Computing and Informatics, Vol. 29, 2010, 117–132

A GENETIC ALGORITHM APPROACH FOR THE CAPACITATED SINGLE ALLOCATION P-HUB MEDIAN PROBLEM

Zorica Stanimirović

Faculty of Mathematics
University of Belgrade
Studentski trg 16/IV
11001 Belgrade, Serbia
e-mail: zoricast@matf.bg.ac.rs

Manuscript received 11 September 2006; revised 15 October 2007
Communicated by János Fodor

Abstract. In this paper the Capacitated Single Allocation p-Hub Median Problem (CSApHMP) is considered. This problem has a wide range of applications within the design of telecommunication and transportation systems. A heuristic method, based on a genetic algorithm (GA) approach, is proposed for solving the CSApHMP. The described algorithm uses binary encoding and modified genetic operators. The caching technique is also implemented in the GA in order to improve its effectiveness. Computational experiments demonstrate that the GA method quickly reaches optimal solutions for hub instances with up to 50 nodes. The algorithm is also benchmarked on large scale hub instances with up to 200 nodes that are not solved to optimality so far.

Keywords: Evolutionary computation, network optimization, graph and network algorithms, randomized algorithms

1 INTRODUCTION

Hub location problems have wide range of application in the design of transportation and telecommunication networks. Postal delivery systems, air or ground transportation networks, computer and satellite systems are often configured as hub networks. In these networks, the traffic between two nodes is not shipped directly, but has

to be routed via a specified set of nodes. These nodes, denoted as hubs, serve as intermediate switching points for traffic. By employing hubs as consolidation and distribution centers for several nodes, it is possible to eliminate many expensive direct connections (origin–destination) and to make the transportation flow more efficient.

Many variants of hub location problems exist in the literature. For example, it may be assumed that exactly p hubs have to be chosen. The non-hub nodes may be allocated either to a single hub (single allocation scheme) or to multiple hubs (multiple allocation scheme). The total flow through a hub node or the flow collected at a hub can be restricted (capacitated versions). The fixed costs for establishing hubs may also be assumed.

Hub location problems are NP-hard, with the exception of few special cases. They are usually much harder to solve in comparison with other, non-hub location problems. Moreover, there does not exist a general mathematical model that describes well all hub location problems. Each hub problem has its own specific structure: objective function, decision variables and constraints. Few additional constraints or a slight modification of the problem structure can substantially change the computational behavior of the designed solution approach. Therefore, there is no general algorithm for solving all hub problems, or at least a smaller group of them. Exact methods can not provide solutions for large-scale hub location problems which arise from practice in a reasonable amount of time. Therefore, heuristic methods are very promising approaches for solving hub location problems. A detailed review of hub location problems and solution methods for solving them can be found in [5].

In this paper, a particular variant of the hub location problem, known as the Capacitated Single Allocation p-Hub Median Problem (CSApHMP) is studied. This problem deals with a network with n nodes where p hubs have to be established. The traffic between any pair of non-hub nodes is routed via one or two hubs, and the single allocation scheme is assumed. In this model, a limited amount of flow can be collected in a hub (capacity restriction). The goal of CSApHMP is to minimize the sum of the overall transportation costs in the network. The traffic in the network consists of collection (from origin nodes to hubs), transfer (between hubs) and distribution (from hubs to destination nodes). The corresponding transportation costs are multiplied by χ, α and δ respectively, which represent the cost per unit distance. Generally, α is used as a discount factor to provide reduced unit costs on arcs between hubs to reflect economies of scale, so $\alpha < \chi$ and $\alpha < \delta$. The CSApHMP is NP-hard, since its uncapacitated subproblem USApHMP was proved to be NP-hard in [16].

2 MATHEMATICAL FORMULATION

For the CSApHMP the following mixed integer linear programming formulation, taken from [4], is used. Consider a network of n distinct locations (nodes) that

exchange some traffic. Exactly p nodes that will serve as hubs have to be located. Each non-hub node can be allocated to a single hub. The traffic between a pair of nodes i and j must be routed through either one or at most two established hubs k and l. The cost of routing a unit of flow along the path $i \to k \to l \to j$ is calculated as: $\chi \cdot d_{ik} + \alpha \cdot d_{kl} + \delta \cdot d_{lj}$, where d_{ij} denotes the distance (in the sense of a metric) between nodes i and j.

The following notation has been used in the mathematical formulation:

- W_{ij} = the number of units of traffic sent from origin i to destination j,
- Γ_k = the capacity of a hub k,
- O_i = total flow departing from an origin i, i.e. $O_i = \sum_{j=1}^{n} W_{ij}$,
- D_j = total flow distributed to destination j, i.e. $D_j = \sum_{i=1}^{n} W_{ij}$.

The decision variables are:

- $Z_{ij} = 1$, if node i is allocated to hub j, 0 otherwise,
- Y_{kl}^{i} = the amount of flow from origin i that is collected at hub k and distributed via hub l.

Using the above notation, the CSApHMP can be written as:

$$\min \sum_{i=1}^{n} \sum_{k=1}^{n} d_{ik} Z_{ik} (\chi O_i + \delta D_i) + \sum_{i=1}^{n} \sum_{k=1}^{n} \sum_{l=1}^{n} \alpha d_{kl} Y_{kl}^{i} \tag{1}$$

subject to:

$$\sum_{k=1}^{n} Z_{kk} = p \tag{2}$$

$$\sum_{k=1}^{n} Z_{ik} = 1 \text{ for every } i = 1, \ldots, n \tag{3}$$

$$\sum_{j=1}^{n} W_{ij} Z_{jk} + \sum_{l=1}^{n} Y_{kl}^{i} = \sum_{l=1}^{n} Y_{lk}^{i} + O_i Z_{ik} \text{ for every } i, k = 1, \ldots, n \tag{4}$$

$$Z_{ik} \leq Z_{kk} \text{ for every } i, k = 1, \ldots, n \tag{5}$$

$$\sum_{i=1}^{n} O_i Z_{ik} \leq \Gamma_k Z_{kk} \text{ for every } k = 1, \ldots, n \tag{6}$$

$$Y_{kl}^{i} \geq 0 \text{ for every } i, k, l = 1, \ldots, n \tag{7}$$

$$Z_{ik} \in \{0, 1\} \text{ for every } i, k = 1, \ldots, n. \tag{8}$$

The objective (1) is to minimize the total transportation cost. Constraint (2) ensures that exactly p hubs are located. Each non-hub node is allocated to a single hub, which is guaranteed by (3). Constraint (4) represents the flow conservation equation, while (5) ensures that hub nodes are established for every distribution/collection step, thus preventing direct transmission between non-hub nodes. Constraint (6) is

