


ITC 2/48 Journal of Information Technology and Control Vol. 48 / No. 2 / 2019 pp. 335-356 DOI 10.5755/j01.itc.48.2.23114	<b>Experimental Analysis of Hybrid Genetic Algorithm for the Grey Pattern Quadratic Assignment Problem</b>	
	Received 2019/04/08	Accepted after revision 2019/05/09
	 <a href="http://dx.doi.org/10.5755/j01.itc.48.2.23114">http://dx.doi.org/10.5755/j01.itc.48.2.23114</a>	

# Experimental Analysis of Hybrid Genetic Algorithm for the Grey Pattern Quadratic Assignment Problem

## Evelina Stanevičienė

Kaunas University of Technology, Department of Multimedia Engineering, Studentų str. 50-408, LT-51368 Kaunas, Lithuania, tel. +370-37-300373, e-mail: evelina.staneviciene@ktu.lt

## Alfonsas Misevičius

Kaunas University of Technology, Department of Multimedia Engineering, Studentų str. 50-416a/400, LT-51368 Kaunas, Lithuania tel. +370-37-300372, e-mail: alfonsas.misevicius@ktu.lt

## Armantas Ostreika

Kaunas University of Technology, Department of Multimedia Engineering, Studentų str. 50-401, LT-51368 Kaunas, Lithuania tel. +370-37-300371, e-mail: armantas.ostreika@ktu.lt

---

Corresponding author: [alfonsas.misevicius@ktu.lt](mailto:alfonsas.misevicius@ktu.lt)

---

In this paper, we present the results of the extensive computational experiments with the hybrid genetic algorithm (HGA) for solving the grey pattern quadratic assignment problem (GP-QAP). The experiments are on the basis of the component-based methodology where the important algorithmic ingredients (features) of HGA are chosen and carefully examined. The following components were investigated: initial population, selection of parents, crossover procedures, number of offspring per generation, local improvement, replacement of population, population restart. The obtained results of the conducted experiments demonstrate how the methodical redesign (reconfiguration) of particular components improves the overall performance of the hybrid genetic algorithm.

**KEYWORDS:** computational intelligence, heuristics, hybrid genetic algorithms, combinatorial optimization, grey pattern quadratic assignment problem, component-based analysis.

## Introduction

The grey pattern quadratic assignment problem (GP-QAP) can be considered as a special case of the quadratic assignment problem (QAP) [2]. It is stated as follows [23]. Given two matrices  $A = (a_{ij})_{n \times n}$  and  $B = (b_{kl})_{n \times n}$  and the set  $\Pi_n$  of permutations of the integers from 1 to  $n$ , find a permutation  $p \in \Pi_n$  which minimizes the function  $z(p)$  as described in the following formula:

$$z(p) = \sum_{i=1}^n \sum_{j=1}^n a_{ij} b_{p(i)p(j)}, \quad (1)$$

where  $a_{ij} = 1$  for  $i, j = 1, \dots, m$  ( $1 \leq m < n$ )<sup>1</sup> and  $a_{ij} = 0$  otherwise. In this way, the objective is changed to finding the permutation elements  $p(1), \dots, p(m)$  ( $1 \leq p(i) \leq n, i = 1, \dots, m$ ) such that the simplified function  $z(p)$  (regarded as an objective function of the GP-QAP) is minimized:

$$z(p) = \sum_{i=1}^m \sum_{j=1}^m b_{p(i)p(j)}. \quad (2)$$

The values of the matrix  $B$  are seen as symmetric distances between each pair of  $n$  objects (elements). According to the definition in [23], the distances are calculated by using the following expressions:

$$b_{kl} = b_{(r-1)n_2+s, (t-1)n_2+u} = \omega_{rstu},$$

$$\omega_{rstu} = \max_{w_1, w_2 \in \{-1, 0, 1\}} \left\{ \frac{1}{(r-t+w_1n_1)^2 + (r-t+w_2n_2)^2} \right\}, \quad (3)$$

where  $k, l = 1, \dots, n, r, t = 1, \dots, n_1, s, u = 1, \dots, n_2, n_1 \times n_2 = n$ . We have a grid of dimensions  $n_1 \times n_2$ . More precisely, we have  $n = n_1 \times n_2$  squares in the grid: there are  $m$  black squares and  $n - m$  white squares. (Other colors may be considered instead of black and white.) This forms a grey (or color) pattern of density  $m/n$ . And the aim is to have a grey (color) pattern where the black points (color points) are distributed in the most uniform

possible way<sup>2,3</sup>.

For solving the GP-QAP, various optimization techniques can be applied, including the exact and heuristic algorithms. The exact algorithms are usually only suitable for small-sized problems [4, 6]. For problem instances that cannot be easily solved, heuristic algorithms are used. In principle, a huge spectrum of general-purpose heuristic methods (like single solution-based (trajectory-based) local search algorithms, population-based evolutionary algorithms, swarm intelligence algorithms, physically and biologically-inspired algorithms) [1] are potentially applicable. To be more precise, a local search-based steepest descent algorithm was examined in [4]. Tabu search methodology-based algorithms were investigated in [4, 23, 25]. Evolutionary/genetic algorithms have been shown to be very efficient [4, 23, 24]. The best results were achieved by applying the hybrid algorithms, which, in particular, combine the genetic algorithms and tabu search-based procedures [4, 13, 15, 16, 18, 20].

The hybrid genetic algorithms can be seen as ensembles of several ingredients, features, i.e., components (like initial population construction, selection of parents, crossover procedure, etc.). These components may, in turn, involve alternative options (including

2 The coordinates  $(r, s)$  of the black/color squares are calculated according to these formulas:  $r = \lfloor (p(i)-1)/n_2 \rfloor + 1$ ,  $s = ((p(i)-1) \bmod n_2) + 1$ .

3 The GP-QAP can alternatively also be represented as follows [4]. Let  $n$  be the number of points,  $m$  be the number of points to be selected out of  $n$  points, and  $d_{ij}$  be the distance between points  $i$  and  $j$ . Also, let  $N$  be the set of  $n$  points and let  $M$  be the subset (cluster) of selected points. The cardinalities of  $N$  and  $M$  are equal to  $n$  and  $m$ , respectively ( $|N| = n, |M| = m$ ). The subset  $M$  can be linked to the permutation elements  $p(1), \dots, p(m)$  such that  $M = \{p(i) : i = 1, \dots, m\}$  and in similar fashion the set  $N$  can be associated with the elements  $p(1), \dots, p(n)$  where  $N = \{p(i) : i = 1, \dots, n\}$ . The goal is to minimize the total distance between all pairs of points in the cluster of  $m$  points, that is, the following objective function

$$f(M) = \sum_{i \in M} \sum_{j \in M} d_{ij} \text{ is to be minimized.}$$

This is similar to the formulation of the maximum diversity problem (MDP) [7, 11], where one wishes to select a subset  $M$  of  $m$  elements from the set  $N$  of  $n$  elements in such a way that the sum of pairwise distances between any two elements in  $M$  is not minimized but maximized instead.

In this work, we use the formulation (2).

1 In our work, we will consider  $m \leq n/2$ .

control parameters). By extracting different particular components and options, we can study their impact and influence [10]. Also, by combining them in a proper way, we can reconfigure the existing algorithms; we can find the most promising redesigned algorithm architectures and build new powerful «synthetic» algorithm variants.

The main contribution of this article is just the demonstration of the benefits of component-based approach and experimental algorithm analysis based on the computational testing of the particular extracted components and options. The novel, original advantageous variants and configurations of combinations of algorithmic components and options are introduced and investigated.

The structure of the paper is as follows. In Section 1, a high-level description of HGA for the GP-QAP is given. Section 2 describes the main essential algorithmic components of HGA for the GP-QAP. The results of the computational experiments with the considered algorithmic components are presented in Section 3. The paper is completed with concluding remarks.

## 1. High-Level Description of Hybrid Genetic Algorithm for the GP-QAP

Our genetic algorithm is based on the hybrid genetic algorithm framework where the explorative search (i.e., population-based evolutionary search) is in combination with the exploitative search (i.e., local improvement of the produced offspring). (For the precise description of genetic algorithms, the interested reader is advised to refer to the fundamental books on the theory of GAs, e.g., [9, 22].)

Very briefly, our algorithm starts with the creation of an initial population. To ensure the high quality of the initial population, all the population members undergo local improvement (optimization) (see below). The genetic algorithm then performs iterations called generations until the pre-defined number of generations,  $N_{gen}$ , has been accomplished. At each generation, the standard genetic operations – selection, crossover, population replacement – take place (without the direct involvement of mutation). Every new solution (offspring) produced by the crossover operator is subject to local improvement. In order to

preserve the diversity of the population members, an enhanced population replacement strategy is adopted. On top of this, a population restart (invasion) process is incorporated to renovate the population if the genetic variability is lost and/or idle generations occur. (A more thorough description of this algorithm is presented in [20].)

The structure of our algorithm differs from the traditional usual schemes of genetic algorithms. The following are the main differences.

