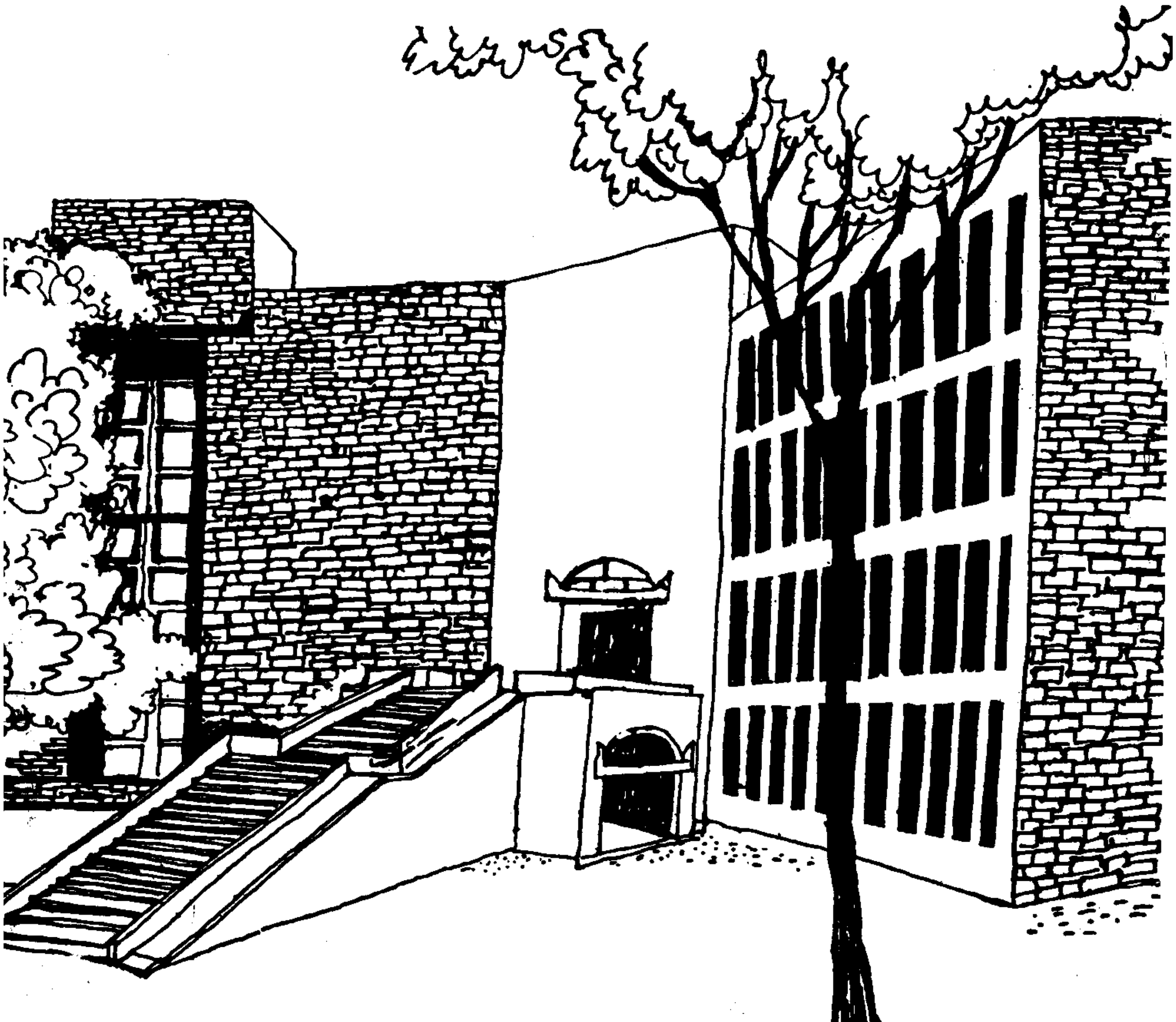

Neighborhood Search Heuristics for the Uncapacitated Facility Location Problem

Diptesh Ghosh

W.P.No. 2002-01-01 /1684
January 2002

INDIAN INSTITUTE OF MANAGEMENT
AHMEDABAD-380 015
INDIA

Neighborhood Search Heuristics for the Uncapacitated Facility Location Problem

Diptesh Ghosh
P&QM Area, Indian Institute of Management Ahmedabad,
Vastrapur, Ahmedabad 380015, India.
Email: diptesh@imahd.ernet.in

January 21, 2002

Abstract

The uncapacitated facility location problem is one of choosing sites among a set of candidates in which facilities can be located, so that the demands of a given set of clients are satisfied at minimum costs. Applications of neighborhood search methods to this problem have not been reported in the literature. In this paper we first describe and compare several neighborhood structures used by local search to solve this problem. We then describe neighborhood search heuristics based on tabu search and complete local search with memory to solve large instances of the uncapacitated facility location problem. Our computational experiments show that on medium sized problem instances, both these heuristics return solutions with costs within 0.075% of the optimal with execution times that are often several orders of magnitude less than those required by exact algorithms. On large sized instances, the heuristics generate low cost solutions quite fast, and terminate with solutions whose costs are within 0.0345% of each other.

Keywords: metaheuristics, tabu search, complete local search with memory, facility location.

