Multiple Trajectory Search for Uncapacitated Facility Location Problems

Lin-Yu Tseng*
National Chung Hsing University, Institute of Networking and Multimedia
National Chung Hsing University, Department of Computer Science and Engineering
250 Kuo Kuang Road, Taichung, Taiwan
lytseng@cs.nchu.edu.tw

Chih-Sheng Wu
National Chung Hsing University, Department of Computer Science and Engineering
250 Kuo Kuang Road, Taichung, Taiwan
tks368@ms22.hinet.net

Abstract

In this study, a novel metaheuristic called the Multiple Trajectory Search (MTS) is proposed to solve the uncapacitated facility location problem (UFLP). The Multiple Trajectory Search hybridizes a global search method (the Trajectory_Search) and a local search method (the Variable_Neighborhood_Search). The application of the Multiple Trajectory Search to the benchmarks ORLIB and GHOSH had been conducted. The performance comparison with other state-of-the-art methods reveals that the proposed method is very competitive.

1. Introduction

The success of a business heavily depends on how it locates its facilities. Therefore, location problems have been widely studied because of their importance, both in theory and in practice. Location problems can be classified into four categories: *p*-center problems, *p*-median problems, uncapacitated facility location problems, and capacitated facility location problems. In this study, we consider the uncapacitated facility location problem (UFLP). A UFLP can be described as follows.

Assume there is a set of m customers $U=\{1, 2, \ldots, m\}$ and a set of n candidate facility sites $F=\{1, 2, \ldots, n\}$. There is no limit to the number of customers a facility can serve. c_{ij} is used to represent the cost of serving customer i from facility j, and f_j is used to represent the cost of opening facility j. Furthermore, c_{ij} for $i=1, 2, \ldots, m$; $j=1, 2, \ldots, n$; and f_j for $i=1, 2, \ldots, n$ are assumed to be greater than zero. Then, a UFLP is defined as [5]:

Minimize:

$$\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{j=1}^{n} f_j p_j$$

Subject to:

$$\sum_{j=1}^{n} x_{ij} = 1 \quad \text{for} \quad i = 1,...,m \tag{1}$$

$$x_{ij} \le p_j \quad \text{for } i = 1,...,m \text{ and } j = 1,...,n \tag{2}$$

$$x_{ij} \in \{0,1\} \quad \text{for } i = 1,...,m \text{ and } j = 1,...,n \tag{3}$$

$$p_j \in \{0,1\} \quad \text{for } j = 1,...,n. \tag{4}$$

In constraint (3), $x_{ij} = 0$ indicates that facility j does not serve customer i, while $x_{ij} = 1$ indicates that facility j serves customer i. In constraint (4), $p_j = 0$ or 1 indicates that facility j is either closed or open, respectively. Constraint (1) states that each customer must be served by one and only one facility. Constraint (2) states that a facility can serve customers only if it is open. The objective of the UFLP is to minimize the sum of the customer service costs and the facility opening costs.

Since the UFLP is an NP-hard problem [5], exact algorithms generally solve only small instances. For large instances, approximation algorithms, heuristics, and metaheuristics have been proposed. Hoefer presented an experimental comparison of five state-of-the-art heuristics: JMS, an approximation algorithm [7]; MYZ, also an approximation algorithm [9]; swap-based local search [2]; tabu search [10]; and the volume algorithm [3]. Hoefer concluded that based on experimental evidence, tabu search achieves optimal solutions in a reasonable amount of time, and is therefore the method of choice for practitioners. Approximation algorithms are more theoretically oriented, but heuristics and metaheuristics often outperform them in practice. Therefore, developing heuristics or metaheuristics for the UFLP has attracted increased attention from researchers.

In recent years, several metaheuristics have been proposed for the UFLP. In general, metaheuristics use more computation time and obtain better quality solutions than heuristics do. These metaheuristics include genetic algorithms [8], tabu search [1] [6] [10][12], and path relinking [11]. In particular, the most recent hybrid multistart heuristic proposed by

978-0-7695-3605-7/09 $25.00 © 2009 IEEE
DOI 10.1109/CSO.2009.340

Resende and Werneck [11] and the tabu search proposed by Sun [12] have made significant improvements in solving the benchmark instances. The hybrid multistart heuristic [11] builds an elite set and applies path-relinking repeatedly to search for good solutions. Sun's tabu search [12] divides its search process into the short-term memory process, the medium-term memory process, and the long-term memory process. A move in a tabu search opens or closes a facility. Sun also designed an efficient method to update, rather than re-compute, the net change in the cost of a move. In spite of all these improvements, there is still room for progress.

In this paper, we present a novel method called the Multiple Trajectory Search (MTS) for the UFLP. The MTS can search the solution space systematically, and it is more powerful than tabu search with respect to diversification, and easier to control than genetic algorithms. More noteworthy is that the MTS may include a variety of neighborhood search methods that are either relevant or irrelevant to problematic properties.

