Annals of Operations Research 40(1992)283–302

AVOIDING LOCAL OPTIMA IN THE p-HUB LOCATION PROBLEM USING TABU SEARCH AND GRASP

John G. KLINCEWICZ
AT & T Bell Laboratories, Holmdel, NJ 07733-3030, USA

Abstract

In the discrete p-hub location problem, various nodes interact with each other by sending and receiving given levels of traffic (such as telecommunications traffic, data transmissions, airline passengers, packages, etc.). It is necessary to choose p of the given nodes to act as hubs, which are fully interconnected; it is also necessary to connect each other node to one of these hubs so that traffic can be sent between any pair of nodes by using the hubs as switching points. The objective is to minimize the sum of the costs for sending traffic along the links connecting the various nodes. Like many combinatorial problems, the p-hub location problem has many local optima. Heuristics, such as exchange methods, can terminate once such a local optimum is encountered. In this paper, we describe new heuristics for the p-hub location problem, based on tabu search and on a greedy randomized adaptive search procedure (GRASP). These recently developed approaches to combinatorial optimization are capable of examining several local optima, so that, overall, superior solutions are found. Computational experience is reported in which both tabu search and GRASP found "optimal" hub locations (subject to the assumption that nodes must be assigned to the nearest hub) in over 90% of test problems. For problems for which such optima are not known, tabu search and GRASP generated new best-known solutions.

1. Introduction

Situations exist in which various locations (nodes) interact with each other by sending and receiving traffic. This could represent telecommunications traffic, data transmissions, airline passengers, express packages, etc. One possible strategy to link these locations or nodes, via a communications or transportation network, requires a certain number p of the locations to be hubs. These hubs are fully interconnected with network links. The remaining locations are each in turn connected ("assigned") to one of the hubs. The hub is said to "serve" those nonhub locations assigned to it. Traffic can be sent between any pair of locations by using the hubs as intermediate switching points.

In the discrete p-hub location problem, the node locations are given, along with the level of traffic that each node sends to, and receives from, every other node. (If the actual traffic patterns are stochastic, then these levels represent the

mean traffic.) Also given are costs per unit for sending traffic between each possible pair of nodes. It is necessary then to choose p of the given nodes to act as hubs and to assign each other node to one of these hubs. The objective is to minimize the sum of the costs incurred along the various network links.

Like many facility location problems, the discrete p-hub location problem is a difficult combinatorial problem. (For a survey of facility location problems, see [4, 22].) Typically, many local optima exist. Because of the computational complexity (the problem is NP-complete), it is natural to apply heuristic techniques, especially for large problems. However, many standard heuristics, such as exchange algorithms [21], terminate once a local optimum is encountered.

In this paper, we describe how some recently developed approaches to combinatorial optimization, specifically, tabu search and greedy randomized adaptive search procedure (GRASP) methods, can be applied to design heuristics for the p-hub location problem. These heuristics examine several local optima in a coherent, systematic way and, thereby, obtain an especially good final solution. In our computational experiments (in which, for comparison purposes, we restricted nodes to be assigned to the closest hub), both tabu search and GRASP methods found the "optimal" hub locations (subject to that restriction on the assignments) in over 90% of test problems. Further, for problems for which such optima are not known, these new heuristics generated solutions that were superior to the previously best-known solutions. These results offer evidence that these heuristics, as well as refinements, variations and extensions of them, should be considered further for this and related problems.

The fundamental principles underlying tabu search are described by Glover [11, 12]. Based on strategies originally developed for nonlinear covering problems [8], this approach has since been applied to a variety of difficult combinatorial problems, e.g. [9, 10, 14, 15, 18, 28]. A similar approach was independently developed by Hansen [16]. Tabu search operates in the context of a search procedure in which one "moves" iteratively from one feasible solution to another. The key idea is to forbid certain possible "moves" at each iteration, based on the past history of moves, in order to force the algorithm to explore new areas of the feasible region.

So-called greedy random adaptive search procedures have also been applied to a variety of difficult combinatorial problems [1, 2, 5, 6, 17]. They were given the acronym GRASP by Feo et al. [7]. In GRASP methods, a search procedure to find a local optimum is replicated multiple times with different starting points. These starting points are determined by "greedy" procedure that has a probabilistic component within it. This probabilistic component consists of randomly choosing one of the best candidates from a list, and not necessarily the top candidate, in the greedy procedure. The goal is to choose many excellent starting points via the greedy procedure (as opposed to purely random starting points) and thereby increase the chances of finding the true optimum on at least one replication.

Tabu search and GRASP are among several new approaches to heuristic search procedures for combinatorial optimization that have been developed in recent

years. A survey of some of these approaches, including simulated annealing, neural networks and genetic algorithms, is given in [13].

The generic discrete p-hub location model for choosing hubs and their assignments was first formulated as a quadratic integer program by O'Kelly]26] who proposed an enumeration-based heuristic. A variety of single-exchange and double-exchange heuristics for the discrete p-hub location problem were proposed by Klincewicz [21]. The related problem of hub locations in a continuous plane is considered in [24] and issues of congestion in hub networks are discussed in [25]. Examples of hubbing networks are found in certain airlines, package delivery systems, and trucking companies, as discussed in [26, 27]. They are also found in certain telecommunications systems, such as backbone packet networks [3, 23].

In section 2, we formulate the discrete p-hub location problem. We describe our adaptation of tabu search to the p-hub location problem in section 3 and our application of GRASP to this problem in section 4. Section 5 reports our computational experience.

