INFORMATICA, 2009, Vol. 20, No. 2, 255–272

Testing of Hybrid Genetic Algorithms for Structured Quadratic Assignment Problems

Alfonsas MISEVIČIUS
Department of Multimedia Engineering, Kaunas University of Technology
Studentų 50-400a/416a, LT-51368 Kaunas, Lithuania
e-mail: alfonsas.misevicius@ktu.lt

Dalius RUBLIAUSKAS
Department of Multimedia Engineering, Kaunas University of Technology
Studentų 50-401, LT-51368 Kaunas, Lithuania
e-mail: dalius@soften.ktu.lt

Received: February 2008; accepted: February 2009

Abstract. In this paper, an efficient hybrid genetic algorithm (HGA) and its variants for the well-known combinatorial optimization problem, the quadratic assignment problem (QAP) are discussed. In particular, we tested our algorithms on a special type of QAPs, the structured quadratic assignment problems. The results from the computational experiments on this class of problems demonstrate that HGAs allow to achieve near-optimal and (pseudo-)optimal solutions at very reasonable computation times. The obtained results also confirm that the hybrid genetic algorithms are among the most suitable heuristic approaches for this type of QAPs.

Keywords: combinatorial optimization, quadratic assignment problem, heuristics, meta-heuristics, hybrid genetic algorithms.

1. Introduction

The quadratic assignment problem (QAP) can be formulated as follows. Given integer matrices $\boldsymbol{A} = (a_{ij})_{n\times n}$, $\boldsymbol{B} = (b_{kl})_{n\times n}$ and a set Π of permutations of the integers from 1 to n, find a permutation $p = (p(1), p(2), \ldots, p(n)) \in \Pi$ that minimizes $z(p) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} b_{p(i)p(j)}$.

The interpretation of $n, \boldsymbol{A}, \boldsymbol{B}$, and p is as follows (Koopmans and Beckmann, 1957): n represents the number of facilities (economic activities); the entries a_{ij} of the matrix $\boldsymbol{A}$ can be seen as the transportation costs (flows) for one unit of distance from facility i to facility j; $\boldsymbol{B}$ is the matrix that contains distances between all the pairs of locations of the facilities; the permutation $p = (p(1), p(2), \ldots, p(n))$ corresponds to the assignment of facilities to locations, one facility to each location ($p(i)$ denotes the location to which the facility i is assigned); the product $a_{ij} b_{p(i)p(j)}$ may be interpreted as the transportation cost between facilities i and j in locations $p(i)$ and $p(j)$. Thus, solving the QAP means searching for an assignment that minimizes the total transportation cost (z) between all the facilities.

A neighbourhood function Θ: $\Pi \to 2^{\Pi}$ assigns for every $p \in \Pi$ a set $\Theta(p) \subseteq \Pi$ of neighbouring solutions of p. With the QAP, a common way is to usethe 2-exchange neighbourhood function Θ_2 which is defined in the following way: $\Theta_2(p) = \{p' | p' \in \Pi, \rho(p,p') = 2\}$, where ρ is the Hamming distance. (Remind that the Hamming distance between two permutations p and p' may be declared as $\rho(p,p') = |\{i | p(i) \neq p'(i)\}|$.) The solution $p^{\bullet} \in \Pi$ is said to be locally optimal with respect to the neighbourhood Θ_2 if $z(p^{\bullet}) \leqslant z(p')$ for each $p' \in \Theta_2(p^{\bullet})$. The solution $p' \in \Theta_2(p)$ can be obtained from the current solution p by a corresponding move $m(p,i,j)$, which swaps the ith and jth element in the given permutation, i.e., $p'(i) = p(j)$, $p'(j) = p(i)$, $p'(k) = p(k)$, $k = 1, 2, \ldots, n$, $k \neq i$, $k \neq j$. A shorter notation of the form m_{ij} may be used, such that $p' = p \oplus m_{ij}$ means that p' is obtained from p by applying $m(p,i,j)$.

The QAP is known to be NP-hard (Sahni and Gonzalez, 1976) and pose a real challenge for many researchers. Since there are no polynomial-time exact algorithms for this problem, heuristic methods are usually applied. (For the exhaustive surveys of the heuristic algorithms for the QAP, see, for example, (Burkard *et al.*, 1998; Çela, 1998; Loiola *et al.*, 2007; Voß, 2000).) Among the numerous heuristic approaches, genetic algorithms (GAs) (in particular, hybrid GAs (HGAs)) have been proven to be highly efficient in solving the QAP (Ahuja *et al.*, 2000; Drezner, 2003; Fleurent and Ferland, 1994; Lim *et al.*, 2000; Misevicius, 2004, 2006; Vázquez and Whitley, 2000). This is especially true for so-called structured quadratic assignment problems.

Most of structured QAPs are real-life problems from practical applications. As a rule, these problems are of small size ($n \leqslant 32$). In 1995, Taillard proposed the real-life like QAP instances of larger size (up to 150 facilities). These instances are generated pseudo-randomly in such a way that the entries of the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ resemble a distribution from real-world problems (Taillard, 1995). The entries of $\boldsymbol{A}$ and $\boldsymbol{B}$ are thus quite irregular (with many zero values) (see the library of the QAP – QAPLIB; Burkard *et al.*, 1997). On the other hand, there exist some regularities in the solution space. In particular, the landscapes of such problems are rather structured with large basins of attraction and a relatively few number of isolated local optima.

The remaining part of this paper is organized as follows. In Section 2, the hybrid genetic algorithm and its variants for the QAP are described in more details. The results of the experiments on the structured QAP instances taken from the electronic library QAPLIB are presented in Section 3. The paper is completed with concluding remarks.

2. Hybrid Genetic Algorithm and its Variants

2.1. *Preliminaries: Framework of the Hybrid Genetic Algorithm*

Genetic algorithms are based on imitation of the natural process of evolution (Goldberg, 1989). Over generations, less fitted individuals fail to have offspring and disappear, while more fitted ones tend to predominate[1]. Selection, crossover (reproduction), mutation, and

[1]For the QAP, the solutions (permutations) $p_1 = (p_1(1), p_1(2), \ldots, p_1(n))$, $\ldots$, $p_i = (p_i(1), p_i(2), \ldots, p_i(n))$, $\ldots$ 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 fitness of the individuals is directly associated with the objective function z.

replacement (culling) are the standard genetic operations, which are applied in an iterative way to seek super-quality solutions[2].

The main drawback of the genetic algorithms in their canonical form is that the convergence time is slow, especially when large populations are maintained. One of the ways to speed up the convergence is to incorporate additional components into the ordinary GAs. Here, we are speaking of the local search-based algorithms (improving algorithms) which are combined with the standard genetic operators. The resulting genetic-local search algorithms are commonly known as hybrid genetic algorithms.