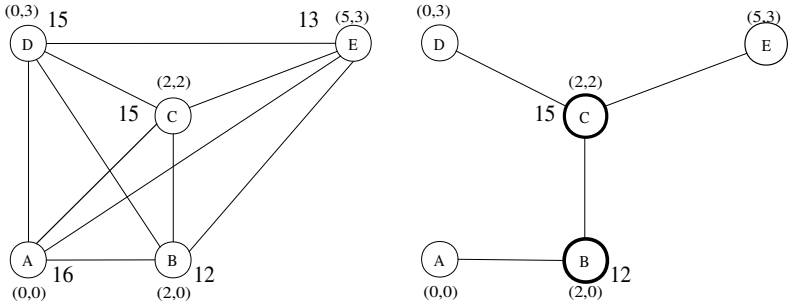

Fig. 1. Hub network $n = 5$, $p = 2$, $\alpha = 0.25$

imposed to restrict the amount of flow collected at a hub. Constraints (6) and (7) reflect non-negative and binary nature of variables Y_{kl}^i and Z_{ik} respectively.

On the left side of Figure 1 an example of a network with $n = 5$ nodes A, B, C, D, E is presented. The nodes are given by their coordinates in the plane and the corresponding distance matrix is:

$$D = \begin{pmatrix} 0 & 2 & 2\sqrt{2} & 3 & \sqrt{34} \\ 2 & 0 & 2 & \sqrt{13} & 3\sqrt{2} \\ 2\sqrt{2} & 2 & 0 & \sqrt{5} & \sqrt{10} \\ 3 & \sqrt{13} & \sqrt{5} & 0 & 5 \\ \sqrt{34} & 3\sqrt{2} & \sqrt{10} & 5 & 0 \end{pmatrix}.$$

The amount of flow that is transported from an origin-node i to a destination-node j is $W_{ij} = 1(i, j = 0, 1, 2, 3)$. The incoming flow capacities $\Gamma_k, k = 0, 1, 2, 3$ are 16, 12, 15, 15 and 13, respectively. The optimal solution of the CSApHMP that is obtained with parameters $p = 2$, $\chi = 1$, $\alpha = 0.25$ and $\delta = 1$ is presented on the right side of Figure 1. Hub nodes are located at B and C, non-hub node A is allocated to hub B, while non-hub nodes D and E are allocated to hub C. The transportation costs between every pair of nodes are: "A-**B**-A" $2+2 = 4$, "A-**B**-**C**" $2+2\cdot0.25 = 2.5$, "A-**B**-**C**-D" $2+2\cdot0.25+\sqrt{5} = 4.736$, "A-**B**-**C**-E" $2+2\cdot0.25+\sqrt{10} = 5.662$, "**B**-**B**" 0, "**B**-**C**" $2\cdot0.25 = 0.5$, "**B**-**C**-D" $2\cdot0.25+\sqrt{5} = 2.736$ "**B**-**C**-E" $2\cdot0.25+\sqrt{10} = 3.662$, "**C**-**C**" 0, "**C**-D" $\sqrt{5}$, "**C**-E" $\sqrt{10}$, "D-**C**-D" $\sqrt{5}+\sqrt{5} = 4.472$, "D-**C**-E" $\sqrt{5}+\sqrt{10} = 5.398$ and "E-**C**-E" $\sqrt{10}+\sqrt{10} = 6.324$ (hub nodes in the paths are bolded). The overall transportation cost (objective value) is equal to 79.983.

3 GENETIC ALGORITHM

A genetic algorithm is an adaptive, global search technique that utilizes concepts of natural adaptation and selective breeding of organisms. The basic GA scheme is presented in Figure 2. The GA works with a population of individuals, each representing a possible solution to a given problem. Each individual is represented

as a string of characters (genetic code) from some alphabet. The representation is often a binary string, but other alphabets of higher cardinality can also be used.

After the decoding of each individual, a fitness value is assigned to it, which measures individual's quality in the population. The selection operator favors individuals with better fitness values to reproduce more often than the worse ones. The selected parent-individuals are subject to crossover operator. Crossover creates offspring-individuals by combining genetic codes of the parents. The result of crossover is a structured, randomized exchange of genetic material between the individuals, with the possibility that good solution can generate better ones. The mutation operator performs small changes of the offspring gene's value with some small probability. The role of mutation is to restore the lost subregions of search space, preventing the premature convergence of the GA.

