORDER IN STRING
2010    ORDER(STRL)=REACH
2020    GO TO 1630
2030  ! BEGIN OUTPUT OF STRING
2040    PRINT #3,KK;",";STRL;",";WEIGHT(KK);",";CANDIDATE(KK)
2050    IF STRL=0 THEN 2160
2060    PRINT #3,ORDER(1);
2070    IF STRL=1 THEN 2110
2080      FOR L=2 TO STRL
2090      PRINT #3,",";ORDER(L);
2100      NEXT L
2110    PRINT #3
2120    PRINT #3,Z(ORDER(1))*WEIGHT(KK);
2130    IF STRL=1 THEN 2170
2140      FOR L=2 TO STRL
2150      PRINT #3,",";Z(ORDER(L))*WEIGHT(KK);
2160      NEXT L
2170    PRINT #3
2180    NEXT KK
2190  ! FINISHED
2200  CLOSE 3
2210  STOP
2220  ! COME HERE ON I/O ERROR
2230  IF ERR<>11 GO TO 2270
2240  CLOSE 2
2250  RESUME 1420
2260  ! SOME OTHER ERROR
2270  PRINT "I/O ERROR NUMBER";ERR
2280  STOP
2290  END

Task?
```

7.4 SPA3

```
1000  ! SPA3 - SPA WITH ORDERED LINKS AND REGIONS
1010  DIM REGION(500),POINT(500),COREDE(1000),CURENT(1000),Z(500)
1020  DIM RTO(200),RDIST(200),WEIGHT(500),CANDIDATE(500),INNODE(25)
1030  DIM INDIST(25),ADDRESS(100),LENG(100),ORDER(100)
1040  MAXN=500
1050  MAXCORE=1000
1060  MAXLIST=200
1070  MAXLEN=100
1080  MAXVCY=25
1090  MAXIN=100
1100  INPUT "NUMBER OF NODES";N
1110  IF N<=MAXN THEN 1140
1120  PRINT "TOO MANY NODES - LIMIT IS ";MAXN
1130  GO TO 1100
1140  INPUT "NUMBER OF REGIONS";R
1150  ! GET FILE OF NODES
1160  INPUT "NAME OF NODES FILE";FILE1$
1170  OPEN FILE1$ FOR INPUT AS FILE 1
1180    FOR KK=1 TO N
1190    INPUT #1,KREAD,REGION(KK),WEIGHT(KK),CANDIDATE(KK),DUMMY,DUMMY
1200  ! CHECK SEQUENCE OF NODES
1210    IF KK=KREAD THEN 1230
1220    PRINT "FILE ";FILE1$;" IS OUT OF SEQUENCE AT RECORD ";KK
1230    NEXT KK
1240  CLOSE 1
1250  ! GET DISTANCE LIMIT
1260  INPUT "VALUE FOR ZLIMIT";ZLIMIT
1270  ! GET FILE OF LINKS
1280  INPUT "NAME OF LINKS FILE";FILE2$
1290  ! FILE FOR OUTPUT STRINGS
1300  INPUT "NAME OF OUTPUT FILE";FILE3$
1310  OPEN FILE3$ FOR OUTPUT AS FILE 3
1320  ! LOOP FOR EACH REGION
1330  ! AND FIND ALL DISTANCES FROM NODES IN THIS REGION
1340    FOR IR=1 TO R
1350  ! ALL REGIONS OUT OF CORE
1360    INCORE=0
1370      FOR KK=1 TO N
1380    ! SET POINTERS FOR EACH NODE
1390      POINT(KK)=0
1400      NEXT KK
1410  ! READ LINKS FOR REGION IR
1420    OPEN FILE2$ FOR INPUT AS FILE 2
1430    NLINKS=0
1440  ! LOOP FOR EACH NODE
1450      FOR KK=1 TO N
1460      INPUT #2,KREAD,VALENCY
1470      IF KK=KREAD THEN 1500
1480      PRINT "FILE ";FILE2$;" IS OUT OF SEQUENCE AT RECORD ";KK
1490    ! ARE NODES WITHIN VALENCY LIMIT?
1500      IF VALENCY<=MAXVCY THEN 1530
1510      PRINT "VALENCY FOR RECORD ";KK;" EXCEEDS LIMIT OF ";MAXVCY
1520      STOP
1530    ! IS THERE ROOM IN CORE?
1540      IF NLINKS+VALENCY<=MAXCORE THEN 1570
1550      PRINT "CORE LIMIT OF";MAXCORE;"EXCEEDED AT NODE";KK
1560      STOP
1570      MAT INPUT #2,INNODE(VALENCY)
1580      MAT INPUT #2,INDIST(VALENCY)
1590    ! IS THIS REGION TO GO IN CORE?
1600      IF REGION(KK)<>IR THEN 1800
1610    ! PUT DATA IN CORE FOR THIS NODE
1620        FOR LINK=1 TO VALENCY
1630        NLINKS=NLINKS+1
1640        COREDE(NLINKS)=INNODE(LINK)
1650      ! CHECK DOMAIN OF NODE NUMBERS
1660        IF COREDE(NLINKS)>0 AND COREDE(NLINKS)<=MAXN THEN 1690
1670        PRINT "NODE";COREDE(NLINKS);"IN LINK RECORD";KK;"IS OUTSIDE DOMAIN"
1680        STOP
1690        CORED1(NLINKS)=INDIST(LINK)
1700        NEXT LINK
1710    ! INCREMENT NUMBER IN CORE
1720      INCORE=INCORE+1
1730      IF INCORE<=MAXIN THEN 1770
1740      PRINT "TOO MANY NODES IN CORE - LIMIT IS ";MAXIN
1750      STOP
1760    ! SET POINTERS FOR THIS NODE'S LINKS
1770      POINT(KK)=INCORE
1780      ADDRESS(INCORE)=NLINKS-VALENCY
1790      LENG(INCORE)=VALENCY
1800      NEXT KK
1810    CLOSE 2
1820    PRINT NLINKS;"LINKS READ FOR REGION";IR
1830  ! LOOP FOR ALL ORIGINS IN THIS REGION
1840      FOR KK=1 TO N
1850    ! IGNORE IF NO WEIGHT
1860      IF WEIGHT(KK)=0 THEN 3010
1870      IF REGION(KK)<>IR THEN 3010
1880    ! ZERO ALL DISTANCES
1890        FOR KL=1 TO N
1900        Z(KL)=-1
1910        NEXT KL
1920    ! ORIGIN IS REACHED
1930      Z(KK)=0
1940      REACH=KK
1950      RNUM=0
1960    ! ZERO SINING LENGTH
1970      SIN1=0
1980    ! IS ORIGIN A CANDIDATE
```