- 1 No encoding/decoding is necessary. The permutation elements can be directly associated with individuals' chromosomes represented as binary bit strings in traditional GAs. (We will use the terms «solutions», «individuals», «chromosomes» interchangeably. We will also use the term «gene», where gene in our case corresponds to the single element  $p(i)$  of the permutation  $p$ .)
- 2 The fitness of the individual is associated with the objective function value corresponding to the given solution. No scaling (recalculation) is needed, except the parent selection procedure (see Section 2.2.2).
- 3 The crossover operator is performed at every generation, i.e., the crossover probability is equal to 1.
- 4 As a local optimizer, we apply the hierarchical iterated tabu search (HITS) algorithm (see Section 2.5).
- 5 The mutation process is integrated into the local improvement algorithm (HITS algorithm) as an inherent constituent part of this algorithm.
- 6 Our algorithm is also distinguished by the fact that the enhanced population replacement strategy is adopted, so that the attention is paid not only to the quality of the individuals, but also to the genetic diversity (the mutual difference (distance) between the population members). Only individuals that are diverse enough survive for the next generation.
- 7 Finally, we utilize the population restart («invasion») mechanism to withstand the premature convergence and staled evolution. Restart process is triggered in the cases of the apparent stagnation of the genetic process.

The high-level description of the hybrid genetic algorithm is presented in Figure 1. The particular components of the hybrid genetic algorithm are discussed in more detail in the subsequent sections.

Figure 1

Component-based description of the hybrid genetic algorithm

```

procedure Hybrid_Genetic_Algorithm;
//input:  $n, m, B$ 
//output:  $p^*$  – the best found solution
//parameters:  $PS$  – population size,  $N_{gen}$  – number of generations,
//       $InitPopVar$  – initial population variant,  $SelectVar$  – selection variant,  $CrossVar$  – crossover variant,
//       $N_{offspr}$  – number of offspring per generation,  $DT$  – distance threshold,  $L_{idle\_gen}$  – idle generations limit,
//       $ReplaceVar$  – population replacement variant,  $RestartVar$  – population restart variant
//      [The local improvement algorithm (hierarchical iterated tabu search algorithm) possesses its own set of parameters
//      (see Table 2)]


---


begin
  get data, parameters;
  initialize algorithm variables;
  create (sorted) INITIAL POPULATION  $P$  of size  $PS$ ;
   $p^* := \arg \min_{p \in P} \{z(p)\}$  //memorize the best solution of the initial population
   $gen\_index := 1$ ;
  while  $gen\_index \leq N_{gen}$  do begin
    for  $i := 1$  to NUMBER OF OFFSPRING PER GENERATION do begin
      perform SELECTION procedure to choose the parents  $p', p'' \in P$  for reproduction;
      produce one (or two) offspring by applying CROSSOVER operator to  $p', p''$ ;
      apply LOCAL IMPROVEMENT to the produced offspring,
      get an (improved) solution  $p^*$  (and  $p^{**}$ );
      if  $z(p^*) < z(p^*)$  (or  $z(p^{**}) < z(p^*)$ ) then  $p^* := p^*$  (or  $p^* := p^{**}$ ); //memorize the best found solution
      add  $p^*$  (and  $p^{**}$ ) to a pool of offspring
    endfor;
    if number of idle generations exceeds the pre-defined limit  $L_{idle\_gen}$  then begin
      POPULATION RESTART; //perform population restart (re-construction) procedure
      if  $\min_{p \in P} \{z(p)\} < z(p^*)$  then  $p^* := \arg \min_{p \in P} \{z(p)\}$ 
    end //of if
    else perform POPULATION REPLACEMENT; //update (and sort) the population  $P$ 
     $gen\_index := gen\_index + 1$  //go to the next generation
  endwhile
end.

```

**Note.** The parameters  $InitPopVar$ ,  $SelectVar$ ,  $CrossVar$ ,  $ReplaceVar$ ,  $RestartVar$  control the various “scenarios” of the genetic algorithm. The parameter  $InitPopVar$  controls the way in which the initial population is created.  $SelectVar$  is to choose the particular procedure for parents selection.  $CrossVar$  enables to switch between several crossover operators.  $ReplaceVar$  and  $RestartVar$  are to manage the population replacement and restart processes, respectively.

## 2. Components of the Hybrid Genetic Algorithm for the Grey Pattern Quadratic Assignment Problem

The following are the essential components of the hybrid genetic algorithm, which are analysed in this work:

- 1 INITIAL POPULATION;
- 2 SELECTION;
- 3 CROSSOVER;

- 4 NUMBER OF OFFSPRING PER GENERATION;
- 5 LOCAL IMPROVEMENT;
- 6 POPULATION REPLACEMENT;
- 7 POPULATION RESTART.

### 2.1. Initial Population

The initial population component is the component that determines the way in which the initial (starting) population of solutions (individuals) is constructed. There are several options for this component (in oth-

er words, there are several variants of the creation of initial population) [17]. For this component, we will use a short syntactic notation in the form of **IP $\mathbf{x}$** , where **IP** is the abbreviation for INITIAL POPULATION and  $\mathbf{x}$  denotes the number of the option. Similar syntax will be used also for other components.

### 2.1.1. Randomly Generated Population

In the simplest way, the genetic algorithm starts from a pure random population. No additional actions (i.e., improvement of the members of initial population) are involved. We denote this option as **IP1**.

### 2.1.2. Improved Population

In the remaining options, the improvement of the individuals of initial population is involved. In fact, the initial population is constructed in two steps: 1) random generation, 2) improvement of the generated individuals. In the second step, all the members of the population are subject to improvement by a given local improvement algorithm (see Section 2.5).

There are the following available options for the improved initial population: *uniformly improved population* (**IP2**), *uniformly extra improved population* (**IP3**) and *non-uniformly improved population* (**IP4**).

#### A Uniformly Improved Population

Any local search methodology-based algorithm can be used for the improvement of initial population, for example, tabu search (which is just the case of the present paper). As long as the time spent for the improvement is essentially the same for each population member, the resultant population may be thought of as uniform (with respect to the fitness of the individuals).

Let  $\mathcal{J}$  be the (default) number of iterations of the local optimizer. Then, the overall number of iterations used for the construction of initial population is equal to  $PS_{C_1} \cdot \mathcal{J}_{C_2}$ , where  $PS_{C_1} = C_1 \cdot PS$ ,  $\mathcal{J}_{C_2} = C_2 \cdot \mathcal{J}$ . Here,  $PS$  denotes the (default) population size and  $C_1$ ,  $C_2$  are user-defined parameters (weights) such that  $C_1 \geq 1$ ,  $C_2 \geq 1$ . The parameter  $C_1$  regulates the size of the pre-initial population, so that, after improvement of the pre-initial population,  $(C_1 - 1) \cdot PS$  worst members of this population are truncated and only  $PS$  best members survive. This strategy slightly resembles a compounded approach proposed in [3], where several starting populations are maintained and the individuals of every population are improved.

The parameter  $C_2$  determines the actual total number of iterations used by the local optimizer during the construction of pre-initial population, so that the resulting number of improvement iterations during the initialization phase is equal to  $C_2 \cdot \mathcal{J}$ .

If  $C_1 = 1$  and  $C_2 = 1$ , we get the option **IP2**, otherwise ( $C_1 > 1$  and/or  $C_2 > 1$ ) the option **IP3** is obtained. Obviously, the larger the values of  $C_1$ ,  $C_2$ , the larger is the probability that the quality of the initial population will be better. However, note that if time-consuming local optimizer is applied, then the overall number of GA's generations should be accordingly decreased to keep the run time fixed.

#### B Non-Uniformly Improved Population

In this particular case, the number of improvement iterations at the population initialization stage is not constant, but varies according to some rule, so that the law of distribution of the fitness of individuals of the obtained population is rather non-uniform (for example, normal-like or Gaussian-like). This concept is linked to what is known as «differential improvement» approach [5], where more intensive improvement is performed on some selected solutions.

In our algorithm, the initial population construction procedure can be chosen from three variants: **IP1**, **IP2** and **IP3**. To operationalize this, a choice-text parameter named *InitPopVar*  $\in \{\text{"rand\_p"}, \text{"unif\_p"}, \text{"extr\_p"}\}$  is used to decide which of these variants will be activated. If *InitPopVar* = "rand\_p", **IP1** will be used; if *InitPopVar* = "unif\_p", **IP2** is activated, otherwise **IP3** will be operational. In the cases of the options **IP2**, **IP3**, our algorithm keeps checks on all improved solutions. That is, after improvement of the particular solution, it is checked if the distance between the improved solution  $p^{\star}$  and the population  $P$  ( $\delta(p^{\star}, P) = \min_{p \in P} \{\delta(p^{\star}, p)\}$ <sup>4</sup>) is greater than or equal to the pre-defined distance threshold,  $DT$ . If it is the case, the improved solution is included into the population (the same is true if the improved solution is better than the best population member). Otherwise, the randomly generated solution enters the population. This is to ensure the genetic variance of the initial population. (More details on the construction of initial population can be found in [20].)

<sup>4</sup> The distance  $\delta$  between two solutions  $p_1$  and  $p_2$  is calculated according to this formula:  $\delta(p_1, p_2) = m - |\{p_1(i) : i = 1, \dots, m\} \cap \{p_2(i) : i = 1, \dots, m\}|$ .