2. Multiple Trajectory Search (MTS)

We now explain the MTS and depict its pseudocode in Figure 1. The MTS begins with a set of randomly generated initial solutions. A solution is represented by a sequence of n bits. The value of the k^{th} bit is 0 (1) if the k^{th} facility is closed (open).

The MTS consists of a predetermined number of runs. In the first phase of each run, the Variable_Neighborhood_Search, which is a local search method, is applied to each solution in the set S. After that, begins the second phase. The second phase consists of k iterations, in each iteration, firstly a set of foreground solutions S' is selected from S, then the Trajectory_Search is applied to each foreground solution. The solutions that are not foreground, that is, solutions that are in S-S', are called background solutions.

Next, let us describe the Variable_Neighborhood_Search, which is shown in Figure 2. This procedure conducts a local search in the neighborhood of a solution s_i. It flips j randomly chosen facilities of solution s_i, that is, changes these j facilities from closed to open or from open to closed, to obtain a new solution s, and then replaces s_i by s if s is better than s_i. This process will be repeated until the consecutive number of non-improvements reaches *maxstuck*. In this method, j ranges from 1 to 4.

```
Multiple-Trajectory-Search( )
 1. Generate Initial solution set  S = {s_1, s_2, ..., s_x}  randomly.
 2. Run←0;
 3. while Run<MaxRun
 4.       for  i←1 to x
 5.            Apply Variable_Neighborhood_Search( s_i , maxstuck1);
 6.       for  j←1 to k
 7.            Choose foreground solution  S' = {s_1', s_2', ..., s_y'}  from S
 8.                 for  i←1 to y
 9.                      Apply Trajectory_Search( s_i' );
10.       Run←Run+1.
```

Figure 1. Pseudocode for the MTS

In the second phase of the MTS, the foreground solutions are selected from the set S and the Trajectory_Search is applied to these solutions. The criteria for selecting foreground solutions are as follows:

1. The first three solutions in S that have the best objective values and are not tabu will be selected.
2. The first two solutions in S that have the largest improvement in the last iteration and are not tabu will be selected.
3. If a foreground solution has been selected for three consecutive iterations and it does not have any improvement in these three iterations, this solution will be tabu for the following t iterations.

```
Variable_Neighborhood_Search( s_i , maxstuck)
 1. for  j←1 to 4
 2.      stuck ← 0;
 3.      while stuck < maxstuck
 4.            s ← s_i ;
 5.            Randomly choose j facilities from s and flip them.
 6.            if  s is better than  s_i
 7.                then  s_i ← s ;
 8.                      stuck ← 0;
 9.                else  stuck ← stuck+1;
```

Figure 2. Pseudocode for the Variable_Neighborhood_Search

Next, let us explain the Trajectory_Search. Its pseudocode is shown in Figure 3. In the UFLP, a solution is represented by a string of n bits with each bit representing the closed or open state of a facility. The Trajectory_Search(s') consists of k_1 iterations. In each iteration, it randomly chooses n/d bit positions first, then it randomly generates z solutions s_1'', s_2'', …, s_z'', each s_j'' differs from the given solution s' on exactly r bits, and the positions of these r bits are among the chosen n/d bit positions. For each s_j'', the Variable_Neighborhood_Search will be applied to search its neighborhood and the replacement of the given solution s' by s_j'' will be made if these is an improvement. The Trajcetory_Search is in fact a large-neighborhood search procedure. It searches a large-

neighborhood of the solution s', and moves to the new solution s'' if it finds that s'' is better than s' in its search process.

```
Trajectory_Search( s′ )
 1. for i←1 to k_1
 2.     Randomly choose n/d bit positions from n bit positions.
 3.     Randomly generate z solutions s″_1, s″_2, ..., s″_z with
            each solution s″_j differing from s′ on exactly r
            bit positions and these r bit positions are among
            the chosen n/d bit positions.
 4.     for j←1 to z
 5.         Apply Variable_Nieghborhood_Search( s″_j , maxstuck2) ;
 6.         if s″_j is better than s′
 7.             then s′ ← s″_j
```

Figure 3. Pseudocode for the Trajectory_Search

3. Experimental Results

Experiments had been conducted to evaluate the performance of the proposed MTS. A personal computer with Intel Core 2 Duo 2.33 GHz CPU was used as the platform to run the program. Although there were two processors, only one processor was used. The program was implemented using C++. In this section, experimental results are given and compared with those of other methods.

As described in Section 2, the MTS requires eleven parameters, and the default parameter values are x = 40, y = 5, z = 10, *MaxRun* = 2, *maxstuck1* = 10, *maxstuck2* = n/2, k = 4, k_1 = 4, d = 6, t = 3, and r is randomly chosen from {1, 2, …, n/d}. The setting of these parameter values are based on the results of preliminary computational studies.