```
1990      IF CANDIDATE(KK)=0 THEN 2040
2000      STRL=1
2010      ORDER(1)=KK
2020    ! BEGIN MAJOR LOOP
2030    ! COMPRESS LIST OF REACHABLE NODES
2040      IF RNUM=0 THEN 2190
2050    ! SET NEW COMPRESSED LIST LENGTH TO ZERO
2060      RNUM1=0
2070        FOR RM=1 TO RNUM
2080      ! HAS ENTRY RM IN LIST BEEN REACHED?
2090        IF Z(RTO(RM))>=0 THEN 2150
2100      ! IF NOT INCREMENT NEW LIST
2110        RNUM1=RNUM1+1
2120      ! MAKE ENTRIES IN NEW LIST
2130        RTO(RNUM1)=RTO(RM)
2140        RDIST(RNUM1)=RDIST(RM)
2150        NEXT RM
2160    ! NEW LIST IS NOW OLD
2170      RNUM=RNUM1
2180    ! APPEND NEW REACHABLE NODES FROM CORE
2190      N1=ADDRESS(POINT(REACH))+1
2200      N2=ADDRESS(POINT(REACH))+LENG(POINT(REACH))
2210    ! LOOP ALL LINKS INCIDENT AT REACH
2220        FOR LINK=N1 TO N2
2230      ! ALREADY REACHED?
2240        IF Z(COREDE(LINK))>=0 THEN 2320
2250        RNUM=RNUM+1
2260        IF RNUM<=MAXLIST THEN 2300
2270        PRINT "TOO MANY REACHABLE NODES - LIMIT IS ";MAXLIST
2280        STOP
2290      ! NEW DESTINATION NODE AND TOTAL COST
2300        RTO(RNUM)=COREDE(LINK)
2310        RDIST(RNUM)=COREDI(LINK)+Z(REACH)
2320        NEXT LINK
2330    ! IF EMPTY THEN FINISHED
2340      IF RNUM=0 THEN 2870
2350    ! SEARCH FOR NEAREST REACHABLE NODE
2360      ZMIN=10E30
2370    ! LOOP THROUGH EACH ENTRY IN LIST
2380        FOR RM=1 TO RNUM
2390        IF RDIST(RM)>=ZMIN THEN 2430
2400      ! NODE IS LEAST TOTAL SO FAR
2410        REACH=RTO(RM)
2420        ZMIN=RDIST(RM)
2430        NEXT RM
2440    ! IF DISTANCE > ZLIMIT THEN FINISHED
2450      IF ZMIN>ZLIMIT THEN 2870
2460    ! STORE TOTAL DISTANCE
2470      Z(REACH)=ZMIN
2480    ! IS REACHED NODE A CANDIDATE?
2490      IF CANDIDATE(REACH)=0 THEN 2570
2500      STRL=STRL+1
2510      IF STRL<=MAXLEN THEN 2550
2520      PRINT "TOO MANY NODES IN OUTPUT STRING - LIMIT IS ";MAXLEN
2530      STOP
2540    ! SAVE ORDER IN STRING
2550      ORDER(STRL)=REACH
2560    ! IS REACHED NODE IN CORE?
2570      IF POINT(REACH)<>0 THEN 2040
2580    ! APPEND NEW REGION
2590      NEWREG=REGION(REACH)
2600      PRINT "APPENDING REGION";NEWREG
2610      OPEN FILE2$ FOR INPUT AS FILE 2
2620        FOR KL=1 TO N
2630        INPUT #2,KREAD,VALENCY
2640        MAT INPUT #2,INNODE(VALENCY)
2650        MAT INPUT #2,INDIST(VALENCY)
2660        IF REGION(KL)<>NEWREG THEN 2830
2670      ! IS THERE ROOM?
2680        IF NLINKS+VALENCY<=MAXCORE THEN 2710
2690        PRINT "CORE LIMIT OF ";MAXCORE;" EXCEEDED AT NODE ";KL
2700        STOP
2710          FOR LINK=1 TO VALENCY
2720          NLINKS=NLINKS+1
2730          COREDE(NLINKS)=INNODE(LINK)
2740          COREDI(NLINKS)=INDIST(LINK)
2750          NEXT LINK
2760        INCORE=INCORE+1
2770        IF INCORE<=MAXIN THEN 2800
2780        PRINT "TOO MANY NODES IN CORE - LIMIT IS ";MAXIN
2790        STOP
2800        POINT(KL)=INCORE
2810        ADDRESS(INCORE)=NLINKS-VALENCY
2820        LENG(INCORE)=VALENCY
2830        NEXT KL
2840      CLOSE 2
2850      GOTO 2040
2860    ! BEGIN OUTPUT OF STRING
2870      PRINT #3, KK;",";STRL;",";WEIGHT(KK);",";CANDIDATE(KK)
2880      IF STRL=0 THEN 3010
2890      PRINT #3,ORDER(1);
2900      IF STRL=1 THEN 2940
2910        FOR L=2 TO STRL
2920        PRINT #3,",";ORDER(L);
2930        NEXT L
2940      PRINT #3
2950      PRINT #3,Z(ORDER(1))*WEIGHT(KK);
2960      IF STRL=1 THEN 3000
2970        FOR L=2 TO STRL
2980        PRINT #3,",";Z(ORDER(L))*WEIGHT(KK);
2990        NEXT L
3000      PRINT #3
```

