Two Hybrid Genetic Algorithms for Solving the Super-peer Selection Problem

Jozef Kratica, Jelena Kojić, Dušan Tošić, Vladimir Filipović, and Djordje Dugošija *

Abstract The problem that we will address here is the Super-Peer Selection Problem (SPSP). Two hybrid genetic algorithm (HGA) approaches are proposed for solving this NP-hard problem. The 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 data set known from the literature. The results of this test show that in 6 out of 12 cases HGAs outreached best known solutions so far, and that our methods are competitive with other heuristics.

Keywords: Genetic algorithms, Evolutionary computation, Peer-to-peer network topology, Combinatorial optimization.

1 Introduction

During the past 20 years, computer networks have been rapidly expanding because of users' need to connect to other computers via local or global computer networks. Some network strategies are based on client/server architecture, while the others are Peer-to-Peer (P2P) systems. P2P networks

Jozef Kratica
Mathematical Institute, Serbian Academy of Sciences and Arts, Kneza Mihajla 36/III, 11 001 Belgrade, Serbia, e-mail: jkratica@mi.sanu.ac.yu

Jelena Kojić · Dušan Tošić · Vladimir Filipović · Djordje Dugošija
University of Belgrade, Faculty of Mathematics, Studentski trg 16/IV, 11 000 Belgrade, Serbia, e-mail: k_jelena@yubc.net, tdusan@mi.sanu.ac.yu, vladaf@matf.bg.ac.yu, dugosija@matf.bg.ac.yu

* This research was partially supported by Serbian Ministry of Science under the grant no. 144007.

are fully decentralized and the advantages of this approach are self-organizing and fault-tolerant behavior. Failure in a single node usually does not affect the entire network at once. However, excessive growth of network communication may affect the scalability of such networks. Namely, in the case of larger networks, communication times tend to increase and the load put on every node grows significantly. This problem can be solved by the introduction of super-peers. Super-peers are peers that act as servers for a number of attached common peers, while at the same time, forming a network of nodes equal among themselves. In this super-peer P2P network, each common peer is assigned to exactly one super-peer, which represents its only link to the rest of the network. Obviously, all communications between different peers must be routed via at least one super-peer. The presentation of all relevant information about Peer-to-Peer networks is out of the scope of this paper and details can be found in [6, 14].

Although there are many papers about P2P networks, we have found only two papers that deal with the selection of super-peers in order to minimize overall network communication. One of the papers [12] proves that the Super-Peer Selection Problem (SPSP) is NP-hard. The other paper [13]] suggests a solution to this problem. A new metaheuristic based on evolutionary techniques combined with local search is presented. The proposed metaheuristic is tested on real-world instances based on actual internet distance measurements. Significant savings in total communication costs are demonstrated for all instances in contrast to a case without super-peer topology.

2 Mathematical Representation

We used an integer programming representation of SPSP similar to the one in [13]. Since the number of nodes assigned to a super-peer network is limited, we incorporated this into the model. We defined n as the number of nodes, p as the number of super-peer nodes, and d_{ij} as the distance between nodes i and j. Let $x_{ij} \in \{0, 1\}$ have the value 1 if node i is allocated to a super-peer j and 0 otherwise. The condition $x_{kk} = 1$ implies that the node k is a super-peer.

The problem can be expressed thus:

$$min \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \sum_{k=1}^{n} \sum_{l=1}^{n} (d_{ik} + d_{kl} + d_{lj}) x_{ik} x_{jl} \tag{1}$$

subject to:

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

$$\sum_{k=1}^{n} x_{ik} = 1 \quad \forall i = 1, ..., n \tag{3}$$

$$\frac{p}{2} \cdot x_{kk} \leq \sum_{i=1}^{n} x_{ik} \leq 2p \cdot x_{kk} \quad \forall k = 1, ..., n \tag{4}$$

$$x_{ik} \in \{0, 1\} \quad \forall i, k = 1, ..., n \tag{5}$$

The objective function (1) minimizes the overall costs. The constraint (2) ensures that exactly p super-peers are chosen, while the constraint (3) guarantees a single super-peer allocation for each node. The constraint (4) limits the number of nodes assigned to each super-peer k to between $p/2$ and $2p$. The constraint (4) implies also that if k is a non-super-peer node ($x_{kk} = 0$), then $x_{ik} = 0$ holds for every i . It is easy to see that the flow goes only via super-peer nodes, thus preventing direct transmission between other nodes. By the constraint (5) we prevent super-peers being allocated to other nodes.

