INT J COMPUT COMMUN, ISSN 1841-9836
8(6):845-853, December, 2013.

An Improved Genetic Algorithm for the Multi Level Uncapacitated Facility Location Problem

V. Korać, J. Kratica, A. Savić

Vanja Korać*, Jozef Kratica
Mathematical Institute, Serbian Academy of Sciences and Arts
Kneza Mihaila 36/III, 11 000 Belgrade, Serbia
vanja@mi.sanu.ac.rs, jkratica@mi.sanu.ac.rs
*Corresponding author: vanja@mi.sanu.ac.rs

Aleksandar Savić
University of Belgrade, Faculty of Mathematics
Studentski trg 16, 11000 Belgrade, Serbia
aleks3rd@gmail.com

Abstract: In this paper, an improved genetic algorithm (GA) for solving the multi-level uncapacitated facility location problem (MLUFLP) is presented. First improvement is achieved by better implementation of dynamic programming, which speeds up the running time of the overall GA implementation. Second improvement is hybridization of the genetic algorithm with the fast local search procedure designed specially for MLUFLP. The experiments were carried out on instances proposed in the literature which are modified standard single level facility location problem instances. Improved genetic algorithm reaches all known optimal and the best solutions from literature, but in much shorter time. Hybridization with local search improves several best-known solutions for large-scale MLUFLP instances, in cases when the optimal is not known. Overall running time of both proposed GA methods is significantly shorter compared to previous GA approach.
Keywords: evolutionary approach, metaheuristics, discrete location, combinatorial optimization.

1 Introduction

During the last decades, there was an expansive growth in the ways of solving facility location problems. Research has been concentrated mostly on location problems, which require minimization of total travel time, physical distance, or some other related cost. In most cases it is assumed that facilities are large enough to meet any demand. So far, there are several deterministic uncapacitated models proposed in the literature.

The multi-level uncapacitated facility location problem (MLUFLP) is NP-hard, since it is a generalization of the uncapacitated facility location problem which is proven to be NP-hard in [7].

The multi-level version of uncapacitated facility problem is discussed only in several papers ([1–5, 8, 9]), compared to several hundreds of papers about the basic version of the problem. Moreover, most of the papers that deal with MLUFLP were of theoretical nature, without experimental results. The only exceptions are papers [5] and [8].

In [5] four methods for solving MLUFLP are implemented. Three algorithms are based on linear programming relaxation of the model, described in Section 2, which has an enormous number of variables and moderate number of constrains. Consequently, these three algorithms are capable of solving only small size MLUFLP instances with up to 52 potential facility locations. Running times of these methods are greater than 100 seconds. Gap values are rather satisfying

but the corresponding running times are very long even for these small size instances. Obviously, all three linear programming based methods are unable to solve larger size problem instances. The fourth method produce very quick results (up to 0.07 seconds), but the gaps are very large - up to 732%. From these facts, it can be concluded that none of these algorithms is capable of solving real medium-size and large-scale MLUFLP instances.

The only method capable of reaching optimal solutions in solving small and medium-size MLUFLP instances was an evolutionary approach presented in [8]. It also gives results on large-scale MLUFLP instances in a reasonable running time. A binary encoding scheme with appropriate objective function containing dynamic programming approach for solving subproblem wad used, which consisted of finding sequence of located facilities on each level to satisfy clients' demands. Used dynamic programming approach has the polynomial number of states and steps, since the given subproblem is a special case of the shortest path problem. That approach enables the GA to use the standard genetic operators and caching technique, which helps genetic algorithm to reach promising search regions.

2 Previous work

Let us first give a mathematical formulation of the MLUFLP problem as presented in [5]. The input data to the MLUFLP consists of a set of facilities F ($|F| = m$) partitioned into k levels, denoted $F_1, \ldots, F_k$, a set of clients D ($|D| = n$), a fixed cost f_i for establishing facility $i \in F$, and a metric that defines transportation costs c_{ij} for each $i, j \in F \cup D$. A feasible solution assigns each client a sequence of k facilities, one from each level $F_k, \ldots, F_1$, respectively. A feasible solution is charged the sum of the fixed costs of the facilities used, plus the transportation costs of the clients' assignments. Each client's transportation cost is the sum of transportation cost from itself to the first facility of its sequence, plus the transportation cost between successive facilities. An optimal solution is a feasible solution with a minimum total cost.