The basic features of HGAs are as follows. Firstly, a high-quality initial population is created by means of a local search (LS) algorithm (like hill climbing, simulated annealing or tabu search). Secondly, the improving algorithm (as a post-crossover procedure) is applied to each offspring produced. Similarly, the parents may be improved. These features ensure that every population consists solely of the outstanding quality individuals.

For the local improvement of solutions, we use an iterated tabu search (ITS) algorithm, which, in turn, is based on an enhanced tabu search (ETS) procedure (Misevicius, 2005) and a special type of mutation called a chained mutation (CM). Very roughly speaking, ETS iteratively explores the neighbourhood of the current solution by finding a locally optimal solution, while CM randomly perturbs the current local optimum in order to escape from it and move towards new regions in the solution space. The mutated solution is then again improved by ETS, and so on. This type of proceeding helps avoiding stagnation of the search and makes the iterated tabu search algorithm more efficient than the ordinary tabu search algorithms. The other favourable aspect of ITS is that there is no need in the mutation operation within the genetic algorithm itself: each solution already undergoes random transformations during the execution of the ITS algorithm.

We will call the genetic algorithm outlined above as a basic hybrid genetic algorithm. The high-level description of this algorithm is given in Fig. 1. The description of the iterated tabu search algorithm used within HGA is shown in Fig. 2. (In Appendix (Figs. A1, A2) we also present the descriptions of the ETS and CM procedures, which are used in the ITS algorithm.)

Implementing the hybrid genetic algorithm in a straightforward naive manner does, however, not necessarily imply that good solutions are obtained at reasonable run (CPU) time. Further, we will demonstrate how the performance of the straightforward HGA may be improved considerably by means of introducing new modifications and enhancements.

2.2. *Variants of the Hybrid Genetic Algorithm*

There exist many variations in designing the components of HGA (like the selection, crossover and mutation operators, improving procedures, population replacement rules, etc.). Several variants of HGA will be described below in some more detail.

[2]A more thorough description of the principles of GAs can be found in Goldberg (1989; Reeves and Rowe (2001).

```
procedure HybridGeneticAlgorithm;
// input: $n$ – the problem size, $A, B$ – the flow and distance matrices,
//        $PS$ – the population size, $N_{gen}$ – # of generations,
//        $N_{offspr}$ – # of offspring (crossovers) per generation,
//        $Q$ – # of improvement iterations, $\tau$ – the search depth
// output: $p^*$ – the resulting solution
begin
  create the high quality initial population $P \subset \Pi$ such that $PS = |P|$;
  $p^* = \arg\min_{p \in P} z(p)$; // $p^*$ denotes the best so far solution
  for $i := 1$ to $N_{gen}$ do begin // main cycle of HGA
    sort the members of $P$ in the ascending order of their fitness;
    for $j := 1$ to $N_{offspr}$ do begin // creation of the offspring
      select parents $p', p'' \in P$;
      apply crossover to $p'$ and $p''$, get the offspring $p'''$;
      $p^\bullet :=$ IteratedTabuSearch($p''', Q, \tau$); // improving the solution produced by the crossover operator
      $P := P \cup \{p^\bullet\}$;
      if $z(p^\bullet) < z(p^*)$ then $p^* := p^\bullet$
    end; // for $j$
    update the population $P$
  end // for $i$
end.
```

Fig. 1. Pseudo-code of the hybrid genetic algorithm.

2.2.1. *Compounded approach*

It is very important that the genetic algorithm starts with as good a population as possible. The compounded approach (CA) is along this line of thinking. In the original version of CA (Drezner, 2005), one starts with several populations (sub-populations). The members of every sub-population are independently optimized by the local improvement procedure. So, it is like having evolving parallel populations. Only PS best individuals are then selected from these sub-populations to form the single initial population P ($|P| = PS$). This resembles migration of the best species to a super-quality population.

We may also use only one initial population instead of many sub-populations (this is just the case used in this paper). However, we have to spend much more time at the local optimization phase than in the first case. This may be simply achieved by using the increased number of improving iterations at the initialization phase – Q_{ini}. In our implementation, $Q_{\text{ini}} = cQ$, where Q is the usual number of improving iterations and c is an integer coefficient ($c \geqslant 1$).

Results of computational experiments show the effectiveness of the compounded approach despite of its evident simplicity.

2.2.2. *Quick local search vs. time-expensive local search*

Generally speaking, genetic-local search is, in fact, based on the intensification and diversification policy, where intensification (improving algorithm) aims at concentrating the search into specific (promising) regions of the solution space, while diversification (crossover, mutation) helps to avoid getting stuck at local optima (Blum and Roli, 2003, pp. 292–302). The properly chosen intensification strategy may have a crucial influence

```
function IteratedTabuSearch($(p, Q, \tau)$);
// input: $p$ – the current solution, $Q$ – the number of ITS iterations,
//        $\tau$ – the tabu search depth (the number of ETS iterations)
// parameters: $\eta_{\min}, \eta_{\max}$ – the minimum and maximum mutation levels
// output: $p^*$ – the best solution found
begin
  $\eta_{\min} := \lfloor 0.3 \cdot n \rfloor$; $\eta_{\max} := \lfloor 0.4 \cdot n \rfloor$;
  $p^\bullet$ :=EnhancedTabuSearch($(p, \tau)$);
  // preliminary improvement of the current solution by the enhanced tabu search
  $p^* := p^\bullet$; $\eta := \eta_{\min} - 1$;
  for $q := 1$ to $Q$ do begin // main cycle of ITS
    if $\eta < \eta_{\max}$ then $\eta := \eta + 1$ else $\eta := \eta_{\min}$; // updating the mutation level
    $p^\sim$ :=ChainedMutation($(p^\bullet, \eta)$); // applying perturbation to $p^\bullet$ with the mutation level $\eta$
    $p^\bullet$ :=EnhancedTabuSearch($(p^\sim, \tau)$); // improving the mutated solution by the enhanced tabu search
    if $z(p^\bullet) < z(p^*)$ then begin
      $p^* := p^\bullet$; $\eta := \eta_{\min} - 1$ // saving the best so far local optimum, resetting the mutation level
    end // if
  end; // for $q$
  return $p^*$
end.
```

Fig. 2. Pseudo-code of the iterated tabu search algorithm.

on the resulting efficiency of the hybrid genetic algorithm. In this work, we use the iterated tabu search-based improving algorithm which includes the enhanced tabu search combined with the chained mutation procedure, as mentioned above. In order to have a flexible control of the improvement process, the ITS algorithm is organized according to a $(Q, \tau, 1)$-strategy. In this case, the total number of global iterations is equal to Q, whereas τ is the search depth, i.e., the number of internal iterations of the tabu search. In addition, the mutation procedure is performed once every τ iterations (see Fig. 2). We found during the preliminary experimentation that the solution quality is much more sensitive to the value of Q than τ. We therefore keep τ fixed (more precisely, τ is equal to the problem size n), whereas the intensity of local search (i.e., the total number of ITS iterations) is controlled by increasing (decreasing) the value of Q. This value is of high importance for HGA. If it is small (this is the case of a quick local search), then the convergence time may possibly be slow; if it is large (this is the case of a time-expensive local search), then the overall computational time increases. In our basic variant of HGA, we use the small value of Q (in particular, it is equal to 5). In the expensive variant of HGA, the value of Q is increased (in particular, it is set to 10). (In the last case, it is important that the small population is maintained to save CPU time.)