## 2.2. Selection

This component is used to decide how the parental solutions (chromosomes) for reproduction are chosen. The following are the popular selection rules: *random selection* (**S1**), *roulette wheel (fitness proportionate) selection* (**S2**) and *rank-based selection* (**S3**).

### 2.2.1. Random Selection

The elementary option is to choose the parental chromosomes in a pure blind random way (the fitness of the parents is not taken into account).

### 2.2.2. Roulette Wheel Selection

The roulette wheel selection is a fitness-based selection mechanism where the fitness of parents is proportional to the value of the objective function calculated for the related solutions.

In the case of roulette wheel selection [9], the scaled fitness values are utilized instead of the raw values of the objective function. The rule of selection is based on the roulette wheel criterion, which assigns to individual  $i$  in the population of  $PS$  individuals a selection probability  $Pr_i$  proportional to the fitness value of the individual as in this equation:  $Pr_i = \frac{F_i}{\sum_{j=1}^{PS} F_j}$ , where  $F_i$ ,  $F_j$  are scaled fitness values<sup>5</sup>.

### 2.2.3. Rank-Based Selection

Assume that all the members of the current population are sorted in the ascending order of their fitness (or the objective function values). Then, according to the rank-based rule [26], the positions ( $u$  and  $v$ ) of the parents within the sorted population  $P$  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 distinct uniform random numbers from the interval  $[1, PS^{1/\sigma}]$ , where  $PS$  is the population size, and  $\sigma$  is a real number in the interval  $[1, 2]$  ( $\sigma$  is

<sup>5</sup> The scaled fitness  $F_i$  can be calculated as follows:  $F_i = k_1 f_i + k_2$ , where  $k_1 = \frac{(1-S_f) \times f_{avg}}{(f_{min} \times S_f - f_{max} + (1-S_f) \times f_{avg})}$ ,  $k_2 = (1-k_1) \times f_{avg}$ ,  $S_f$  is the scaling factor,  $f_{avg}, f_{min}, f_{max}$  are respectively the average, minimum and maximum fitness values before scaling.  $f_i$  is a raw fitness value (fitness before scaling) of the  $i$ th individual, which is linked to the objective function value through the following relation:  $f_i = \begin{cases} 0, & z(p_i) > zerofit \\ 1 - z(p_i)/zerofit, & \text{otherwise} \end{cases}$ , here  $z(p_i)$  is the objective function value corresponding to the  $i$ th individual,  $zerofit = K \times W$ ,  $K$  is a coefficient,  $W$  is the lower bound or the best known value of the objective function for a given problem instance (for more details, see [12]).

referred to as a selection factor). It is obvious that the better the individual, the larger probability of selecting it for the crossover.

The choice-text parameter  $SelectVar \in \{\text{"rand\_sel"}, \text{"rw\_sel"}, \text{"rb\_sel"}\}$  is utilized to choose between three options: **S1** ( $SelectVar = \text{"rand\_sel"}$ ), **S2** ( $SelectVar = \text{"rw\_sel"}$ ) and **S3** ( $SelectVar = \text{"rb\_sel"}$ ).

## 2.3. Crossover

Crossover operators play a very important role in genetic algorithms. One of the main purposes of crossover is to diversify the search process by recombining the genetic information present in the parental chromosomes. It is highly desirable that the crossover process be explorative enough to be capable of discovering new regions in the search space (space of potential solutions).

There is a great variety of the crossover procedures, which are usually oriented to work on binary bit strings [21]. However, in the GP-QAP, the chromosomal locations of genes are irrelevant, so the straightforward replication of standard procedures to the GP-QAP might not be effective. Anyway, it is important to preserve shared elements of predecessors during the recombination process. Based on the shared elements, a concept of backbone solutions is defined. The solution  $p^{\otimes} \in \Pi_n$  is a backbone solution (with respect to two underlying parental solutions  $p_1, p_2$ ) if the inequalities  $\delta(p^{\otimes}, p_1) \leq \lceil m/2 \rceil$ ,  $\delta(p^{\otimes}, p_2) \leq \lceil m/2 \rceil$  simultaneously hold; here  $\delta$  denotes the distance between solutions. The backbone solution thus shares information with its both underlying solutions and is close enough to both of them (or possibly equivalent to them in the sense that  $\delta(p^{\otimes}, p_1) = 0$  and/or  $\delta(p^{\otimes}, p_2) = 0$ ).

Foreign genes (the genes that are not present in the parents) may also be useful. Such genes may be thought of as opposite genes – opposition-based genes. The solution  $\bar{p} \in \Pi_n$  is an opposition-based solution with respect to the given solution  $p$  if  $\delta(p, \bar{p}) = m$ . The opposition-based solutions might possibly be more preferable than purely randomly generated solutions [28].

So, we see that the main point is the preserving of the common elements (genes) in the case of backbone solutions, while in the case of opposition-based solutions, the focus is on introducing completely

new genes. The crossover operators that follow these conceptions (conceptions of backbone and opposition-based solutions) are known as a *backbone-based crossover* (C1) and *backbone and opposition-based crossover* (C2). Similar to them are the *multi-parent backbone-based crossover* (C3) and *multi-parent backbone and opposition based crossover* (C4).

The other promising conception is to adopt the greedy-based approach by choosing the genes from the parents. This principle is put into practice in the *greedy crossover* (C5) and *greedy opposition-based crossover* (C6).

There are also crossover procedures which take into account the problem-specific information and are tailored to the particular properties of the problem. An example is the *averaging crossover* (C7), which is based on calculating the average integer value between the two corresponding values in the parental solutions. A specific crossover operator entitled as *tabu merging process* (C8), which is proposed in [4], also belongs to this type of crossover procedures.

Finally, an extra variant is possible with no crossover at all (C9). This option may be viewed as an *asexual self-replication*.

The parameter  $CrossVar \in \{"1", "2", "3", "4", "5", "6", "7", "8", "9"\}$  serves as a switch between nine options: C1, ..., C9. All the crossover operators corresponding to these options (including the crossover from [4]) are implemented by the authors of this paper<sup>6</sup>. Some operational implementation aspects are as follows.

To create the backbone solution, it is enough to maintain one-dimensional array of gene frequency values,  $freq$ . The values of  $freq$  are calculated by this expression:  $freq(i) = \left| \left\{ i : i \in \{p'(j) : j = 1, \dots, m\}, i \in \{p''(j) : j = 1, \dots, m\} \right\} \right|$ , where  $i = 1, \dots, n$  and  $p', p''$  are the parental solutions (more than two parents may be used). The  $m$  genes with the largest frequency values are then picked up (ties are broken randomly). The chosen genes constitute the backbone solution.

The obtained solution can be partially optimized to ensure a higher quality of the offspring. A greedy adaptive procedure (GAP) is applied for this purpose. The GAP receives a partial (backbone) solution  $p^{\sim}$  (the el-

ements  $p^{\sim}(1), \dots, p^{\sim}(\gamma)$ ) as an input. The integer number  $\gamma$  ( $0 < \gamma < m$ ) is the controlling parameter for the GAP, which determines the actual size of the partial solution. The GAP chooses the element, one at a time, and adds it to the current partial solution. In particular, GAP adds at each iteration  $q$  ( $q = 1, \dots, m - \gamma$ ) the element from the set of unselected elements with the minimum possible contribution to the value of the objective function. This is continued until the solution is completed. The detailed description of GAP is provided in [20].

In the case of greedy crossover, the same procedure is used, except that  $\gamma = 0$  (which means that the input solution is empty) and that the elements are chosen from the parental solutions.

Regarding the opposition-based solution, this is operationalized by maintaining a long-term memory array  $ltm$ , where  $ltm(i)$  contains the number of times the element (gene)  $i$  ever appeared in an offspring solution. The memory is initialized with zeros once before starting the genetic algorithm. Its values are updated whenever a new offspring solution is created. To obtain the opposition-based solution, it is sufficient to pick up  $m$  items with the smallest frequency from the long-term memory (ties are broken randomly).

## 2.4. Number of Offspring per Generation

This is to control how many offspring are created before updating the current population and continuing with the next generation. Usually, in the standard versions of GAs, a single offspring per generation is produced. We call this option *single offspring* (NOG1).

The other options are also possible, where several offspring are created [19]. For example, suppose that  $\lambda$  offspring are produced (in our algorithm,  $\lambda = N_{offspr}$ ). In other words, a generation consists of producing  $\lambda$  offspring. With this, we are a bit closer to the nature. We can assign a «juvenile» status for all newly born offspring, so that the juveniles are not reproductive for some time. By the next generation, the juveniles develop into adults and become reproductive.

We have tried three options: 1)  $\lambda = 2$  (NOG2), 2)  $\lambda = PS/2$  (NOG3) and 3)  $\lambda = PS$  (NOG4). In all cases, the newly produced offspring (juveniles) are added to a pool (temporary population) consisting of  $\lambda$  offspring. The extended population of  $PS + \lambda$  members is yielded. The offspring in the pool are not altered during the current generation. At the end of the gen-

<sup>6</sup> The source texts (in C# programming language) of the crossover operators can be found at: <https://www.personalas.ktu.lt/~alfmise/>

eration,  $PS$  individuals are selected out of  $PS + \lambda$  individuals to form the new population for the next generation (also see Section 2.6).