1 Introduction

Location problems are some of the most widely studied problems in combinatorial optimization (see Mirchandani and Francis [15] for a detailed introduction). The basic setting of the problems is the following. We have a set of sites in which facilities can be located, and a set of clients who have requirements that are to be satisfied by the facilities to be set up. The objective is to determine the sites in which facilities must be set up to satisfy the client requirements at minimum cost. Based on additional assumptions, location problems can be classified into four basic categories: p-median problems, p-center problems, uncapacitated facility location problems, and capacitated facility location problems. In the p-median problem, the cost to be minimized is simply that of transporting commodities from facilities to clients. In the p-center problem, the cost of a solution is defined as the maximum cost incurred in order to satisfy any single client. For both the problems described above, facilities can be located in exactly p sites. In the last two types of problems, the cost of satisfying the client requirements has two components — a fixed cost component of setting up a facility in a given site, and a transportation cost component of satisfying the client requirements. The uncapacitated problem assumes infinite capacities for all the facilities, while the capacitated problem assumes a finite capacity for each facility. In this paper we study the uncapacitated facility location problem (UFLP). Formally defined, the UFLP is the following:

Problem: **Uncapacitated Facility Location (UFLP)**

Instance: Sets $I = \{i_1, i_2, \ldots, i_m\}$ of sites in which facilities can be located, $J = \{j_1, j_2, \ldots, j_n\}$ of clients, a vector $F = (f_i)$ of fixed costs for setting up facilities at sites $i \in I$, and a matrix $C = [c_{ij}]$ of transportation costs from $i \in I$ to $j \in J$.

Output: $\arg\min\{\mathfrak{F}(S) = \sum_{i \in S} f_i + \sum_{j \in J} \min\{c_{i,j} | i \in S\} : \emptyset \subset S \subseteq I\}$.

The *size* of a UFLP instance is denoted by $m \times n$, where m and n are the cardinalities of the sets I and J respectively. Each set S ($\emptyset \subset S \subseteq I$) represents a *solution* to the instance according to the rule: $i \in S \iff$ a facility is located at site i in the solution represented by S.

The UFLP forms the underlying model in several combinatorial problems, like set covering, set partitioning, information retrieval, simplification of logical Boolean expressions, airline crew scheduling, vehicle despatching (Christofides [5]), assortment (among others, Beresnev *et al.* [3], Goldengorin [10], Pentico [16, 17]), and is a subproblem for various location analysis problems (Revelle and Laporte [18]). The function $\mathfrak{F}(S)$ is called the *cost* or the *objective function* of the solution S. The UFLP is known to be $\mathcal{NP}$-hard (Cornuejols *et al.* [6]), and many exact and heuristic algorithms to solve the problem have been discussed in the literature. An annotated bibliography of several exact solution approaches appears in Labbé and Louveaux [14]. Methods for generating challenging data sets of large instances of these problems have been suggested in Körkel [12].

Neighborhood search based methods have been overlooked for the UFLP, although a tabu search based method for solving the related p-median problem has been studied (see Rolland *et al.* [19]). A tabu search procedure for the location-allocation problem has also been reported in Tuzun and Burke [20]. Our aim in this paper is to study neighborhood search based methods for the UFLP. In the next section we study the performance of the steepest descent local search heuristic using various neighborhood structures. We choose the neighborhood structure among these that performs best on test problems, and discuss implementations of tabu search and complete local search with memory in Section 3. We report our computational experience with these heuristics on randomly generated large instances of the UFLP in Section 4, and summarize the paper in Section 5.

2 Local Search and Choice of Neighborhoods

Local search is perhaps the simplest among neighborhood search methods. It starts with a given initial solution and checks its neighborhood for a better solution. If such solutions exist, then local search designates the best solution found in the neighborhood as the current solution and repeats the process. In case the neighborhood of the current solution does not contain any solution better than it, local search returns the current solution and terminates.

This method does not guarantee globally optimal solutions to most combinatorial problems, but generally returns relatively good quality solutions. Of course, the effectiveness of the method depends on the neighborhood structure used. In this section we test the performance of local search on UFLP instances using three neighborhood structures, the *Add-Swap* neighborhood, the *2-Swap* neighborhood, and the *Permutation* neighborhood.

2.1 Neighborhood Structures

Add-Swap Neighborhood This neighborhood structure was used in Tuzun and Burke [20] to solve location problems, and seems to be motivated by local search heuristics for the p-median problem. The neighborhood is defined by two different kinds of moves: *swap moves* and *add moves*. A swap moves removes a facility from one of the sites where it was located in the current solution and simultaneously opens a facility in a site that had none. This kind of move keeps the number of open facilities in the solution constant. An add move, on the other hand, simply opens a facility in one of the sites where no facility was open in the current solution. Thus it increases the number of open facilities located by one. The local search procedure on which the tabu search procedure described in Tuzun and Burke [20] is based starts by opening a facility in one of the locations on the instance. It then enters a *swap phase* in which swap moves are executed until no more swap moves improve the solution. After that local search enters an *add phase* in which add moves are executed until no more add moves improve the solution. The swap and add phases alternate until a local optimum is reached.

2-Swap Neighborhood This is a very common neighborhood structure used in combinatorial optimization problems. In this neighborhood, a move from a solution to another can be executed by one of three ways — either by executing an add move, or by executing a swap move, or by removing a facility from one of the sites where it was located in the current solution.

Permutation Neighborhood Since the objective function value of the UFLP is supermodular, the following result holds true.