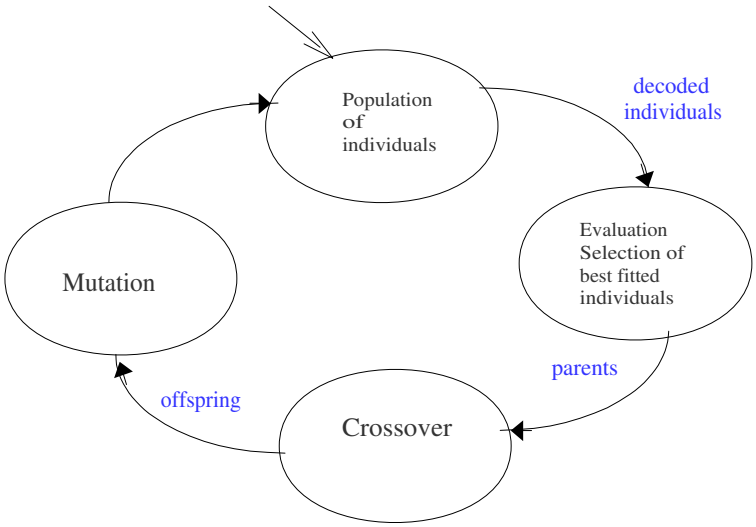

Fig. 2. Basic GA scheme

Genetic operators: selection, crossover and mutation are iteratively applied until certain stopping criterion is satisfied and the best individual of the last generation is considered as a solution to the problem. The stopping criterion of the GA can be a fixed number of generations or the GA can terminate when this solution has not been improved during a given number of iterations. Various extensions of genetic algorithms have been developed, including new data structures for representing individuals, problem-specific genetic operators and hybridization with other exact or heuristic methods.

The genetic algorithm approach is widely used for solving various combinatorial optimization problems, which include location problems such as: Simple Plant Location Problem [11], Index Selection Problem [11], Dynamic Facility Layout Prob-

lem [7], Capacitated Cell Formation Problems with Multiple Routings [17], Periodic Arc Routing Problem [15], Discrete Ordered Median Problem [6, 18], etc.

Genetic algorithms are also used for solving some other hub location problems: Uncapacitated Multiple Allocation p-hub Median Problem – UMApHMP in [19], Uncapacitated Multiple Allocation HUb Location Problem – UMAHLP in [12], Uncapacitated Single Allocation Hub Location Problem – USAHLP in [9, 20], Uncapacitated Multiple Allocation p-hub Center Problem – UMApHCP in [13], Uncapacitated Single Allocation p-hub Median Problem – USApHMP in [14]. Although these problems are similar by structure, their properties are substantially different and proposed genetic algorithms for solving these problems have quite different nature.

4 PROPOSED GENETIC ALGORITHM

4.1 Representation of the Individuals

The genetic code of an individual consists of n genes, each referring to a network node. The first bit in each gene takes the value of 1 if the current node is a located hub, 0 if not. Considering these bit values, the array of opened hub facilities is formed. Remaining bits of the gene are referring to the hub that is assigned to the current node.

If $0 < \alpha < \chi$ and $0 < \alpha < \delta$, the optimal solution usually does not include allocation of each origin/destination node to its nearest hub. The indices of hubs that are closer to non-hub nodes appear often in the optimal solution, while the indices of far away hubs are rare. For this reason, the genetic search is directed to "closer" hub facilities, while the "distant" ones are considered with small probability. In this GA implementation, the strategy named "nearest neighbor ordering" is used. For each non-hub node the created array of located hubs is arranged in non-decreasing order of their distances from the current non-hub node. The applied strategy ensures that closer hubs have higher assigning priority to non-hub nodes, while hub nodes are assigned to themselves by default.

Example 1. The genetic code: 00|10|10|00|00 corresponds to the optimal solution presented in Figure 1. The first bits in each gene (0, 1, 1, 0, 0) denote established hubs (B and C), while the remaining bits of the genes (0, 0, 0, 0, 0) show assignments. For non-hub node A, the arranged array of hubs is B,C, with distances 2 and $2\sqrt{2}$, respectively. For non-hub node D, the arranged array of hubs is C, B, with distances $\sqrt{5}$ and $\sqrt{13}$, respectively. For non-hub node E, the arranged array of hubs is also C, B, with distances $\sqrt{10}$ and $3\sqrt{2}$, respectively. Non-hub nodes A, D and E are assigned to their closest established hubs (node A to hub B, nodes D and E to hub C). Hub nodes B and C are obviously assigned to themselves.

The applied binary encoding ensures that information whether the node is hub or not, and the node's assignment to a hub is in the same gene. In this way, all the

information about each node is grouped in the genetic code, which implies shorter building-blocks. The building-block hypothesis and schema theorem [1, 2] favor both shorter building-blocks and binary encoding, so that the applied encoding scheme yields to effectiveness of the GA.

4.2 Objective Function

The indices of established hubs are obtained from the first bits of each gene. For each non-hub node, the array of established hubs is arranged in non-decreasing order with respect to distances from the current node. The index of a hub that is assigned to the current non-hub node is obtained from the remaining part of the gene. If its value is r, the r^{th} hub is taken from the previously arranged array ($r = 0, 1, \ldots, p-1$).

Arranging the array of established hubs is performed $n-p$ times for each individual in every generation. This may require additional CPU time, but p is relatively small in tested CSApHMP instances ($p \leq 20$), so it does not affect the total CPU time significantly. Although the total running time is slightly longer, the experiments show a significant improvement by using this strategy.

After the assigning procedure described above has been performed, the objective value is simply evaluated only by summing up the distances origin-hub, hub-hub and hub-destination, multiplied by flows and corresponding parameters χ, α and δ.

It may happen that a non-hub node is allocated to a hub whose remaining capacity is not enough to satisfy the node's demand. In this case, the next hub from the array of established hubs for the current node which satisfies the capacity constraint is taken. If there is no such a hub we consider the individual infeasible by setting its fitness to 0. This case is very rare in practice and it usually happens if the sum of hub capacities is less than the overall flow in the network. So, the infeasible individuals (in the sense of insufficient hub capacities) will be generated in the initial population with very small probability.

The applied strategy of correcting individuals with insufficient capacities to feasible ones may slightly affect the quality of the GA solution, but it preserves the diversity of the genetic material. If all the infeasible individuals were encountered in the population, they may become dominant in the following generations and the algorithm might provide no solution or finish in a local optimum. If the incorrect individuals were excluded from the population, the possibility of premature convergence would rapidly increase.