Note that if the number of offspring is increased ( $\lambda > 1$ ), then the total number of generations should be accordingly decreased in order to stay within the same run time.

## 2.5. Local Improvement

The local improvement (local optimizer) is very likely the most important component of the hybrid genetic algorithm. In contrast to the crossover operator, this component is responsible for the intensification of the search process. It is rather exploitative and concentrates the search in rather limited areas of the search space, staying in the neighbourhood of the offspring solution. Various heuristic algorithms can be adopted for this task. In most cases, these are the single solution-based algorithms, which get the offspring solution as an initial solution and return (a possibly) improved solution for the further consideration.

In this work, we use the hierarchical iterated tabu search approach, where the basic principle is to multiply reuse the tabu search (TS) [8] algorithm. We remind that the main idea of tabu search is based on the prohibition of returning to the solutions that have been visited recently. The prohibition period, i.e., the tabu tenure is one of the controlling parameters of TS. The memory structure called a tabu list is used for the prohibited solutions. In our case, the TS process itself consists of iterations trying to interchange the elements of the current solution. In particular, TS iteratively swaps an element of the set  $\{p(i) : i = 1, \dots, m\}$  with another element of the set  $\{p(j) : j = m + 1, \dots, n\}$ . If the found best solution is not in the tabu list or the tabu aspiration criterion is satisfied, then the new obtained solution replaces the current one. A secondary memory is utilized to archive high-quality solutions. The TS procedure restarts from time to time from one of these solutions if the minimum objective function value is not improved for  $L_{idle\_iter}$  iterations. Additionally, we apply a special technique [27] to reduce the number of interchangeable elements during the TS procedure (for more details, see [20]).

The HITS algorithm is  $k$ th-level<sup>7</sup> iterated tabu search (ITS) algorithm, which consists of  $(k - 1)$ th-level ITS

algorithm, candidate acceptance rule and perturbation procedure. The  $k$ th-level ITS algorithm transforms the current solution into the optimized solution using  $(k - 1)$ th-level procedure. Perturbation is applied to the chosen optimized candidate solution, which is selected by a defined candidate acceptance rule. The perturbed solution serves as an input for the  $(k - 1)$ th-level procedure, which starts immediately after the perturbation process has been executed. The  $(k - 1)$ th-level algorithm includes the  $(k - 2)$ th-level algorithm, and so on (all the way down to the 0th-level procedure). The best found solution is the result of the HITS algorithm.

The solution perturbation procedure consists of random mutation (shuffling) and reconstruction of the mutated solution. For reconstruction, we use the greedy adaptive procedure (see Section 2.3), where  $\gamma = m - \mu_{mut}$ , here  $\mu_{mut}$  is referred to as a mutation rate. (More details can be found in [20].)

Assume that the (default) total number of (global) iterations of the hierarchical iterated tabu search algorithm is equal to  $\mathfrak{J}$ ; meanwhile,  $\tau$  is the number of (internal) iterations of the self-contained tabu search procedure (in our algorithm,  $\mathfrak{J} = Q_{hier}$ ). The perturbation procedure (including the mutation process) is then performed once every  $\tau$  iterations. We will call this scheme  $((\mathfrak{J}, \tau, 1)$ -scheme) as a *neutral variant (LI1)*. The other options (schemes) are as follows.

### 2.5.1. Diversified Quick Search

If the value of  $\tau$  is decreased and the value of  $\mathfrak{J}$  remains constant, this means that the search process is rather quick and, at the same time, diversified, scattered over several regions in the search space. We call this option *diversified quick search (LI2)*. (In particular, we use the scheme  $(\mathfrak{J}, \tau/2, 1)$  in our experiments.)

### 2.5.2. Extensive Search

On the contrary, if the value of  $\tau$  is increased (and the value of  $\mathfrak{J}$  stays unchanged), this means that the search process is quite intensive, rather extensified than diversified. We refer to this option as *extensive search (LI3)*. (Particularly, we use the scheme  $(\mathfrak{J}, 2\tau, 1)$  in our work.)

### 2.5.3. Quick Search

If the value of  $\tau$  remains unchanged and the value of  $\mathfrak{J}$  is decreased instead, this results in the search process

<sup>7</sup> We used  $k = 7$ .



which is pretty quick (like in the option **LI2**), but is not so diversified (scattered). We call this option *quick (localized) search* (**LI4**). (The scheme  $(\mathfrak{J}/2, \tau, 1)$  is, in particular, used in this work.)

#### 2.5.4. Diversified Extensive Search

Finally, if the value of  $\tau$  stays constant and the value of  $\mathfrak{J}$  is increased, this leads to the search process which is relatively extensive (like in the option **LI3**); at the same time, the search is diversified, rather than concentrated in some localized region. This option is termed as *diversified extensive search* (**LI5**). (The scheme  $(2\mathfrak{J}, \tau, 1)$  is, particularly, used in our experimentation.)

The number of generations of HGA should be accordingly adjusted in the case of options **LI2–LI5** (in order to stay within the fixed run time).

## 2.6. Population Replacement

The purpose of this component is to update the population according to some pre-defined strategy. Two basic population replacement policies are *steady state replacement* and *generational replacement* [22]. In the first case, new individuals are inserted into the current population as soon as they are produced. In the generational replacement strategy, an intermediate population (pool) is usually created before updating the current population. This means that individuals can only reproduce with individuals from the same generation before moving to the next generation. Two popular generational replacement strategies are known as the " $\mu + \lambda$ "-update and " $\mu, \lambda$ "-update.

Suppose that the size of a population is equal to  $\mu$  and the number of newly created individuals is equal to  $\lambda$  (in our algorithm,  $\mu = PS$ ,  $\lambda = N_{\text{offspr}}$ ). Then, in the case of " $\mu + \lambda$ "-update, the individuals chosen for the next generation are the best  $\mu$  members of  $P_\mu \cup P_\lambda$ , where  $P_\mu$  is the current generation's population and  $P_\lambda$  denotes the pool of newly created individuals. (For example, 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).)

In the " $\mu, \lambda$ " scheme,  $\lambda$  new individuals replace their related predecessors in the current population. Typically, the children replace their worse parents. The fitness of the children is not taken into consideration.

Based on these standard strategies, modified strategies may be introduced. In the schemes we formally denote as " $\mu + \lambda, \varepsilon$ "-update and " $\mu, \lambda, \varepsilon$ "-update, a *minimum distance criterion* is introduced. That is, after the offspring solution is improved, it is checked whether the new solution ( $p^*$ ) differs enough from the other solutions in the population. Particularly, it is checked if the minimum distance between the new solution and the remaining members of population ( $\delta(p^*, P) = \min_{p \in P} \{\delta(p^*, p)\}$ ) is not less than the distance threshold,  $\varepsilon$  (in our algorithm,  $\varepsilon = DT$  (see Section 2.1.2)). If this condition is not satisfied, the new solution is not allowed to enter the population (unless the new solution is better than the best population member). These modified strategies are to maintain the sufficient diversity of the members of population.

Based on the above strategies, the following replacement options are considered: *worst individual conditional replacement* (or **PR1** according to our rule of notation), *worst individual unconditional replacement* (**PR2**), *modified conditional replacement (worst-or-best update)* (**PR3**), *worse parent conditional replacement* (**PR4**), *worse parent unconditional replacement* (**PR5**). (Let the value of  $\lambda$  be equal to 1 without a loss of generality.)

#### 2.6.1. Worst Individual Conditional Replacement

In this particular case, the newly produced offspring solution simply takes the place of the worst member of the current population, but under the necessary condition that the new offspring is better than the worst individual in terms of the objective function value. The minimum distance criterion must be satisfied, i.e., the condition  $\min_{p \in P} \{\delta(p^\circ, p)\} \geq \varepsilon$  must hold for the offspring solution  $p^\circ$ . If this criterion is not met, the offspring is deleted (the population remains unaltered) and the algorithm continues with the next generation. (Note, however, that this criterion is disobeyed if the offspring appears better than the best population individual.)

#### 2.6.2. Worst Individual Unconditional Replacement

In this case, the newly created offspring replaces the worst individual in the current population ignoring the fitness of this individual. Still, the minimum distance criterion must be satisfied.

### 2.6.3. Modified Conditional Replacement

This is similar to the option **PR1**. Additionally, it is tested if the offspring is better than the best individual of the current population. If this is the case, then exactly the best individual (rather than the worst individual) is replaced. Again, the minimum distance criterion must be fulfilled.

### 2.6.4. Worse Parent Conditional Replacement

The current option is also very close to the option **PR1**. The only difference is that the offspring replaces its respective worse parent (but not the worst individual) if only the offspring is better than its related parent with respect to the value of the objective function.

### 2.6.5. Worse Parent Unconditional Replacement

This option is very similar to the option **PR4**, except that the successor solution automatically replaces its respective worse predecessor disregarding the fitness of the predecessor.

