ELSEVIER

European Journal of Operational Research 101 (1997) 499–508

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

Theory and Methodology

A dynamic programming heuristic for the P-median problem

Michelle Hribar [a], Mark S. Daskin [b,*]

[a] Department of Electrical and Computer Engineering, Northwestern University, Evanston, IL 60208, USA
[b] Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL 60208, USA

Received 1 January 1996; accepted 1 May 1996

Abstract

A new heuristic algorithm is proposed for the P-median problem. The heuristic restricts the size of the state space of a dynamic programming algorithm. The approach may be viewed as an extension of the myopic or greedy adding algorithm for the P-median model. The approach allows planners to identify a large number of solutions all of which perform well with respect to the P-median objective of minimizing the demand weighted average distance between customer locations and the nearest of the P selected facilities. In addition, the results indicate regions in which it is desirable to locate facilities. Computational results from three test problems are discussed. © 1997 Elsevier Science B.V.

Keywords: Location; Heuristics; Dynamic programming

1. Introduction

Facility location decisions are inherently strategic in nature. They relate to the long term placement of warehouses, production plants, emergency service bases, schools, and hospitals. Because facility location decisions almost always impact multiple groups in different ways, no single solution is likely to be the best for everyone. Nevertheless, facility location models tend to focus on a single objective and most algorithms are designed to find a single solution to a narrowly defined objective.

In this paper, we outline a new solution approach to the P-median model. The P-median problem, originally proposed by Hakimi (1964, 1965) is that of locating P facilities to minimize the sum of the demand-weighted distance between each demand node and the nearest of the P facilities. Hakimi showed that at least one optimal solution to the problem consists of locating the P facilities on the nodes of the network. Nevertheless, the problem is NP-complete for variable P (Garey and Johnson, 1979). A large number of solution algorithms has been proposed for this model, including neighborhood search techniques (Maranzana, 1964), exchange algorithms (Teitz and Bart, 1968) and Lagrangian relaxation coupled with branch and bound (Christofides and Beasley, 1982). Densham and Rushton (1992) outline data structures that are useful in solving large-scale P-median problems. Mirchandani (1990) summarizes solution approaches and extensions of the P-median model.

We propose a dynamic programming based heuristic to solve the P-median problem. This ap-

* Corresponding author.

proach differs from those outlined above in that it identifies a large number of very good solutions; all of the algorithms mentioned above only find a single solution. The approach is loosely similar in concept to a genetic algorithm in which multiple solutions are maintained throughout the execution of the algorithm (Denning, 1992; Goldberg, 1989; Holland, 1992). In addition, the algorithm we propose allows the user to determine the frequency with which candidate sites appear in the group of good solutions. As such, the algorithm identifies facility locations that should be given very serious consideration and those that can probably be excluded from any location plan. This information is similar to that used in some tabu search algorithms to diversify the search process (Glover, 1990). While the algorithm does not guarantee optimal solutions, it tends to do very well when compared to known optimal (or near optimal) solutions. The algorithm has the further advantages of having polynomial complexity and of being a derivative of the well-known myopic algorithm for the P-median problem.

The algorithm can readily be applied to a variety of location models. In fact, the algorithm is similar to the approach adopted by Malandraki and Dial (1991) for the time-dependent traveling salesman problem and by Pinter et al. (1989) for VLSI design of a computer chip. We selected the P-median model because of its versatility. Hillsman (1984) has shown that by modifying the cost coefficients, the P-median model can be used to represent a large number of other facility location problems including covering problems and certain types of fixed charge facility location problems. It can also be used as the basis for multi-objective analyses as suggested by Daskin (1995).

The remainder of this paper is organized as follows. Section 2 outlines the algorithm we propose and indicates its relationship to the myopic algorithm for the problem. Section 3 discusses data structures needed to implement the algorithm we propose. Section 4 presents a small example that illustrates the somewhat counter-intuitive result that doing more work may actually degrade the quality of the solutions found by the algorithm. Computational results for 49, 55 and 88-node problems are summarized in Section 5. Conclusions and recommendations for future work are outlined in Section 6.

2. A new approach to solving P-median problems

The algorithm we propose is a cross between the greedy or myopic algorithm for the P-median problem and a dynamic programming approach. We begin by briefly outlining each of these two algorithms before presenting the hybrid algorithm.

