ISSN 1392 – 124X, ISSN 2335 – 884X (online) INFORMATION TECHNOLOGY AND CONTROL, 2013, Vol. 42, No. 2

A Memetic Algorithm for Solving Two Variants of the Two-Stage Uncapacitated Facility Location Problem

Stefan Mišković, Zorica Stanimirović

Faculty of Mathematics, University of Belgrade,
Studentski trg 16/IV, 11 000 Belgrade, Serbia
e-mail: stefan@matf.bg.ac.rs, zoricast@matf.bg.ac.rs

crossref http://dx.doi.org/10.5755/j01.itc.42.2.1768

Abstract. This paper deals with a Two-Stage Uncapacitated Facility Location Problem (TSUFLP), which has important applications in designing telecommunication systems. Given a set of demand points and a set of possible locations for the first and second level concentrators (switches, multiplexors), the goal of the TSUFLP is to define the structure of two-level concentrator access network, such that the total cost of establishing such a network is minimized. We consider two variants of the TSUFLP from the literature and propose an efficient memetic algorithm (MA), based on hybridization of an evolutionary approach and two local-search heuristics. A greedy heuristic is incorporated in the MA frame for efficient calculation of the fitness function, which additionally decreases the overall MA running time. The described MA approach is benchmarked on test instances of medium and large dimensions from the literature, which are adapted for the TSUFLP and involve from 50 to 500 user nodes. On these instances, the proposed MA method quickly reaches all known optimal solutions, previously obtained by a linear programming method from the literature or CPLEX solver. In order to test effectiveness of the MA, we further modify some largescale instances from the literature involving 1000 and 2000 demand points, which can not be solved to optimality. Exhaustive computational experiments show that the MA provides solutions for the newly generated data set in relatively short CPU times. Regarding both solution quality and running times, we conclude that the proposed MA represents a powerful metaheuristic method for solving the TSUFLP and other similar network design problems.

Keywords: memetic algorithms, multi-level facility location problem, combinatorial optimization.

1. Introduction

In the literature there are numerous location problems related to the design and efficiency of telecommunication services. Facility location models have been widely used in optimization of telecommunication networks. While designing a telecommunication network, it is often impractical or inefficient to transfer data from terminal nodes (users) directly to one or more central units. In a typical telecommunication network, concentrator nodes are inserted between terminals and a central unit, in order to collect, route and distribute the data flow. In more complex telecommunication networks, concentrators are usually located on several levels and may provide different types of service. A modern telecommunication network is often structured in a multi-level hierarchical architecture, which includes different types of service. However, in most optimization models that arise from a telecommunication sector, only a part of network, including one or two levels, is subject to optimization.

The models of a telecommunication network may include different structures of access network, which connects terminals and concentrators with a backbone network. A backbone net work is used to interconnect concentrators or to connect them to a central unit. Both access and backbone network may have a star, ring, path or tree structure and may be fully connected or meshed, which results in various topologies of network models.

Regarding the way we evaluate efficiency, relialability, and economy of scale, models of telecommunication network may involve different objective functions. Several factors may be taken into account when defining an objective function: the costs of installing concentrators and central unit, assignment costs, the cost of operating a network, data transportation costs, the capability to satisfy users' demands in a given amount of time, response delay costs, etc. Some advanced models also involve the costs of expanding a network over time, which allow the installation of new concentrators and links over time. For detailed review on facility location models in telecommunication sector we refer the reader to [24].

In the literature, one can find various multilevel location problems that arise from telecommunication sector and other real-life systems. A multi-level facility location model is needed whenever facilities to be located may be grouped in levels by different properties and may interact with each other, such that it is not possible to locate facilities independently at each level. Some applications of multi-level facility location problems are: health care delivery systems [22], [23], [38], [53], solid waste disposal systems [3], education systems [43], [58], emergency medical services [37], [50], production-distribution systems [11], [57], [61], and others. Up to now, several attempts have been made to classify multi-level location problems with respect to different network properties, such as the number of levels, the type of facility hierarchy, flow pattern, service variety, spatial configuration, horizontal interactions, etc (see [17], [44], [51]).

Since network's structure and properties may change over time, recent studies on multi-level facility location problems involve dynamical aspects. Melachrinoudis and Min [40] deal with a two-level facility network structure, considering the possibility of dynamic relocation of an existing facility during a planning horizon. Hinojosa et al. [28] consider a dynamic variant of the twoechelon multicommodity capacitated plant location problem. The model proposed in the paper by Canel et al. [9] involves the possibility of reopening a plant in more than one time period. Dias et al. [19] present uncapacitated and capacitated dynamic multi-level location models, which consider the possibility of a facility being opened, closed, and reopened more than once during the planning horizon.

In this paper, we consider a multilevel facility location problem, named the Two-Stage Uncapacitated Facility Location Problem (TSUFLP). We start with a given set of locations of terminals, a set of potential locations for installing the first-level concentrators and a set of potential sites for locating the second-level concentrators. The objective is to choose locations on the first and second level for installing certain number of concentrators, and to make necessary assignments of each terminal to an installed first-level concentrators, which further have to be assigned to one of the concentrators established at locations on the second-level. These assignments, as well as the installation of concentrators, assume certain fixed costs. The goal of the problem is to establish a two-level telecommunication network, such that the sum of assignment and installation costs is minimized. No capacity restriction is assumed in the network. The problem involves a single allocation scheme, which means that each terminal is assigned to exactly one, previously established concentrator on the first-level, while each concentrator is assigned to at most one, previously located concentrator on the second level.

There is a meaningful number of papers in the literature devoted to the TSUFLP and its variants. The problem was studied by Chardaire et al. in [12], as a natural extension of the uncapacitated facility location problem (UFLP). The authors proposed two integer formulations of the problem and developed a Lagrangian relaxation method (LR) to compute lower bounds on the optimal value of linear programming formulations, and feasible solutions of integer programming models. Chardaire et al. further used a simulated annealing algorithm (SA) to improve some of the upper bounds returned by the Lagrangian relaxation [12]. The combination of the LR and SA approaches provided low differences in upper and lower bounds on LP formulations, when solving problem instances with up to 150 l ocations of terminals, first and second level concentrators.

Landete and Marin [32] considered the TSFULP on a supply network consisting of production plants, depots, and customers. They formulated the TSUFLP as a set packing problem, and used its characteristics to develop new facets for the associated polyhedron. The proposed formulations with triple-indexed variables cover the transportation of the product from the production plant to the final destination and involve demands for each customer [32]. A solution procedure based on families of facet-defining inequalities for the problem under consideration was also described in [32]. Computational tests were carried out on modified smaller size one-level facility location instances, involving up to 50 l ocations of customers, plants on the first level, and depots on the second level. Obtained solutions show that the applied cuts additionally improved the formulations from [32].

Many other variants of the TSUFLP have been studied in the literature up to now. In the paper by Helm and Magnanti [27], it is assumed that all concentrators are assigned to a central unit, which implies additional costs for establishing the network. The study by Chung et al. [15] considers a fully connected backbone network of concentrators, and assumes that each terminal is connected to each concentrator. This case was further studied by Current and Pirkul in [16], who proposed two heuristic methods based on Lagrangean relaxation. Matheus et al. [39] imposed a limit on the number of concentrators to be installed, and proposed a solution technique based on simulated annealing. In the study by Pirkul and Nagarajan [46], the authors considered a m inimal spanning tree structure for the backbone network of concentrators and develop solution methods inspired by Lagrangean relaxation. Marin and Pelegrin in [36] imposed capacity conditions on the first-level concentrators. By using Lagrangian relaxation, lower bounds and heuristic solutions were obtained for two variants of the capacitated two-stage location problem. Another capacitated variant of the TSUFLP was studied by Wildbore in [62]. Wildbore proposed a combination of Lagrangean relaxation and branch-and-bound technique was as a solution method to the considered problem [62]. A capacitated model of a twolevel freight distribution system was considered by Bocca et

al. in [6]. The problem was decomposed in several sub-problems and a t abu-search heuristic was efficiently applied to the composing subproblems.

In this paper, we propose an efficient memetic algorithm (MA) for solving two variants of the TSUFLP, which are presented in [12] and [32]. The proposed MA uses binary representation of solutions and appropriate evolutionary operators. A greedy heuristic method is applied for calculating the fitness function of each potential solution. Two local search heuristics are incorporated in the MA frame, directing the algorithm to promising search regions. The MA is further enriched with several strategies, which preserve the diversity of solutions in the population and prevent a premature convergence of the algorithm. Finally, the proposed MA is benchmarked on the set of newly generated TSFLUP test instances, including challenging large-scale problem instances with up to 2000 locations of terminals. The MA solutions are compared with optimal solutions obtained by CPLEX 12.1 or existing exact methods. Presented computational results clearly indicate the robustness and efficiency of the proposed memetic algorithm for solving the TSUFLP.

2. Mathematical formulations

In this paper, we first consider the integer formulation of the TSUFLP given in [12]. Two levels of locations for concentrators are involved in the network: each terminal is connected to a concentrator located on the first level, which is further connected to at most one concentrator located on the second level. All concentrators on the second level are connected to a central unit (see Figure 1). As it can be seen from Figure 1, the considered model has star topology for both access and backbone networks (star-star network).

The mathematical model from [12] uses the following notation:

- N denotes the set of locations of terminals;
- M is the set of possible locations for installing first-level concentrators;
- K represents the set of possible locations for establishing second-level concentrators;
- C_{ij} is the cost of assigning terminal at location $i \in N$ to a concentrator at location $j \in M$ on the first-level;
- B_{jk} represents the sum of installing concentrator at location $j \in M$ on the first-level and connecting it to the concentrator at location $k \in K$ on the second level;
- F_k denotes the costs of installing a concentrator at site $k \in K$ on the second-level.

Three sets of decision binary variables are used. A decision binary variable $x_{i,j} \in \{0,1\}$, $i \in N$, $j \in M$ is equal to 1 if terminal at $i \in N$ is assigned to concentrator installed at location $j \in M$ on the first level, and 0 otherwise. A decision binary variable $y_{jk} \in \{0,1\}$, $j \in M$, $k \in K$ is equal to 1 if a concentrator at location $j \in M$ on the first-level is connected to a concentrator at location $k \in K$ on the second level, and 0 otherwise. A decision binary variable $z_k \in \{0,1\}$, $k \in K$ takes the value of 1 if a concentrator is installed at location $k \in K$ on the second level, and 0 otherwise.

For sake of simplicity, in the remainder of the manuscript, a concentrator that is installed at one of the potential locations from the set M will be referred as a *first-level concentrator*. Similarly, a concentrator that is established at one of the potential locations from the set K will be referred as a *second-level concentrator*.