2.2.3. *Reinforced improvement*

In the reinforced approach, the idea of the combination of genetic and local search is further exploited. The underlying principle is to apply the improving algorithm to the selected parents. This improvement is followed by the crossover procedure (of course, the improving algorithm is applied to the produced offspring, too). With the pre-crossover improvement, we are a bit closer to the nature – indeed, in the real life, only the best (young and healthy) members of a population are usually "licensed" to produce their offspring. The reinforced improvement just takes this aspect into account.

In this work, we tried a slightly modified variation of the above approach. In particular, only one of the parents – the better parent – undergoes the reinforced improvement. It is to prevent the significant increase in the run time of the algorithm. In our implementation, the number of iterations of the reinforced improvement, $Q_{\text{reinforced}}$, is equal to $5Q$, where Q is the usual number of improving iterations.

2.2.4. *Blind selection vs. fitness-based selection*

By selecting parents for the crossover, we can choose between the blind (random) selection and fitness-based selection. So, assume that the members of the population are sorted in the ascending order of their fitness (objective function value). Then, a simple way to achieve the fitness-based selection is to apply a rank based rule (Tate and Smith, 1995). In this case, the actual positions (u and v) of the parents within the sorted population are determined by the formulas: $u = \lfloor \xi_{(1)}^{\sigma} \rfloor$, $v = \lfloor \xi_{(2)}^{\sigma} \rfloor$, $u \neq v$, here $\xi_{(1)}, \xi_{(2)}$ are uniform random numbers from the interval [1, $PS^{1/\sigma}$], where PS is the population size, and σ is a real number in the interval [1, 2] (it is referred to as a selection factor). It is obvious that the better the individual, the larger probability of selecting it for the crossover.

Note that the pure random selection can be easily obtained by setting the value of the selection factor σ to 1. We use blind selection in the basic variant of our hybrid genetic algorithm.

2.2.5. *Gender modification*

The central idea of a gender modification is that the individuals of a population are differentiated according to their gender (Drezner and Drezner, 2006). This modification is easily to implement. All we need to do is to ensure that the parents selected for the crossover are of opposite sex. Only one additional bit (identifying the gender – male or female) per individual is required. Also, it is 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.

2.2.6. *Swap path crossover vs. cohesive crossover*

The recombination of solutions remains one of the critical things by creating state-of-the-art genetic algorithms. In hybrid genetic algorithms, the role of crossover operators is probably even more important. It is highly desirable that the crossover be strong (disruptive) enough to allow to escape the current local optimum and to move to new areas of the search space.

We experimented with two crossover operators: the swap path crossover (SPX; Ahuja *et al.*, 2000) and the cohesive crossover (COHX; Drezner, 2003) slightly modified by the authors of this paper. Let us describe them in more details.

So, let p' and p'' be two solutions (parents). SPX starts at the first (or some random) gene, and the parents (chromosomes) are examined from left to right until all the genes have been considered. If the genes at the position being looked at are the same, one moves to the next position; otherwise, one performs a swap (interchange) of two genes in p' or in

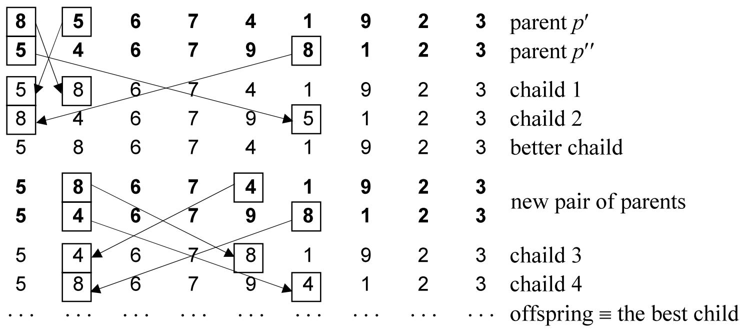
8 5 6 7 4 1 9 2 3 parent p'
5 4 6 7 9 8 1 2 3 parent p''
5 8 6 7 4 1 9 2 3 chaild 1
8 4 6 7 9 5 1 2 3 chaild 2
5 8 6 7 4 1 9 2 3 better chaild
5 8 6 7 4 1 9 2 3
5 4 6 7 9 8 1 2 3
new pair of parents
5 4 6 7 8 1 9 2 3 chaild 3
5 8 6 7 9 4 1 2 3 chaild 4
offspring ≡ the best child

Fig. 3. Example of producing children in the swap path crossover.

p'' so that the values at the current position become the same. For example, if the current location is i and $x = p'(i)$, $y = p''(i)$, then, after a swap, either $p'(i)$ becomes y, or $p''(i)$ becomes x. It is suggested to perform the swap for which the corresponding solution has a lower objective function value. The genes in the two resulting chromosomes are then considered, starting at the next position, and so on. The best solution obtained serves as an offspring. The illustrative example of the swap path crossover is shown in Fig. 3. SPX is used in the basic configuration of HGA.

The key principle of the cohesive crossover is based on maintaining a set of special distance vectors. In Drezner (2003), it is proposed to maintain n vectors $\mathbf{d}^{(1)}, \mathbf{d}^{(2)}, \ldots, \mathbf{d}^{(i)}, \ldots, \mathbf{d}^{(n)}$ such that $d_j^{(i)} = b_{ij}$, $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, n$, where b_{ij} is the "real distance", i.e., the corresponding entry of the matrix $\boldsymbol{B}$.

The ith recombined solution (child) $p^{(i)}$ $(i = 1, 2, \ldots, n)$ is then created in the following four steps:

– the median, ω, of $\mathbf{d}^{(i)}$ is calculated;

– the positions which are closer than the median to the ith (pivot) position are assigned the genes from the first (better) parent, i.e., $p^{(i)}(j) = p_{\text{better}}(j)$ if $d_j^{(i)} < \omega$, $j = 1, 2, \ldots, n$, $p_{\text{better}} = \arg\min\{z(p'), z(p'')\}$;

– all other positions are assigned the genes from the second (worse) parent, i.e., $p^{(i)}(j) = p_{\text{worse}}(j)$ if $d_j^{(i)} \geqslant \omega$, $j = 1, 2, \ldots, n$, $p_{\text{worse}} = \arg\max\{z(p'), z(p'')\}$;

– it is possible that some genes are assigned twice and some are not assigned at all; so, a list of unassigned genes is created and all genes from the second parent that are assigned twice are replaced with genes from the list.

The illustrative example of producing a child in the cohesive crossover is shown in Fig. 4.

There are in all n different children. Only the best child (the child that has the smallest objective function value) is returned by the COHX operator.