The assignment of a client $j \in D$ to a valid sequence of facilities can now be represented as the assignment of j to a path p from j to one of the top level facilities F_1. The set of all valid sequences of facilities is defined with $P = F_k \times \ldots \times F_1$ and the transportation cost of client j's assignment to sequence p $= (i_k, \ldots, i_1)$ by $c_{pj} = c_{ji_k} + c_{i_k i_{k-1}} + \ldots + c_{i_2 i_1}$. Variable y_i has value of 1 if facility i is established and 0 otherwise. Similarly, variable x_{pj} represents whether or not a client j is assigned to the path p. Using the notation mentioned above, the problem can be written as:

$$\min \sum_{i \in F} f_i y_i + \sum_{p \in P} \sum_{j \in D} c_{pj} x_{pj} \tag{1}$$

$$\sum_{p \in P} x_{pj} = 1, \qquad for\ each\ j \in D, \tag{2}$$

$$\sum_{p \ni i} x_{pj} \leq y_i, \qquad for\ each\ i \in F, j \in D, \tag{3}$$

$$x_{pj} \in \{0, 1\}, \qquad for\ each\ p \in P, j \in D, \tag{4}$$

$$y_i \in \{0, 1\}, \qquad for\ each\, i \in F. \tag{5}$$

Table 1: Fixed costs

Facilities	f1	f2	f3	f4	f5	f6
Fixed cost	70	50	30	20	20	40

Table 2: Distance between facilities on level 1 and level 2

	f1	f2
f3	58	23
f4	44	74
f5	67	15
f6	29	38

The objective function (1) minimizes the sum of overall transportation cost and fixed costs for establishing facilities. Constraint (2) ensures that every client is assigned to a path while constraint (3) guarantees that any facility on a path used by some client is paid for. Constraints (4) and (5) reflect binary nature of variables x_{pj} and y_i.

The problem can be illustrated with one small example.

Example 1 An example of the MLUFLP is shown below. It assumes $k = 2$ levels of $m = 6$ facilities: the first level contains 2 potential facilities and the second 4 potential facilities. in this example there are $n = 5$ clients to be served. The fixed costs of establishing facilities are given in Table 1, the distances between facilities of different levels and the distances between clients and facilities on the second level are given in Table 2 and Table 3, respectively.

The total enumeration technique, described in Section 4, is used to obtain an optimal solution. Established facilities are: f2 on the first and f3, f5 on the second level. The objective function value is 329. The sequences of facilities for each client are shown in Table 4.

3 Improved GA method

Proposed genetic algorithm proposed in this paper is an improved version of GA used in [8], so after the short summary of the overall genetic algorithm implementation we only give the detailed description of the improvement parts. The outline of our GA implementation is given below, where N_{pop} denotes the overall number of individuals in the population, N_{elite} is a number of elite individuals and *ind* and obj_{ind} mark the individual and its value of objective function.

Input_Data();
Population_Init();
while not Stopping_Criterion() **do**
for *ind*:= (N_{elite} + 1) to N_{pop} **do**

Table 3: Distance between clients and facilities

	f3	f4	f5	f6
client 1	38	13	57	41
client 2	47	52	63	15
client 3	42	48	13	54
client 4	9	54	41	43
client 5	15	18	36	22

Table 4: Sequences of facilities for clients

	level 2	level 1
client 1	f3	f2
client 2	f3	f2
client 3	f5	f2
client 4	f3	f2
client 5	f3	f2

```
      if (Exist_in_Cache(ind)) then
        obj_ind:= Get_Value_From_Cache(ind);
      else
        obj_ind:= Objective_Function(ind);
        Local_Search(ind,obj_ind);
        Put_Into_the_Cache_Memory(ind,obj_ind);
        if (Full_Cache_Memory()) then
          Remove_LRU_Block_From_Cache_Memory();
        endif
      endif
    endfor
    Fitness_Function();
    Selection();
    Crossover();
    Mutation();
endwhile
Output_Data();
```

The encoding of the individuals used in this implementation is binary. The set of facilities F which will be used is naturally represented as the individual represented with a binary string of length m. Digit 1 at the i-th place of the string denotes that $y_i = 1$, while 0 shows the opposite ($y_i = 0$).