2. Problem formulation

In the p-hub location problem, there are n distinct locations or nodes. For any pair of nodes i and j, W_{ij} denotes the number of units of traffic sent from i to j. (We assume $W_{ii} = 0$.) If a node i is connected to a hub at node j, then a standard cost C_{ij} per unit is incurred for all traffic on the link between i and j. Typically, C_{ij} is proportional to the distance between i and j. If i and j are both hubs, then this cost is discounted to account for economies of scale in the volume of traffic between hubs. In the standard p-hub formulation considered here, these economies of scale are represented by a discount parameter a, so that all traffic sent between i and j incurs a cost at the rate of aC_{ij} per unit. This "linear" model might serve as an approximation to more complicated concave functions on interhub flows. These might arise, for example, in an airline network, where interhub traffic would use larger, more efficient airplanes, or in a communications network, where interhub traffic would use more efficient, high capacity links.

The integer decision variables used in the formulation are defined as follows:

$$Y_j = \begin{cases} 1 & \text{node } j \text{ is a hub,} \\ 0 & \text{otherwise;} \end{cases}$$

$$X_{ij} = \begin{cases} 1 & \text{node } i \text{ is connected to a hub at } j, \\ 0 & \text{otherwise.} \end{cases}$$

Although costs on a link are linear in the flows, the overall problem is quadratic in the facility assignment variables X_{ij}. In its most general form, the problem formulation can then be written as a quadratic integer program as follows:

$$\text{Minimize} \sum_i \sum_j W_{ij} \left(\sum_k X_{ik} C_{ik} + \sum_m X_{jm} C_{jm} + a \sum_k \sum_m X_{ik} X_{im} C_{km} \right) \tag{1a}$$

$$\text{subject to} \quad \sum_j X_{ij} = 1, \qquad \text{for } i = 1, \ldots, n \tag{1b}$$

$$\sum_j Y_j = p, \tag{1c}$$

$$X_{ij} \leq Y_j, \qquad \text{for } i = 1, \ldots, n \text{ and } j = 1, \ldots, n, \tag{1d}$$

$$X_{ij}, Y_j \in \{0, 1\}, \qquad \text{for } i = 1, \ldots, n \text{ and } j = 1, \ldots, n. \tag{1e}$$

The objective function (1a) sums costs for each ordered pair (i, j), i.e. the cost from origin i to its hub node k, plus the cost to destination j from its hub at some node m, plus the cost from hub k to hub m. Constraints (1b) require that every node be connected to a hub and constraint (1c) requires that exactly p hubs be chosen. Constraints (1d) ensure that a node i is assigned to j only if j is, in fact, a hub.

This formulation differs just slightly from the original formulation of O'Kelly [26], which uses variables X_{jj} in the role of Y_j. O'Kelly also uses an alternate form of constraints (1d).

Often, cost elements will be symmetric, i.e. $C_{ij} = C_{ji}$, thus simplifying certain computations. As pointed out in [26], by defining $O_i = \sum_j W_{ij}$ to be the total amount of traffic originating at node i and $D_i = \sum_j W_{ji}$ to be the total traffic destined for node i, the objective (1a) can be rewritten:

$$\text{Minimize} \sum_i \sum_k X_{ik} C_{ik} (O_i + D_i) + \sum_j \sum_m X_{jm} \sum_i \sum_k X_{ik} (a W_{ij} C_{km}). \tag{2}$$

Here, the first summation term represents the cost of ingoing traffic on the link between each node i and its associated hub k (assuming cost symmetry). The second term is a compact way of representing the costs on interhub links.

Note that, even if all Y_j variables were fixed (i.e. hub locations were known) in problem (1), the remaining problem of assigning nodes to hubs is still an NP-complete quadratic assignment problem. (See [22, chapter 9] for a discussion of quadratic assignment.) However, within the context of heuristics, it is important to

be able to evaluate quickly a given choice of p-hubs, and, therefore, relatively simple procedures for assigning nodes to hubs need be considered. For example, O'Kelly [26] considers cost-based assignments where $X_{ik} = 1$ if $C_{ik} = \min_j\{C_{ij}|Y_j = 1\}$, and where ties can be broken arbitrarily. Since costs are typically proportional to distance, these can also be called distance-based assignments. (In addition to cost or distance-based assignments, Klincewicz [21] also considers some more general schemes that take traffic volumes into account.)

In this paper, we restrict our attention to the cost or distance-based assignments. There are two reasons for this. First, in most applications, it makes sense for a hub to serve a reasonably compact group of nodes, i.e. nodes that are "close" to the hub. For this very reason, more sophisticated assignment heuristics may be unnecessary. Second, a relatively simple assignment procedure such as this allows us to compute "optimal" hub locations (i.e. the best possible hub locations assuming distance-based assignments) for small problems by enumeration. In this way, we have a consistent and verifiable benchmark to compare the quality of the hub solutions generated by various heuristics, particularly our new tabu search and GRASP procedures.

3. Tabu search

In this section, we outline our particular implementation of tabu search for the p-hub location problem. For additional background on tabu search, see [11, 12]. Tabu search operates in the context of a search procedure in which at each iteration one moves from one feasible solution to another. For the case of the p-hub location problem, a feasible solution consists of a set K of hubs, assuming all nodes are assigned to the "nearest" hub. We will consider a "move" to be an exchange in which some node q replaces some node r as a hub in the current set K.

As is described below, tabu search, unlike other search or exchange procedures, is not stopped once it hits a local optimum. It can continue despite a lack of "improving" moves, and, because of a "tabu list", it can avoid falling back into the same local optimum from which it just emerged. In this way, many local optima may be encountered. Tabu search approaches may be particularly appropriate to try for this type of problem, given other successful implementations for other quadratic integer programming problems (e.g. [28]).

