Designing and Implementing Strategies for Solving Large Location-Allocation Problems with Heuristic Methods

June 1991

Paul J. Densham

NCGIA
Department of Geography
State University of New York at Buffalo
Buffalo, NY 14261
USA

and

Gerard Rushton

Department of Geography
San Diego State University
San Diego, CA 92182

National Center for Geographic Information and Analysis

Report 91-10

DESIGNING AND IMPLEMENTING STRATEGIES FOR SOLVING LARGE LOCATION-ALLOCATION PROBLEMS WITH HEURISTIC METHODS_

Paul J. Densham[1] and Gerard Rushton[2]

ABSTRACT

Solution techniques for location-allocation problems usually are not a part of micro-computer based geo-processing systems. The large volumes of data to process and store and the complexity of algorithms present a barrier to implementation of these solution techniques in a micro-computer environment. Yet decision-makers need analysis systems that return solutions to location selection problems in real time. We show that processing requirements for the most accurate heuristic, location-allocation algorithm can be drastically reduced by pre-processing inter-point distance data as both candidate and demand strings and exploiting the spatial structure of location-allocation problems by updating an allocation table. Consequently, solution times increase approximately linearly with problem size. These developments allow the solution of large problems (3,000 nodes) in a microcomputer-based, interactive decision-making environment. These methods are implemented in a micro-computer system and tests on three network problems validate our claims.

INTRODUCTION

To solve many practical spatial planning problems, spatial decision support systems (SDSS) are being developed to integrate the data manipulation and display capabilities of geographic information systems (GIS) with spatial analysis techniques and with human problem-solving skills (Densham and Rushton, 1988). Among this triad of challenges, incorporating techniques of spatial analysis has proved to be difficult. In the microcomputer-based environment in which many decision-makers work (Newton et al., 1988), many techniques of spatial analysis have not been implemented because typical applications require large data storage, manipulation and processing capabilities. Location-allocation problems are typical of this genre. Instead, many software systems developed to date provide merely evaluation of the implications of locational decisions selected by intuitive approaches - see, for example, Schneider (1976) and Lupien et al. (1987).

By capitalizing on features of the general structure of location-allocation problems, we have developed a set of principles for implementing the best of currently known heuristic algorithms; these principles enable us to solve large problems in times that are realistic in a microcomputer-based, interactive decision-making environment. Existing software for solving location-allocation problems on microcomputers, such as the PLACE suite (Goodchild and Noronha, 1983) and UDMS (Robinson, 1983), can accommodate small to medium problems of the order of 200 to 500 nodes. These analysis systems were designed for "pilot or demonstrator applications," (Beaumont, 1988, p. 1141). They are not suitable for solving most real-size applications where questions of efficiency of resource allocation and the effectiveness of alternate, proposed solutions need to be answered. Designed to work within the limits of microcomputer technology of five to ten years ago, and using the best data reduction and processing techniques for location-allocation problems known at that time, their design is not suitable for the typical characteristics of the current generation of microcomputers. We found, in a number of practical applications, that a new design was necessary to reach our goal of an analysis system for location-allocation problems in which solution times would increase approximately linearly with problem size and in which total computation time would be within limits that a decision-maker, who wished to work in an interactive manner with the analysis system, would find acceptable.

Such an analysis system must be able to solve problems on networks consisting of up to 3,000 nodes, all of which may be candidate facility sites, 12,000 links and up to 1,500 facilities. These parameters are somewhat arbitrary, although our experience in solving problems of practical interest to decision-makers has been that, at a basic scale of geographic data disaggregation, many problems are of this size. Furthermore, attempts to solve problems of this size by aggregating data into fewer demand units encounter serious problems involving error in estimating the values of objective functions and in identifying optimal locations, (Goodchild, 1977; Casillas, 1987; Current and Schilling, 1985, 1990). To solve problems of this size, which are of practical interest to decision-makers, without data aggregation, requires a fresh approach to system design.

QUESTIONS TO BE ANSWERED

The need for a location-allocation. modelling capability in a spatial decisionsupport system arises whenever multiple alternative locations for providing services need to be evaluated. In addition to providing information about the locations which are optimal for a given objective, an important phase in the decision-processes of many users is that of problem exploration and definition.

[1] National Center for Geographic Information and Analysis, State University of New York at Buffalo, Buffalo, NY 14261, USA. (716) 636-3822. Email: geopjd@ubvms.bitnet.
[2] Department of Geography, San Diego State University, San Diego, CA 92182. Email: geogr@calstate.bitnet

Ibis need arises more often than is commonly thought. The needs of users of an SDSS in this phase is for rapid evaluation of many locational alternatives and a need to compare evaluations of alternative configurations in the space of relevant decision-making criteria - the solution space - with evaluations in geographic space. Typical questions asked by decision-makers can be grouped according to the component of interest:

1. Efficiency:
   - o How much improvement in the criteria of interest is possible from re-locating the same resources that are currently being used?
   - o How many fewer resources would be needed to provide the same level of service if the resources were optimally located?
   - o What proportion of the greatest potential improvement in the criteria of interest was achieved by the k most recent locational decisions?
2. Multiple Criteria Evaluation:
   - o What are the alternative locations for the resources that provide acceptable levels of performance on several criteria of interest?
   - o What are the location sets that are optimal for the alternative weighted criteria of interest?
3. Impact on Defined Components:
   - o Given location sets that are optimal with respect to alternative criteria, what are the distributions of benefits?
   - o What locations, if selected, will most increase the benefits to defined groups?
4. Effect of Constraints on Decision-Making:
   - o What single exchange of a site not in the solution with one in the solution will most improve the objective function value?
   - o What is the effect on any of the questions described above of constraining locational decisions?

We have encountered examples of each of these questions in applications of location-allocation models to location decision-making problems. Our experience was that while none of these questions were new, existing geo-processing systems could answer many of them for small problems but not for large problems, such as we have defined above. We also found that these systems indiscriminately evaluate all possible pairwise interchanges of eligible sites with facility locations, sometimes many times, to answer these questions. Consequently, inefficiencies in processing information to compute the change in the objective function value following an exchange of locations increase solution times beyond those considered reasonable by decision-makers. A primary motivation for our work is to remedy these deficiencies, common to all published micro-computer based geo-processing systems.

HEURISTIC SOLUTION TECHNIQUES FOR SOLVING LOCATION-ALLOCATION MODELS

There are two major groups of solution techniques for location-allocation models: exact, or programming, techniques and heuristic methods. The former group employ linear, integer or mixed-integer programming methods, to yield an optimal solution; heuristic techniques include a variety of solution methods which may yield optimal or sub-optimal solutions.

Heuristic Solution Methods

Heuristic solution methods have a number of advantages when compared with programming techniques: as we will show, large problems can be solved relatively quickly; many objective functions can be used; and a range of alternative, marginally sub-optimal solutions can be identified. The major drawback to heuristic methods is that they are not exact, none can be guaranteed to find the optimum solution. Heuristics have been developed to solve both continuous (Cooper, 1964, 1967) and discrete space formulations (Kuehn and Hamburger, 1963; Maranzana, 1964; Feldman, Lehrer and Ray, 1966; Teitz and Bart, 1968).

The Teitz and Bart (1968) vertex substitution heuristic has a number of characteristics that make it attractive for use in a microcomputer setting. First, it frequently converges to the optimum solution, performing very well when compared with exact techniques and other heuristics (Rosing, Hillsman and Rosing-Vogelaar, 1979). In addition, irrespective of problem size, it is usually very close to finding a solution after two iterations with convergence usually taking place within four iterations - making it possible to predict an upper bound on solution times with some degree of accuracy. Third, although it was originally designed to solve the p-median problem, appropriate editing of the data enables the heuristic to solve many other formulations (Hillsman, 1984), including non-linear ones. Fourth, the heuristic is not tied to any particular data structure or implementation strategy.

The solution process evaluates marginal changes in the current solution - the best set of facility locations identified at any stage in the algorithm. A substitution involves relocating one facility temporarily and comparing the resulting objective function with that for the current solution. Where a substitution results in a lower objective function, the new locational pattern may be adopted as the current solution - this relocation of the facility is termed a swap.

The algorithm underlying the solution process is as follows. An initial solution, a list of facility sites termed the current solution, is compiled and the value of the objective function is calculated. The heuristic consists of two loops. The outer loop selects every candidate - a feasible facility location - on the network in sequence. As each candidate is selected, it is passed to the inner loop which calculates the values of the objective function that would result from substituting the candidate for each of the facility sites in the current solution. On completion of the inner loop, if a substitution yields an objective function value lower than that of the current solution, the candidate site is swapped into the current solution. If more than one substitution results in lower objective function values, the candidate site yielding the lowest value is swapped into the current solution. At the end of the outer loop (the end of one iteration), if no swaps have been made the heuristic terminates and the current solution is the final-solution; if a swap has been made the heuristic commences another iteration.

Exploiting Spatial Structure

Teitz and Bart's (1968) original discussion of their algorithm does not address implementation in any detail. A close examination of the heuristic's solution process, however, provides insight into the general spatial structure of location-allocation problems. This structure can be exploited to reduce processing work in comparison with alternative implementations that literally perform the algorithmic steps they outlined over all elements of the distance matrix.

Figure 1 shows two candidate locations, M_1 and M_2; M_1 currently is a facility site and M_2 is being evaluated as an alternative site for that facility. (To make the explanation as simple as possible, assume that the area depicted in Figure 1 is an isotropic plain; that travel between any two points on the plain can be achieved using a straight line; and, finally, that there is some range for the service being provided which defines a potential service area for each facility.) The service area of each facility is the subset of demand nodes on the plain that are serviced by that facility. The size of each service area depends on both the number and the spatial arrangement of other facilities on the plain. In Figure 1, the service area of the facility at M_1 is shown as a hatched polygon; the potential service area of M_2 is shown as a hatched circle - the radius of the circle is equal to the range of the service provided.

As a result of the substitution of M_2 for M_1 , all of the demand nodes in the service area of M_1, will be reallocated to other facilities; of the demand nodes inside the potential service area of M_2, however, only those that are closer to M_2 than any other facility site will be reallocated to M_2. Thus, in an uncapacitated problem, only those demand nodes that lie inside the actual service area of M_1 and the potential service area of M_2 are affected by the substitution - no other nodes on the plain will be reallocated. This statement is also true in a network representation of the problem. Consequently, to evaluate the net change in the objective function resulting from a substitution, only a subset of demand nodes needs to be examined. Teitz and Bart overlook this point, however, and, for each and every substitution, they determine the allocation of every demand node on the network to evaluate the objective function.

Large savings in processing are realized by examining only those nodes which are allocated to a different facility if a substitution is implemented as a swap. These savings can be gained in many situations, ranging from the case where the network is extensive but the service area of a candidate is tiny in comparison, to that where service areas potentially are large but there is an extremely high density of facilities; in both these extreme cases, and many in between, small service areas result. Many problems we have encountered exhibit this characteristic.

Strategies for implementing the Teitz and Bart heuristic that exploit this spatial structure can be used to decrease solution times considerably over previous codes. Moreover, as the size of problems increase, these strategies become more effective.

EFFICIENT STRATEGIES FOR FINDING SOLUTIONS

The implementation of any location-allocation heuristic requires three issues to be addressed: the number of calculations carried out; the volume of data to be processed; and access times to that data. These three issues are interdependent being related to the choice of data structure and the mechanisms used to evaluate the substitutions and to execute the swaps. Strategies can be formulated for the Teitz and Bart heuristic which seek to minimize computation, the volume of data to be processed and access times to that data. The viability of any particular strategy, however, depends on the features and constraints of a microcomputer's architecture and operating system. Thus, in later sections, the discussion focuses on a strategy for a personal computer with 640 kilobytes of RAM operating under MS/PC-DOS.

Data structures

The Teitz and Bart heuristic requires data on:

1) the nodes where the facilities in the initial solution are located;
2) the facilities, if any, that are fixed at their current locations;
3) the candidate sites for facilities; and
4) the demand nodes, a measure of their demand, and distances to all candidate nodes that might serve them.