For these reasons, local search heuristics generally tend to work less well for capacitated hub problems, even for small-size problem dimensions. The capacity constraints can make them more difficult to move through the solution space without encountering infeasible solutions.

4.3 Genetic Operators

The GA method uses fine-grained tournament selection – FGTS [8] that is an improvement of the standard tournament selection. It is used in cases when the average

tournament size F_{tour} is desired to be fractional. In this GA implementation F_{tour} is set to 5.4.

The standard crossover operator is not appropriate for the applied encoding scheme. If this operator is applied, the parents with exactly p located hubs may generate offspring with the number of located hubs different from p. To overcome this problem, the standard crossover operator is modified to produce only individuals with p established hubs.

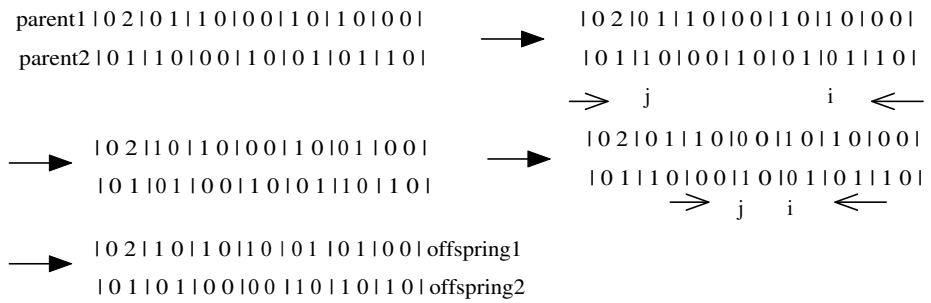

Fig. 3. Modified crossover operator for $n = 7$, $p = 3$

The crossover operator is simultaneously tracing through genetic codes of the parents from right to left searching the gene position i where the *parent1* has "1-bit" and *parent2* "0-bit" on the first bit position (see Figure 3). Then the individuals exchange whole genes at the found gene position i. The same process is simultaneously performed starting from the left side of parents' genetic codes. The operator is searching the gene position j where the *parent1* has "0-bit" and *parent2* "1-bit" on the first bit position, and all genes at the found position are exchanged. The described process is repeated until $j \geq i$.

The applied crossover operator preserves the feasibility of individuals, since the number of located hubs in both offsprings remains unchanged compared to their parents. The crossover is performed with the rate $p_{cross} = 0.85$, which means that around 85 % of individuals take part in producing offsprings, while in approximately 15 % of cases no crossover takes place and the offsprings are identical to one of the parents.

Offsprings generated by crossover operator are subject to mutation. The mutation operator is performed by changing a randomly selected gene in the genetic code. Considering the applied encoding scheme (i.e. different nature of bits according to their position in each gene), it is obvious that the mutation rate should differ for the first bit and the remaining bits in each gene. Moreover, for the bits on the second, third, fourth... bit position in each gene the mutation of the bits has different impact, since they are arranged according to the "nearest neighbor ordering" strategy (see [14] for detailed explanation). Since mutation impact for these bits increases

by the factor of 2, it is reasonable to decrease the corresponding mutation rates by the same factor.

Basic mutation rates used in the GA implementation are:

- $\frac{0.4}{n}$ for the bit on the first position,
- $\frac{0.1}{n}$ for the bit on the second position.

The next bits in the gene have repeatedly two times smaller mutation rate: $\left(\frac{0.05}{n}, \frac{0.025}{n}, \ldots\right)$.

During the GA execution, it is possible that (almost) all individuals in the population have the same bit value at certain bit position. These bits are called *frozen* (see Figure 4). If the number of frozen bits is l, the search space becomes 2^l times smaller, and the possibility of premature convergence increases rapidly. The selection and crossover operators can not change any frozen bit value, and the basic mutation rate is often insufficiently small to restore the lost sub-regions of the search space. If the basic mutation rate is increased significantly, genetic algorithm behaves like a random search. For that reason, basic mutation rates are increased, but only for frozen bits. When compared to basic mutation rates, frozen bits are mutated at 2.5 times higher rate $\left(\frac{1.0}{n}\right.$ instead of $\left.\frac{0.4}{n}\right)$ if they are at the first position of the gene, and at 1.5 times higher rate $\left(\frac{0.075}{n}, \frac{0.0375}{n}, \ldots\right)$ otherwise.

```
|0 2 | 0 1 | 1 0 |0 0 |1 0 | 0 2 |0 2 |1 0|
|0 2 | 0 1 | 1 0 |0 0 |1 0 | 1 0 |0 2 |0 0|
|1 2 | 0 1 | 0 2 |0 1 |1 0 | 1 0 |0 1 |0 1|
|0 2 | 1 0 | 0 0 |0 0 |1 0 | 1 0 |0 2 |0 0|
|0 2 | 0 1 | 1 0 |0 0 |1 1 | 0 0 |0 1 |1 0|
|0 2 | 0 1 | 1 0 |0 0 |1 0 | 1 0 |0 2 |0 0|
   f           f   f         f
```

Fig. 4. Frozen bits in GA reprezentation for $n = 8$, $p = 3$

The number of mutated ones and zeros at the first bits of genes is counted and compared for each individual. In case these numbers are not equal, it is necessary to mutate additional leading bits of the genes. By equalizing the number of mutated ones and zeros on the leading positions, the mutation operator preserves exactly p located hubs.

4.4 Other GA Aspects

The initial population numbers 150 individuals and it is randomly generated. According to the applied "nearest neighbor ordering" strategy and minimization objective function, the "closer" hubs are favored for each non-hub node. Therefore, while generating the initial population, the first bit and the remaining ones of each genetic code segment are generated with different probabilities.