Result 1 (Cherenin [4]) *Consider solutions* $S_1, S_2, \ldots, S_m$, *where* $|S_k| = k$ *and* $\emptyset, S_1 \subset S_2 \subset \ldots \subset S_m = I$. *Assume that* $\mathfrak{F}(\emptyset) = \infty$. *Then there exists* p, $1 \leq p \leq m$ *such that*

$$\mathfrak{F}(\emptyset) = \mathfrak{F}(S_0) > \mathfrak{F}(S_1) \geq \cdots \geq \mathfrak{F}(S_p) \leq \cdots \leq \mathfrak{F}(S_m) = \mathfrak{F}(I).$$

This result allows us to define the greedy heuristic GREEDY of Figure 1 that runs in $\mathcal{O}(mn)$ time.

```
Heuristic GREEDY
 Input:   I, J, F, C, and a permutation $\Pi = (\pi_1, \ldots, \pi_m)$ of I.
 Output:  A solution to the instance.
 Code:
 begin
      $S := \emptyset$;
      for $i := 1$ to $m$ do
      begin
           $gain := \mathfrak{F}(S \cup \{i_{\pi_k}\}) - \mathfrak{F}(S)$;
           if $gain \geq 0$
                $S := S \cup \{i_{\pi_k}\}$;
           else return S;
      end;
      return S;
 end.
```

Figure 1: The GREEDY algorithm

Now consider the set $\mathfrak{S}$ of all possible solutions to an UFLP instance obtained by running GREEDY on it using various permutation vectors. Clearly, all optimal solutions to the instance are present in this set, so that we can restrict local search to its members. Again, since each of the solutions in $\mathfrak{S}$ is defined by a permutation Π of the elements of I, (although not uniquely, since more than one permutation can result in the same solution) we can construct a permutation neighborhood based on these permutation vectors. In this neighborhood structure, two solutions S_A and S_B, $(S_A, S_B \in \mathfrak{S})$ obtained by running GREEDY with permutations Π_A and Π_B respectively, are neighbors if and only if Π_A and Π_B differ in exactly two positions. A similar neighborhood structure has been used in Ghosh and Chakravarti [7] on subset-sum problems.

2.2 Performance on Test Problems

We implemented local search using each of the three neighborhood structures mentioned in the previous subsection. In order to test their relative performance, we used test instances similar to those described in Körkel [12]. The problems had size $m = n = 100$, and were of two types, symmetric and asymmetric. The symmetric problems were identical to the "small-scale" problems described in Körkel [12]. They were divided into four classes, SYM-1 through SYM-4. SYM-1 and SYM-2 containing 45 instances each, and SYM-3 and SYM-4 containing 15 instances each.

The asymmetric problems closely mimicked the symmetric problems, but the initial choice of the transportation costs was random, chosen from a uniform distribution supported on $[0, 2500]$. The performance of the local search heuristics were evaluated using two parameters — their suboptimality, defined as

$$\text{Suboptimality} = \frac{\text{cost of the solution returned by local search} - \text{cost of an optimal solution}}{\text{cost of an optimal solution}},$$

and the execution time required by local search on a computer with a 650 MHz Intel Mobile Celeron Processor.

Table 1: Performance of local search on symmetric small Körkel instances

Average suboptimality

Problem	Neighborhoods		
Set	Add-Swap	2-Swap	Permutation
SYM-1	0.004	0.002	0.003
SYM-2	0.020	0.023	0.019
SYM-3	0.006	0.004	0.005
SYM-4	0.015	0.014	0.016

Average execution time (CPU seconds)

Problem	Neighborhoods		
Set	Add-Swap	2-Swap	Permutation
SYM-1	0.267	0.267	6.289
SYM-2	0.089	0.044	2.756
SYM-3	0.200	0.200	2.133
SYM-4	0.067	<0.001	1.400

Table 2: Performance of local search on asymmetric small Körkel-type instances

Average suboptimality

Problem	Neighborhoods		
Set	Add-Swap	2-Swap	Permutation
ASYM-1	0.018	0.014	0.014
ASYM-2	0.024	0.029	0.026
ASYM-3	0.008	0.010	0.016
ASYM-4	0.034	0.032	0.031

Average execution time (CPU seconds)

Problem	Neighborhoods		
Set	Add-Swap	2-Swap	Permutation
ASYM-1	0.222	0.067	6.022
ASYM-2	0.156	0.067	3.400
ASYM-3	0.200	<0.001	3.467
ASYM-4	0.133	0.067	2.467

Tables 1 and 2 summarize our computational experience with local search using various neighborhood structures on the test instances. The results tabulated is the average of the results for all the instances in a particular set. We see that the use of different neighborhood structures do not result in markedly different solution qualities, either for the symmetric problems or for the asymmetric problems. However, the execution times required by local search using the 2-swap neighborhood is clearly less than those required by the other two, especially for the asymmetric problems. Therefore we will use the 2-swap neighborhoods in the implementations of tabu search and complete local search with memory discussed in the next section.

3 Advanced Neighborhood Search Procedures

In spite of the advantages of the local search procedure described in the previous section, it is rarely used to solve large problems. The main disadvantage of the procedure is that it gets stuck at local optima, which may be far from a global optimum in terms of the solution quality. Popular enhancements to local search generally include strategies to move out of such local optima. In this section we describe two such enhancements for local search applied to the UFLP — a tabu search procedure, which makes use of tabu lists to guide the search, and a complete local search with memory procedure that backtracks out of local optima once it reaches them.

3.1 Tabu Search

Tabu Search (TS — see, e.g., Glover and Laguna [9]) is one of the most effective improvements on local search known in the literature. It follows the basic principle of local search, moving at each iteration from the current solution to the best solution available in the neighborhood. The neighborhood of all solutions change continuously depending on the search history, thereby guiding the search to regions in the search space that it has not visited earlier. This helps the search to move out of local optima. The TS procedure described in this subsection closely follows the TSpMP procedure described in Rolland *et al.* [19]. It uses the 2-swap neighborhood structure, which we found to be effective for the UFLP (see Section 2), incorporates recency and frequency based memory, and an aspiration criterion to guide the search.

