Deterministic and Randomized Heuristic Algorithms for Uncapacitated Facility Location Problem

Soumen Atta, Priya Ranjan Sinha Mahapatra and Anirban Mukhopadhyay

Abstract A well-known combinatorial optimization problem, known as the Uncapacitated Facility Location Problem (UFLP) is considered in this paper. Given a set of customers and a set of potential facilities, the objective of UFLP is to open a subset of the potential facilities such that sum of the opening cost for opened facilities and the service cost of customers is minimized. In this paper, deterministic and randomized heuristic algorithms are presented to solve UFLP. The effectivenesses of the proposed algorithms are tested on UFLP instances taken from the OR-Library. Although the proposed deterministic algorithm gives optimal results for most of the instances, the randomized algorithm achieves optimal results for all the instances of UFLP considered in this paper including those for which the deterministic algorithm fails to achieve the optimal solutions.

Keywords Uncapacitated facility location problem (UFLP) · Simple plant location problem (SPLP) · Warehouse location problem (WLP) · Heuristics Randomization

1 Introduction

The *uncapacitated facility location problem* (*UFLP*) is the problem of finding the optimal placement of facilities of unrestricted capacities among n potential facility locations such that the cost of satisfying demands of all the customers is minimized [1–5]. Here, the cost is of two types: the *service* or *connection cost* to

S. Atta (✉) · P. R. S. Mahapatra · A. Mukhopadhyay
Department of Computer Science and Engineering, University of Kalyani,
Nadia, Kalyani 741235, West Bengal, India
e-mail: soumen.atta@klyuniv.ac.in

P. R. S. Mahapatra
e-mail: priya@klyuniv.ac.in

A. Mukhopadhyay
e-mail: anirban@klyuniv.ac.in

S. C. Satapathy et al. (eds.), *Information and Decision Sciences*,
Advances in Intelligent Systems and Computing 701,
https://doi.org/10.1007/978-981-10-7563-6_22

provide service to a customer by a facility and the *opening cost* to open a facility. UFLP is also known as the *Simple Plant Location Problem* (SPLP) [1, 6] and the *Warehouse Location Problem* (*WLP*) [7]. UFLP is known to be an NP-hard problem [8, 9]. So, different heuristic approaches are used to solve this problem to obtain near-optimal solution. Some of the approaches are branch-and-bound algorithm [10, 11], tabu search [4, 5], constant factor approximation algorithm [12], greedy heuristic [13], neighborhood search [14], hybrid multi-start heuristic [15], semi-Lagrangian relaxation [16], message-passing [17], surrogate semi-Lagrangian dual [18], discrete unconscious search [19], etc.

In this paper, two heuristic algorithms are proposed. We call these two algorithms as the *deterministic BFR* and the *randomized BFR*. Here, *BFR* is the acronym for *backward–forward–replacement phase*. As the name suggests the first algorithm is deterministic in nature, i.e., it always produces same output for a particular input data set instance. The output of the second algorithm depends on random behavior of some steps. Both the algorithms consist of three phases. These phases are *forward phase*, *backward phase*, and *replacement phase*. The detailed description of these phases is given in Sect. 3. The effectiveness of these two algorithms is tested on UFLP instances of different sizes taken from the literature.

The organization of the rest of the paper is as follows: Sect. 2 formally defines the problem. The proposed algorithms are described in Sect. 3. Computational results are reported and compared in Sect. 4. Finally, Sect. 5 concludes the paper.

2 Problem Definition

The uncapacitated facility location problem (UFLP) [1, 3–5] can be stated as follows: A set $J = \{j_1, j_2, \ldots, j_M\}$ of M customers or cities and a set $I = \{i_1, i_2, \ldots, i_N\}$ of N potential facility locations (sites) are given. A nonnegative opening cost f_i associated with each facility location and a nonnegative service or connection cost c_{ij} between facility i and each customer or city j are also given. The objective of UFLP is to connect each customer or city to the nearest opened facility such that the *total cost*, i.e., the sum of service or connection cost and opening cost of opened facilities is minimized. It is worthy to note that the demand of any customer is fulfilled by only one facility and hence the capacity of each facility is assumed to be infinite.