Using the notation mentioned above, the TSUFLP can be formulated as:

$$min \sum_{i\in N}\sum_{j\in M} C_{ij}x_{ij} + \sum_{j\in M}\sum_{k\in K} B_{jk}y_{jk} + \sum_{k\in K} F_k z_k \quad (1)$$

subject to:

$$\sum_{j\in M} x_{ij} = 1 \quad for\ every\ i \in N \quad (2)$$

$$x_{ij} \le \sum_{k\in K} y_{jk} \quad for\ every\ i \in N, j \in M \quad (3)$$

$$y_{jk} \le \mathrm{z_k} \quad for\ every\ j \in M, k \in K \quad (4)$$

$$\sum_{k\in K} y_{jk} \le 1 \quad for\ every\ j \in M \quad (5)$$

$$x_{ij} \in \{0,1\} \quad for\ every\ i \in N, j \in M \quad (6)$$

$$y_{jk} \in \{0,1\} \quad for\ every\ \ j \in M, k \in K \quad (7)$$

$$z_k \in \{0,1\} \quad for\ every\ i \in N \quad (8)$$

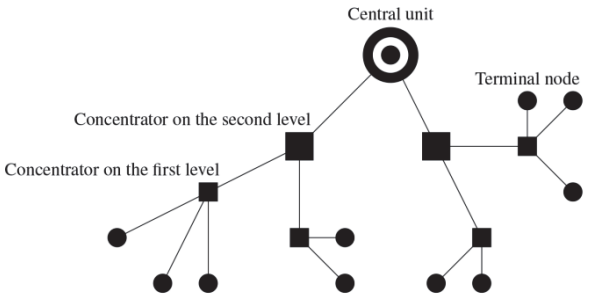

Figure 1. Star-star network

The objective function (1) minimizes the sum of costs of establishing the concentrators at locations on the first and second levels, and the costs of assigning terminals to first-level concentrators and assigning first-level concentrators to secondlevel concentrators. Constraints (2) guarantee that each terminal is allocated to exactly one concentrator located on the first level, which is further connected to a concentrator established on the second level (3). Each first-level

concentrator is assigned to at most one second-level concentrator, which is ensured by constraints (5) and (4), respectively. Finally, constraints (6), (7) and (8) indicate binary nature of variables x_{ij}, y_{jk} and z_k, respectiveely. The TSUFLP is an NP-hard optimization problem, since it represents a generalization of the UFLP, which was proved to be NP-hard in [31].

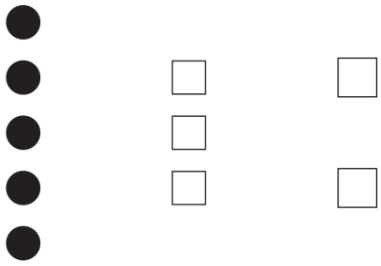

Figure 2. Locations of terminals and potential locations for concentrators on the first and the second level ($N = 5$, $M = 3$ and $K = 2$)

Example 1. Consider a network with $|N| = 5$ terminal nodes (t_1, t_2, t_3, t_4, t_5), $|M| = 3$ potential locations for first-level concentrators (p_1,p_2,p_3) and $|K| = 2$ potential locations for second-level concentrators (q_1, q_2), as it is shown in Figure 2. The assignment costs of terminals to firstlevel concentrators are given in Table 1, while the costs of installation of first-level concentrators and their assignments to second-level concentrators are presented in Table 2. Finally, the costs of establishing second-level concentrators q_1 and q_2 are equal to 20 a nd 16, respectively.

Table 1. The assignment costs of demand nodes to concentrators on the first level

	p_1	p_2	p_3
t_1	12	22	18
t_2	14	19	20
t_3	20	31	13
t_4	22	21	2
t_5	17	24	9

Table 2. The costs of installation and assignment of concentrators on the first level to concentrators on the second level

	q_1	q_2
p_1	29	12
p_2	28	31
p_3	21	13

The optimal solution of Example 1 is presented in Figure 3. Concentrators p_1 and p_3 are installed on the first level, while concentrator q_2 is established on the second level. Terminal nodes t_1 are t_2 assigned to p_1, while t_3, t_4 and t_5 are as-signed to p_3. Both first-level concentrators p_1 and p_3 are allocated to the second-level concentrator q_2. The objective function that corresponds to optimal solution takes the value of 91.

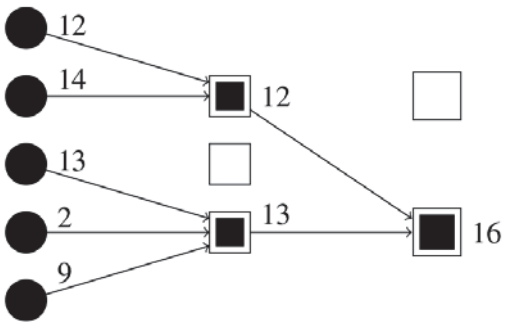

Figure 3. Optimal solution for Example 1

In the paper by Landete and Marin [32], a model of a supply chain network is considered. The model involves a set of locations of customers – N, a set of potential locations for depots – M and a s et of potential locations for plants – K. Note that a customer's location in this model corresponds to a location of a terminal node in the model by [12]. Potential locations for depots and plants correspond to the possible locations of first-level and second-level concentrators from [12], respectively. Installing a depot at location $j \in M$ assumes certain location costs f_j. Similarly, building a plant at location $k \in K$ requires location costs g_k. A customer at location $i \in N$ requires $D_i > 0$ units of a product. Allocation costs considered in [12] correspond to transportation costs in [32]:

- $d_{kj} > 0$ is the cost of transporting one unit of product from a plant located at $k \in K$ to a depot located at $j \in M$,
- $c_{ij} > 0$ is the cost of transporting one unit of product from a depot established at $j \in M$ to a customer at $i \in N$.

Note that in this model, location costs for installing depots are considered separately from allocation costs, while in the model by [12] installation costs for a first-level concentrator are added to the costs of allocating this first-level concentrator to a s econd-level concentrator. One should also observe that the model in [32] considers the flow originating from a plant located at $k \in K$, which is further routed via depot installed at location $j \in M$ and distributed to a customer at $i \in N$ (see Figure 4).

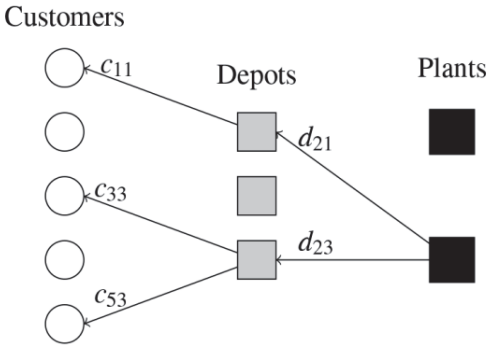

Figure 4. Example of a supply chain network

The goal of the TSUFLP variant in [32] is to establish a supply chain network in such a way that the total costs are minimized. The total costs are obtained by summing up the transportation costs of both stages, installation costs of all installed depots that send products to a customer, and installation costs of all located plants that send products to an installed depot.

The formulation of TSUFLP model considered in [32] uses the following variables:

- binary variable y'_j, $j \in M$, takes the value of 1 if a depot at location j is installed and 0 otherwise,
- binary variable z'_k, $k \in K$, takes the value of 1 if a plant at location k is opened, and 0 otherwise,
- three-indexed continuous variable $0 \leq x_{ijk} \leq 1$, $i \in N$, $j \in M$, $k \in K$, represents a fraction of the demand of a customer at location $i \in N$, which is supplied from a plant opened at $k \in K$, via a depot located at $j \in M$.

Using the notation mentioned above, the formulation of the TSUFLP from [32] can be written as:

$$min \sum_{j \in M} f_j y'_j + \sum_{k \in K} g_k z'_k + \sum_{i \in N} \sum_{j \in M} \sum_{k \in K} C_{ijk} x_{ijk} \quad (9)$$

subject to:

$$\sum_{j \in M} \sum_{k \in K} x_{ijk} = 1 \quad for\ every\ i \in N \quad (10)$$

$$x_{ij} \leq y'_j \quad for\ every\ i \in N, j \in M, k \in K \quad (11)$$

$$x_{ijk} \leq z'_k \quad for\ every\ i \in N, j \in M, k \in K \quad (12)$$

$$y'_j \in \{0,1\} \quad for\ every\ j \in M \quad (13)$$

$$z'_k \in \{0,1\} \quad for\ every\ k \in K \quad (14)$$

$$0 \leq x_{ijk} \leq 1\ for\ every\ i \in N, j \in M, k \in K \quad (15)$$

The objective function (9) minimizes the sum of costs of establishing depots and plants, and the costs $C_{ijk} = D_i(c_{ij} + d_{kj})$ of transporting the demand of a customer located at $i \in N$ from a plant established at $k \in K$ K, via a depot installed at $j \in M$. Constraints (10) guarantee that the demand of each customer is supplied via one or more one plant-depot combinations. Constraints (11) ensure that each customer is supplied via an installed depot. All customers are supplied from installed plants only, which is guaranteed by constraint (12). Constraints (13) and (14) indicate binary nature of variables y'_j and z'_k, respectively. Variables x_{ijk} are continuous and take the values from the interval [0;1], which is stated by constraints (15).

Note that Landete and Marin in [32] proved that variables x_{ijk} are also binary. In the sake of simplicity, in remaining part of the manuscript, we will identify customers with terminals, depots with first-level concentrators and plants with second-level concentrators. Without the loss of generality, it may be assumed that the demands D_i are equal to 1 for every $i \in N$ in the second variant of the TSUFLP model from [32].

3. Proposed memetic algorithm

The main motivation behind a hybridization of different algorithms is to exploit the complementary character of different optimization strategies. Hybridization is not restricted to a combination of two or more different heuristic or metaheuristic methods. It also includes, for example, the combination of exact algorithms with approximative algorithms, such as heuristics or metaheuristics. Choosing an adequate combination of algorithmic concepts may be the key for achieving excellent performance in solving many hard optimization problems. In the literature, one can find powerful hybrid metaheuristic algorithms, obtained by combining elements from two or more different optimization techniques. A detailed survey of state of-the-art hybrid metaheuristics in combinatorial optimization can be found in [5].

Genetic algorithm (GA) is a problem-solving metaheuristic, based on the concept of natural evolution. However, in some cases, a pure GA cannot provide high-quality solutions, especially when solving complex combinatorial optimization problems. In these situations, it is useful to combine a pure GA with other approaches, in order to improve GA's search capabilities. For example, a GA may be hybridized with particle swarm optimization [52], [10], simulated annealing [42], extremal optimization [13], Lagrangian decomposition [47], neural networks and constraint constraint programming [48], iterated local search [41], variable neighborhood search [35], etc.

Memetic algorithm is a hybrid meta-heuristic method that combines a genetic algorithm and a local search procedure [45]. Local search directs a genetic algorithm to promising regions of the search space, and if applied efficiently, it r esults in obtaining high quality solutions in significantly shorter running time. In general, developing an effective memetic algorithm is a difficult task, which requires the expertise from both population based and single solution based optimization techniques. A certain memetic algorithm might work well for one specific problem, but it might perform poorly for others. Different variants of the MA have been presented in the literature for solving various combinatorial optimization problems, see [18], [14], [7], and [25].