A simple way of meeting these requirements is to produce lists for items one to three and a weighted distance matrix for item four. As problem size increases, however, distance matrices are extremely inefficient for storage and retrieval. With distance matrices, storage requirements increase as the square of problem size (number of demand nodes). A more efficient approach is to use distance strings. A distance string is a base node with a list of other nodes and the distances between each of these nodes and the base node - ordered by increasing distance value (Hillsman, 1980, pp. 81-90). With such an organization for storing and retrieving distance data, storage requirements are much reduced: the upper bound on the volume of data to be stored is now given by (M * N) -the number of demand nodes multiplied by the number of candidates - rather than by the square of the number of demand nodes - as in a distance matrix. Two kinds of distance strings can be compiled, candidate and demand strings. A demand string is built for every node in the network that has a demand associated with it. Each string lists all the potential facility sites (candidates) on the network that can serve a given demand node and indicates whether the demand node is itself a candidate. To increase the efficiency of searching a string for the closest facility, the candidates are listed in ascending order of distance from the demand node. The length of the string is determined by the total number of candidates that might conceivably serve the demand node. Where the string contains only a subset of the candidates, this often is defined as the number of candidates within a specified distance or "path radius" of the demand node. In contrast, a candidate string contains a list of all the demand nodes that can be served by a given candidate node, again listed in increasing order of distance. In a similar manner to demand strings, the length of a candidate string is determined by the number of demand nodes on the network that might conceivably be served by the candidate node.

A set of candidate strings contains the same information as a set of demand strings except that the information is organized differently; the difference is that their respective organizations are designed for contrasting forms of search and retrieval. Candidate strings optimize retrieval of all the demand nodes that can be served by a candidate site. Demand strings optimize retrieval of all the candidate nodes that can serve a particular demand node. To find the demand nodes that can be served by a candidate site using demand strings would require an exhaustive search of all the demand strings. Similarly, to find the candidate nodes that can serve a particular demand node using candidate strings would require an exhaustive search of all the candidate strings. No previously published implementation strategy has used both forms of strings: Hillsman (1980) used candidate strings whereas Goodchild and Noronha (1983) used demand strings.

It is our experience that large savings in processing costs are possible when both candidate and demand strings are used. We compensate for the concomitant increase in data storage by using a new technique which very substantially reduces the need to access data during the course of analysis. This technique - described below as the "allocation table approach" - also considerably reduces the internal storage needed to evaluate the objective function following a substitution. As shown below, demand strings are accessed only after a swap is made - that is, after it is known that a substitution of a candidate site for a site in the solution will result in a reduction in the objective function value.

The allocation table approach

Teitz and Bart's (1968) paper discusses how to calculate the change in the value of the objective function that results from a substitution. Although incorrect in some details (see Rushton and Kohler, 1973, p. 170), we follow the principle they used of keeping track of the objective function for possible pair-wise substitutions before selecting the optimal exchange. A variety of techniques have been used in published codes to evaluate and record the objective function values of all possible substitutions. Goodchild and Noronha

(1983), for example, employ an (M * P) swap matrix - where M is the number of candidate nodes and P is the number of facility sites to be located. Teitz and Bart (1968) use an (N * (P + 1)) matrix - where N is the total number of demand nodes on the network - accessing, in turn, each of the remaining (M - P) candidate sites that do not have a facility. Because different substitutions are evaluated before a swap is made, the final results at termination of the algorithms are not necessarily the same even when the initial configuration of nodes is identical. Nor, interestingly, is the number of iterations of the algorithms the same - the swap matrix of Teitz and Bart is constructed in such a way that the number of iterations before termination is not related to problem size whereas the Goodchild and Noronha swap matrix method will generally require more iterations to termination as problem size increases. Both algorithms preserve the property that, at termination, no pairwise swap of a candidate not in the solution for one in the solution will reduce the objective function value.

An alternative strategy for finding the change in the objective function that results from substituting a candidate for a facility location is to build an allocation table. An allocation table has six rows and N columns, where N is the number of demand nodes on the network, each containing information on a demand node. The first four rows store information about the allocation of each node to sites in the current solution: respectively, the identifier of the closest facility's site; the weighted distance from the node to this site; the second closest site's identifier, and the weighted distance to the second site. If a node does not lie within the service areas of two facilities, a zero is entered in place of a site identifier and a penalty value - an arbitrarily large positive number - is used as the weighted distance. The fifth and sixth rows of the table are used to calculate the change in the objective function resulting from a substitution. Each candidate considered for a substitution can service some or all of the demand nodes. For each node that can be served, the candidate's identifier is entered in row five of the node's column and the weighted distance to the candidate is recorded in row six. Nodes that cannot be served by the candidate have values of zero in row five and the penalty in row six of their columns. This information can be compiled by simply entering the information stored in a candidate's candidate string into the appropriate columns of the table. To calculate the change in the objective function, three classes of relations, occurring between the nodes listed in rows one, three and five, have to be found. Each relation yields an incremental change to the objective function - the sum of these individual changes is the net change. The three classes are:

1) A fixed facility is located at the node; as it will always serve itself, no change to the objective function results.

2) The closest site to the demand node, listed in row one of the table, is the site being substituted out. The net change to the objective function is calculated by subtracting the weighted distance in row two from the smaller of the two weighted distances in rows four (to the second site) and six (to the candidate). This method is used in situations where:

    a) the candidate is closer to the node than the first site, yielding a net decrease in the objective function.

    b) the candidate site and the first site are equidistant from the node, yielding no change in the objective function.

    c) the candidate site is further from the node than the first site, but closer than the second site, yielding a net increase in the objective function.

    d) the candidate site is further from the node than both the first and second sites; the node is allocated to the second site and either no change or a net increase in the objective function results.
    e) the candidate cannot serve the node which is allocated to the second site, where one exists, resulting in either no change or a net increase in the objective function.

3) The closest site to the demand node is not being considered for substitution; the candidate, however, is closer to the demand node than the first site. The incremental change, a net decrease, is calculated by subtracting the weighted distance in row two from that in row six.

For every candidate, all three sets of calculations must be evaluated for each substitution.

Figure 2 shows seven columns of an allocation table. It depicts a situation with three facilities - located at nodes one (a fixed facility), eight and ten - which are serving demand nodes one to seven. We are substituting a candidate at node eight for the facility currently located at node nine - row five indicates which of the demand nodes can be served if a facility is located at node nine. The columns of the table exhibit each of the seven situations listed above: column one corresponds to situation 1; columns two to six show situations 2a to 2e respectively; and column seven is situation 3. Figure 3 depicts graphically the configurations of nodes in the allocation table.

Allocation tables are attractive because they economize on RAM for storage and because they minimize the number of times the distance data must be accessed in substituting many candidate sites for any one facility site. As each candidate is processed, the information contained in its candidate string is entered into rows five and six of the table only once - regardless of the number of substitutions which have to be evaluated for that candidate. The economical use of RAM is shown by the small size of the table in relation to typical problem sizes. The table has N columns, each consists of three integers and three real numbers which require eighteen bytes of storage. Consequently, the total storage required for the table, N multiplied by eighteen bytes, increases linearly with the problem size; this is because the length of the table is related to the number of demand nodes, not the number of candidates and facilities, so any problem on a particular network can be solved using the same table. A network with 3,000 demand nodes, for example, necessitates a table of 3,000 columns, using 53.7 kilobytes of storage. This table can accommodate problems ranging from 1 facility and 2 candidates to 3,000 candidates and 2,999 facilities. Whilst it is unlikely that either of these problems would be solved formally, the flexibility of the allocation table is apparent.

Updating the allocation table after a swap

After a candidate has been evaluated for substitution against all the non-fixed sites in the current solution, one or more of the substitutions may lead to a smaller objective function than that of the current solution. The substitution resulting in the biggest decrease in the objective function is made by replacing the old site's node identifier in the current solution with the new site's - a swap. The allocation table has to be updated to reflect the new allocations of demand nodes to facilities.

Figure 4 shows the result of updating the partial allocation table in Figure 2 after swapping the facility at node eight to node nine. Figure 5 depicts the allocation of demand nodes to facilities before the swap and Figure 6 shows the situation after the swap. The steps needed to carry out this update are described below. Note that in addition to the seven situations used to calculate the change in the objective function, another pair must be considered:

1) if the candidate site is closer than the second site, replace the second site's data with the candidate's.

2) a), b) and c) replace the first site's identifier and distance with those of the candidate.

   d) the first site's values are replaced with the second site's; however, it is not known whether the new second closest site should be the candidate or the old third closest site, if one exists, which is not listed in the allocation table.

   e) the first site is replaced with the second site; but it is not known whether a third closest site exists that can become the second closest site.

3) the first site's values replace the second site's, and the candidate is recorded as the first site.

4) two more situations arise where the second closest site is the one being swapped out:

   a) it is not known if the candidate becomes the new second site or if it should be the old third closest site, where one exists.

   b) The node cannot be served by the candidate, and it is not known whether a third closest site exists that can be entered as the new second site.

Situations 3a, 3b, 4a, and 4b all require knowledge about the existence or otherwise of a third closest site, its identifier and weighted distance - this is indicated by the question marks in Figures 4 and 6.

Allocation tables and data structures

The data structure has to provide two sets of information to the allocation table: the first is used to calculate the change in the objective function; the second is needed to update the table after a swap is made. To determine the net change in the objective function for any substitution, rows five and six of the allocation table must contain two lists - the demand nodes that can be served by the candidate site and the weighted distances that would be incurred if these allocations were made. These data are to be found in the candidate's candidate string.

After a swap, additional information, resolving situations 3a, 3b, 4a and 4b, is sometimes needed to update the allocation table. A list of the candidates that can serve a demand node and the associated weighted distances, are required so that secondary facility allocations can be made. This information is contained in the node's demand string. Using candidate strings to provide this list is highly inefficient because an exhaustive search of the strings is necessary.

The data requirements of an allocation table indicate that a data structure consisting of both candidate and demand strings is necessary. This structure minimizes searching and optimizes retrieval of data by both demand node and candidate site; the cost of this rapid access is dual encoding of the data. The storage overhead is reduced by culling data that is not required in solving a particular model. In addition, the features of the data structure can be exploited when calculating the objective function and making a swap, further reducing computation.

Reducing computation

The Teitz and Bart heuristic evaluates the changes in the objective function that would result from substituting a candidate for each of the sites in the current solution. The number of substitutions evaluated during an iteration is directly related to the solution time. The upper bound on the number of substitutions during an iteration is (M * P), where M is the number of candidates and P is the number of facilities. Of these substitutions, however, P^2 represent situations in which a facility location would be substituted for either its existing site or that of another facility; clearly, these substitutions should not be evaluated as they will lead to no change in the objective function. Similarly, if there are X fixed sites, (M * X) substitutions involve relocating those fixed sites and must not be considered. When a model has fixed facilities some substitutions are included in both the P^2 and (M * X) terms; to avoid double counting of these substitutions the number of facilities should be subtracted from the number of candidates, M, yielding a term ((M - P) * X). Consequently, the number of substitutions that must be evaluated during an iteration, S, is given by:

$$S = ((M * P) - P^2 - ((M - P) * X)) \qquad (1)$$

The final term reflects those substitutions of fixed sites which are accounted for in the second term. Table 1 shows the number of substitutions, S, that would be evaluated for a variety of combinations of M, P and X; the value of S decreases considerably as the values of both P and X increase for a given value of M.

To evaluate the number of substitutions, S, using equation 1, the candidate's candidate string can be used to enter weighted distances into the appropriate columns of row six of the allocation table. The objective function for each substitution can be calculated by finding and evaluating the seven situations that determine the net change to the objective function. One way to compute the net change is to examine every column of the table, testing the configuration of the contents for one of the seven conditions. Where there is a large number of demand nodes, and the candidate string is short, this is extremely inefficient.

It is clear from Figure 1 that the demand nodes corresponding to situations 2 and 3 described above are within the service areas of M_1, and M_2. Thus, we need examine only those columns of the table where a node is allocated to a different site if the substitution is implemented as a swap. A rapid method for finding these columns in the allocation table reduces computation dramatically. Of the seven situations in which a node will be re-allocated, six of them arise because the candidate being considered can serve the node. Furthermore, the one condition where this is not the case, 2e, always leads to a net increase or no change in the objective function. Consequently, the nodes in the candidate string are used as pointers to those columns that correspond to situations 1, 2a-d and 3. If the sum of these increments is positive, then there is no point processing the substitution any further - the remaining increments, contributed by columns conforming to situation 2e, cannot decrease the objective function. If the sum is negative, however, the remaining increments must be evaluated. In condition 2e the node can not be served by the candidate and its closest site is being substituted out. One way to find these occurrences, if any, is to search all the elements of row three of the allocation table. The quickest way to find them, however, is to use the candidate string of the site being substituted out. This candidate string lists all the demand nodes which might be served by the site - some or all of these nodes will currently be served by the site. Using the string to find these nodes saves a huge amount of processing where candidate strings are short.

After a swap is made, the columns of the allocation table must be updated. Again, the table can be searched exhaustively, this time for the nine configurations described above, or the two candidate strings can be used as pointers to save processing time.