Using the above descriptions of variables, the mathematical formulation of UFLP [4] is as follows:

$$minimize \sum_{i=1}^{N} \sum_{j=1}^{M} c_{ij} x_{ij} + \sum_{i=1}^{N} f_i y_i$$

subject to

$$\sum_{i=1}^{N} x_{ij} = 1, \quad j = 1, \ldots M,$$

$$x_{ij} \leq y_j, \quad i = 1, \ldots N, \; j = 1, \ldots M,$$

$$x_{ij}, \; y_i = \{0, 1\}, \quad i = 1, \ldots n, \; j = 1, \ldots M,$$

where

$$x_{ij} = \begin{cases} 1, & \text{if customer } j \in J \text{ is served from site } i \in I \\ 0, & \text{otherwise;} \end{cases}$$

$$y_i = \begin{cases} 1, & \text{if a facility is established at location } i \in I \\ 0, & \text{otherwise.} \end{cases}$$

3 Proposed Heuristic Algorithms

Each of the proposed algorithms, viz., deterministic BFR and randomized BFR consists of forward phase, backward phase, and replacement phase. So, these phases are described in details in the following Sects. 3.1, 3.2, and 3.3, respectively, before the two proposed heuristic algorithms. Each of these phases takes two data structures, viz., cost matrix and a set of opened facilities as its inputs. Throughout the paper, the cost matrix, the set of opened facilities, and the set of non-opened facilities are denoted by C, $\mathcal{F}$ and $\overline{\mathcal{F}}$, respectively. The cost matrix C is a matrix of order $N \times (M + 1)$ where (*i*) $C(i, j)$ denotes the service cost between the *i*th facility location to the *j*th customer, $1 \leq i \leq N$, $1 \leq j \leq M$ and (*ii*) $C(i, M + 1)$, $1 \leq i \leq N$ denotes the opening cost of the *i*th facility.

In the proposed algorithms, a function, named as *TotalCost*, is used to compute the total cost. This function takes C and $\mathcal{F}$ as its inputs and gives the corresponding total cost. The time complexity of this function is $\mathcal{O}(N|\mathcal{F}|)$.

3.1 Forward Phase

In this phase, new facilities are opened if and only if the total cost is reduced. The *if* block from line 4 to 7 of Algorithm 1 is executed only when the existing set of opened facilities, $\mathcal{F}_e$ is empty. The sum of each row of C is computed and then these sums are sorted in line 5 to find the *index* of each facility. The first facility according to this sorted index is assigned to $\mathcal{F}_e$ in line 6. So, after the execution of the *if* block from lines 4 to 7, the cardinality of $\mathcal{F}_e$ must be at least one. The *while* loop in line 8 is executed at least once to add a new facility, if possible, in $\mathcal{F}_e$ and the execution of this loop stops when there is no improvement in terms of total cost by adding a new facility in $\mathcal{F}_e$. So, this *while* loop in line 8 may be executed at most $(N - |\mathcal{F}_e|)$-times. For each $i \in \overline{\mathcal{F}_e}$, the total cost corresponding to the set $\mathcal{F}_e \cup \{i\}$ is computed and the *i*th facility for which the total cost is minimum is opened if the total cost is

reduced. The *for* loop in line 11 is executed $(N - |\mathcal{F}_e|)$-times. At the end of the *while* loop in line 27, $\mathcal{F}_e$ is assigned to the new set of opened facilities, $\mathcal{F}_n$ in line 28. The pseudocode of forward phase is given in Algorithm 1 and the time required for the execution of each statement is mentioned there.

Algorithm 1: Forward Phase

Input: Cost matrix $\mathcal{C}$, existing set of opened facilities $\mathcal{F}_e$
Output: New set of opened facilities $\mathcal{F}_n$