Recency based memory The recency based memory used in our implementation discourages sites that were involved in recent moves to participate in a move at the current iteration. This is done by maintaining a tabu list. After each iteration, the sites involved in the move at the current iteration are put in the tabu list, (they are said to achieve a tabu status), and stay there for a pre-defined number of iterations (called the tabu tenure). Sites with a tabu status cannot participate in moves during their tabu tenure, except if they satisfy an aspiration criterion described later. Once the tabu status of a site is removed, it can participate in future moves, and no permanent record of its past tabu status is maintained. This kind of memory is thus called short-term or recency based memory.

The length of the tabu tenure can be set using one of three approaches: fixed, dynamic, and random. We use a random tabu tenure, very similar to the one used in Rolland *et al.* [19]. In our implementation, the tabu tenure is set to a randomly selected number within a pre-specified interval.

Frequency based memory Frequency based memory is one of the tools used for diversification in TS implementations. The rationale behind using this kind of memory is to discourage the search from being restricted to a small region of the solution space by discouraging frequently made moves. In our implementation, we maintain a list freq that keeps a record of the number of times a certain site has participated in a move during the history of the search. We then construct a penalty function which penalizes the use of sites that have a high value in the list. Following the implementation described in Rolland *et al.* [19], we construct the penalty function $\Pi(i)$ for a site i of the form $\Pi(i) = k \times \mathsf{freq}(i)$, where k is a pre-defined constant. The penalized cost function for a solution S, reached from a solution S_0 to the instance is thus

$$\mathfrak{F}_P(S, S_0) = \mathfrak{F}_P(S) - \sum_{i \in (S \setminus S_0) \cup (S_0 \setminus S)} \Pi(i)$$

The freq list is maintained and updated throughout the history of the search. This type of memory is thus also called long term memory.

Aspiration criterion Aspiration criteria are employed in TS implementations to ensure that moves which are exceptionally promising are not ignored due to the tabu status of some component element. Sophisticated TS methods use aspiration criteria that are dependent on the the portion of the solution space that the method is searching, or use multiple criteria. Our implementation uses the following simple rule:
A move is said to satisfy the aspiration criterion if it results in a solution with a cost lower than that of the best solution found thus far.

The pseudocode for our TS implementation is presented in Figure 2. We performed computational experiments to compute the values of the various parameters of the algorithm. Our results are based on eight instances with various values of fixed and transportation costs. These instances were generated using the guidelines in Körkel [12] and had 400 locations and 400 clients each. We observed that a small value of k for the penalty function $\Pi(\cdot)$ resulted in the best solution costs. Therefore k was set to 10. Several combinations of the values of $\mathsf{maxtabu}$ and $\mathsf{maxiter}$ (refer Figure 2) were used to solve the eight instances in order to find the best combination. Table 3 summarizes our findings.

The most surprising finding from these experiments was that short term memory did not help to improve the performance of TS for the UFLP. In fact the existence of a tabu list worsened the

```
Heuristic TS
 Input:   I, J, F, C, and an initial solution $S_0$ to the instance.
 Output:  A solution to the instance.
 Code:
 begin
     best := $S_0$;
     for iteration := 1 to maxiter do
     begin
         remove all sites from the tabu list whose tabu tenure is complete;
         $S_{nt} := \arg\min\{\mathfrak{F}_P(S, S_0) : \text{a move from } S_0 \text{ to } S \text{ is non-tabu}\}$;
         $S_t := \arg\min\{\mathfrak{F}_P(S, S_0) : \text{a move from } S_0 \text{ to } S \text{ is tabu}\}$;
         if $\mathfrak{F}_P(S_{nt}) < \mathfrak{F}_P(S_t)$
         begin
             if $\mathfrak{F}(S_{nt}) < \mathfrak{F}(\text{best})$
                 $\mathfrak{F}(\text{best}) := \mathfrak{F}(S_{nt})$;
             if $\mathfrak{F}(S_{nt}) < \mathfrak{F}(S_0)$
                 $\mathfrak{F}(S_0) := \mathfrak{F}(S_{nt})$;
             set the tabu tenure for each site participating in the
             move from $S_0$ to S to a random number between 1 and maxtabu ;
         end;
         else
         begin
             if $\mathfrak{F}(S_t) < \mathfrak{F}(\text{best})$
             begin
                 $\mathfrak{F}(\text{best}) := \mathfrak{F}(S_t)$;
                 $\mathfrak{F}(S_0) := \mathfrak{F}(S_t)$;
             end;
             set the tabu tenure for each site participating in the
             move from $S_0$ to S to a random number between 1 and maxtabu ;
         end;
     end;
     return best;
 end.
```

Figure 2: Our TS implementation

Table 3: Choice of parameters for our TS implementation

Average solution costs

maxtabu	maxiter 50	75	100
0	730629.88	723949.13	723153.88
5	735204.38	732275.00	731502.38
10	735204.38	732275.00	731502.38

Average execution time (CPU seconds)

maxtabu	maxiter 50	75	100
0	30.19	44.77	61.43
5	30.30	45.31	60.32
10	30.49	45.09	60.19

quality of the solution, and increased the execution time slightly. Therefore in our implementation, we set $\mathtt{maxtabu}$ at 0. This means that we do not make use of short term memory. The second observation we made is that the solution cost decreased with an increase in $\mathtt{maxiter}$ but the solution time increased. We therefore decided to allow TS iterations to continue until a pre-specified execution time is exceeded.

3.2 complete local search with memory

complete local search with memory (CLM — see Ghosh and Sierksma [8]) is a new variant of local search. This heuristic uses a graph search based approach to search the neighborhood graph for the UFLP. It does this by manipulating three sets, called LIVE, DEAD, and NEWGEN. LIVE contains solutions that are available for the heuristic for future exploration. DEAD contains solutions that have been already considered by the heuristic. This ensures that no solution is explored by CLM more than once. The third list, NEWGEN temporarily stores solutions that are generated by CLM during the current iteration.