When all the candidates have been processed the end of an iteration is reached. At this point, Teitz and Bart check whether or not the objective function has decreased during the iteration: if it has not, the heuristic terminates and the current solution is the final solution, otherwise a new iteration is started. This step is inefficient because substitutions evaluated during the penultimate iteration will be re-evaluated needlessly during the final iteration. The average number of substitutions re-evaluated (S a) is:

$$S_a = ((M * P) – P^2 ((M - P) * X)) / 2 \qquad (2)$$

In the worst case, the last swap made involves the first candidate in the list, sufficient substitutions (Sw) will be re-evaluated that nearly one full iteration is added to the solution time:

$$S_w = (((M - 1) * P) - P^2 - (((M -1) - P) * X)) \qquad (3)$$

A simple modification is to use a counter to record the number of candidates processed since the last swap occurred. The counter is checked after each candidate is processed; when it reaches (M - P) a complete set of substitutions, S (given by (1), have been evaluated and the heuristic terminates. If, as experience shows, the heuristic terminates after three or four iterations, this modification saves 12 to 17 percent of the substitutions evaluated during an average run. Table 2 shows the number of substitutions saved for both the average and the worst cases - S_a and S_w - for the configurations of M, P and X presented in Table 1.

Reducing access times to data

Two factors influence data retrieval times: first, how much searching is required to find a piece of data; and second, the physical access time of the medium where the data are stored. Employing a data structure which minimizes searching and resides in random access memory (RAM) will yield low retrieval times. An allocation table necessitates data retrieval by candidate and demand node - so both candidate and demand strings are needed to minimize searching. This dual encoding of the data doubles the data volume, which affects access times because it determines the proportion of the data that can be stored in RAM. Any residual data has to reside on slower peripheral storage. Clearly, the access times achieved in any particular implementation win depend on the available RAM, the type of peripheral storage, and the speed of the bus linking it with the processor.

Reducing data volume: editing techniques

Hillsman (1980) recognized that strings, in addition to being more economical in storage and retrieval than a distance matrix, also may contain information which is not necessary to solve a problem. He pointed out that removing such redundant information from a distance string brings the benefit of reducing both information processing costs and information storage costs. When fixed facilities exist, a demand node will never be allocated to a site which is further away than the closest fixed site in an uncapacitated problem. This knowledge can be used to cull entries from the strings, a process Hillsman terms retrenchment. Demand strings are retrenched by reading each until a fixed facility is found, the remainder of the string is truncated and discarded. Retrenching candidate strings involves removing individual entries rather than truncation: every string has to be read to determine which is the closest fixed facility to each node, and a second pass used to retrench the strings. Where both types of strings are used, the distance to the closest fixed facility is obtained by first retrenching the demand strings; these distances are then used to remove entries from the candidate strings in one pass.

The savings gained from retrenchment are a function of the path radius used to produce the strings, the number of fixed sites, and their location and spatial dispersion. The path radius used to construct the strings may be chosen so that every demand node appears in the string of every candidate; where every demand node is also a candidate, this results in N^2 entries - the equivalent of a distance matrix. For a given set of fixed facilities, the benefits of retrenchment diminish as both the number of candidates and the length of the strings decrease. In contrast, increasing the number of fixed facilities, making their spatial dispersion more uniform, or both, results in a larger cull. We found that no literature exists on the reductions in processing time that, typically, are achieved when Hillsman's editing techniques are used. Nor could we find, in the ten years since his work appeared, any example that had used retrenchment. We demonstrate the effects of retrenchment on the total number of string elements below.

IMPLEMENTATION

A suite of eight programs, for microcomputer-based location-allocation analysis, has been written in Microsoft Pascal, version 3.32, on a PC's Limited 386-16 and a Leading Edge Model D. Microsoft Pascal was chosen because, at the time of development, it supported the use of segmented memory and overlays on Intel 80X86 series processors and 80X87 series math coprocessors. Any IBM PC/XT/AT, PS/2 or true compatible, operating under MS/PC-DOS with 640 kilobytes of RAM and a hard disk, can run this software.

The location-allocation program

The location-allocation program uses an allocation table and a data structure consisting of candidate and demand strings, culled by retrenchment. Large locationallocation problems - defined here as consisting of up to 3,000 nodes, all of which may be candidate facility sites, 12,000 links and up to 1,500 facilities - can be solved. These parameters are somewhat arbitrary and can all be increased at the cost of reduced RAM storage for data; conversely, decreasing the size of the parameters frees more RAM for the storage of string elements. The program's code is broken into seven overlays to maximize the amount of RAM available for data storage; the result is that 54,000 string elements (a demand node's or candidate's identifier, and the associated weighted distance) can be stored.

A flow-chart of the location-allocation program appears in Figure 7. The modularity of the code enables the program to solve both the original formulation of the Teitz and Bart heuristic and the modified version developed by Goodchild and Noronha (1983). Their modification involves selecting the single best substitution from all those evaluated during an iteration, making that swap at the end of the iteration. This approach is useful for locating a single facility, most often as an addition to an existing set of facilities. In this code, for both the Teitz and Bart and the Goodchild and Noronha options, the number of substitutions evaluated is given by $((M * P) - (M * X) - (P - X)^2)$, as described above. To prevent Figure 7 from becoming overly complicated, the counter recording the number of substitutions evaluated since the last actual swap, the logic for constraining the number of substitutions to $((M * P) - (M * X) - (P - X)^2)$, and the associated flows of control, are not shown.

The first stage in the location-allocation program is initialization (see Figure 7): obtaining the names of the string files; prompting the user for the names of output files, if desired, and whether or not to blank the screen to reduce solution times. The user chooses between the solution methods (Teitz and Bart or Goodchild and Noronha's modification) and supplies the initial solution via the keyboard or from a file. This solution is scanned for fixed facilities.

The allocation table has to access candidate strings more frequently than demand strings. Consequently, to fit as many candidate strings as possible in RAM, they are stored first. When retrenchment is possible, the demand strings are read to build a table recording each node's closest fixed facility; this table is used to edit the candidate strings which are then stored; finally, the demand strings are reread, truncated and stored. As stated, 54,000 string elements, each requiring 6 bytes, can be stored in the available RAM. Once this space (approximately 316 kilobytes) has been exhausted, random access files are opened on disk to store the residual strings. A data structure, containing records for every node on the network, indicates whether the node has demand or candidate strings, or both; the start of the strings in RAM or on disk; and the length of each string.

Whilst the strings are being stored, the first four rows of the allocation table are being constructed - the closest and second closest sites in the initial solution, where they exist, are found in the demand strings. The objective function of the initial solution is calculated simply by summing all the values in the second row of the allocation table.

Supporting programs in the system

Three of the eight programs in the system are based on Goodchild and Noronha's PLACE suite, but all are coded in Pascal and can accommodate problems of 3,000 nodes and 12,000 links. Where possible, to make the system familiar to users of the PLACE suite, Goodchild and Noronha's terminology is used in prompting the user for input. Similarly, the shortest path algorithm accepts data in the same format as the PLACE suite to maintain backward compatibility. Programs in our system are selected using a lightbar menu which is automatically recalled to the screen upon termination of each program.

The first option is a program which converts a links file in random format (Goodchild and Noronha, 1983) to ordered format for subsequent processing by a shortest path algorithm. Whilst random format files are easy to produce with a digitizer, they are not an efficient data structure for a shortest path algorithm. The second option is a discrepancy check for links files which ensures that link entries are consistent: in ordered format files bi-directional links must appear in the records of both their origin and destination nodes. The third program is a network contiguity check; it ensures that the network does not consist of two or more sub-graphs by checking that every node on the network is linked to node one. On completion of this test, the links and nodes files are processed by the shortest path algorithm to produce candidate and demand strings.

An editing program prompts the user to select one of six objective functions using a light-bar menu. The six functions are a subset of those that make up the unified linear model (ULM, Hillsman, 1984). These six objective functions can be used: to solve the pmedian problem with a maximum distance constraint, to solve the maximal covering problem, to solve the maximal covering problem with a maximum distance constraint, to identify the minimum number of facilities to be located, to incorporate a user-specified linear distance decay function, and, finally, to place a user-specified exponent on the distance values (d_{ij}). (Note that the editing program is not used if the user is solving an unconstrained p-median problem.) These objective functions, like all those in the

ULM, are derived by editing the coefficients of the objective function: the contents of the strings files. The editing program processes pairs of candidate and demand strings files, ensuring that both are edited using the same parameters. The final program in the system generates a series of statistical reports on the solution to the model. The user can open the system's output files and scroll through them, one screen at a time, using the menu program, obviating the need to return to the operating system.

TESTING AND PERFORMANCE

Testing

Testing the accuracy and the speed of the allocation table software proved difficult for large problems. Goodchild and Noronha's software can accommodate problems with no more than 500 nodes and, as noted in above, their algorithm sometimes yields different results from Teitz and Bart's. Consequently, two additional programs were written to test the allocation table code, both can accommodate up to 3,000 nodes. Test Program 1 is a Pascal version of PLACE (Goodchild and Noronha's code), it uses a swap matrix and demand strings, which are stored in RAM. In contrast, Test Program 2 has an allocation table, employs the Teitz and Bart algorithm, uses demand strings stored on the hard disk, and, instead of reducing computation by using nodes in the candidate strings as pointers to columns in the allocation table, it exhaustively searches all N columns to calculate the change in the objective function and to update the table. Testing proceeded in the following manner:

1) Results produced by PLACE, Test Program 1, and the Goodchild and Noronha option in the allocation table code were compared for problems of up to 500 nodes. Results from Test Program 1 and the Goodchild and Noronha option were compared for larger problems.

2) Solutions generated by the allocation table code and Test Program 2 were compared against each other for the same data sets used in stage 1.

3) Because Goodchild and Noronha's algorithm and Teitz and Bart's do not always yield the same solutions, the same problems were ran through Test Program 1, Test Program 2 and both options in the allocation table code with all but one facility fixed - in this instance all codes must yield identical solutions if they are working correctly.

Since testing, the system has been used at the State University of New York at Buffalo and at the University of Iowa to work with data sets derived from USGS digital line graphs and various other digitized networks - no problems have been found to date. The Iowa Department of Transportation used the system intensively between November, 1987 and April, 1988 - on an IBM PS/2 Model 80 - and no errors were discovered.

Many of the error traps, and associated messages, used to debug the code are retained. Most of them involve logical operators or numerical comparisons to ensure the integrity of data storage and retrieval, undoubtedly resulting in slower execution times. The experiences reported above suggest that some of these error traps could be removed to enhance performance further without exposing the user to an increased risk of erroneous results.

Performance

Performance statistics are presented for four data sets. 'Me characteristics of these data sets are summarized in Table 3. The first data set is the London, Ontario, fire station example used by Goodchild and Noronha (1983); it contains 150 nodes, all of which are candidates. The second data set consists of 300 nodes, all candidates, which form a square lattice 15 nodes wide by 20 nodes long. All but the edge and comer nodes have a valency of four. Similarly, there are 500 nodes in the third data set which form a square lattice 20 nodes wide by 25 nodes long. The fourth data is a state-wide road network developed by the Iowa Department of Transportation (DOT) for locating road maintenance garages; it contains 2,844 nodes, 2,764 candidates and 6,372 links. In Table 3, statistics are presented on the length of the candidate and demand strings and the number of swaps evaluated per iteration. The first set of statistics result when all the facilities in each data set are mobile. The second set of statistics are obtained when a randomly selected subset of the facilities in the solution to the all-mobile problem are fixed and the problem is solved again.

The design of the allocation table software makes it best suited to large problems with short candidate strings. The London problem is small, all the nodes are candidates, and the candidate and demand strings are short. Data sets Two and Three are both medium-sized problems with very long candidate and demand strings. In both cases, the internode distance in the lattice is 10 miles, but a very long "path radius" of 100 miles was used to generate the strings. The Iowa DOT problem, however, is large and was ran with a "path radius" of 25 miles - a small distance when compared with the dimensions of the state - generating short strings.

Typical solution times for all four data sets are presented in Table 4 for the allocation table code, Test Program 1, Test Program 2 and the PLACE suite. (All results are for a Dell System 310 with an ISA bus, 20 Mhz 80386 and 80387 chips, a 150 Mb ESDI hard-disk, and a color VGA graphics card and monitor, no disk-caching or other performance enhancing software was used). Because the other codes do not use retrenchment, solution times for both the all-mobile and the fixed facility data sets are reported only for the allocation table code.