In this paper, we propose a novel memetic algorithm (MA) that is designed in order to solve the TSUFLP in an efficient manner, especially large-scale problem instances. The proposed MA showed to be a successful hybrid metaheuristic for solving the TSUFLP, regarding both solutions' quality and

running times. Its efficiency lies in the tradeoff between the exploration abilities of the GA and the solution improvement achieved by two implemented local search procedures. In the implemented MA approach, a greedy procedure has been used in order to evaluate fitness function of each potential solution, which additionally reduces the overall MA running time. A new modified uniform crossover operator is proposed and implemented in the MA frame. Selection and mutation operators adopted to problem under consideration are used. Several additional strategies have been applied in order to increase the efficiency of the MA. The basic pseudo-code of the proposed MA is given in Algorithm 1, while detailed description of the MA components will be presented in the following subsections.

Note that the proposed memetic algorithm is constructed in such a way that it can be applied for solving both variants of the TSUFLP. The only difference is in the procedure for greedy fitness calculation, due to different objective functions in the two considered TSUFLP formulations.

Algorithm 1 The basic structure of the MA

1: *Read Input()*
2: *Generate Initial Population()*
3: **while not** *Termination Criteria* **do**
4: *Greedy Fitness Calculation()*
5: *Selection operator()*
6: *Crossover operator()*
7: *Mutation operator()*
8: *Local Search-Inversion()*
9: *Local Search-Transposition()*
10: **end while**
11: *Write Output()*

3.1. Representation of solutions

In our MA implementation, a binary encoding of individuals is used. Each solution is represented by a two-segment binary string of length $|M| + |K|$. The first segment in the genetic code is of length $|M|$, where each bit (gene) corresponds to one potential location for installing a first-level concentrator. The second segment is of length $|K|$ and it encodes potential locations for the secondlevel concentrators. The bit value of 1 in the genetic code denotes that particular concentrator is installed, while 0 shows it is not.

Since terminals and first-level concentrators can be assigned only to established first-level and second level-concentrators, respectively, from the genetic code we only obtain the set of locations of installed concentrators on both levels. From the first segment of the genetic code, we read the locations of installed first-level concentrators. From the second part of the genetic code, we read the locations of the established second-level concentrators, which give us the values of variables z_k, $k \in K$.

For instance, the genetic code $(101|01)$ corresponds to optimal solution presented in Figure 3. Bit values in the first segment of the genetic code imply that concentrators are installed at locations 1 and 3 on the first-level. From the second segment, we conclude that only concentrator at location 2 is installed on the second level. This further implies that $z_1 = 0$ and $z_2 = 1$.

3.2. Fitness function calculation

Fitness function is used for evaluating the quality of individuals in the population. The calculation of a fitness function is the most timeconsuming part of the algorithm, since the fitness has to be determined for each individual in each MA generation. An inefficient fitness calculation may influence the algorithm's running time significantly. In order to ensure the MA's efficiency, we have used a greedy heuristic to calculate the fitness function of an individual in the MA population.

The applied greedy procedure for the first variant of the TSUFLP [12] starts from the sets of locations of installed first-level and second-level concentrators, obtained from an individual's genetic code. For each terminal node at $i \in N$, the procedure tries to find the its best assignment to a concentrator installed at some location $j \in M$ on the first level. Once the best allocation of a terminal node at $i \in N$ to an installed concentrator at $j \in M$ is found, the value of x_{ij} is set to 1, the corresponding cost C_{ij} is added to the fitness value, and the location $j \in M$ of the installed first-level concentrator is marked as *used*.

The same procedure is performed in order to find the best assignment of each concentrator installed at location $j \in M$ on the first level to an established concentrator at location $k \in K$ on the second level. When this best assignment (in the sense of minimal assignment costs) is found, the value of y_{jk} is set to 1, the cost B_{jk} is added to the fitness function, and the location $k \in K$ of the installed second-level concentrator is marked as *used*. Fixed costs F_k for all used locations of second-level concentrators $k \in K$ are added to the fitness value.

Finally, we consider all locations of the installed first-level and second-level concentrators, which are not marked as *used*. On these locations the concentrators are being uninstalled, i.e. the corresponding indices in the genetic code are changed from 1 to 0. By uninstalling concentrators on locations that are not used, we additionally help the algorithm to converge to well-fitted solutions. The pseudocode of the Greedy Fitness function calculation for the first variant of the TSUFLP is presented in Algorithm 2.

Similar procedure for fitness calculation is applied in the the case of the second variant of the TSUFLP. The procedure, named Greedy Fitness function calculation 2, is presented in Algorithm 3. In the same step, this procedure tries to find the best assignment of a terminal at location $i \in N$ to a concentrator installed

Algorithm 2 Greedy Fitness function calculation 1

1: Get locations of installed first-level and secondlevel concentrators from the genetic code
2: *Fitness* = 0
3: **for all** $i \in N$ **do**
4: Find the best assignment of a terminal at location i to a first-level concentrator installed at location $j \in M$
5: Add the cost of the found best assignment to *Fitness*
6: Mark location the j of installed first-level concentrator in the found best assignment as *used*
7: **end for**
8: **for all** $j \in M$ **do**
9: **if** the location j of installed first-level concentrator is *used* **then**
10: For the installed first-level concentrator at location j find the best assignment to an installed concentrator at some location $k \in K$ on the second level
11: Mark the location k of the installed secondlevel concentrator in the found best assignment as *used*
12: Add the cost of the found best assignment to *Fitness*
13: **end if**
14: **end for**
15: **for all** $k \in K$ **do**
16: **if** location k of the installed second-level concentrator is *used* **then**
17: Add the cost of installing the second-level concentrator at k to *Fitness*
18: **end if**
19: **end for**
20: **for all** $j \in M$ **do**
21: **if** a first-level concentrator at location j is installed, but j is not *used* **then**
22: Uninstall the first-level concentrator at location j
23: **end if**
24: **end for**
25: **for all** $k \in K$ **do**
26: **if** a second-level concentrator at location k is installed, but k is not *used* **then**
27: Uninstall the second-level concentrator at location k
28: **end if**
29: **end for**
30: **return** *Fitness*

at some location $j \in M$ on the first-level and to a concentrator installed at some location $k \in K$ on the second level. After the best allocations for a terminal at $i \in N$ are found for each $i \in N$ the values of x_{ijk} are determined and corresponding costs C_{ijk} are added to the fitness value. Locations $j \in M$ and $k \in K$ with installed concentrators on the first and second level, respectively, are marked as $used$. Fixed costs f_j and g_k for all installing concentrators at $j \in J$ and $k \in K$ are added to the fitness value. Finally, installed concentrators on locations on the both levels that are not marked as used are being uninstalled.

Algorithm 3 Greedy Fitness function calculation 1

1: Get locations of installed first-level and second-level concentrators from the genetic code
2: *Fitness* = 0
3: **for all** $i \in N$ **do**
4: Find the best assignment of a terminal at location i to a concentrator installed at location $j \in M$ on the first level to a concentrator installed at location $k \in K$ on the second level
5: Mark location the $j \in M$ and $k \in K$ in the found best assignment as *used*
6: Add the cost of the found best assignment to *Fitness*
7: **end for**
8: **for all** $j \in M$ **do**
9: **if** location j of installed first-level concentrator is *used* **then**
10: Add the cost of installing the first-level concentrator at location j to *Fitness*
11: **end if**
12: **end for**
13: **for all** $k \in K$ **do**
14: **if** location k of the installed second-level concentrator is *used* **then**
15: Add cost of installing the second-level concentrator at k to *Fitness*
16: **end if**
17: **end for**
18: **for all** $j \in M$ **do**
19: **if** a first-level concentrator at location j is installed, but j is not *used* **then**
20: Uninstall the first-level concentrator at location j
21: **end if**
22: **end for**
23: **for all** $k \in K$ **do**
24: **if** a second-level concentrator at location k is installed, but k is not *used* **then**
25: Uninstall the second-level concentrator at location k
28: **end if**
29: **end for**
30: **return** *Fitness*

3.3. Generating Initial Population

In order to ensure the diversity of genetic material, an initial MA population, numbering $N_{pop} = 90$ individuals, is randomly generated. Due to different nature of two segments in the individual's genetic code, we have set different probabilities for generating bits with the value of 1 in each segment. The probability of generating 1 in the first segment is set to 0.35, while in the second segment it is set to 0.25. The pseudocode of this step of the MA is presented in Algorithm 4.

3.4. MA operators

The MA implementation uses a fine-grained tournament selection (FGTS), which involves tournaments of different size [21]. The FGTS operator

Algorithm 4 Generating initial population

1: **for all** $Individual \in Population$ **do**
2: **for** $i \leftarrow 1$ **to** $|M|$ **do**
3: **if** $Prob() \leq 0.35$ {$Prob()$ is a function that returns random number in the range [0,1]} **then**
4: $Gene_{Individual}[i] \leftarrow 1$
5: **else**
6: $Gene_{Individual}[i] \leftarrow 0$
7: **end if**
8: **end for**
9: **for** $i \leftarrow |M|+1$ **to** $|M|+|K|$ **do**
10: **if** $Prob() \leq 0.25$ **then**
11: $Gene_{Individual}[i] \leftarrow 1$
12: **else**
13: $Gene_{Individual}[i] \leftarrow 1$
14: **end if**
15: **end for**
16: **end for**

depends on a real parametre F_{tour}, representing the average tournament size. In our MA approach, the FGTS performs two types of tournaments with different number of competitors. The size of the first tournament type is set to $\lceil F_{tour} \rceil$, while the second type has $\lfloor F_{tour} \rfloor$ individuals participating. The value of F_{tour} is calculated as $F_{tour} = \frac{k_1 \lceil F_{tour} \rceil + k_2 \lfloor F_{tour} \rfloor}{k_1 + k_2}$, where k_1 and k_2 represent the number of tournaments of the first and second type, respectively. Parameter value $F_{tour} = 5.4$ is used in our MA, which means that $k_1 = 12$ tournaments have 6 participants and $k_2 = 18$ tournaments have 5 participants.

Most of evolutionary approaches proposed in the literature use simple one-point crossover operator [26], [59], [56], two-point crossover operator [1], [29], or standard uniform crossover [30], [60]. In our MA implementation, an uniform crossover operator is applied on selected pair of parents producing two offspring. This operator is modified to the problem under consideration and adapted to MA's representation scheme. For each pair of parent-individuals, the modified uniform crossover randomly generates a two-segment binary mask of length $|M| + |K|$. The first segment of the mask is of length $|M|$ and in this segment exactly $\lceil \sqrt{|M|} \rceil$ bits on randomly chosen positions have the value of 1. The second segment has $|K|$ bits, of which exactly $\lceil \sqrt{|K|} \rceil$ randomly chosen ones have the value of 1. A selected pair of parent-individuals exchange genetic material according to bit values of a generated two-segment mask. If a bit on the *i*-th position in the mask is equal to 1, the parents will exchange bits on this position, otherwise no exchange will occur (see Figure 5). Crossover is performed with the rate $p_{cross} = 0.85$. It means that around 85% pairs of individuals will exchange genetic material and produce offspring.

