Available online at www.sciencedirect.com
ScienceDirect
European Journal of Operational Research 182 (2007) 15–28
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/ejor

Discrete Optimization

Two genetic algorithms for solving the uncapacitated single allocation *p*-hub median problem

Jozef Kratica [a,*], Zorica Stanimirović [b], Dušan Tošić [b], Vladimir Filipović [b]

[a] *Mathematical Institute, Serbian Academy of Sciences and Arts, Kneza Mihajla 35/I, pp. 367, 11 001 Belgrade, Serbia*
[b] *Faculty of Mathematics, University of Belgrade, Studentski trg 16/IV, 11 000 Belgrade, Serbia*

Received 24 March 2005; accepted 29 June 2006
Available online 25 October 2006

Abstract

This paper deals with the Uncapacitated Single Allocation *p*-Hub Median Problem (USApHMP). Two genetic algorithm (GA) approaches are proposed for solving this NP-hard problem. New encoding schemes are implemented with appropriate objective functions. Both approaches keep the feasibility of individuals by using specific representation and modified genetic operators. The numerical experiments were carried out on the standard ORLIB hub data set. Both methods proved to be robust and efficient in solving USApHMP with up to 200 nodes and 20 hubs. The second GA approach achieves all previously known optimal solutions and achieves the best-known solutions on large-scale instances.

Keywords: Genetic algorithms; Evolutionary computations; Location; *p*-Hub median problem; Single allocation

1. Introduction

Hub networks are used in transportation and telecommunications systems (airline, ground transportation, postal delivery systems, computer networks, etc.). Instead of serving every destination node (facility) from an origin with a direct link, a hub network provides services via a specified set of hub nodes (hubs).

The flow departing from an origin node is collected in a hub, transported between hubs and finally distributed to a destination node. Smaller transportation rates between the hubs can provide a reduction in total transportation costs in the network. Hub location problems deal with the location of hubs and assigning of non-hub nodes to located hubs.

There are several kinds of hub location problems based on characteristics of a particular hub network. If the number of hub nodes is fixed to p, we are dealing with p-hub location problems. Each non-hub node can be allocated either to one (single allocation scheme) or to more hubs (multiple allocation scheme). Capacitated versions of hub problems may involve different kinds of capacity constraints

* Corresponding author.
E-mail addresses: jkratica@mi.sanu.ac.yu (J. Kratica), zoricast@mi.sanu.ac.yu (Z. Stanimirović), tdusan@mi.sanu.ac.yu (D. Tošić), vladaf@matf.bg.ac.yu (V. Filipović).
URL: http://www.geocities.com/jkratica (J. Kratica).

0377-2217/$ - see front matter
doi:10.1016/j.ejor.2006.06.056

in a network. Fixed hub locating costs may also be assumed. A general survey of different hub location problems can be found in [9].

This paper considers the Uncapacitated Single Allocation p-Hub Median Problem. The objective is to minimize the overall flow cost in a network under the following assumptions:

- The number of hubs to be located is predetermined (p).
- There are no capacities or fixed costs involved.
- Each origin/destination node is assigned to a single hub.
- Direct transportation between non-hub nodes is not allowed.

USApHMP belongs to the class of NP-hard problems. Even when the set of hubs is given, the assignment sub-problem of optimal allocation of non-hub nodes to hubs is also NP-hard [25].

The p-hub median formulation can sometimes lead to unsatisfactory results, for example, when the worst origin–destination distance (cost) is important. Difficulties of this kind can be avoided by using the p-hub center formulation, which minimizes the maximum distance between origin–destination pairs [24].

Although the capacitated/uncapacitated p-hub median and hub location problems have received most attention from researchers so far, there are more complex hub location models that are more realistic and useful in practice. These are some of the new proposed models:

- Models with flow-based discounts [6,7] have shown to be appropriate for air freight in order to make allocation decisions based on the ability of flows to capture scale economies.
- The cost of flow through each link is discounted if and only if the amount of flow exceeds a certain threshold. The flow threshold model, [2,30,31,33,34] encourages the concentration of flows and use of a relatively small number of links. This model captures real operations and reflects the actual characteristics of a network (especially for high-speed telecommunication networks), and addresses some drawbacks of hub models.
- Models that include arcs with discounted cost rates (hub arcs). The objective in these models is to locate a fixed number of hub arcs [10–12] in order to minimize the overall cost.
- Another possibility is to impose a maximum travel time constraint and then to minimize the cost of setting up such a network (the number of hubs required), resulting in a hub covering problem [19].
- Hub location models with competition [26], where customer capture from competitor hubs is sought, which happens whenever the location of a new hub results in the reduction of time or distance needed from the traffic generated by the passenger to travel from the origin to the destination node. These models are suitable for both air passenger and cargo transportation.

Detailed information about various hub location problems can be found in [9].

The first formulation of UMApHMP can be found in [27]. The problem was formulated as a quadratic integer program with a non-convex objective function. A mixed integer linear programming formulation (MILP) is given in [8]. This formulation also involves a large number of variables and constraints.

Several heuristic methods for solving this problem can be found in literature. Klincewicz [20] uses several heuristics based on local neighbourhood search and clustering of nodes. A neural network approach was proposed in [13,35], solving small size hub instances. Tabu search heuristics for solving USApHMP were presented in [21,32]. New MILP formulations of the problem that involve fewer variables and constraints were given in [15,16]. The authors use a simulated annealing (SA) approach to obtain upper bounds for an LP based branch-and-bound method. The computational results on the standard ORLIB hub instances with up to 200 nodes are reported, showing that optimal solutions were obtained (with few exceptions) for $n \leqslant 50$. In [28,29] an application of the path relinking method (PR) for solving USApHMP was presented. This is an evolutionary metaheuristic procedure based on the scatter search design. The proposed PR method was tested on hub instances with $n \leqslant 100$ nodes, and then compared with a version of the improved tabu search method TABUHUB [32].

2. Mathematical formulation

In this paper, we use the integer formulation given in [27]. Let $H_{ij} \in \{0,1\}$ have value 1 if node i is allocated to a hub node j and 0 otherwise. The condition $H_{kk} = 1$ implies that the node k is a

hub. We define C_{ij} as the distance between nodes i and j, and W_{ij} as the amount of flow from node i to node j. Parameters χ, α and δ reflect the unit rates (costs) for collection (origin–hub), transfer (hub–hub), and distribution (hub–destination) respectively. 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 $\alpha < \delta$.