It should be noted that in all above options, the survival of the best member in the population is ensured. This aspect is commonly known as *elitism*.

The parameter  $ReplaceVar \in \{"wi\_r\_1", "wi\_r\_2", "mod\_r", "wp\_r\_1", "wp\_r\_2"\}$  is utilized to operationalize the replacement process. The following is the correspondence between the parameter values and the actual replacement options: "wi\_r\_1" – **PR1**, "wi\_r\_2" – **PR2**, "mod\_r" – **PR3**, "wp\_r\_1" – **PR4**, "wp\_r\_2" – **PR5**.

## 2.7. Population Restart

The population restart (invasion) component also plays an important role since it is responsible for diversification of the evolutionary search in the situations where the diversity of the individuals is lost and the search process becomes stagnated [14]. In such cases, some mechanisms should be incorporated to restart the overall process by starting from the renovated (regenerated) population. These mechanisms differ in essence in how the new population is obtained. Two main opposite strategies may be seen as «hot restart» and «cold restart». In the hot restart, only small perturbations are applied to the stagnated population, whereas large population changes take place in the case of the cold restart.

The rate (frequency) at which the restarts occur is also a very significant factor by designing the restart-based genetic algorithm.

The simplest option is *no restarts* (**PRS1**). In this case, no restarts are involved at all.

### 2.7.1. Restart from Random Population

This option is to disregard all the individuals of the current population and generate new random population from scratch. This may be seen as a *restart from new random population* (**PRS2**).

### 2.7.2. Multi-Mutation

The restart from an entirely new population may seem too aggressive. A more gentle option is *multi-mutation*, where mutation process is applied to all the members of population instead of completely destroying the current population. The advantage of mutation is that the strength of mutation can be flexibly controlled by the user. For example, the user can decide to choose between *strong multi-mutation* (**PRS3**) (50% of genes are mutated) and *mild multi-mutation* (**PRS4**) (10% of genes are mutated).

### 2.7.3. Opposition-Based Reconstruction

The next option is called *opposition-based reconstruction* (**PRS5**). In this particular case, the newly built solutions are not purely randomly generated solutions. Instead, they are opposition-based (opposite) solutions with respect to the existing solutions of the stagnated population. The rule of construction of opposite solutions is analogous to that used in the opposition-based crossover operators.

### 2.7.4. Gene Translocation

*Gene translocation* (**PRS6**) procedure is slightly similar to the multi-parent crossover procedure. The distinguishing feature is that many children are produced instead of a single child. The principle is: «many-parents-many-children». The genes «migrate» between the chromosomes of all the individuals in the population. Usually, «foreign» genes are excluded and there are no explicit mutations, which are the case of multi-mutation.

### 2.7.5. Chaotic Generation

This is similar to the option **PRS2**. The difference is that the new random population is generated by the

means of chaotic logistic mapping function (logistic map)<sup>8</sup>:  $x_{l+1} = rx_l(1-x_l)$ , where  $x_l$  is a real number between zero and one (in our case,  $x_0 = 0.49$ ),  $l = 0, 1, \dots$ ;  $r$  is a parameter (in our case,  $r = 4.0$ ). This way, the real numbers  $x_0, x_1, \dots$  are used to constitute the random sequences of genes of the new population. This option is denoted as **PRS7**.

In our algorithm, the restart process is triggered if the solutions of the population are not improved for  $L_{idle\_gen}$  generations (here  $L_{idle\_gen}$  is an idle generations limit). After restart, all renovated individuals are improved by the local optimizer using the increased number of iterations,  $Q_{rhier}$  (see Table 2).

The parameter

$RestartVar \in$

$\{ "no\_rest", "rest\_rand\_p", "strong\_mut", "mild\_mut", "oppos", "gene\_trans", "chaot\_gen" \}$

serves as a switch between seven options: **PRS1**, ..., **PRS7**. The correspondence between the parameters and options is as follows: "no\_rest" – **PRS1**, "rest\_rand\_p" – **PRS2**, "strong\_mut" – **PRS3**, "mild\_mut" – **PRS4**, "oppos" – **PRS5**,

"gene\_trans" – **PRS6**, "chaot\_gen" – **PRS7**. All the restart procedures are implemented by the authors of this paper.

Before presenting the results of the computational experiments, we declare the following collection (set) of options – **IP3, S1, C2, NOG2, LI1, PR3, PRS2** – as the «basic configuration» of our hybrid genetic algorithm. The choice of these particular options is on the basis of the preliminary experimentation described in [20] and also the pre-experimental knowledge and experience. The values of the control parameters corresponding to the basic configuration (variant) of HGA are presented in Tables 1 and 2. The basic configuration is shortly denoted by **BASIC**.

For the modified configurations of HGA, we will use short notations like **IP1, S3**, and so on. For example, the notation **IP1** means that the assemblage of options of the corresponding configuration of HGA is obtained from the basic set of options by simply removing the existing option (**IP3**) and replacing it by the new option (**IP1**) ( $IP1 \equiv IP3, S1, C2, NOG2, LI1, PR3, PRS2 \setminus IP3 \cup IP1$ ).

**Table 1**

Values of the control parameters of the basic configuration of hybrid genetic algorithm

Parameter	Value	Remarks
Population size, $PS$	20	
Number of generations, $N_{gen}$	100	
Initial population variant, $InitPopVar$	«unif_p»	
Selection variant, $SelectVar$	«rand_sel»	
Crossover variant, $CrossVar$	«2»	
Number of offspring per generation, $N_{offspr}$	1	
Distance threshold, $DT$	$\lfloor 0.15m \rfloor$	$0 \leq DT \leq m$
Idle generations limit, $L_{idle\_gen}$	$\lfloor 0.15N_{gen} \rfloor$	$0 \leq L_{idle\_gen} \leq N_{gen}$
Population replacement variant, $ReplaceVar$	«mod_r»	
Population restart variant, $RestartVar$	«strong_mut»	

<sup>8</sup> See the website: [https://en.wikipedia.org/wiki/Logistic\\_map](https://en.wikipedia.org/wiki/Logistic_map)

**Table 2**

Values of the control parameters of the local improvement algorithm (hierarchical iterated tabu search algorithm)

Parameter	Value	Remarks
Number of hierarchical iterated tabu search iterations, $Q_{hier}$	256	$Q_{hier} = Q_0 \times Q_1 \times Q_2 \times Q_3 \times Q_4 \times Q_5 \times Q_6 \times Q_7^\dagger$
Number of initial hierarchical iterated tabu search iterations, $Q_{ihier}$	4096	$Q_{ihier} = Q_0 \times Q_1 \times Q_2 \times Q_3 \times 2Q_4 \times 2Q_5 \times 2Q_6 \times 2Q_7$
Number of restart hierarchical iterated tabu search iterations, $Q_{rhier}$	1296	$Q_{rhier} = Q_0 \times Q_1 \times Q_2 \times Q_3 \times 1.5Q_4 \times 1.5Q_5 \times 1.5Q_6 \times 1.5Q_7$
Number of tabu search iterations, $\tau$	50	
Idle iterations limit, $L_{idle\_iter}$	$\lfloor 0.2\tau \rfloor$	$0 < L_{idle\_iter} < \tau$
Neighbourhood size factor for tabu search, $\rho$	0.4	$\rho > 0$
Tabu tenure, $h$	$\lfloor 0.3m \rfloor$	$h > 0$
Randomization coefficient for tabu search, $\alpha$	0.02	$0 < \alpha < 1$
Mutation rate for hierarchical iterated tabu search, $\mu_{mut}$	$\lfloor 0.15m \rfloor$	$0 < \mu_{mut} < m$

$^\dagger Q_0 = Q_1 = Q_2 = Q_3 = Q_4 = Q_5 = Q_6 = Q_7 = 2$ .  $Q_0, \dots, Q_7$  denote respectively the corresponding numbers of iterations of the 0th-level, ..., 7th-level iterated tabu search algorithm.

### 3. Computational Experiments

Our hybrid genetic algorithm was implemented by using C# programming language. The C# 5.0 compiler was used with the flag «Optimize code». The computational experiments have been carried out on a personal computer running Windows 7 Enterprise. The CPU parameters are as follows: model – Intel Core i5-3450, cores – 4, instruction set – 64 bit, base frequency – 3100 MHz.

We have tested our algorithm on the medium and large-scaled GP-QAP instances with  $n = 256$  and  $n = 1024$ , respectively. These instances are generated according to the method described in [23]<sup>9</sup>. The grids are of dimensions  $16 \times 16$  ( $n_1 = n_2 = 16$ ) and  $32 \times 32$  ( $n_1 = n_2 = 32$ ), respectively. The values of the grey density parameter  $m$  varies from 95 to 104 for  $n = 256$ . For  $n = 1024$ , the values of  $m$  are as follows: 50, 60, 70, 80, 90, 100, 110, 120, 130, 140.