3.1. THE VALUE FUNCTION

To evaluate how "good" a given move is, we will use a local improvement measure that computes the savings that would result from the move, assuming all nodes previously assigned to r are now assigned to q. (Since node assignments can, in fact, change, this measure of savings is just an estimate.) This savings estimate is used, for example, for the heuristics in [21]. We denote the interhub traffic from a hub k to a hub l for the current set of assignments by:

$$I_{kl} = \sum_{\{i|X_{ik}=1\}} \sum_{\{j|X_{jl}=1\}} W_{ij} . \tag{3}$$

If a node q replaces node r as a hub, the savings estimate is:

$$R_{qr} = \sum_{\{i|X_{ir}=1\}} O_i(C_{ir} - C_{iq}) + \sum_{\{i|X_{ir}=1\}} D_i(C_{ri} - C_{qi})$$

$$+ a \sum_{\substack{k \in K \\ k \neq r}} I_{rk}(C_{rk} - C_{qk}) + a \sum_{\substack{k \in K \\ k \neq r}} I_{kr}(C_{kr} - C_{kq}). \tag{4}$$

This represents the sum of savings on outgoing and incoming traffic both from and to nodes now served by r, as well as savings on interhub traffic on links to other hubs $k \in K$. We will consider a move to be an "improving" move if the savings estimate R_{qr} is positive.

Note that this is just one of potentially many different savings estimates that could be used as a value function. In this paper, we obtain good results using this particular function (4), which provides a good estimate with moderate computational effort and is consistent with the estimate used in the exchange heuristics considered in [21]. Other estimates, ranging from the very simple to the computationally complex, might also be considered. (See the discussion in section 5.5.)

3.2. THE TABU LIST

At each iteration, there is a "tabu list" of moves that are forbidden to be made. This list is based on the past history of moves. In our implementation, after a move (q, r) is made, in which node q replaces node r, the move (q, r) and its inverse move (r, q) are placed on the tabu list. The tabu list is initially built up over M consecutive moves and updated circularly in later iterations. That is, in later iterations, the oldest elements on the tabu list are removed and replaced with those based on the current move.

The only way that it is possible to make a move (q, r), if (q, r) is on the tabu list, is for the move (q, r) to satisfy some "aspiration level" criterion. In our implementation, a move (q, r) on the tabu list satisfies the "aspiration level" criterion if the current objective value minus R_{qr} is less than the best objective value found thus far.

3.3. INITIAL SOLUTION AND SEARCH PROCEDURE

One initializes the tabu search by constructing a good feasible solution. In our implementation, we choose the p nodes with the largest values of $O_i + D_i$ to be the initial set of hubs.

Thereafter, at each iteration, we choose to make the move (q, r), from among those moves that are permissible, that maximizes R_{qr}. "Permissible" moves are those that either

(1) are not on the tabu list, or

(2) are on the tabu list, but satisfy the aspiration level criterion.

Note that it is not necessary for the chosen move to be an "improving" move, i.e. the maximum value for R_{qr} could be negative. After each move, the objective function value (1a) is computed for the current feasible solution. If it is the best objective function value encountered thus far, the solution is recorded.

Computing R_{qr} for all possible q and r represents a significant part of the computational effort in the tabu search. It is therefore important that we seek efficient ways to implement this procedure. One specific way that this can be done, for a moderate to large number of nodes q, is to first compute

$$\hat{R}_r = \sum_{\{i|X_{ir}=1\}} O_i C_{ir} + \sum_{\{i|X_{ir}=1\}} D_i C_{ri} + a \sum_{\substack{k \in K \\ k \neq r}} I_{rk} C_{rk} + a \sum_{\substack{k \in K \\ k \neq r}} I_{kr} C_{kr}. \tag{5}$$

Then, for each q,

$$S_{qr} = \sum_{\{i|X_{ir}=1\}} O_i C_{iq} + \sum_{\{i|X_{ir}=1\}} D_i C_{qi} + a \sum_{\substack{k \in K \\ k \neq r}} I_{rk} C_{qk} + a \sum_{\substack{k \in K \\ k \neq r}} I_{kr} C_{kq} \tag{6}$$

and

$$R_{qr} = \hat{R}_r - S_{qr}. \tag{7}$$

This requires several additional multiplications (and additions) to compute $\hat{R}_r$ initially; however, for each q, we avoid subtracting $C_{ir} - C_{iq}$ and $C_{ri} - C_{qi}$ for each i and $C_{rk} - C_{qk}$ and $C_{kr} - C_{kq}$ for each k. For a moderate to large number of q, this is advantageous; the exact break-even point is computer dependent.

3.4. RESTARTS

Periodically, e.g. every N iterations, one can restart the tabu search procedure by erasing the current tabu list and choosing a new initial solution. In our implementation, we always use the best solution encountered thus far as the new initial solution, unless it is unchanged since the last restart. In this case, one can either terminate the search, or else, construct a new initial solution based on what is called "long-term memory".

Long-term memory consists of a matrix T in which element T_{qr} equals the number of times move (q, r) has been made since the very start of the algorithm. The purpose of long-term memory is to have a measure of what areas of the feasible region have already been well-explored, so that we might construct a solution in a less explored area. To do this, we construct the new initial solution, not on the basis of $O_i + D_i$, but on the basis of $O_i + D_i - \mu \sum_j (T_{ij} + T_{ji})$, where μ is a penalty parameter. (The parameter μ should be on the order of $O_i + D_i$. In our computational experiments in section 5, this implied $\mu = 10^4$.) Thus, if node i has already been

involved in a large number of moves, it is less likely to show up in the new initial solution.

In our computational experiments described in section 5, we allow at most one restart based on long-term memory. (Clearly, other criteria are possible.) With these conditions, our implementation of tabu search will terminate when the best solution encountered does not improve between restarts and a restart based on long-term memory has already been tried once.

