ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2009, Vol. 38, No. 4

SOME FURTHER EXPERIMENTS WITH THE GENETIC ALGORITHM FOR THE QUADRATIC ASSIGNMENT PROBLEM*

Alfonsas Misevičius, Dalius Rubliauskas, Vytautas Barkauskas

Kaunas University of Technology, Department of Multimedia Engineering
Studentų St. 50, LT–51368 Kaunas, Lithuania
e-mail: alfonsas.misevicius@ktu.lt, dalius@soften.ktu.lt, vytautas.barkauskas@ktu.lt

Abstract. In this paper, some further experiments with the genetic algorithm (GA) for the quadratic assignment problem (QAP) are described. We propose to use a particle-swarm-optimization-based approach for tuning the values of the parameters of the genetic algorithm for solving the QAP. The resulting combined self-adaptive swarm optimization-genetic algorithm enables to efficiently auto-configure the control parameters for GA — which leads to excellent quality solutions, especially for the real-life like (structured) QAP instances.

Keywords: combinatorial optimization, heuristics, particle swarm optimization, genetic algorithms, quadratic assignment problem.

Introduction

The quadratic assignment problem (QAP) [10] is formulated as follows. Given two matrices $\boldsymbol{A} = (a_{ij})_{n\times n}$, $\boldsymbol{B} = (b_{kl})_{n\times n}$, and the set Π of all possible permutations of the integers from 1 to n, find a permutation $\pi = (\pi(1), \pi(2), ..., \pi(n)) \in \Pi$ that minimizes the following function:

$$z(\pi) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} b_{\pi(i)\pi(j)} \ . \qquad (1)$$

The quadratic assignment problem is a typical combinatorial optimization problem, where solutions are represented by permutations and the objective function is defined according to the above formula. The QAP is also a good testing platform for different intelligent optimization techniques.

Due to its quadratic nature, this problem appears to be one of the most difficult problems in the class of combinatorial optimization. Since there are no efficient (polynomial time) exact algorithms for the QAP, the attention of researchers is concentrated on designing heuristic methods. Genetic algorithms (GAs) are one of the examples of such methods [1, 5, 11, 12, 13, 22, 25, 30, 31]. In this work, we propose the hybrid genetic-tabu search algorithm to be used in conjunction with a particle-swarm-optimization-based algorithm. Swarm optimization algorithm efficiently finds a good configuration of the values of the control parameters and the tuned genetic algorithm quickly and productively searches for high-quality solutions of the QAP.

The remaining part of this paper is structured as follows. In Section 1, an approach of combining particle swarm optimization and genetic algorithm is outlined. The results of the computational experiments with the proposed approach on the random and real-life like QAP instances taken from the QAP library QAPLIB are presented in Section 2. Section 3 completes the paper with concluding remarks.

1. The approach

Genetic algorithms are among the most powerful tools in various areas of computer science, including optimization. The basic concepts of GAs were developed as far back as the 1970s [8], however the investigations in this field still are continued in both theoretical and experimental studies. GAs are nature-inspired population-based algorithms, where determinism (deterministic recombination of parents' genetic code) is in harmony with randomness (random mutations applied to individuals' genetic information). One of the main important features of GAs is the flexibility and the conceptual richness of the paradigm. Many modifications and extensions of the canonical GA scheme are possible, among them: hybridization (combining genetic and local/heuristic search), various parents' selection, recombination variations, different population replacement variants, using parallel populations, in-

* This work is supported by Lithuanian State Science and Studies Foundation (grant No. T-09293).

corporating a new sort of operations, and so on (see, for example, [7, 24, 26] for thorough description of the genetic algorithms).

The main philosophy of our approach is in proposing a parameter-free like (self-tuning) genetic algorithm without the need for tuning the algorithm to match an optimization problem at hand. In particular, we propose a particle swarm optimization (PSO) based algorithm to be used in conjunction with the hybrid genetic-tabu search algorithm for the solution of the QAP. More precisely, the swarm optimization algorithm effectively finds a good configuration of the values of the control parameters for the genetic algorithm. Then, the fine-tuned genetic algorithm – which itself can be viewed as a black-box function for the particle swarm optimizer – successfully searches for excellent quality solutions of the QAP. Over many iterations, the swarm optimization procedure generates more and more tuned configuration of the control parameters for GA. This, in turn, increases the performance of the genetic search. In addition, there is no need in human intervention, i.e. "manual repairing" of the values of the parameters — which would be very exhausting and require a huge amount of time. The only exception is choosing the total number of generations of GA and setting the ranges (limits) for varying the values of the parameters. The overall global optimization process is continued in an iterative way until the iteration number reaches a predetermined maximum available number of global iterations.