2.2.7. *One offspring vs. many offspring*

In the basic variant of HGA, one offspring per generation is produced. However, we may not limit ourselves with a single offspring only. We empirically found that the larger number of the offspring seems to be better than the smaller one. It should however be stressed

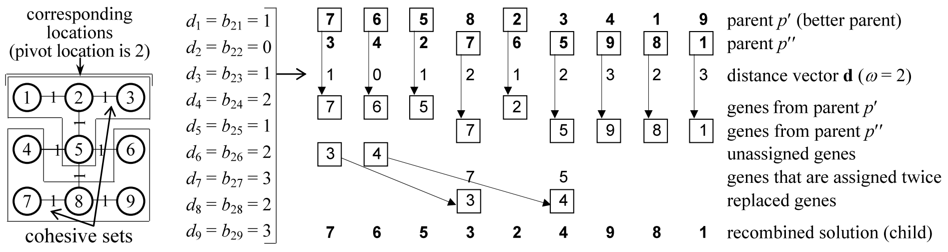

Fig. 4. Example of producing a child in the cohesive crossover.

that an inadequate (enormous) number of the offspring would increase the computation time. In our modified version of HGA, the offspring number (N_{offspr}) is equal to a half of the population size, $\frac{PS}{2}$. Note that if the offspring number is increased, then the number of generations should be accordingly decreased to keep the run time fixed.

2.2.8. *Random replacement vs. elitism*

Two main population replacement strategies (schemes) are termed as the "μ, λ" strategy and "$\mu + \lambda$" strategy. Suppose that the size of a population is equal to μ and the number of newly created individuals is equal to λ(in our algorithm, $\mu = PS$, $\lambda = N_{\text{offspr}}$). Then, in the case of "μ, λ" strategy, λ new individuals replace the corresponding members of the current population. Typically, the child replaces its worse parent. The fitness of the child is not taken into consideration. Thus, we call this strategy as "random replacement". This strategy was used in the basic variant of our genetic algorithm.

In the case of "$\mu + \lambda$" scheme, the individuals chosen for the next generation are the best μ members of $P_\mu \cup P_\lambda$, where P_μ is the population at the beginning of the current generation and P_λ denotes the set of newly created individuals. If $\lambda = 1$, then the single offspring simply replaces the worst member of the population (provided that the offspring is better than the worst population member – otherwise, the offspring is ignored). The last approach is called "elitism".

2.2.9. *Using restarts*

The performance of HGA can be further enhanced by incorporating a restart-based diversification mechanism. The restart-based approach ensures a very high degree of genetic variance of individuals of a population, which is extremely important in avoidance of the premature convergence and stagnation of the genetic process. The other advantage of using restarts is that compact (tiny) populations are enabled, which allow saving both the computational time and memory resources.

More specifically, the restart-based genetic algorithm exploits the entropy of populations – as a quantitative measure of diversity of individuals within populations. The entropy of the population, e, is calculated according to the following formula (Taillard, 1995)

$$e = \sum_{i=1}^{n}\sum_{j=1}^{n} \varepsilon_{ij}/n\log_2 n, \quad \text{where } \varepsilon_{ij} = \begin{cases} 0, & \nu_{ij} = 0, \\ -\frac{\nu_{ij}}{PS}\log_2\frac{\nu_{ij}}{PS}, & \text{otherwise;} \end{cases}$$

here, PS denotes the population size and ν_{ij} is the number of times that the gene i occupies the locus j in the current population. Note that the entropy e takes values between 0 and 1.

Thus, we test whether the entropy of the current population, e, is below a certain small value, ET (entropy threshold). If it is (this just indicates the loss of diversity), then the restart process takes place; otherwise, the algorithm is continued with the current population. After the restart, GA proceeds in the standard way. There are two main steps during the restart: a) mutation of the members of the population, b) improvement of the mutated solutions. Mutation enables to maintain a sufficient degree of diversity within the population, avoiding stagnation in unpromising areas of the solutions space. The goal of the improvement step is to guarantee the local optimality of all the members of the restarted population.

In the next sections, we will use the following short notations for the described variants of HGA: CA – compounded approach, ELS – expensive local search, RI – reinforced improvement, FBS – fitness-based selection, GM – gender modification, COHX – cohesive crossover, MO – many offspring, E – elitism, R – restarts. The basic variant of HGA is denoted as BASIC.

3. Computational Experiments

To investigate the performance of our hybrid genetic algorithm and its variants, a number of computational experiments were carried out. In the experiments, we used the structured QAP instances taken from the publicly available electronic library of the QAP instances – QAPLIB (Burkard *et al.*, 1997). In particular, the real-life like instances proposed by Taillard (1995) were used. In QAPLIB, these instances are denoted by tai20b, tai25b, tai30b, tai35b, tai40b, tai50b, tai60b, tai80b, tai100b, and tai150b (tai*b) (the corresponding numeral in the instance name indicates the size of the problem).

All the experiments were performed on a 900 MHz personal computer. The algorithms were coded by programming language PASCAL.

As a performance criterion for the algorithms, we use the average relative deviation ($\bar{\theta}$) of the solutions from the best known (pseudo-optimal) solution (BKS). It is defined by the following formula: $\bar{\theta} = 100(\bar{z} - z_{\mathrm{bks}})/z_{\mathrm{bks}}[\%]$, where $\bar{z}$ is the average objective function value over 10 runs of the algorithm, while z_{bks} denotes the objective function value that corresponds to BKS (i.e., the best known value – BKV). (BKVs can be found in QAPLIB.)

Firstly, we have conducted preliminary experiments to determine the most suitable values of the control parameters. In particular, we experimented with the different values of population size, number of offspring, selection factor, and entropy threshold. The results of the experimentation are presented in Fig. 5. It may be seen that the population size and the entropy threshold have a considerable influence on the quality of solutions, whereas the effect of the remaining parameters is less significant. The very important observation is that our hybrid algorithms operate with quite miniature populations. For the