1 $N \leftarrow$ no. of facilities; // $\mathcal{O}(1)$
2 $\mathcal{F}_n \leftarrow [\,]$; // $\mathcal{O}(1)$
3 *improve* $\leftarrow$ *True*; // $\mathcal{O}(1)$
4 **if** $\mathcal{F}_e$ *is empty* **then**
5 Compute the sum of each row of $\mathcal{C}$ and sort these sums in ascending order to find sorted *index* of each facility; // $\mathcal{O}(N \log N)$
6 $\mathcal{F}_e \leftarrow$ *index*(1); // $\mathcal{O}(1)$
7 **end**
8 **while** *improve* = *True* **do**
`// This while loop may iterate at most` $(N - |\mathcal{F}_e|)$`-times.`
9 *old_tcost* $\leftarrow$ *TotalCost*$(\mathcal{C}, \mathcal{F}_e)$; // $\mathcal{O}(N|\mathcal{F}_e|)$
10 *flag* $\leftarrow 0$; // $\mathcal{O}(1)$
11 **for** *each* $i \in \overline{\mathcal{F}}_e$ **do**
`// This for loop is iterated` $(N - |\mathcal{F}_e|)$`-times.`
12 $\mathcal{F}_{temp} \leftarrow [\,]$; // $\mathcal{O}(1)$
13 $\mathcal{F}_{temp} \leftarrow \mathcal{F}_e \cup \{i\}$; // $\mathcal{O}(1)$
14 *temp_tcost* $\leftarrow$ *TotalCost*$(\mathcal{C}, \mathcal{F}_{temp})$; // $\mathcal{O}(N|\mathcal{F}_{temp}|)$
15 **if** *temp_tcost* $<$ *old_tcost* **then**
16 *min_cost_f* $\leftarrow i$; // $\mathcal{O}(1)$
17 *old_tcost* $\leftarrow$ *temp_tcost*; // $\mathcal{O}(1)$
18 *flag* $\leftarrow 1$; // $\mathcal{O}(1)$
19 **end**
20 **end**
21 **if** *flag* $= 1$ **then**
22 $\mathcal{F}_e \leftarrow \mathcal{F}_e \cup \{min\_cost\_f\}$; // $\mathcal{O}(1)$
23 **else**
24 *improve* $\leftarrow$ *False*; // $\mathcal{O}(1)$
25 **end**
26 *new_tcost* $\leftarrow$ *TotalCost*$(\mathcal{C}, \mathcal{F}_e)$; // $\mathcal{O}(N|\mathcal{F}_e|)$
27 **end**
28 $\mathcal{F}_n \leftarrow \mathcal{F}_e$; // $\mathcal{O}(|\mathcal{F}_e|)$
29 **return** $\mathcal{F}_n$

3.2 Backward Phase

In this phase, the facilities are closed from the already opened facilities if and only if the total cost is reduced. The *if* condition in line 4 of Algorithm 2 checks the cardinality of $\mathcal{F}_e$. If the set $\mathcal{F}_e$ is empty or its cardinality is 1 then $\mathcal{F}_n$ is assigned as $\mathcal{F}_e$ in line 5 and the algorithm is terminated. If the set $\mathcal{F}_e$ contains more than one facility then the algorithm performs the following steps to close one facility at a time. The *while* loop in line 7 may iterate at most $(|\mathcal{F}_e| - 1)$-times and the execution of this loop stops when there is no improvement in terms of total cost by deleting a facility from $\mathcal{F}_e$. For each $i \in \mathcal{F}_e$, the total cost corresponding to the set $\mathcal{F}_e \backslash \{i\}$ is computed in line 13 and the ith facility for which the total cost is minimum is closed provided that it reduces the total cost. These steps are repeated for closing one facility at a

Algorithm 2: Backward Phase

Input: Cost matrix C, existing set of opened facilities $\mathcal{F}_e$
Output: New set of opened facilities $\mathcal{F}_n$