4. GRASP

In a Greedy Randomized Adaptive Search Procedure (GRASP) method, a given heuristic is replicated many times; the best result obtained is then kept as the solution. Each replication consists of two steps: first, a greedy procedure with a probabilistic component that generates an initial feasible solution, and, second, a search procedure to improve on that initial solution.

4.1. THE INITIAL SOLUTION

4.1.1. A myopic savings criterion

In a greedy procedure, one adds one element at a time to the solution set, this one element being chosen according to some myopic criterion. In GRASP, however, the one element is chosen from among a list of the best candidate elements, not necessarily the top candidate.

In the case of the discrete p-hub location problem, we add one element (node) at a time to the set K of hubs. Many possible myopic criteria might be proposed. The myopic criterion we will use in our implementation is the true improvement S_i in the objective function (1a) that would result if a node i were added to the set K. This involves summing the cost changes due to each node j that would be reassigned to i, if i were added as a hub. (For choosing the first hub, when K is initially empty, the savings or improvement is calculated by assuming that the costs for the initial null assignments are equal to some large number.) The specific procedure for computing the improvement S_i for each node i can be described as follows:

COMPUTING S_i

Step 1. Initialize $S_i = 0$. Let $j = 1$.

Step 2. Let $\tilde{C}_j$ denote the minimum cost to assign j to a node in the current set K, and let k be the node in K associated with that cost, i.e. $\tilde{C}_j = C_{jk} = \min_{q \in K}\{C_{jq}\}$. (If K is empty, let $\tilde{C}_j$ equal some large number.) If $\tilde{C}_j \leq C_{ji}$, then go to step 5. Otherwise, execute steps 3 and 4.

Step 3. Add the savings for reassigning j to i:

$$S_i \leftarrow S_i + (O_j + D_j)(\tilde{C}_j - C_{ji}).$$

Step 4. (Skip if K is currently empty.) To compute changes in the interhub costs, perform the following procedure for each node $l \neq j$:

- Let $\tilde{C}_l = C_{lm} = \min_{q \in K}\{C_{lq}\}$, i.e. l would be assigned to hub m in the current set K.
- If $\tilde{C}_l > C_{li}$, so that l would be reassigned to i, let $S_i \leftarrow S_i + (W_{jl} + W_{lj})\, a\, C_{km}$.
- If $\tilde{C}_l \leq C_{li}$, let $S_i \leftarrow S_i + (W_{jl} + W_{lj})\, a\, (C_{km} - C_{im})$.

Step 5. Increment $j \leftarrow j + 1$. If $j \leq n$, go to step 2. Otherwise, stop. □

4.1.2. Efficient implementation

Although the above steps provide a clear description of the cost savings included in S_i, efficient computer implementations might perform the computations differently. As one obvious example, one could store $\hat{W}_{jl} = W_{jl} + W_{lj}$ and $OD_j = O_j + D_j$ in advance so that they need not be recomputed each time. Computations could also be reduced by storing the interhub-cost for traffic to and from node j, given the current set of hubs K, i.e.

$$H_j = \sum_l \hat{W}_{jl} C_{km}, \tag{8}$$

where j is assigned to hub k and each l is assigned to hub m in the current solution. (We have suppressed the notational dependence of k on j and m on l.) Likewise, it is convenient to denote $\tilde{H}_j(i)$ to be the interhub cost for traffic to and from node j, if node i were added to K. In addition, one could define quantities T_m to be the traffic between i and hub m if i were added as a hub; T_m could then be accumulated in step 4, before any multiplication by C_{im} and a. Given these, the following efficient implementation results:

COMPUTING S_i (ALTERNATIVE PROCEDURE)

Step 1. Initialize $s_1 = 0$, $s_2 = 0$, $H_j(i) = 0$ for all j, and $T_m = 0$ for all $m \in K$. Let $j = 1$.

Step 2. If $\tilde{C}_j \leq C_{ji}$, then go to step 5. Otherwise, execute steps 3 and 4.

Step 3. Add the savings for reassigning j to i:

$$s_1 \leftarrow s_1 + OD_j(\tilde{C}_j - C_{ji}).$$

Step 4. (Skip if K is currently empty.) For each node $l \neq j$: if $\tilde{C}_l < C_{li}$ and l is assigned to hub m in the current solution, then $T_m \leftarrow T_m + \hat{W}_{jl}$.

Step 4.a. Compute:

$$H_j(i) = \sum_{m \in K} T_m C_{im},$$

$$s_2 \leftarrow s_2 + H_j - \tilde{H}_j(i).$$

Step 5. Increment $j \leftarrow j + 1$. If $j \leq n$, go to step 2. Otherwise, compute

$$S_i = s_1 + a s_2$$

and stop. □

(Once it is later decided which node i will actually be added to set K, the interhub costs can be updated: $H_j \leftarrow \tilde{H}_j(i)$.)

4.1.3. Choosing from a candidate list

At each step of the greedy procedure, the nodes i corresponding to the largest values of S_i are kept in a candidate list. This list has a maximum length L. The list can be truncated by requiring that all candidate elements i on the list have S_i values within α percent of the top candidate.

The overall initial solution can be summarized as follows:

INITIAL GREEDY SOLUTION

Step 1. Let $k = 1$, $K = \emptyset$.

Step 2. Compute S_i for all nodes i not in K and construct a candidate list of maximum length L as described above.

Step 3. Choose randomly from among the elements of the candidate list with each element having equal probability. Add this random choice to set K.

Step 4. Increment $k \leftarrow k + 1$. If $k \leq p$, go to step 2. Otherwise, stop. □

As noted above, many myopic measures could be used in place of S_i as defined above. We note, however, that the value of S_i defined above computes the true cost improvement for adding node i, and excellent results have been obtained using it in computational experiments (see section 5).