3 Proposed Hybrid GA Methods

GAs are stochastic search techniques imitating some spontaneous optimization processes in the natural selection and reproduction. At each iteration (generation), a GA manipulates a set (population) of encoded solutions (individuals), starting from either randomly or heuristically generated one. Individuals from a current population are evaluated using a fitness function to determine their qualities. Good individuals are selected to produce new ones (offspring), applying operators inspired from genetics (crossover and mutation), and they replace some of the individuals from the current population. A detailed description of GAs is out of this paper's scope and can be found in [9, 8, 10]. Extensive computational experience of various optimization problems shows that a GA often produces high quality solutions within a reasonable time. Some of the recent applications related to this problem are [4, 5, 11, 7].

3.1 Description of HGA1 Representation

The genetic code of each 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 super-peer, 0 if it is not. Considering these bit values, we form an array of opened super-peers. If a node is a super-peer the remaining bits of the gene are ignored, while if it is not a super-peer node, the remaining part of the gene r_{nsp} refers to the super-peer assigned to the current non-super-peer

node nsp. Details of this assignment will be explained later as the nearest neighbour ordering.

3.2 Description of HGA2 Representation

In this representation of the GA, the genetic code of an individual consists of two segments. The first segment is a string of length p, where the digits (genes) take values from the set $\{0, 1, ..., n-1\}$. Each digit in this segment shows which nodes are set as super-peers. The duplication of a super-peer index is resolved in the following way: if an index repeats in a genetic code, we replace it by the next previously unused one. If there are no such indices, we use the previous index that was not already taken. Since p is smaller than n, we will always find a "free" index to replace the duplicated one. This approach ensures that exactly p distinct super-peer indices are obtained from a genetic code.

The second segment in the genetic code has exactly $n-p$ genes, where each gene r_{nsp} corresponds to the non-super-peer node nsp. The gene value refers to the super-peer assigned to the current node, also applying the nearest neighbor ordering.

3.3 Objective Function

For each non-super-peer node nsp the nearest neighbor ordering is applied in several steps:

Step 1 super-peers that reached $2p$ assignments are discarded from consideration;
Step 2 the remaining super-peers are arranged in non-decreasing order of their distances from a particular node;
Step 3 the super-peer with the index r_{nsp} in the arranged order is assigned for the current non-super-peer node nsp.

This procedure guarantees that every super-peer node has at most $2p$ node assignments. If some super-peers have less than $p/2$ node assignments, we look for nodes that are assigned to super-peers with more than $p/2$ assignments and choose the one with the best change of overall cost. This is repeated until all super-peers have at least $p/2$ node assignments.

The super-peers that are closer to non-super-peer nodes appear often in the optimal solution, while the far away super-peers are rare. For this reason, we directed our search to "closer" super-peers, while the "distant" ones were considered of small probability. As a result, the values of r_{nsp} had to be

relatively small, otherwise the nearest neighbor ordering would become a classical GA search. If all $r_{nsp} = 0$, this would turn into a greedy search.

On this solution, we applied local search procedures as proposed in [13]. There were three different neighborhoods: replacing the super-peer, swapping two peers and reassigning a peer to another super-peer. Since their running times were very time consuming, we used these local search procedures occasionally, in contrast to [13], where all three procedures were used on each individual in every generation. In the first 20 generations, we did not use the local search at all. Later, if the best individual was changed, we improved it, but if the best individual was not changed in the last 5 generations, we randomly chose some individual and applied all three local search procedures to it.

3.4 Selection

The selection operator, which chooses parent individuals for producing offspring in the next generation, is an improved tournament selection operator known as the fine-grained tournament selection (see [2]). This operator uses a real (rational) parameter F_{tour} denoting the desired average tournament size. Two types of tournament were performed: the first type was held k_1 times on $\lfloor F_{tour} \rfloor$ individuals, while the second type was applied k_2 times with $\lceil F_{tour} \rceil$ individuals participating, so $F_{tour} \approx \frac{k_1 \cdot \lfloor F_{tour} \rfloor + k_2 \cdot \lceil F_{tour} \rceil}{N_{nnel}}$, where N_{pop} and N_{elite} was the overall number of individuals and number of elitist individuals in the population, and $N_{nnel} = N_{pop} - N_{elite}$.