Greedy algorithms find the best that can be done given the current state of the solution. "Best" is defined by some measure appropriate to the problem at hand. For example, for the minimum spanning tree problem, Kruskal's algorithm (Kruskal, 1956) adds the shortest link whose addition does not form a cycle with any of the portions of the tree already included in the solution. In this case, the "best" link is the shortest link that does not create a cycle. This algorithm is known to be optimal for this problem. If a problem can be described as a matroid, then the greedy algorithm will optimally solve the problem (Lawler, 1976). In general, greedy algorithms tend not to be optimal, though they are easy to understand and easy to implement. In addition, for some problems the criterion on which the algorithm should be greedy is not always apparent a priori. Changing the measure is generally quite easy in greedy algorithms. As such, the efficacy of new measures can readily be tested.

A greedy or myopic algorithm for the P-median problem adds as the qth site the location that reduces the total demand-weighted distance as much as possible *holding* the locations of the first $q-1$ selected sites fixed. Sites continue to be added until P sites have been included in the solution. In pseudo-code the algorithm may be written as shown in Fig. 1.

```
s ← $\emptyset$                                   {Set solution to empty set}
for k=1 to P do                                 {Loop over number of sites to add}
      best_node_to_add←0                        {Set best site to add to 0}
      best_obj←$\infty$                                  {Set best known objective function value to infinity}
      for j=1 to N do                           {Loop over candidate sites}
            if $g(s \cup \{j\})$ <best_obj then            {Compare objective function with candidate site j to best known objective funtion}
                  best_obj ← $g(s \cup \{j\})$             {Store improved objective function}
                  best_node_to_add←j            {Store site to add}
            endif
      end for
      s ← s $\cup$ {best_node_to_add}                {Update emerging solution}
end for
```

Fig. 1. Greedy heuristic for the P-median problem.

```
$S_0 \leftarrow \emptyset$                                   {Set solution to empty set}
for i=1 to P do                              {Loop over the number of facilities to add}
    for s in $S_{i-1}$ do                             {Loop over sets of solutions
                                                       with i-1 facilities}
        for j=1 to N do                              {Loop over candidate sites}
            if $\{s \cup \{j\}\} \notin S_i$ then $S_i \leftarrow S_i \cup \{s \cup \{j\}\}$
                                                 {Include in $S_i$ all possible
                                                  combinations of i facilities}
            endif
        end for
    end for
end for
Find $\hat{s} = \arg\min_{s \in S_P}\{g(s)\}$              {Find best solution among all
                                                  solutions with P facilities}
```

Fig. 2. Dynamic programming algorithm for the P-median problem.

In this algorithm, S is the set of candidate facility sites that have been selected and N is the number of nodes in the problem. The function $g(X)$ is the P-median objective function when facilities are located at the sites given by the set X. For $P = 1$, the greedy algorithm will clearly give the optimal solution. For larger values of P, the algorithm may give rather poor solutions (Daskin, 1995).

A number of algorithms have been proposed for finding optimal solutions to the P-median problem. We note that no polynomial time algorithms are known for the P-median problem since the associated decision problem is NP-complete (Garey and Johnson, 1979) as indicated above. Nevertheless, one approach to solving the problem is to use dynamic programming with the stage variable being the number of facilities added to the solution so far and the state variable being the set of selected sites. Letting S_i denote the set of all possible states with i selected facilities, Fig. 2 gives a dynamic programming algorithm for the P-median problem.

This algorithm essentially builds all possible combinations of i nodes out of the N possible candidate nodes for all values of i between 1 and P, the desired number of facilities. The algorithm then finds the best set of nodes from among all of the possible combinations of P nodes selected from the N candidate sites.

Two points are worth noting. First, the algorithm can be streamlined slightly from that represented by the pseudo-code above by having the outermost loop extend only to $P - 1$ facilities and then building the best solution for exactly P sites from the solutions with $P - 1$ facilities. The second point, however, is that even with this minor improvement, the algorithm above is not practical for any large (or even moderate) values of P and N. The algorithm suffers from the "curse of dimensionality" common to many dynamic programming algorithms. In short, the size of the state space explodes. For example, for an 88-node problem, there are over 39 million combinations of 5 facilities and over 4.5×10^{12} combinations of 10 facilities. If a computer could evaluate one million combinations every second, it would still require over one and a half months to evaluate all combinations of 10 facilities to find the best combination. In essence, the dynamic programming approach outlined above is nothing more than a total enumeration scheme.