4.2 THE SEARCH PROCEDURE

The second step of a GRASP replication is to attempt to improve upon the greedy initial solution. Many search procedures might be designed for this purpose, e.g. variations of the exchange procedures in [21]. We choose here to implement a procedure analogous to that used in the tabu search algorithm. We summarize that procedure below:

THE SEARCH PROCEDURE

Step 1. Start with the initial greedy solution set K as computed above.

Step 2. Compute R_{qr} for all nonhub nodes q and all $r \in K$, as in eq. (4).

Step 3. Let (q', r') represent the exchange that maximizes R_{qr}. If $R_{q'r'} > 0$, let q' replace r' in set K, and return to step 2. Otherwise, stop. □

Because the initial greedy solution tends to be an excellent one, the search procedure generally terminates in very few (less than 5) iterations.

5. Computational experience

5.1. COMPUTER ENVIRONMENT AND TEST PROBLEMS

The heuristics described here were implemented in FORTRAN and run on an Amdahl 5890 computer operating under UNIX® SYSTEM 5.2.6B. The programs were compiled using F77. All processing times listed are in CPU seconds.

Test data were obtained from two sources. First is a set of problems used in [26] and later in [21]. For these, the nodes consist of lists of US cities and the traffic measures are based on 1970 airline passenger interactions as evaluated by the Civil Aeronautics Board (CAB). Problems with 10 nodes, 15 nodes, 20 nodes, and 25 nodes are used. We consider $p = 3$ and $p = 4$. These problems are small enough that all $\binom{n}{p}$ possible hub configurations can be enumerated and, assuming that nodes are to be assigned to the closest hub, optima can be determined.

The second set consists of two 52-node problems. These are based on internode traffic data used in the testing of the packet network design algorithm in [23]. They are also used in [21]. We consider cases of $p = 4$ and $p = 10$, thus generating some larger-sized problems.

5.2. TABU SEARCH AND THE CAB PROBLEMS

Table 1 summarizes the results of some of our computational experience with the Civil Aeronautics Board (CAB) data using tabu search with long-term memory. Problems with $n = 10, 15, 20, 25$ and $p = 3, 4$ were tested. For each set of values for n and p, 4 different runs were made, corresponding to values of $a = 0.4, 0.6, 0.8$ and 1.0. Tabu search parameters tested include $M = 10$ and $N = 30$, $M = 20$ and $N = 60$, $M = 30$ and $N = 90$, and $M = 40$ and $N = 120$. (Recall that M denotes the number of past iterations considered in the tabu list and N denotes the number of iterations between restarts.)

Each row of table 1 corresponds to a pair of n and p values. For each setting of the tabu search parameters, we list how many of the problems obtained optimal solutions and how many obtained suboptimal solutions. In each category (optimal and suboptimal) we also list the average time required to complete the tabu search

Table 1

Results of tabu search on CAB problems: Number of problems, average total time, average time until best solution is first entered (CPU seconds).

(M = length of tabu list, N = iterations between restarts.)

		Tabu search: $M = 10$, $N = 30$				Tabu search: $M = 20$, $N = 60$				Tabu search: $M = 30$, $N = 90$				Tabu search $M = 40$, $N = 120$				
		Optimal runs		Suboptimal runs		Optimal runs		Suboptimal runs		Optimal runs		Suboptimal runs		Optimal runs		Suboptimal runs		Enumeration time
n	p	Total time	Time to best	Total time	Time to best	Total time	Time to best	Total time	Time to best	Total time	Time to best	Total time	Time to best	Total time	Time to best	Total time	Time to best	
10	3	4		0		4		0		4		0		4		0		0.65
		0.12	0.02	N/A	N/A	0.26	0.02	N/A	N/A	0.36	0.02	N/A	N/A	0.48	0.02	N/A	N/A	
10	4	4		0		4		0		4		0		3		1		5.60
		0.15	0.09	N/A	N/A	0.34	0.17	N/A	N/A	0.39	0.09	N/A	N/A	0.56	0.09	0.28	0.08	
15	3	3		1		4		0		4		0		4		0		3.98
		0.23	0.09	0.23	0.03	0.54	0.12	N/A	N/A	0.74	0.19	N/A	N/A	0.98	0.11	N/A	N/A	
15	4	4		0		3		1		4		0		4		0		58.66
		0.33	0.20	N/A	N/A	0.80	0.18	0.80	0.24	0.91	0.26	N/A	N/A	1.13	0.26	N/A	N/A	
20	3	2		2		4		0		3		1		4		0		13.94
		0.23	0.01	0.42	0.06	0.46	0.10	N/A	N/A	0.81	0.03	1.36	0.35	1.15	0.33	N/A	N/A	
20	4	3		1		3		1		4		0		3		1		289.43
		0.33	0.05	0.64	0.49	0.89	0.11	1.27	1.01	1.10	0.30	N/A	N/A	1.68	0.04	1.68	1.21	
25	3	3		1		3		1		4		0		3		1		36.75
		0.52	0.06	0.64	0.10	1.14	0.21	1.80	1.29	1.13	0.47	N/A	N/A	1.03	0.14	3.59	2.85	
25	4	2		2		2		2		2		2		3		1		977.18
		0.62	0.06	0.76	0.15	1.20	0.06	1.50	0.72	1.80	0.06	2.02	0.32	1.99	0.63	4.78	3.59	

heuristic and the average time required until the solution was first found. Also listed in a separate column is the average time to obtain the optimum by enumeration. The results with $M = 30$ and $N = 90$ are, overall, the best, with optimal solutions being found in 29 out of the 32 cases (90.6%). Furthermore, the value of the 3 suboptimal solutions differed from the optimal by an average of only 1.0%.