Now, for any individual, through its genetic code, string of established facilities is given. Objective function value can now be computed using dynamic programming explained in [8] if the array of established facilities has at least one established facility. Otherwise, solution is not valid and individual is marked as infeasible.

Genetic operators are the same as in [8]. We can briefly summarize them

- Selection operator is fine-grained tournament selection - FGTS [6],
- Crossover operator is a standard one-point crossover operator,
- Mutation operator is a standard simple mutation modified for frozen genes,
- Initialization of first population is random.

The elitist strategy is applied to N_{elite} elite individuals, which are directly passing to the next generation. The genetic operators are applied to the rest of the population ($N_{nnel} = N_{pop} - N_{elite}$ non-elite individuals).

All duplicates of every individual are eliminated. Individuals with the same objective function value, but different genetic codes are limited with the number of their appearance - N_{rv}.

The Fitness_Function(), the fitness f_{ind} of individual ind 1,2, ..., N_{pop} is computed by scaling values of objective function obj_{ind} of all individuals into the interval [0,1], so that the best individual ind_{min} has fitness 1 and the worst one ind_{max} has fitness 0.

Caching technique applied in this improved version of GA is the same as described in [8].

3.1 Improved objective function

In [8] for feasible individual ind, the Objective_Function(ind), is evaluated in four steps.

1. In the first step,the values of variables y_i are obtained from the genetic code. Let us denote the number of established facilities (number of y_i's with value 1) with m_1.

2. In the second step, the array of minimal costs cs is initialized. The array cs carries information about total minimal costs for serving clients and established facilities (except the ones on the first level), regarding the costs for serving facilities on the upper level. The minimal cost values in cs, for the established facilities on the first level are initially set to zero, while the costs for established facilities on remaining levels and clients are set to a large constant $INF = 10^{30}$.

3. The minimal costs for each client and each facility are calculated by dynamic programming. For each facility in each level (except the first one), the array of total minimal costs (initialized in the second step) is updated. The minimal cost value for each facility is the minimum of the sum of the minimal cost for established facility on the upper level and transportation cost between the two facilities. The same procedure is done for each client: Among the established facilities from the last level, the one with the minimal sum of the corresponding minimal cost value and the transportation cost facility-client, is taken.

4. Finally, the objective value is computed by adding all transportation costs facility-client and fixed costs for all established facilities.

As can we seen from the previous algorithm, some of the operation is unnecessary. For example, all pairs of facilities from the two consecutive levels are taken into consideration where as only pairs of established facilities are what matters. In Example 1. there is an optimal solution only for 1 out of 2 facilities established on the first level and only for 2 out 4 facilities established on the second level. Therefore, for calculating minimal cost between the first and the second level, previous algorithm performs 2*4=8 operations (one operation per facility pair), while as only 1*2=2 is really needed. A similar situation arises in calculating facility-client costs, which in previous algorithm as 5*4=20 operation, while only 2*5=10 is really needed.

In order to correct shortcomings of previous algorithm, the following improvements are proposed. New array of established facility indices is constructed. When calculating minimal costs with dynamic programming approach, instead of using all pairs of facilities on consecutive levels we used only pairs of established facilities obtained from the newly constructed array. Similarly, when calculating minimal facility-client costs, , instead of using of all facilities on the last level, only the established once are taken into consideration.

Therefore, usage of this procedure effectively improves objective function calculation which makes the major part in running time of the overall GA implementation. In presented example we have a significant improvement, but for large scale instances improvement can be even greater.

This can be seen on finding objective function for the largest instance $mt1\_5L\_60\_120\_250\_500\_1070.2000$ Best known solutions has 1 established facility on the first 4 levels and 5 established facilities on the 5th level.

Now, instead of calculating

1. 60*120 pairs of facilities between level 1 and 2, there is only one pair of facilities;
2. 120*250 pairs of facilities between level 2 and 3, there is only one pair of facilities;
3. 250*500 pairs of facilities between level 3 and 4, there is only one pair of facilities;
4. 500*1070 pairs of facilities between level 4 and 5, there are five pairs of facilities;
5. 1070*2000 pairs of facilities-clients we have 5*2000 pairs of facilities-clients.