```
3010      NEXT KK
3020  ! FINISHED THIS REGION
3030    NEXT IR
3040  ! FINISHED
3050  CLOSE 3
3060  STOP
3070  END

Task?
```

7.5 SPA4

```
1000  ! SPA4 - SPA WITH RANDOM LINKS AND REGIONS
1010  DIM REGION(500),INREG(25),OG(1000),DE(1000),DI(1000)
1020  DIM ORDER(200),Z(500),WEIGHT(500),CANDIDATE(500)
1030  MAXN=500
1040  MAXREG=25
1050  MAXLINKS=1000
1060  MAXLEN=200
1070  INPUT "NUMBER OF NODES";N
1080  IF N<=MAXN THEN 1110
1090  PRINT "TOO MANY NODES - LIMIT IS";MAXN
1100  GO TO 1070
1110  INPUT "NUMBER OF REGIONS";R
1120  IF R<=MAXREG THEN 1160
1130  PRINT "TOO MANY REGIONS - LIMIT IS";MAXREG
1140  STOP
1150  ! GET FILE OF NODES
1160  INPUT "NAME OF NODES FILE";FILE1$
1170  OPEN FILE1$ FOR INPUT AS FILE 1
1180  ! READ RECORD FOR EACH NODE
1190    FOR KK=1 TO N
1200    INPUT #1,KREAD,REGION(KK),WEIGHT(KK),CANDIDATE(KK),DUMMY,DUMMY
1210  ! CHECK SEQUENCE OF NODES
1220    IF KK=KREAD THEN 1240
1230    PRINT "FILE ";FILE1$;" IS OUT OF SEQUENCE AT RECORD";KK
1240    NEXT KK
1250  CLOSE 1
1260  ! GET DISTANCE LIMIT
1270  INPUT "VALUE FOR ZLIMIT";ZLIMIT
1280  ! GET FILE OF LINKS
1290  INPUT "NAME OF LINKS FILE";FILE2$
1300  ! FILE FOR OUTPUT STRINGS
1310  INPUT "NAME OF OUTPUT FILE";FILE3$
1320  OPEN FILE3$ FOR OUTPUT AS FILE 3
1330  ! LOOP FOR EACH REGION
1340  ! AND FIND ALL DISTANCES FROM NODES IN THIS REGION
1350    FOR IR=1 TO R
1360  ! ALL REGIONS OUT OF CORE
1370      FOR JR=1 TO R
1380      INREG(JR)=0
1390      NEXT JR
1400  ! READ LINKS FOR REGION IR
1410    OPEN FILE2$ FOR INPUT AS FILE 2
1420    NLINKS=0
1430  ! REGION IR IN CORE
1440    INREG(IR)=1
1450  ! END OF FILE CHECK
1460    ON ERROR GOTO 2730
1470    INPUT #2, ORIG,DEST,DIST
1480  ! CHECK DOMAIN OF NODE NUMBERS
1490    IF ORIG>0 AND ORIG<=MAXN THEN 1520
1500    PRINT "NODE";ORIG;"IN LINK RECORD";NLINKS;"IS OUTSIDE DOMAIN"
1510    STOP
1520    IF DEST>0 AND DEST<=MAXN THEN 1560
1530    PRINT "NODE";DEST;"IN LINK RECORD";NLINKS;"IS OUTSIDE DOMAIN"
1540    STOP
1550  ! IF ORIGIN REGION IS IN CORE THEN LOAD LINK
1560    IF REGION(ORIG)=IR THEN 1590
1570  ! IF ORIGIN AND DESTINATION REGIONS ARE NOT IN CORE BYPASS
1580    IF REGION(DEST)<>IR THEN 1470
1590    NLINKS=NLINKS+1
1600  ! CHECK NUMBER OF LINKS
1610    IF NLINKS<=MAXLINKS THEN 1640
1620    PRINT "TOO MANY LINKS IN CORE - LIMIT IS";MAXLINKS
1630    STOP
1640    OG(NLINKS)=ORIG
1650    DE(NLINKS)=DEST
1660    DI(NLINKS)=DIST
1670    GOTO 1460
1680    PRINT NLINKS;"LINKS READ FOR REGION";IR
1690  ! LOOP FOR ALL ORIGINS IN THIS REGION
1700      FOR KK=1 TO N
1710      ! IGNORE IF NO WEIGHT
1720      IF WEIGHT(KK)=0 THEN 2660
1730      IF REGION(KK)<>IR THEN 2660
1740      ! ZERO ALL DISTANCES
1750        FOR KL=1 TO N
1760        Z(KL)=-1
1770        NEXT KL
1780      ! ORIGIN IS REACHED
1790        Z(KK)=0
1800        REACH=KK
1810      ! ZERO STRING LENGTH
1820        STRL=0
1830      ! IS ORIGIN A CANDIDATE?
1840        IF CANDIDATE(KK)=0 THEN 1870
1850        STRL=1
1860        ORDER(1)=KK
1870        ZMIN=1.0E30
1880      ! LOOP THROUGH EACH LINK IN CORE
1890          FOR LC=1 TO NLINKS
1900          ! IS ORIGIN ALREADY REACHED?
1910            IF Z(OG(LC))>=0 THEN 1970
1920          ! IS DESTINATION ALREADY REACHED?
1930            IF Z(DE(LC))>=0 THEN 2060
1940          ! BOTH ARE REACHED
1950            GO TO 2130
1960          ! COME HERE IF ORIGIN IS REACHED
1970            IF Z(DE(LC))>=0 THEN 2130
1980          ! COMPUTE TOTAL DISTANCE TO DESTINATION
```

```
1990        ZTOT=Z(OG(LC))+DI(LC)
2000        IF ZTOT>=ZMIN THEN 2130
2010      ! NODE IS CLOSEST SO FAR
2020        ZMIN=ZTOT
2030        REACH=DE(LC)
2040        GO TO 2130
2050      ! COME HERE IF DESTINATION IS REACHED
2060        IF Z(OG(LC))>=0 THEN 2130
2070      ! COMPUTE TOTAL DISTANCE TO ORIGIN OF THIS LINK
2080        ZTOT=Z(DE(LC))+DI(LC)
2090        IF ZTOT>=ZMIN THEN 2130
2100      ! NODE IS CLOSEST SO FAR
2110        ZMIN=ZTOT
2120        REACH=OG(LC)
2130        NEXT LC
2140    ! IF CLOSEST IS FURTHER THAN ZLIMIT FINISH THIS ORIGIN
2150      IF ZMIN>ZLIMIT THEN 2520
2160      Z(REACH)=ZMIN
2170    ! OUTPUT ONLY IF NODE IS A CANDIDATE
2180      IF CANDIDATE(REACH)=0 THEN 2270
2190      STRL=STRL+1
2200    ! CHECK STRING LENGTH
2210      IF STRL<=MAXLEN THEN 2240
2220      PRINT "TOO MANY NODES IN OUTPUT STRING - LIMIT IS";MAXLEN
2230      STOP
2240    ! SAVE ORDER IN STRING
2250      ORDER(STRL)=REACH
2260    ! IS REACHED NODE IN CORE?
2270      IF INREG(REGION(REACH))=1 THEN 1870
2280    ! APPEND ANOTHER REGION
2290      NEWREG=REGION(REACH)
2300      PRINT "APPENDING REGION";NEWREG
2310      OPEN FILE2$ FOR INPUT AS FILE 2
2320      ON ERROR GO TO 2800
2330      INPUT #2, ORIG,DEST,DIST
2340      IF REGION(ORIG)<>NEWREG THEN 2430
2350    ! IS LINK ALREADY IN CORE?
2360      IF INREG(REGION(DEST))=1 THEN 2330
2370      NLINKS=NLINKS+1
2380      IF NLINKS>MAXLINKS THEN 2490
2390      OG(NLINKS)=ORIG
2400      DE(NLINKS)=DEST
2410      DI(NLINKS)=DIST
2420      GO TO 2330
2430      IF REGION(DEST)<>NEWREG THEN 2330
2440      IF INREG(REGION(ORIG))=1 THEN 2330
2450      GO TO 2370
2460    ! REGION IS NOW IN CORE
2470      INREG(NEWREG)=1
2480      GO TO 1870
2490      PRINT "TOO MANY LINKS IN CORE - LIMIT IS";MAXLINKS
2500      STOP
2510    ! BEGIN OUTPUT OF STRING
2520      PRINT #3,KK;",";STRL;",";WEIGHT(KK);",";CANDIDATE(KK)
2530      IF STRL=0 THEN 2660
2540      PRINT #3, ORDER(1);
2550      IF STRL=1 THEN 2590
2560        FOR L=2 TO STRL
2570        PRINT #3,",";ORDER(L);
2580        NEXT L
2590      PRINT #3
2600      PRINT #3, Z(ORDER(1))*WEIGHT(KK);
2610      IF STRL=1 THEN 2650
2620        FOR L=2 TO STRL
2630        PRINT #3,",";Z(ORDER(L))*WEIGHT(KK);
2640        NEXT L
2650      PRINT #3
2660      NEXT KK
2670  ! FINISHED THIS REGION
2680    NEXT IR
2690  ! FINISHED
2700  CLOSE 3
2710  STOP
2720  ! COME HERE ON ERROR
2730  IF ERR<>11 GO TO 2770
2740  CLOSE 2
2750  RESUME 1600
2760  ! SOME OTHER ERROR
2770  PRINT "I/O ERROR NUMBER";ERR
2780  STOP
2790  ! COME HERE ON ERROR
2800  IF ERR<>11 GOTO 2840
2810  CLOSE 2
2820  RESUME 2470
2830  ! SOME OTHER ERROR
2840  PRINT "I/O ERROR NUMBER";ERR
2850  STOP
2860  END