The solutions obtained for other parameter settings were also very good, indicating a certain robustness in the tabu search procedure. Note also that the average time when the eventual solution is first encountered is only a fraction of the average total time. This indicates that good tabu search solutions are often found in the early stages of search.

5.3. GRASP AND THE CAB PROBLEMS

Table 2 summarizes some of our computational experience using GRASP on the CAB problems. As above, we examine problems with $n = 10, 15, 20, 25$ and $p = 3,4$. For each problem size, 4 different runs ($a = 0.4$, 0.6, 0.8 and 1.0) were made. Each run consisted of 100 replications of the algorithm.

We examine GRASP with a maximum candidate list length $L = 5$ and $L = 7$. (In both cases, we truncate the list if candidate elements are not within $\alpha = 20\%$ of the top candidate.) For each parameter setting, we list, as in table 1, how many problems obtained optimal solutions and how many obtained suboptimal solutions. In each category, we also list the average total CPU time required to complete 100 replications, as well as the average CPU time between appearances of the best solution (i.e. the average of, for the various runs, the total time divided by the number of replications on the best solution was found). Once again, for comparison purposes, we also list the time required for complete enumeration.

As can be seen, for $L = 7$, 30 out of 32 problems (93.75%= obtained optimal solutions. (For the two that did not, the value of the objective was only 0.03% and 0.16% above the optimum.) For $L = 5$, only 28 out of 32 problems obtained optimal solutions.

5.4. RESULTS FOR 52-NODE PROBLEMS

In order to examine the performance of the new heuristics on some larger problems, we ran tabu search and GRASP on some 52-node examples [23, 21]. (For these large problems, with $p = 4$ and $p = 10$, it was prohibitive to perform a complete enumeration to determine an optimal solution.) In addition to the results of our tabu search and GRASP heuristics, we also compare the results of a "double-exchange heuristic" from [21] and the results of a variation of GRASP in which initial solutions are chosen completely at random, instead of through the "greedy random" candidate list procedure.

Table 3 summarizes these computational results. Each row of the table corresponds to a particular problem with specified values of n, p and a. For each

Table 2

Results of GRASP on CAB problems: Number of problems, average total time, average time between enconters of the best solution (CPU seconds).

n	p	Optimal Runs		Suboptimal Runs		Optimal Runs		Suboptimal Runs		Enumeration time
		Total time	Time to best	Total time	Time to best	Total time	Time to best	Total time	Time to best	
10	3	4		0		4		0		0.65
		0.33	0.03	N/A	N/A	0.29	0.09	N/A	N/A	
10	4	4		0		4		0		5.60
		0.50	0.05	N/A	N/A	0.35	0.11	N/A	N/A	
15	3	4		0		4		0		3.98
		0.74	0.06	N/A	N/A	0.63	0.07	N/A	N/A	
15	4	4		0		4		0		58.66
		0.93	0.17	N/A	N/A	0.83	0.19	N/A	N/A	
20	3	1		3		3		1		13.94
		1.79	0.12	1.69	0.64	1.44	0.67	1.36	0.34	
20	4	3		1		4		0		289.43
		2.10	0.73	2.14	0.30	1.84	0.83	N/A	N/A	
25	3	4		0		4		0		36.75
		2.43	0.22	N/A	N/A	2.26	0.30	N/A	N/A	
25	4	4		0		3		1		977.18
		3.19	1.95	N/A	N/A	2.95	1.04	2.78	2.78	

algorithm tested, we list the best normalized objective function found. (These are normalized so that the best known solution equals 100.) For the columns corresponding to the tabu search procedures, we also list the total time to complete the search and the time when the best solution was first encountered, similar to table 1. For columns corresponding to GRASP procedures and to the variant with totally random starts, we also list the total CPU time expended and the average time between appearances of the best solution, similar to table 2. (For the true GRASP procedures, the total time represents 100 replications; for the variant with totally random starts, the total time represents 200 replications for $p = 4$ and 300 replications for $p = 10$.) The final column corresponds to the double-exchange heuristic described in [21]; in addition to the normalized objective, we list the total required CPU time.

Tabu search algorithms

Many different parameter settings were tested for tabu search and GRASP; we report results for the best of those examined. For tabu search, we list results for $M = 40$ and $N = 120$ and for $M = 50$ and $N = 150$. Each obtained the best known solution 8 out of 16 times.

For $n = 52$ and $p = 4$, total times for completing the tabu search varied between 6 and 16 CPU seconds, but what turned out to be the eventual solution was actually found in from 0.05 to 7.59 seconds. Similarly, for $n = 52$ and $p = 10$, total times ranged from 11.6 to almost 28 CPU seconds, but the time to first encounter the eventual solution ranged from 0.03 to 14.48 seconds. As in table 1, this suggests that good solutions are frequently found in the èarlier stages of the tabu search.

GRASP algorithms

For GRASP, we again report results for maximum candidate list length $L = 5$ and $L = 7$, and we truncate the list if candidate elements are not within $\alpha = 20\%$ of the top candidate. The setting with list length $L = 7$ generated the best known solution in 14 out of 16 cases. (With $L = 5$, this happened in only 7 of 16 cases.)

With $L = 7$, for $n = 52$ and $p = 4$, 100 replications took from 16.5 to just over 23 CPU seconds; the average time between replications that yielded the best solution ranged from 0.71 to 16.83. Likewise, for $L = 7$, $n = 52$ and $p = 10$, 100 replications ranged between 32 and 44 seconds; the average time between encounters of the best solution ranges from over 7 to over 40 seconds.