In our implementation $F_{tour} = 5.4$ and, corresponding values k_1 and k_2 for $N_{nnel} = 50$ non-elitist individuals were 30 and 20, respectively.

3.5 Crossover and Mutation

After a pair of parents was selected, the crossover operator was applied to them producing two offspring. The crossover operator in HGA1 simultaneously traced genetic codes of the parents from right to left, searching the gene position i where the parent1 had "1-bit" and the parent2, "0-bit" on the first bit position of these genes. The individuals then exchanged whole genes on the gene position i. The corresponding process was simultaneously performed on gene position j, starting from the left side of the parents' genetic codes. The described process was repeated until $j \geq i$.

Note that the number of located super-peers in both offspring remained unchanged compared to their parents. Since the parents were correct, their offspring were correct, too. The implemented crossover operator was applied with the rate ($p_{cross} = 0.85$).

In HGA2 each parent's genetic code consisted of two segments of different nature. We applied a double one-point crossover: in each segment of the parents' genetic codes, one crossover point was randomly chosen and the genes were exchanged after the chosen position.

The offspring generated by a crossover operator were subject to mutation. The mutation operator, when applied, changed a randomly selected gene in the genetic code. During the GA execution, it is possible that 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 significant, the search space becomes much smaller, and the possibility of premature convergence rapidly increases. For this reason, basic mutation rates are increased, but only for the frozen genes. The basic mutation rates are:

- $0.2/n$ for the bit in the first position.
- $0.05/n$ for the bit in the second position. Next bits in the gene have repeatedly two times smaller mutation rate ($0.025/n$, ...).

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

- 2.5 times higher rate ($0.5/n$ instead of $0.2/n$) if they are in the first position in the gene.
- 1.5 times higher rate ($0.075/n$, $0.0375/n$, ...) otherwise.

In HGA1, the previous process could not guarantee the feasibility of individuals, so we counted and compared the number of mutated ones and zeros in the first bits of genes in each individual. In cases where these numbers were not equal, it was necessary to mutate additional leading bits of genes. Equalizing the number of mutated ones and zeros in leading positions, the mutation operator preserved exactly p super-peers and preserved the feasibility of the mutated individuals.

3.6 Caching GA

The main purpose of caching was to avoid the calculation of objective values for individuals that reappeared during the a GA run. The evaluated objective values were stored in a hash-queue data structure, which was created using the Least Recently Used (LRU) caching strategy. When the same code was obtained again, its objective value was taken from the hash-queue table, instead of calculating its objective function. The implemented caching technique improved the GA running time (see [3]). The number of cached objective values in the hash-queue table was limited to $N_{cache} = 5000$ in our HGA implementations.

3.7 Other GA Aspects

The population numbered 150 individuals, but for the applied encodings, random population in the first generation was not appropriate. These encodings, based on the nearest neighbor ordering, as described above, showed that numbers r_{nsp} must be small. If pr_i, $i = 0, ..., p-1$ is the probability that $r_{nsp} = i$, for every non-super-peer node must hold $\sum_{i=1}^{p-1} pr_i = 1$ and $pr_0 \geq pr_1 \geq ... \geq pr_{p-1}$. The appropriate model was the geometric progression for $pr_0 = \frac{1-q}{1-q^p}$, where q was the common ratio. In our case, q was 0.4. The initial population was generated according to these probabilities.

Steady-state generations were replaced by applying elitist strategy. In this replacement scheme, only $N_{nonel} = 50$ individuals were replaced in every generation, while the best $N_{elite} = 100$ individuals directly passed into the next generation preserving highly fitted genes. The elite individuals did not need recalculation of objective values, since each of them was evaluated in one of the previous generations.

Duplicated individuals were removed from each generation. Their fitness values were set to zero, so that the selection operator prevented them from entering the next generation. This was a highly effective method of preserving the diversity of genetic material and keeping the algorithm away from premature convergence. Individuals with the same objective function but different genetic codes can in some cases dominate the population. If their codes are similar, the GA can lead to a local optimum. For that reason, it is useful to limit their appearance to some constant. In this GA application this constant was set to 40.

4 Computational Results

The GA tests were performed on an Intel 2.66 GHz with 4 GB RAM, under Linux (Knoppix 5.3.1) operating system. The GA stopping criterion was the maximum number of generations equal to 5,000 or at most 2,000 generations without an improvement of the objective value. Testing of both HGA approaches was performed on real world instances from [13] based on node-to-node delay information from PlanetLab [1], a world-wide platform for performing Internet measurements. The dimension of these instances varied from $n = 70$ to $n = 419$ nodes and $p \approx \sqrt(n)$. Note that for these instances triangle inequalities are frequently violated due to different routing policies and missing links most likely due to firewalls (for more details see [1, 13]).