Using the notation mentioned above, the problem can be written as

$$\min \sum_{i,j,k,l \in N} W_{ij}(\chi C_{ik} H_{ik} + \alpha C_{kl} H_{ik} H_{jl} + \delta C_{jl} H_{jl}) \quad (1)$$

subject to

$$\sum_{k=1}^{n} H_{kk} = p, \quad (2)$$

$$\sum_{k=1}^{n} H_{ik} = 1 \quad \text{for every } i = 1, \ldots, n, \quad (3)$$

$$H_{ik} \leqslant H_{kk} \quad \text{for every } i, k = 1, \ldots, n, \quad (4)$$

$$H_{ik} \in \{0, 1\} \quad \text{for every } i, k = 1, \ldots, n. \quad (5)$$

The objective function (1) minimizes the sum of origin–hub, hub–hub and hub–destination flow costs multiplied by χ, α and δ factors respectively. Constraint (2) ensures that exactly p hubs are chosen, while constraint (3) guarantees a single hub allocation for each node. By constraint (5) we prevent hub nodes being allocated to other nodes. Constraint (4) enforces that the flow is sent only via open hubs, thus preventing direct transmission between non-hub nodes.

3. Genetic algorithms

Genetic algorithms (GAs) represent problem-solving metaheuristic method rooted in the mechanisms of evolution and natural genetics. The main idea was introduced by Holland [18], and in the last three decades GAs have emerged as effective, robust optimization and search methods.

GAs solve problems by creating a population of individuals (usually 10–200), represented by chromosomes which are encoded solutions of the problem. The representation is the genetic code of an individual and it is often a binary string, although other alphabets of higher cardinality can be used. A chromosome is composed of basic units named genes, which control the features of an individual. To each chromosome a fitness value measuring its success is assigned. The initial population (the first generation of individuals) is usually randomly initialized. The individuals in the population then pass through a procedure of simulated "evolution" by means of randomized processes of selection, crossover, and mutation.

The selection operator favors better individuals to survive through the generations. The probability that a chromosome will be chosen depends on its fitness. The higher fitness value of a chromosome provides higher chances for its survival and reproduction. There are different ways of selecting the best-fitted individuals. One of the most often used is tournament selection (for more details see [3,4,17]). Crossover and mutation operators are also used during reproduction. The crossover operator provides a recombination of genetic material by exchanging portions between the parents with the chance that good solutions can generate even better ones.

Mutation causes sporadic and random changes by modifying individual's genetic material with some small probability. Its role is to regenerate the lost or unexplored genetic material into the population. Mutation has a direct analogy with nature and it should prevent premature convergence of the GA to suboptimal solutions.

There are different policies for generation replacement. Certain numbers of individuals (elite individuals) may skip selection or even all genetic operators going directly into the next generation. This approach is named the steady-state generation replacement policy with elitist strategy. It provides a smaller gradient in the genetic search, but preserves good individuals from the past generations.

There can be many modifications of the GA, but implementing the GA usually involves the following steps:

- Evaluating the fitness of all individuals in a population.
- Selecting the best-fitted individuals.
- Creating a new population by performing crossover and mutation operators.

The process of reproduction and population replacement is repeated until a stopping criterion (fixed number of generations or satisfied quality of solutions obtained) is met.

The genetic algorithm approach is widely used for solving various combinatorial optimization

problems, which include location problems such as: simple plant location problem [23], index selection problem [23], dynamic facility layout problem [14], etc.

GAs are also used for solving some other hub location problems: Uncapacitated Multiple Allocation p-Hub Median Problem-UMApHMP in [36], Uncapacitated Single Allocation Hub Location Problem-USAHLP in [1,37], Uncapacitated Multiple Allocation p-Hub Center Problem-UMApHCP in [24]. These problems have similar names as our problem, but up to now known solution approaches for solving these problems are substantially different. For example, different allocation schemes in UMApHMP and USApHMP have great impact on the problem complexity. For the fixed set of hubs, the multiple allocation sub-problem is solved in polynomial $O(n^2 * p)$ time, while the single allocation sub-problem remains NP-hard. Therefore, proposed genetic algorithms for solving these problems have quite different natures.

4. GAHUB1 heuristic method

4.1. Representation

The genetic code of each individual consists of two segments. The first segment is a string of length p where the digits (genes) within a string correspond to the indices of open hub nodes. The digits in the first string take values from the set $\{0,1,\ldots,n-1\}$. The second segment in the genetic code has exactly $n-p$ genes. Each gene corresponds to one non-hub node and contains a number r between 0 and $p-1$ that points to a particular hub which is the rth closest.

For each non-hub node we create an array of located hub facilities and arrange it in non-decreasing order of their distances from the current node (named the "nearest neighbour ordering"). Hub nodes are assigned to themselves.

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, we direct our search to "closer" hub facilities, while the "distant" ones are considered with small probability. Sorting the array of hubs in non-decreasing order of their distances from each non-hub node, we ensure that closer hubs have higher assigning priority to non-hub nodes.

As can be seen from the results, our approach is robust for various values of parameter α. By decreasing α, the GA approach naturally obtains slightly better results, because the nearest hubs appear more often in the optimal solution.

4.2. Objective function

The indices of established hubs are obtained from the first segment of a genetic code. The duplication of a hub index is resolved in the following way: if an index repeats in a genetic code, we replace it by the next previously not used one. If there are no such indices, we use the previous index that was not already taken. Since p is smaller than n, we shall always find a "free" index to replace the duplicated one. This approach ensures that exactly p distinct hub indices are obtained from the genetic code.

After the set of established hubs is obtained and possible duplication of hub indices is resolved, we proceed to the next step. For each non-hub node, the array of established hubs is arranged in non-decreasing order with respect to distances to the current node. From the second segment of the genetic code, we take the gene that corresponds to the current non-hub node. If that gene has value r, we take the rth hub 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 USApHMP instances ($p \leqslant 20$), so it does not affect the total CPU time significantly. Although the total running time is slightly longer, our experiments show a significant improvement by using this strategy. Therefore, results obtained without arranging the array of hubs are omitted in Section 6.

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

Example 1. For $n=5$, $p=3$, $W_{ij}=1$ ($i,j=0,1,\ldots,n-1$) and distance matrix

$$C = \begin{bmatrix} 0 & 12 & 4 & 7 & 8 \\ 12 & 0 & 9 & 11 & 6 \\ 4 & 9 & 0 & 8 & 17 \\ 7 & 11 & 8 & 0 & 12 \\ 8 & 6 & 17 & 12 & 0 \end{bmatrix}$$

the genetic code

1 1 4 | 2 0

means that the nodes 1, 2 and 4 are chosen as hubs. Number 1 is duplicated in the genetic code, so the next unused node (2) is chosen. For non-hub node 0, the arranged array of hubs is 2, 4, 1, with distances 4, 8 and 12 respectively. For non-hub node 3, the arranged array of hubs is 2, 1, 4, with distances 8, 11 and 12 respectively. The second segment of the genetic code shows that the third closest hub (node 1) is assigned to node 0, and the closest hub (node 2) is assigned to node 3. Hub nodes 1, 2 and 4 are assigned to themselves. For $\chi = 3$, $\alpha = 0.75$ and $\delta = 2$, the flow cost from node 2 to node 3 is $1 \times (3 \times 12 + 0.75 \times 9 + 2 \times 8) = 58.75$.