The results in Table 4 can be used to determine the effects of our implementation strategies on solution times. To help gauge the effects of these strategies, an index is reported with the allocation table solution times, without the benefit of retrenchment, taking a value of 100 for each data set. The results for the PLACE suite are included to provide a frame of reference. 'Me Pascal and BASIC codes should not be directly compared for several reasons. First, the PLACE suite was written for a very restrictive computing environment, severely constraining the options open to its authors. Second, the Intel 80X86 series of chips running under DOS have a segmented memory architecture, not a "flat" or linear address space. The PLACE suite, because it is written in BASIC, is constrained to a 64 Kb segment and uses only NEAR calls to access data. The Pascal code, however, uses multiple 64 Kb segments for data storage and has to use FAR calls to access them. Third, as noted above, the Pascal codes have a considerable amount of diagnostic and error checking code which adds a considerable overhead to their operation. With these caveats in mind, Table 4 shows that, as expected, the difference in solution times between the PLACE suite and the allocation table code increases as problems get larger.

The results for the Pascal codes can be compared directly. Test Program 1 uses the Goodchild and Noronha swap matrix with the demand strings stored in RAM. Although this program is faster than Test Program 2, it is slower than the allocation table code. The results we have obtained from this code bear out our contention that solution times from the Goodchild and Noronha algorithm do increase at a greater than linear rate with the size of the problem. Test Program 2 uses an allocation table, but only with demand strings stored on disk and without the benefit of the pointers into the table to minimize computation. This program uses the approach suggested by Teitz and Bart (1968), in which for each and every substitution they determine, the allocation of every demand node on the network is determined to evaluate the objective function. Because it does not exploit the spatial structure of the problems it is solving, this program is much slower than the allocation table code. On Data Set Two, the savings reaped from using both types of string, storing them in RAM, and exploiting the spatial structure to minimize calculation amounts to two orders of magnitude. When the additional savings gained from retrenchment are factored in, the solution time from Test Program 2 is over 32 times larger than that from the allocation table code; it is over 4 times larger for Data Set 1. These and other, unreported results suggest that an allocation table, unsupported by the other strategies, is a poor implementation strategy. When implemented together, however, our strategies yield solution times which increase almost linearly with the size of the problem.

Table 4 shows that retrenchment is beneficial for even small problems. Every element culled from a candidate string saves, on average, four columns of the allocation table from being processed - one in each of four iterations. Savings from retrenching demand strings are much harder to quantify because the pattern and number of string reads are problem- specific. The number of string elements culled when retrenchment is applied to the four data sets is shown in Table 5. For the London problem with two fixed centers, roughly nineteen percent of the string elements were culled to reduce total run time by two percent. This problem is small enough to fit entirely in RAM whether retrenchment takes place or not, thus, the saving reflects fewer string elements being processed. Initially, not all of the string elements fit in RAM for Data Sets 2 and 3. Thirty-five per cent of the string elements in Data Set 2 have to be stored on disk compared with sixty-five per cent for Data Set 3. After retrenchment, both data sets fit entirely in RAM. This accounts for the much greater reduction in the solution time after retrenchment for Data Set 3 than for Data Set 2 reported in Table 4. For the Iowa DOT problem, the nearly sixty-one percent cull rate helped to produce a sixty percent reduction in total run time. Sufficient elements were culled from the candidate strings to store them all, plus a few demand strings, in RAM. The sixty percent saving is less than that obtained for Data Sets 2 and 3 because many of the demand string elements had to be stored on disk.

The saving in total run time to be gained from our implementation strategies is a function of many variables including: the proportion of string elements in RAM; the speed of the processor and the memory chips used in the system; the speed and bandwidth of the bus between the memory and the processor, and the seek time and throughput of the hard disk subsystem. Consequently, any findings on this relation will be machine-specific. Nevertheless, the key finding on performance is that run times for the allocation table approach, with dual encoding of distance strings and retrenchment of those strings, increase approximately linearly with problem size whereas other approaches tend to increase by the square of the problem size. This difference is significant as the discussion of problems of implementing location-allocation models moves from the context of classroom exercises to the world of real problems.

FURTHER DEVELOPMENT

The system is designed to be modified easily. Under MS/PC- DOS, larger problems can be accommodated by increasing the length of the allocation table, at the cost of storing fewer string elements in RAM. Porting the system to other platforms with operating systems that support larger address spaces (a Sun SPARCstation running Unix, for example) will permit larger problems to be solved with all the string elements stored in RAM, reducing solution times. Running on an 80386 or 80486 PC, a 32-bit version of the code would provide an approximately fourfold increase in performance over the standard version because of the increased RAM available for storage and the faster execution of such code. The benefits to be reaped from such ports, allied with the rapid pace of advance in Microcomputer capabilities, suggest that the run time of the Iowa DOT problem (Table 1) could be reduced to under an hour; similarly, the London fire station problem could be solved in a matter of seconds. While these developments will increase the speed of solution, they will not address some shortcomings of the code: lack of integrated data management and graphical display capabilities.

There are two approaches to exploiting the allocation table code to yield a spatial decision support system (SDSS) environment. The first approach is to link the code to a GIS. Existing systems, such as ARC/INF0 (ESRI, 1987) and TransCAD (Caliper Corporation, 1988), provide the capabilities needed to link external analysis procedures, to export data sets from their DBMS, and to import and display the results of analyses. Taken in isolation, the data requirements of the location-allocation code are modest and can be supported by the relational databases underlying many GIS. Specifically, three ASCII files must be stored and output: a nodes file, a links file, and an initial solution file, all of which are tabular in form. The nodes file contains information on the weight, region (spatial partition), candidacy and locational coordinates of each node; the links file contains the origin and destination node of each link and its length in some metric; and the initial solution file lists the facility sites and whether they are mobile or fixed. The nodes and links files are used to generate candidate and demand strings. To enable solutions to be displayed in a graphical form, the location-allocation code outputs an ASCII file. Each line of this file contains the identifier of a demand node followed by the identifier of the facility from which it is served and the weighted distance incurred by this allocation. The information in the file is obtained simply by reading the contents of the first and second rows of each column of the allocation table. Willer (1990) has linked the allocation table code to TransCAD (Caliper, 1988); TransCAD's relational database and its SQL module are used to generate the three data files and its graphical capabilities display the solutions to models. In contrast, Armstrong *et al*. (1991) describe a SDSS that uses the allocation table code with a mapping package and a relational DBMS.

The second approach to exploiting the location-allocation code is to embed it in a set of custom-coded modules. Armstrong and Densham (1990), for example, describe a database, implemented using the extended network data model, which is designed to support spatial query, locational modelling and cartographic display. The advantage of this second approach is that a more integrated system can be developed. The shortest path algorithm, for instance, can query the database directly, extracting the contents of the nodes and links files and writing them directly into RAM for immediate processing, saving the overhead of writing the files to disk from the database and then reading them into the shortest path algorithm. Similarly, a graphics module can read the allocations in the solution directly from the allocation table, rather than via an intermediate file. Densham and Armstrong (1987) describe a prototype system of this type.

CONCLUSIONS

We have presented a series of principles for implementing the Teitz and Bart (1968) vertex substitution heuristic that minimize three fundamental determinants of solution times: the number of calculations carried out; the volume of data to be processed, and access times to that data. Using both candidate strings and demand strings, we have shown that it is possible to access efficiently and then process data only for the two local geographic areas that are affected when a facility is relocated. Ile size of these two areas, the service area of the candidate site and the service area of the site in the solution set, are independent of the size of the total area being studied: they are a function both of the number and of the current locational arrangement of facilities. By implementing strategies which focus on these two areas, solution times increase in an approximately linear way with problem size. This approach permits, for the first time, the solution of large location-allocation problems in a microcomputer environment and thus paves the way for wider practical application of location-allocation methods.

The principles developed in this paper, if adopted by developers of software systems for locational. analysis, would reverse the recent trend toward constructing analysis systems that merely evaluate subjective location choices and furnish spatial accounting information to the decision-maker (see for example, Lupien and Moreland, 1987; ESRI, 1987; Caliper Corporation, 1988). Although valuable in many cases and essential in some, spatial accounting techniques fall short of the space-searching capability of location-allocation methods that can find single or multiple locations that optimally meet the decision-maker's stated objectives. In short, current approaches do not bring to the locational decision-maker the fruits of twenty-five years of algorithmic developments in location-allocation analysis methods. These systems solve location/allocation rather than location-allocation problems.

The solution times yielded by software developed with these new principles suggest that location-allocation models can be used during microcomputer-based, interactive decision-making sessions. Moreover, location-allocation models can be integrated with location/allocation and intuitive, graphics-based approaches to locational decision-making to provide more effective decision support for locational decision-makers. These systems, with integrated database management, analysis, display and reporting capabilities, will truly be spatial decision support systems.

ACKNOWLEDGEMENTS

Partial support for this research was provided by the Iowa Department of Transportation, the National Science Foundation (SES-8714103), and The Midwest Transportation Center, Project Number 7. We also thank John Whited, William McCall and the staff of the Iowa Department of Transportation's Maintenance Division for their help in defining the problems, developing the data base and testing the location-allocation programs for the Iowa problems. This paper is a contribution to Research Initiative 6 (Spatial Decision Support Systems) of the National Center for Geographic Information and Analysis, supported by the National Science Foundation (SES-8810917).

REFERENCES

Armstrong, M.P. and Densham, P.J., 1990. Database organization alternatives for spatial decision support systems. International Journal of Geographic Information Systems. 4: 3-20.

Armstrong, M.P., Rushton, G., Honey, R., Dalziel, B., Lolonis, P., De, S. and Densham, P.J., 1991. A spatial decision support system for regionalizing service delivery systems. Computers, Environment and Urban Systems. 15: in press.

Beaumont, J.R., 1988. Editorial Environment and Planning A. 20: 1141-1144.

Caliper Corporation (1988. TransCAD Users Manual V. 1.20 (Caliper Corporation, Newton).

Casillas, P., 1987. Aggregation problems in location-allocation modeling. In: Spatial Analysis and Location-Allocation Models eds. A. Ghosh and G. Rushton (Van Nostrand Reinhold, New York) pp 327-344.

Cooper, L., 1964. Heuristic methods for location-allocation problems. SIAM Review. 6: 37-52.

Cooper, L., 1967. Solutions of generalized locational. equilibrium models. Journal of Regional Science. 7: 2-18.

Current, J. and Schilling, D., 1987. Elimination of source A and B errors in p-median location problems. Geographical Analysis. 19: 95-110.

Current, J. and Schilling, D., 1990. Analysis of errors due to demand data aggregation in the set covering and maximal covering location problem. Geographical Analysis. 22: 116-126.

Densham, P.J. and Rushton, G., 1988. Decision support systems for locational planning. In: Behavioural Modelling in Geography and Planning eds. R. Golledge and H. Timmermans (Croom-Helm, London) pp. 56-90.

Densham, P.J.. and Armstrong, M.P., 1987. A spatial decision support system for locational planning: design, implementation and operation. Proceedings, Auto Carto 8 (ACSM, Falls Church. pp. 112-121.

ESRI (1987. ARCIINFO Network Users Guide (Environmental Systems Research Institute, Redlands).

Feldman, E., Lehrer, F.A. and Ray, T.L., 1966. Warehouse locations under continuous economies of scale. Management Science. 12: 670-684.

Goodchild, M.F., 1978. Spatial choice in location-allocation problems: the role of endogenous attraction. Geographical Analysis. 13: 65-72.

Goodchild, M.F. and Noronha, V., 1983. Location-Allocationfor Small Computers. Monograph No. 8, Department of Geography (The University of Iowa, Iowa City).

Hillsman, E.L., 1980. Heuristic Solutions to Location-Allocation Problems: A Users' Guide to ALLOC IV, V, and VI. Monograph No. 7, Department of Geography (The University of Iowa, Iowa City).

Hillsman, E.L., 1984. The p-median structure as a unified linear model for locationallocation analysis. Environment and Planning A 16: 305-318.

Kuehn, A.A. and Hamburger, M.J., 1963. A heuristic program for locating warehouses. Management Science. 9: 643-666.

Lupien, A.E., Moreland, W.H. and Dangermond, J., 1987. Network analysis in geographic information systems. Photogrammetric Engineering and Remote Sensing. 53:1417-1421.

Maranzana, F.E., 1964. On the location of supply points to minimize transport costs. Operational Research Quarterly. 15: 261-270.

Newton, P.W., Taylor, M.A.P. and Sharpe, R., 1988. Desktop Planning (Hargreen, Melbourne).

Robinson, V.B., 1983. Urban Data Management Software (U.D.M.S.) Package - User's Manual (United Nations Center for Human Settlements (HABITAT), Nairobi).

Rosing, K.E., I-Mlsman, E.L. and Rosing-Vogelaar, H., 1979. A note comparing optimal and heuristic solutions to the p-median problem. Geographical Analysis. 11: 86-89.