Task?
```

7.6 SPA5

```
100  ! SPA5 - STRAIGHT LINE DISTANCES FROM COORDINATES
110  DIM X(500),Y(500),WEIGHT(500),CANDIDATE(500),Z(500)
120  DIM ORDER(200),FLAG(500)
130  MAXN=500
140  MAXLEN=200
150  INPUT "NUMBER OF NODES";N
160  MAXLEN=200
170  IF N<=MAXN THEN 210
180  PRINT "TOO MANY NODES - LIMIT IS";MAXN
190  GO TO 150
200  ! GET FILE OF NODES
210  INPUT "NAME OF NODES FILE";FILE1$
220  OPEN FILE1$ FOR INPUT AS FILE 1
230  ! READ RECORD FOR EACH NODE
240    FOR KK=1 TO N
250    INPUT #1,KREAD,DUMMY,WEIGHT(KK),CANDIDATE(KK),X(KK),Y(KK)
260  ! CHECK SEQUENCE OF NODES
270    IF KK=KREAD THEN 290
280    PRINT "FILE ";FILE1$;" IS OUT OF SEQUENCE AT RECORD";KK
290    NEXT KK
300  CLOSE 1
310  ! GET DISTANCE LIMIT
320  INPUT "VALUE OF ZLIMIT";ZLIMIT
330  ! FILE FOR OUTPUT STRINGS
340  INPUT "NAME OF OUTPUT FILE";FILE3$
350  OPEN FILE3$ FOR OUTPUT AS FILE 3
360  ! FIND DISTANCES FOR EACH ORIGIN NODE
370    FOR KK=1 TO N
380  ! IGNORE IF NO WEIGHT
390    IF WEIGHT(KK)=0 THEN 880
400  ! LOOP FOR EACH DESTINATION
410      FOR KL=1 TO N
420      Z(KL)=SQR((X(KK)-X(KL))**2+(Y(KK)-Y(KL))**2)
430    ! ROUND DISTANCE TO INTEGER
440      Z(KL)=INT(Z(KL)+0.5)
450    ! SET NODE UNREACHED
460      FLAG(KL)=0
470      NEXT KL
480  ! ZERO STRING LENGTH
490    STRL=0
500  ! FIND ITH CLOSEST CANDIDATE NODE
510      FOR I=1 TO N
520      ZMIN=1.0E20
530    ! LOOP THROUGH EACH DESTINATION
540        FOR KL=1 TO N
550      ! IS NODE ALREADY REACHED?
560        IF FLAG(KL)=1 THEN 640
570      ! IS NODE A CANDIDATE?
580        IF CANDIDATE(KL)=0 THEN 640
590      ! IS NODE CLOSER THAN CURRENT CLOSEST?
600        IF Z(KL)>=ZMIN THEN 640
610      ! NODE IS CLOSEST
620        ZMIN=Z(KL)
630        REACH=KL
640        NEXT KL
650    ! IF CLOSEST IS FURTHER THAN ZLIMIT FINISH THIS ORIGIN
660      IF ZMIN>ZLIMIT THEN 730
670    ! FLAG NODE AS REACHED
680      FLAG(REACH)=1
690    ! INCREMENT STRING LENGTH
700      STRL=STRL+1
710      ORDER(STRL)=REACH
720      NEXT I
730  ! BEGIN OUTPUT OF STRING
740    PRINT #3,KK;",";STRL;",";WEIGHT(KK);",";CANDIDATE(KK)
750    IF STRL=0 THEN 880
760    PRINT #3,ORDER(1);
770    IF STRL=1 THEN 810
780      FOR L=2 TO STRL
790      PRINT #3,",";ORDER(L);
800      NEXT L
810    PRINT #3
820    PRINT #3,Z(ORDER(1))*WEIGHT(KK);
830    IF STRL=1 THEN 870
840      FOR L=2 TO STRL
850      PRINT #3,",";Z(ORDER(L))*WEIGHT(KK);
860      NEXT L
870    PRINT #3
880    NEXT KK
890  ! FINISHED
900  CLOSE 3
910  END
```

7.7 HILLS

```
1000  ! HILLS - HILLSMAN WEIGHTED DISTANCE EDITING ROUTINE
1010  DIM TONODE(200),WTDIST(200)
1020  MAXLEN=200
1030  PENALTY=1.0E20
1040  ! GET INPUT FILE
1050  INPUT "NAME OF INPUT FILE";FILE1$
1060  OPEN FILE1$ FOR INPUT AS FILE 1
1070  INPUT "NAME OF OUTPUT FILE";FILE3$
1080  OPEN FILE3$ FOR OUTPUT AS FILE 3
1090  ! GET OPTION
1100  PRINT "EDITING OPTIONS:"
1110  PRINT "    1 P-MEDIAN WITH MAX DISTANCE CONSTRAINT"
1120  PRINT "    2 MIN CENTRES WITH MAX DISTANCE CONSTRAINT"
1130  PRINT "    3 MAXIMAL COVERING PROBLEM"
1140  PRINT "    4 MAXIMAL COVERING WITH MAX DISTANCE CONSTRAINT"
1150  PRINT "    5 ATTENDANCE MAXIMIZING (LINEAR DECAY)"
1160  PRINT "    6 MINIMIZE TOTAL POWERED DISTANCE"
1170  INPUT "OPTION";OPTION
1180  ! CHECK DOMAIN OF OPTION
1190  IF OPTION>0 AND OPTION<7 THEN 1220
1200  PRINT "RANGE OF OPTIONS IS 1 TO 6"
1210  GO TO 1100
1220  ! GET SUPPLEMENTARY CONSTANTS
1230  IF OPTION<5 THEN INPUT "VALUE OF DISTANCE CONSTRAINT S";S
1240  IF OPTION=4 THEN INPUT "VALUE OF DISTANCE CONSTRAINT T";T
1250  ! CHECK DOMAIN OF T
1260  IF OPTION<>4 OR T>=S THEN 1290
1270  PRINT "T MUST BE GREATER THAN OR EQUAL TO S"
1280  GO TO 1240
1290  IF OPTION=5 THEN INPUT "VALUE OF DECAY CONSTANT";B
1300  IF OPTION=6 THEN INPUT "VALUE OF DISTANCE POWER";B
1310  ! BEGIN READING INPUT FILE
1320  ! END OF FILE CHECK
1330  ON ERROR GO TO 2030
1340  INPUT #1,KK,STRL,WEIGHT,CANDIDATE
1350  ! CHECK STRING LENGTH
1360  IF STRL<=MAXLEN THEN 1390
1370  PRINT "TOO MANY NODES IN STRING - LIMIT IS";MAXLEN
1380  STOP
1390  IF STRL=0 THEN 1830
1400  MAT INPUT #1,TONODE(STRL)
1410  MAT INPUT #1,WTDIST(STRL)
1420  ! OFFSET FOR OPTION 5
1430  A=0.0
1440  ! LOOP FOR EACH ENTRY IN STRING
1450    FOR L=1 TO STRL
1460  ! COMPUTE DISTANCE
1470    DIST=WTDIST(L)/WEIGHT
1480    ON OPTION GO TO 1500, 1530, 1580, 1620, 1670, 1740
1490  ! COME HERE FOR OPTION 1
1500    IF DIST>S THEN WTDIST(L)=PENALTY
1510    GO TO 1750
1520  ! COME HERE FOR OPTION 2
1530    IF TONODE(L)=KK THEN WTDIST(L)=1
1540    IF TONODE(L)<>KK AND DIST<=S THEN WTDIST(L)=0
1550    IF TONODE(L)<>KK AND DIST>S THEN WTDIST(L)=PENALTY
1560    GO TO 1750
1570  ! COME HERE FOR OPTION 3
1580    IF DIST<=S THEN WTDIST(L)=0
1590    IF DIST>S THEN WTDIST(L)=WEIGHT
1600    GO TO 1750
1610  ! COME HERE FOR OPTION 4
1620    IF DIST<=S THEN WTDIST(L)=0
1630    IF DIST>S AND DIST<=T THEN WTDIST(L)=WEIGHT
1640    IF DIST>T THEN WTDIST(L)=PENALTY
1650    GO TO 1750
1660  ! COME HERE FOR OPTION 5
1670    WTDIST(L)=WEIGHT*(1.0-B*DIST)
1680  ! RESET NEGATIVE DEMAND TO ZERO
1690    IF WTDIST(L)<0 THEN WTDIST(L)=0
1700  ! FIND LARGEST VALUE IN THIS ROW
1710    IF WTDIST(L)>A THEN A=WTDIST(L)
1720    GO TO 1750
1730  ! COME HERE FOR OPTION 6
1740    WTDIST(L)=WEIGHT*DIST**B
1750    NEXT L
1760  ! OFFSET FOR OPTION 5
1770  IF OPTION<>5 THEN 1830
1780    FOR L=1 TO STRL
1790    WTDIST(L)=A-WTDIST(L)
1800    NEXT L
1810  ! BEGIN OUTPUT
1820  ! BEGIN OUTPUT
1830  PRINT #3,KK;",";STRL;",";WEIGHT;",";CANDIDATE
1840  IF STRL=0 THEN 1340
1850  PRINT #3,TONODE(1);
1860  IF STRL=1 THEN 1900
1870    FOR L=2 TO STRL
1880    PRINT #3,",";TONODE(L);
1890    NEXT L
1900  PRINT #3
1910  PRINT #3,WTDIST(1);
1920  IF STRL=1 THEN 1960
1930    FOR L=2 TO STRL
1940    PRINT #3,",";WTDIST(L);
1950    NEXT L
1960  PRINT #3
1970  ! FINISHED THIS NODE
1980  GO TO 1340
```