As a performance criterion for our algorithm, we use the average relative percentage deviation ( $\bar{\theta}$ ) of the yielded solutions from the best known solu-

tion (BKS). It is calculated by the following formula:  $\bar{\theta} = 100(\bar{z} - BKV)/BKV [\%]$ , where  $\bar{z}$  is the average objective function value over 10 runs of the algorithm, while  $BKV$  denotes the best known value of the objective function that corresponds to the BKS. (BKVs are from [20].) At every run, the algorithm is applied to the given values of  $n$  and  $m$ , each time starting from a new random initial population. Note that the current run is interrupted if BKS is found, even without reaching the maximum number of generations,  $N_{gen}$ .

Firstly, we have experimented with the following configurations (variants) of HGA:

1) **BASIC**, 2) **IP1**, 3) **IP3**, 4) **S2**, 5) **S3**, 6) **C1**, 7) **C3**, 8) **C4**, 9) **C5**, 10) **C6**;

11) **C7**, 12) **C8**, 13) **C9**, 14) **NOG1**, 15) **NOG3**, 16) **NOG4**, 17) **LI2**, 18) **LI3**, 19) **LI4**, 20) **LI5**;

21) **PR1**, 22) **PR2**, 23) **PR4**, 24) **PR5**, 25) **PRS1**, 26) **PRS3**, 27) **PRS4**, 28) **PRS5**, 29) **PRS6**, 30) **PRS7**.

Some configurations are omitted for the sake of brevity, because we think that the results of these configurations are of insufficient quality.

The obtained results of computational experiments are presented in Tables 3–8.

<sup>9</sup> These instances can also be found at the website: <https://www.personal.ktu.lt/~alfmise/>.

**Table 3**

Results of the comparison of different configurations (variants) of HGA ( $n = 256$ )(part I)

$m$	BKV <sup>‡</sup>	$\bar{\theta}^{**}$										Time <sup>***</sup> (sec.)
		BASIC	IP1	IP3*	S2	S3	C1	C3	C4	C5	C6	
95	48081112	0.021	0.077	0.000	0.020	0.014	0.011	0.015	0.021	0.005	0.017	80
96	49182368	0.057	0.134	0.014	0.078	0.080	0.078	0.077	0.080	0.074	0.085	120
97	50344050	0.053	0.105	0.000	0.033	0.062	0.048	0.039	0.061	0.046	0.031	100
98	51486642	0.079	0.120	0.023	0.104	0.083	0.095	0.082	0.083	0.086	0.086	120
99	52660116	0.085	0.105	0.023	0.079	0.074	0.074	0.069	0.077	0.071	0.078	120
100	53838088	0.076	0.082	0.040	0.075	0.069	0.069	0.054	0.075	0.056	0.077	130
101	55014262	0.085	0.096	0.042	0.078	0.083	0.079	0.075	0.077	0.076	0.086	120
102	56202826	0.081	0.092	0.050	0.088	0.082	0.088	0.071	0.081	0.071	0.078	110
103	57417112	0.047	0.070	0.026	0.052	0.054	0.052	0.052	0.055	0.051	0.067	110
104	58625240	0.026	0.073	0.017	0.031	0.040	0.021	0.026	0.041	0.028	0.034	110
<b>Average:</b>		<b>0.061</b>	<b>0.095</b>	<b>0.024</b>	<b>0.064</b>	<b>0.064</b>	<b>0.062</b>	<b>0.056</b>	<b>0.065</b>	<b>0.056</b>	<b>0.064</b>	

‡ BKV – the best known value;

\*\*  $\bar{\theta}$  – average relative percentage deviation from the best known value;

\*\*\* average CPU time per one run;

\* enlarged number of iterations of the hierarchical iterated tabu search for the initial population improvement is used ( $C_2 = 4$ ).

Notes. 1. In the case of S2, we used  $S_f = 3, K = 5$  (see Section 2.2.2). 2. In the case of S3, we used  $\sigma = 2.0$  (see Section 2.2.3). 3. In the cases of C1, C2, we used  $\gamma = \lfloor m/2 \rfloor$  (see Section 2.3). In all cases, enlarged pre-initial population is used ( $C_1 = 2$ ).

**Table 4**

Results of the comparison of different configurations (variants) of HGA ( $n = 1024$ ) (part I)

$m$	BKV	$\bar{\theta}$										Time (sec.)
		BASIC	IP1	IP3	S2	S3	C1	C3	C4	C5	C6	
50	2730510	0.000	0.002	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	230
60	4136400	0.031	0.080	0.012	0.022	0.035	0.022	0.030	0.030	0.033	0.027	2830
70	5868614	0.067	0.177	0.041	0.079	0.065	0.081	0.067	0.053	0.050	0.079	3900
80	7919112	0.018	0.052	0.011	0.016	0.015	0.017	0.013	0.018	0.015	0.019	2710
90	10331422	0.345	0.477	0.280	0.369	0.340	0.338	0.372	0.373	0.355	0.363	3570
100	13103420	0.276	0.304	0.268	0.276	0.273	0.260	0.262	0.278	0.260	0.291	2840
110	16177106	0.239	0.284	0.223	0.244	0.242	0.225	0.227	0.238	0.227	0.254	3150
120	19631156	0.102	0.292	0.047	0.083	0.078	0.063	0.077	0.084	0.081	0.080	3020
130	23460170	0.495	0.732	0.296	0.530	0.490	0.460	0.331	0.413	0.454	0.571	2790
140	27873238	0.352	0.403	0.337	0.334	0.345	0.336	0.294	0.323	0.344	0.354	2730
<b>Average:</b>		<b>0.193</b>	<b>0.280</b>	<b>0.152</b>	<b>0.195</b>	<b>0.188</b>	<b>0.180</b>	<b>0.167</b>	<b>0.181</b>	<b>0.182</b>	<b>0.204</b>	

**Table 5**Results of the comparison of different configurations (variants) of HGA ( $n = 256$ ) (part II)

$m$	BKV	$\bar{\theta}$										Time (sec.)
		C7	C8	C9	NOG1	NOG3	NOG4	LI2	LI3	LI4	LI5	
95	48081112	0.014	0.021	0.009	0.011	0.000	0.000	0.025	0.032	0.019	0.021	140
96	49182368	0.076	0.076	0.064	0.078	0.014	0.000	0.080	0.068	0.079	0.086	180
97	50344050	0.052	0.031	0.027	0.048	0.014	0.000	0.060	0.039	0.047	0.054	140
98	51486642	0.088	0.065	0.071	0.095	0.036	0.028	0.100	0.084	0.093	0.106	190
99	52660116	0.067	0.072	0.066	0.074	0.052	0.052	0.072	0.073	0.069	0.083	190
100	53838088	0.062	0.071	0.059	0.069	0.065	0.054	0.068	0.067	0.068	0.088	160
101	55014262	0.065	0.065	0.075	0.079	0.077	0.061	0.078	0.066	0.084	0.080	180
102	56202826	0.083	0.084	0.067	0.088	0.081	0.083	0.089	0.087	0.079	0.084	160
103	57417112	0.036	0.049	0.026	0.052	0.042	0.040	0.051	0.051	0.051	0.057	120
104	58625240	0.034	0.028	0.021	0.021	0.020	0.010	0.047	0.028	0.034	0.034	130
<b>Average:</b>		<b>0.058</b>	<b>0.056</b>	<b>0.049</b>	<b>0.062</b>	<b>0.040</b>	<b>0.033</b>	<b>0.067</b>	<b>0.060</b>	<b>0.062</b>	<b>0.069</b>	

**Table 6**Results of the comparison of different configurations (variants) of HGA ( $n = 1024$ ) (part II)

$m$	BKV	$\bar{\theta}$										Time (sec.)
		C7	C8	C9	NOG1	NOG3	NOG4	LI2	LI3	LI4	LI5	
50	2730510	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	240
60	4136400	0.019	0.028	0.028	0.022	0.033	0.038	0.046	0.017	0.027	0.025	3610
70	5868614	0.045	0.065	0.037	0.081	0.091	0.091	0.120	0.037	0.065	0.074	3090
80	7919112	0.013	0.009	0.012	0.017	0.019	0.016	0.025	0.012	0.018	0.017	3430
90	10331422	0.347	0.319	0.296	0.338	0.372	0.378	0.420	0.291	0.358	0.385	3310
100	13103420	0.277	0.258	0.256	0.260	0.280	0.286	0.300	0.267	0.271	0.281	3320
110	16177106	0.238	0.237	0.209	0.225	0.267	0.257	0.302	0.237	0.238	0.252	4170
120	19631156	0.080	0.070	0.068	0.063	0.118	0.080	0.233	0.016	0.079	0.105	3190
130	23460170	0.494	0.299	0.456	0.460	0.512	0.548	0.609	0.140	0.552	0.529	4080
140	27873238	0.339	0.346	0.320	0.336	0.361	0.363	0.385	0.315	0.355	0.356	4090
<b>Average:</b>		<b>0.185</b>	<b>0.163</b>	<b>0.168</b>	<b>0.180</b>	<b>0.205</b>	<b>0.206</b>	<b>0.244</b>	<b>0.133</b>	<b>0.196</b>	<b>0.202</b>	