Rushton, G. and J A Kohler, J.A., 1973. ALLOC: Heuristic solutions to multifacility location problems on a graph. In: Computer Prograrnsfor Location-Allocation Problems eds. G. Rushton, M.F. Goodchild and L.M. Ostresh, Monograph No. 6, Department of Geography (The University of Iowa, Iowa City).

Schneider, J.B., Miller, D.G. and Friedman, T.W., 1976. Locating and sizing park-ride lots with interactive computer graphics. Transportation. 5: 389-406.

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

Willer, D.J., 1990. A Spatial Decision Support Systemfor Bank Location: A Case Study. NCGIA Technical Report 90-9 (National Center for Geographic Information and Analysis, Santa Barbara).

Figure 1: Potential and actual service areas affected by a substitution.

M_1 and M_2 are candidate locations, candidate M_2 is being evaluated as a site for the facility currently located at M_l.

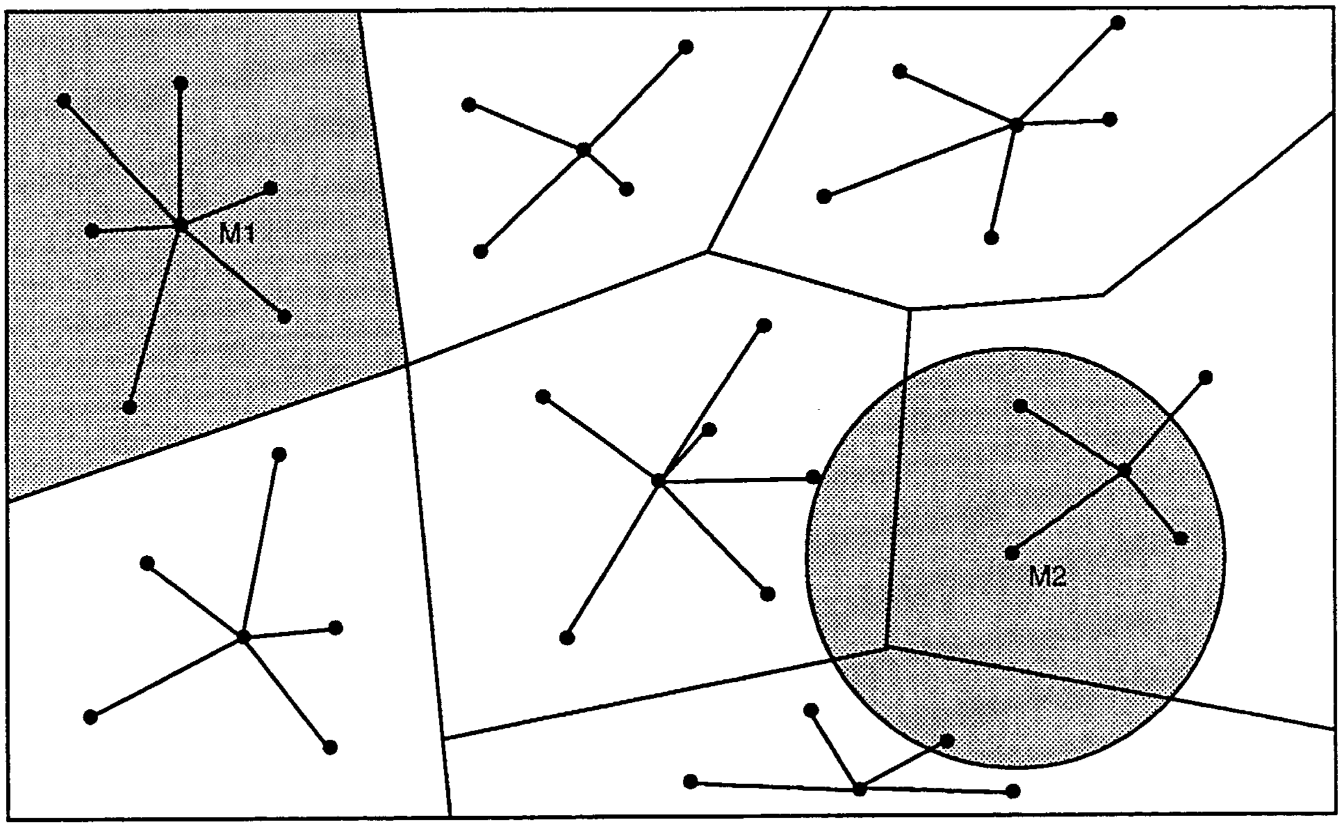


Figure 2: Using an allocation table to calculate changes in the objective function

Candidates in current solution are 8 and 10. Fixed facility at candidate 1.

1	2a	2b	2c	2d	2e	3	**Situation**
1	2	3	4	5	6	7	**Demand nodes**

Substituting candidate 9 for candidate 8:

Row								
								1st facility:
1	1	8	8	8	8	8	10	identifier
2	0	8	7	4	4	2	3	weighted dist.
								2nd facility:
3	8	1	1	1	1	1	1	identifier
4	6	9	8	7	7	5	4	weighted dist.
								Candidate:
5	9	9	9	9	9	0	9	identifier
6	3	5	7	5	8	P	2	weighted dist.
	0	-	0	+	+ 0	+ 0	-	**Net change in objective function**

Figure 3: Geometric configurations of nodes and facilities used to calculate changes in the objective function

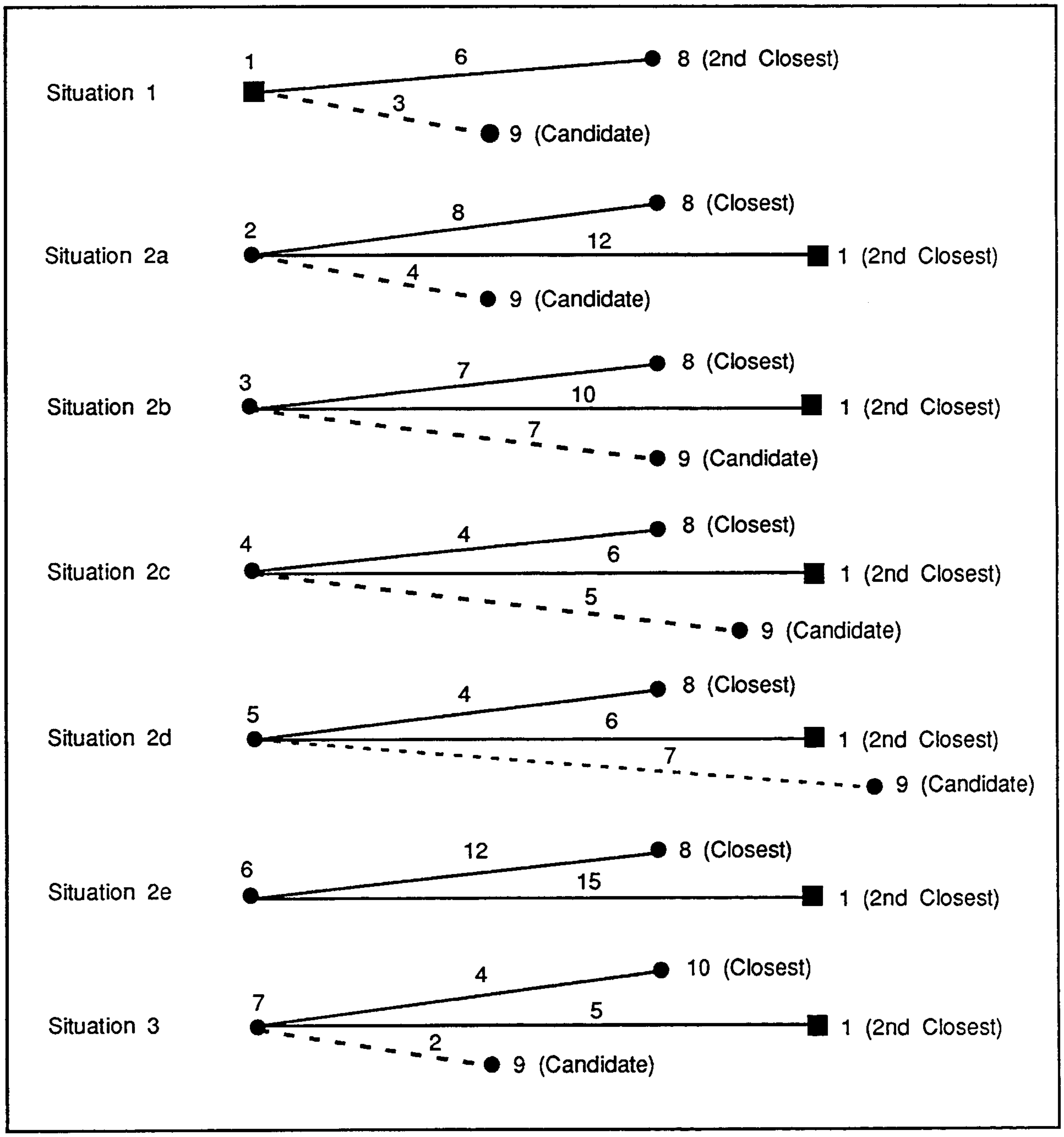


Figure 4: Making a swap

Fixed facility at candidate 1, swapping out candidate 8 and swapping in candidate 9.

	1	2a	2b	2c	2d	2e	3	4a	4b	**Situation**
	1	2	3	4	5	6	7	11	12	**Demand nodes**

Substituting candidate 9 for candidate 8:

Row										
										1st facility:
1	1	8	8	8	8	8	10	10	10	identifier
2	0	8	7	4	4	2	3	7	8	weighted dist.
										2nd facility:
3	8	1	1	1	1	1	1	8	8	identifier
4	6	9	8	7	7	5	4	8	10	weighted dist.
										Candidate:
5	9	9	9	9	9	0	9	9	0	identifier
6	3	5	7	5	8	P	2	9	P	weighted dist.
	0	-	0	+	+ 0	+ 0	-			**Net change in objective func.**

Swapping out candidate 8 and swapping in candidate 9:

Row										
										1st facility:
1	1	9	9	9	1	1	9	10	10	identifier
2	0	5	7	5	7	5	2	7	8	weighted dist.
										2nd facility:
3	9	1	1	1	?	?	10	?	?	identifier
4	3	9	8	7	?	?	3	?	?	weighted dist.
										Candidate:
5	9	9	9	9	9	0	9	9	0	identifier
6	3	5	7	5	8	P	2	9	P	weighted dist.
					*	*		*	*	

Figure 5: Geometric configurations before a swap

Fixed facility at candidate 1, swapping out candidate 8 and swapping in candidate 9.

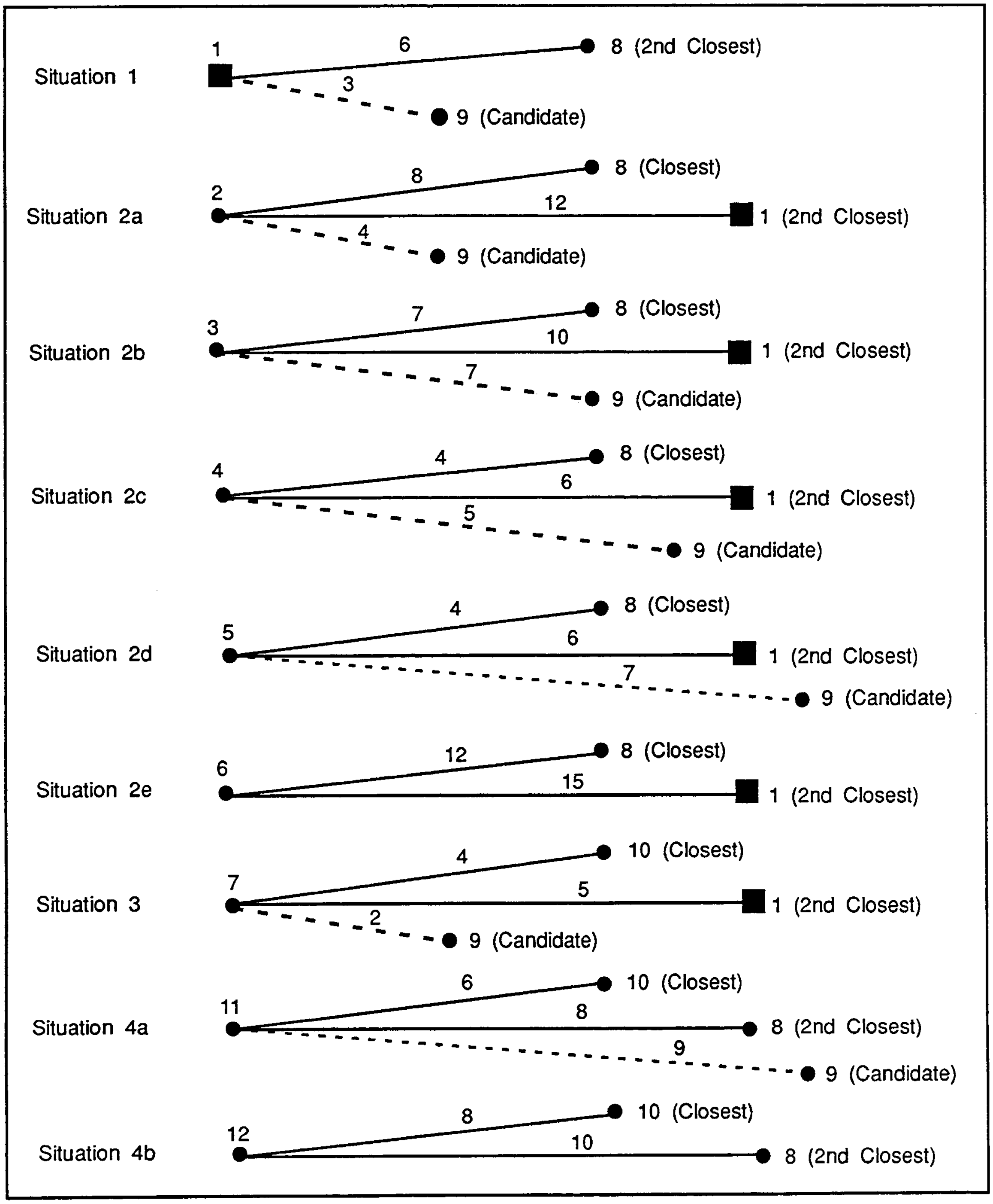


Figure 6: Geometric configurations after a swap

Fixed facility at candidate 1, swapping out candidate 8 and swapping in candidate 9.