4.3. Genetic operators

The GAHUB1 method uses fine-grained tournament selection – FGTS [17] that is an improvement of standard tournament selection. It is used in cases when the average tournament size F_{tour} is desired to be fractional. FGTS realizes two types of tournaments: the first type is held k_1 times and its size is $[F_{\text{tour}}] + 1$. The second type is realized k_2 times with $[F_{\text{tour}}]$ individuals participating. In the GAHUB1 implementation, F_{tour} is set to 5.4. If FGTS is applied to $N_{\text{nonel}} = 50$ non-elitist individuals, for example, tournaments are held $k_1 = 20$ and $k_2 = 30$ times with sizes 5 and 6 respectively. Running time for FGTS operator is $O(N_{\text{nonel}} * F_{\text{tour}})$. In our GA implementations, F_{tour} and N_{nonel} are considered to be constant (not depending on a problem size) giving a constant time complexity.

After a pair of parents is selected, the crossover operator is applied on them producing two offspring. Since each parent's genetic code consists of two segments of different nature, the standard one-point crossover operator cannot be directly used. Therefore, we apply the double one-point crossover: in each segment of parents' genetic codes one crossover point is randomly chosen and genes are exchanged after the chosen position. The crossover is performed with the rate $p_{\text{cross}} = 0.85$, which means that around 85% of individuals take part in producing offspring, while in approximately 15% of cases no crossover takes place and the offspring are identical to one of the parents.

Offspring generated by a crossover operator are subject to mutation. The mutation operator is performed by changing a randomly selected gene in the genetic code. Considering the different nature of genetic code segments, it is obvious that the mutation rate should differ for the first and the second segment. Moreover, for each gene in the second segment (representing the "nearest neighbour ordering") the mutation of the bits has different impact. For example, the mutation of the first bit in the gene changes the ordering by ± 1. The bit change $0 \rightarrow 1$ has $+1$ impact, which means that we take the next hub from the sorted array of located hubs for the current node. The bit change $1 \rightarrow 0$ will result in taking the previous hub from the same array (-1 impact). The mutation of the second bit has ± 2 impact, the third ± 4, the fourth ± 8, etc. Since the mutation impact increases by the factor of 2, we have decided to decrease the mutation rates by the same factor.

Basic mutation rates used in the GAHUB1 implementation are:

- Genes in the first segment are mutated with probability (rate) $0.7/n$.
- In each gene of the second segment, the first bit is changed with $0.1/n$ rate. The following bits in that gene are mutated at a rate two times smaller than their predecessor bit ($0.05/n$, $0.025/n$, ...).

During the GA execution, it is possible that (almost) all individuals in the population have the same gene in a certain position. These genes are called frozen. If the number of frozen genes is l, the search space becomes 2^l times smaller, and the possibility of premature convergence rapidly increases. The selection and crossover operators cannot change the bit value of any frozen gene, 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, the basic mutation rates are increased, but only for the frozen genes. In the GAHUB1 implementation, the frozen genes in the first segment are mutated at a 2.5 times higher mutation rate than the non-frozen ones ($1.75/n$

instead of $0.7/n$). In the second segment, each frozen gene is changed at a rate 1.5 higher than the corresponding basic mutation rate ($0.15/n$ for the first bit, $0.075/n$ for the second, etc.).

4.4. Other GA aspects

The initial population consists of 150 individuals. Each gene in the first segment of the initial individual's genetic code is randomly generated by uniform distribution on the set $0,\ldots,n-1$. According to the applied "nearest neighbour ordering" and minimization objective function, we favour "closer" hubs for each non-hub node. Therefore, it is desirable that the second segment in the initial genetic code should contain many zeros (each zero value represents the assignment of the closest hub to the current node).

Similar to the mutation operator, in the second segment the "1-bits" have different impacts depending on their position in the gene. The first "1-bit" has impact +1, the second "1-bit" +2, the third +4, the fourth +8, etc. Since impact increases by the factor of 2, we have also decided to decrease the probability of generating the "1-bit" with the same factor. The gene value (explained in Section 4.2) is obviously equal to the sum of its "1-bit" impacts. The probability of generating a gene is the product of "1-bit" and "0-bit" probabilities ("0-bit" probability = 1 − "1-bit" probability). The increasing order of gene values does not necessarily correspond to the non-increasing order of gene probabilities. Therefore, gene values are permuted so that their probabilities are in a non-increasing order. This arranging is performed only once, because gene probabilities are constant through the initialization part of GAHUB1.

The "1-bit" probability for the first bit position in the second segment is set to $1.0/n$. The following "1-bits" are generated with two times smaller probability compared to their predecessor ($\frac{1}{2n}$, $\frac{1}{4n}$, $\frac{1}{8n}$, etc.). This 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.

One-third of the population is replaced in every generation ($N_{\text{nonel}} = 50$), 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 only in the first generation.

The applied encoding scheme, described in Section 4.1, excludes the appearance of incorrect individuals in the initial population. Genetic operators implemented in GAHUB1 preserve the fixed number of hubs and keep them distinct. In this way, incorrect individuals do not appear in the following generations.

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 from entering the next generation. This strategy helps preserve the diversity of genetic material and keep 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 this reason, we have limited the appearance of these individuals to some constant N_{rv}. In GAHUB1 implementation N_{rv} is set to 40.

The main purpose of caching is to avoid the 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 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 calculating its objective function. The implemented caching technique improves the GA running time (see [22,23]). The number of cached objective values in the hash-queue table is limited to $N_{cache} = 5000$ in our GAHUB implementation.

5. GAHUB2 heuristic method

5.1. Representation and objective function

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

Note that this encoding scheme prevents duplication of hub indices, so the repairing procedure described in Section 4.2 is not necessary. However, it may happen that some individuals (infeasible ones) have the number of established hubs different from p. Since the application of genetic operators (described in the following sections) preserves the feasibility of individuals, it is necessary to repair

unfeasible individuals to feasible ones only for the initial population, as described in Section 5.3.