1. 1. Swarm optimization

There is some quite tight analogy between swarm optimization [9] and genetic algorithms. Indeed, a swarm of particles may be thought of as a population of individuals, and moving the swarm in a space might be associated with evolution of population in time. Our PSO algorithm was designed in such a way that every particle corresponds to a certain configuration of the control parameters for GA. In particular, we experimented with the fixed number of tunable parameters, *NTP*, which is equal, in our case, to 12 (see Tables 1 and 2).

Table 1. Control parameters for the genetic algorithm

	Parameter	**Domain (ranges)**	**Type**
1	Population size (*PS*)	[2, *n*]	discrete
2	Number of offspring per generation (N_{offspr})	[1, *n*/2]	discrete
3	Number of improving iterations (Q)	[1, 10]	discrete
4	Depth of the search (W)	[0.1*n*, 10*n*]	discrete
5	Selection factor (σ)	[1, 2]	continuous
6	Mutation rate (ρ)	[0.01, 0.99]	continuous
7	Population improvement factor (γ)	[1, 10]	discrete
8	Entropy (variability) threshold (*ET*)	[0, 1]	continuous
9	Variant of invasion	{1, 2, 3, 4}	discrete
10	Elitism flag	{FALSE, TRUE}	boolean
11	Gender flag	{FALSE, TRUE}	boolean
12	Restart variant	{1, 2}	discrete

Table 2. Structure of particle

p
p(1)
p(2)
p(3)
p(4)
p(5)
p(6)
p(7)
p(8)
p(9)
p(10)
p(11)
p(12)

Every particle thus "consists" of *NTP* coordinates (components), where each coordinate is associated with the single parameter. For the convenience, the values of the coordinates $p(i)$ ($i = 1, 2, \ldots, NTP$) are all from the unite interval [0, 1], so that the particle p may be viewed as being contained in a hypothetical *NTP*-dimensional unite hypercube $[0, 1]^{NTP}$. The values of $p(i)$ ($i = 1, 2, \ldots, NTP$) are optimized by using a modified version of the swarm optimization based self-adaptive algorithm, as described in [23]. The pseudo-code of this version of the algorithm is shown in Figure 1. In the experiments, we set the swarm size to be equal to 5 and the number of global iterations to be equal to 20.

The current configuration of the parameters for the genetic algorithm is obtained by using the following formula: $param(i) = range_{lower}(i) + p(i)(range_{upper}(i) - range_{lower}(i))$; where $i = 1, 2, \ldots, NTP$; $param(i)$ denotes the actual value of the ith parameter, $p(i)$ is the scalar value for the ith coordinate of particle p, $range_{lower}(i)$, $range_{upper}(i)$ denote the lower and upper limits for the range of the values of the ith parameter, respectively. (For example, $PS = param(1) = np(1) + 2(1 - p(1))$, provided that the minimum and maximum available sizes of the population are equal to 2 and n, respectively.) In case of discrete (integer) parameters, we simply round up values of $p(i)$ to the closest integer number. In case of boolean values (FALSE, TRUE), we associate 0 with FALSE and 1 with TRUE.

The lower and upper limits for the parameters (see Table 1) are defined once before running the algorithm.

```
procedure SwarmOptimizationAlgorithm;
//input: Glob_Iter_N – number of global iterations, SS – swarm size (number of particles)
//output: best parameter configuration found, best QAP solution found
begin
  generate initial swarm S in a random way (|S| = SS);
  for iteration_number := 1 to Glob_Iter_N do //main cycle
    for every particle p from S do begin
      randomly select p~ ∈ S such that p~ ≠ p;
      for every coordinate i of particle p do begin
        Δ := |p(i) – p~(i)|;
        generate random value ξ ∈ [–Δ, Δ];
        p'(i) := max{0, min{p(i) + ξ, 1}}
      endfor;
      if z(GeneticAlgorithm using p') < z(GeneticAlgorithm using p) then p := p';
      //p, p' are to be converted to corresponding configurations of the control parameters
      //before using GeneticAlgorithm
  endfor;
  return best found parameter configuration and QAP solution
end.
```