Computing, saving and evaluating all of the possible combinations of $P - 1$ facility sites out of N candidate sites en route to finding the optimal P-site solution is clearly prohibitive. To find a very good, and probably optimal, solution, it is also likely to be unnecessary. In other words, it is unlikely that the best P sites will include $P - 1$ sites that perform poorly as a solution to the $P - 1$ median problem and then adding some Pth site. Rather, good P-site solutions are likely to come from good solutions to the $P - 1$ median problem. This suggests that storing all possible combinations of q facilities ($1 \le q \le P - 1$) is not necessary and that one need only store the good q-facility solutions. This leads to the dynamic programming-based heuristic for the P-median problem shown in Fig. 3. In this algorithm we use the following additional notation:

H	= the maximum number of solutions to store at any stage,
h	= the number of solutions already stored,
$g_i^m(s^m)$	= the mth best objective function found for the i-median problem (which is obtained by locating facilities at the nodes in the set s^m).

To simplify the exposition of the algorithm, the pseudo code in Fig. 3 does not explicitly note that in any emerging set of solutions, S_i, we only want to store one copy of any particular configuration of facility sites. In other words, if the set {1, 3, 7} is among the top H solutions for a 3-median problem, we may obtain this solution in any one of three ways: adding node 7 to {1, 3} in a 2-median solution; adding node 3 to {1, 7} in a 2-median solution; or

```
$S_0 \leftarrow \emptyset$                                {Set solution to empty set}
for i=1 to P do                      {Loop over the number of facilities to add}
    $h \leftarrow \emptyset$                       {Initialize number of solutions stored}
    for s in $S_{i-1}$ do                          {Loop over sets of solutions
                                                            with $i-1$ facilities}
        for j=1 to N do                              {Loop over candidate sites}
            if $h < H$ then{Have we stored less than $H$ solutions?}
                $h \leftarrow h + 1$                      {Increment number stored}
                $S_i \leftarrow S_i \cup \{s \cup \{j\}\}$ {Update list of solutions
                                                              for $i$ facilities}
                $g_i^h(s^h) \leftarrow g(s \cup \{j\})$              {Store solution}
                SORT the h solutions in $S_i$
                                              {Sort the $h$ solutions so that
                                           $g_i^1(s^1)$ is the best and so on}
            else                   {Here if we have $H$ or more solutions}
                if $g(s \cup \{j\}) < g_i^H(s^H)$ then
                    $S_i \leftarrow S_i \setminus s^H$
                    $S_i \leftarrow S_i \cup \{s \cup \{j\}\}$
                          {Update list of solutions for $i$ facilities}
                    $g_i^H(s^H) \leftarrow g(s \cup \{j\})$ {Store solution}
                    SORT the H solutions in $S_i$
                                              {Sort the $H$ solutions so that
                                           $g_i^1(s^1)$ is the best and so on}
                endif
            endif
        end for
    end for
end for
```

Fig. 3. Dynamic programming based heuristic for the P-median problem.

adding node 1 to {3, 7} in a 2-median solution. We would only want to store solution {1, 3, 7} once. Thus, when it is generated the second and third times, we would need to be able to check whether it is already in the set S_3.

It is worth noting that this heuristic is equivalent to the greedy heuristic of Fig. 1 if $H = 1$ and to the dynamic programming algorithm if H is sufficiently large, that is, if

$$H > \max_{0 \le j \le P} \binom{N}{j}.$$

3. Data structures

To implement the dynamic programming-based heuristic shown in Fig. 3, efficient data structures must be used to allow new solutions to be added to and deleted from the sets S_i, to allow the algorithm to check if a solution has already been added to the set S_i, and to easily find which solution is the best and which is the Hth best.

The data structure must hold the set S_i of H best solutions at stage i. Thus, each element of the structure must contain one solution – its objective function value and the identity of the facility locations. Storing the identity of the facility locations is accomplished using an array of $B = N/m$ m-bit numbers. (In the implementation described below we used 32-bit numbers.) Each bit represents a node. If the bit is set to 1, it is included in the solution. Determining if two solutions are equal is accomplished by evaluating the exclusive or of the two corresponding arrays.

If we store no information about the assignment of each node to the closest open facility, the evaluation of the objective function value of each solution is $O(N^2)$. That is, for each node, we have to find the closest open facility using the distance matrix which takes $O(N)$ time for each node. We can reduce this time by adding an array of length N to each element of the data structure. This array will store the distance from each demand node to the closest open facility for that particular solution. Thus, when we form a new solution by adding an open facility to this particular solution, we need only compare the distances in the array with the distances from each node to the new open facility (in $O(1)$ time). As we do these comparisons, we can update the array and sum the demand weighted distance to evaluate the objective function value in $O(N)$ total time. In the results discussed below, we have opted to store these distances, thereby increasing the memory requirements associated with each solution, but decreasing the time required to evaluate solutions.