For each non-hub node, the obtained array of opened hubs is also arranged in non-decreasing order of their distances from a particular node. Objective value is then calculated in the same way as in GAHUB1 implementation (see Section 4.2).

Example 2. The genetic code:

02|10|10|00|10

corresponds to the same solution as in Example 1. The first bits in each gene $(0,1,1,0,1)$ denote established hubs (1, 2 and 4), while the remaining parts of the genes $(2,0,0,0,0)$ show assignments. By default, each hub node is assigned to itself.

5.2. Genetic operators

The FGTS selection operator was also used in the GAHUB2 method, with the same value of F_{tour} parameter ($F_{\text{tour}} = 5.4$).

The double one-point crossover operator applied in GAHUB1 is not appropriate for GAHUB2 representation. Both correct parents may produce incorrect offspring, i.e. parents with exactly p located hubs may generate offspring with the number of located hubs different from p. To overcome this problem, we have modified the standard crossover operator to produce only correct individuals.

The crossover operator simultaneously traces 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 of these genes. The individuals then exchange whole genes from that gene position i. The corresponding process is simultaneously performed starting from the left side of parents' genetic codes. The operator searches the gene position j where the parent1 has "0-bit" and parent2 "1-bit" on the first bit position, and the whole genes from the jth gene position are exchanged. The described process is repeated until $j \geqslant i$.

Note that the number of located hubs in both offspring remains unchanged compared to their parents. Since the parents were correct, their offspring remain correct. The implemented crossover operator is performed with the same rate as in GAHUB1 ($p_{\text{cross}} = 0.85$).

The modified two-level mutation operator with frozen bits is implemented in GAHUB2. For the same reason as we explained in Section 4.3, the basic mutation rates are:

- $0.4/n$ for the bit on the first position.
- $0.1/n$ for the bit on the second position. Next bits in the gene have repeatedly two times smaller mutation rate $(0.05/n, 0.025/n, \ldots)$.

When compared to the basic mutation rates, frozen bits are mutated by:

- 2.5 times higher rate ($1.0/n$ instead of $0.4/n$) if they are at the first position of the gene.
- 1.5 times higher rate $(0.075/n, 0.0375/n, \ldots)$ otherwise.

We count and compare the number of mutated ones and zeros at the first bits of genes in each individual. In cases where these numbers are not equal, it is necessary to mutate additional leading bits of the genes. Equalizing the number of mutated ones and zeros on the leading positions, the mutation operator preserves exactly p opened hubs and preserves mutated individuals feasible.

5.3. Other GA aspects

The population size for GAHUB2 is also 150 individuals. Each individual in the initial population is generated with the following strategy:

- The first bit in each gene takes the value of 1 with the probability p/n.
- The second bit in each gene is generated with the probability $1.0/n$ and the following bits take the value of 1 with two times smaller probability than the previous ones $\left(\frac{1}{2n}, \frac{1}{4n}, \frac{1}{8n}, \ldots\right)$.

Although the probability p/n for generating first bits in each gene leads to establishing p hubs, this may be slightly different in practice. The individuals that have the number of ones in their genetic codes that is different from p (denoted as k, $k \neq p$) are corrected by adding/erasing $|p - k|$ hubs going backwards from the end of the genetic code.

Other aspects of GAHUB2 are the same as for GAHUB1:

- Caching GA.
- Removing duplicated individuals from the population.
- Limiting the number of individuals with the same objective value.
- One hundred elite individuals.

- Permutation of gene values in non-increasing order of their probabilities.

6. Computational results

In this section the computational results of both presented GA methods and a comparison with existing algorithms are given. All tests were carried out on the AMD Athlon 1.33 GHz with 256 MB RAM. The algorithms were coded in C programming language and tested on two sets of ORLIB instances taken from [5].

The CAB (Civil Aeronautics Board) data set is based on airline passenger flow between cities in the United States. It contains 60 instances with up to 25 nodes and up to 4 hubs. It is assumed that the collection and distribution costs χ and δ are equal to one, while α takes values from 0.2 to 1.

The AP (Australian Post) data set is derived from a study of the postal delivery system. The largest instance includes 200 nodes (representing postcode districts) with parameters $\chi = 3$, $\alpha = 0.75$ and $\delta = 2$. Smaller size instances, with 10, 20, 25, 50, 100, nodes can be obtained by aggregating the largest initial instance. The number of hubs (mail sorting/consolidation centres) in tested instances is up to 20.

The parameters mentioned above, shown to be robust and appropriate for this problem, were used. In order to provide a fair comparison of GAHUB1 and GAHUB2 we used the same parameters wherever possible. The maximum number of generations is $N_{gen} = 500$ for smaller, and $N_{gen} = 5000$ for larger problem instances. Algorithm also stops if the best individual or the best objective value remains unchanged through $N_{rep} = 200$ ($N_{rep} = 2000$) successive generations respectively. In the case of all the instances we considered, this criterion allowed GAs 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 1–4.

Table 1
GAHUB1 results on CAB instances

n	*p*	α	*Opt.sol*	*GA best*	*t* (seconds)	t_{tot} (seconds)	*gen*	*agap* (%)	σ (%)	*eval*	*cache* (%)
20	2	0.2	979.087	opt	0.010	0.083	213	0.000	0.000	2892	73.2
20	2	0.4	1042.566	opt	0.010	0.082	211	0.000	0.000	2813	73.7
20	2	0.6	1106.044	opt	0.011	0.085	214	0.000	0.000	2881	73.5
20	2	0.8	1169.523	opt	0.009	0.082	211	0.000	0.000	2690	74.8
20	2	1.0	1210.077	opt	0.013	0.086	224	0.000	0.000	2839	75.0
20	3	0.2	724.538	opt	0.017	0.143	221	0.000	0.000	5498	50.8
20	3	0.4	847.767	opt	0.016	0.144	220	0.000	0.000	5634	49.4
20	3	0.6	970.996	opt	0.023	0.148	229	0.000	0.000	5724	50.7
20	3	0.8	1091.050	1094.225	0.056	0.181	278	0.351	0.062	7100	49.1
20	3	1.0	1156.072	opt	0.033	0.154	241	0.055	0.244	5794	52.4
20	4	0.2	577.621	opt	0.032	0.180	232	0.000	0.000	6548	44.3
20	4	0.4	727.099	opt	0.027	0.177	226	0.000	0.000	6524	43.1
20	4	0.6	869.157	opt	0.034	0.185	236	0.000	0.000	6766	43.3
20	4	0.8	1008.492	opt	0.082	0.227	299	0.310	0.352	8082	46.2
20	4	1.0	1111.015	1111.798	0.106	0.249	327	0.278	0.260	8966	45.5
25	2	0.2	1000.907	opt	0.009	0.099	212	0.000	0.000	2763	74.2
25	2	0.4	1101.629	opt	0.014	0.100	219	0.076	0.342	2756	75.0
25	2	0.6	1201.206	opt	0.021	0.108	230	0.000	0.000	2935	74.8
25	2	0.8	1294.085	opt	0.032	0.122	253	0.000	0.000	3628	71.3
25	2	1.0	1359.190	1362.156	0.015	0.107	220	0.218	0.000	2998	73.2
25	3	0.2	767.349	opt	0.025	0.170	223	0.000	0.000	4963	56.1
25	3	0.4	901.699	opt	0.033	0.180	233	0.000	0.000	5293	55.2
25	3	0.6	1033.565	opt	0.045	0.192	249	0.040	0.180	5772	54.1
25	3	0.8	1158.831	opt	0.060	0.206	264	0.032	0.144	6208	53.6
25	3	1.0	1256.630	opt	0.060	0.208	265	0.276	0.566	6173	54.0
25	4	0.2	629.634	opt	0.092	0.258	288	0.123	0.300	7147	50.7
25	4	0.4	787.515	opt	0.062	0.235	257	0.044	0.198	6609	49.0
25	4	0.6	939.206	opt	0.053	0.232	245	0.102	0.456	6519	47.5
25	4	0.8	1087.662	opt	0.056	0.229	249	0.444	0.334	6560	48.1
25	4	1.0	1211.232	opt	0.090	0.261	285	0.000	0.000	7224	49.6