Although objective function calculation has spped up factor greater than 200, speed up of overall GA implementation is about 6.5 times. This discrepancy can be explained that the running time of calculating objective function was conducted previously, instead of representing the major part, becomes dominated by running time of other parts of GA implementation. It is obvious that this way of calculating objective function is much faster so its running time is significantly shorter than those of genetic operators (selection, crossover and mutation).

3.2 Local search

In order to improve the accuracy of the solutions additionally, the proposed GA approach incorporates a local search procedure. It considers the array of facilities and regards which facilities are established and, which are not. In the array facilities which are established they are marked with 1 and those that are not, are marked with 0. Local search starts from the first facility in the array and tries to change its status. If there is an improvement in the value of objective function, local search starts from the beginning of the array, otherwise it moves to the next facility. If the facility that is momentarily in consideration is the only established facility on that level, local search moves to the next facility in the array. This procedure is repeating until there are no more facilities in the array.

Improvement can be determined in two ways, according to the status of the regarded facility.

1) Let us first consider the case where the facility is non-established, and its status is changed into being established. Then new value of objective function is determined in the following way. All established facilities on the previous levels are unchanged with their fixed transport costs. On the current level costs are added with the fact that running facility is now established. All other transport costs are unchanged. On the next level already found transport costs are compared with transport costs through the new established facilities and changed if the latter is smaller. On the levels following this all transport costs must be evaluated anew. This includes transport costs from the last level of facilities to the clients.

2) In the second case an established facility becomes non-established. Transport costs for that level are then reduced for transport cost of that facility. The array is not reduced in length but all paths through that facility are out of consideration. On the next level transport costs are again calculated only for those that were connected through facility considered. On the following levels, transport costs are determined only for those that were on paths leading through the facility considered.

Local search is applied only on the best individual in population and this only if it stayed unchanged in N_{rep} generations (in this work $N_{rep} = 5$). On the one individual, local search is applied only once, because of its deterministic nature. If the best individual is replaced with another individual with equal value of an objective function but different genetic code and that individual stays unchanged through N_{rep} generations, local search is again applied. In this paper, local search was not applied in the first 50 generations, whatsoever.

Table 5: GA results on the instances with previously known optimal solution

Instance name	*Optimal*	*GA*		*ImprovedGA*		*GA + LS*	
	solution	sol	t_{tot}	sol	t_{tot}	sol	t_{tot}
cap71_2L_6_10.50	1813375.51	*opt*	0.202	*opt*	0.195	*opt*	0.226
cap71_3L_2_5_9.50	4703216.31	*opt*	0.22	*opt*	0.195	*opt*	0.233
cap101_2L_8_17	1581551.39	*opt*	0.274	*opt*	0.224	*opt*	0.269
cap101_3L_3_7_15.50	3227179.81	*opt*	0.258	*opt*	0.223	*opt*	0.265
cap131_2L_13_37.50	1592548.45	*opt*	0.557	*opt*	0.358	*opt*	0.407
cap131_3L_6_14_30.50	3201970.46	*opt*	0.546	*opt*	0.355	*opt*	0.396
cap131_4L_3_7_15_25.50	3630297.67	*opt*	0.515	*opt*	0.32	*opt*	0.352

4 Computational results

All tests were carried out on an Intel 2.5 GHz with 2 GB memory. The algorithms were coded in C programming language.

The GA is tested on the same instances as in [8] to show improvement obtained by improved objective function and hybridization with local search. In this paper, only those instances with multiple levels of facilities are taken into consideration, since for the basic problem with only one level several hundreds of papers are available. Therefore, instances with multiple levels are the main interest of this research, while results on instances with only one level of facilities are omitted. For small size instances, optimal values are known from the literature which is indicated in Table 1.

For fair and direct comparison of the results, we leave same GA parameters as in [8]. The finishing criterion of GA is the maximal number of generations $N_{gen} = 5000$. The algorithm also stops if the best individual or best objective value remains unchanged through $N_{rep} = 2000$ successive generations. Since the results of GA are nondeterministic, the GA was run 20 times on each problem instance.