Each individual in the initial population is generated as follows. The first bit in each gene takes the value of 1 with the probability of $\frac{p}{n}$. The second bit in each gene is generated with the probability of $\frac{1.0}{n}$, while the following bits take the value of 1 with two times smaller probability than the previous ones $\left(\frac{1}{2n}, \frac{1}{4n}, \frac{1}{8n}, \dots\right)$. The applied strategy ensures that in the initial population each non-hub node is frequently assigned to its closest/closer hub and rarely to a distant hub. For more detailed explanation see [14].

Although the probability of $\frac{p}{n}$ for generating first bits in each gene leads to establishing p hubs, it may be slightly different in practice. If an individual has a number of ones in the genetic code that is different from p (denoted as k, $k \neq p$), it is corrected by adding/erasing $|p - k|$ hubs going backwards from the end of the genetic code. In this way, the initial population becomes feasible. Genetic operators implemented in the GA preserve the fixed number of hubs and keep them distinct. Therefore, all individuals in the following generations remain feasible.

If an individual with the same genetic code repeats in the population, its objective value is set to zero. The selection operator disables duplicated individuals in entering the next generation. This strategy helps in preserving the diversity of genetic material and in keeping the algorithm away from the local optima trap. The individuals with the same objective value, but different genetic codes may dominate in the population after certain number of iterations. If their codes are similar, it may cause a premature convergence of the GA. For that reason, the appearance of these individuals is limited to a constant N_{rv}. In this GA implementation N_{rv} is set to 40. One third of the population is replaced in every generation, excluding the best 100 individuals that directly pass to the next generation. These elite individuals preserve highly fitted genes of the population. Their objective values are calculated in the first generation only.

The main purpose of caching is to avoid the re-calculation of objective values for individuals that appear again during the GA run. The evaluated objective values are stored in a hash-queue data structure, which is created by using the Least Recently Used (LRU) caching strategy. When the same code is obtained again, its objective value is taken from the hash-queue table, instead of re-calculating its objective function. The implemented caching technique improves the GA running time (see [10, 11]). The number of cached objective values in the hash-queue table is limited to $N_{cache} = 5\,000$ in our GA implementation.

5 COMPUTATIONAL RESULTS

In this section the computational results of the proposed GA method and comparisons with the results obtained by using CPLEX 8.1.0 Optimizer are given. The GA implementation was coded in C programming language and tested on an AMD Sempron 2.4+ at 1 597 MHz with 256 MB RAM memory under Linux (Knoppix 3.7) operating system. The experiments using CPLEX were carried out on an Intel 1.8 GHz with 256 MB RAM. Both methods were tested on a set of ORLIB instances taken from [3].

The AP (Australian Post) data set is derived from the study of postal delivery system. The largest instance includes $n = 200$ nodes (representing postcode districts) with parameters $\chi = 3$, $\alpha = 0.75$ and $\delta = 2$. Smaller size instances, with $n = 10, 20, 25, 50, 100$, nodes are obtained by aggregating the largest initial instance. The number of hubs (mail sorting/consolidation centres) in tested instances is up to 20. Two types of capacities are considered: tight (T) and loose (L) that give tightly and loosely capacitated problems, respectively.

The GA parameters mentioned in previous sections showed to be robust and appropriate for CSApHMP. The algorithm stops if the maximum number of generations has been reached, which is $N_{gen} = 5\,000$. The algorithm also terminates if the best individual or the best objective value remained unchanged through $N_{rep} = 2\,000$ successive generations, respectively. On all instances that were tested, this criterion allows the GA to converge to high-quality solutions. Only minor improvements in the quality of final solutions can be expected when prolonging the runs, which can be seen from Tables 1 and 2. The GA method was run 20 times on each AP instance.

The optimal solutions, obtained by CPLEX solver on smaller size AP instances ($n \leq 50$), are presented in Table 1. In the first column the instance's dimensions n, p and the capacity type of problem are given. For example, 40L5 refers to AP instance with $n = 40$ nodes and $p = 5$ hubs to be located, while "L" stands for loose capacities on hubs. The next two columns contain the optimal solution of the current instance obtained by CPLEX and the corresponding running time $t[s]$ in seconds. If no feasible solution exists for the current instance, *inf* is written. In the next column the number of iterations n_{iter} is given.

The remaining columns of Table 1 contain the results of the proposed GA. The best value of GA is given in the fifth column of Table 1, with mark *opt* in cases when at least one of the twenty runs of the GA reaches optimal solution known in advance. If GA gave no solution for a particular instance, a dash "-" is written. The average time (in seconds) needed to detect the best value is written in the $t[s]$ column, while $t_{tot}[s]$ represents the average total time (in seconds) that the algorithm needed to finish.

The solution quality in all 20 executions ($i = 1, 2, \ldots, 20$) is evaluated as a percentage gap named $agap = \frac{1}{20}\sum_{i=1}^{20} gap_i$, where $gap_i = 100\frac{sol_i - Opt.sol}{Opt.sol}$ is evaluated with respect to the optimal solution $Opt.sol$, or the best-known solution $Best.sol$, i.e. $gap_i = 100\frac{sol_i - Best.sol}{Best.sol}$ in cases when no optimal solution is found (sol_i represents the GA solution obtained in the i^{th} execution). Standard deviation of the average