```
1990  ! FINISHED
2000  CLOSE 3
2010  STOP
2020  ! COME HERE ON ERROR
2030  IF ERR<>11 THEN 2060
2040  RESUME 2000
2050  ! SOME OTHER ERROR
2060  PRINT "I/O ERROR NUMBER";ERR
2070  STOP
2080  END

Task?
```

7.8 ALLOC

```
1000  ! ALLOC - VERTEX SUBSTITUTION ALGORITHM
1010  DIM IN(500),SUM(1000),CID(50),INDEX(500)
1020  DIM TONODE(500),WTDIST(500)
1030  MAXN=500
1040  MAXP=50
1050  MAXLEN=500
1060  MAXSIZ=1000
1070  PENALTY=1.0E25
1080  INPUT "NUMBER OF NODES";N
1090  IF N<=MAXN THEN 1120
1100  PRINT "TOO MANY NODES - LIMIT IS";MAXN
1110  GO TO 1080
1120  INPUT "NUMBER OF SITES TO BE LOCATED";P
1130  IF P<=MAXP THEN 1160
1140  PRINT "TOO MANY SITES - LIMIT IS";MAXP
1150  GO TO 1120
1160  ! GET INITIAL SOLUTION
1170    FOR J=1 TO P
1180  ! IN(ID)=0 IF OUT,1 IF IN AND MOBILE, 2 IF FIXED -
1190    PRINT "SOLUTION NODE"; J;
1200    INPUT ID
1210    INPUT "ENTER '1' IF MOBILE, '2' IF FIXED"; IN(ID)
1220    CID(J)=ID
1230    NEXT J
1240  ! GET INPUT FILE
1250  INPUT "NAME OF INPUT FILE";FILE1$
1260  OPEN FILE1$ FOR INPUT AS FILE 1
1270  ! ZERO OBJECTIVE FUNCTION
1280  OLDOBJ=0
1290  ! ZERO CANDIDATE COUNTER
1300  M=0
1310  ! END OF FILE CHECK
1320  ON ERROR GO TO 2930
1330  INPUT #1,KK,STRL,WEIGHT,CANDIDATE
1340  IF STRL=0 THEN 1330
1350  ! CHECK STRING LENGTH
1360  IF STRL<=MAXLEN THEN 1390
1370  PRINT "TOO MANY NODES IN STRING - LIMIT IS";MAXLEN
1380  STOP
1390  MAT INPUT #1, TONODE(STRL)
1400  MAT INPUT #1, WTDIST(STRL)
1410  ! CHECK CANDIDACY OF INITIAL SOLUTION
1420  IF IN(KK)=0 OR CANDIDATE=1 THEN 1460
1430  PRINT "NODE"; KK; "IS NOT A CANDIDATE"
1440  STOP
1450  ! COUNT CANDIDATE NODES AND BUILD AN INDEX
1460  IF CANDIDATE=0 THEN 1500
1470  M=M+1
1480  INDEX(KK)=M
1490  ! LOOP TO EVALUATE OBJECTIVE FUNCTION
1500    FOR L=1 TO STRL
1510    IF IN(TONODE(L))=0 THEN 1540
1520    OLDOBJ=OLDOBJ+WTDIST(L)
1530    GO TO 1330
1540    NEXT L
1550  OLDOBJ=OLDOBJ+PENALTY
1560  GO TO 1330
1570  PRINT M;"CANDIDATES FOUND"
1580  PRINT "OBJECTIVE FUNCTION IS";OLDOBJ
1590  ! IN FIRST PASS NO ELEMENTS OF SUM HAVE BEEN PROCESSED
1600  LAST=0
1610  BEST=1.0E30
1620  ! IN THIS PASS NEXT 'NUMBER' OF ELEMENTS OF SUM PROCESSED
1630  ! BEGINNING AT LAST+1
1640  NUMBER=P*M-LAST
1650  IF NUMBER>MAXSIZ THEN NUMBER=MAXSIZ
1660  ! INITIALIZE ELEMENTS OF SUM
1670    FOR K=1 TO NUMBER
1680    SUM(K)=1
1690    NEXT K
1700  ! BEGIN READING DATA
1710  OPEN FILE1$ FOR INPUT AS FILE 1
1720  ! END OF FILE CHECK
1730  ON ERROR GOTO 3000
1740  ! LOOP FOR EACH DEMAND NODE
1750  ! SET CENTRES COUNTER TO ZERO
1760  CENTERS=0
1770    FOR K=1 TO NUMBER
1780  ! ERROR CONDITION: SHOULD BE NO NEGATIVES AT START OF LOOP
1790    IF SUM(K)<0 THEN 2500
1800  ! MAKE ALL SUM NEGATIVE AT START OF LOOP
1810    SUM(K)=-SUM(K)
1820    NEXT K
1830  ! READ A STRING RECORD
1840  INPUT #1, KK,STRL,WEIGHT,CANDIDATE
1850  IF STRL=0 THEN 1840
1860  MAT INPUT #1, TONODE(STRL)
1870  MAT INPUT #1,WTDIST(STRL)
1880  ! LOOP FOR ALL CANDIDATE NODES IN ASCENDING ORDER OF DISTANCE
1890    FOR L=1 TO STRL
1900  ! COMPUTE INCREMENT TO OBJECTIVE FUNCTION
1910    ADD=WTDIST(L)
1920    IF IN(TONODE(L))=0 THEN 2320
1930  ! COUNT SOLUTION NODES FOUND
1940    CENTERS=CENTERS+1
1950    IF CENTERS<2 THEN 2190
1960  ! COME HERE IF FIRST SOLUTION NODE FOUND
1970  ! FILL IN ALL BUT ONE COLUMN OF SUM
1980      FOR J=1 TO P
1990      IF CID(J)<>TONODE(L) THEN 2030
```