CLM starts by defining empty sets LIVE, DEAD, and NEWGEN, and putting an initial solution P into LIVE. It then performs iterations until a pre-specified stopping condition is reached. Each iteration starts by choosing a pre-specified number k of solutions from LIVE. In our implementation $k = 1$. The neighborhood of these solutions is then searched (the solution is said to have been explored), and each neighbor that has a cost better than a given threshold value τ is added to NEWGEN. In our implementation we fix τ to the cost of the solution currently being explored. When all k solutions have been explored, then the solutions in NEWGEN that are not already present in either LIVE or DEAD are moved to LIVE and the remaining solutions discarded. This is because, if a solution is present in LIVE, then it is already under consideration by CLM, but has not been explored since other more attractive solutions are present in LIVE. If it is present in DEAD, CLM has already explored it, and knows the outcome of the exploration. This transfer of solutions from NEWGEN to LIVE marks the end of an iteration. If LIVE$\neq \emptyset$ when the stopping condition is reached, then a postprocessing operation is carried out. In our implementation, the postprocessing operation involves carrying out local search using the r lowest cost (elite) solutions present in LIVE. If any solution in LIVE is encountered while performing local search on an elite solution, then that solution is not considered while choosing the remaining elite solutions. This is done to increase the possibility of choosing good solutions from diverse areas of the solution space for the postprocessing operation. The heuristic returns the lowest cost solution it encounters.

The pseudocode for our CLM implementation is presented in Figure 3. We stop iterations either if LIVE is empty at the beginning of an iteration, or if we have already stored a sufficient number ($\mathtt{maxnode}$) of nodes in LIVE, DEAD, and NEWGEN. We performed preliminary computational experiments to choose the values of $\mathtt{maxnode}$ and r. The computations were carried out on the same set of UFLP instances the same set of instances that we used for selecting the parameters for our TS implementation. Table 4 summarizes our findings.

Table 4: Choice of parameters for our CLM implementation

Average solution costs

	maxnode			
r	100	200	500	1000
1	723068.13	723068.13	722068.13	724442.25
3	720578.63	720768.38	722995.63	724370.75
5	714877.88	714607.50	714799.75	722726.25
10	714784.00	714513.63	713270.75	717915.50

Average execution time (CPU seconds)

	maxnode			
r	100	200	500	1000
1	68.71	69.37	68.79	69.73
3	141.97	145.72	138.28	140.55
5	211.65	206.21	209.45	209.01
10	385.09	391.38	384.41	377.44

Note that the cost of the solution returned by CLM did not improve appreciably when value of $\mathtt{maxnode}$ was increased. In fact the solutions returned when $\mathtt{maxnode}$ was set at 1000 were consistently worse than when $\mathtt{maxnode}$ was set at 500. For our implementation therefore, we set $\mathtt{maxnode}$ at 500. The cost of the solution returned by the CLM heuristic improved significantly

```
Heuristic CLM
 Input:    I, J, F, C, and an initial solution $S_0$ to the instance.
 Output:   A solution to the instance.
 Code:
 begin
      best := $S_0$;
      LIVE := $\{S_0\}$;
      DEAD := $\emptyset$; NEWGEN := $\emptyset$;
      while LIVE $\neq \emptyset$ do
      begin
      choose solution P from LIVE;
           for each neighbor $P_n$ of P such that $\mathfrak{F}(P_n) \leq \mathfrak{F}(P)$ do
           begin
                NEWGEN := NEWGEN $\cup \{P_n\}$;
                if |LIVE| + |DEAD| + |NEWGEN| > maxnode
                begin
                     transfer all solutions in NEWGEN that are not already in LIVE
                     or DEAD to LIVE;
                     NEWGEN := $\emptyset$;
                     go to postproc;
                end;
           end;
           LIVE := LIVE $\setminus \{P\}$;
           DEAD := DEAD $\cup \{P\}$;
           transfer all solutions in NEWGEN that are not already in LIVE or DEAD to LIVE;
           NEWGEN := $\emptyset$;
      end;
      if LIVE = $\emptyset$ go to complete;
 postproc:
      for count := 1 to r do
      begin
           perform local search on a lowest cost solution P in LIVE to obtain a locally
           optimal solution $P_{lo}$;
           if $\mathfrak{F}(P_{lo}) < \mathfrak{F}(\text{best})$
                best := $P_{lo}$;
           remove all solutions visited in the previous step from LIVE;
      end;
 complete:
      return best;
 end.
```

Figure 3: Our CLM implementation

when the value of r increased. This is intuitively clear since a larger value of r allows the heuristic to check diverse areas of the solution space for good solutions. However, the ratio of the rate of improvement of the solution cost and the rate of increase in execution time was observed to be the maximum at $r = 5$. Thus we fix the value of r to 5 for our computational experiments.

4 Computational Experience

The TS and CLM implementations discussed in the previous section were used to solve 180 UFLP instances, of sizes varying from 75×75 to 750×750. The elements of the transportation cost matrix C for each of these instances were chosen from a uniform distribution supported on $[1000, 2000]$. Two types of transportation cost matrices were considered, symmetric and asymmetric. In the symmetric type of matrices, $c_{ij} = c_{ji}$, but no such restrictions were imposed for the asymmetric matrices. The elements f_i of the fixed cost vector F were also chosen from uniform random distributions. Three different intervals were used as supports in our experimentation:

- $[100, 200]$ to simulate instances with low values of fixed costs,
- $[1000, 2000]$ to simulate instances with medium values of fixed costs, and
- $[10000, 20000]$ to simulate instances with high values of fixed costs.