Offspring generated by modified uniform two-segment crossover are subject to a two-segment mutation operator. Mutation is performed by changing a randomly selected bit in the individual's genetic

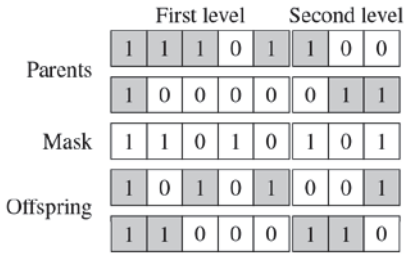

Figure 5. Modified uniform crossover

code (0 to 1, 1 to 0). Mutation rates differ for the first and second part of the genetic code and depend on the problem size, i.e., the number of potential locations for concentrators on the first and second levels. In our implementation, bits in the first segment of the genetic code are inverted with the probability of $1/|M|$, while the bits in the second segment are inverted with the probability of $0.5/|K|$. These mutation rates are constant through all generations of the MA.

Generational and elitist replacement strategies are often used in GA implementations from the literature. Generational replacement in each generation replaces entire population with new individuals [2], [8]. Elitist strategy preserves the best individual from the current population and replaces the remaining members of the population with new chromosomes [26], [59], [33].

Another approach is a steady-state generation replacement with elitist strategy, which is used in our MA implementation. The idea is to is to keep not only the best individual, but also some of the highly-fitted individuals from the current population [55], [56]. In every MA generation, all individuals are ranked according to their objective function value, and the best-fitted $\frac{2}{3}N_{pop}$ individuals from the population are selected as *elite* ones. *Elite* individuals are directly passing in the next generation, preserving highly fitted genes. Remaining $\frac{1}{3}N_{pop}$ *non-elite* individuals are subject to MA operators and they are replaced in the following generation. An individual with the best objective value is denoted as the *best individual*, which is updated through MA generations, whenever we achieve some improvement of the best objective value. The advantage of the applied strategy is that elite individuals do not need recalculation of the objective value, since each of them is evaluated in one of the previous generations. In this way, we provide additional time savings through the MA's run.

In each MA generation, duplicated individuals, i.e., individuals with the same genetic code, are removed from the population by setting their fitness to an undesirable value. However, after certain number of MA generations, it may happen that the number of similar individuals with the same fitness value, but different genetic codes, significantly increases. If these similar individuals are well-fitted, they may become dominant in the population, increasing the possibility of premature convergence to a local optimum. In order to avoid a local optimum trap,

standard approaches in the literature simply delete one or more individuals with similar genetic codes from the population, see [20], [34], [54], and [55].

In our MA implementation, we apply another strategy for solving the problem of similar individuals: we change their genetic codes until their fitness values become different. More precisely, we iteratively change the value of randomly chosen gene, or perform a transposition of two randomly chosen genes in the genetic code of a similar individual until we obtain different fitness value from the initial one. This strategy is applied only if the number of similar individuals exceeds some constant (20 in our MA implementation), because we don't want to significantly decrease the quantity of good genetic material in the population. By using this strategy, we keep the diversity of genetic material, preserve highly fitted genes and keep the algorithm away from premature convergence.

The combination of two stopping criteria is used in the proposed MA. The algorithm stops if the number of generations exceeds $200 \cdot |N|$ or the number of the best individual repetitions exceeds $100 \cdot |N|$.

3.5. Local Search Procedures

Solutions generated in the GA stage are improved by two local search heuristics based on inversijon and transpositions of bits in the genetic code. Both local search procedures tend to explore regions of the search space that would otherwise be left unexplored, hence providing better solutions and avoiding local optimum trap. Local search heuristics are performed in an iterative way, in order to find a better choice for locations of the first-level and second-level concentrators.

Both local search procedures are applied on each individual in the population, ensuring that each individual has equal chances to be improved. An exhaustive application of a local search on an individual may consume a large amount of computation time, due to many calls of fitness function calculation. However, the efficiency of the greedy approach, which is used for evaluating individuals in the MA, allows us to explore the neighborhood of each individual thoroughly, without significant prolonging the MA's total running time.

By applying procedure *Local Search-Inversion*, we first try to change the value of randomly chosen bit (0 to 1, 1 to 0) in an individual's genetic code, looking for an improvement of the fitness function. The second improvement heuristic *Local Search-Transposition* randomly chooses two bits in the genetic code and exchange their values, trying to obtain a better solution. In both local search procedures, it is necessary to calculate the fitness function of the modified individual. Only in the case that the new fitness value is better than the previous one, we perform necessary changes in the individual's genetic code and produce a new, improved individual. The described process is repeated on an individual as long as we are obtaining some improvement (see Algorithm 5).

Algorithm 5 Local search heuristics

```
1: Local search heuristic-Inversion:
2: for all Individual ∈ Population do
3:     while exists improvement for Individual do
4:       Invert randomly selected bito f Individual
5:     end while
6: end for
7: Local search heuristic-Transposition:
8: for all Individual ∈ Population do
9:     while exists improvement for Individual do
10:      if Prob() ≤ 0.65 then
11:          Exchange two randomly selected bits
             positioned in range [1, |M|] of Individual
12:      else
13:          Exchange two randomly selected bits positio-
             ned in range [|M| + 1, |M| + |K|] of
             Individual
14:      else if
15:    end while
16: end for
```

By changing the value of a bit from 0 to 1, we install a first-level or second level concentrator at location that corresponds to the position of the inverted bit in the genetic code. If a first-level concentrator is installed, for each terminal node we need to check whether it is better to assign it to a newly installed concentrator, or to keep the current assignment. If the assignment of a terminal to the new first-level concentrator is cheaper, we re-assign the terminal to the new concentrator. Similar re-assignment procedure is performed in the case that we have installed a new concentrator at location on the second level (see Algorithm 6).

If the value of the bit is changed from 1 to 0, it means that one of previously installed concentrators is uninstalled. The position of the bit that has been changed gives us the location of the uninstalled concentrator (on the first or second level). In the case that a concentrator at location on the first level has been uninstalled, it is sufficient to consider only the subset of terminals that were assigned to it. For each terminal from this subset, it is necessary to find the best reassignment to one of the currently installed firstlevel concentrators. The re-assignment process is realized in the similar way as in the case of uninstalling one of the previously established concentrators at locations on the second-level (see Algorithm 7).

4. Computational Results

All experiments were carried out on an AMD Athlon II X4 640 on 3.0 GHz with 4GB RAM memory, underWindows 7 operating system. The MA

implementation was coded in C# programming language.

Algorithm 6 Changing *i*-th gene of *Individual* from 0 to 1

```
1: Gene_Individual[i] = 1
2:   if i ≤ |M| then
3:     for j = 1 to |N| do
4:       if first-level concentrator at location i is better
         assignment for terminal at location j then
5:           Update the best assignment for terminal at
             location j
6:           Update the fitness of Individual
7:       end if
8:     end for
9:     Among all locations of installed second-level con-
       centrators find the best assignment for a first-level
       concentrator at location i
10:  else if |M| < i ≤ |M| + |K| then
11:    for j = 1 to |M| do
12:      if second-level concentrator at location i is better
         assignment for the first-level concentrator at lo-
         cation j then
13:          Update the best assignment of first-level con-
             centrator at location j
14:          Update the fitness of Individual
15:      end if
16:    end for
17: end if
```

In order to verify the quality of the MA solutions, the optimization package CPLEX, versijon 12.1, was used to solve considered instances to optimality (if possible). The CPLEX 12.1 was run in the same computational environment as the proposed MA. On each instance from the considered data set, the MA was run 15 times with different random seeds.

Algorithm 7 Changing *i*-th gene of *Individual* from 1 to 0

```
1: Gene_Individual[i] = 1
2:   if i ≤ |M| then
3:     for j = 1 to |N| do
4:       if first-level concentrator at location i is better
         assignment for terminal at location j then
5:           Among all installed concentrators at loca-
             tions on the first level, find the best assign-
             ment for terminal at location j
6:           Update the fitness of Individual
7:       end if
8:     end for
9:   else if |M| < i ≤ |M| + |K| then
10:    for j = 1 to |M| do
11:      if second-level concentrator at location i is the
         best assignment for the first-level concentrator at
         location j then
12:          Among all installed concentrators on the se-
             cond level, find the best assignment for the
             first-level concentrator at location j
13:          Update the fitness of Individual
14:      end if
15:    end for
16: end if
```

The MA was first tested on the set of smaller size instances generated by Landete and Marin for the second variant of the TSUFLP from [32]. These problem instances were obtained by modifying GapA, GapB and GapC instances of dimensions $|N| \leq 50$, $|M| \leq 50$ and $|K| \leq 50$. For more details on generating modified GapA, GapB and GapC data sets, we refer to paper [32]. In Tables 3-5, we present the results of the proposed MA on these data sets, and provide comparisons with the results of the linear programming method for solving the TSUFLP, given by Landete and Marin in [32] (we will denote this method as LP-LM). The LP-LM method from [32] uses facets for a polyhedron associated to the problem, which helps in decreasing the CPU time significantly. However, as the authors state in [32], the duality gaps remained large. The LP-LM algorithm was tested on a Sun Java Workstation W2100z with 2 Opteron dual processors 2.6 GHz and 4 RAM GB.

The results Tables 3-5 are presented as follows:

- The name of a modified GapA, GapB or GapC instance;
- Optimal solution of the current instance– *Optimal* obtained by CPLEX 12.1 solver;
- The best solution – *Sol*: of the LP-LM method from [32]. The mark *Opt* stands for the cases when the best solution of the LP-LM coincides with the optimal solution obtained by CPLEX 12.1;
- Running time od the LP-LM procedure $t[s]$ in seconds;
- The best solution value of MA method– MA_{best}, with mark *Opt* in cases when it reached optimal solution;
- Average running time in which the MA reaches the best/optimal solution – $t[s]$ in seconds;
- Average number of MA generations – *Gen*;
- Average percentage gap – $ag[\%]$ of the MA solution from the Optimal solution or MA_{best};
- Standard deviation – $\sigma[\%]$ of the MA solution from the *Optimal* solution or MA_{best}.