There are several possible data structures that can be used to store these elements efficiently. In this paper, we use balanced binary trees. Balanced binary trees are binary tree data structures in which the height is approximately balanced; that is, the height of the tree is log H. The worst case search, insert and delete time is a function of the height of the tree, so it is important for this to be a minimum. Guibas and Sedgewick (1978) developed conventions for balancing trees. In our work we used red-black balanced binary trees. The reader is referred to Cormen et al. (1990) for a discussion of implementation details for this sort of tree.

Using red-black trees in the heuristic results in a worst case running time of $O(PHN(N + \log H))$. That is, for all P stages, for all of the H solutions of the previous stage, each of the N demand nodes is added to the previous solution to form a new solution. The objective function is evaluated for this new solution ($O(N)$ time). In the worst case, each new solution falls within the H best and was not already found (search is $O(\log H)$ time). In such a case, it is inserted into the set of H solutions at the current stage ($O(\log H)$ time) and the worst solution is deleted ($O(\log H)$ time).

4. More is not always better

As indicated above, the dynamic programming heuristic reduces to the greedy heuristic when $H = 1$ and to the optimal dynamic programming algorithm if H is sufficiently large. This suggests that the results should improve with increasing values of H. Unfortunately this is not always so as the example shown in Fig. 4 below indicates.

Table 1 gives the solution for $P = 1, 2, 3$. The table gives the optimal solution as well as the solutions obtained by using the dynamic programming heuristic algorithm for $H = 1$ and $H = 2$. As expected, the *DP*-heuristic finds the optimal 1-median solution. For the 2-median problem, the *DP*-heuristic fails to find the optimal solution when $H = 1$ since the optimal 2-median solution does not include locating at the optimal 1-median site and the heuristic has no way of removing inferior nodes. When $H = 2$, however, the DP heuristic finds the optimal 2-median solution by adding a facility to the second best solution for the 1-median solution. The second best 2-median solution is also found by adding to the second best 1-median solution. For the 3-median problem, the *DP*-heuristic finds the optimal solution when $H = 1$, but cannot construct the optimal solution from either of the two 2-median solutions found when $H = 2$.

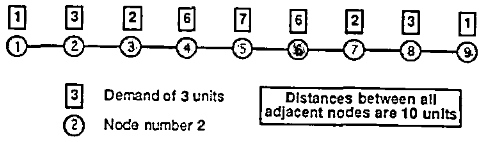

Fig. 4. Example network in which more is not always better.

This example illustrates the somewhat counter-intuitive result that increasing the number of solutions stored at each stage (i.e., increasing H) will not always result in finding better solutions. We encountered this in some of the computational results outlined below.

5. Computational results

5.1. Overview

This heuristic was coded in C and executed on a Sparc20 Workstation. The solutions from this heuristic are compared to "optimal" solutions obtained by the Lagrangian relaxation implemented in the SITATION software (Daskin, 1995).

This heuristic was tested using three data sets: $N = 49$, 55 and 88. The 49-node data set is given in Appendix H of (Daskin, 1985). This data set repre-

Table 1
Results for the network of Fig. 4

P	Optimal solution		*DP*-heuristic; $H = 1$		*DP*-heuristic; $H = 2$	
	Locations	Objective function	Locations	Objective function	Locations	Objective function
1	5	460	5	460	5	460
					4	530
2	3, 6	290	2, 5	320	4, 6	290
					4, 7	290
3	2, 5, 8	180	2, 5, 8	180	4, 6, 2	210
					4, 6, 8	210

Table 2
Minimum H values for the 49-node data set

P	CAPITAL.GRT $N = 49$	
	Minimum H	Minimum execution time (sec)
5	25	0.083
10	25	0.183
20	20	0.317
30	1	0.017
40	1	0.033

Table 4
Minimum H values for the 88-node data set

P	CITY1990.GRT $N = 88$	
	Minimum H	Minimum exec. time (sec.)
5	200	1.850
10	200	4.433
20	400	19.300
30	3,750	312.304
40	400	41.032

sents the 48 capitals of the 48 contiguous states in the United States along with Washington, D.C. Distances are great circle distances and the demands are the 1990 populations of the states (or Washington, D.C.). The 88-node data set is a minor variant of the data set given in Appendix G of (Daskin, 1985). The 88 nodes represent the 50 most populous cities in the United States along with the capitals of the 48 contiguous states (with duplicate cities removed from the data set). Demands are the 1990 city populations in this case. Distances were again computed as the great circle distances between nodes. The 55-node problem is that given by Swain (1971) and has been used by a number of authors in testing location problems (e.g., Daskin, 1982, 1983; Eaton et al., 1977; and Masog, 1980). This data set represents 55 communities in the Washington, D.C. area. Demands were generated in a pseudo-random manner with most large demands at the center of the region. Euclidean distances were used. For each data set we solved problems for $P = 5$, 10, 20, 30 and 40. H was varied from 1 to 1,000 for all cases except for the 30-median problem on the 88-node data set for which H ranged up to 4,500.