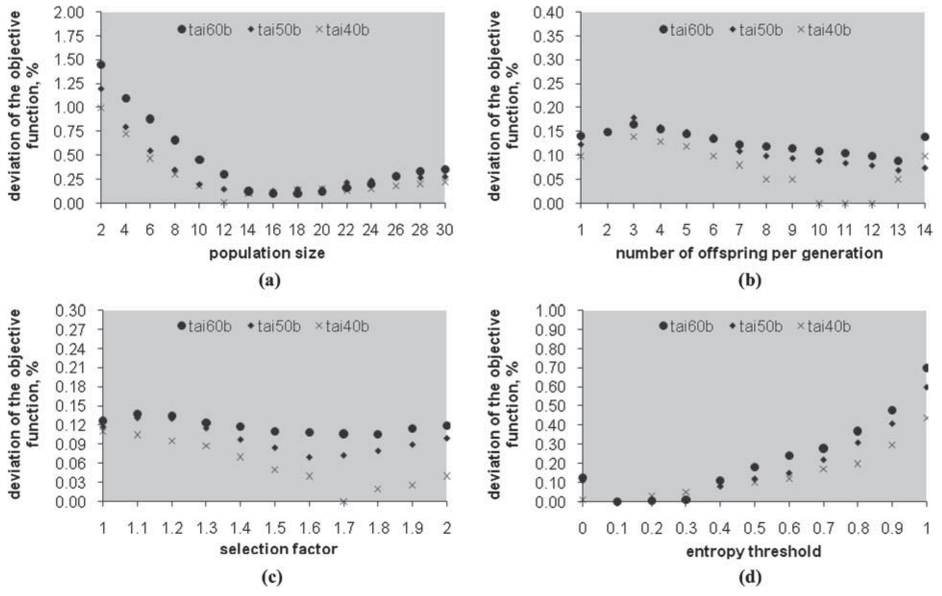

Fig. 5. Results of the preliminary experimentation with the different values of population size (a), number of offspring (b), selection factor (c), and entropy threshold (d). *Note.* Three instances (tai40b, tai50b, tai60b) were used in the experiments.

Table 1

Main control parameters of the basic variant of HGA and its variants

Parameter	Value	Remarks
Total number of generations, N_{gen}	$10n$	1. In CA, $N_{\text{gen}} = 8n$.
		2. In ELS, $N_{\text{gen}} = 6n$.
		3. In RI, $N_{\text{gen}} = 8n$.
		4. In MO, $N_{\text{gen}} = 10n/N_{\text{offsprn}}$
Number of offspring per generation, N_{offsprn}	1	In MO, $N_{\text{offsprn}} = PS/2$
Number of improving iterations, Q	5	1. In CA, $Q_{\text{ini}} = 7Q$.
		2. In ELS, $Q = 10$.
		3. In RI, $Q_{\text{reinforced}} = 5Q$
Tabu search depth, τ	n	
Selection factor, σ	1	In FBS, $\sigma = 1.7$
Entropy threshold, ET	0	In the restart (R) modification, $ET = 0.1$

examined instances, we empirically found that the optimal population size (PS) is proportional to approximately $2\sqrt{n}$, where n is the problem size. So, we used this value in all remaining experiments. The values of the other control parameters of the basic variant of HGA and its modifications are collected in Table 1.

Further, the basic variant of HGA and nine modifications described above were examined. The results of these experiments are presented in Table 2.

Table 2

Results of the comparison of different variants of HGA (part I)

Instance	BKV	$\bar{\theta}$										Time[‡] (s)
		BASIC	CA	ELS	RI	FBS	GM	COHX	MO	E	R	
tai20b	122455319	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	*1.5*
tai25b	344355646	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	*3.3*
tai30b	637117113	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	*6.8*
tai35b	283315445	0.066	0.019	0.056	0.057	0.094	0.075	0.083	0.037	0.102	0.000	*15.0*
tai40b	637250948	0.008	0.000	0.000	0.002	0.000	0.000	0.000	0.000	0.000	0.000	*33*
tai50b	458821517	0.114	0.000	0.066	0.099	0.075	0.243	0.301	0.072	0.071	0.000	*85*
tai60b	608215054	0.124	0.000	0.000	0.126	0.114	0.076	0.000	0.150	0.152	0.000	*200*
tai80b	818415043	0.439	0.166	0.197	0.511	0.537	0.517	0.446	0.455	0.564	0.011	*320*
tai100b	1185996137	0.224	0.098	0.146	0.206	0.196	0.146	0.189	0.108	0.214	0.042	*650*
tai150b	498896643	0.584	0.371	0.390	0.393	0.498	0.501	0.376	0.491	0.424	0.236	*3000*
	Average:	0.156	0.065	0.086	0.139	0.151	0.156	0.140	0.131	0.153	0.029	

[‡] Average time per one run is given.

It can be viewed from Table 2 that CA, ELS, RI, MO and R modifications have a crucial influence on the quality of solutions, whereas the effect of the remaining modifications (FBS, GM, COHX, E) is not so significant. (For example, the fitness-based selection and cohesive crossover are somewhat preferable to the blind selection and swap-path crossover, respectively. The elitism strategy is only slightly better than random replacement, while the performance of gender modification is essentially equivalent to that of the basic variant).

The typical behaviour of three different variants of the genetic process is graphically illustrated in Fig. 6. For example, we can observe that, in the compounded approach, one starts from very high-quality initial population, however the process converges less rapidly (see Fig. 6b). Meanwhile, in the restart approach, there are obvious fluctuations in the average quality of solutions, however the restarts eventually result in better final solutions (see Fig. 6c). In both cases, the positive effect of incorporating the modifications into the standard algorithm is clearly visible.

Based on the modifications CA, ELS, RI, MO and R, we composed 20 new variants (see Figs. 7a, 7b). In these new variants, we still use the fitness-based selection and gender modification. The cohesive crossover is applied in the role of recombination of the parents' genes. Elitism is included as well. The results of the comparison of the new variants of HGA are summarized in Tables 3, 4.

The results from Tables 3, 4 confirm the powerfulness and advantage of using the compounded approach and restart technique. Incorporating the expensive local search and maintaining many offspring have a substantial impact on the efficiency of HGA, as well. In particular, the compounded approach coupled with the expensive local search and restarts (i.e., the CA-ELS-R variant) yielded the best quality solutions. The similar

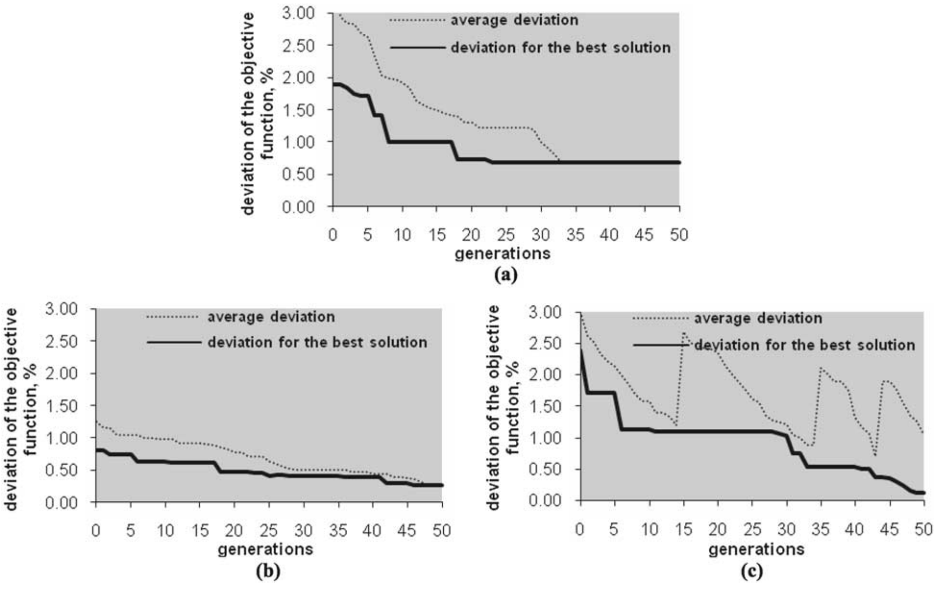

Fig. 6. Illustration of the dynamics of deviation of the objective function during the genetic process: (a) basic variant, (b) compounded approach (CA), (c) restart modification (R). *Notes*. 1. Both the average deviation for the solutions of the current population and the deviation for the best solution of the population are shown. 2. The instance tai30b was used in the experiments.