Comparisons

It is difficult to make comparisons of time requirements for tabu search and GRASP. With most other heuristics, the final solution is not determined until the end of the algorithm. With tabu search and GRASP, however, the solution that is eventually reported as the best can be encountered at any time during the computation. In addition, there are no "obvious" choices for the stopping rules for terminating the algorithm. Different stopping rules result in different time requirements as well as different solutions. Furthermore, since GRASP is a randomized algorithm, the same algorithm with a given stopping rule could yield different solutions on different runs.

One possible measure for the time requirements of tabu search is the reported "time until the best solution is first encountered". On the other hand, since GRASP is a randomized algorithm, we reported the "average CPU time between appearances of the best solution". This can give some measure of how long we might expect to take before this best solution is first encountered. In table 3, for $L = 7$, $n = 52$ and $p = 10$, GRASP yielded average times between appearances of the best solution of from over 7 to over 40 seconds. On the other hand, tabu search yielded, on the same problems, times of between 0.03 and over 14 CPU seconds until the best solution

Table 3

Results on 52-node problems: Normalized objective function and CPU times (seconds).
(M = length of tabu list, N = iterations between restarts, L = length of candidate list.)

n	p	a	Tabu search $M=40$, $N=120$	Tabu search $M=50$, $N=150$	GRASP $L=5$ $\alpha=20\%$	GRASP $L=7$ $\alpha=20\%$ tc	GRASP with totally random starts	Exchange heuristic [21]
52	4	0.4	100.00	100.00	100.00	100.00	100.14	103.54
			6.32 0.17	9.82 0.16	25.08 2.50	23.15 5.78	27.84 6.96	0.65
52	4	0.4	100.00	100.00	100.00	100.00	100.00	104.41
			7.79 0.42	9.68 0.42	16.49 0.53	16.55 0.71	17.49 0.69	1.07
52	4	0.6	100.55	100.00	102.06	100.55	100.55	103.17
			9.46 1.60	15.78 4.71	24.78 2.25	18.99 9.49	29.08 4.84	0.65
52	4	0.6	100.00	100.00	100.00	100.00	100.00	100.95
			7.81 0.65	9.71 0.64	16.40 1.82	16.50 4.12	16.94 0.94	1.07
52	4	0.8	100.00	100.00	101.12	100.00	102.02	101.55
			11.09 3.68	9.86 7.59	25.07 2.78	23.06 2.88	29.76 29.76	1.29
52	4	0.8	100.00	100.00	100.86	100.00	100.00	101.20
			7.81 0.05	9.71 0.04	16.76 2.79	16.83 16.83	17.01 17.01	1.07
52	4	1.0	100.00	100.00	100.62	100.00	101.87	103.55
			9.47 1.73	9.89 7.14	25.04 12.52	23.13 5.78	29.72 14.86	1.29
52	4	1.0	100.29	100.29	100.00	100.00	101.12	101.69
			7.80 0.41	9.70 0.40	16.27 1.35	16.53 5.51	16.18 16.18	2.10

52	10	0.4	100.00		100.15		100.53		100.61		102.36		103.04	
			14.44	9.30	21.68	14.48	41.51	20.75	35.91	35.91	48.64	48.64	3.65	
52	10	0.4	100.00		100.19		100.00		100.00		100.42		106.13	
			17.38	13.51	17.42	0.88	31.42	31.42	32.40	8.10	36.98	36.98	5.19	
52	10	0.6	100.06		100.00		100.00		100.00		103.44		101.62	
			11.60	2.27	14.67	9.90	42.37	42.37	40.83	40.83	45.06	45.06	3.65	
52	10	0.6	100.31		100.72		100.00		100.00		100.78		100.72	
			17.39	11.84	17.36	0.04	32.56	32.56	33.18	33.18	37.45	18.72	10.33	
52	10	0.8	100.54		100.54		100.00		100.00		103.64		100.54	
			14.51	0.04	14.49	8.70	42.95	42.95	41.43	20.71	48.23	48.23	3.69	
52	10	0.8	102.22		102.00		100.29		100.00		102.54		103.60	
			27.77	3.51	30.42	8.74	35.18	35.18	34.25	8.56	37.48	37.48	10.38	
52	10	1.0	102.73		102.68		100.14		100.00		106.30		100.05	
			14.48	0.03	18.13	10.97	45.46	45.46	43.86	10.96	53.89	53.89	3.72	
52	10	1.0	102.85		102.85		100.53		100.00		103.77		105.56	
			17.43	2.03	26.12	4.39	38.55	38.55	37.89	7.57	38.44	38.44	5.72	

is first encountered. This is, admittedly, an imperfect measure since it does ignore stopping rules.

In summary, for our particular implementations on this given set of test problems, tabu search showed potential advantage in CPU time, whereas GRASP generated optimal solutions somewhat more frequently. Based on these indicators, it would seem that both tabu search and GRASP are worthy of further consideration for this class of problems. Extensions and variations of both these algorithms should be considered in future research (see section 5.5).

Other algorithms

In table 3, comparisons of tabu search and GRASP against the exchange heuristic and against the search procedure with completely random solutions are also favorable. In only one instance did the exchange heuristic match the best known solution.

For search procedure with completely random initial solutions, enough replications were performed so that the CPU time required was comparable to the CPU time required for the true GRASP algorithms. (As noted above, this represents 200 replications for $p = 4$ and 300 replications for $p = 10$.) Although fairly good solutions were obtained, in only 3 cases did this approach obtain the best known solution. This indicates the value of the greedy random candidate list procedure for generating exceptionally good initial solutions.

As indicated in the introduction, tabu search and GRASP are two of many new approaches [13] to combinatorial optimization that have been developed in recent years, including simulated annealing [19, 20]. Although this author has experimented with some simulated annealing for the p-hub location problem, efforts in this direction have thus far not matched the success of the tabu search and GRASP procedures that have been developed in this paper. Therefore, we do not report detailed results here.