Initially, the MTS was tested on the benchmark of the OR Library. This benchmark was proposed by Beasley [4] and posted in the OR-Library. It consists of 15 problems that are divided into four classes. The performance of the MTS is compared with that of the simple tabu search (STS) [10] on the OR-Library benchmark. This comparison is listed in Table 1. The computer used by the simple tabu search is the Pentium IV 2GHz running a Linux operating system. In this experiment, the MTS and the simple tabu search were run 100 times on each problem, and the number of optimum findings (Opt Hit), the best deviation from the optimum (Best_D), the average deviation from the optimum (Avg_D), the worst deviation from the optimum (Worst_D), and the average CPU time (Avg_T) were taken. The MTS can find all the optimal solutions at each of 100 runs for all problems, but there are 4 problems on which the simple tabu search cannot hit all optimal solutions at all 100 runs. The CPU time needed by the MTS is comparable to that needed by the simple tabu search. From Table 1, it can be observed that the MTS outperforms the simple tabu search in terms of solution quality.

The performance of the MTS is also compared with those of Ghosh's method [6], the hybrid multistart heuristic [11], and Sun's tabu search [12] on the GHOSH benchmark [6]. There are in total 90 problems in this benchmark that are divided according to size and type into 18 sets with five problems in each set. The performance comparison is shown in Table 2, within which the previously best known solutions are taken from [12]. Both the data of the hybrid multistart heuristic [11] and the data of the MTS were the average of 50 runs. It is noted in Table 2 that Ghosh's method achieves one (out of eighteen) of the previously best known solution, the hybrid multistart heuristic achieves four (out of eighteen) previously best known solutions, Sun's tabu search achieves sixteen (out of eighteen) previously best known solutions. The MTS achieves thirteen (out of eighteen) previous best known solutions. In addition to this, the MTS found solutions to the other two sets that are better than the previously best known solutions (the solutions printed in italics in Table 2). The last row of Table 2 lists the average values over the eighteen sets; and from the average values, it is noted that the MTS outperforms Ghosh's method in both solution quality and computation time, the MTS outperforms the hybrid multistart heuristic in solution quality and the MTS outperforms the Tabu Search in computation time.

4. Conclusion

In this paper, the multiple trajectory search (MTS) is proposed to solve the UFLP. The MTS utilizes two procedures: the Trajectory Search and the Variable Neighborhood Search. The Trajectory Search is a global search method that aims to search the whole solution space systematically. The Variable Neighborhood Search is a local search method that aims to search the neighborhood of a solution thoroughly. Experimental results reveal that the MTS outperforms other state-of-the-art heuristics on benchmarks ORLIB and CHOSH. However, the local search ability of the Variable Neighborhood Search is still not good enough for hard problems (for example, the class A of the GHOSH), and this is because that the Variable Neighborhood Search is a general local search method. Therefore in future studies, we will try to design local search methods which utilize good properties specific to the problem properties.

Table 1. Performance comparison of the MTS and the simple tabu search on benchmark ORLIB

			Simple Tabu Search (STS)					Multiple Trajectory Search (MTS)				
Name	Size	OPT	Opt Hit	Best_D	Avg_D	Worst_D	AvgT	Opt Hit	Best_D	Avg_D	Worst_D	AvgT
cap71	16X50	932615.75	100	0	0	0	0.05	100	0	0	0	0.033
cap72	16X50	977799.40	100	0	0	0	0.05	100	0	0	0	0.037
cap73	16X50	1010641.45	100	0	0	0	0.07	100	0	0	0	0.046
cap74	16X50	1034976.97	100	0	0	0	0.07	100	0	0	0	0.046
cap101	25X50	796648.44	80	0	0.022	0.108	0.07	100	0	0	0	0.054
cap102	25X50	854704.20	100	0	0	0	0.06	100	0	0	0	0.060
cap103	25X50	893782.11	94	0	0.002	0.025	0.08	100	0	0	0	0.065
cap104	25X50	928941.75	100	0	0	0	0.08	100	0	0	0	0.071
cap131	50X50	793439.56	84	0	0.017	0.108	0.10	100	0	0	0	0.124
cap132	50X50	851495.32	100	0	0	0	0.09	100	0	0	0	0.132
cap133	50X50	893076.71	96	0	0.003	0.079	0.12	100	0	0	0	0.139
cap134	50X50	928941.75	100	0	0	0	0.13	100	0	0	0	0.144
	Average			0.00	0.003	0.247	0.080		0.00	0.00	0.00	0.078

Table 2. Performance comparison of the MTS and three other methods on benchmark GHOSH