Figure 1. High-level pseudo-code of the swarm optimization algorithm

1. 2. Genetic algorithm

The genetic algorithm used in this work is a slight extension of the algorithms described in [15,17,21]. In fact, it is a hybrid genetic-tabu search algorithm with so-called invasions and some other extensions (like gender modification). As a local search algorithm (improving algorithm), we use the fast version of iterated tabu search (ITS) [16,19,20], which consists of enhanced tabu search algorithm and special mutation procedure. As the solutions already undergo mutations within the ITS algorithm, there is no need in mutations within GA itself (except the restart invasion (see below)). The number of improving iterations, depth of the search, and mutation rate are the tunable parameters of ITS (the tabu list size is between $0.1n$ and $0.5n$, where n is the problem size). Concerning invasions, they can be seen as a specific sort of conceptual operations – some artificial transformations of individuals (or groups of individuals), which may be of both "constructive" or "destructive" nature, for example, "initial burst" ("pre-improvement"), parent reinforcement, reinforced improvement of offspring, combined parent-offspring improvement, periodic recreation, restart invasion (see also [18]).

Every time, the genetic algorithm starts with the current configuration of the control parameters actually produced by the swarm optimization procedure. Each time, GA is initiated by creation of a new fixed-size initial population. If the "initial burst" is on, the improving algorithm is applied for every solution of the starting population to obtain higher quality population. In addition, we increase the number of improving iterations (the number of iterations of the ITS algorithm) by a factor of γ (improvement factor). (At the same time, the number of generations of GA should be correspondingly decreased.)

For the parents selection, rank based selection rule [29] is applied. In case of reinforcement, one randomly selected parent undergoes the additional improvement. Also, if the parameter "*Gender*" is on, then the parents are "forced" to be of opposite sex. (All one needs in this particular case is the additional bit per individual identifying the gender – male or female. In this case, it may be convenient (but not necessary) that the number of the population members is even and the number of males is equal to the number of females [4].) The offspring are produced by cohesive crossover proposed by Drezner [5]. N_{offspr} children are produced at each generation. Every produced offspring is improved by the ITS algorithm. After improving, it is checked if the new obtained offspring is better than its parents: if this is not the case, the offspring is additionally reinforced (this is done by simply increasing the number of improving iterations); otherwise, the algorithm continues in an ordinary way. (By using the reinforced improvement of parents and/or offspring, the number of generations should be accordingly decreased to prevent the significant increase in run time.)

After this, the replacement of the population takes place to determine which individuals survive to the next generation. There are two strategies: "$\mu + \lambda$" (elitism) and "μ, λ" (no elitism) (here, μ denotes the population size (PS) and λ is the number of newly created individuals (N_{offspr})). In the first case, the new offspring replace the corresponding number of the worst individuals of the old population. In the second case, the new individuals (children) simply replace their predecessors (parents) (the fitness of the children is not taken into consideration).

Then, the genetic variability test [15] is performed before moving to the next generation. In particular, we test if the entropy of the population is less than the pre-defined threshold, *ET*. If it is, then restart invasion is activated; otherwise, the algorithm continues with the current population. During the first phase of invasion, we apply the mutation procedure with the increased mutation rate. Optionally, one randomly chosen individual undergoes mutation with maximally available strength, which means that all the genes change their loci in the chromosome. Mutations are applied to all individuals but the best. During the second phase of invasion, the improving algorithm is used for all mutated members to keep high quality population.

In addition, the periodic recreation of individuals takes place every ω generations, where ω is the parameter for controlling the frequency of recreation.

The overall process is continued until the pre-defined number of generations, N_{gen}, have been performed.

The high level pseudo-code of the genetic algorithm is presented in Appendix (Figure A1).

2. Results of computational experiments

We have examined our genetic algorithm on the benchmark problems taken from the quadratic assignment problem library – QAPLIB [3]. The experiments were performed on a personal computer with an Intel Pentium IV 3 GHz single-core processor. The following types of the QAP instances were tested:

a) uniform random instances (these instances are randomly generated according to a uniform distribution; in QAPLIB, they are denoted by tai20a, tai25a, tai30a, tai35a, tai40a, tai50a, tai60a, tai80a, and tai100a);
b) real-life like instances (they are designed to resemble real world problems (the distribution of the data is not uniform); these instances are denoted by tai20b, tai25b, tai30b, tai35b, tai40b, tai50b, tai60b, tai80b, tai100b, and tai150b).

In order to compare the efficiency of the algorithms, we use time-to-target plots [2]. In this case, for any given "target" solution value and the time to obtain this value, the time-to-target plot represents the probability that the target value will be obtained. So, for a given target value, the run time of the algorithm to achieve this value (or lower) is recorded. This is repeated multiple times and the recorded times are then sorted. With the ith shortest time, a probability $P_i = \dfrac{i - 0.5}{m}$ (i = 1, 2, …) is associated. Here, m is the number of trials (we used m = 30).