Inst.	CPLEX			GA							
name	Solution	t[s]	N_{iter}	*best*	$t[s]$	$t_{tot}[s]$	*gen*	*gap*[%]	σ[%]	*eval*	*cache*[%]
10L2	167 493.06497	0.01	120	opt	0.005	0.368	2 013	0.000	0.000	3 564.8	96.5
10L3	136 008.12591	0.18	290	opt	0.004	0.500	2 010	0.000	0.000	17 984.5	82.1
10L4	122 396.06809	0.21	322	opt	0.010	0.519	2 022	0.000	0.000	18 871.5	81.4
10L5	91 105.370684	0.17	262	opt	0.015	0.637	2 037	0.000	0.000	24 984.8	75.5
10T2	198 060.79558	0.10	299	opt	0.004	0.364	2 018	0.000	0.000	4 102.9	95.9
10T3	166 219.04818	0.22	299	opt	0.004	0.491	2 015	0.000	0.000	18 148.3	82.0
10T4	137 344.52962	0.19	368	opt	0.009	0.508	2 015	0.000	0.000	19 029.5	81.1
10T5	122 903.55479	0.42	983	opt	0.052	0.673	2 151	0.001	0.003	30 402.3	71.8
20L2	172 816.68972	0.55	558	opt	0.025	0.629	2 053	0.000	0.000	11 259.1	89.0
20L3	151 533.08378	2.45	1 481	opt	0.028	0.986	2 041	0.000	0.000	28 273.5	72.3
20L4	135 624.88360	2.85	1 797	opt	0.024	1.077	2 029	0.000	0.000	30 892.3	69.6
20L5	123 130.09462	4.41	3 043	opt	0.136	1.488	2 180	0.000	0.000	41 536.8	62.2
20L10	81 870.548582	2.13	1 359	opt	0.050	1.926	2 038	0.013	0.056	46 326.6	54.6
20T2	194 077.66234	1.16	712	-	0.000	0.001	1	0.000	0.000	150.0	0.0
20T3	165 290.85005	4.51	3 982	opt	0.090	0.978	2 175	0.497	0.259	32 762.0	69.9
20T4	139 415.28208	1.70	1 010	opt	0.037	1.031	2 054	0.000	0.000	31 449.9	69.4
20T5	123 453.01027	1.42	800	opt	0.029	1.262	2 031	0.000	0.000	36 050.0	64.6
20T10	846 61.065508	1.62	1 093	opt	0.044	1.698	2 040	0.000	0.000	45 794.0	55.2
25L2	177 869.30164	1.92	1 079	opt	0.065	0.837	2 114	0.000	0.000	17 481.0	83.5
25L3	155 256.32315	4.03	1 380	opt	0.063	1.322	2 078	0.000	0.000	33 468.4	67.9
25L4	139 197.16909	7.38	2 813	opt	0.107	1.562	2 126	0.137	0.217	40 405.8	62.0
25L5	123 574.28868	4.73	1 908	opt	0.153	1.983	2 158	0.007	0.021	47 502.7	56.0
25L10	81 870.548582	2.12	1 359	opt	0.114	2.636	2 068	0.000	0.000	47 681.3	54.0
25T2	inf	2.22	2 777	inf	0.000	0.003	1	0.000	0.000	150.0	0.0
25T3	195 500.88511	9.88	4 792	opt	0.065	1.144	2 096	0.518	0.460	36 549.2	65.2
25T4	168 226.97385	20.66	12 684	opt	0.122	1.390	2 159	0.223	0.066	39 530.6	63.4
25T5	145 318.64862	10.25	5 881	opt	0.341	1.989	2 392	0.386	0.942	54 966.8	54.2
25T10	99 700.96103	4.78	2 719	opt	0.088	2.244	2 063	0.000	0.000	44 977.8	56.5
40L2	178 602.58220	10.21	2 134	opt	0.148	1.495	2 169	0.000	0.000	27 426.0	74.8
40L3	161 641.90679	73.36	7 695	opt	0.147	2.475	2 102	0.000	0.000	43 089.8	59.1
40L4	145 931.30224	60.22	6 402	opt	0.476	3.222	2 322	0.000	0.000	53 733.7	53.7
40L5	135 908.09368	110.11	13 131	opt	0.472	3.742	2 264	0.012	0.056	55 454.3	51.1
40L10	104 631.51100	108.21	12 933	opt	0.642	5.397	2 238	0.045	0.137	58 528.5	47.8
40T2	inf	25.77	6 845	inf	0.000	0.001	1	0.000	0.000	150.0	0.0
40T3	179 795.59780	34.29	4 434	opt	0.520	1.954	2 277	5.376	11.836	47 284.8	50.1
40T4	155 041.20503	35.33	4 379	opt	0.473	2.846	2 363	0.000	0.000	53 030.1	55.2
40T5	141 412.38851	29.03	3 197	opt	0.169	3.290	2 093	0.106	0.376	50 690.9	51.6
40T10	107 089.58661	72.65	9 848	opt	0.953	5.450	2 389	0.066	0.296	62 842.3	47.5
50L2	178 548.98256	41.42	3 467	opt	0.401	2.281	2 345	0.000	0.000	35 509.4	69.8
50L3	161 008.12675	200.26	9 813	opt	0.322	3.638	2 163	0.000	0.000	48 118.4	55.6
50L4	145 578.81661	334.50	23 473	opt	0.409	4.217	2 184	0.000	0.000	52 359.7	52.1
50L5	132 651.43955	190.22	10 346	opt	0.522	5.114	2 201	0.054	0.242	57 019.3	48.3
50L10	102 706.85734	365.61	30 055	opt	1.559	8.339	2 424	0.284	0.356	67 992.6	44.0
50T2	inf	97.06	12 813	inf	0.000	0.002	1	0.000	0.000	150.0	0.0
50T3	185 779.91419	563.40	51 631	opt	0.889	3.106	2 764	1.130	1.413	61 518.3	55.5
50T4	153 334.47291	292.06	24 079	opt	1.725	5.130	2 956	0.654	1.008	70 842.1	52.1
50T5	139 682.21196	630.55	51 642	opt	0.775	5.165	2 323	0.080	0.267	60 252.3	48.2
50T10	104 470.15626	441.52	32 852	opt	1.772	8.165	2 516	0.354	0.412	69 817.2	44.6

Table 1. Results of CPLEX and GA on smaller instances

gap $\sigma = \sqrt{\frac{1}{20}\sum_{i=1}^{20}(gap_i - agap)^2}$ is also presented. The last two columns are related to caching: *eval* represents the average number of evaluations, while *cache* displays savings (in percent) achieved by using the GA caching technique. Since the hash-queue table is being searched and updated in O(1) time, the percentage of cache savings is very similar to running-time savings in practice. In average, instead of 250 000 evaluations of the objective function, the algorithm used between 44.0 and 95.9 percent of the values stored in the hash-queue table.