As it can be seen for Tables 3-5, the proposed MA method reaches all optimal solutions, previously obtained by CPLEX 12.1 solver. Average gap and standard deviation are 0%, which means that the MA produced optimal solution in each run. Comparing columns "Optimal" and "LP-LM", we have noticed several differences in optimal solution values obtained by CPLEX and solution values of the LP-LM method from [32]. These differences occur for instances: GapA09, GapA23, GapB30, GapC09, GapC17, GapC20, GapC22 and GapC24. In these cases, solution values obtained by CPLEX 12.1 (and later confirmed by the MA) have lower values than the ones from [32]. Since we have generated the instances from GapA, GapB and GapC data set by using the generator code obtained from the authors of [32], it is possible that in some cases the generated instances do

Table 3. Results and comparisons on GapA instances

Instance		LP-LM		Memetic algorithm				
Name	**Optimal**	***Sol.***	***t[s]***	MA_{best}	***t*[s]**	***Gen***	***ag*[%]**	**σ[%]**
GapA01	43148	Opt	78	Opt	16.1	16663.9	0.000	0.000
GapA02	45137	Opt	37	Opt	23.5	24555.7	0.000	0.000
GapA03	45152	Opt	284	Opt	26.3	27271.7	0.000	0.000
GapA04	48155	Opt	67	Opt	19.0	19911.9	0.000	0.000
GapA05	42124	Opt	115	Opt	25.8	26666.9	0.000	0.000
GapA06	48135	Opt	156	Opt	21.2	22022.7	0.000	0.000
GapA07	45148	Opt	127	Opt	19.3	20106.8	0.000	0.000
GapA08	42169	Opt	59	Opt	25.7	26587.0	0.000	0.000
GapA09	45177	54150	70	Opt	28.4	29642.9	0.000	0.000
GapA10	45145	Opt	71	Opt	15.0	15703.4	0.000	0.000
GapA11	42105	Opt	100	Opt	28.1	29278.4	0.000	0.000
GapA12	42166	Opt	181	Opt	21.0	21675.5	0.000	0.000
GapA13	42203	Opt	190	Opt	20.2	20871.7	0.000	0.000
GapA14	42177	Opt	81	Opt	14.5	15035.3	0.000	0.000
GapA15	52149	Opt	53	Opt	22.2	23172.5	0.000	0.000
GapA16	48142	Opt	105	Opt	16.6	17185.8	0.000	0.000
GapA17	42165	Opt	38	Opt	26.5	27618.4	0.000	0.000
GapA18	42178	Opt	46	Opt	18.3	19123.2	0.000	0.000
GapA19	42149	Opt	116	Opt	18.2	18996.6	0.000	0.000
GapA20	45175	Opt	142	Opt	15.6	16210.5	0.000	0.000
GapA21	43183	Opt	97	Opt	21.0	22024.0	0.000	0.000
GapA22	42137	Opt	126	Opt	22.6	23346.9	0.000	0.000
GapA23	42133	42139	67	Opt	18.6	19429.1	0.000	0.000
GapA24	42168	Opt	97	Opt	24.1	25055.5	0.000	0.000
GapA25	42180	Opt	53	Opt	17.0	17696.0	0.000	0.000
GapA26	42169	Opt	52	Opt	24.6	25444.9	0.000	0.000
GapA27	45170	Opt	32	Opt	17.8	18629.0	0.000	0.000
GapA28	42146	Opt	86	Opt	15.8	16372.4	0.000	0.000
GapA29	43205	Opt	36	Opt	22.6	23470.4	0.000	0.000
GapA30	45138	Opt	130	Opt	23.1	23970.2	0.000	0.000
		Average:	**96.4**	**Average**	**21.0**	**21791.3**	**0.000**	**0.000**

not coincide with the ones from [32]. The original set of instances used in [32] is no longer available, and therefore, we couldn't identify the reason why these differences occur.

Regarding CPU times, the average running time of the proposed MA is 21.0 seconds for the GapA data set, 20.9 seconds for the GapB instances and 20.8 seconds for the GapC data set. The average running times of the LP-LM method are 96.4 seconds, 69.9 seconds and 171.8 seconds for GapA, GapB and GapC data set, respectively. According to SPEC fp2006 benchmarks (*www:spec:org*), the machine used for computational experiments with the LP-LM method in [32] has around two times slower performance compared to the configuration used for MA tests in this study. Having in ming these differences, we may conclude that the proposed MA is more than two times faster than the LP-LM method, since the average CPU time on GapA, GapB and GapC data set is 20.9 seconds for the MA and 112.7 seconds for the LP-LM.

In order to evaluate the MA's performance for larger dimensions of the TSUFLP, we have generated the set of 120 test instances, starting from standard ORLIB instances [4] and M^* instances [49]. These instances were initially designed for solving the well-known Uncapacitated Facility Location Problem (UFLP), or its capacitated variant. We have used 15 ORLIB instances cap, capa, capb, capc and 22 M^* instances to derive medium, large and large-scale instances for the first variant of the TSUFLP from [12].

For each UFLP instance with $|N|$ locations of terminals and $|C|$ potential locations for concentrators, we have generated several TSFULP instances with different number of potential locations for concentrators on the first and second levels, $|M|$ and $|K|$, respectively. The numbers $|M|$ and $|K|$ were chosen such that that $|M| + |K| = |C|$. The set C of potential locations for concentrators in an UFLP

Table 4. Results and comparisons on GapB instances

Instance		LP-LM		Memetic algorithm				
Name	**Optimal**	***Sol.***	***t[s]***	MA_{best}	***t[s]***	***Gen***	***ag*[%]**	**σ[%]**
GapB01	52213	Opt	72	Opt	22.1	22821.1	0.000	0.000
GapB02	51150	Opt	69	Opt	28.0	29119.6	0.000	0.000
GapB03	54167	Opt	133	Opt	19.8	20513.4	0.000	0.000
GapB04	66158	Opt	89	Opt	16.0	16828.7	0.000	0.000
GapB05	53126	Opt	98	Opt	17.9	18683.4	0.000	0.000
GapB06	58151	Opt	34	Opt	16.5	17311.0	0.000	0.000
GapB07	52140	Opt	19	Opt	20.0	20741.0	0.000	0.000
GapB08	51157	Opt	122	Opt	26.4	27316.9	0.000	0.000
GapB09	59121	Opt	73	Opt	13.3	13833.7	0.000	0.000
GapB10	45139	Opt	47	Opt	28.2	29238.2	0.000	0.000
GapB11	53166	Opt	100	Opt	15.1	15687.4	0.000	0.000
GapB12	49192	Opt	82	Opt	19.5	20367.5	0.000	0.000
GapB13	55154	Opt	49	Opt	25.3	26408.5	0.000	0.000
GapB14	56147	Opt	61	Opt	26.1	27088.8	0.000	0.000
GapB15	51147	Opt	104	Opt	23.3	24288.5	0.000	0.000
GapB16	52221	Opt	48	Opt	24.3	25376.3	0.000	0.000
GapB17	52132	Opt	63	Opt	14.9	15609.4	0.000	0.000
GapB18	52230	Opt	34	Opt	23.8	24607.5	0.000	0.000
GapB19	56125	Opt	77	Opt	22.2	23191.2	0.000	0.000
GapB20	53150	Opt	65	Opt	23.1	24046.5	0.000	0.000
GapB21	48156	Opt	28	Opt	15.4	16067.3	0.000	0.000
GapB22	48162	Opt	76	Opt	15.9	16717.6	0.000	0.000
GapB23	49152	Opt	91	Opt	21.1	21918.5	0.000	0.000
GapB24	53128	Opt	41	Opt	27.0	28091.2	0.000	0.000
GapB25	45119	Opt	42	Opt	18.6	19401.3	0.000	0.000
GapB26	54136	Opt	93	Opt	14.5	15138.2	0.000	0.000
GapB27	48174	Opt	65	Opt	22.7	23586.2	0.000	0.000
GapB28	52137	Opt	112	Opt	18.4	19297.6	0.000	0.000
GapB29	48196	Opt	79	Opt	27.0	27917.9	0.000	0.000
GapB30	51165	51168	32	Opt	20.6	21398.0	0.000	0.000
		Average:	**69.9**	**Average:**	**20.9**	**21753.7**	**0.000**	**0.000**

instance was divided into two subsets: locations on the first and the second levels. Locations to be placed on each of the two levels were chosen randomly from the set C in the original UFLP instance. The costs of installing concentrators and assignment costs of a terminal $i \in M$ to an installed concentrator at location $j \in M$ are taken from the original UFLP instance. We have also generated the assignment costs of a first-level concentrator at $j \in M$ to a s econdlevel concentrator at $k \in K$, such that possible deviations from the triangle inequality are relatively small. The generator of the assignment costs is presented in Algorithm 8.

Note that $CostNM_{ij}$ and $CostNK_{ik}$ are the costs of assignment of a terminal at location $i \in N$ to a first-level concentrator at location $j \in J$ and a second-level concentrator at location $k \in K$, respectively. Here

Algorithm 8 Generation Assignment costs

for all $j \in M$ **do**
  for all $k \in K$ **do**
    $min \leftarrow \infty$
    $max \leftarrow 0$
    for all $i \in N$ **do**
      if $CostNM_{ij} + CostNK_{ik} < min$ **then**
        $min \leftarrow CostNM_{ij} + CostNK_{ik}$
      end if
      if $CostNM_{ij} + CostNK_{ik} > max$ **then**
        $max \leftarrow CostNM_{ij} + CostNK_{ik}$
      end if
    end for
    $u \leftarrow (max + \min)/2$
    $\boldsymbol{CostMK_{ik}} \in \boldsymbol{U}[\mathbf{0.65}\boldsymbol{u}, \mathbf{1.35}\boldsymbol{u}]$
  end for
end for

Table 5. Results and comparisons on GapC instances

Instance		LP-LM		Memetic algorithm				
Name	**Optimal**	***Sol.***	***t[s]***	MA_{best}	***t[s]***	***Gen***	***ag[%]***	***σ[%]***
GapC01	42231	Opt	118	Opt	27.7	28733.7	0.000	0.000
GapC02	42193	Opt	385	Opt	25.7	26601.5	0.000	0.000
GapC03	45129	Opt	264	Opt	25.2	26118.5	0.000	0.000
GapC04	45162	Opt	77	Opt	17.2	17832.0	0.000	0.000
GapC05	48113	Opt	185	Opt	15.3	15795.6	0.000	0.000
GapC06	45154	Opt	85	Opt	24.8	25743.5	0.000	0.000
GapC07	45103	Opt	80	Opt	24.1	24956.8	0.000	0.000
GapC08	48126	Opt	80	Opt	26.1	26994.2	0.000	0.000
GapC09	48106	48125	161	Opt	21.4	22092.7	0.000	0.000
GapC10	42183	Opt	178	Opt	13.4	14040.0	0.000	0.000
GapC11	47154	Opt	290	Opt	17.4	17968.6	0.000	0.000
GapC12	45126	Opt	347	Opt	25.0	26014.8	0.000	0.000
GapC13	45134	Opt	102	Opt	24.4	25506.0	0.000	0.000
GapC14	42137	Opt	303	Opt	13.5	14047.5	0.000	0.000
GapC15	42129	Opt	356	Opt	14.5	15022.5	0.000	0.000
GapC16	45143	Opt	112	Opt	14.6	15141.0	0.000	0.000
GapC17	51169	51191	42	Opt	21.3	22093.9	0.000	0.000
GapC18	47191	Opt	228	Opt	13.8	14519.1	0.000	0.000
GapC19	45154	Opt	111	Opt	20.8	21754.0	0.000	0.000
GapC20	42133	45106	205	Opt	25.8	26786.0	0.000	0.000
GapC21	42143	Opt	159	Opt	13.6	14090.2	0.000	0.000
GapC22	45158	45161	176	Opt	14.7	15465.4	0.000	0.000
GapC23	42169	Opt	43	Opt	24.6	25511.4	0.000	0.000
GapC24	48127	52185	11	Opt	28.4	29452.8	0.000	0.000
GapC25	45144	Opt	225	Opt	13.0	13427.9	0.000	0.000
GapC26	45164	Opt	196	Opt	20.1	20834.4	0.000	0.000
GapC27	42162	Opt	60	Opt	26.7	27814.6	0.000	0.000
GapC28	47211	Opt	215	Opt	28.5	29433.5	0.000	0.000
GapC29	42128	Opt	230	Opt	24.6	25683.0	0.000	0.000
GapC30	39177	Opt	130	Opt	18.2	18897.7	0.000	0.000
		Average:	**171.8**	**Average:**	**20.8**	**21612.4**	**0.000**	**0.000**