1 $N \leftarrow$ no. of facilities; // $\mathcal{O}(1)$
2 $\mathcal{F}_n \leftarrow [\,]$; // $\mathcal{O}(1)$
3 *improve* $\leftarrow$ *True*; // $\mathcal{O}(1)$
4 **if** $\mathcal{F}_e$ *is empty* ***or*** $|\mathcal{F}_e| = 1$ **then**
5 $\quad \mathcal{F}_n \leftarrow \mathcal{F}_e$; // $\mathcal{O}(|\mathcal{F}_e|)$
6 **else**
7 $\quad$ **while** *improve* = *True* **do**
$\quad\quad$ // This *while loop* may iterate at most $(|\mathcal{F}_e| - 1)$-times.
8 $\quad\quad$ *old_tcost* $\leftarrow$ *TotalCost*$(C, \mathcal{F}_e)$; // $\mathcal{O}(N|\mathcal{F}_e|)$
9 $\quad\quad$ *flag* $\leftarrow 0$; // $\mathcal{O}(1)$
10 $\quad\quad$ **for** *each* $i \in \mathcal{F}_e$ **do**
$\quad\quad\quad$ // This *for loop* is iterated $|\mathcal{F}_e|$-times.
11 $\quad\quad\quad$ $\mathcal{F}_{temp} \leftarrow [\,]$; // $\mathcal{O}(1)$
12 $\quad\quad\quad$ $\mathcal{F}_{temp} \leftarrow \mathcal{F}_e \backslash \{i\}$; // $\mathcal{O}(1)$
13 $\quad\quad\quad$ *temp_tcost* $\leftarrow$ *TotalCost*$(C, \mathcal{F}_{temp})$; // $\mathcal{O}(N|\mathcal{F}_{temp}|)$
14 $\quad\quad\quad$ **if** *temp_tcost* < *old_tcost* **then**
15 $\quad\quad\quad\quad$ *min_cost_f* $\leftarrow i$; // $\mathcal{O}(1)$
16 $\quad\quad\quad\quad$ *old_tcost* $\leftarrow$ *temp_tcost*; // $\mathcal{O}(1)$
17 $\quad\quad\quad\quad$ *flag* $\leftarrow 1$; // $\mathcal{O}(1)$
18 $\quad\quad\quad$ **end**
19 $\quad\quad$ **end**
20 $\quad\quad$ **if** *flag* = 1 **then**
21 $\quad\quad\quad$ $\mathcal{F}_e \leftarrow \mathcal{F}_e \backslash \{min\_cost\_f\}$; // $\mathcal{O}(1)$
22 $\quad\quad$ **else**
23 $\quad\quad\quad$ *improve* $\leftarrow$ *False*; // $\mathcal{O}(1)$
24 $\quad\quad$ **end**
25 $\quad\quad$ *new_tcost* $\leftarrow$ *TotalCost*$(C, \mathcal{F}_e)$; // $\mathcal{O}(N|\mathcal{F}_e|)$
26 $\quad$ **end**
27 $\quad \mathcal{F}_n \leftarrow \mathcal{F}_e$; // $\mathcal{O}(|\mathcal{F}_e|)$
28 **end**
29 **return** $\mathcal{F}_n$

time as long as the total cost is reduced. The *for* loop in line 10 is iterated $|\mathcal{F}_e|$-times. The pseudocode of backward phase is given in Algorithm 2 and the time required for the execution of each statement is mentioned there.

3.3 Replacement Phase

The objective of replacement phase is to check whether it is possible to replace already opened facilities in $\mathcal{F}_e$ with non-opened facilities in $\overline{\mathcal{F}}_e$ to reduce the total cost without changing the number of opened facilities. The algorithm performs the following steps to replace the opened facility j in $\mathcal{F}_e$ with a non-opened facility i in $\overline{\mathcal{F}}_e$. For each $i \in \overline{\mathcal{F}}_e$, the total cost for each of the sets $(\mathcal{F}_e \backslash \{j\}) \cup \{i\}$ is computed and the ith facility for which the total cost is minimum is opened and the facility j is closed if it improves the total cost. These steps are repeated as long as improvement occurs in terms of the total cost. The pseudocode of replacement phase is given in Algorithm 3 and the time required for the execution of each statement is mentioned there.

Algorithm 3: Replacement Phase

Input: Cost matrix $\mathcal{C}$, existing set of opened facilities $\mathcal{F}_e$
Output: New set of opened facilities $\mathcal{F}_n$

1 **for** *each* $j \in \mathcal{F}_e$ **do**
  // This *for loop* is iterated $|\mathcal{F}_e|$-times.
2 $old\_tcost \leftarrow TotalCost(\mathcal{C}, \mathcal{F}_e)$; // $\mathcal{O}(N|\mathcal{F}_e|)$
3 $flag \leftarrow 0$; // $\mathcal{O}(1)$
4 **for** *each* $i \in \overline{\mathcal{F}}_e$ **do**
    // This *for loop* is iterated $(N - |\mathcal{F}_e|)$-times.
5 $\mathcal{F}_{temp} \leftarrow [\,]$; // $\mathcal{O}(1)$
6 $\mathcal{F}_{temp} \leftarrow (\mathcal{F}_e \backslash \{j\}) \cup \{i\}$; // $\mathcal{O}(1)$
7 $temp\_tcost \leftarrow TotalCost(\mathcal{C}, \mathcal{F}_{temp})$; // $\mathcal{O}(N|\mathcal{F}_{temp}|)$
8 **if** $temp\_tcost < old\_tcost$ **then**
9 $min\_cost\_f \leftarrow i$; // $\mathcal{O}(1)$
10 $old\_tcost \leftarrow temp\_tcost$; // $\mathcal{O}(1)$
11 $flag \leftarrow 1$; // $\mathcal{O}(1)$
12 **end**
13 **end**
14 **if** $flag = 1$ **then**
15 $\mathcal{F}_e \leftarrow (\mathcal{F}_e \backslash \{j\}) \cup \{min\_cost\_f\}$; // $\mathcal{O}(1)$
16 **end**
17 **end**
18 $\mathcal{F}_n \leftarrow \mathcal{F}_e$; // $\mathcal{O}(|\mathcal{F}_e|)$
19 **return** $\mathcal{F}_n$