```
2000    ! REMEMBER FIRST CENTRE FOUND
2010      FIRST=J
2020      GO TO 2150
2030      K=J*M
2040      IF K<=LAST THEN 2150
2050    ! COMPUTE OFFSET
2060      K=K-M-LAST
2070    ! LOOP THROUGH ALL ELEMENTS IN COLUMN
2080        FOR I=1 TO M
2090        K=K+1
2100        IF K>NUMBER THEN 2150
2110        IF K<1 THEN 2140
2120      ! INCREMENT SUM
2130        IF SUM(K)<0 THEN SUM(K)=ABS(SUM(K))+ADD
2140        NEXT I
2150      NEXT J
2160    GOTO 2420
2170  ! COME HERE IF SECOND SOLUTION NODE FOUND
2180  ! FILL IN REMAINING COLUMN OF SUM
2190    KLOW=(FIRST-1)*M+1-LAST
2200    KHIGH=FIRST*M-LAST
2210    IF KLOW<1 THEN KLOW=1
2220    IF KHIGH>NUMBER THEN KHIGH=NUMBER
2230  ! NO ENTRIES IN THIS SET
2240    IF KLOW>KHIGH THEN 1760
2250  ! LOOP THROUGH ELEMENTS IN COLUMN
2260      FOR K=KLOW TO KHIGH
2270    ! INCREMENT SUM
2280      IF SUM(K)<0 THEN SUM(K)=ABS(SUM(K))+ADD
2290      NEXT K
2300  ! NEXT DEMAND NODE
2310    GOTO 1760
2320  ! COME HERE IF NODE IS NOT IN SOLUTION
2330  ! FILL IN ONE ROW OF SUM
2340      FOR J=1 TO P
2350      K=(J-1)*M+INDEX(TONODE(L))
2360      IF K<=LAST THEN 2410
2370      K=K-LAST
2380      IF K>NUMBER THEN 2420
2390    ! INCREMENT SUM
2400      IF SUM(K)<0 THEN SUM(K)=ABS(SUM(K))+ADD
2410      NEXT J
2420    NEXT L
2430  ! COME HERE IF END OF DATA WITHOUT FINDING 2 SOLN NODES
2440  ! ADD PENALTY TO ANY REMAINING NEGATIVE ELEMENTS
2450    FOR K=1 TO NUMBER
2460    IF SUM(K)<0 THEN SUM(K)=ABS(SUM(K))+PENALTY
2470    NEXT K
2480  ! NEXT DEMAND NODE
2490  GOTO 1760
2500  PRINT 'ERROR CONDITION'
2510  PRINT 'LAST =';LAST;' K =';K;' LAST NODE READ';KK
2520  STOP
2530  ! COME HERE AT END OF DATA
2540  ! LOOP TO FIND BEST SWAP
2550    FOR K=1 TO NUMBER
2560  ! REMOVE INITIAL VALUE FROM SUM
2570    SUM(K)=ABS(SUM(K))-1
2580    J=INT((K+LAST-1)/M)+1
2590  ! IGNORE IF FIXED
2600    IF IN(CID(J))=2 THEN 2650
2610    IF SUM(K)>=BEST THEN 2650
2620  ! NEW BEST SWAP
2630    BEST=SUM(K)
2640    KSAVE=K+LAST
2650    NEXT K
2660  LAST=LAST+NUMBER
2670  ! RETURN FOR NEW PASS
2680  IF LAST<P*M THEN 1640
2690  ! MAJOR LOOP FINISHED
2700  IF BEST>=OLDOBJ THEN 2850
2710  J=INT((KSAVE-1)/M)+1
2720  I=KSAVE-(J-1)*M
2730  PRINT 'BEST SWAP IS';I;'TH CANDIDATE FOR';CID(J)
2740    FOR KK=1 TO N
2750    IF INDEX(KK)=I THEN 2770
2760    NEXT KK
2770  PRINT I;'TH CANDIDATE IS NODE';KK
2780  ! MAKE SWAP
2790  IN(KK)=1
2800  IN(CID(J))=0
2810  CID(J)=KK
2820  OLDOBJ=BEST
2830  ! RETURN TO REPEAT MAJOR LOOP
2840  GOTO 1580
2850  PRINT 'NO BETTER SOLUTION EXISTS'
2860  PRINT 'SOLUTION IS:';
2870    FOR J=1 TO P
2880    PRINT CID(J);
2890    NEXT J
2900  PRINT
2910  STOP
2920  ! COME HERE ON ERROR
2930  IF ERR<>11 THEN 2970
2940  CLOSE 1
2950  RESUME 1570
2960  ! SOME OTHER I/O ERROR
2970  PRINT 'I/O ERROR NUMBER';ERR
2980  STOP
2990  ! COME HERE ON ERROR
3000  IF ERR<>11 THEN 3040
3010  CLOSE 1
```

```
3020  RESUME 2530
3030  ! SOME OTHER ERROR
3040  PRINT "I/O ERROR NUMBER";ERR
3050  STOP
3060  END

Task?