Table 3
Minimum H values for the 55-node data set

P	NODE55 $N = 55$	
	Minimum H	Minimum execution time (sec)
5	30	0.117
10	60	0.567
20	25	0.500
30	200	6.150
40	45	1.900

Tables 2–4 report the smallest value of H after which the dynamic programming heuristic consistently gave the same solution as the Lagrangian relaxation algorithm. The minimum execution time is the execution time needed for the minimum H. As these tables show, as N and P get larger, the H required must be larger as well. For example, for the 49-node problem, the minimum H needed is 25 and the execution times are all under a second. For the 55-node problem, the minimum H required is 200 and the execution times are less than 7 seconds. For the 88-node problem, the minimum H required is 3,750 and the execution time is approximately 5 minutes.

While the results are reasonable for the 49- and 55-node problems, the results for the 88-node data set, particularly when $P = 30$, are discouraging. Having to increase H so that the execution time is around 5 minutes is not desirable since SITATION gives the optimal solution in under 10 seconds on a 486-grade personal computer. (It is worth noting that the dynamic programming heuristic often provided the optimal solution at lower values of H, but the values reported are the smallest values such that the optimal solution was found in all tests with larger values of H.)

5.2. Convergence behavior

We expect the best objective function value found by the *DP*-heuristic to improve as H increases. Fig. 5 shows this behavior for the 55-node data set when $P = 10$ as H varies from 1 to 1,000. (Note that the abscissa is plotted as a logarithmic scale.) The generally expected behavior is exhibited in this case.

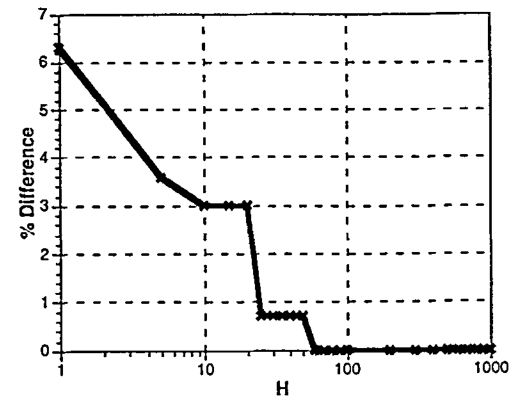

Fig. 5. Percent difference between *DP*-heuristic and optimal solution for 55-node problem and $P = 10$.

However, as indicated in Table 1 of Section 4 above, solutions may degrade as H increases. As shown in Fig. 6, the difference between the best solution found by the *DP*-heuristic and the optimal solution decreases and increases several times before the solution converges to the true optimal. Clearly increasing the number of solutions stored at each stage may not improve the results obtained by the *DP*-heuristic. Also note that the optimal solution is found for all values of $H > 3{,}750$ and this is the smallest tested value of H for which this is true. This is therefore the value reported for $P = 30$ in Table 4.

Fig. 7 shows the percentage difference between the value found by the *DP*-heuristic and the optimal solution for each value of P, from 1 to 87 for the

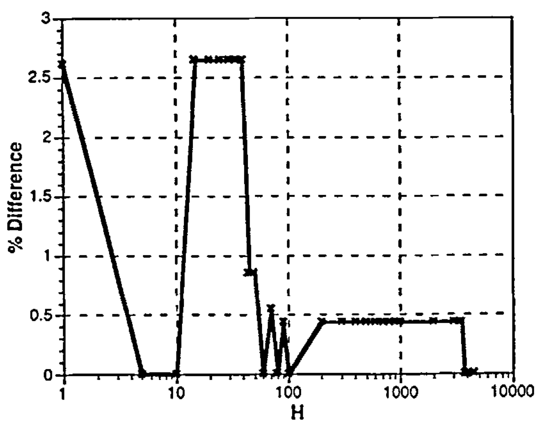

Fig. 6. Percent difference between *DP*-heuristic and optimal solution for 88-node problem and $P = 30$.