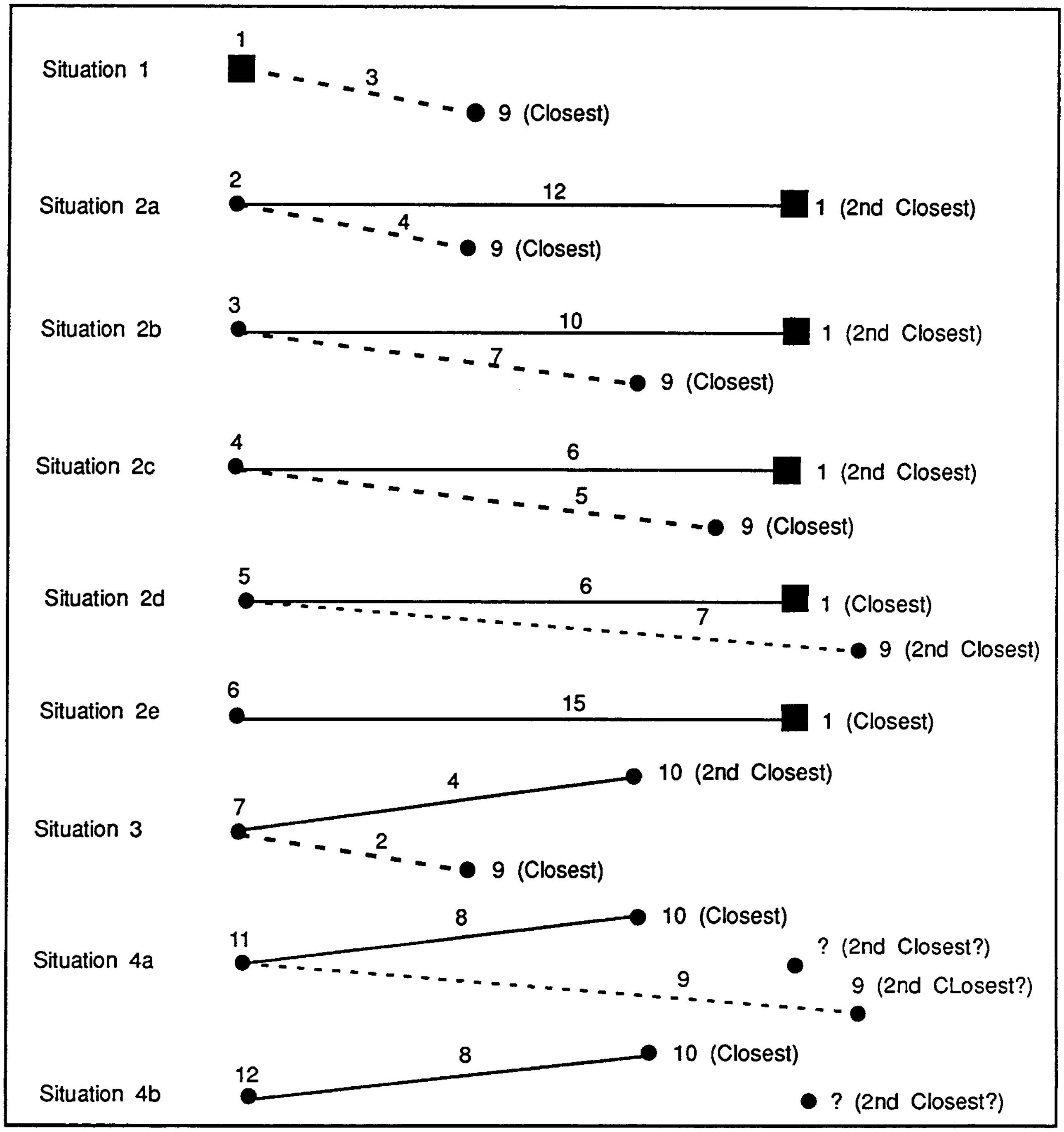


Figure 7: Flowchart of implemented location-allocation heuristic

(CS = Candidate Strings, DS = Demand Strings)

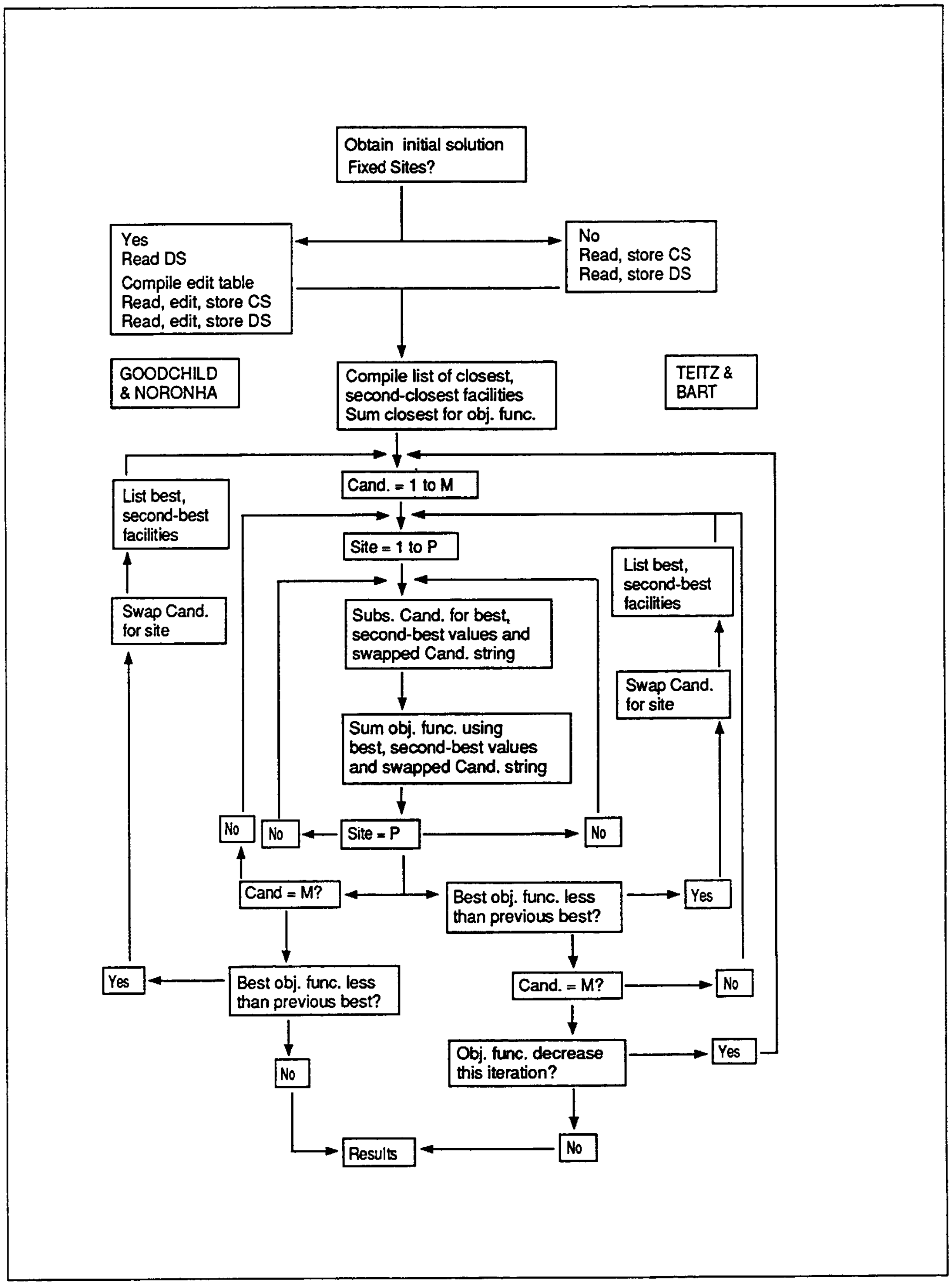

Obtain initial solution
Fixed Sites?
Yes
Read DS
Compile edit table
Read, edit, store CS
Read, edit, store DS
No
Read, store CS
Read, store DS
GOODCHILD & NORONHA
Compile list of closest, second-closest facilities
Sum closest for obj. func.
TEITZ & BART
Cand. = 1 to M
List best, second-best facilities
Site = 1 to P
List best, second-best facilities
Swap Cand. for site
Subs. Cand. for best, second-best values and swapped Cand. string
Swap Cand. for site
Sum obj. func. using best, second-best values and swapped Cand. string
No
No
Site = P
No
Cand = M?
Best obj. func. less than previous best?
Yes
Yes
Best obj. func. less than previous best?
Cand. = M?
No
No
Obj. func. decrease this iteration?
Yes
Results
No

Table 1: Substitutions to be evaluated during an iteration with various configurations of candidates, facilities and fixed facilities

M	P	X	(M * P)	S	%
30	5	0	150	125	83.3
30	5	2	150	75	50.0
30	15	0	450	225	50.0
30	15	7	450	120	26.7
30	25	0	750	125	16.7
30	25	12	750	65	8.7
300	50	0	15,000	12,500	83.3
300	50	25	15,000	6,250	41.7
300	150	0	45,000	22,500	50.0
300	150	75	45,000	11,250	25.0
300	250	0	75,000	12,500	16.7
300	250	125	75,000	6,250	8.3
3,000	500	0	1,500,000	1,250,000	83.3
3,000	500	250	1,500,000	625,000	41.7
3,000	1,500	0	4,500,000	2,250,000	50.0
3,000	1,500	750	4,500,000	1,125,000	25.0
3,000	2,500	0	7,500,000	1,250,000	16.7
3,000	2,500	1,250	7,500,000	625,000	8.3

M	is the number of candidate facility locations,
P	is the number of facilities to be located,
X	is the number of fixed facilities,
(M * P)	is the upper bound on the number of substitutions,
S	is the actual number of substitutions
%	is S expressed as a percentage of (M * P).

Table 2: Reduction in processing achieved using a counter for the number of substitutions evaluated

M	P	X	(M * P)	S	%	S_a	S_w
30	5	0	150	125	83.3	63	120
30	5	2	150	75	50.0	38	68
30	15	0	450	225	50.0	113	210
30	15	7	450	120	26.7	60	98
30	25	0	750	125	16.7	63	100
30	25	12	750	65	8.7	33	28
300	50	0	15,000	12,500	83.3	6,250	12,450
300	50	25	15,000	6,250	41.7	3,125	6,175
300	150	0	45,000	22,500	50.0	11,250	22,350
300	150	75	45,000	11,250	25.0	5,625	11,025
300	250	0	75,000	12,500	16.7	6,250	12,250
300	250	125	75,000	6,250	8.3	3,125	5,875
3,000	500	0	1,500,000	1,250,000	83.3	625,000	1,249,500
3,000	500	250	1,500,000	625,000	41.7	312,500	624,250
3,000	1,500	0	4,500,000	2,250,000	50.0	1,125,000	2,248,500
3,000	1,500	750	4,500,000	1,125,000	25.0	562,500	1,122,750
3,000	2,500	0	7,500,000	1,250,000	16.7	625,000	1,247,500
3,000	2,500	1,250	7,500,000	625,000	8.3	312,500	621,250

M	is the number of candidate facility locations,
P	is the number of facilities to be located,
X	is the number of fixed facilities,
(M * P)	is the upper bound on the number of substitutions,
S	is the number of substitutions evaluated per iteration,
%	is S expressed as a percentage of (M * P),
S_a	is the average number of substitutions saved, and
S_w	is the number of substitutions saved in the worst case.