      RUN NESEQ
NAME OF INPUT FILE? EVAL
LOOP INDENTATION? 2
COMMENT INDENTATION? -2

Task? TY EVAL.RSQ
```

7.9 EVAL

```
1000  ! EVAL - EVALUATION OF SOLUTIONS
1010  DIM IN(500),TONODE(500),WTDIST(500),TOTDEM(50),TOTDIST(50)
1020  DIM MAXDIST(50),FAR(50),TOTDIST2(50),MAXDIST2(50),FAR2(50),CID(50)
1030  MAXP=50
1040  MAXN=500
1050  MAXLEN=500
1060  LEGEND$(1)="YES"
1070  LEGEND$(0)="NO "
1080  ! GET INPUT FILE
1090  INPUT "NAME OF INPUT FILE";FILE1$
1100  ! ZERO SOLUTION
1110    FOR KK=1 TO N
1120    IN(KK)=0
1130    NEXT KK
1140  INPUT "NUMBER OF SITES, OR 0 TO EXIT";P
1150  IF P=0 THEN STOP
1160  IF P<=MAXP THEN 1200
1170  PRINT "TOO MANY SITES - LIMIT IS";MAXP
1180  GO TO 1140
1190  ! GET SOLUTION
1200    FOR J=1 TO P
1210    PRINT "SOLUTION NODE";J;
1220    INPUT ID
1230    IF ID<=MAXN THEN 1260
1240    PRINT "TOO MANY NODES - LIMIT IS";MAXN
1250    GO TO 1210
1260    IN(ID)=J
1270    CID(J)=ID
1280    NEXT J
1290  ! ZERO SUMS
1300  ! WEIGHT NOT SERVED
1310  NOSERVE=0
1320  ! NO SECOND CENTRE
1330  NOSERVE2=0
1340    FOR J=1 TO P
1350  ! TOTAL DEMAND BY SITE
1360    TOTDEM(J)=0
1370  ! TOTAL DISTANCE BY SITE
1380    TOTDIST(J)=0
1390  ! MAXIMUM DISTANCE BY SITE
1400    MAXDIST(J)=0
1410  ! FURTHEST NODE
1420    FAR(J)=0
1430  ! TOTAL TO SECOND NEAREST
1440    TOTDIST2(J)=0
1450  ! MAXIMUM TO SECOND NEAREST
1460    MAXDIST2(J)=0
1470  ! FURTHEST SECOND CENTRE
1480    FAR2(J)=0
1490    NEXT J
1500  ! OUTPUT ALLOCATION?
1510  PRINT "OUTPUT OPTIONS:"
1520  PRINT "        1 SUMMARY ONLY"
1530  PRINT "        2 ALLOCATION AND SUMMARY"
1540  INPUT "OPTION";OPTION
1550  IF OPTION=1 THEN 1610
1560  PRINT
1570  PRINT "NODE      WEIGHT CANDIDATE";
1580  PRINT "  NEAREST  DISTANCE";
1590  PRINT " SECOND DISTANCE"
1600  ! GET INPUT FILE
1610  OPEN FILE1$ FOR INPUT AS FILE 1
1620  ! END OF FILE CHECK
1630  ON ERROR GO TO 2860
1640  INPUT #1,KK,STRL,WEIGHT,CANDIDATE
1650  IF STRL=0 THEN 1640
1660  ! CHECK STRING LENGTH
1670  IF STRL<=MAXLEN THEN 1700
1680  PRINT "TOO MANY NODES IN STRING - LIMIT IS";MAXLEN
1690  STOP
1700  MAT INPUT #1,TONODE(STRL)
1710  MAT INPUT #1,WTDIST(STRL)
1720  ! CHECK CANDIDACY OF SOLUTION
1730  IF IN(KK)=0 OR CANDIDATE=1 THEN 1770
1740  PRINT "NODE";KK;"IS NOT A CANDIDATE"
1750  STOP
1760  ! SKIP IF NO WEIGHT
1770  IF WEIGHT>0 THEN 1810
1780  PRINT KK,"NO WEIGHT"
1790  GO TO 1640
1800  ! SCAN STRING FOR FIRST AND SECOND CENTRES
1810  CENTERS=0
1820    FOR L=1 TO STRL
1830    IF IN(TONODE(L))=0 THEN 2090
1840    CENTERS=CENTERS+1
1850    IF CENTERS=2 THEN 2000
1860  ! COME HERE FOR FIRST CENTRE
1870    FIRST=TONODE(L)
1880    J=IN(FIRST)
1890  ! TOTAL DEMAND
1900    TOTDEM(J)=TOTDEM(J)+WEIGHT
1910  ! TOTAL DISTANCE
1920    TOTDIST(J)=TOTDIST(J)+WTDIST(L)
1930  ! MAXIMUM DISTANCE
1940    DIST1=WTDIST(L)/WEIGHT
1950    IF DIST1<=MAXDIST(J) THEN 2090
1960    MAXDIST(J)=DIST1
1970    FAR(J)=KK
1980    GO TO 2090
1990  ! COME HERE FOR SECOND NEAREST CENTRE
```

```
2000    SECOND=TONODE(L)
2010  ! TOTAL DISTANCE
2020    TOTDIST2(J)=TOTDIST2(J)+WTDIST(L)
2030  ! MAXIMUM DISTANCE
2040    DIST2=WTDIST(L)/WEIGHT
2050    IF DIST2<=MAXDIST2(J) THEN 2270
2060    MAXDIST2(J)=DIST2
2070    FAR2(J)=KK
2080    GO TO 2270
2090    NEXT L
2100  ! CENTRES NOT FOUND
2110  IF CENTERS=1 THEN 2200
2120  ! NO NEAREST CENTRE
2130  NOSERVE=NOSERVE+WEIGHT
2140  IF OPTION=1 THEN 1640
2150    FORM$="###      ######    \   \"
2160    PRINT USING FORM$,KK,WEIGHT,LEGEND$(CANDIDATE)!
2170    PRINT "         NOT SERVED"
2180    GO TO 1640
2190  ! NO SECOND CENTRE
2200    NOSERVE2=NOSERVE2+WEIGHT
2210    IF OPTION=1 THEN 1640
2220      FORM$="###      ######    \   \     ###    ######"
2230      PRINT USING FORM$,KK,WEIGHT,LEGEND$(CANDIDATE),FIRST,DIST1!
2240      PRINT "    NO SECOND CENTRE"
2250      GO TO 1640
2260    ! COME HERE AFTER SECOND CENTRE
2270      IF OPTION=1 THEN 1640
2280        FORM$="###      ######    \   \     ###    ######    ###    ######"
2290        PRINT USING FORM$,KK,WEIGHT,LEGEND$(CANDIDATE),FIRST,DIST1,SECOND,DIST2
2300        GO TO 1640
2310      ! OUTPUT SUMMARY
2320        PRINT
2330        PRINT "UNSERVED DEMAND";NOSERVE
2340        PRINT "DEMAND WITH NO ALTERNATE SITE";NOSERVE2
2350      ! OUTPUT BY SITE
2360          FOR J=1 TO P
2370          PRINT
2380          PRINT "SITE";J;"IS AT NODE";CID(J)
2390          PRINT "DEMAND ALLOCATED";TOTDEM(J)
2400          PRINT "DISTANCE TRAVELLED";TOTDIST(J)
2410          PRINT "MEAN DISTANCE";TOTDIST(J)/TOTDEM(J)
2420          PRINT "MAXIMUM DISTANCE";MAXDIST(J)
2430          PRINT "FURTHEST NODE ALLOCATED";FAR(J)
2440          PRINT "DISTANCE TRAVELLED TO NEAREST ALTERNATE";TOTDIST2(J)
2450          PRINT "MEAN DISTANCE TO NEAREST ALTERNATE";TOTDIST2(J)/TOTDEM(J)
2460          PRINT "MAXIMUM DISTANCE TO NEAREST ALTERNATE";MAXDIST2(J)
2470          PRINT "FURTHEST NODE FROM AN ALTERNATE";FAR2(J)
2480          NEXT J
2490        PRINT
2500      ! ZERO GENERAL SUMS
2510        STOTDEM=0
2520        STOTDIST=0
2530        SMAXDIST=0
2540        SFAR=0
2550        DISPTOT=1.0E20
2560        DISPMAX=1.0E20
2570          FOR J=1 TO P
2580          STOTDEM=STOTDEM+TOTDEM(J)
2590          STOTDIST=STOTDIST+TOTDIST(J)
2600          IF MAXDIST(J)<=SMAXDIST THEN 2640
2610          SMAXDIST=MAXDIST(J)
2620          SFAR=FAR(J)
2630        ! FIND MOST DISPOSABLE CENTRES
2640          IF TOTDIST2(J)-TOTDIST(J)>=DISPTOT THEN 2670
2650          DISPTOT=TOTDIST2(J)-TOTDIST(J)
2660          MOST1=J
2670          NEXT J
2680          FOR J=1 TO P
2690          IF MAXDIST2(J)<=SMAXDIST THEN 2730
2700          IF MAXDIST2(J)-SMAXDIST>=DISPMAX THEN 2730
2710          DISPMAX=MAXDIST2(J)-SMAXDIST
2720          MOST2=J
2730          NEXT J
2740        PRINT "TOTAL DEMAND SERVED";STOTDEM
2750        PRINT "TOTAL DISTANCE TRAVELLED";STOTDIST
2760        PRINT "MEAN DISTANCE TRAVELLED";STOTDIST/STOTDEM
2770        PRINT "MAXIMUM DISTANCE TRAVELLED";SMAXDIST
2780        PRINT "FURTHEST NODE FROM ALLOCATED CENTRE";SFAR
2790        PRINT
2800        PRINT "MOST DISPOSABLE CENTRES"
2810        PRINT "    TOTAL DISTANCE CRITERION - CENTRE";MOST1;"AT NODE";CID(MOST1)
2820        PRINT "    MAXIMUM DISTANCE         - CENTRE";MOST2;"AT NODE";CID(MOST2)
2830        PRINT
2840        GOTO 1110
2850      ! COME HERE ON ERROR
2860        IF ERR<>11 THEN 2900
2870        CLOSE 1
2880        RESUME 2310
2890      ! SOME OTHER I/O ERROR
2900        PRINT "I/O ERROR NUMBER";ERR
2910        STOP
2920        END