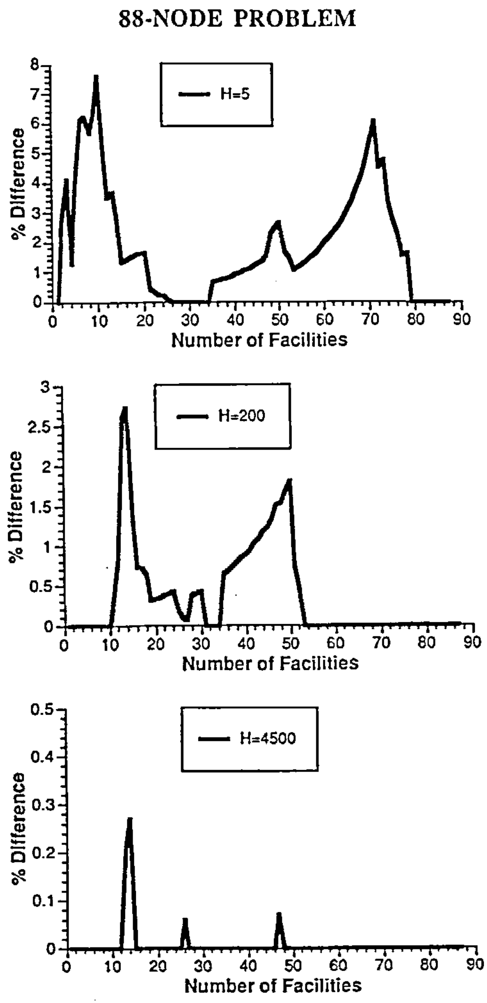

Fig. 7. Percent difference between *DP*-heuristic and optimal solution for 88-node problem.

88-node problem, using $H = 5$, 200 and 4,500. Increasing H generally results in more accurate solutions, but this is not always the case. For example, at $P = 30$, using $H = 5$, gives the optimal solution, but does not when H is increased to 200. Again, these seemingly anomalous results are explained by the small example discussed in Section 4.

5.3. Execution time

Fig. 8 shows the execution times for the 88-node data set and the different values of P and H. Similar results were obtained for the 49 and 55-node data

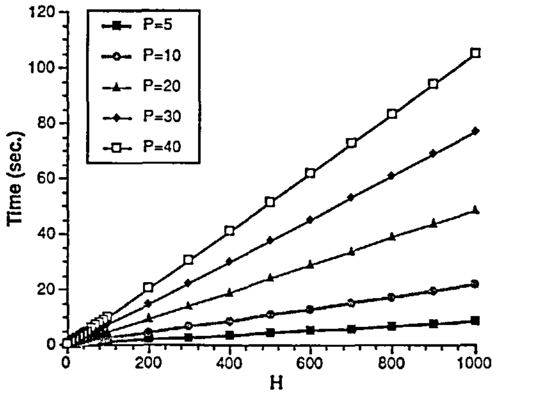

Fig. 8. Execution time for the 88-node problem as a function of H.

sets. Holding N and P constant, the worst case running time of the algorithm is $O(H \log H)$. However, our experimental results showed a linear increase in the execution time as H increased. This is possible if we do not experience the full $\log(H)$ worst case time for the insert, search and delete. Furthermore, not every one of the NH possible solutions generated at each stage will fall within the H current best and will need to be inserted into the current tree thereby incurring the extra $O(\log H)$ time. In short, these experimental results suggest that the algorithm may not experience the worst case behavior in practice. Additional tests on other data sets are needed to confirm this tentative finding.

Execution times as a function of the other problem parameters were as predicted by the $O(PHN(N + \log H))$ worst case performance of the algorithm.

5.4. Advantages of the DP-heuristic

While the running time of this algorithm can be considerably longer than those of other heuristics, the algorithm has several benefits. First, the algorithm does not only give a single solution; it gives H good solutions. This is advantageous when decision makers want to evaluate several solutions, a likely occurence in strategic decision making.

Second, the algorithm does not only find the best ways to locate P facilities, it also finds the best ways to locate $P - 1$ down through 1 facility. Again, this is beneficial when the number of facilities to locate is subject to debate.