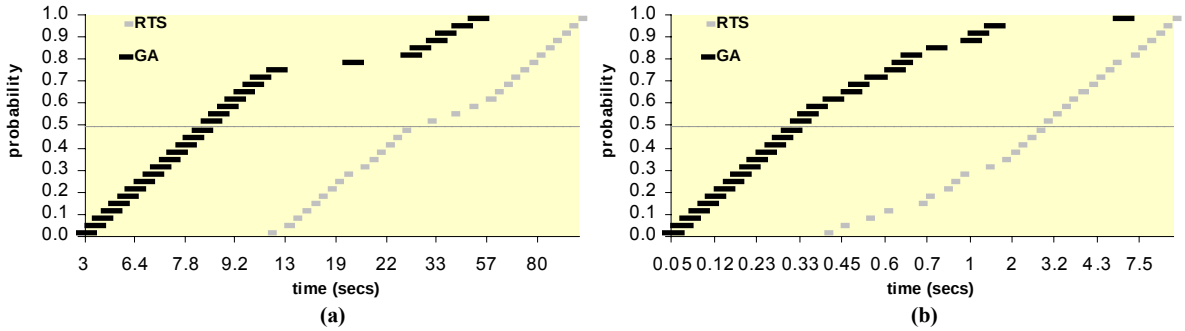

Figure 2. Probabilities of obtaining target values of 3150000 (a) and 637250948 (b) versus time for the instances tai40a (a) and tai40b (b)

In Figure 2, we present the time-to-target plots for our genetic algorithm (GA) tuned with the self-adaptive particle swarm optimization and the robust tabu search (RTS) algorithm by Taillard [27]. The instances examined are tai40a and tai40b and the target values are 3150000 and 637250948, respectively. The first value is about 0.3% above the best known value for the instance tai40a – which is equal to 3139370 (see QAPLIB [3]). The second target value is exactly equal to the best known value (BKV) for the instance tai40b. Such target values were chosen taking into account the results of early preliminary experimentation. During this experimentation, it was detected that the best known (pseudo-optimal) solutions are found quite easily for the real-life like instances (like tai40b); whereas, for the random instances (like tai40a), the time needed to find the best known solutions was drastically long (so far, we were unable to obtain the best known solutions for the instances tai80a and tai100a during our experiments). So, we used to experiment with slightly larger target values for the random instances.

As Figure 2 illustrates, the performance of the genetic algorithm tuned with the particle swarm optimization is clearly better than the one of the tabu search. We observed similar type of behaviour also for all other random and real-life like instances.

Formally, the performance improvement factor, *PIF*, of the genetic algorithm (GA) to the robust tabu search (RTS) algorithm may be defined by the formula $PIF = \dfrac{t_{0.5}(RTS)}{t_{0.5}(GA)}$; where $t_{0.5}$ denotes the time needed to obtain the given target value with probability equal

to 0.5. For example, the performance improvement factor is approximately equal to 3.3 and 9.7 for the instances tai40a and tai40b, respectively. The approximate values of the performance improvement factor for other QAP instances are shown in Table 3.

Table 3. Performance improvement for the random and real-life like QAP instances

Instance	BKV	Target value	*PIF*	Instance	BKV	Target value	*PIF*
tai20a	703482	703482	1.7	**tai20b**	122455319	122455319	2.5
tai25a	1167256	1169000	2.3	**tai25b**	344355646	344355646	3.2
tai30a	1818146	1822000	2.4	**tai30b**	637117113	637117113	3.5
tai35a	2422002	2430000	3.5	**tai35b**	283315445	283315445	5.8
tai40a	3139370	3150000	3.3	**tai40b**	637250948	637250948	9.7
tai50a	4938796	4970000	5.1	**tai50b**	458821517	458821517	10.0