Table 2
GAHUB2 results on CAB instances

n	p	α	*Opt.sol*	*GA best*	t (seconds)	t_{tot} (seconds)	*gen*	*agap* (%)	σ (%)	*eval*	*cache* (%)
20	2	0.2	979.087	opt	0.013	0.097	221	0.000	0.000	2782	75.2
20	2	0.4	1042.566	opt	0.013	0.099	218	0.000	0.000	2804	74.6
20	2	0.6	1106.044	opt	0.011	0.097	213	0.000	0.000	2732	74.7
20	2	0.8	1169.523	opt	0.011	0.360	214	0.000	0.000	2678	75.3
20	2	1.0	1210.077	opt	0.018	0.104	228	0.000	0.000	2910	74.8
20	3	0.2	724.538	opt	0.027	0.152	233	0.000	0.000	4619	60.9
20	3	0.4	847.767	opt	0.029	0.156	238	0.000	0.000	4823	59.9
20	3	0.6	970.996	opt	0.044	0.169	259	0.000	0.000	5241	60.0
20	3	0.8	1091.050	opt	0.055	0.179	281	0.328	0.101	5335	61.8
20	3	1.0	1156.072	opt	0.048	0.173	266	0.000	0.000	5143	61.7
20	4	0.2	577.621	opt	0.028	0.158	232	0.000	0.000	4495	61.8
20	4	0.4	727.099	opt	0.038	0.171	243	0.000	0.000	4834	60.6
20	4	0.6	869.157	opt	0.033	0.168	236	0.000	0.000	4783	60.0
20	4	0.8	1008.492	opt	0.057	0.192	272	0.041	0.099	5499	59.7
20	4	1.0	1111.015	opt	0.100	0.223	332	0.076	0.042	6016	64.1
25	2	0.2	1000.907	opt	0.009	0.117	209	0.000	0.000	2725	74.2
25	2	0.4	1101.629	opt	0.009	0.117	208	0.000	0.000	2758	73.8
25	2	0.6	1201.206	opt	0.029	0.567	241	0.000	0.000	3132	74.3
25	2	0.8	1294.085	opt	0.056	0.160	286	0.139	0.285	3910	72.6
25	2	1.0	1359.190	opt	0.023	0.132	229	0.207	0.049	3268	71.7
25	3	0.2	767.349	opt	0.039	0.200	239	0.000	0.000	4947	59.1
25	3	0.4	901.699	opt	0.041	0.203	242	0.000	0.000	5099	58.3
25	3	0.6	1033.565	opt	0.071	0.230	277	0.000	0.000	5794	58.5
25	3	0.8	1158.831	opt	0.098	0.255	311	0.065	0.199	6326	59.5
25	3	1.0	1256.630	opt	0.124	0.370	337	0.135	0.417	7192	57.4
25	4	0.2	629.634	opt	0.049	0.362	244	0.000	0.000	5189	58.0
25	4	0.4	787.515	opt	0.049	0.221	244	0.000	0.000	5258	57.4
25	4	0.6	939.206	opt	0.054	0.301	248	0.000	0.000	5322	57.5
25	4	0.8	1087.662	opt	0.068	0.240	263	0.068	0.210	5702	56.9
25	4	1.0	1211.232	opt	0.081	0.255	277	0.029	0.103	6128	56.2

Tables 1 and 2 provide results of GA approaches for CAB instances, while Tables 3 and 4 contain results obtained on AP instances. Both GA methods were run 20 times in each instance.

In the first two/three columns instance's dimensions (n, p and possibly α) are given. The next column contains the optimal solution of the current instance, if it is previously known. If it is not, a dash is written. The best value of GA is given in the following column with mark *opt* in cases where GA reached the optimal solution known in advance. The average time needed to detect the best value is given in t column, while t_{tot} represents the total time (in seconds) needed for finishing GA. On average, GA has finished after *gen* generations.

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 where no optimal solution is found (sol_i represents the GA solution obtained in the ith execution). Standard deviation of the average 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 caching technique. Since the hash-queue table is being searched and updated in O(1) time, the percentage of savings cache is very similar to run-time savings in practice.

The GA concept cannot prove optimality and adequate finishing criteria that will fine-tune solution quality does not exist. Therefore, as the columns t_{tot} in Tables 1–4 show, our algorithm runs through additional $t_{tot} - t$ time (until finishing criteria is satisfied), although it already reached its best solution. After all, the total running time of GAHUB2 is reasonably short, in respect that it reaches all optimal/best-known solutions.

It is evident from Tables 1 and 3 that GAHUB1 quickly reaches all previously known optimal solutions, with the exception of three CAB instances.