For example, 200T20, the (average) number of the GA generations is 4 389 (see column *gen* in Table 2). Since the initial population numbers 150 individuals and 50 non-elite individuals are replaced in each GA generations, it means that the algorithm would need $1 \cdot 150 + 4\,388 \cdot 50 = 218\,550$ objective function evaluations. As can be seen from the *eval* column in Table 2, the GA performs 149 937 objective function evaluations (in average), that is 68.239 percent of the initial number of evaluations (218 550). It means that 31.707 percent of the objective function values are obtained from the hash-queue-table.

Moreover, instead of extensive use of the time-consuming local search, the GA method includes the "nearest neighbour ordering" search, that does not require additional CPU time, but improves GA performance significantly. The GA improves the solutions somewhat more slowly in the beginning of the search, but it continues to improve the solutions steadily, and finally reaches high quality solutions.

As can be seen from Table 1, for all smaller size AP instances the proposed GA quickly reached optimal solutions that were previously obtained by CPLEX, with the exception of one instance 20T2, where GA gave no solutions. In three cases, 25T2, 40T2, 40T2, 50T2, neither CPLEX nor GA obtained a feasible solution.

The computational experiments that have been carried out with CPLEX solver showed that CPLEX can not provide solutions on larger AP instances in reasonable CPU time. On these instances the CPLEX did not terminate after few day's computation, though it found some upper bounds in most cases. Therefore, only the results of the proposed GA on large-scaled AP instances are presented in Table 2. These results are given in the same way as in Table 1.

The GA concept cannot prove optimality and an adequate finishing criteria that will fine-tune solution quality does not exist. Therefore, as the column t_{tot} in Table 1 shows, the algorithm runs through additional $t_{tot} - t$ time (until finishing criteria are satisfied), although it already reached its best solution. For technical reasons, the GA and CPLEX could not be tested on the same platform, so exact comparisons can not be carried out. According to the SPEC-fp2000 benchmarks (`www.spec.org`), computers AMD Sempron 2.4+ at 1 597 MHz and Intel 1.8 GHz have average (base) speedup values 641 and 669, respectively. Considering this, we can assume that two platforms mentioned above have very similar performances and some descriptive comparisons of CPU times for smaller AP instances can be made. Comparing the $t[s]$ column in Table 1 for the CPLEX and the GA, it can be seen that the GA concept reaches optimal solution in significantly shorter CPU time compared to CPLEX. After all, the total running time of the proposed GA is reasonably short, even for practical size AP instances that could not be solved by CPLEX (see the t_{tot} column of Table 2).

6 CONCLUSIONS

In this paper, an evolution-based approach for solving CSApHMP is proposed. The problem has large practical real-world application in network design. The described

Inst.	GA_{best}	$t[s]$	$t_{tot}[s]$	gen	$gap[\%]$	$\sigma[\%]$	$eval$	$cache[\%]$
100L2	189 351.552203	3.386	9.238	3 062	0.042	0.075	63 514.5	58.6
100L3	165 714.455236	5.937	16.604	3 069	0.006	0.019	81 070.6	47.2
100L4	146 197.756744	4.573	16.566	2 711	0.014	0.035	75 511.4	44.4
100L5	137 232.521623	5.046	18.872	2 678	0.087	0.162	79 411.3	40.8
100L10	107 207.722491	10.969	29.531	3 076	0.353	0.523	95 923.6	37.7
100L15	91 285.263160	12.915	36.091	3 052	0.573	0.330	96 059.8	37.1
100L20	81 034.594907	15.808	43.294	3 084	0.609	0.530	97 763.2	36.7
100T4	156 486.756614	5.772	15.498	3 103	0.487	0.513	86 243.9	44.4
100T5	143 567.783356	8.336	21.672	3 205	0.718	0.946	94 624.1	41.0
100T10	109 633.407766	13.528	32.119	3 332	1.191	0.777	103 623.7	37.8
100T15	92 635.948394	11.458	34.640	2 925	0.596	0.387	92 245.7	37.0
100T20	82 425.427067	15.370	42.901	3 050	0.689	0.498	96 779.6	36.6
200L2	182 459.254460	30.658	58.915	3 927	0.024	0.059	92 531.9	52.9
200L3	162 887.031354	43.092	93.141	3 655	0.002	0.004	105 562.4	42.3
200L4	147 814.839498	50.356	107.518	3 650	0.024	0.041	111 076.5	39.2
200L5	140 563.766161	54.518	118.703	3 615	0.262	0.158	114 759.5	36.6
200L10	110 400.539154	102.042	182.995	4 262	0.861	0.689	142 324.0	33.3
200L15	94 525.394936	123.002	223.727	4 347	0.611	0.412	147 403.2	32.2
200L20	85 290.576635	143.854	261.820	4 342	0.548	0.328	148 658.9	31.6
200T4	158 161.970642	90.691	115.712	4 886	0.368	0.512	145 076.5	40.7
200T5	143 422.425191	97.293	139.989	4 672	1.702	1.235	146 129.9	37.5
200T10	112 056.605241	152.022	204.915	4 919	2.505	1.374	162 483.5	34.0
200T15	96 448.953395	128.705	223.062	4 452	1.378	1.514	149 817.9	32.7
200T20	86 496.090607	154.482	260.028	4 389	1.040	0.551	149 937.3	31.7

Table 2. Results of GA on larger instances

GA method uses a binary encoding of the individuals and an appropriate objective function. The initial population is generated to be feasible and genetic operators adapted to the CSApHMP are designed and implemented. The applied "nearest neighbor ordering" strategy directs GA to promising search regions. Genetic operators preserve exactly p located hubs in the individuals of every generation. The idea of frozen bits is used to increase the diversity of genetic material. The caching GA technique additionally improves the computational performance of the algorithm.