3.4 Deterministic BFR

The deterministic BFR (i.e., Algorithm 4) takes the cost matrix C as its input and gives the set of opened facilities $\mathcal{F}$ and the corresponding total cost as its outputs. Algorithm 4 starts with opening all the facilities as shown in line 2. Then, the following steps are repeated to reduce the total cost. In line 6, the backward phase is used to close some opened facilities (if possible) and this is followed by the replacement phase in line 7 that may replace some opened facilities. Then, the forward phase is used in line 8 to open new facilities (if possible) which is again followed by the replacement phase in line 9. The steps at lines 6 to 9 are repeated as long as the total cost improves.

Algorithm 4: Deterministic BFR

Input: Cost matrix C
Output: Set of opened facilities $\mathcal{F}$, total cost $\mathcal{T}$

```
1  n ← no. of potential facilities;
2  F ← {1, 2, ..., n};
3  old_tcost = TotalCost(C, F);
4  improve ← True;
5  while improve = True do
6  |   F = BackwardPhase(C, F);
7  |   F = ReplacementPhase(C, F);
8  |   F = ForwardPhase(C, F);
9  |   F = ReplacementPhase(C, F);
10 |   new_tcost = TotalCost(C, F);
11 |   if new_tcost < old_tcost then
12 |   |   old_tcost ← new_tcost;
13 |   else
14 |   |   improve ← false;
15 |   end
16 end
17 T ← TotalCost(C, F);
18 return F, T
```

3.5 Randomized BFR

It is likely that the deterministic BFR may stuck into a local optima. The randomized BFR tries to overcome this problem by the help of randomness. The randomized BFR given in Algorithm 5 also takes the cost matrix C as its input and gives the set $\mathcal{F}$ and the corresponding total cost as its outputs. The deterministic BFR is called in line 2 to

produce a solution $\mathcal{F}_1$. In line 3, the solutions $\mathcal{F}_2$, $\mathcal{F}_3$, and $\mathcal{F}_4$ are generated randomly from the solution $\mathcal{F}_1$ by arbitrarily changing the elements in $\mathcal{F}_1$ by the elements in $\overline{\mathcal{F}}_1$ keeping the cardinality same as $\mathcal{F}_1$. We now modify each of the solutions $\mathcal{F}_i$, $1 \leq i \leq 4$, in the following way. Each opened facility in $\mathcal{F}_i$ is swapped with a randomly chosen non-opened facility in $\overline{\mathcal{F}}_i$ with probability 0.5. If any swapping occurs at all then both the sets $\mathcal{F}_i$ and $\overline{\mathcal{F}}_i$ are modified. At the end of the *for* loop in line 13, the total cost for each $\mathcal{F}_i$ is computed in line 14 and the solution with the maximum total cost is replaced by the solution with the minimum total cost of the previous iteration. Here, the variables *iterate* and *count* are used to terminate Algorithm 5. The value of *count* is incremented by one if the minimum total cost for two consecutive iterations remains same, otherwise it is set to zero. Algorithm 5 terminates if the value of either *iterate* or *count* exceeds *max_iterate* or *max_count* respectively.