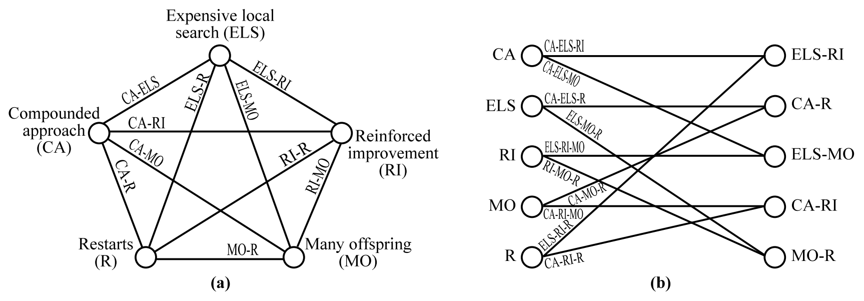

Fig. 7. Graphs of the relationships between the different variants of HGA.

variant CA-MO-R – where the expensive local search is substituted with many offspring – also demonstrates very promising performance.

The efficiency of the CA-ELS-R modification (as well as the other modifications) may be improved even more by a careful tuning of the control parameters (like the population size, number of improving iterations, selection factor, entropy threshold, etc.). The results of the tuned CA-ELS-R algorithm (denoted as CA-ELS-R^t) are presented in Table 5. Here, we also give the results of the following six algorithms: (1) robust tabu search (RTS; Taillard, 1991); (2) fast ant system (FANT; Taillard, 1998); (3) genetic-tabu search (GTS; Fleurent and Ferland, 1994); (4) genetic-local search (GLS; Lim *et al.*, 2000); (5) greedy genetic algorithm (GGA; Ahuja *et al.*, 2000); (6) fast hybrid genetic algorithm

Table 3

Results of the comparison of different variants of HGA (part II)

Instance	BKV	$\bar{\theta}$										Time (s)
		CA-ELS	CA-RI	CA-MO	CA-R	ELS-RI	ELS-MO	ELS-R	RI-MO	RI-R	MO-R	
tai20b	122455319	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	*1.5*
tai25b	344355646	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.007	0.000	*3.3*
tai30b	637117113	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	*6.8*
tai35b	283315445	0.000	0.000	0.037	0.000	0.000	0.019	0.000	0.000	0.000	0.019	*15.0*
tai40b	637250948	0.000	0.000	0.000	0.000	0.000	0.202	0.000	0.000	0.000	0.000	*33*
tai50b	458821517	0.000	0.000	0.000	0.000	0.000	0.066	0.000	0.099	0.000	0.000	*85*
tai60b	608215054	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	*200*
tai80b	818415043	0.062	0.185	0.140	0.007	0.133	0.272	0.000	0.353	0.057	0.000	*320*
tai100b	1185996137	0.032	0.077	0.112	0.010	0.030	0.044	0.040	0.040	0.067	0.064	*650*
tai150b	498896643	0.180	0.201	0.281	0.087	0.161	0.176	0.129	0.150	0.250	0.201	*3000*
	Average:	0.027	0.046	0.057	0.010	0.032	0.078	0.017	0.064	0.038	0.028	

Table 4

Results of the comparison of different variants of HGA (part III)

Instance	BKV	$\bar{\theta}$										Time (s)
		CA -ELS -RI	CA -ELS -MO	CA -ELS -R	CA -RI -MO	CA -RI -R	CA -MO -R	ELS -RI -MO	ELS -RI -R	ELS -MO -R	RI -MO -R	
tai20b	122455319	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	*1.5*
tai25b	344355646	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	*3.3*
tai30b	637117113	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.001	0.000	*6.8*
tai35b	283315445	0.007	0.007	0.000	0.010	0.017	0.000	0.028	0.000	0.000	0.006	*15.0*
tai40b	637250948	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	*33*
tai50b	458821517	0.003	0.000	0.000	0.000	0.000	0.000	0.000	0.033	0.000	0.000	*85*
tai60b	608215054	0.000	0.000	0.000	0.000	0.000	0.000	0.002	0.000	0.000	0.000	*200*
tai80b	818415043	0.088	0.059	0.003	0.140	0.006	0.005	0.175	0.057	0.003	0.110	*320*
tai100b	1185996137	0.032	0.031	0.000	0.057	0.018	0.007	0.063	0.020	0.012	0.016	*650*
tai150b	498896643	0.179	0.199	0.078	0.096	0.081	0.082	0.105	0.101	0.099	0.109	*3000*
	Average:	0.031	0.030	0.008	0.030	0.012	0.009	0.037	0.021	0.012	0.024	

(FHGA; Misevicius, 2006). (The intermediate results of tuning the CA-ELS-R algorithm are omitted.) From Table 5, it can be seen that the CA-ELS-R modification outperforms all other algorithms tested in terms of quality of the solutions.

Finally, we have conducted some additional experiments in an attempt to find out how quickly the tuned CA-ELS-R algorithm converges to the best known (pseudo-optimal) solutions. We have performed several hundreds of runs of CA-ELS-R^t. The experimen-

Table 5

Results of the comparison of the algorithms

Instance	BKV	$\bar{\theta}/c_{\mathrm{bks}}^{\dagger}$							Time[‡] (s)
		RTS	FANT	GTS	GLS	GGA	FHGA	CA-ELS-R$^{\mathrm{t}}$	
tai20b	122455319	**0**	0.094 /8	0.045 /9	0.107 /7	**0**	**0**	**0**	*0.1*
tai25b	344355646	0.056 /8	0.013 /9	**0**	0.014 /8	0.007 /9	**0**	**0**	*0.6*
tai30b	637117113	0.397 /3	0.042 /7	0.014 /9	0.498 /1	0.127 /5	**0**	**0**	*1.2*
tai35b	283315445	0.254 /5	0.201 /1	0.134 /4	0.279 /0	0.103 /5	**0**	**0**	*2.5*
tai40b	637250948	0.198 /6	0.012 /9	**0**	0.601 /0	0.008 /5	**0**	**0**	*4.8*
tai50b	458821517	0.251 /0	0.215 /0	0.041 /7	0.948 /0	0.071 /4	0.008 /9	**0**	*17*
tai60b	608215054	0.306 /0	0.185 /2	0.029 /6	0.802 /0	0.105 /2	0.009 /9	**0**	*25*
tai80b	818415043	0.297 /0	0.367 /0	0.404 /2	0.947 /0	0.225 /2	0.019 /8	**0**	*125*
tai100b	1185996137	0.201 /0	0.121 /0	0.119 /3	0.696 /0	0.212 /0	0.010 /9	**0**	*380*
tai150b	498896643	0.394 /0	0.538 /0	0.409 /0	0.550 /0	0.343 /0	0.050 **/3**	**0.048 /3**	*2000*
	Average:	0.235	0.179	0.120	0.544	0.120	0.010	**0.005**	

[†] c_{bks} denotes the number of times (out of 10) that the best known (pseudo-optimal) solution was found;
[‡] Average time per one run is given.

tation was designed in such a way that the runs are interrupted as soon as BKS is found; the next run is then started, and so on. The process stops when the predefined number of runs have been performed. This is repeated for each instance. The results of these experiments are shown in Table 6. These results are apparently better than those reported in Misevicius (2005) and confirm once again the excellent performance of the CA-ELS-R modification of HGA.