**Table 7**Results of the comparison of different configurations (variants) of HGA ( $n = 256$ ) (part III)

$m$	BKV	$\bar{\theta}$										Time (sec.)
		PR1	PR2	PR4	PR5	PRS1	PRS3	PRS4	PRS5	PRS6	PRS7	
95	48081112	0.025	0.019	0.006	0.032	0.005	0.016	0.000	0.005	0.012	0.028	70
96	49182368	0.086	0.067	0.072	0.067	0.023	0.078	0.028	0.075	0.059	0.077	110
97	50344050	0.042	0.055	0.041	0.048	0.023	0.058	0.000	0.030	0.049	0.053	90
98	51486642	0.092	0.083	0.094	0.097	0.049	0.082	0.000	0.060	0.080	0.082	110
99	52660116	0.084	0.074	0.072	0.084	0.062	0.078	0.031	0.081	0.083	0.081	100
100	53838088	0.068	0.081	0.065	0.079	0.065	0.070	0.027	0.072	0.067	0.079	100
101	55014262	0.078	0.085	0.079	0.090	0.078	0.076	0.044	0.066	0.082	0.082	90
102	56202826	0.078	0.090	0.087	0.088	0.092	0.094	0.051	0.073	0.084	0.094	90
103	57417112	0.049	0.067	0.049	0.052	0.055	0.046	0.023	0.041	0.046	0.056	90
104	58625240	0.028	0.034	0.026	0.041	0.028	0.021	0.015	0.028	0.028	0.028	80
<b>Average:</b>		<b>0.063</b>	<b>0.066</b>	<b>0.059</b>	<b>0.068</b>	<b>0.048</b>	<b>0.062</b>	<b>0.022</b>	<b>0.053</b>	<b>0.059</b>	<b>0.066</b>	

**Table 8**Results of the comparison of different configurations (variants) of HGA ( $n = 1024$ ) (part III)

$m$	BKV	$\bar{\theta}$										Time (sec.)
		PR1	PR2	PR4	PR5	PRS1	PRS3	PRS4	PRS5	PRS6	PRS7	
50	2730510	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	250
60	4136400	0.032	0.037	0.029	0.040	0.038	0.026	0.012	0.033	0.025	0.031	3060
70	5868614	0.060	0.067	0.053	0.078	0.063	0.060	0.008	0.076	0.078	0.090	3270
80	7919112	0.016	0.022	0.018	0.016	0.023	0.012	0.001	0.015	0.011	0.020	2890
90	10331422	0.329	0.404	0.339	0.392	0.345	0.347	0.170	0.362	0.385	0.359	2750
100	13103420	0.278	0.279	0.278	0.286	0.281	0.268	0.259	0.274	0.283	0.279	2520
110	16177106	0.244	0.268	0.252	0.269	0.255	0.243	0.217	0.244	0.245	0.247	2230
120	19631156	0.102	0.087	0.091	0.103	0.103	0.076	0.009	0.091	0.094	0.082	2490
130	23460170	0.506	0.546	0.498	0.581	0.511	0.476	0.045	0.498	0.494	0.486	2480
140	27873238	0.352	0.364	0.358	0.370	0.360	0.347	0.301	0.356	0.361	0.354	2180
<b>Average:</b>		<b>0.192</b>	<b>0.207</b>	<b>0.192</b>	<b>0.214</b>	<b>0.198</b>	<b>0.186</b>	<b>0.102</b>	<b>0.195</b>	<b>0.198</b>	<b>0.195</b>	

We found out that the following configurations (variants) are quite efficient: **IP3, S3, C1, C3, C5, C8, NOG2, NOG3, NOG4, LI3, PR3, PR4, PRS4**.

Based on these promising configurations, we have designed another 8 algorithmic configurations, which intermix the available algorithm components in previously unexamined ways<sup>10</sup>:

- 1) **S3-LI3**, 2) **S3-C1-LI3-PRS4**, 3) **S3-NOG3-LI3-PRS4**,
- 4) **S3-C1-NOG3-LI3-PRS4**, 5) **S3-C3-NOG3-LI3-PRS4**,
- 6) **S3-C5-NOG3-LI3-PRS4**, 7) **S3-C8-NOG3-LI3-PRS4**,
- 8) **S3-C8-NOG4-LI3-PRS4**.

Remind that we use the short syntax. For example, **S3-LI3**  $\equiv$   $\{IP3, S1, C2, NOG2, LI1, PR3, PRS2\} \setminus \{S1\} \setminus \{LI1\} \cup \{S3\} \cup \{LI3\}$ , and so on.

The results of these new composed configurations are summarized in Tables 9 and 10.

The results of experiments provide strong evidence that the initial population construction is one of the most important components of HGA (not counting the local improvement). The conclusion is that the quality of the initial population should be as high as possible. Improved and extra-improved populations are clearly preferable to random or low-quality initial populations despite the fact that the smaller number of generations is adopted in the case of improved initial population.

The selection component influences the behaviour of HGA less than the remaining components according to our analysis.

Regarding the crossover operators, it is of high importance that these operators respect the problem-specific information of the problem at hand. The main conclusion is as follows: the specific crossover operators are preferable to general-purpose crossover procedures, or at least they are very good alternative ways to the canonical operators. Also, it is observed that good crossover procedures not only include recombination of genes, but also greedy like or fast local search operations, which enable to obtain the partially optimized offspring already at the stage of recombination of the parents' genetic material.

There is no straightforward answer as to the number of offspring per generation. Despite the analogy of

natural behaviour, many offspring are not necessary in the GA. Maintaining a single or few offspring has its advantage in that the updating of the population occurs more rapidly, which helps the speeding up of the evolution process.

It is also shown that the local improvement process, along with the related numerical parameters, is indeed of the highest importance for HGA. The prolonged extensive improvement (more iterations of the improvement algorithm) is deemed to have greater potential as compared to the quick, concentrated improvement (less iterations of the improvement algorithm) for a fixed CPU time budget. The values of the parameters  $Q_{hier}$  and  $\tau$  should be carefully calibrated to achieve the best desirable effect (here,  $Q_{hier}$  denotes the total number of iterations of the hierarchical improvement algorithm,  $\tau$  is the number of the TS iterations). Overall, it is recommended that the value of the ratio  $\frac{Q_{hier}}{N_{gen}}$  (or  $\frac{\tau}{N_{gen}}$ ) be large enough (given that  $Q_{hier} \cdot N_{gen} = const$ ), where  $N_{gen}$  is the number of generations of HGA. (For example,  $Q_{hier} = 100$ ,  $N_{gen} = 50$  would be better than  $Q_{hier} = 50$ ,  $N_{gen} = 100$ .) We observed that the solution quality is more sensitive to the value of  $\tau$  than  $Q_{hier}$  (if other conditions remain unchanged). So, larger values of  $\tau$  are recommended, especially if the problem size  $n$  is also large.

We notice that the problem-specific, tuned local optimizer should be adopted to save the computation time.

More attention should be paid to the design and implementation of the population replacement component. It is worth to try new non-traditional replacement schemes. Not only the fitness of the individuals is important, but also the distance between population members. The variability of the individuals of the population must be ensured.

As to the population restarts (invasions), we think that this aspect might have been in part underestimated by the researchers. In many GAs, the restarts are not used at all. But we believe that including restarts into the standard GAs has big potential. Only it is questionable what kind of restart techniques are more advantageous. So far, we have learned that adopting the «hot restarts» (for example, mild mutations applied to all population members) might be quite promising. The additional experiments are required to determine the

<sup>10</sup> We present the results of these particular configurations because the results of other tried configurations are of lower quality.



**Table 9**

Results of the comparison of different configurations (variants) of HGA ( $n = 256$ ) (part IV)

$m$	BKV	$\bar{\theta}$										Time (sec.)
		S3-LI3	S3-C1-LI3-PRS4	S3-NOG3-LI3-PRS4	S3-C1-NOG3-LI3-PRS4	S3-C3-NOG3-LI3-PRS4	S3-C5-NOG3-LI3-PRS4	S3-C8-NOG3-LI3-PRS4	S3-C8-NOG3-LI3-PRS4*	S3-C8-NOG3-LI3-PRS4**	S3-C8-NOG4-LI3-PRS4**	
95	48081112	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	200
96	49182368	0.049	0.035	0.021	0.000	0.000	0.035	0.007	0.000	0.000	0.007	450
97	50344050	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	210
98	51486642	0.037	0.030	0.015	0.000	0.000	0.000	0.000	0.000	0.000	0.000	230
99	52660116	0.044	0.052	0.035	0.000	0.021	0.040	0.012	0.017	0.013	0.003	550
100	53838088	0.051	0.058	0.052	0.042	0.023	0.029	0.015	0.011	0.016	0.016	610
101	55014262	0.076	0.059	0.059	0.036	0.035	0.030	0.024	0.019	0.024	0.015	610
102	56202826	0.059	0.063	0.064	0.045	0.045	0.050	0.027	0.032	0.041	0.027	740
103	57417112	0.031	0.026	0.026	0.026	0.026	0.026	0.015	0.021	0.023	0.018	600
104	58625240	0.021	0.021	0.021	0.018	0.021	0.019	0.011	0.013	0.013	0.015	540
<b>Average:</b>		<b>0.037</b>	<b>0.034</b>	<b>0.029</b>	<b>0.017</b>	<b>0.017</b>	<b>0.023</b>	<b>0.011</b>	<b>0.011</b>	<b>0.013</b>	<b>0.010</b>	

\* enlarged initial population is used ( $C_1 = 3$ );

\*\* enlarged number of tabu search iterations is used ( $\tau = 200$ ).