Algorithm 5: Randomized BFR

```
   Input: Cost matrix C
   Output: Set of opened facilities F, total cost T
1  n ← no. of potential facilities;
2  Create a set of opened facilities F_1 using the deterministic BFR (Algorithm 4);
3  Create other three solutions F_2, F_3, F_4 randomly using F_1;
4  iterate ← 1, count ← 1;
5  while iterate ≤ max_iterate and count ≤ max_count do
6      Find the total cost for each of the solutions F_1, ..., F_4;
7      Let F_best be the solution with the minimum total cost;
8      for i ← 1 to 4 do
9          r ← a random number in [0, 1];
10         if r ≥ 0.5 then
11             Swap each of the opened facilities in F_i with a randomly chosen non-opened
               facility in F̄_i;
12         end
13     end
14     Find the total cost for each of the solutions F_1 ... F_4;
15     Replace the solution having the maximum total cost value with F_best;
16     if iterate ≠ 1 then
17         if the minimum total cost for two successive iterations are same then
18             count ← count + 1;
19         else
20             count ← 0;
21         end
22     end
23     iterate ← iterate + 1;
24 end
25 F ← the solution from the set F_1, ..., F_4 with minimum total cost;
26 T = TotalCost(C, F);
27 return F, T
```

Table 1 Results obtained for OR-library benchmark data (uncapacitated)

Data file [21]	Size of data file	Optimal value [21]	Deterministic BFR			Randomized BFR		
			Total cost	*Gap*%	Time (s)	Total cost	*Gap*%	Time (s)
Cap71	16×50	932615.75	932615.75	0.0	0.087	–	–	–
Cap72	16×50	977799.4	977799.4	0.0	0.099	–	–	–
Cap73	16×50	1010641.45	1010641.45	0.0	0.118	–	–	–
Cap74	16×50	1034976.975	1034976.975	0.0	0.120	–	–	–
Cap101	25×50	796648.437	796648.4375	0.0	0.206	–	–	–
Cap102	25×50	854704.2	854704.2	0.0	0.242	–	–	–
Cap103	25×50	893782.112	893782.112	0.0	0.257	–	–	–
Cap104	25×50	928941.75	928941.75	0.0	0.276	–	–	–
Cap131	50×50	793439.562	793439.562	0.0	0.931	–	–	–
Cap132	50×50	851495.325	851495.325	0.0	1.031	–	–	–
Cap133	50×50	893076.712	893782.112	0.079	1.033	893076.712	0.0	2.524
Cap134	50×50	928941.75	928941.75	0.0	1.107	–	–	–
Capa	100×1000	17156454.47830	17156454.47830	0.0	12.827	–	–	–
Capb	100×1000	12979071.58143	12979071.58143	0.0	10.259	–	–	–
Capc	100×1000	11505594.32878	11535265.915	0.258	13.118	1505594.32878	0.0	35.572

4 Experimental Results and Discussion

The efficiency of the proposed two algorithms is tested on 15 benchmark instances of Beaslay's OR-Library [20]. Here, all the benchmark instances and the corresponding optimal costs are taken from UflLib [21]. The proposed algorithms are coded with MATLAB R2013a and all the computations are performed in a machine with Intel Core i3 2.30 GHz processor having Ubuntu 14.40 LTS with 4 GB of RAM. To evaluate the performance of the proposed algorithms, we define two performance metrics *Gap%* and *Quality%* which are defined as follows:

$$Gap\% = \left(\frac{\text{Total Cost} - \text{Optimal Value}}{\text{Optimal Value}} \right) \times 100,$$

$$Quality\% = 100 - Gap\%.$$

At first, we run the deterministic BFR on the instances of UFLP. If optimal results given in UflLib [21] are obtained for these instances then we do not run the randomized BFR. The randomized BFR is applied on UFLP instances only when the deterministic BFR fails to give optimal results. In our experiments, the values of *max_iterate* and *max_count* are set to 10 and 4 respectively. The results for the benchmark instances are given in Table 1. Out of the total 15 instances, the deterministic algorithm achieves optimal results for 13 instances and for the remaining two instances, the randomized algorithm achieves the optimal results. In Table 2, the results of large size OR-Library instances [20] are compared with the Lagrangian-type relaxation algorithm proposed by Monabbati [18]. It is observed from Table 2 that the proposed algorithms perform better.