We generated five instances for any given type of instance (specified by the size of the instance, the nature of the transportation cost matrix, and the support for the fixed cost distribution). Since the costs of the solutions returned by both the heuristics for different instances of the same type were not found to be significantly different, we use the average of the costs of the solutions for presentation and comparison purposes.

The initial solutions for both the TS and CLM implementations are constructed using the GREEDY algorithm described in Figure 1. We observed that this initial solution gives rise to very good quality solutions for small sized instances within reasonable time, and so we did not experiment with other methods of generating initial solutions. We also did not use any special data structures to make the swap process efficient. We used an array implementation of the memory structures used in TS, and a binary tree to store the solutions of LIVE, DEAD, and NEWGEN in our CLM implementation. We allowed the two heuristics to run for the same duration in order to make a fair comparison between them. To achieve this, we solved each of the instances using CLM, and then allowed TS the same amount of time to solve the instance. The experiments were conducted on a computer with a 650 MHz Intel Mobile Celeron processor and 64 MB RAM.

The first results that we present are on 90 medium sized UFLP instances with $m = n = 75$, 100, and 125. These instances are small enough to be solved by exact algorithms (for example, an adaptation of the data correcting algorithm in Goldengorin *et al.* [11]), and are yet large enough to occupy TS and CLM for more than 0.01 CPU seconds. We saw that TS returns optimal solutions to 67 of the 90 instances solved, while CLM returns optimal solutions to 59 of the instances. We present the details of the performance of the heuristics on these problems in Table 5. Each of the values in the table is the average of the five instances generated for that problem type. Since we know the cost of an optimal solution for each of the instances in the problems in Table 5, we can compute the suboptimality of the solutions that are returned by the two heuristics. The suboptimality of a heuristic solution S^H compared to an optimal solution S^* to a UFLP instance is defined as

$$\text{Suboptimality of } S^H = \frac{\mathfrak{F}(S^H) - \mathfrak{F}(S^*)}{\mathfrak{F}(S^*)}$$

and is expressed in Table 5 as a percentage.

The results show that the solutions returned by both CLM and TS are very close to optimal for all the medium sized instances studied here. In general, the quality of solutions returned by TS were better for the symmetric UFLP instances than for the asymmetric instances, but no such trend was clear for CLM. The best quality solutions for both the heuristics were for instances with high f_i values. The quality of solutions returned by TS were worst for instances with medium f_i

values, while the quality of solutions returned by CLM were worst for instances with low f_i values. The execution time taken by the heuristics were often several orders of magnitude less than the time taken by the exact algorithm. Both the exact algorithm and the heuristics normally required longer execution times to solve instances with low f_i values, although the effect was much more pronounced for the exact algorithm than for the heuristics. Both the exact algorithm and the heuristics required longer execution times for symmetric instances than for asymmetric instances. In sum, both TS and CLM generate high quality solutions to medium sized UFLP instances within relatively short execution times. TS however, returns marginally better solutions than CLM for these instances.

Three sets containing 30 large sized UFLP instances each, with $m = n = 250$, 500, and 750 respectively, were also solved using TS and CLM. These instances are too large to solve within reasonable time using exact algorithms, so we cannot comment on the suboptimality of the heuristic solutions. Table 6 presents the results of our experiments with these instances. As with Table 5, the figures presented are average values over the results for the five instances in each problem type. The relative performance of TS and CLM is measured by the gap between the solutions returned by them. The gap is defined as

$$\text{Gap} = \frac{|\text{cost of the solution returned by CLM} - \text{cost of the solution returned by TS}|}{\text{cost of the solution returned by CLM}}$$

and is expressed in Table 6 as a percentage. We see that the gap between the solutions returned by TS and CLM remain small for all instances (the average gap over all the instances being 0.0345%). In most cases, the gap is seen to reduce when the problem size increases.

The last three columns in Table 6 respectively present the number of instances (out of 5) in which TS returned a better solution than CLM, in which they both returned identical solutions, and in which CLM returned a better solution that TS. We see that the solutions returned by TS returns better solutions in 51 of the 90 instances, and CLM returns better solutions in 16 instances. In the other 23 instances they return identical solutions. TS is seen to out-perform CLM for most instances with low values of fixed costs. CLM however performs better on instances with high values of fixed costs.

We also see that for all three problem sizes, the time required to solve instances in which the elements of the fixed cost vector were drawn from $\mathcal{U}[1000, 2000]$ were usually easier to solve than the other types of instances. The instances in which the elements of the fixed cost vector were drawn from $\mathcal{U}[100, 200]$ usually required the longest execution times. Execution times were seen to increase rapidly with problem sizes. But since the logarithm of the execution times is an increasing function of problem sizes with decreasing slope, the increase in execution times with problem size is not likely to be exponential. However the high rates of increase indicate that these heuristics would require very long execution times on larger sized instances.

The variation of solution quality with execution times is also interesting for both the heuristics. Figure 5 shows this variation for one asymmetric UFLP instance with $m = n = 500$, but the graph is quite characteristic. Note that relatively high quality solutions are achieved quite early into the search by both the heuristics. The subsequent time is spent searching for relatively minor improvements. Therefore, even if the available execution time is limited, both the heuristics would be able to generate high quality solutions.