Table 1 contains current best known solutions, based on results from [13] and new results from HGA1 and HGA. The table is organized as follows:

- the first column contains the instance name;

Table 1 Characteristic of SPSP instances

Inst	*n*	*p*	*Best known solution*	
01-2005	127	12	2927946	*new*
02-2005	321	19	18596988	*new*
03-2005	324	18	20642064	*new*
04-2005	70	9	739954	
05-2005	374	20	25717036	
06-2005	365	20	22311442	*new*
07-2005	380	20	31042374	*new*
08-2005	402	21	30965218	
09-2005	419	21	33014358	*new*
10-2005	414	21	32922594	
11-2005	407	21	27902552	
12-2005	414	21	28516682	

- in the second and the third columns are the number of nodes, n, and the number of super-peers, p, respectively;
- current best known solutions are given in the last column. Solutions achieved by HGAs are marked with *new*.

Table 2 Results of the HGA1

Inst	$Best_{sol}$	GA_{best}	t (sec)	t_{tot} (sec)	*gen*	*agap* (%)	σ (%)	*eval*	*cache* (%)
01-2005	2927946	*best*	50.8	73.7	4675	0.126	0.287	128152	45.2
02-2005	18596988	18621460	495.7	660.6	4642	0.417	0.283	146473	37.0
03-2005	20642064	20650488	590.4	730.7	4785	1.733	0.757	137183	42.6
04-2005	739954	*best*	7.5	13.2	3851	0.609	1.480	89222	53.1
05-2005	25717036	25727144	929.7	1098.1	4875	0.285	0.328	148494	39.1
06-2005	22311442	*best*	605.0	854.8	4527	0.175	0.125	132700	41.4
07-2005	31042374	31082632	784.4	992.6	4840	0.332	0.237	136918	43.4
08-2005	30965218	30971996	902.5	1145.0	4793	0.634	0.292	154109	35.7
09-2005	33014358	*best*	924.1	1233.8	4653	0.917	0.441	152024	34.8
10-2005	32922594	32966074	1017.5	1237.6	4778	0.820	0.520	154744	35.1
11-2005	27902552	27942502	909.7	1198.1	4783	1.208	0.662	160861	32.9
12-2005	28516682	28561004	1009.7	1300.8	4849	0.564	0.371	152419	37.2

The HGA1 and HGA2 were run 30 times for each instance and the results were summarized in Tables 2 and 3, respectively. The tables are organized as follows:

- the first two columns contain the instance name and the best known solution from Table 1;
- the third column, named GA_{best}, contains the best HGA solutions. The solutions that are equal to the best known are marked *best*;

- the average running time (t) used to reach the final GA solution for the first time is given in the fourth column, while the fifth and sixth columns (t_{tot} and gen) show the average total running time and the average number of generations for finishing GA, respectively. Note that running time t_{tot} includes t;
- the seventh and eighth columns ($agap$ and σ) contain information on average solution quality: $agap$ is a percentage gap defined as $agap = \frac{1}{20}\sum_{i=1}^{20} gap_i$, where $gap_i = 100 * \frac{GA_i - Best_{sol}}{Best_{sol}}$ and GA_i represents the GA solution obtained in the i-th run, while σ is the standard deviation of gap_i, $i = 1, 2, ..., 20$, obtained by formula $\sigma = \sqrt{\frac{1}{20}\sum_{i=1}^{20}(gap_i - agap)^2}$.
- in the last two columns, $eval$ represents the average number of objective function evaluations, while $cache$ displays savings (in percentages) achieved by using the caching technique.