$U[0.65u, 1.35u]$ denotes a uniform distribution on the interval $[0.65u, 1.35u]$, where $u = (max + min)/2$, $max = \max\{CostNM_{ij} + CostNK_{ik} \mid i \in N, j \in M, k \in K\}$, and $min = \min\{CostNM_{ij} + CostNK_{ik} | i \in N, j \in M, k \in K\}$.

Table 6 shows the results of CPLEX 12.1 solver and the proposedMAapproach on medium size instances with $|N| \leq 100, |M| \leq 92$ and $|K| \leq 45$. For larger problem dimensions, CPLEX 12.1 was unable to provide any solutions within 3h of running time, which was imposed time limit. Therefore, in Tables 7 and 8 we present the MA results only.

Column headings in Tables 6-8 mean:

- The identifier assigned to a considered TSUFLP instance;
- The name of original ORLIB instance used for generating a TSFULP instance;
- Instance's parameters: the number of terminal nodes – $|N|$, the number of potential locations for concentrators on the first level $|M|$ and the number of potential locations for concentrators on the second level – $|K|$;
- Optimal solution – *Optimal* obtained by the CPLEX 12.1 solver;
- Total CPLEX 12.1 running time – *t[s]*, in seconds;
- Number of nodes of the CPLEX 12.1 solver – *Nodes*;
- The last five columns contain data related to the proposed MA method, which are presented in the same way as in Tables 3-5.

From the results presented in Table 6, it can be seen that the proposed MA method quickly reached all optimal solutions on medium size instances, previously solved by CPLEX 12.1 solver. In average, CPLEX 12.1 needed 170.073 seconds to solve problem instances to optimality, while the average number of CPLEX nodes was 2480.9. The proposed MA method was around 24.5 times faster, since it needed 6.931 seconds in average to reach optimal

Table 6. Results of CPLEX and MA on medium size TSUFLP instances

Instance					CPLEX			Memetic algorithm				
No	**Instance**	$\|N\|$	$\|M\|$	$\|K\|$	***Optimal***	***t[s]***	***Nodes***	MA_{best}	***t[s]***	***Gen***	***ag[%]***	***σ[%]***
A01	cap71	50	12	4	2267611.700	0.291	0	Opt	0.015	5.1	0.000	0.000
A02	cap72	50	11	5	2149399.500	0.222	2	Opt	0.013	2.7	0.000	0.000
A03	cap73	50	10	6	1235119.356	0.014	0	Opt	0.014	2.6	0.000	0.000
A04	cap74	50	9	7	1235119.356	0.014	0	Opt	0.012	1.5	0.000	0.000
A05	cap101	50	20	5	1876623.606	0.870	7	Opt	0.034	14.1	0.000	0.000
A06	cap102	50	18	7	2150745.494	0.847	15	Opt	0.039	19.2	0.000	0.000
A07	cap103	50	16	9	2561284.738	0.932	27	Opt	0.032	13.0	0.000	0.000
A08	cap104	50	14	11	2727064.050	0.381	0	Opt	0.016	6.6	0.000	0.000
A09	cap131	50	45	5	1923278.219	1.748	26	Opt	0.125	66.4	0.000	0.000
A10	cap131	50	42	8	1773278.219	1.824	41	Opt	0.106	54.4	0.000	0.000
A11	cap132	50	40	10	2395821.538	2.264	19	Opt	0.184	91.7	0.000	0.000
A12	cap132	50	37	13	1991443.725	2.921	15	Opt	0.051	20.6	0.000	0.000
A13	cap133	50	35	15	2241911.562	4.770	19	Opt	0.091	47.5	0.000	0.000
A14	cap133	50	33	17	2634635.669	6.823	81	Opt	0.156	67.4	0.000	0.000
A15	cap134	50	30	20	2707135.669	5.175	53	Opt	0.081	39.0	0.000	0.000
A16	cap134	50	28	22	3216485.699	9.260	57	Opt	0.121	57.0	0.000	0.000
A17	MO1	100	92	8	4226.420	362.740	5567	Opt	12.108	5327.5	0.000	0.000
A18	MO1	100	90	10	7671.286	745.998	14803	Opt	8.701	3978.2	0.000	0.000
A19	MO1	100	88	12	4834.337	299.432	4007	Opt	15.411	7198.1	0.000	0.000
A20	MO1	100	86	14	6050.172	597.185	11686	Opt	4.536	2101.5	0.000	0.000
A21	MO1	100	85	15	5492.915	537.595	8615	Opt	1.790	805.2	0.000	0.000
A22	MO2	100	84	16	7416.173	412.653	3771	Opt	4.161	1956.8	0.000	0.000
A23	MO2	100	81	19	4075.441	366.833	4885	Opt	7.211	3422.2	0.000	0.000
A24	MO2	100	80	20	5023.186	208.507	3651	Opt	3.367	1537.9	0.000	0.000
A25	MO2	100	77	23	7079.032	431.622	3693	Opt	5.201	2406.6	0.000	0.000
A26	MO2	100	76	24	5218.749	327.743	2884	Opt	10.924	5078.8	0.000	0.000
A27	MO3	100	75	25	6669.206	52.359	307	Opt	9.416	4359.9	0.000	0.000
A28	MO3	100	73	27	6690.544	244.677	2325	Opt	4.271	1844.2	0.000	0.000
A29	MO3	100	72	28	6166.135	197.369	3200	Opt	4.330	1843.3	0.000	0.000
A30	MO3	100	70	30	7681.095	170.932	2230	Opt	16.550	7286.3	0.000	0.000
A31	MO3	100	69	31	8370.142	531.512	4650	Opt	8.647	3726.4	0.000	0.000
A32	MO4	100	67	33	4963.441	284.063	6992	Opt	7.060	3123.6	0.000	0.000
A33	MO4	100	65	35	4963.441	181.151	2223	Opt	7.844	3466.0	0.000	0.000
A34	MO4	100	64	36	5863.441	182.198	2519	Opt	11.846	5153.1	0.000	0.000
A35	MO4	100	63	37	6263.441	263.618	3466	Opt	4.689	2017.6	0.000	0.000
A36	MO4	100	61	39	6320.795	362.340	8246	Opt	28.201	12142.8	0.000	0.000
A37	MO5	100	60	40	7077.742	34.436	122	Opt	4.707	1925.5	0.000	0.000
A38	MO5	100	59	41	6335.654	47.105	452	Opt	13.445	5808.6	0.000	0.000
A39	MO5	100	58	42	5085.654	26.115	422	Opt	31.745	14271.0	0.000	0.000
A40	MO5	100	56	44	3985.654	24.653	245	Opt	30.107	13041.2	0.000	0.000
A41	MO5	100	55	45	7285.654	41.791	396	Opt	26.822	11506.0	0.000	0.000
					Average:	170.073	2480.9	Average	6.931	3069.2	0.000	0.000

solutions. The maximal CPU time of the MA was 31.745 seconds, in the case of instance A39 ($|N| = 100$, $|M| = 55$ and $|K| = 45$). In average, the MA needed 3069.2 generations before a stopping criterion was reached.

We further considered larger instances with $|N| \leq 500$ demand nodes, $|M| \leq 400$ potential locations for concentrators on the first level, and $|K| \leq 200$ potential locations for concentrators on the second level. The CPLEX 12.1 was unable to solve these instances, due to memory or time limits. From the results presented in Table 7, it can be seen that larger TSUFLP instances were solved successfully by the proposed MA in short CPU times. The maximal MA running time was 378.143 seconds for instance B39 ($|N| = 500$, $|M| = 300$ and $|K| = 200$). On the set of larger problem instances, our MA needed 152.986 seconds and 152.986 generations (in average)