In [8] experimental results are not totally in accordance with the instances. It was possible to repeat testing and about 1/3 of instances values of objective function cannot be validated as presented in that quoted paper. For example, value of objective function, for instance *cap*131_4*l*_3_7_15_25.50, is 3630297.67 in the paper and in repeated testing, but for instance *capa*_3*l*_15_30_55.100 value function in the paper is 40725103.254, and 25424361.91 in the repeated testing. Furthermore, for some instances, results obtained in literature and in repeated testing slightly differ, but that can be caused by slightly different random seed. For example, for instance *ms*1_4*l*_64_128_256_552.1000 values of objective function in the literature and in repeated testing are 30936.585 and 31257.27 respectively which means that result in the quoted paper is slightly better. For instance *mt*1_4*l*_120_250_520_1110.2000 respective results are 65044.003 and 64995.454 which means that result in repeated testing is slightly better. Because of these differences, results of repeated testing will be presented in the whole paper as results obtained by original GA implementation.

Table 1 presents the GA result on smaller and medium instances. In the first column names of instances are given. The instance's name carries information about the number of levels, the number of facilities on each level and the number of clients respectively. For example, the instance *capb*_3*l*_12_25_63.1000 is created by modifying ORLIB instance *capb*, which has 3 levels with 12, 25, 63 facilities, respectively and 1000 clients.

The second column contains the optimal solution on the current instance, if it is previously known, otherwise sign as $-$. The best GA value GA_{best} and running time t_{tot} of the original GA is is given in the following two columns, with marked optimum in cases when GA reached

Table 6: GA results on the instances with unknown optimal solution

Instance name	GA		$Improved\ GA$		$GA + LS$	
	sol	t_{tot}	sol	t_{tot}	sol	t_{tot}
capa_2L_30_70.1000	14829245.63	32.715	14829245.63	16.589	14829245.63	17.517
capa_3L_15_30_55.1000	25424361.91	17.76	25424361.91	8.679	25424361.91	8.933
capa_4L_6_12_24_58.1000	35421258.15	16.231	35421258.15	7.49	35421258.15	7.636
capb_2L_35_65.1000	14479223.79	27.062	14479223.79	14.118	14479223.79	14.283
capb_3L_12_25_63.1000	25986997.29	28.16	25986997.29	12.147	25986997.29	12.477
capb_4L_6_13_31_50.1000	41787432.24	17.371	41787432.24	8.631	41787432.24	9.083
capc_2L_32_68.1000	14072575.52	29.437	14072575.52	16.218	14072575.52	16.901
capc_3L_13_27_60.1000	26751918.75	26.85	26751918.75	12.932	26751918.75	12.773
capc_4L_4_9_27_60.1000	47109818.66	24.491	47109818.66	11.316	47109818.66	11.184
mq1_2L_100_200.300	8341.287	25.561	8341.287	3.091	8341.287	3.180
mq1_3L_30_80_190.300	12994.871	23.95	12994.871	2.903	12994.871	2.951
mq1_4L_18_39_81_162.300	18048.0305	21.233	18048.0305	3.002	18048.0305	3.000
mq1_4L_20_40_80_160.300	17648.0095	20.759	17648.0095	3.141	17648.0095	3.070
mr1_2L_150_350.500	6733.815	88.939	6733.815	8.979	6733.815	10.277
mr1_2L_160_340.500	6707.505	88.878	6707.505	8.752	6707.505	10.096
mr1_3L_55_120_325.500	10911.319	80.546	10911.319	8.39	10911.319	9.401
mr1_4L_30_65_140_265.500	15237.2605	76.126	15237.2605	7.852	15237.2605	8.297
ms1_2L_320_680.1000	13361.3895	479.724	13361.3895	66.767	13361.3895	98.085
ms1_3L_120_250_630.1000	21923.331	364.135	21881.384	64.965	21881.384	89.360
ms1_4L_64_128_256_552.1000	31257.27	385.73	30902.742	54.715	30902.742	79.729
ms1_5L_25_55_120_250_550.1000	40494.7435	373.781	40249.2415	57.169	40094.335	80.764
mt1_2L_650_1350.2000	27733.057	2410.92	27733.057	457.427	27733.057	685.593
mt1_3L_255_520_1225.2000	46278.719	2398.943	46529.979	426.48	46278.719	622.719
mt1_3L_256_600_1144.2000	46095.09	2403.649	46095.09	406.855	46095.09	585.418
mt1_4L_120_250_520_1110.2000	64995.454	2318.705	64995.454	402.143	64851.16	567.867
mt1_5L_60_120_250_500_1070.2000	83486.9185	2265.78	83363.586	371.867	83363.586	543.857

the optimal solution. In the following two columns are given best values and running times, *Improved GA* and t_{tot} of the improved GA algorithm. Finally, in the last two columns best values and running times are given, $GA+LS$ and t_{tot} of the hybridized version of GA algorithm.