Table 3 Results of the HGA2

$Inst$	$Best_{sol}$	GA_{best}	t (sec)	t_{tot} (sec)	gen	$agap$ (%)	σ (%)	$eval$	$cache$ (%)
01-2005	2927946	*best*	31.0	52.9	4111	1.386	0.310	88090	57.1
02-2005	18596988	*best*	478.9	597.4	4784	0.457	0.202	106383	55.5
03-2005	20642064	*best*	553.3	675.6	4821	1.437	1.023	107753	55.3
04-2005	739954	*best*	3.6	10.1	2984	0.246	0.068	76212	49.1
05-2005	25717036	25722412	762.0	942.9	4773	1.669	0.546	108867	54.5
06-2005	22311442	22326972	657.3	822.0	4865	0.385	0.232	106119	56.3
07-2005	31042374	*best*	716.0	885.1	4810	4.955	0.995	104526	56.6
08-2005	30965218	31050464	856.9	1053.2	4763	0.542	0.280	111634	53.2
09-2005	33014358	33188802	927.1	1094.5	4783	0.834	0.351	105898	55.7
10-2005	32922594	33054968	959.7	1121.7	4921	1.257	0.444	109063	55.7
11-2005	27902552	27956910	874.0	1042.9	4795	1.707	0.707	113642	52.4
12-2005	28516682	28536494	1014.1	1191.7	4916	0.441	0.269	106272	56.7

5 Conclusions

We have described two evolutionary metaheuristics for solving SPSP. For each method, two-segment encoding of individuals and appropriate objective functions were used. Arranging super-peers in non-decreasing order of their distances from each node directed GA to promising search regions. The initial population was generated to be feasible and genetic operators adapted to SPSP were designed and implemented. Improving heuristic based on local search additionally improved solutions given by genetic algorithms. Genetic

operators preserved the feasibility of solutions, so the incorrect individuals did not appear throughout all generations. We have used the idea of frozen bits to increase the diversity of the genetic material. The caching technique additionally improved the computational performance of both HGAs.

The results clearly demonstrated usefulness of our hybrid GA approaches with new best solutions for six of the SPSP instances. Hence, our future work could concentrate on the parallelization of the HGAs and their hybridization with exact methods. Based on the results, we believe that our HGA approaches have a potential as useful metaheuristics for solving other similar problems that arise in Peer-to-Peer network communications.

References

1. Banerjee, S., Griffin, T.G., Pias, M.: The Interdomain Connectivity of PlanetLab Nodes. In: Barakat, C., Pratt, I. (eds.) Proc. of the 5th International Workshop on Passive and Active Network Measurement, Boston, MA (2004)
2. Filipović, V.: Fine-grained tournament selection operator in genetic algorithms. Computing and Informatics **22**, 143–161 (2003)
3. Kratica, J.: Improving performances of the genetic algorithm by caching. Computers and Artificial Intelligence **18**, 271–283 (1999)
4. Kratica, J., Stanimirović, Z.: Solving the Uncapacitated Multiple Allocation p-Hub Center Problem by Genetic Algorithm. Asia-Pacific Journal of Operational Research **23**, 425–438 (2006)
5. 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 **182**, 15–28 (2007)
6. Li, D., Xiao, N., Lu, X.: Topology and resource discovery in Peer-to-Peer overlay networks. In: Grid and Cooperative Computing GCC 2004. LNCS, vol. 3252, pp. 221228. Springer, Heidelberg (2004)
7. Marić, M.: An efficient genetic algorithm for solving the multi-level uncapacitated facility location problem. Computing and Informatics, (in press)
8. Merz, P.: Memetic Algorithms for Combinatorial Optimization Problems: Fitness Landscapes and Effective Search Strategies. PhD thesis, Department of Electrical Engineering and Computer Science, University of Siegen, Germany (2000)
9. Merz, P., Freisleben, B.: Fitness landscapes and memetic algorithm design. In: Corne D, Dorigo M, Glover F (eds.) New ideas in optimization. McGrawHill, London, pp. 245260 (1999)
10. Mitchell, M. An introduction to genetic algorithms. MIT Press, Cambridge, Massachusetts (1999)
11. Stanimirović, Z.: A genetic algorithm approach for the capacitated single allocation p-hub median problem. Computing and Informatics 27, (in press)
12. Wolf, S.: On the complexity of the uncapacitated single allocation p-hub median problem with equal weights. Technical report, University of Kaiserslautern, Distributed Algorithms Group, Internal Report No. 363/07 (July 2007)
13. Wolf, S., Merz, P.: Evolutionary local search for the super-peer selection problem and the p-hub median problem. In: Bartz-Beielstein et al. (eds.) HM 2007, University of Dortmund, Germany, LNCS, vol. 4771, pp. 1-15 Springer (2007)
14. Yang, B., Garcia-Molina, H.: Designing a super-peer network. In: Proceedings of the 19th International Conference on Data Engineering, Bangalore, India, pp. 4962 (2003)