Table 7. The results of the MA on larger TSFULP instances

Instance					Memetic algorithm				
No	**Instance**	$\|N\|$	$\|M\|$	$\|K\|$	MA_{best}	*t[s]*	*Gen*	*ag[%]*	*σ[%]*
C01	capa	1000	94	6	152945133.744	91.335	10068.8	0.000	0.000
C02	capa	1000	93	7	136474427.354	67.619	6859.1	0.000	0.000
C03	capa	1000	92	8	115073096.128	95.159	10672.4	0.000	0.000
C04	capa	1000	91	9	111702081.201	122.663	12660.2	0.000	0.000
C05	capa	1000	90	10	119115244.508	76.553	7474.2	0.000	0.000
C06	capa	1000	88	12	100665269.697	57.162	6639.7	0.000	0.000
C07	capa	1000	86	14	117523041.478	96.733	9609.3	0.000	0.000
C08	capa	1000	84	16	102031208.697	82.252	9584.0	0.000	0.000
C09	capb	1000	82	18	52924102.150	85.948	9878.8	0.000	0.000
C10	capb	1000	80	20	53080056.537	71.645	8390.8	0.000	0.000
C11	capb	1000	78	22	51753302.557	55.327	6723.9	0.000	0.000
C12	capb	1000	76	24	48896026.614	93.211	11193.1	0.000	0.000
C13	capb	1000	74	26	48700811.081	51.138	6319.8	0.000	0.000
C14	capb	1000	72	28	41783670.435	77.026	9710.0	0.000	0.000
C15	capb	1000	70	30	41828511.147	97.874	12448.5	0.000	0.000
C16	capb	1000	68	32	48580219.8475	66.427	8331.4	0.000	0.000
C17	capc	1000	66	34	33337201.9823	62.567	8126.7	0.000	0.000
C18	capc	1000	64	36	32338771.527	88.620	11515.2	0.000	0.000
C19	capc	1000	62	38	35266649.335	98.142	13034.8	0.000	0.000
C20	capc	1000	60	40	31409338.593	70.213	9624.6	0.000	0.000
C21	capc	1000	58	42	34319409.101	104.404	13993.2	0.000	0.000
C22	capc	1000	56	44	34443152.363	111.726	15191.9	0.000	0.000
C23	capc	1000	54	46	30327868.400	86.301	11983.5	0.000	0.000
C24	capc	1000	52	48	30339454.019	78.094	8673.7	0.000	0.000
C25	MS1	1000	980	20	41970.316	517.905	27855.2	0.038	0.049
C26	MS1	1000	970	30	30644.2440	530.684	29686.4	0.000	0.000
C27	MS1	1000	950	50	36743.604	307.328	17226.2	0.000	0.000
C28	MS1	1000	900	100	32281.987	566.629	30701.5	0.000	0.000
C29	MS1	1000	850	150	26816.630	544.257	24221.1	0.025	0.023
C30	MS1	1000	800	200	31750.216	649.724	31738.4	0.000	0.000
C31	MS1	1000	750	250	226233.784	871.152	39109.0	0.000	0.000
C32	MS1	1000	700	300	258529.730	1432.409	59791.9	0.000	0.000
C33	MS1	1000	650	350	260159.418	1383.366	54994.7	0.129	0.230
C34	MS1	1000	600	400	220624.399	1417.818	58312.2	0.046	0.081
C35	MT1	2000	1900	100	73184.559	2385.430	35875.3	0.000	0.000
C36	MT1	2000	1700	300	47816.891	2324.550	27960.7	0.022	0.021
C37	MT1	2000	1500	500	696619.483	2144.032	24985.5	0.014	0.026
C38	MT1	2000	1300	700	705656.536	1836.424	20776.6	0.148	0.189
C39	MT1	2000	1100	900	720792.910	1943.759	20374.2	0.063	0.044
					Average:	**534.451**	**19033.8**	**0.012**	**0.017**

to produce solutions. Standard deviation and average gap were 0:0%, which indicates good stability of the proposed MA.

Regarding the fact that large-scale instances often appear in practice, we have generated 39 instances with $|N| = 1000$, $|M| \leq 980$, $|K| \leq 400$ and $|N| = 2000, |M| \leq 1900, |K| \leq 900$. In Table 8, we present the MA results on the newly generated, large-scale data set. The obtained results clearly indicate the efficiency of the proposed MA approach when solving problem instances of real-life dimensions. The average MA running time was 534.451 seconds, while the average number of MA generations was 19033.8. Low values of average gap (0.012%) and Standard deviation (0.017%) indicate the stability of the MA method on the considered large-scale data set. Although the optimality of the best MA solutions can not be proven, we believe that the proposed MA produced high quality solutions on TSFULP instances of large and large-scale dimensions.

Table 8. The results of the MA on large-scale TSFULP instances

Instance					Memetic algorithm				
No	**Instance**	$\|N\|$	$\|M\|$	$\|K\|$	MA_{best}	*t[s]*	*Gen*	*ag[%]*	*σ[%]*
C01	capa	1000	94	6	152945133.744	91.335	10068.8	0.000	0.000
C02	capa	1000	93	7	136474427.354	67.619	6859.1	0.000	0.000
C03	capa	1000	92	8	115073096.128	95.159	10672.4	0.000	0.000
C04	capa	1000	91	9	111702081.201	122.663	12660.2	0.000	0.000
C05	capa	1000	90	10	119115244.508	76.553	7474.2	0.000	0.000
C06	capa	1000	88	12	100665269.697	57.162	6639.7	0.000	0.000
C07	capa	1000	86	14	117523041.478	96.733	9609.3	0.000	0.000
C08	capa	1000	84	16	102031208.697	82.252	9584.0	0.000	0.000
C09	capb	1000	82	18	52924102.150	85.948	9878.8	0.000	0.000
C10	capb	1000	80	20	53080056.537	71.645	8390.8	0.000	0.000
C11	capb	1000	78	22	51753302.557	55.327	6723.9	0.000	0.000
C12	capb	1000	76	24	48896026.614	93.211	11193.1	0.000	0.000
C13	capb	1000	74	26	48700811.081	51.138	6319.8	0.000	0.000
C14	capb	1000	72	28	41783670.435	77.026	9710.0	0.000	0.000
C15	capb	1000	70	30	41828511.147	97.874	12448.5	0.000	0.000
C16	capb	1000	68	32	48580219.8475	66.427	8331.4	0.000	0.000
C17	capc	1000	66	34	33337201.9823	62.567	8126.7	0.000	0.000
C18	capc	1000	64	36	32338771.527	88.620	11515.2	0.000	0.000
C19	capc	1000	62	38	35266649.335	98.142	13034.8	0.000	0.000
C20	capc	1000	60	40	31409338.593	70.213	9624.6	0.000	0.000
C21	capc	1000	58	42	34319409.101	104.404	13993.2	0.000	0.000
C22	capc	1000	56	44	34443152.363	111.726	15191.9	0.000	0.000
C23	capc	1000	54	46	30327868.400	86.301	11983.5	0.000	0.000
C24	capc	1000	52	48	30339454.019	78.094	8673.7	0.000	0.000
C25	MS1	1000	980	20	41970.316	517.905	27855.2	0.038	0.049
C26	MS1	1000	970	30	30644.2440	530.684	29686.4	0.000	0.000
C27	MS1	1000	950	50	36743.604	307.328	17226.2	0.000	0.000
C28	MS1	1000	900	100	32281.987	566.629	30701.5	0.000	0.000
C29	MS1	1000	850	150	26816.630	544.257	24221.1	0.025	0.023
C30	MS1	1000	800	200	31750.216	649.724	31738.4	0.000	0.000
C31	MS1	1000	750	250	226233.784	871.152	39109.0	0.000	0.000
C32	MS1	1000	700	300	258529.730	1432.409	59791.9	0.000	0.000
C33	MS1	1000	650	350	260159.418	1383.366	54994.7	0.129	0.230
C34	MS1	1000	600	400	220624.399	1417.818	58312.2	0.046	0.081
C35	MT1	2000	1900	100	73184.559	2385.430	35875.3	0.000	0.000
C36	MT1	2000	1700	300	47816.891	2324.550	27960.7	0.022	0.021
C37	MT1	2000	1500	500	696619.483	2144.032	24985.5	0.014	0.026
C38	MT1	2000	1300	700	705656.536	1836.424	20776.6	0.148	0.189
C39	MT1	2000	1100	900	720792.910	1943.759	20374.2	0.063	0.044
					Average:	**534.451**	**19033.8**	**0.012**	**0.017**

Computational results presented in Tables 3–8 indicate that the proposed MA represents an efficient metaheuristic for solving the TSUFLP. The MA reached all known optimal solutions on smaller and medium-size test instances that were previously solved by the CPLEX 12.1 or the LPLM method. Average gap and standard deviation through all MA runs were 0%, which proves the reliability of the MA in producing optimal solutions. Based on presented results, we believe that the MA would also reach optimal solutions when solving instances that differ from the ones considered here (if optimal solutions can be found). On larger and large-scale data sets, optimal solutions are not known, and therefore, it is difficult to provide quality guarantees of the obtained solutions. The stability of the MA on these instances, reflected in low values of average gap and Standard deviation, as well as the efficiency and reliability in providing optimal solutions for lower dimensions of the problem, may indicate that the MA has produced high-quality solutions on large and large-scale data set. Future work may be directed in hybridizing the

proposed MA with some exact method for the TSUFLP, in order to obtain optimal solutions on larger problem instances and to give some quality guarantees of the solutions. There is also possibility to modify the proposed MA for solving the multi-level variant of the considered problem, which is another direction for future work.

5. Conclusions

In this paper we present a memetic algorithm MA for solving the two variants of the two-stage uncapacitated facility location problem. The proposed MA is based on incorporation of two efficient local search procedure into an evolutionary algorithm frame, in order to improve the solution's quality. The first local-search is based on inversion procedure, while the second one performs transposition of randomly chosen genes in an individual's genetic code. Several strategies are additionally applied in order to improve the MA's performance.

The proposed MA method was subject to a broad set of computational experiments on smaller size problem instances from the literature and on newly generated benchmark set of medium, larger and large-scale instances. The MA method showed to be very efficient in returning optimal solutions, previously obtained by CPLEX solver, on smaller and medium size problem instances. For instances of larger and real-life dimensions, which couldn't be solved by CPLEX, the proposed MA provided solutions efficiently. The average MA's running time was less than 9 minutes for the largest considered TSUFLP instances with 1000 and 2000 demand nodes.

Good performance of the proposed MA on the considered benchmark data set makes it a valuable tool for solving the TSUFLP, especially large-scale problem instances. The investigation of possible use of the MA approach in other applications needs further research and may lead to the development of more advanced memetic algorithms.

Acknowledgement

This research was partially supported by Serbian Ministry of Science and Technological Development under the grants no. 174010 and 47017. We are thankful to M. Landete and A. Marin for providing us the code for generating modified GapA, GapB and GapC instances.

References

[1] **B. M. Baker, M. A. Ayechew.** A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, 2003, Vol. 30, pp. 787–800.

[2] **J. Balakrishnan, C. H. Cheng.** Genetic search and the dynamic layout problem. *Computers & Operations Research*, 2000, Vol. 27, pp. 587–593.

[3] **A. I. Barros, R. Dekker, V. Scholten.** A two-level network for recycling sand: a case study. *European Journal of Operational Research*, 1998, Vol. 110, pp. 199-214.

[4] **J. E. Beasley.** Obtaining test problems via internet. *Journal of Global Optimization*, 2006, Vol. 8, pp. 429–433.

[5] **C. Blum, J. Puchingerb, G. R. Raidl, A. Roli.** Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 2011, Vol. 11, pp. 4135–4151.

[6] **M. Boccia, T. G. Crainic, A. Sforza, C. Sterle.** A metaheuristic for a t wo echelon location-rounting problem. In: *Procceding of SEA2010. Lecture Notes in Computer Science*, 2010, Vol. 6049, pp. 288–301.

[7] **B. Bontoux, C. Artigues, D. Feillet.** A Memetic. algorithm with a large neighborhood crossover operator for the generalized traveling salesman problem. *Computers and Operations Research*, 2010, Vol. 37, pp. 1844-1852.

[8] **E. C. Brown, R. T. Sumichrast.** Impact of the replacement heuristic in a grouping genetic algorithm. *Computers & Operations Research*, 2003, Vol. 30, pp. 1575–1593.

[9] **C. Canel, B. Khumawala, J. Law, A. Loh.** An algorithm for the capacitated, multi-commodity, multi-period facility location Problem. *Computers and Operations Research*, 2001, Vol. 28, pp. 411–427.

[10] **D. Capko, A. Erdeljan, S. Vukmirović, I. Lendak**. A hybrid genetic algorithm for partitioning of data model in distribution management systems. *Information Technology and Control*, 2011, Vol. 40, No. 4, pp. 316–322.