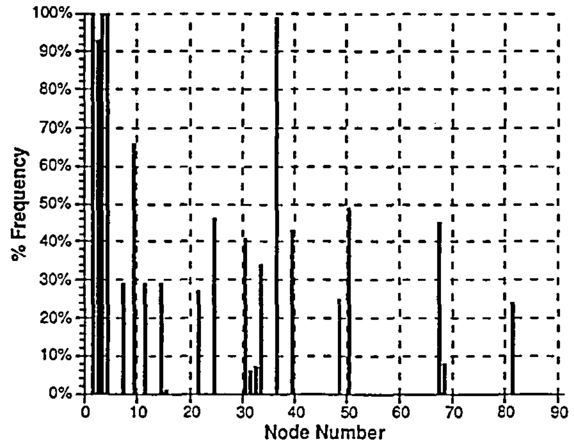

Fig. 9. Percent of solutions found that use each candidate site for the 88-node data set with $P = 10$ and $H = 1,000$.

Finally, the algorithm can determine the frequency with which the nodes appear in the H solutions. Fig. 9 gives the frequency histogram of nodes in the 119 solutions found to be within 1% of the optimal solution for the 88-node problem with $P = 10$ and $H = 1,000$. Nodes 1, 3, and 4 appear in all of the solutions that are within 1% of the best solution found. This suggests strongly that it may be advantageous to locate at these nodes. The optimal solution is to locate at nodes 1, 2, 3, 4, 9, 24, 30 36, 39, and 50 and does, in fact, include these three high-frequency sites.

Of the 88 nodes, only 22 – exactly 25% – are selected at all in the 119 solutions represented in Fig. 9. Furthermore, as shown in Fig. 10, these 22 sites can be loosely grouped into 10 areas in which facilities should be located. These 10 areas are listed in Table 5. One of the cities in each of regions 1

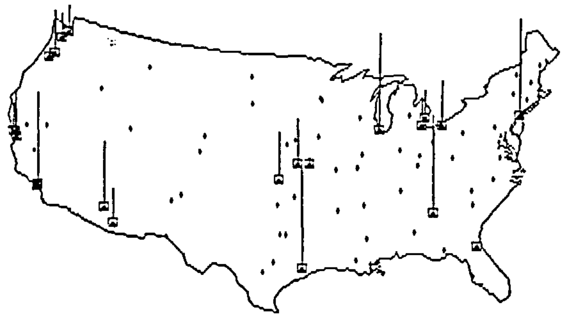

Fig. 10. Locations of selected cities.

Table 5
Approximate grouping of areas selected in 119 solutions shown in Fig. 9 and Fig. 10

Region	Cities in region (city no. | no. times)
1	New York, NY (1 | 119)
2	Chicago, IL (3 | 119)
3	Houston, TX (4 | 119)
4	Los Angeles, CA (2 | 111); Long Beach, CA (32 | 8)
5	Phoenix, AZ (9 | 79); Tucson, AZ (33 | 40)
6	Wichita, KS (50 | 58); Topeka, KS (67 | 54); Kansas City, MO (31 | 7)
7	Portland, OR (30 | 49); Seattle, WA (21 | 32); Olympia, WA (81 | 29) Salem, OR (68 | 9)
8	Oakland, CA (39 | 51); San Jose, CA (11 | 34); San Francisco, CA (14 | 34)
9	Cleveland, OH (24 | 55); Detroit, MI (7 | 34); Toledo, OH (48 | 30)
10	Atlanta, GA (36 | 118)

through 9 is selected in each of the 119 solutions. Region 10 (Atlanta, GA) is selected in 118 of the 119 solutions. (Jacksonville, FL is selected in one solution and is not shown in the table). This analysis suggests that the good (though certainly not optimal) solutions may be obtained by selecting one city from each of these 10 regions. Furthermore, the analysis shows that good siting plans for 10 facilities will almost certainly include New York, Chicago, Houston, Los Angeles, and Atlanta (nodes 1, 3, 4, 2, and 36, respectively).

6. Conclusions

This paper has outlined a new heuristic that is a cross between the polynomial time greedy algorithm for the P-median problem and a non-polynomial time dynamic programming algorithm. The approach is a dynamic programming algorithm in which only the best H solutions are stored at each stage of the algorithm. The resulting algorithm has polynomial time complexity, but does not guarantee optimal solutions. While the algorithm does not provide bounds on the solution and the execution times tend to be large when compared to other heuristic algorithms, the approach gives very good solutions. It also allows the user to identify the frequency with which facility sites are selected in good solutions. It thereby allows decision makers to gain insight into the nature of good solutions as it tends to identify regions in which facilities should be placed. Finally, the algorithm provides this information for siting 1 through P facilities, not just for locating P facilities.