5 Summary

In this paper, we study the performance of generic local search, tabu search, and complete local search with memory on the uncapacitated facility location problem. Generic local search was implemented with three different types of neighborhood structures, the add-interchange neighborhood, the 2-swap neighborhood, and the permutation neighborhood. We observed that local search returned almost similar quality solutions with each of these neighborhood structures, but took the least amount of execution times with the 2-swap neighborhoods. We used the 2-swap neighborhood structure in our implementations of tabu search and complete local search with memory. Our tabu search implementation closely follows the TSpMP implementation in Rolland

Table 5: Performance of CLM and TS for medium sized UFLP instances

Problem Details			Optimal solution		Heuristic solutions			Suboptimality (%)	
$m\ (=n)$	C matrix	f_i chosen from	Cost	Time (CPU Sec.)	Cost-TS	Cost-CLM	Time (CPU Sec.)	TS	CLM
75	Symmetric	$\mathcal{U}[100, 200]$	79285.8	2.954	79285.8	79290.4	0.396	0.0000	0.0058
		$\mathcal{U}[1000, 2000]$	89016.8	1.194	89028.2	89034.8	0.504	0.0128	0.0202
		$\mathcal{U}[10000, 20000]$	117960.0	0.460	117960.0	117960.0	0.030	0.0000	0.0000
	Asymmetric	$\mathcal{U}[100, 200]$	79384.4	3.110	79384.4	79422.4	0.406	0.0000	0.0479
		$\mathcal{U}[1000, 2000]$	89071.4	1.276	89120.2	89071.4	0.770	0.0548	0.0000
		$\mathcal{U}[10000, 20000]$	117333.4	0.448	117333.4	117333.4	0.054	0.0000	0.0000
100	Symmetric	$\mathcal{U}[100, 200]$	105147.6	142.706	105157.6	105191.0	0.988	0.0095	0.0413
		$\mathcal{U}[1000, 2000]$	116669.2	14.630	116680.0	116680.0	0.867	0.0093	0.0093
		$\mathcal{U}[10000, 20000]$	149509.6	1.315	149509.6	149509.6	0.161	0.0000	0.0000
	Asymmetric	$\mathcal{U}[100, 200]$	104908.2	71.274	104923.6	104961.0	0.946	0.0147	0.0503
		$\mathcal{U}[1000, 2000]$	116214.8	12.789	116295.2	116301.6	0.528	0.0692	0.0747
		$\mathcal{U}[10000, 20000]$	149135.4	1.261	149135.4	149135.4	0.095	0.0000	0.0000
125	Symmetric	$\mathcal{U}[100, 200]$	130651.8	627.262	130664.6	130747.4	1.670	0.0098	0.0732
		$\mathcal{U}[1000, 2000]$	143529.4	77.842	143591.6	143595.0	0.944	0.0433	0.0457
		$\mathcal{U}[10000, 20000]$	182359.4	4.072	182359.4	182359.4	1.790	0.0000	0.0000
	Asymmetric	$\mathcal{U}[100, 200]$	130457.0	453.694	130494.8	130498.4	1.616	0.0290	0.0317
		$\mathcal{U}[1000, 2000]$	143092.0	47.972	143182.0	143118.0	0.648	0.0629	0.0182
		$\mathcal{U}[10000, 20000]$	181556.2	3.944	181556.2	181556.2	0.264	0.0000	0.0000

Table 6: Relative performance of CLM and TS on large sized UFLP instances

Problem Details			Execution	Solution Costs			Better solution from		
m (= n)	C matrix	f_i chosen from	time (CPU Sec.)	TS	CLM	Gap (%)	TS	equal	CLM
250	Symmetric	$\mathcal{U}[100, 200]$	18.256	257832.6	257895.2	0.0243	4	0	1
		$\mathcal{U}[1000, 2000]$	6.470	276185.2	276352.2	0.0604	3	0	2
		$\mathcal{U}[10000, 20000]$	17.322	333820.0	333671.6	0.0445	0	4	1
	Asymmetric	$\mathcal{U}[100, 200]$	18.060	257978.4	258032.6	0.0210	5	0	0
		$\mathcal{U}[1000, 2000]$	6.402	276467.2	276184.2	0.1025	1	1	3
		$\mathcal{U}[10000, 20000]$	24.730	333243.6	333058.4	0.0556	0	4	1
500	Symmetric	$\mathcal{U}[100, 200]$	213.316	511383.6	511487.2	0.0203	4	0	1
		$\mathcal{U}[1000, 2000]$	71.394	538480.4	538685.8	0.0381	4	0	1
		$\mathcal{U}[10000, 20000]$	146.482	621107.2	621172.8	0.0106	1	4	0
	Asymmetric	$\mathcal{U}[100, 200]$	207.070	511251.6	511393.4	0.0277	5	0	0
		$\mathcal{U}[1000, 2000]$	79.192	538144.0	538421.0	0.0514	4	1	0
		$\mathcal{U}[10000, 20000]$	134.576	621881.8	621990.8	0.0175	3	0	1
750	Symmetric	$\mathcal{U}[100, 200]$	824.288	763831.2	763978.0	0.0192	5	0	0
		$\mathcal{U}[1000, 2000]$	409.372	796919.0	797173.4	0.0319	4	1	0
		$\mathcal{U}[10000, 20000]$	347.414	901158.4	900785.2	0.0414	0	2	3
	Asymmetric	$\mathcal{U}[100, 200]$	843.206	763840.4	764019.2	0.0234	5	0	0
		$\mathcal{U}[1000, 2000]$	395.958	796859.0	796754.2	0.0132	2	2	1
		$\mathcal{U}[10000, 20000]$	499.738	900514.2	900349.8	0.0183	0	4	1