Table 3
GAHUB1 results on AP instances

n	p	*Opt.sol*	*GA best*	t (seconds)	t_{tot} (seconds)	*gen*	*agap* (%)	σ (%)	*eval*	*cache* (%)
10	2	167,493.065	opt	0.002	0.051	209	0.000	0.000	2148	79.7
10	3	136,008.126	–	–	–	–	–	–	–	–
10	4	112,396.068	opt	0.018	0.108	233	0.000	0.000	7050	40.2
10	5	91,105.371	opt	0.033	0.160	247	0.014	0.044	10,050	19.4
20	2	172,816.690	opt	0.011	0.084	215	0.000	0.000	2825	74.0
20	3	151,533.084	opt	0.020	0.141	222	0.000	0.000	5347	52.5
20	4	135,624.884	opt	0.025	0.175	224	0.000	0.000	6362	44.1
20	5	123,130.095	opt	0.058	0.267	247	0.000	0.000	8450	32.3
25	2	175,541.977	opt	0.034	0.122	258	0.000	0.000	3539	72.4
25	3	155,256.323	opt	0.038	0.185	239	0.023	0.070	5521	54.3
25	4	139,197.169	opt	0.072	0.242	267	0.035	0.024	6664	50.6
25	5	123,574.289	opt	0.033	0.281	220	0.010	0.025	6926	38.0
40	2	177,471.674	opt	0.048	0.185	241	0.000	0.000	3310	72.8
40	3	158,830.545	opt	0.089	0.337	256	0.000	0.000	5975	53.8
40	4	143,968.876	opt	0.052	0.350	224	0.037	0.164	5766	49.2
40	5	134,264.967	opt	0.177	0.573	275	0.118	0.283	8243	40.6
50	2	178,484.286	opt	0.136	0.309	312	0.000	0.000	4125	73.7
50	3	158,569.933	opt	0.126	0.424	261	0.000	0.000	5412	59
50	4	143,378.046	opt	0.196	0.547	284	0.222	0.629	6615	53.7
50	5	132,366.953	opt	0.229	0.733	274	0.172	0.338	7897	42.9
100	2	–	180,223.801	0.510	3.821	2141	0.173	0.661	14,688	86.3
100	3	–	160,847.001	0.941	8.203	2195	0.000	0.000	32,913	70.0
100	4	–	145,896.578	1.230	9.581	2241	0.018	0.037	37,183	66.8
100	5	–	136,929.444	2.146	13.585	2319	0.309	0.309	48,765	58.1
100	10	–	106,829.151	11.147	32.590	2927	1.597	1.150	92,724	36.5
100	15	–	90,534.785	13.179	48.500	2718	1.145	0.625	106,704	21.5
100	20	–	80,471.845	34.362	81.860	3378	0.874	0.540	147,972	12.5
200	2	–	182,459.254	3.795	19.143	2244	0.026	0.030	16,759	85.1
200	3	–	162,887.031	6.652	36.796	2316	0.851	1.080	31,410	72.9
200	4	–	147,767.303	15.155	48.338	2692	0.948	1.342	38,557	71.4
200	5	–	140,450.089	18.344	63.417	2701	0.493	0.646	47,014	65.3
200	10	–	110,648.724	57.337	135.878	3201	1.611	1.387	85,514	46.6
200	15	–	95,857.693	81.918	199.222	3241	1.309	1.007	107,512	33.6
200	20	–	86,069.213	151.197	297.513	3675	1.652	1.042	140,164	23.8

It also provides results on large-scale AP instances ($n = 100, 200$) reported in Table 3. In three cases ($n = 100$, $p = 15$; $n = 100$, $p = 20$ and $n = 200$, $p = 10$), GAHUB1 obtains solutions that are better than the best ones obtained by the existing methods (SA, TABUHUB and PR).

The results presented in Tables 2 and 4 show that GAHUB2 performs even better. It reaches all known optimal solutions on both CAB and AP instances. For large-scale AP instances with an unknown optimal solution, GAHUB2 obtains better (or equal) solutions in comparison with all other heuristics for USApHMP. In Table 5 we present detailed comparisons of GAHUB1 and GAHUB2 with the following heuristic methods:

- Simulated annealing method (SA), proposed in [15], tested on DEC 3000/700 (200 MHz alpha chip).
- Tabu search heuristic (TABUHUB), given in [29,32], tested on Intel XEON 1.4 GHz.
- Path relinking method (PR), proposed in [29], also tested on Intel XEON 1.4 GHz.

The best-known solutions in Table 5 are bolded and underlined. Note that the SA method run 10 times, while the PR and TABUHUB methods run 30 times on each instance. Only the best solutions in the same instances are compared in Table 5, while the corresponding average gap values are not directly comparable, since they are calculated with different best values. For example, on the largest AP instance $n = 100$, $p = 20$ which was solved by all methods, the reported average gap values are: SA 0.24%, TABUHUB 0.26%, PR 0.12%, GAHUB1 0.87% and GAHUB2 0.17%. In order to provide a fair comparison, we scaled the average

Table 4
GAHUB2 results on AP instances

n	*p*	*Opt.sol*	*GA best*	*t* (seconds)	t_{tot} (seconds)	*gen*	*agap* (%)	σ (%)	*eval*	*cache* (%)
10	2	167,493.065	opt	0.003	0.054	206	0.000	0.000	1346	87.1
10	3	136,008.126	opt	0.005	0.074	213	0.000	0.000	2912	73.0
10	4	112,396.068	opt	0.006	0.074	216	0.000	0.000	2645	75.8
10	5	91,105.371	opt	0.013	0.102	221	0.000	0.000	3566	68.1
20	2	172,816.690	opt	0.020	0.107	235	0.000	0.000	3103	73.8
20	3	151,533.084	opt	0.018	0.144	222	0.000	0.000	4305	61.7
20	4	135,624.884	opt	0.032	0.163	236	0.000	0.000	4570	61.8
20	5	123,130.095	opt	0.038	0.213	231	0.000	0.000	5306	54.7
25	2	175,541.977	opt	0.037	0.150	253	0.000	0.000	3850	69.5
25	3	155,256.323	opt	0.039	0.198	238	0.000	0.000	5127	57.4
25	4	139,197.169	opt	0.059	0.233	256	0.000	0.000	5631	56.2
25	5	123,574.289	opt	0.049	0.280	233	0.010	0.025	5787	50.9
40	2	177,471.674	opt	0.143	0.344	316	0.000	0.000	5780	63.4
40	3	158,830.545	opt	0.114	0.404	266	0.000	0.000	6550	51.2
40	4	143,968.876	opt	0.155	0.476	278	0.000	0.000	7212	48.6
40	5	134,264.967	opt	0.351	0.750	344	0.018	0.045	9855	43
50	2	178,484.286	opt	0.237	0.516	342	0.018	0.019	6649	60.9
50	3	158,569.933	opt	0.450	0.808	394	0.010	0.028	9796	50.4
50	4	143,378.046	opt	0.328	0.764	331	0.011	0.044	8738	47.4
50	5	132,366.953	opt	0.514	1.053	365	0.049	0.085	10,445	43.1
100	2	–	180,223.801	4.236	13.293	2854	0.030	0.018	57,731	59.7
100	3	–	160,847.001	4.271	16.640	2638	0.000	0.000	69,337	47.5
100	4	–	145,896.578	3.863	17.761	2513	0.000	0.000	70,939	43.6
100	5	–	136,929.444	5.769	22.108	2652	0.116	0.394	79,894	39.8
100	10	–	106,469.566	18.782	40.733	3494	0.240	0.347	112,386	36.0
100	15	–	90,533.523	28.853	57.453	3744	0.426	0.320	121,000	35.4
100	20	–	80,270.962	43.603	79.200	4173	0.166	0.281	136,157	34.8
200	2	–	182,459.254	51.870	100.722	3906	0.044	0.026	91,264	53.3
200	3	–	162,887.031	46.213	111.774	3329	0.000	0.000	96,475	42.1
200	4	–	147,767.303	61.439	131.161	3531	0.033	0.008	107,273	39.3
200	5	–	140,175.645	89.729	169.733	4022	0.212	0.108	128,659	36.0
200	10	–	110,147.657	182.210	259.142	4794	0.213	0.187	162,768	32.1
200	15	–	94,496.406	217.431	313.017	4829	0.504	0.339	166,280	31.2
200	20	–	85,129.343	310.625	374.751	4937	0.674	0.268	172,360	30.2