This work can be extended in a number of ways. First, additional computational experiments can be conducted to determine whether or not the tentative finding that the actual execution time increases only linearly with H holds for other data sets. Second, tests can be conducted to determine whether or not guidelines for the selection of H as a function of N and P can be devised. Third, the usefulness of results like those shown in Fig. 9 to actual decision makers can be assessed through the application of this algorithm in actual decision contexts. Finally, the basic approach outlined above – that of a dynamic programming algorithm in which the state space is restricted to include only the H best solutions at any stage – may be applied to other more complicated facility location problems.

Acknowledgements

The authors gratefully acknowledge the comments of the referees. These comments helped strengthen the paper. The work of the second author was supported, in part, by grant number DMI-9224998 from the National Science Foundation. This support is gratefully acknowledged as well.

References

Christofides, N. and Beasley, J.E., 1982. A Tree Search Algorithm for the P-Median Problem. *European Journal of Operational Research*, 10, 196–204.

Cormen, T.H., Lierson, C.E. and Rivest, R.L., 1990. *Introduction to Algorithms*, M.I.T. Press, Cambridge, MA.

Daskin, M.S., 1982. Application of an Expected Covering Model to EMS System Design. *Decision Sciences*, 13, 416–439.

Daskin, M.S., 1983. A Maximum Expected Covering Location Model: Formulation, Properties and Heuristic Solution. *Transportation Science*, 17, 48–70.

Daskin, M.S., 1995. *Network and Discrete Location: Models, Algorithms and Applications*, John Wiley and Sons, Inc., New York.

Denning, P.J., 1992. Genetic Algorithms. *American Scientist*, 80, 12–14.

Densham, P.J. and Rushton, G., 1992. A More Efficient Heuristic for Solving Large *P*-Median Problems. *Papers in Regional Science: The Journal of the RSAI*, 71, 307–329.

Eaton, D., Church, R. and ReVelle, C., 1977. Location Analysis: A New Tool for Health Planners. Methodological Working Document 53, Sector Analysis Division, Bureau for Latin America, Agency for International Development.

Garey, M.R. and Johnson, D.S., 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., New York.

Glover, F., 1990. Tabu Search: A Tutorial. *Interfaces*, 20, 74–94.

Goldberg, D.E., 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, Reading, MA.

Guibas, L.J. and Sedgewick, R., 1978. A Dichromatic Framework for Balanced Trees. *Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science*, 8–21.

Hakimi, S.L., 1964. Optimum Locations of Switching Centers and the Absolute Centers and Medians of a Graph. *Operations Research*, 12, 450–459.

Hakimi, S.L., 1965. Optimum Distribution of Switching Centers in a Communication Network and Some Related Graph Theoretic Problems. *Operations Research*, 13, 462–475.

Hillsman, E.L., 1984. The *P*-median structure as a unified linear model for location-allocation analysis. *Environment and Planning A*, 15, 305–318.

Holland, J.H., 1992. Genetic Algorithms. *Scientific American*, July, 66–72.

Kruskal, J.B., 1956. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *Proceedings of the American Mathematical Society*, 7, 48–50.

Lawler, E., 1976. *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York.

Malandraki, C. and Dial, R.B., 1991. A Dynamic Programming Heuristic Algorithm for the Time-Dependent Traveling Salesman Problem. Technical Report, Roadnet Technologies, Inc., Timonium, MD.

Maranzana, F.E., 1964. On the Location of Supply Points to Minimize Transport Costs. *Operational Research Quarterly*, 15, 261–270.

Masog, T.G., 1980. Modelling Techniques for a Flexible Emergency Aid Location System, unpublished M.S. thesis, Department of Civil Engineering and the LBJ School of Public Affairs, University of Texas at Austin, Austin, TX.

Mirchandani, P.B., 1990. The *P*-Median Problem and Generalizations. In: P.B. Mirchandani and R.L. Francis (eds), *Discrete Location Theory*, John Wiley Inc., NY., 55–117.

Pinter, R.Y., Bar-Yehuda, R., Feldman, J.A. and Wimer, S., 1989. Depth-first-search and Dynamic Programming Algorithms for Efficient CMOS Cell Generation. *IEEE Transactions on Computer Aided Design*, 8, 737–743.

Swain, R., 1971. A Decomposition Algorithm For a Class of Facility Location Problems. Unpublished Ph.D. dissertation, Cornell University, Ithaca, NY.

Teitz, M.B. and Bart, P., 1968. Heuristic Methods for Estimating Generalized Vertex Median of a Weighted Graph. *Operations Research*, 16, 955–961.