Table 4. Results of the comparison of the algorithms on the random QAP instances

Instance	BKV	$\bar{\delta}$ /c_{bks}†							Time‡
		RTS	FANT	GTS	HGLSA	GGA	FHGA	PSO-GA	
tai20a	703482	0.077/ 6	0.991/ 1	0.410/ 2	1.371/ 0	0.356/ 1	**0.000/10**	**0.000/10**	*0.4*
tai25a	1167256	0.150/ 6	1.358/ 2	0.399/ 2	1.544/ 0	0.428/ 2	0.002/ 9	**0.000/10**	*1.3*
tai30a	1818146	0.081 /5	1.121/ 1	0.378/ 3	1.569/ 0	0.383/ 1	**0.000/10**	**0.000/10**	*3.2*
tai35a	2422002	0.193 /3	1.288/ 0	0.646/ 0	1.698/ 0	0.642/ 0	0.003/ **9**	**0.001/ 9**	*9.8*
tai40a	3139370	0.465 /0	1.556/ 0	0.677/ 0	1.956/ 0	0.707/ 0	0.211/ 1	**0.197/ 2**	*23*
tai50a	4938796	0.793 /0	1.769/ 0	0.872/ 0	2.090/ 0	0.974/ 0	**0.323/ 2**	0.346/ 1	*110*
tai60a	7205962	0.844 /0	1.795/ 0	1.019/ 0	1.982/ 0	1.112/ 0	0.354/ **1**	**0.352/ 1**	*240*
tai80a	13511780	0.825 /0	1.540/ 0	0.800/ 0	1.557/ 0	0.860/ 0	0.465/ 0	**0.451**/ 0	*1200*
tai100a	21052466	0.797 /0	1.472/ 0	0.804/ 0	1.403/ 0	0.955/ 0	0.490/ 0	**0.476**/ 0	*4000*

† number of successful runs (out of 10); ‡ average CPU time per run (in seconds)

Table 5. Results of the comparison of the algorithms on the real-life like QAP instances

Instance	BKV	$\bar{\delta}$ /c_{bks}†							Time‡
		RTS	FANT	GTS	HGLSA	GGA	EHGA	PSO-GA	
tai20b	122455319	**0.000/10**	0.111/ 7	0.048/ 9	0.177/ 5	**0.000/10**	**0.000/10**	**0.000/10**	*0.02*
tai25b	344355646	0.066/ 7	0.015/ 8	**0.000/10**	0.024/ 7	0.008/ 8	0.003/ 9	**0.000/10**	*0.1*
tai30b	637117113	0.407/ 2	0.044/ 7	0.017/ 8	0.509/ 1	0.155/ 5	**0.000/10**	**0.000/10**	*0.3*
tai35b	283315445	0.264/ 4	0.211/ 1	0.156/ 3	0.337/ 0	0.124/ 4	0.004/ 9	**0.000/10**	*0.6*
tai40b	637250948	0.208/ 5	0.018/ 7	**0.000/10**	0.651/ 0	0.010/ 5	**0.000/10**	**0.000/10**	*1.3*
tai50b	458821517	0.271/ 0	0.245/ 0	0.049/ 6	1.003/ 0	0.123/ 3	**0.000/10**	**0.000/10**	*4.1*
tai60b	608215054	0.334/ 0	0.258/ 1	0.039/ 5	0.882/ 0	0.135/ 2	0.001/ 9	**0.000/10**	*6.8*
tai80b	818415043	0.299/ 0	0.396/ 0	0.444/ 2	0.971/ 0	0.240/ 2	0.004/ 8	**0.000/10**	*40*
tai100b	1185996137	0.240/ 0	0.147/ 0	0.149/ 1	0.760/ 0	0.254/ 0	0.005/ 9	**0.000/10**	*120*
tai150b	498896643	0.394/ 0	0.538/ 0	0.409/ 0	0.550/ 0	0.343/ 0	0.048/ **3**	**0.045/ 3**	*700*

† number of successful runs (out of 10); ‡ average CPU time per run (in seconds)

We also used the other performance criterion, the average relative deviation ($\bar{\delta}$) of the solutions from the best known (pseudo-optimal) solution (BKS). It is defined by the formula $\bar{\delta} = 100(\bar{z} - z^{\Diamond})/z^{\Diamond}$ [%], where $\bar{z}$ is the average objective function value over R runs of the algorithm and $z^{\Diamond}$ denotes the best known value (BKV) of the objective function (we used $R = 10$; BKVs are from QAPLIB). We have compared our fine-tuned genetic algorithm with other heuristic algorithms. The algorithms used in the comparison are as follows: robust tabu search (RTS) [27], fast ant system (FANT) [28], genetic-tabu search (GTS) [6], hybrid genetic-local search algorithm (HGLSA) [11], greedy genetic algorithm (GGA) [1], fast hybrid genetic algorithm (FHGA) [14], enhanced hybrid genetic algorithm (manually tuned) (EHGA) [22], and the current hybrid genetic algorithm tuned with the particle swarm optimization (PSO-GA). Note that in these experiments, GA is run alone (with already optimized configuration of the control parameters). The results of the comparison of different

algorithms are shown in Tables 4 and 5. (All algorithms consume roughly the same amount of CPU time (denoted as "Time").)