5.5. CONCLUSIONS

Some particular implementations of both tabu search and GRASP heuristics have demonstrated excellent results in obtaining optimal or best known solutions for a restricted version of the p-hub location problem. As noted above, for a given set of test problems, these implementations of tabu search and GRASP generated known optima in 90.60% and 93.75% of the cases, respectively. Both these algorithms, and variations on them, therefore seem worthy of further consideration.

For tabu search, variations may include other parameter settings, restart strategies or value functions. To consider one example, an alternative value function might be designed that would take into account changes in node assignments when hubs are exchanged. (The value function (4) described in section 3.1 assumes that node assignments remain unchanged.) Such a function would, naturally, be more difficult

and time-consuming to compute, but the potentially more accurate savings estimate might lead to improved searches.

For GRASP, one may consider other parameter settings, other myopic criteria for choosing nodes, or other search procedures. In particular, alternative value functions, such as the one described above for tabu search, can likewise be incorporated into the GRASP search procedure (section 4.2).

Of particular interest, however, would be extending these tabu search and GRASP approaches to consider other types of assignment schemes besides distance-based assignments. (The "optimal" solutions given here are, clearly, not necessarily optimal when assignments are unrestricted.) The success of tabu search and GRASP approaches, for the restricted case of distance-based assignments, does give encouragement that appropriate extensions and variations of these approaches can be successful for the general case of the p-hub problem as well.

Acknowledgements

The author thanks Professor Thomas A. Feo for helpful discussions, also Professor Morton O'Kelly for providing the CAB data, and Dr. Beth S. Munson for providing the 52-node data.

References

[1] J.F. Bard and T.A. Feo, An algorithm for the manufacturing equipment selection problem, IIE Trans. 23(1991)83–92.

[2] J.F. Bard and T.A. Feo, Operations sequencing in discrete parts manufacturing, Manag. Sci. 35(1989)249–255.

[3] R.R. Boorstyn and H. Frank, Large-scale network topological optimization, IEEE Trans. Commun. COM-25(1977)29–47.

[4] M.L. Brandeau and S.S. Chiu, An overview of representative problems in location research, Manag. Sci. 35(1989)645–674.

[5] T.A. Feo and J.F. Bard, Flight scheduling and maintenance base planning, Manag. Sci. 35(1989)1415–1432.

[6] T.A. Feo and M.G.C. Resende, A probabilistic heuristic for a computationally difficult set covering problem, Oper. Res. Lett. 8(2)(1989).

[7] T.A. Feo, M.G.C. Resende and S.H. Smith, A greedy randomized adaptive search procedure for maximum independent set, unpublished manuscript (1989).

[8] F. Glover, Heuristics for integer programming using surrogate constraints, Dec. Sci. 8(1977)156–166.

[9] F. Glover, Future paths for integer programming and links to artificial intelligence, Comput. Oper. Res. 13(1986)533–549.

[10] F. Glover, Tabu search methods in artificial intelligence and operations research, ORSA Art. Int. Newsletter 1(2)(1987)6.

[11] F. Glover, Tabu search, Part I, ORSA J. Comput. 1(1989)190–206.

[12] F. Glover, Tabu search, Part II, ORSA J. Comput. 2(1990)4–32.

[13] F. Glover and H.J. Greenberg, New approaches for heuristic search: A bilateral linkage with artificial intelligence, Europ. J. Oper. Res. 39(1989)119–130.

[14] F. Glover, C. McMillan and B. Novick, Interactive decision software and computer graphics for architectural and space planning, Ann. Oper. Res. 5(1985)557–573.
[15] F. Glover and C. McMillan, The general employee scheduling problem: An integration of management science and operations research, Comput. Oper. Res. 13(1986)563–593.
[16] P. Hansen, The steepest ascent mildest descent heuristic for combinatorial programming, presented at the *Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italy (1986).
[17] J.P. Hart and A.W. Shogan, Semi-greedy heuristics: An empirical study, Oper. Res. Lett. 6(1987)107–114.
[18] A. Hertz and D. de Werra, Using tabu search techniques for graph coloring, Computing 29(1987)345–351.
[19] D.S. Johnson, C.R. Aragon, L.A. McGeoch and C. Schevon, Optimization by simulated annealing: An experimental evaluation (Part I), Oper. Res. 37(1989)865–892.
[20] S. Kirkpatrick, C.D. Gelatt, Jr. and M.P. Vecchi, Optimization by simulated annealing, Science 22(1983)671–680.
[21] J.G. Klincewicz, Heuristics for the p-hub location problem, Europ. J. Oper. Res. 53(1991)25–37.
[22] R.F. Love, J.G. Morris and G.O. Wesolowsky, *Facilities Location: Models and Methods*, Publications in Operations Research, vol. 7(North-Holland/Elsevier, New York, 1988).
[23] C.L. Monma and D.D. Sheng, Backbone network design and performance analysis: A methodology for packet switching networks, IEEE J. Sel. Areas Commun. SAC-4(1986)946–965.
[24] M.E. O'Kelly, The location of interacting hub facilities, Transp. Sci. 20(1986)92–106.
[25] M.E. O'Kelly, Activity levels at hub facilities in interacting networks, Geographic Anal. 18(1986)343–356.
[26] M.E. O'Kelly, A quadratic integer program for the location of interacting hub facilities, Europ. J. Oper. Res. 32(1987)393–404.
[27] W.B. Powell and Y. Sheffi, Design and implementation of an interactive optimization system for network design in the motor carrier industry, Oper. Res. 37(1989)12–29.
[28] J. Skorin-Kapov, Tabu search applied to the quadratic assignment problem, ORSA J. Comput. 2(1990)33–45.