Task? RUN NESEQ
NAME OF INPUT FILE? HILLS
LOOP INDENTATION? 2
COMMENT INDENTATION? -2
ON-GOTO STATEMENT FOUND ... (NO PROBLEM!)

Task? TY HILLS.RSQ
```

```
2000    SECOND=TONODE(L)
2010  ! TOTAL DISTANCE
2020    TOTDIST2(J)=TOTDIST2(J)+WTDIST(L)
2030  ! MAXIMUM DISTANCE
2040    DIST2=WTDIST(L)/WEIGHT
2050    IF DIST2<=MAXDIST2(J) THEN 2270
2060    MAXDIST2(J)=DIST2
2070    FAR2(J)=KK
2080    GO TO 2270
2090    NEXT L
2100  ! CENTRES NOT FOUND
2110  IF CENTERS=1 THEN 2200
2120  ! NO NEAREST CENTRE
2130  NOSERVE=NOSERVE+WEIGHT
2140  IF OPTION=1 THEN 1640
2150    FORM$="###     ######    \   \"
2160    PRINT USING FORM$,KK,WEIGHT,LEGEND$(CANDIDATE);
2170    PRINT "         NOT SERVED"
2180    GO TO 1640
2190  ! NO SECOND CENTRE
2200    NOSERVE2=NOSERVE2+WEIGHT
2210    IF OPTION=1 THEN 1640
2220      FORM$="###.    ######    \   \     ###    ######"
2230      PRINT USING FORM$,KK,WEIGHT,LEGEND$(CANDIDATE),FIRST,DIST1;
2240      PRINT "    TO SECOND CENTRE"
2250      GO TO 1640
2260    ! COME HERE AFTER SECOND CENTRE
2270      IF OPTION=1 THEN 1640
2280        FORM$="###     ######    \   \     ###    ######    ###    ######"
2290        PRINT USING FORM$,KK,WEIGHT,LEGEND$(CANDIDATE),FIRST,DIST1,SECOND,DIST2
2300        GO TO 1640
2310      ! OUTPUT SUMMARY
2320        PRINT
2330        PRINT "UNSERVED DEMAND";NOSERVE
2340        PRINT "DEMAND WITH NO ALTERNATE SITE";NOSERVE2
2350      ! OUTPUT BY SITE
2360          FOR J=1 TO P
2370          PRINT
2380          PRINT "SITE";J;"IS AT NODE";CID(J)
2390          PRINT "DEMAND ALLOCATED";TOTDEM(J)
2400          PRINT "DISTANCE TRAVELLED";TOTDIST(J)
2410          PRINT "MEAN DISTANCE";TOTDIST(J)/TOTDEM(J)
2420          PRINT "MAXIMUM DISTANCE";MAXDIST(J)
2430          PRINT "FURTHEST NODE ALLOCATED";FAR(J)
2440          PRINT "DISTANCE TRAVELLED TO NEAREST ALTERNATE";TOTDIST2(J)
2450          PRINT "MEAN DISTANCE TO NEAREST ALTERNATE";TOTDIST2(J)/TOTDEM(J)
2460          PRINT "MAXIMUM DISTANCE TO NEAREST ALTERNATE";MAXDIST2(J)
2470          PRINT "FURTHEST NODE FROM AN ALTERNATE";FAR2(J)
2480          NEXT J
2490        PRINT
2500      ! ZERO GENERAL SUMS
2510        STOTDEM=0
2520        STOTDIST=0
2530        SMAXDIST=0
2540        SFAR=0
2550        DISPTOT=1.0E20
2560        DISPMAX=1.0E20
2570          FOR J=1 TO P
2580          STOTDEM=STOTDEM+TOTDEM(J)
2590          STOTDIST=STOTDIST+TOTDIST(J)
2600          IF MAXDIST(J)<=SMAXDIST THEN 2640
2610          SMAXDIST=MAXDIST(J)
2620          SFAR=FAR(J)
2630        ! FIND MOST DISPOSABLE CENTRES
2640          IF TOTDIST2(J)-TOTDIST(J)>=DISPTOT THEN 2670
2650          DISPTOT=TOTDIST2(J)-TOTDIST(J)
2660          MOST1=J
2670          NEXT J
2680          FOR J=1 TO P
2690          IF MAXDIST2(J)<=SMAXDIST THEN 2730
2700          IF MAXDIST2(J)-SMAXDIST>=DISPMAX THEN 2730
2710          DISPMAX=MAXDIST2(J)-SMAXDIST
2720          MOST2=J
2730          NEXT J
2740        PRINT "TOTAL DEMAND SERVED";STOTDEM
2750        PRINT "TOTAL DISTANCE TRAVELLED";STOTDIST
2760        PRINT "MEAN DISTANCE TRAVELLED";STOTDIST/STOTDEM
2770        PRINT "MAXIMUM DISTANCE TRAVELLED";SMAXDIST
2780        PRINT "FURTHEST NODE FROM ALLOCATED CENTRE";SFAR
2790        PRINT
2800        PRINT "MOST DISPOSABLE CENTRES"
2810        PRINT "    TOTAL DISTANCE CRITERION - CENTRE";MOST1;"AT NODE";CID(MOST1)
2820        PRINT "    MAXIMUM DISTANCE         - CENTRE";MOST2;"AT NODE";CID(MOST2)
2830        PRINT
2840        GOTO 1110
2850      ! COME HERE ON ERROR
2860        IF ERR<>11 THEN 2900
2870        CLOSE 1
2880        RESUME 2310
2890      ! SOME OTHER I/O ERROR
2900        PRINT "I/O ERROR NUMBER";ERR
2910        STOP
2920        END
```