Finally, we have conducted some additional experiments in an attempt to find out how quickly the tuned genetic algorithm converges to the best known solutions. The experimentation was designed as follows. Let K be the pre-defined number of runs (single applications of the algorithm to a given instance). The parameters of GA are set to optimized values previously obtained by PSO. At every run, the tuned genetic algorithm alone starts from a new random population. The current run is interrupted as soon as BKS is found. The next run is then started, and so on. The overall process stops when K runs have been performed. This is repeated for each instance.

The results of these experiments are presented in Tables 6 and 7. We were not successful in finding the best known solutions for the random instances tai80a and tai100a (instead, we have achieved super-quality target values of 13518124 and 21058106, which are only 0.047% and 0.027% above the best known values). Meanwhile, we have obtained very encouraging results for the real-life instances. For all of these instances, we have found the best known solutions (which all seem to be also the optimal solutions) in pretty reasonable computation times (see Table 7).

Table 6. Run time performance for the random QAP instances

Instance	# of runs	# of BKSs	$Time_{bks}$	Instance	# of runs	# of BKSs	$Time_{bks}$
tai20a	30	30	*0.2* s	**tai40a**	10	10	~*2800* s
tai25a	20	20	*0.8* s	**tai50a**	5	5	~*5* h
tai30a	10	10	*1.5* s	**tai60a**	5	5	~*20* h
tai35a	10	10	*8.0* s	**tai80a, tai100a**	not available		

Note. $Time_{bks}$ denotes the average CPU time (per one run) needed to find the best known solution provided that all runs succeeded in finding the best known solution.

Table 7. Run time performance for the real-life like QAP instances

Instance	# of runs	# of BKSs	$Time_{bks}$	Instance	# of runs	# of BKSs	$Time_{bks}$
tai20b	50	50	*0.01* s	**tai50b**	10	10	*3* s
tai25b	50	50	*0.03* s	**tai60b**	10	10	*4* s
tai30b	30	30	*0.15* s	**tai80b**	10	10	*28* s
tai35b	20	20	*0.30* s	**tai100b**	10	10	*75* s
tai40b	10	10	*0.40* s	**tai150b**	5	5	*2900* s

Note. $Time_{bks}$ denotes the average CPU time (per one run) needed to find the best known solution provided that all runs succeeded in finding the best known solution.

3. Concluding remarks

In this paper, we have proposed a parameter-free like genetic algorithm, which can be used by an end-user without the need of becoming an expert in heuristic optimization algorithms. In particular, the hybrid genetic algorithm is used in conjunction with a particle swarm optimization for solving the quadratic assignment problem.

Our approach was tested on the random and real-life like QAP instances from the library of the QAP instances QAPLIB. The experiments demonstrate that a lot of computation time can be saved by finding suitable control parameter settings. The results obtained from the experiments also show the encouraging performance of PSO-based genetic algorithm and its superiority to other algorithms used in the experimentation. This is especially seen on the real-life like QAP instances.

We think that the proposed combined swarm optimization-genetic-tabu search algorithm is one more step towards self-adaptive, self-controlling, and possibly self-organizing (self-creative) algorithms and systems. The further investigations in this promising area would be worthwhile.

References

[1] **R.K. Ahuja, J.B. Orlin, A. Tiwari.** A greedy genetic algorithm for the quadratic assignment problem. *Computers & Operations Research*, 2000, *Vol*.27, 917–934.

[2] **R.M. Aiex, M.G.C. Resende, C.C. Ribeiro.** Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics*, 2002, *Vol*.8, 343–373.

[3] **R.E. Burkard, S. Karisch, F. Rendl.** QAPLIB – a quadratic assignment problem library. *Journal of Global Optimization*, 1997, *Vol*.10, 391–403. *http://www.seas.upenn.edu/qaplib, cited* 14 *June* 2008.

[4] **T. Drezner, Z. Drezner.** Gender specific genetic algorithms. *INFOR (Information Systems and Operations Research)*, 2006, *Vol*.44, 117–128.

[5] **Z. Drezner.** A new genetic algorithm for the quadratic assignment problem. *INFORMS Journal on Computing*, 2003, *Vol*.15, 320–330.

[6] **C. Fleurent, J.A. Ferland.** Genetic hybrids for the quadratic assignment problem. *In P.M.Pardalos, H. Wolkowicz (eds.), Quadratic Assignment and Related Problems*. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, *Vol*.16, *AMS, Providence*, 1994, 173–188.