gap values by taking the best-known value (reached by GAHUB2) and obtained the following values: SA 0.75%, TABUHUB 0.77%, PR 0.63%, GAHUB1 1.13% and GAHUB2 0.17%.

According to the SPEC_fp95 and SPEC_fp2000 benchmarks (www.spec.org), in comparison with DEC 3000/700, computers Intel XEON 1.4 GHz and AMD 1.33 GHz have average speedup values 6.9 and 7.3 respectively. Considering this, we observe that SA running time, divided by 7.3 is similar to GAHUB1 and GAHUB2 running times. In the same way, we conclude that the TABUHUB and PR methods are significantly slower compared to SA, GAHUB1 and GAHUB2.

We have noticed that both GAHUB1 and GAHUB2 behave quite differently from other methods during the process of searching for an optimal solution. From the related papers [28,29] it can be seen that PR and TABUHUB reach previously known solutions, obtained by SA and reported in [15]. However, PR and TABUHUB do not improve any of the sub-optimal SA solutions. This fact may indicate that these three methods, different in nature, have something in common. We have observed that the local search component dominates in TABUHUB, SA and PR methods. A local search usually quickly reaches good solutions, but later it may be trapped in the local optimum. These three methods use different global strategies to keep the search away from the local optimum. Regarding the computational results, it is obvious that the applied global strategies are not always capable of preventing convergence in sub-optimal solutions, especially on large-scale USApHMP instances.

Instead of an extensive use of the time-consuming local search, GA methods include the ‘‘nearest

Table 5
Comparison of results for larger AP instances

n	p	Ernst SA heur		TABUHUB		Pérez PR		GAHUB1		GAHUB2	
		Best sol.	Alpha 200 MHz	Best sol.	Intel 1.4 GHz	Best sol.	Intel 1.4 GHz	Best sol.	AMD 1.33 GHz	Best sol.	AMD 1.33 GHz
50	5	**opt**	19.5	**opt**	1240	**opt**	204	**opt**	0.733	**opt**	1.053
100	5	**136,929.44**	80.6	**136,929.44**	805	**136,929.44**	301	**136,929.444**	13.585	**136,929.444**	22.108
100	10	**106,469.57**	161.9	**106,469.57**	1259	**106,469.57**	396	106,829.151	32.590	**106,469.566**	40.733
100	15	90,605.10	279.8	90,605.10	1480	90,605.10	590	90,534.785	48.500	**90,533.523**	57.453
100	20	80,682.71	522.7	80,682.71	2330	80,682.71	1008	80,471.845	81.860	**80,270.962**	79.200
200	5	140,409.41	399.7	–	–	–	–	140,450.089	63.417	**140,175.645**	169.733
200	10	111,088.33	776.6	–	–	–	–	110,648.724	135.878	**110,147.657**	259.142
200	15	95,460.54	1105.3	–	–	–	–	95,857.693	199.222	**94,496.406**	313.017
200	20	85,560.39	1555.9	–	–	–	–	86,069.213	297.513	**85,129.343**	374.751

neighbour ordering'' search, which does not require additional CPU time, and this may be an explanation for the computational results obtained. GA refines solutions slightly slower in the beginning of the search, but it continues to refine solutions steadily, and finally reaches high quality solutions. Moreover, our genetic-based approach is good at diversifying the search and it is more robust than related work.

From the results presented above, it can be seen that GAHUB2 performs better than GAHUB1. This can be explained in the following way: GAHUB1 uses two-segment representation, while GAHUB2 has one-segment representation. Integer representation of hubs in GAHUB1 implies that information whether a node is a hub or not, and the node's assignment to a hub is separated in different genetic code segments. Each bit in the binary representation of hubs used in GAHUB2 allows us to join that bit with the index of a hub assigned to the current node. In this way, all information about each node is grouped in the genetic code, which implies shorter building-blocks in GAHUB2 compared to GAHUB1. The building-block hypothesis and schema theorem [3,4] both favour shorter building-blocks and binary encoding, which could be an explanation for the results obtained.

7. Conclusions

In this paper, we have described two evolutionary metaheuristics for solving USApHMP. For each method, two-segment encoding of individuals and appropriate objective functions are used. Arranging located hubs in non-decreasing order of their distances from each non-hub node directs GA to promising search regions. The initial population is generated to be feasible and genetic operators adapted to USApHMP are designed and implemented. Genetic operators preserve the feasibility of solutions, so incorrect individuals do not appear throughout all generations. We have used the idea of frozen bits to increase the diversity of genetic material. The caching technique additionally improves the computational performance of both GAs.

The results clearly demonstrate that even for large problems our GA approach surpasses related work with respect to both solution quality and computational time. The results of GAHUB2, better of the two proposed GA methods, match with all previously known optimal solutions. For large-scale

AP instances, best-known solutions from literature are significantly improved.

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 that our GA approaches have a potential as useful metaheuristics for solving other single allocation hub problems and more complex hub location models.

Acknowledgements

This work is partially supported by the Serbian Ministry of Science and Ecology under grant no. 144007. The authors are grateful to Andreas Ernst for useful communication and his most recent results. We would also like to thank three anonymous referees for their valuable comments and suggestions.