Table 3: Test data sets

Characteristics:	Data Sets: One (London)	Two	Three	Four (Iowa)
Demand nodes	150	300	500	2844
Candidates	150	300	500	2764
No. of Facilities:	9	18	30	135
- substitutions per iteration	1269	5076	14,910	354,915
- average length of a demand string	65	138	156	49
- average length of a candidate string	65	138	156	50
No. of Fixed facilities:	2	6	10	80
- substitutions per iteration	1025	3456	9,600	148,995
- average length of a demand string	53	51	33	21
- average length of a candidate string	53	57	39	20

Table 4: Solution times (hours/minutes/seconds) for data sets

	Data Sets:			
	One (London)	**Two**	**Three**	**Four** (Iowa)
Number of nodes:	150	300	500	2844
Allocation Table:				
- With strings retrenched	0:04:26 96%	0:20:41 46%	0:50:56 12%	3:23:17 40%
- No retrenchment of strings	0:04:37 100%	0:44:55 100%	7:08:22 100%	8:28:49 100%
Test Program 1: (Swap matrix approach)	0:16:55 366%	4:09:17 555%	(1)	(1)
Test Program 2: (Allocation table with demand strings)	0:20:14 438%	11:05:28 14,816%	(2)	(2)
Place suite:				
- Interpreted BASIC (Swap matrix)	1:05:22 1,416%	(2)	(2)	(2)
- Compiled BASIC (Swap matrix)	0:06:20 137%	3:36:15 482%	63:58:39 896%	(3)

All times in hours, minutes and seconds.

(1)These data sets contain more than 50,000 string elements, exceeding the capacity of the program.

(2)Solutions were not generated for these data sets because of the run times involved.

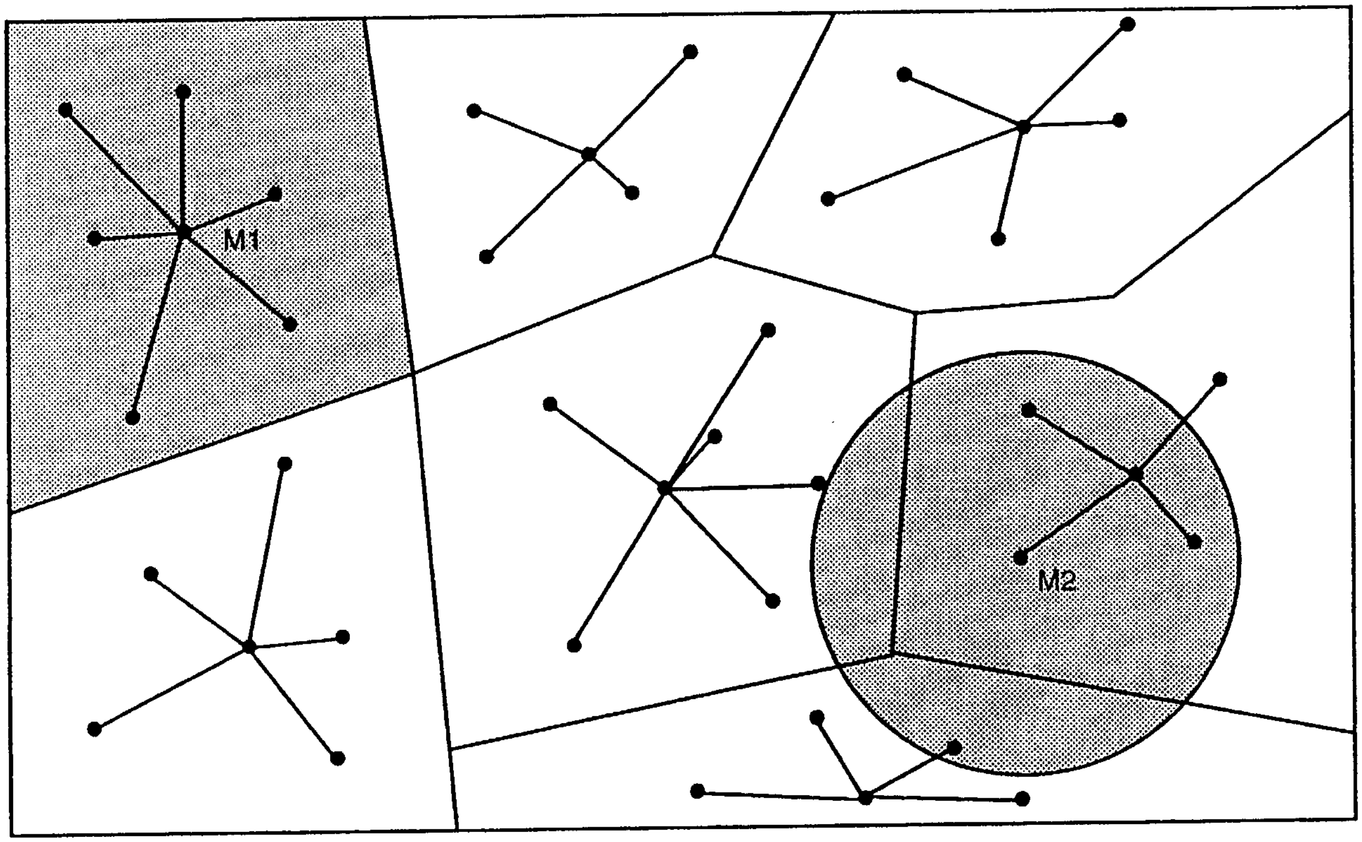
(3)Data set exceeds the 500 node limit of the PLACE suite.

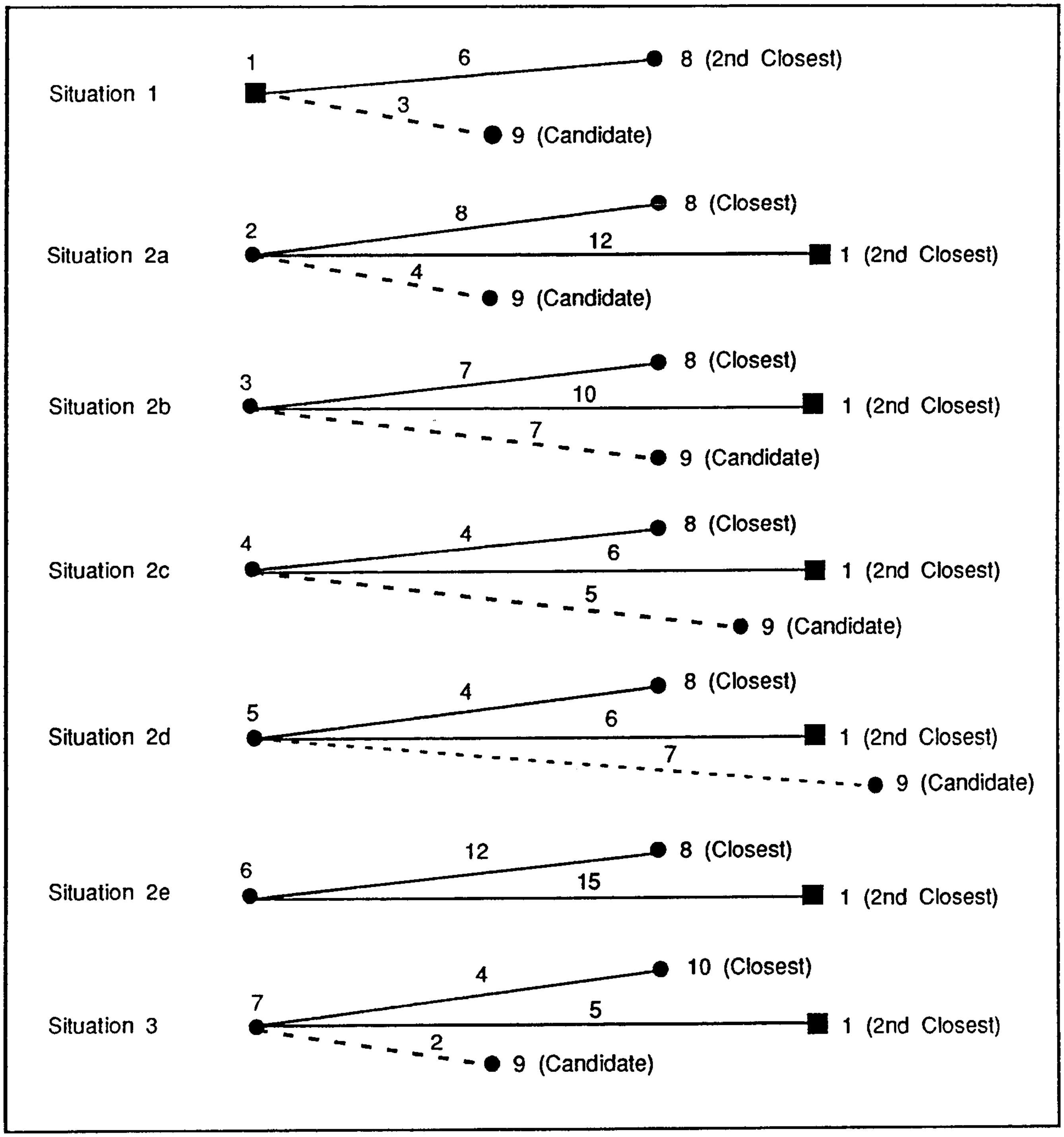
(N.B. These results are machine-specific, see text for an explanation.)

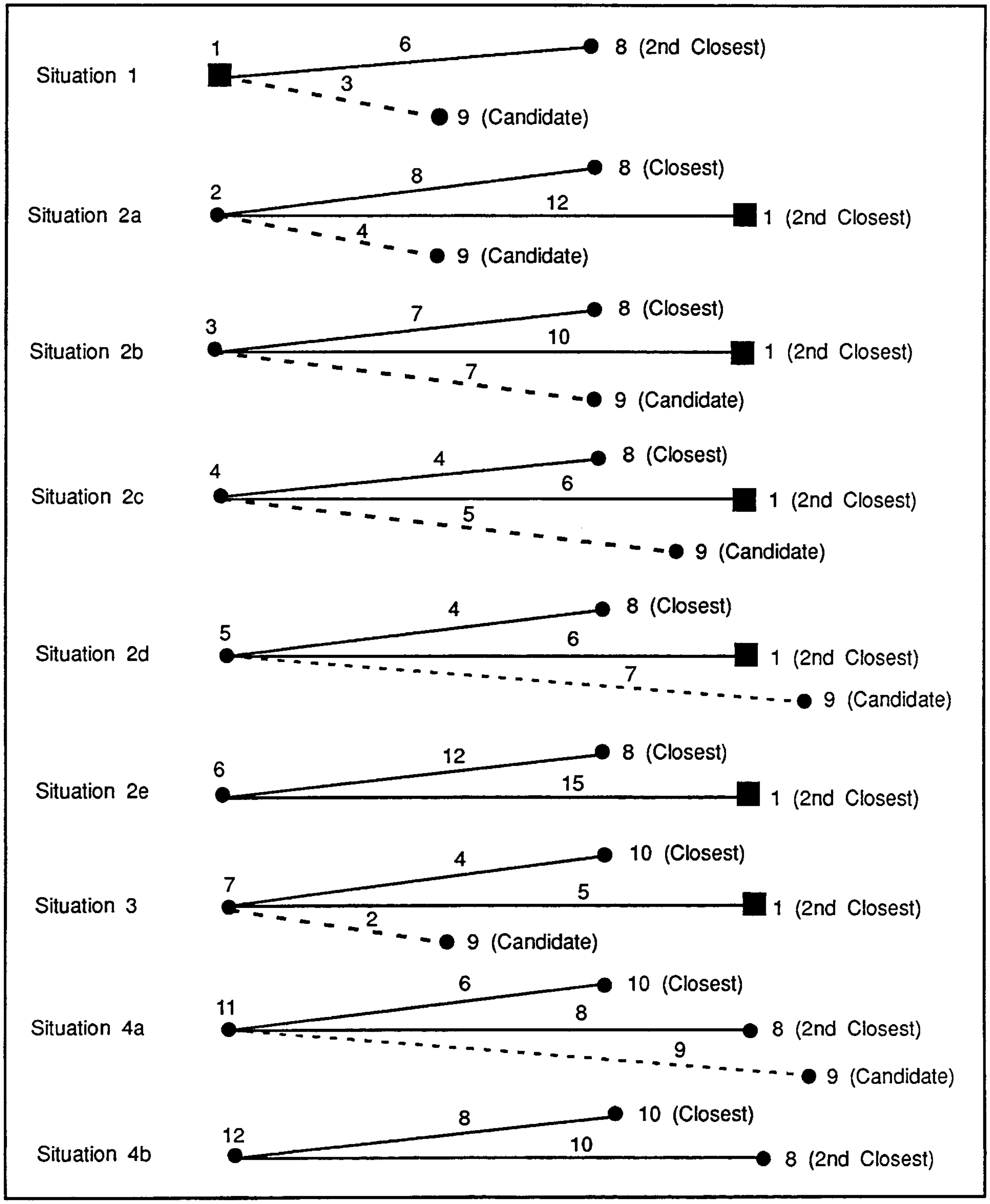
Table 5: Effects of retrenchment on the length of the candidate and demand strings