As it can be seen from Table 5, there are three variants of GA reached optimal solutions. As expected, running times of the improved GA has been better than for original GA for all instances, and improvement is up to 40 percent. Hybridized GA with LS in some instances also has better running times (4 of 7) than the original GA. It can be noticed that better running times in both variants are accomplished on instances with the greater number of levels and greater number of facilities.

In Table 6 results on larger instances are given. In this case, optimal solutions are not known so that the column is omitted and all other columns have same meaning as in Table 1.

From Table 6 can be concluded that improvements in both new variants of GA were validated. Improved GA has all running times faster than original GA (in some cases like the instances with 500 clients up to 10 times faster)without losing on quality of results. Only in one instance, was result was obtained by improved GA, greater than that obtained by original GA (instance mt1_3L_255_520_1225.2000). In all other instances improved GA has better results. Hybridization with local search also produced improvements. Results obtained from this variant are the best in all instances. This variant of GA produced better results than original GA in 5 instances (mostly in the largest instances), and was better than the improved GA in 3 instances. An interesting fact is that GA with LS has running times shorter almost in all instances (except the smallest instances), and sometimes like in those instances with 300 clients GA with LS run

up to 8 times faster than the original GA. Even in the largest instances hybridized version ran up to 44 times faster and also produced better results. As it could be expected, running times of the hybridized version were always longer than those of Improved GA.

5 Conclusions and Future Works

This paper presented improved genetic algorithm for solving multi-level uncapacitated facility location problem. Improvements are achieved, firstly, in formulating the new version of finding objective function for the problem which resulted in much shorter running times, and secondly, through hybridization with fast and reliable local search procedure, which achieved better results and in almost all cases run faster than original GA approach. These improvements are considerable, since hybrid GA is obtained 5 new best-known solutions and running times are shorter sometimes up to one order of the magnitude compared to original GA version. It can be concluded that both of these versions represent considerable improvements in solving MLUFLP.

Future research can be directed to parallelization of presented GA, hybridization with some other heuristics and its application in solving similar facility location problems.

Bibliography

[1] K. Aardal, F. Chudak, D.B. Shmoys, A 3-approximation algorithm for the k-level uncapacitated facility location problem, *Information Processing Letters*, 72:161–167, 1999.

[2] A. Ageev, Improved approximation algorithms for multilevel facility location problems, *Operations Research Letters*, 30: 327–332, 2002.

[3] A. Ageev, Y. Ye, J. Zhang, Improved combinatorial approximation algorithms for the k-level facility location problem, *SIAM Journal on Discrete Mathematics*, 18: 207–217, 2005.

[4] A.F. Bumb, W. Kern, A Simple Dual Ascent Algorithm for the Multilevel Facility Location Problem, *Proceedings of the 4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems and 5th International Workshop on Randomization and Approximation Techniques in Computer Science: Approximation, Randomization and Combinatorial Optimization*, 55-62, August 18–20, 2001.

[5] N.J. Edwards, Approximation algorithms for the multi-level facility location problem. *Ph.D. Thesis, Cornell University*, 2001.

[6] V. Filipović, J. Kratica, D. Tošić, D. I. Ljubić, Fine Grained Tournament Selection for the Simple Plant Location Problem. it Proceedings of the 5th Online World Conference on Soft Computing Methods in Industrial Applications - WSC5, 152–158, September 2000.

[7] J. Krarup, P. M. Pruzan, The simple plant location problem: Survey and synthesis, *European Journal of Operational Research*, 12: 36–81, 1983.

[8] M. Marić, An efficient genetic algorithm for solving the multi-level uncapacitated facility location problem, *Computing and Informatics*, 29(2): 183–201, 2010.

[9] J. Zhang, Approximating the two-level facility location problem via a quasi-greedy approach. *Mathematical Programming*, 108: 159–176, 2006.