[7] **D.E. Goldberg.** Genetic Algorithms in Search, Optimization and Machine Learning. *Addison-Wesley, Reading*, 1989.

[8] **J.H. Holland.** Adaptation in Natural and Artificial Systems. *University of Michigan Press, Ann Arbor*, 1975.

[9] **J. Kennedy, R. Eberhart.** Particle swarm optimization. *Proceedings of the* 4*th IEEE International Conference on Neural Networks*, *Perth, Australia*, 1995, 1942–1948.

[10] **T. Koopmans, M. Beckmann.** Assignment problems and the location of economic activities. *Econometrica*, 1957, *Vol*.25, 53–76.

[11] **M.H. Lim, Y. Yuan, S. Omatu.** Efficient genetic algorithms using simple genes exchange local search policy for the quadratic assignment problem. *Computational Optimization and Applications*, 2000, *Vol*.15, 249–268.

[12] **P. Merz, B. Freisleben.** Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions on Evolutionary Computation*, 2000, *Vol*.4, 337–352.

[13] **A. Misevicius.** An improved hybrid genetic algorithm: new results for the quadratic assignment problem. *Knowledge-Based Systems*, 2004, *Vol*.17, 65–73.

[14] **A. Misevicius.** A fast hybrid genetic algorithm for the quadratic assignment problem. *In M.Keijzer et al. (eds.), Proceedings of Genetic and Evolutionary Computation Conference* (*GECCO*-2006), *ACM Press, New York*, 2006, 1257–1264.

[15] **A. Misevičius.** An entropy-based genetic algorithm. *In L. Sakalauskas, G.W. Weber, E.K. Zavadskas (eds.), Proceedings of the* 20*th International Conference, EURO Mini Conference "Continuous Optimization and Knowledge-Based Technologies"* (*EurOPT* '2008), *Institute of Mathematics and Informatics/VGTU Publishing House "Technika", Vilnius*, 2008, 7–12.

[16] **A. Misevičius.** An implementation of the iterated tabu search algorithm for the quadratic assignment problem. *Working Paper, Kaunas University of Technology, Kaunas, Lithuania*, 2008 (*under review*).

[17] **A. Misevicius.** Restart-based genetic algorithm for the quadratic assignment problem. *In M. Bramer, F. Coenen, M. Petridis (eds.), Research and Development in Intelligent Systems, Proceedings of AI*-2008, *the Twenty-eighth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence, Springer, London*, 2008, 91–104.

[18] **A. Misevičius.** Genetic algorithm with invasions for the quadratic assignment problem. *In A. Targamadzė, R. Butleris, R. Butkienė (eds.), Proceedings of the* 15*th International Conference on Information and Software Technologies, IT*-2009, *Technologija, Kaunas*, 2009, 17–22.

[19] **A. Misevičius, A. Lenkevičius, D. Rubliauskas.** Iterated tabu search: an improvement to standard tabu search. *Information Technology and Control*, 2006, *Vol*.35, *No*.3, 187–197.

[20] **A. Misevičius, A. Ostreika.** Defining tabu tenure for the quadratic assignment problem. *Information Technology and Control*, 2007, *Vol*.36, *No*.4, 341–347.

[21] **A. Misevičius, D. Rubliauskas.** Enhanced improvement of individuals in genetic algorithms. *Information Technology and Control*, 2008, *Vol*.37, *No*.3, 179–186.

[22] **A. Misevičius, D. Rubliauskas.** Testing of hybrid genetic algorithms for structured quadratic assignment problems. *Informatica*, 2009, *Vol*.20, 255–272.

[23] **L. Nolle.** On a control parameter free optimization algorithm. *In M. Bramer, F. Coenen, M. Petridis (eds.), Research and Development in Intelligent Systems* XXV, *Proceedings of AI*-2008, *the Twenty-eighth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence, Springer, London*, 2008, 119–130.

[24] **C.R. Reeves, J.E. Rowe.** Genetic Algorithms: Principles and Perspectives. *Kluwer, Norwell*, 2001.

[25] **J.M. Rodriguez, F.C. Macphee, D.J. Bonham, V.C. Bhavsar.** Solving the quadratic assignment and dynamic plant layout problems using a new hybrid metaheuristic approach. *In M.R. Eskicioglu (ed.), Proceedings of the* 18*th Annual International Symposium on High Performance Computing Systems and Applications (HPCS), Department of Computer Science, Winnipeg*, 2004, 9–16.

[26] **S.N. Sivanandam, S.N. Deepa.** Introduction to Genetic Algorithms. *Springer, Berlin-Heidelberg-New York*, 2008.