Table 6

Run time performance of CA-ELS-R$^{\mathrm{t}}$

Instance	BKV	# of runs	# of successful runs	Time[‡] (s)	Instance	BKV	# of runs	# of successful runs	Time[‡] (s)
tai20b	122455319	100	100	*0.05*	tai50b	458821517	30	30	*8.5*
tai25b	344355646	100	100	*0.1*	tai60b	608215054	20	20	*12*
tai30b	637117113	50	50	*0.5*	tai80b	818415043	10	10	*85*
tai35b	283315445	50	50	*1.1*	tai100b	1185996137	10	10	*240*
tai40b	637250948	50	50	*1.2*	tai150b	498896643	5	5	*9000*

[‡] The average CPU time needed to find the best known solution under condition that all consecutive runs of CA-ELS-R$^{\mathrm{t}}$ succeeded in finding the best known solution (900 MHz computer was used in the experiments).

4. Concluding Remarks

Hybrid genetic algorithms (HGAs) are among the most efficient intelligent optimization techniques. In contrast to the standard GAs that are based on principles of natural evolution in a quite straightforward manner, the hybrid GAs imitate a more complex, cultural environment, where the lifecycle transformations and adaptations are at least as much important as the transmission of the parents' genetic information.

In this paper, several variants of HGA for solving structured quadratic assignment problems were investigated. In particular, we examined the following nine variants: compounded approach (CA), incorporating the expensive local search (ELS), reinforced improvement (RI), fitness-based selection (FBS), gender modification (GM), cohesive crossover (COHX), maintaining many offspring (MO), elitism (E) strategy, and using restarts (R).

The results of the computational experiments with the structured QAP instances taken from the electronic library QAPLIB demonstrate the significant benefit of the proposed modifications. This is especially true for the compounded approach and the entropy-based restart technique. The experiments with these variants indicate that it is of great importance to have high-quality populations and make use of the proper restart mechanism for avoiding the loss of diversity and the premature convergence of the genetic algorithm. The other observation is that very compact populations are enabled which allow saving both the computational time and memory resources. These small-sized populations are fully compensated by including the computationally expensive, but effective local search algorithm and producing a suitable large number of the offspring.

It should be noted that the efficiency of our algorithms may be improved even more by the careful juxtaposing of the proposed modifications. This could be one of the future research directions. It may also be worthy to apply these modifications to other combinatorial optimization problems.

Acknowledgments

We would like to thank prof. Zvi Drezner for supplying his papers and valuables discussions that helped in designing the hybrid genetic algorithm and its variants. The authors also thank anonymous referees for the valuable comments and suggestions that contributed to improve the quality of the paper.

This work is supported by Lithuanian State Science and Studies Foundation.

Appendix

```
function EnhancedTabuSearch($(p, \tau)$);
// input: $p$ – the current solution, $\tau$ – the tabu search depth;
   parameters: $h$ – the tabu tenure, $\alpha, \beta, \gamma$ – the coefficients
// output: $p^\bullet$ – the best solution found
begin
    $h := \lfloor 0.2 \cdot n \rfloor$; $\alpha := \lfloor 0.1 \cdot n \rfloor$; $\beta := \lfloor 1.5 \cdot n \rfloor$; $\gamma := 0.05$;
    $p^\bullet := p$; $T := 0$; $i := 1$; $j := 1$; $r' := 1$; delay_interval:= $\lfloor \alpha \cdot n \rfloor$; intensification_interval:= $\lfloor \beta \cdot h \rfloor$;
    calculate differences in the objective function values $d_{kl} = \Delta_z(p, k, l)$,
      where $k = 1, \ldots, n-1, l = k+1, \ldots, n$;
    for $r := 1$ to $\tau$ do begin // main cycle of ETS
      $\Delta_{\min} := \infty$;
      for $w := 1$ to $|\Theta_2(p)|$ do begin // exploration of the neighbourhood $\Theta_2(p)$ (target analysis)
        $i :=$ IIF$(j < n, i, $IIF$(i < n-1, i+1, 1))$; $j :=$ IIF$(j < n, j+1, i+1)$;
        tabu:= IIF($t_{ij} \geqslant r$ and RANDOM$(0,1) \geqslant \gamma$, TRUE, FALSE);
        aspired:= IIF($z(p) + d_{ij} < z(p^\bullet)$ and tabu, TRUE, FALSE);
        if ($d_{ij} < \Delta_{\min}$ and NOT(tabu)) or aspired then begin $\Delta_{\min} := d_{ij}$; $u := i$; $v := j$ end
      end; // for $w$
      if $\Delta_{\min} < \infty$ then begin //replacement of the current solution by the new one
        $p := p \oplus m_{uv}$; update differences $d_{kl} = \Delta_z(p, k, l), k = 1, \ldots, n-1, l = k+1, \ldots, n$;
        if $r >$delay_interval then $t_{uv} := r + h$ // the move $m_{uv}$ becomes tabu
      end; // if
      if ($\Delta_{\min} < 0$) and ($r - r' \geqslant$ intensification_interval) then begin
        $p^\nabla := p$; $p :=$ FastSteepestDescent($p^\nabla$); $r' := r$
      end;
      if $z(p) < z(p^\bullet)$ then $p^\bullet := p$ // saving the best so far solution
    end; // for $r$
    return $p^\bullet$
end.
```

Fig. A1. Pseudo-code of the enhanced tabu search algorithm. *Notes.* 1. The function IIF ("Immediate IF") returns one value if the given condition is met, and another value if the condition is not met. 2. The function RANDOM(0,1) generates a pseudo-random real number from the interval [0,1].

```
function ChainedMutation($(p, \eta)$);
// input: $p$ – the current solution, $\eta$ – the mutation level; output: $p$ – the mutated solution
begin
    // generation of a random permutation $\xi = (\xi(1), \xi(2), \ldots, \xi(n))$
    for $i := 1$ to $n$ do $\xi[i] := i$;
    for $i := 1$ to $n-1$ do begin
      generate $j$, randomly, uniformly, $i \leqslant j \leqslant n$;
      $\xi := \xi \oplus m_{ij}$
    end; // for $i$
    // mutation of $p$ based on chained random pairwise interchanges
    for $i := 1$ to $\eta - 1$ do $p := p \oplus m_{\xi[i], \xi[i+1]}$;
    return $p$
end.
```