	Instance		Previous	Ghosh[a][6]		Hybrid[a][11]		Tabu[b][12]		MTS[c]	
Class	Size	Type	best know	Best	Time	Best	Time	Best	Time	Best	Time
A	250	Sym	257805.0	257832.6	18.26	257807.9	5.3	257805.0	2.83	***257804.0***	5.59
		Asym	257917.8	257978.4	18.06	257922.1	5.7	**257917.8**	2.62	**257917.8**	5.62
A	500	Sym	511180.4	511383.6	213.32	511203.0	43.5	**511180.4**	15.62	511188.8	27.40
		Asym	511140.0	511251.6	207.07	511147.4	40.3	511140.0	13.76	***511137.8***	27.43
A	750	Sym	763693.4	763831.2	824.29	763713.9	112.6	**763693.4**	39.81	763708.8	67.63
		Asym	763717.0	763840.4	843.21	763741.0	117.5	**763717.0**	39.65	763735.8	67.91
B	250	Sym	276035.2	276185.2	6.47	**276035.2**	8.0	**276035.2**	5.63	**276035.2**	7.37
		Asym	276053.2	276184.2	6.40	276053.6	8.2	**276053.2**	5.79	**276053.2**	7.33
B	500	Sym	537912.0	538480.4	71.39	537919.1	52.6	**537912.0**	31.43	**537912.0**	32.70
		Asym	537847.6	538144.0	79.19	537868.2	52.2	**537847.6**	34.75	**537847.6**	32.91
B	750	Sym	796571.8	796919.0	409.37	796593.7	126.3	**796571.8**	93.35	**796571.8**	77.67
		Asym	796374.4	796754.2	395.96	796393.5	127.1	**796374.4**	95.43	**796374.4**	78.06
C	250	Sym	333671.6	**333671.6**	17.32	**333671.6**	8.3	**333671.6**	9.88	**333671.6**	8.49
		Asym	332897.2	333058.4	24.73	**332897.2**	7.4	**332897.2**	9.20	**333897.2**	8.51
C	500	Sym	621059.2	621107.2	146.48	**621059.2**	50.8	**621059.2**	71.11	**621059.2**	36.72
		Asym	621463.8	621881.8	134.58	621475.2	57.4	**621463.8**	72.06	**621463.8**	36.61
C	750	Sym	900158.6	900785.2	347.41	900183.8	130.3	**900158.6**	229.91	**900158.6**	88.17
		Asym	900193.2	900349.8	499.74	900198.6	136.5	**900193.2**	236.90	**900193.2**	87.17
	Average			555535.5	236.85	555493.6	60.56	555316.2	56.09	555318.4	39.07

a: Intel Mobile Celeron 650MHz, C++; b: SUN Enterprise 3000m, Fortran; c: Intel core2 Duo 2.33 GHz, C++

References

[1] K.S. Al-Sultan and M.A. Al-Fawzan, A tabu search approach to the uncapacitated facility location problem, *Annals of Operations Research*, 86**:** 91-103, 1999.

[2] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala and V. Pandit, Local Search heuristics for k-median and facility location problems, *ACM Symposium on Theory of Computing*, 21-29, 2001.

[3] F. Barahona and F. Chudak, Near-optimal solutions to large scale facility location problems, Technical Report RC21606, IBM, Yorktown Heights, NY, USA, 1999.

[4] J.E. Beasley, OR-Library: Distributing test problems by electronic mail, *Journal of the Operational Research Society*, 41**:** 1069-1072, 1990. Available: http://mscmga.ms.ic.ac.uk/info.html.

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

[6] D. Ghosh, Neighborhood search heuristics for the uncapacitated facility location problem, *European Journal of Operational Research*, 150**:** 150-162, 2003.

[7] K. Jain, M. Mahdian and A. Saberi, A new greedy approach for facility location problems, in: *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, ACM Press, 731-740, 2002.

[8] J. Kratica, D. Tosic, V. Fillipovic and I. Ljubic, Solving the simple plant location problem by genetic algorithm, *RAIRO Operations Research*, 35**:** 127-142, 2001.

[9] M. Mahdian, Y. Ye, and J. Zhang, Improved approximation algorithms for metric facility location problems, in: *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, volume 2462 of Lecture Notes in Computer Science, Springer-Verlag, 229-242, 2002.

[10] L. Michel and P. Van Hentenryck, A simple tabu search for warehouse location, *European Journal of Operational Research*, 157**:** 576-591, 2003.

[11] M.G.C. Resende and R.F. Werneck, A hybrid multistart heuristic for the uncapacitated facility location problem, *European Journal of Operational Research*, 174**:** 54-68, 2006.

[12] M. Sun, Solving the uncapacitated facility location problem using tabu search, *Computers & Operations Research*, 33**:** 2563-2589, 2006.