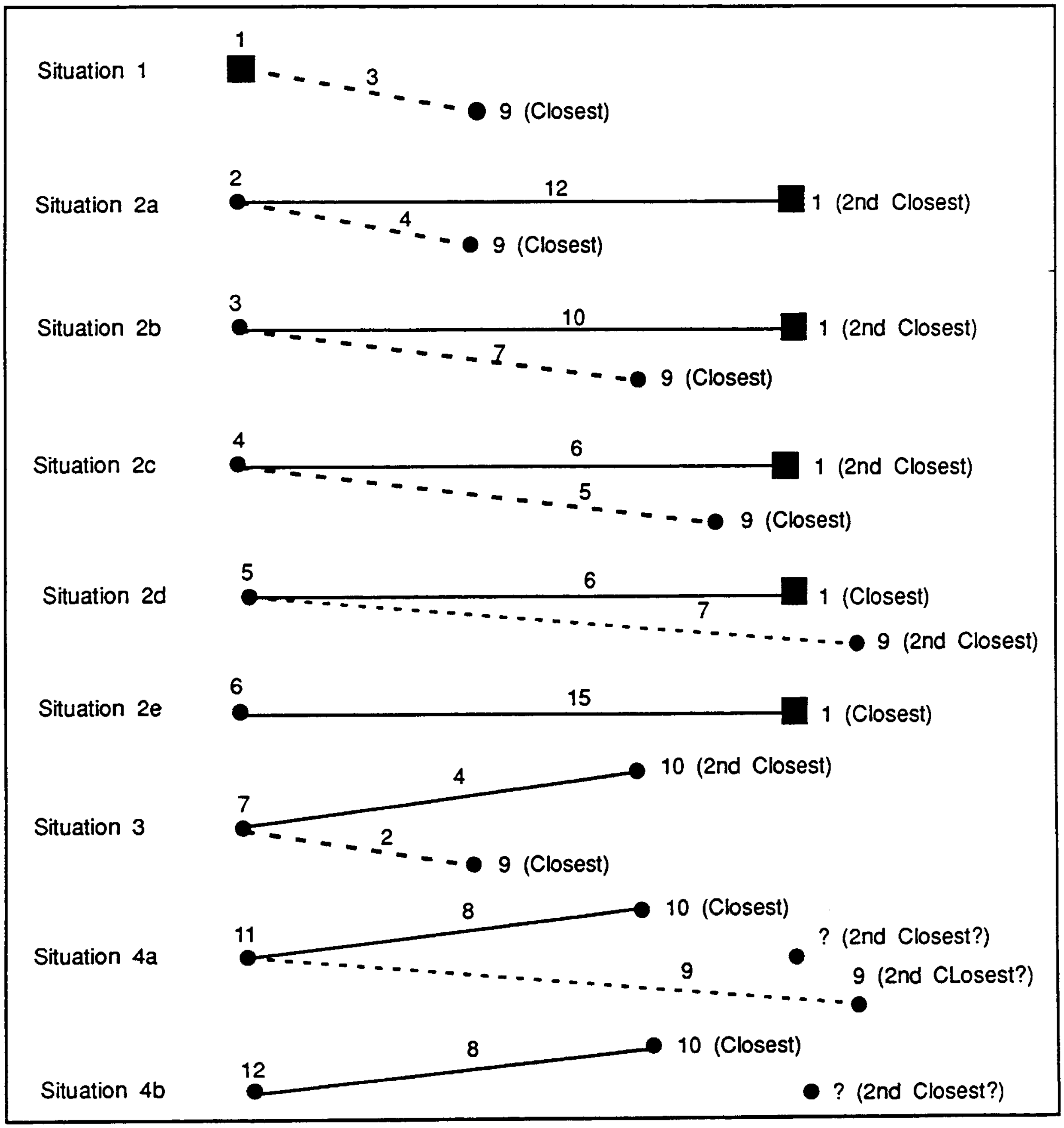
	Data Sets:			
	One (London)	**Two**	**Three**	**Four** (Iowa)
Demand nodes:	150	300	500	2844
Candidates:	150	300	500	2764
Facilities:	9	18	30	135
Fixed facilities:	2	6	10	80
Demand strings:	150	300	500	2844
- elements (read)	9,706	41,330	77,830	139,442
- elements stored	7,872	15,266	16,301	54,918
- elements culled	1,834	26,064	61,529	84,524
Candidate strings:	150	300	500	2764
- elements (read)	9,706	41,330	77,830	137,419
- elements stored	7,875	17,149	19,454	53,765
- elements culled	1,831	24,181	58,376	83,654
Total cull:	18.9%	60.8%	77.0%	60.8%

(N.B. Varying either the chosen "path radius" or the locations of the fixed facilities will alter these results.)

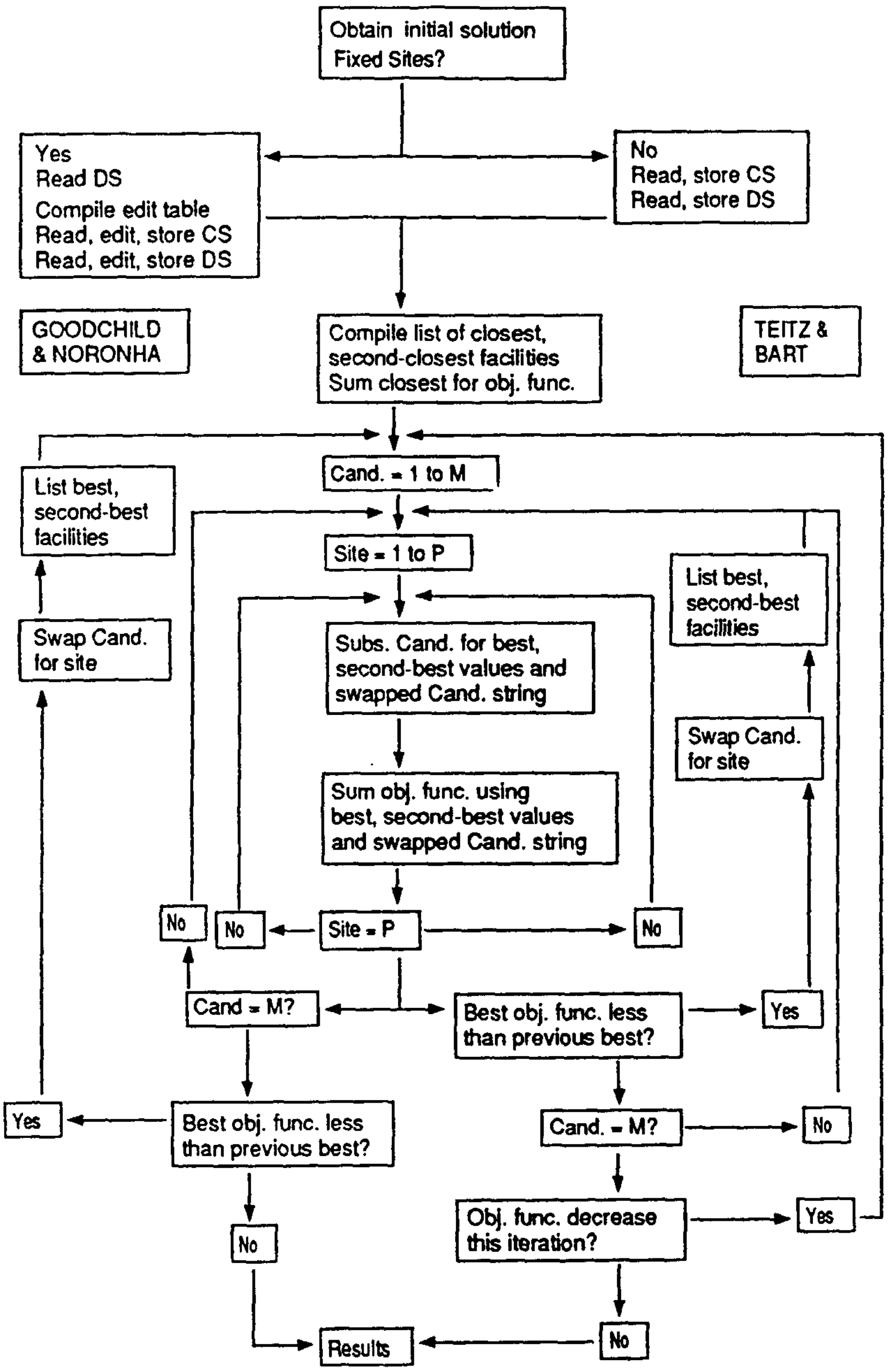

M1
M2


Situation 1
1
6
8 (2nd Closest)
3
9 (Candidate)
Situation 2a
2
8
8 (Closest)
12
1 (2nd Closest)
4
9 (Candidate)
Situation 2b
3
7
8 (Closest)
10
1 (2nd Closest)
7
9 (Candidate)
Situation 2c
4
4
8 (Closest)
6
1 (2nd Closest)
5
9 (Candidate)
Situation 2d
5
4
8 (Closest)
6
1 (2nd Closest)
7
9 (Candidate)
Situation 2e
6
12
8 (Closest)
15
1 (2nd Closest)
Situation 3
7
4
10 (Closest)
5
1 (2nd Closest)
2
9 (Candidate)


Situation 1
1
6
8 (2nd Closest)
3
9 (Candidate)
Situation 2a
2
8
8 (Closest)
12
1 (2nd Closest)
4
9 (Candidate)
Situation 2b
3
7
8 (Closest)
10
1 (2nd Closest)
7
9 (Candidate)
Situation 2c
4
4
8 (Closest)
6
1 (2nd Closest)
5
9 (Candidate)
Situation 2d
5
4
8 (Closest)
6
1 (2nd Closest)
7
9 (Candidate)
Situation 2e
6
12
8 (Closest)
15
1 (2nd Closest)
Situation 3
7
4
10 (Closest)
5
1 (2nd Closest)
2
9 (Candidate)
Situation 4a
11
6
10 (Closest)
8
8 (2nd Closest)
9
9 (Candidate)
Situation 4b
12
8
10 (Closest)
10
8 (2nd Closest)



Situation 1
1
3
9 (Closest)
Situation 2a
2
12
1 (2nd Closest)
4
9 (Closest)
Situation 2b
3
10
1 (2nd Closest)
7
9 (Closest)
Situation 2c
4
6
1 (2nd Closest)
5
9 (Closest)
Situation 2d
5
6
1 (Closest)
7
9 (2nd Closest)
Situation 2e
6
15
1 (Closest)
Situation 3
10 (2nd Closest)
4
7
2
9 (Closest)
Situation 4a
10 (Closest)
8
11
? (2nd Closest?)
9
9 (2nd CLosest?)
Situation 4b
10 (Closest)
8
12
? (2nd Closest?)



Obtain initial solution
Fixed Sites?
Yes
Read DS
Compile edit table
Read, edit, store CS
Read, edit, store DS
No
Read, store CS
Read, store DS
GOODCHILD & NORONHA
Compile list of closest, second-closest facilities
Sum closest for obj. func.
TEITZ & BART
Cand. = 1 to M
List best, second-best facilities
Site = 1 to P
List best, second-best facilities
Swap Cand. for site
Subs. Cand. for best, second-best values and swapped Cand. string
Swap Cand. for site
Sum obj. func. using best, second-best values and swapped Cand. string
No
No
Site = P
No
Cand = M?
Best obj. func. less than previous best?
Yes
Yes
Best obj. func. less than previous best?
Cand. = M?
No
No
Obj. func. decrease this iteration?
Yes
Results
No