Table 2 Comparison for OR-library benchmark

Data file [21]	HDA [18]		CPLEX time (s) [18]		Deterministic BFR		Randomized BFR	
	Time (s)	*Quality%*	With HDA	Without HDA	Time (s)	*Quality%*	Time (s)	*Quality%*
Capa	4.391	96.45	65.88	28.02	12.827	**100**	–	–
Capb	4.625	99.29	74.58	30.84	10.259	**100**	–	–
Capc	3.704	98.14	106.5	100.8	13.118	**99.742**	35.572	**100**

5 Conclusion

In this paper, two heuristic algorithms are proposed to solve the Uncapacitated Facility Location Problem (UFLP). For most of the instances, the deterministic BFR gives optimal or near-optimal results. The randomized BFR has been found to provide optimal results for all the instances where the deterministic BFR fails to give optimal results. It is to be noted that the result found by the randomized BFR is always better or at least same as the result obtained by the deterministic BFR. For future work, the effects of the three phases used in the proposed algorithms on the final result can be analyzed and more experiments on other UFLP instances available in the literature can be performed.

References

1. Krarup, J., Pruzan, P.M.: The simple plant location problem: survey and synthesis. Eur. J. Oper. Res. **12**(1), 36–81 (1983)
2. Balinski, M.: On finding integer solutions to linear programs. Technical Report, DTIC Document (1964)
3. Erlenkotter, D.: A dual-based procedure for uncapacitated facility location. Oper. Res. **26**(6), 992–1009 (1978)
4. Al-Sultan, K., Al-Fawzan, M.: A tabu search approach to the uncapacitated facility location problem. Ann. Oper. Res. **86**, 91–103 (1999)
5. Sun, M.: Solving the uncapacitated facility location problem using tabu search. Comput. Oper. Res. **33**(9), 2563–2589 (2006)
6. Kratica, J., Tošic, D., Filipović, V., Ljubić, I.: Solving the simple plant location problem by genetic algorithm. RAIRO Oper. Res. **35**(01), 127–142 (2001)
7. Khumawala, B.M.: An efficient branch and bound algorithm for the warehouse location problem. Manag. Sci. **18**(12), B–718 (1972)
8. Garey, M.R., Johnson, D.S.: Computers and intractability: a guide to NP-completeness (1979)
9. Lenstra, J., Kan, A.R.: Complexity of Packing, Covering and Partitioning Problems. Econometric Institute (1979)
10. Akinc, U., Khumawala, B.M.: An efficient branch and bound algorithm for the capacitated warehouse location problem. Manag. Sci. **23**(6), 585–594 (1977)
11. Bilde, O., Krarup, J.: Sharp lower bounds and efficient algorithms for the simple plant location problem. Ann. Discrete Math. **1**, 79–97 (1977)
12. Shmoys, D.B., Tardos, É., Aardal, K.: Approximation algorithms for facility location problems. In: Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, pp. 265–274. ACM (1997)
13. Guha, S., Khuller, S.: Greedy strikes back: improved facility location algorithms. J. Algorithms **31**(1), 228–248 (1999)
14. Ghosh, D.: Neighborhood search heuristics for the uncapacitated facility location problem. Eur. J. Oper. Res. **150**(1), 150–162 (2003)
15. Resende, M.G., Werneck, R.F.: A hybrid multistart heuristic for the uncapacitated facility location problem. Eur. J. Oper. Res. **174**(1), 54–68 (2006)
16. Beltran-Royo, C., Vial, J.P., Alonso-Ayuso, A.: Solving the uncapacitated facility location problem with semi-Lagrangian relaxation. Stat. Oper. Res., Rey Juan Carlos University, Mostoles, Madrid, España (2007)

17. Lazic, N., Frey, B.J., Aarabi, P.: Solving the uncapacitated facility location problem using message passing algorithms. In: International Conference on Artificial Intelligence and Statistics, pp. 429–436 (2010)
18. Monabbati, E.: An application of a Lagrangian-type relaxation for the uncapacitated facility location problem. Jpn. J. Ind. Appl. Math. **31**(3), 483–499 (2014)
19. Ardjmand, E., Amin-Naseri, M.R.: Unconscious search-a new structured search algorithm for solving continuous engineering optimization problems based on the theory of psychoanalysis. In: Proceedings of Advances in Swarm Intelligence, pp. 233–242. Springer (2012)
20. Beasley, J.E.: OR-library: distributing test problems by electronic mail. J. Oper. Res. Soc. 1069–1072 (1990)
21. Hoefer, M.: UflLib, benchmark instances for the uncapacitated facility location problem (2003). http://www.mpi-inf.mpg.de/departments/d1/projects/benchmarks/UflLib