References

[1] S. Abdinnour-Helm, M.A. Venkataramanan, Solution approaches to hub location problems, Annals of Operations research 78 (1998) 31–50.
[2] T. Aykin, On modeling scale economies in hub-and-spoke network design, in: Presented at the Fall Conference of INFORMS, Atlanta, 1996.
[3] T. Bäck, D.B. Fogel, Z. Michalewicz, Evolutionary Computation 1: Basic Algorithms and Operators, Institute of Physics Publishing, Bristol–Philadelphia, 2000.
[4] T. Bäck, D.B. Fogel, Z. Michalewicz, Evolutionary Computation 2: Advanced Algorithms and Operators, Institute of Physics Publishing, Bristol–Philadelphia, 2000.
[5] J.E. Beasley, Obtaining test problems via Internet, Journal of Global Optimization 8 (1996) 429–433, Downloadable from website <http://www.brunel.ac.uk/depts/ma/research/jeb/orlib>.
[6] D.L. Bryan, Extensions to the hub location problem: Formulations and numerical examples, Geographical Analysis 30 (1998) 315–330.
[7] D.L. Bryan, M.E. O'Kelly, Hub-and-spoke networks in air transportation: An analytical review, Journal of Regional Science 39 (1999) 275–295.
[8] J.F. Campbell, Integer programming formulations of discrete hub location problems, European Journal of Operational Research 72 (1994) 387–405.
[9] J.F. Campbell, A. Ernst, M. Krishnamoorthy, Hub location problems, in: Z. Drezner, H. Hamacher (Eds.), Location Theory: Applications and Theory, Springer-Verlag, Berlin–Heidelberg, 2002, pp. 373–407.
[10] J.F. Campbell, G. Stiehr, A.T. Ernst, M. Krishnamoorthy, Solving hub arc location problems on a cluster of workstations, Parallel Computing 29 (2003) 555–574.
[11] J.F. Campbell, A.T. Ernst, M. Krishnamoorthy, Hub arc location problems: Part I – Introduction and results, Management Science 51 (2005) 1540–1555.
[12] J.F. Campbell, A.T. Ernst, M. Krishnamoorthy, Hub arc location problems: Part II – Formulations and optimal algorithms, Management Science 51 (2005) 1556–1571.
[13] E. Domnguez, J. Muñoz, E. Mérida, A recurrent neural network model for the *p*-hub problem, Lecture Notes in Computer Science 2687 (2003) 734–741.
[14] T. Dunker, G. Radons, E. Westkämper, Combining evolutionary computation and dynamic programming for solving a dynamic facility layout problem, European Journal of Operational Research 165 (2005) 55–69.
[15] A.T. Ernst, M. Krishnamoorthy, Efficient algorithms for the uncapacitated single allocation *p*-hub median problem, Location Science 4 (1996) 130–154.
[16] A.T. Ernst, M. Krishnamoorthy, An exact solution approach based on shortest-paths for *p*-hub median problems, INFORMS Journal on Computing 10 (1998) 149–162.
[17] V. Filipović, Fine-grained tournament selection operator in genetic algorithms, Computing and Informatics 22 (2003) 143–161.
[18] J. Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press, Ann Arbor, 1975.
[19] B.Y. Kara, B.C. Tansel, On the single-assignment *p*-hub covering problem, Technical report, Department of Industrial Engineering, Bilkent University, Bilkent 06533, Ankara, Turkey, 1999.
[20] J.G. Klincewicz, Heuristics for the *p*-hub location problem, European Journal of Operational Research 53 (1991) 25–37.
[21] J.G. Klincewicz, Avoiding local optima in the *p*-hub location problem using tabu search and grasp, Annals of Operations Research 40 (1992) 283–302.
[22] J. Kratica, Improving performances of the genetic algorithm by caching, Computers and Artificial Intelligence 18 (1999) 271–283.
[23] J. Kratica, Parallelization of genetic algorithms for solving some NP-complete problems (in Serbian), PhD thesis, Faculty of Mathematics, University of Belgrade, 2000.
[24] J. Kratica, Z. Stanimirović, Solving the uncapacitated multiple allocation *p*-hub center problem by genetic algorithm, Asia-Pacific Journal of Operational Research 23 (4) (2006), in press.
[25] R.F. Love, J.G. Moris, G. Wesolowsky, Facility location: Models and methodsPublication in Operations Research, vol. 7, North-Holland, New York, 1988.
[26] V. Marianov, D. Serra, C. ReVelle, Location of hubs in a competitive environment, European Journal of Operational Research 114 (1999) 363–371.
[27] M. O'Kelly, A quadratic integer program for the location of interacting hub facilities, European Journal of Operational Research 32 (1987) 393–404.
[28] M. Pérez, M. Pérez, F. Almeida, J.M. Moreno Vega, Genetic algorithm with multistart search for the *p*-hub median problem, in: Proceedings of the 24th EUROMICRO Conference – EUROMICRO'98, IEEE Computer Society, 2000, pp. 702–707.
[29] M. Pérez, M. Pérez, F. Almeida Rodríguez, J.M. Moreno Vega, On the use of the path relinking for the *p*-hub median problem, Lecture Notes in Computer Science 3004 (2004) 155–164.
[30] H. Podnar, J. Skorin-Kapov, D. Skorin-Kapov, Network cost minimization using threshold-based discounting, European Journal of Operational Research 137 (2002) 371–386.
[31] H. Podnar, J. Skorin-Kapov, Genetic algorithm for network cost minimization using threshold based discounting,

Journal of Applied Mathematics and Decision Sciences 7 (2003) 207–228.
[32] D. Skorin-Kapov, J. Skorin-Kapov, On tabu search for the location of interacting hub facilities, European Journal of Operational Research 73 (1994) 502–509.
[33] D. Skorin-Kapov, On cost allocation in hub-like networks, Annals of Operations Research 106 (2001) 63–78.
[34] D. Skorin-Kapov, J. Skorin-Kapov, Threshold based discounting networks: The cost allocation provided by the nucleolus, European Journal of Operational Research 166 (2005) 154–159.
[35] K. Smith, M. Krishnamoorthy, M. Palaniswami, Neural versus traditional approaches to the location of interacting hub facilities, Location Science 4 (1996) 155–171.
[36] Z. Stanimirović, An efficient genetic algorithm for the uncapacitated multiple allocation *p*-hub median problem, Control and Cybernetics, submitted for publication.
[37] H. Topcuoglu, F. Corut, M. Ermis, G. Yilmaz, Solving the uncapacitated hub location problem using genetic algorithms, Computers and Operations Research 32 (2005) 967–984.