Fig. A2. Pseudo-code of the chained mutation procedure.

```
function FastSteepestDescent($p$);
// input: $p$ – the current solution; output: $p$ – the (possibly) improved solution
begin
  repeat
    $i := 1$; $j := 1$; $\Delta_{\min} := 0$;
    for $w := 1$ to $|\Theta_2(p)|$ do begin // local search in the neighbourhood $\Theta_2(p)$
      $i := \mathsf{IIF}(j < n, i, \mathsf{IIF}(i < n-1, i+1, 1))$; $j := \mathsf{IIF}(j < n, j+1, i+1)$;
      if $d_{ij} < \Delta_{\min}$ then begin $\Delta_{\min} := d_{ij}$; $u := i$; $v := j$ end
    end; // for $w$
    if $\Delta_{\min} < 0$ then begin
      $p := p \oplus m_{uv}$; update differences $d_{kl} = \Delta_z(p,k,l), k = 1,\dots,n-1, l = k+1,\dots,n$;
      $t_{uv} := r + h$ // the move $m_{uv}$ is included in the tabu list
    end // if
  until $\Delta_{\min} = 0$;
  return $p$
end.
```

Fig. A3. Pseudo-code of the fast steepest descent procedure.

References

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

Blum, C., and A. Roli (2003). Metaheuristics in combinatorial optimization: overview and conceptual comparison. *ACM Computing Surveys*, **35**, 268–308.

Burkard, R.E., S. Karisch and F. Rendl (1997). QAPLIB – a quadratic assignment problem library. *Journal of Global Optimization*, **10**, 391–403. See also `http://www.seas.upenn.edu/qaplib/`

Burkard, R.E., E. Çela, P.M. Pardalos and L. Pitsoulis (1998). The quadratic assignment problem. In D.Z. Du and P.M. Pardalos (Eds.), *Handbook of Combinatorial Optimization*, Vol. 3. Kluwer, Dordrecht. pp. 241–337.

Çela, E. (1998). *The Quadratic Assignment Problem*: *Theory and Algorithms*. Kluwer, Dordrecht.

Drezner, Z. (2003). A new genetic algorithm for the quadratic assignment problem. *INFORMS Journal on Computing*, **15**, 320–330.

Drezner, Z. (2005). Compounded genetic algorithms for the quadratic assignment problem. *Operations Research Letters*, **33**, 475–480.

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

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

Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading.

Koopmans, T., and M. Beckmann (1957). Assignment problems and the location of economic activities. *Econometrica*, **25**, 53–76.

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

Loiola, E.M., N.M.M. de Abreu, P.O. Boaventura-Netto, P. Hahn and T. Querido (2007). A survey for the quadratic assignment problem. *European Journal of Operational Research*, **176**, 657–690.

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

Misevicius, A. (2005). A tabu search algorithm for the quadratic assignment problem. *Computational Optimization and Applications*, **30**, 95–111.

Misevicius, A. (2006). A fast hybrid genetic algorithm for the quadratic assignment problem. In M. Keijzer *et al.* (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference* (*GECCO-2006*). ACM Press, New York. pp. 1257–1264.
Reeves, C.R., and J.E. Rowe (2001). *Genetic Algorithms*: *Principles and Perspectives*. Kluwer, Norwell.
Sahni, S., and T. Gonzalez. (1976). P-complete approximation problems. *Journal of ACM*, **23**, 555–565.
Taillard, E. (1991). Robust taboo search for the QAP. *Parallel Computing*, **17**, 443–455.
Taillard, E. (1995). Comparison of iterative searches for the quadratic assignment problem. *Location Science*, **3**, 87–105.
Taillard, E. (1998). FANT: fast ant system. *Tech. Report*, IDSIA-46-98, Lugano, Switzerland.
Tate, D.M., and A.E. Smith (1995). A genetic approach to the quadratic assignment problem. *Computers & Operations Research*, **1**, 73–83.
Vázquez, M., and L.D. Whitley. (2000). 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. pp. 135–142.
Voß, S. (2000). Heuristics for nonlinear assignment problems. In P.M. Pardalos and L. Pitsoulis (Eds.), *Nonlinear Assignment Problems*. Kluwer, Boston. pp. 175–215.

A. Misevičius was born in 1962, in Marijampolė, Lithuania. He received dipl. eng. degree from Kaunas Polytechnic Institute, Lithuania, in 1986. A. Misevičius got doctor degree in 1996, Kaunas University Technology. He was conferred 3rd award in Young Scientists' Competition, Kaunas University Technology, in 1997. A. Misevičius is currently assoc. prof. at Department of Multimedia Engineering, Kaunas University Technology. Author and co-author of over 70 res. papers and acad. texts on various topics of computer science. The main research interests include: computer-aided design, combinatorial optimization, design and applications of heuristics and meta-heuristics.

D. Rubliauskas was born in 1949, in Panevėžys, Lithuania. He received dipl. eng. degree from Kaunas Polytechnic Institute, Lithuania, in 1971. D. Rubliauskas got doctor's degree in 1984, Kaunas Polytechnic Institute. He is currently assoc. prof. at Department of Multimedia Engineering, Kaunas University Technology. Author and co-author of over 50 res. papers and acad. texts on different topics of computer science. The main research interests include: computer-aided design, computer graphics, data exchange, combinatorial optimization.

Eksperimentai su hibridiniais genetiniais algoritmais struktūrizuotiems kvadratinio paskirstymo uždaviniams

Alfonsas MISEVIČIUS, Dalius RUBLIAUSKAS

Hibridiniai genetiniai algoritmai (HGA) yra plačiai taikomi sprendžiant įvairius optimizavimo uždavinius. Šiame straipsnyje nagrinėjami hibridiniai genetiniai algoritmai gerai žinomam kombinatorinio optimizavimo uždaviniui – kvadratinio paskirstymo uždaviniui (KPU). Aprašoma patobulinto hibridinio genetinio algoritmo realizacija ir jos variantai (modifikacijos) svarbiai KPU klasei – vadinamiesiems struktūrizuotiems kvadratinio paskirstymo uždaviniams. Eksperimentinių tyrimų rezultatai, gauti, išbandžius HGA variantus šio tipo kvadratinio paskirstymo uždaviniams, liudija, jog hibridiniai genetiniai algoritmai įgalina per ypatingai mažą skaičiavimų laiką pasiekti (pseudo-)optimalius arba jiems labai artimus sprendinius. Gauti rezultatai taip pat patvirtina, kad HGA yra vieni iš pačių efektyviausių euristinių optimizavimo metodų būtent struktūrizuotiems KPU.