The computational experiments on the well-known AP hub data set demonstrate the robustness of the proposed algorithm with respect to the solution quality and running times. Computational results show that the GA reaches all optimal solutions obtained by CPLEX for smaller problem dimensions. The proposed GA method also provided solutions for capacitated large-scale AP instances with more than 50 nodes that were unsolved up to now. The presented computational results clearly indicate the usefulness of the proposed GA approach.

Hence, our future work could also concentrate on the parallelization of the GA and its hybridization with exact methods. Based on the results, we also believe the

GA approach has a potential as useful metaheuristics for solving other capacitated hub problems and more complex hub location models.

Acknowledgement

This research was partially supported by Serbian Ministry of Science and Technological Development under the grant No. 144 007. The author is grateful to Jozef Kratica for his useful comments and suggestions on a draft version of this paper.

REFERENCES

[1] Bäck, T.—Fogel, D. B.—Michalewicz, Z.: Basic Algorithms and Operators. Evolutionary Computation 1, Institute of Physics Publishing, Bristol-Philadelphia, 2000.

[2] Bäck, T.—Fogel D. B.—Michalewicz, Z.: Advanced Algorithms and Operators. Evolutionary Computation 2, Institute of Physics Publishing, Bristol-Philadelphia, 2000.

[3] Beasley, J. E.: Obtaining Test Problems via Internet. Journal of Global Optimization, Vol. 8, 1996, pp. 429–433.

[4] Campbell, J. F.: Integer Programming Formulations of Discrete Hub Location Problems. European Journal of Operational Research, Vol. 72, 1994, pp. 387–405.

[5] Campbell, J. F.—Ernst, A.—Krishnamoorthy, M.: Hub Location Problems. H. Hamacher and Z. Drezner (Eds.): Facility Location: Applications and Theory, Springer-Verlag, Berlin-Heidelberg, 2002, pp. 373–407.

[6] Dominguez-Marin, P.—Hansen, P.—Mladenovic, N.—Nickel, S.: Heuristic Procedures for Solving the Discrete Ordered Median Problem. ITWM Bericht, Fraunhofer Institut fur Tecno-und Wirtschaftsmathematik (ITWM), Kaiserslautern, Germany, Vol. 46, 2003.

[7] Dunker, T.—Radons, G.—Westkamper, E.: Combining Evolutionary Computation and Dynamic Programming for Solving a Dynamic Facility Layout Problem. European Journal of Operational Research, Vol. 165, 2005, pp. 55–69.

[8] Filipović, V.: Fine-Grained Tournament Selection Operator in Genetic Algorithms. Computing and Informatics, Vol. 22, 2003, pp. 143–161.

[9] Abdinnour-Helm, S.—Venkataramanan, M. A.: Solution Approaches to Hub Location Problems. Annals of Operations research, Vol. 78,1998, pp. 31–50.

[10] Kratica, J.: Improving Performances of the Genetic Algorithm by Caching. Computers and Artificial Intelligence, Vol. 18, 1999, pp. 271–283.

[11] Kratica, J.: Parallelization of Genetic Algorithms for Solving Some Np-Complete Problems (Ph. D. thesis in Serbian). Faculty of Mathematics, University of Belgrade, 2000.

[12] Kratica, J.—Stanimirović, Z.—Tošić, D.—Filipović, V.: Genetic Algorithms for Solving Uncapacitated Multiple Allocation Hub Median Problem. Computing and Informatics, Vol. 22, 2005, pp. 1001–1012.

[13] Kratica, J.—Stanimirović, Z.: Solving the Uncapacitated Multiple Allocation P-Hub Center Problem by Genetic Algorithm. Asia-Pacific Journal of Operational Research, Vol. 23, 2006, pp. 425–437.

[14] Kratica, J.—Stanimirović, Z.—Tošić, D.—Filipović, V.: Two Genetic Algorithms for Solving the Uncapacitated Single Allocation P-Hub Median Problem. European Journal of Operational Research, Vol. 182, 2006, pp. 15–28.

[15] Lacomme, P.—Prins, C.—Ramdane-Cherif, W.: Evolutionary Algorithms for Periodic Arc Routing Problems. European Journal of Operational Research, Vol. 165, 2005, pp. 535–553.

[16] Love, R. F.—Moris, J. G.—Wesolowsky, G.: Facility location: Models and methods. Publication in Operations Research, North-Holland, New York, Vol. 7, 1988.

[17] Nsakanda, A. L.—Diaby, M.—Price, W. L.: Hybrid Genetic Approach for Solving Large-Scale Capacitated Cell Formation Problems With Multiple Routings. European Journal of Operational Research, Vol. 171, 2006 No. 3, pp. 1051–1070.

[18] Stanimirović, Z.—Kratica, J.—Dugošija, D. J.: Genetic Algorithms for Solving the Discrete Ordered Median Problem. European Journal of Operational Research, Vol. 182, 2007, pp. 983–1001.

[19] Stanimirović, Z.: An Effcient Genetic Algorithm for the Uncapacitated Multiple Allocation P-Hub Median Problem. Control and Cybernetics, Vol. 37, 2006, No. 3, pp.669-692.

[20] Topcuoglu, H.—Corut, F.—Ermis, M.—Yilmaz, G.: Solving the Uncapacitated Hub Location Problem Using Genetic Algorithms. Computers and Operations Research, Vol. 32, 2005, pp. 967–984.

Zorica Stanimirović received her B. Sc. degrees in mathematics (2000), M. Sc. in mathematics (2004) and Ph. D. in mathematics (2007) from the University of Belgrade, Faculty of Mathematics. Since 2008 she is Assistant Professor at the Faculty of Mathematics and Vice-Dean for science and research at the Faculty of Mathematics. Her research interests are facility location problems, hub location problems, combinatorial optimization and genetic algorithms.