[11] **M. Eben-Chaime, A. Mehrez, G. Markovich.** Capacitated location-allocation problems on a line. *Computers and Operations Research*, 2002, Vol. 29, pp. 459–470.

[12] **P. Chardaire, J. L. Luton, R. Sutter.** Upper and lower bounds for the two-level simple plant location problem. *Annals of Operations Research*, 1999, Vol. 86, pp. 117–140.

[13] **Y. W. Chen, Y. Z. Lu, G. K. Yang.** Hybrid evolutionary algorithm with marriage of genetic algorithm and extremal optimization for production scheduling. In: *The International Journal of Advanced Manufacturing Technology*, 2008, Vol. 36, No. 9–10, pp. 959–968.

[14] **T. C. Chiang, L. C. Fu.** A rule-centric memetic algorithm to minimize the number of tardy jobs in the job shop. *International Journal of Production* Research, 2008, Vol. 46, pp. 6913–6931.

[15] **S. Chung, Y. Myung, D. Tcha.** Optimal design of a distributed network with a two-level hierarchical structure. *European Journal of Operations Research*, 1992, Vol. 62, pp. 105–115.

[16] **J. Current, H. Pirkul.** The hierarchical network design problem with transshipment facilities. *European Journal of Operation Research*, 1991, Vol. 52, pp. 338–347.

[17] **M. S. Daskin.** Network and discrete location models. *Algorithms and Applications. John Wiley & Sons*, 1995.

[18] **J. Dias, M. Captivo, J. Clímaco.** A memetic algorithm for dynamic location problems. *Operations Research and Computer Science Interfaces Series*, 2007, Vol. 39, pp. 225–244.

[19] **J. Dias, M.E. Captivo, J. Clímaco.** Dynamic multi-level capacitated and uncapacitated location problems: an approach using primal-dual heuristics. In: *Operational Research. An International Journal*, 2008, Vol. 7, pp. 345–379.

[20] **Z. Drezner.** Extensive experiments with hybrid genetic algorithms for the solution of the quadratic assignment problem. In: *Computers & Operations Research*, 2008, Vol. 35, pp. 717–736.

[21] **V. Filipović.** Fine-grained tournament selection operator in genetic algorithms. In: *Computing and Informatics*, 2003, Vol. 22, pp. 143–161.

[22] **R. D. Galvao, L. G. Espejo, B. Boffey.** A hierarchical model for the location of perinatal facilities in the municipality of Rio de Janeiro. In: *European Journal of Operational Research*, 2002, Vol. 138, pp. 495– 517.

[23] **R. D. Galvao, L. G. Espejo, B. Boffey, D. Yates.** Load balancing and capacity constraints in a hierarchical location model. In: *European Journal of Operational Research*, 2006, Vol. 172, pp. 631–646.

[24] **E. Gourdin, M. Labb, H. Yaman.** Telecommunication and location. In: *Z. Drezner, H.W. Hamacher (eds). Facility location: applications and theory. Springer-Verlag, New York*, 2002, pp. 274–305.

[25] **L. Gao, G. Zhang, L. Zhang, X. Li.** An efficient memetic algorithm for solving the job shop scheduling problem. *Computers and Industrial Engineering*, 2011, Vol. 60, pp. 699–705.

[26] **S. A. Helm, M. A. Venkataramanan.** Solution approaches to hub location problems. *Annals of Operations Research*, 1998, Vol. 78, pp. 31–50.

[27] **M. P. Helme, T. L. Magnanti.** Designing satelite communication networks by zero-one quadratic programming. *Networks*, 1999, Vol. 19, pp. 427–450.

[28] **Y. Hinojosa, J. Puerto, F. R. Fernndez.** A multiperiod two-echelon multicommodity capacitated plant location problem. *European Journal of Operational Research*, 2000, Vol. 123, pp. 271–291.

[29] **N. Jawahar, A. N. Balaji.** A genetic algorithm for the two-stage supply chain distribution problem associated with a fixed charge. *European Journal of Operational Research*, 2009, Vol. 194, pp. 496–537.

[30] **M. Khouja, Z. Michalewicz, M. Wilmot.** The use of genetic algorithms to solve the economic lot size scheduling problem. In: *European Journal of Operational Research*, 1998, Vol. 110, pp. 509–524.

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

[32] **M. Landete, A. Marin.** New facets for the two-stage uncapacitated facility location polytope. *Computational Optimization and Applications*, 1999, Vol. 44, pp. 487–519.

[33] **Y. K. Lin, C. T. Yeh.** Optimal resource assignment to maximize multi state network reliability for a computer network. *Computers & Operations Research*, 2010, Vol. 37, pp. 2229–2238.

[34] **Z. L. F. Glover, J. K. Hao.** A hybrid metaheuristic approach to solving the UBQP problem. *European Journal of Operational Research*, 2010, Vol. 207, pp. 1254–1262.

[35] **M. Marić, Z. Stanimirović, S.Božović.** Hybrid metaheuristic method for determining locations for long-term health care facilities. *Annals of Operations Research*, 2013, DOI: 10.1007/s10479-013-1313-8.

[36] **A. Marn, B. Pelegrin.** Applying Lagrangian relaxation to the resolution of two-stage location problems. *Annals of Operations Research*, 1999, Vol. 86, pp. 179–198.

[37] **M. B. Mandell.** Covering models for two-tiered emergency medical services systems. *Location Science*, 1998, Vol. 6, pp. 355–368.

[38] **V. Marianov, P. Taborga.** Optimal location of public health centers which provide free and paid services. *Journal of Operational Research Society*, 2001, Vol. 52, pp. 391–400.

[39] **G. R. Matheus, F. R. B. Cruz, H. P. L. Luna.** An algorithm for hierarchical network design. *Location Science*, 1994, Vol. 2, pp. 149–164.

[40] **E. Melachrinoudis, H. Min.** The dynamic relocation and phase-out of a hybrid, two-echelon plant/warehouse facility: A multiple objective approach. *European Journal of Operational Research*, 2000, Vol. 123, pp. 1–15.

[41] **A. Misevičius, D. Rubliauskas.** Enhanced improvement of individuals in genetic algorithms. *Information Technology and Control*, 2006, Vol. 37, No. 3, pp. 179–186.

[42] **A. Misevičius, D. Rubliauskas, V. Barkauskas.** Some further experiments with the genetic algorithm for the quadratic assignment problem. *Information Technology and Control*, 2009, Vol. 38, No. 4, pp. 325–332.

[43] **G. Moore, C. ReVelle.** The hierarchical service location problem. *Management Science*, 1982, Vol. 28, pp. 775–780.

[44] **S. C. Narula.** Hierarchical location-allocation problems: A classification scheme. *European Journal of Operational Research*, 1984, Vol. 5, pp. 93–99.

[45] **F. Neri, C. Cotta.** Memetic algorithms and memetic computing optimization: A literature review. *Swarm and Evolutionary Computation*, 2012, Vol. 2, pp. 1–14.

[46] **H. Pirkul, V. Nagarajan.** Location concentrators in centralized computer networks. *Annals of Operations Research*, 1992, Vol. 36, pp. 247–262.

[47] **S. Pirkwieser, G. R. Raidl, J. Puchinger.** Combining Lagrangian decomposition with an evolutionary algorithm for the knapsack constrained maximum spanning tree problem, In: *C. Cotta, J.I. van Hemert (Eds.), Evolutionary Computation in Combinatorial Optimization, Springer Berlin Heidelberg*, 2007, pp. 176–187.

[48] **S. Prestwich, S. Tarim, R. Rossi, B. Hnich.** Evolving parameterised policies for stochastic constraint programming, In: *I. Gent (Ed.), Principles and Practice of Constraint Programming–CP. Lecture Notes in Computer Science, Springer-Verlag, Berlin Heidelberg, Germany*, 2009, 5732, pp. 684–691.

[49] **M. G. C.Resende, R. F. Werneck.** A hybrid multistart heuristic for the uncapacitated facility location problem. *European Journal of Operational Research*, 2006, Vol. 174, pp. 54–68.

[50] **G. Sahin, H. Sral, S. Meral.** Locational analysis for regionalization of Turkish Red Crescent blood services. *Computers and Operations Research*, 2007, Vol. 34, pp. 692–704.

[51] **G. Sahin, H. Sral.** A review of hierarchical facility location networks. *Computers and Operations Research*, 2007, Vol. 34, pp. 2310–2331.

[52] **X. H. Shi, Y. C. Liang, H. P. Lee, C. Lu, L. M. Wang.** An improved GA and a novel PSO-GA-

based hybrid algorithm. In: *Information Processing Letters*, 2005, Vol. 93, No. 5, pp. 255–261.

[53] **H. K. Smith, P. R. Harper, C. N. Potts, A. Thyle.** Planning sustainable community health schemes in rural areas of developing countries. *European Journal of Operational Research*, 2009, Vol. 193, pp. 768–777.

[54] **Z. Stanimirović, J. Kratica, Dj. Dugošija.** Genetic algorithms for solving the discrete ordered median problem. *European Journal of Operational Research*, 2007, Vol. 182, pp. 983–1001.

[55] **Z. Stanimirović.** An Efficient Genetic Algorithm for Solving the uncapacitated multiple allocation p-hub median problem. *Control and Cybernetics*, 2008, Vol. 37, pp. 415–426.

[56] **Z. Stanimirović, M. Marić, S. Božović, P. Stanojević.** An efficient evolutionary algorithm for locating long-term care facilities. *Information Technology and Control*, 2012, Vol. 41, No. 1, pp. 77–89.

[57] **D. W. Tcha, B. Lee.** A branch-and-bound algorithm for the multi-level uncapacitated facility location problem. *European Journal of Operational Research*, 1984, Vol. 18, pp. 35–43.

[58] **J. C. Teixeira, A. P. Antunes.** A hierarchical location model for public facility planing. In*: European Journal of Operational Research*, 2008, Vol. 185, pp. 92–104.

[59] **H. Topcuoglu, F. Court, M. Ermis, G. Yilmaz.** Solving the uncapacitated hub location problem using genetic algorithms. *Computers & Operations Research*, 2005, Vol. 32, pp. 967–984.

[60] **A. Troncoso, J. C. Riquelme, J. S. Aguilar-Ruiz, J. M. Riquelme-Santos.** Multi point evolutionary techniques applied to the optimal short-term scheduling of the electrical energy production. *European Journal of Operational Research*, 2008, Vol. 185, pp. 1114–1127.

[61] **T. J. Van Roy.** Multi-level production and distribution planning with transportation fleet optimization. *Management Science*, 1989, Vol. 35, pp. 1443–1453.

[62] **B. L. Wildbore.** Theoretical and computational analysis of the two-stage capacitated location problem. *PhD thesis, Massey University, Palmerston North, New Zeland*, 2008

Received May 2012.