[27] **E. Taillard.** Robust taboo search for the QAP. *Parallel Computing*, 1991, *Vol*.17, 443–455.

[28] **E. Taillard.** FANT: fast ant system. *Tech. Report IDSIA*-46-98, *Lugano, Switzerland*, 1998.

[29] **D.M. Tate, A.E. Smith.** A genetic approach to the quadratic assignment problem. *Computers & Operations Research*, 1995, *Vol*.1, 73–83.

[30] **M. Vázquez, L.D. Whitley.** A hybrid genetic algorithm for the quadratic assignment problem. *In L.D. Whitley, D.E. Goldberg, E. Cantú-Paz et al. (eds.), Proceedings of the Genetic and Evolutionary Computation Conference* (*GECCO*'00), *Morgan Kaufmann, San Francisco*, 2000, 135–142.

[31] **Y. Wu, P. Ji.** Solving the quadratic assignment problems by a genetic algorithm with a new replacement strategy. *Proceedings of World Academy of Science, Engineering and Technology*, 2007, *Vol*.24, 310–314.

Received June 2009.

Appendix

```
function GeneticAlgorithm;
//input: n – problem size, A,B – matrices,
//       N_gen – # of generations, PS – population size, N_offspr – # of offspring per generation, Q – # of improving iterations, W – depth of the search,
//       σ – selection factor, ρ – mutation rate, γ – population improvement factor, ET – entropy threshold
//       ω – periodic recreation frequency factor
//       flags: invasion_variant, elitism, gender, restart_variant
//output: π* – best solution found
begin
  create initial population P ⊂ Π (|P| = PS) in two steps:
   (1) generate members of P in a random way;
   (2) optimize each member of P by using the improving algorithm;
   π* := argmin_{π ∈ P} z(π) ; //π* denotes the best so far solution
  for i := 1 to N_gen do begin //main cycle
    sort the members of P according to the ascending order of the values of the objective function;
    P• := ∅;
    for j := 1 to N_offspr do begin //creation of the offspring
      select parents π', π'' ∈ P; if flag gender is on, then the parents must be of opposite gender;
      if invasion_variant = 1 or invasion_variant = 3 then
         apply the improving algorithm to one parent randomly chosen from π' and π'';
      apply crossover to π' and π'', get the offspring π''';
      optimize the offspring π''' by using the improving algorithm;
      if invasion_variant = 2 or invasion_variant = 3 then
        if z(π''') ≥ z(argmax{z(π'), z(π'')}) then apply the improving algorithm to π''';
      P• := P• ∪ {π'''};
      if z(π''') < z(π*) then π* := π''' //saving the best so far solution
    endfor;
    if elitism = TRUE
       then remove N_offspr worst individuals from P ∪ P•, get new population P such that |P| = PS
       else remove N_offspr random individuals from P ∪ P•, get new population P such that |P| = PS;
    if entropy of population P is below τ then begin //restart
      if restart_variant = 1 then begin
        replace one of the members of population (except the best one) by a randomly
        generated solution;
        apply random mutation to all members of P, except the randomly generated solution
        and the best solution;
      endif;
      if restart_variant = 2 then apply random mutation to all members of P, except the best one;
      recreate each member of P by using the improving algorithm;
      if z(argmin_{π ∈ P} z(π)) < z(π*) then π* := argmin_{π ∈ P} z(π)
    endif;
    if invasion_variant = 4 then if i mod ω = 0 then recreate each member of P by using the improving algorithm
  endfor;
  return π*
end.
```

Notes. 1. The genetic algorithm consists, in general, of the selection, crossover, improvement/mutation and replacement operations, which are applied iteratively until a given number of generations are fulfilled. 2. For the QAP, the assignments (permutations) $p_1 = (p_1(1), p_1(2), ..., p_1(n))$, $p_2 = (p_2(1), p_2(2), ..., p_2(n))$, … may be thought of as chromosomes of the individuals; then, the single element $p_i(j)$ corresponds to a gene occupying the jth locus of the ith chromosome. The values of the objective function z are associated with the fitness of individuals. 3. The invasion type is defined by the parameter (flag) "*invasion_variant*". The following are the values of "*invasion_variant*": 1 – parent reinforcement; 2 – reinforced improvement of offspring; 3 – combined reinforced improvement of offspring; 4 – periodic recreation. 4. The initial burst and/or restart invasions are automatically activated if $\gamma > 1$ and/or $ET > 0$. 5. The parameter ω is fixed at value equal to 10.

Figure A1. High-level pseudo-code of the genetic algorithm for the QAP