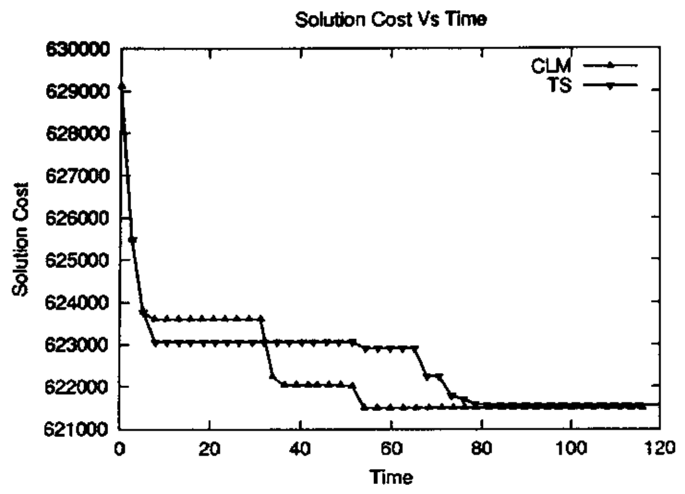

Figure 4: Variation of the cost of the best solution observed with time

et al. [19]. We decided to use tabu lists to implement short term memory structures, and chose to fix the tabu tenures randomly. Our preliminary computations with this implementation however showed a surprising result. We saw that setting a positive tabu tenure actually worsened the quality of solutions returned by tabu search. In our implementation we therefore ignored short term memory structures. Our implementation also included a long term memory structure to help diversification, in which we penalized moves involving sites that have been components of large numbers of previous moves, and an aspiration criterion that overrides the tabu status of a move if it results in a solution better than any obtained thus far. Our complete local search with memory implementation closely follows the implementation in Ghosh and Sierksma [8]. However the postprocessing operation performed local search on a few 'elite' solutions in LIVE rather than on all of them.

We used our tabu search and complete local search with memory implementations to solve 180 instances of the uncapacitated facility location problem with sizes varying from 75 × 75 to 750 × 750, covering a wide variation in fixed and transportation costs. Our results indicate that for instances with size not more than 125 × 125, both tabu search and complete local search with memory perform very well. They return solutions with costs within 0.075% of the optimal, often within one hundredth of the time required by exact algorithms. For larger instances, the solutions returned by our tabu search and complete local search with memory implementations are of very similar costs (being, on an average, within 0.0345% of each other). However, our experimentation shows that tabu search returns marginally better solutions on an average. The execution times of both the implementations increase rapidly with increasing problem size. In general, high quality solutions are often found quite early in the search process.

In summary, both tabu search and complete local search with memory offer effective ways of generating high quality solutions for large instances of uncapacitated facility location problems. Further research needs to be done to design more efficient tabu search and complete local search with memory implementations that would help to solve larger sized problems.

References

[1] J.E. Beasley, Lagrangian heuristics for location problems, European Journal of Operational Research 65 (1993)383–399.

[2] J.E. Beasley, OR-Library, `http://mscmga.ms.ic.ac.uk/info.html`

[3] V.L. Beresnev, E.Kh. Gimadi, V.T. Dementyev, Extremal Standardization Problems, Novosibirsk, Nauka, 1978 (in Russian).

[4] V.P. Cherenin, Solving some combinatorial problems of optimal planning by the method of successive calculations, Proceedings of the Conference on Experiences and Perspectives of the Applied Mathematical Methods and Electronic Computer Planning, Novosibirsk, Russia, 1962 (in Russian).

[5] N. Christofides, Graph Theory: An Algorithmic Approach, Academic Press Inc. Ltd., London, 1975.

[6] G. Cornuejols, G.L. Nemhauser, L.A. Wolsey, The uncapacitated facility location problem, in: P.B. Mirchandani and R.L. Francis (Eds.), Discrete Location Theory, Wiley-Interscience, New York,1990, pp. 119–171.

[7] D. Ghosh, N. Chakravarti, A competitive local search heuristic for the subset sum problem, Computers & Operations Research 26 (1999)271–279.

[8] D. Ghosh, G. Sierksma, Complete local search with memory, Forthcoming in Journal of Heuristics. (Downloadable from website `http://www.ub.rug.nl/eldoc/som/a/00a47/00a47.pdf`)

[9] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, 1997.

[10] B. Goldengorin, Requirements of Standards: Optimization Models and Algorithms, ROR, Hoogezand, The Netherlands, 1995.

[11] B. Goldengorin, G. Sierksma, G.A. Tijssen, M. Tso, The data-correcting algorithm for minimization of supermodular functions, Management Science 45 (1999)1539–1551.

[12] M. Körkel, On the exact solution of large-scale simple plant location problems, European Journal of Operational Research 39 (1989)157–173.

[13] J. Krarup, P.M. Pruzan, The simple plant location problem: A survey and synthesis, European Journal of Operational Research 12 (1983)36–81.

[14] M. Labbé, F.V. Louveaux, Location problems, in: M. Dell'Amico , F. Maffioli, S. Martello (Eds.), Annotated Bibliographies in Combinatorial Optimization, John Wiley & Sons, 1997, pp. 264–271.

[15] P.B. Mirchandani, R.L. Francis (Eds.) Discrete Location Theory, Wiley-Interscience, New York, 1990.

[16] D.W. Pentico, The assortment problem with nonlinear cost functions, Operations Research 24 (1976)1129–1142.

[17] D.W. Pentico, The discrete two-dimensional assortment problem, Operations Research 36 (1988)324–332.

[18] C.S. Revelle, G. Laporte, The plant location problem: New models and research prospects, Operations Research 44 (1996)864–874.

[19] E. Rolland, D.A. Schilling, J.R. Current, An efficient tabu search procedure for the p-median problem, European Journal of Operational Research 96 (1996)329–342.

[20] D. Tuzun, L.I. Burke, A two-phase tabu search approach to the location routing problem, European Journal of Operational Research 116 (1999)87–99.