**Table 10**

Results of the comparison of different configurations (variants) of HGA ( $n = 1024$ ) (part IV)

$m$	BKV	$\bar{\theta}$										Time (sec.)
		S3-LI3	S3-C1-LI3-PRS4	S3-NOG3-LI3-PRS4	S3-C1-NOG3-LI3-PRS4	S3-C3-NOG3-LI3-PRS4	S3-C5-NOG3-LI3-PRS4	S3-C8-NOG3-LI3-PRS4	S3-C8-NOG3-LI3-PRS4	S3-C8-NOG3-LI3-PRS4	S3-C8-NOG4-LI3-PRS4	
50	2730510	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	860
60	4136400	0.012	0.012	0.012	0.012	0.012	0.012	0.012	0.012	0.012	0.012	3600
70	5868614	0.043	0.035	0.031	0.032	0.032	0.026	0.018	0.019	0.021	0.021	4750
80	7919112	0.016	0.011	0.009	0.009	0.008	0.010	0.005	0.004	0.006	0.004	5440
90	10331422	0.328	0.308	0.336	0.276	0.253	0.212	0.029	0.037	0.063	0.058	4570
100	13103420	0.269	0.274	0.265	0.270	0.275	0.274	0.237	0.223	0.243	0.231	3910
110	16177106	0.230	0.224	0.217	0.217	0.233	0.223	0.171	0.190	0.201	0.177	4650
120	19631156	0.004	0.014	0.008	0.006	0.009	0.005	0.005	0.001	0.002	0.001	5030
130	23460170	0.127	0.082	0.079	0.009	0.007	0.009	0.007	0.007	0.007	0.007	3570
140	27873238	0.309	0.303	0.320	0.250	0.223	0.256	0.140	0.124	0.164	0.155	6180
<b>Average:</b>		<b>0.134</b>	<b>0.126</b>	<b>0.128</b>	<b>0.108</b>	<b>0.105</b>	<b>0.103</b>	<b>0.062</b>	<b>0.062</b>	<b>0.072</b>	<b>0.067</b>	

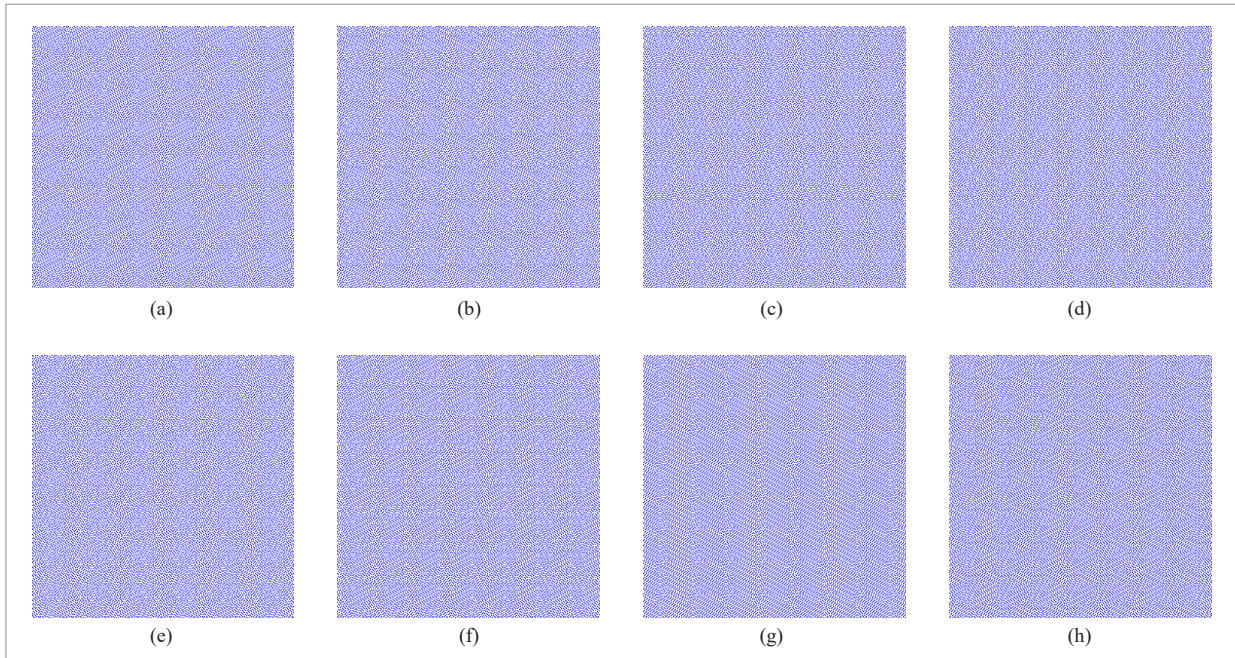
**Table 11**Best known values for the large-sized GP-QAP instances ( $n = 1024$ )

<i>m</i>	BKV	<i>m</i>	BKV	<i>m</i>	BKV	<i>m</i>	BKV	<i>m</i>	BKV	<i>m</i>	BKV	<i>m</i>	BKV	<i>m</i>	BKV
2	390	66	5132250	130	23460170	194	57086766	258	106260632	322	<b>176517006</b>	386	262819150	450	363665156
3	1954	67	5312762	131	23897592	195	57739124	259	107273354	323	<b>177709740</b>	387	264317294	451	365361310
4	3908	68	5493398	132	24335656	196	58392270	260	108286116	324	<b>178948624</b>	388	265791714	452	367056444
5	9488	69	5675784	133	24773832	197	59055298	261	109299348	325	<b>180153234</b>	389	267231448	453	368754728
6	15882	70	5868614	134	25213730	198	59710314	262	110313464	326	<b>181363520</b>	390	268698768	454	370451410
7	24290	71	6061636	135	25653062	199	60357328	263	111323160	327	<b>182597560</b>	391	270205824	455	372157104
8	32808	72	6253544	136	26091040	200	61005880	264	112349256	328	<b>183801048</b>	392	271688930	456	373863658
9	45844	73	6451748	137	26536474	201	61656140	265	113366358	329	<b>185037426</b>	393	<b>273140696</b>	457	375575430
10	60310	74	6658646	138	26980064	202	62313154	266	114381534	330	<b>186243166</b>	394	<b>274657412</b>	458	377289052
11	75878	75	6866464	139	27426740	203	62979360	267	115403394	331	<b>187471730</b>	395	276180114	459	379005274
12	91852	76	7077272	140	27873238	204	63648372	268	116415772	332	<b>188682406</b>	396	<b>277657516</b>	460	380724964
13	114040	77	7287952	141	28319430	205	64329116	269	117454596	333	<b>189927124</b>	397	279172978	461	382449898
14	136706	78	7497962	142	28761578	206	65021762	270	118462682	334	<b>191124810</b>	398	280677404	462	384174962
15	160770	79	7708934	143	29211334	207	65721964	271	119472414	335	<b>192368640</b>	399	282179032	463	385902750
16	185552	80	7919112	144	29649520	208	66422364	272	<b>120505296</b>	336	193596576	400	283712492	464	387628048
17	218392	81	8147012	145	30118164	209	67136312	273	<b>121573256</b>	337	194844496	401	<b>285200154</b>	465	389368514
18	251618	82	8363950	146	30588480	210	67852496	274	122629730	338	196104848	402	<b>286711938</b>	466	391105794
19	288006	83	8600584	147	31065948	211	68585494	275	<b>123624616</b>	339	197349714	403	<b>288225764</b>	467	392843892
20	324794	84	8839620	148	31546098	212	69315338	276	<b>124651172</b>	340	198600114	404	<b>289724016</b>	468	394587900
21	365546	85	9079818	149	32025690	213	70050648	277	<b>125732424</b>	341	199870596	405	291260654	469	396332818
22	407406	86	9322672	150	32508848	214	70789536	278	<b>126783124</b>	342	<b>201177000</b>	406	<b>292824902</b>	470	398083462
23	451448	87	9563920	151	32992712	215	71527862	279	<b>127839732</b>	343	<b>202505966</b>	407	<b>294381376</b>	471	399837320
24	496888	88	9818424	152	33479148	216	72259864	280	<b>128932972</b>	344	<b>203822602</b>	408	<b>295853078</b>	472	401592882
25	549180	89	10074140	153	33968988	217	72990332	281	<b>129989952</b>	345	<b>205164326</b>	409	<b>297441076</b>	473	403351666
26	603368	90	10331422	154	34461110	218	73726832	282	<b>131044966</b>	346	<b>206500130</b>	410	298998414	474	405112104
27	659044	91	10600710	155	34955468	219	74466810	283	<b>132108970</b>	347	<b>207849032</b>	411	<b>300563294</b>	475	406875344
28	716280	92	10871062	156	35450196	220	75201458	284	133187012	348	<b>209218954</b>	412	<b>302088470</b>	476	408637620
29	777436	93	11138470	157	35944108	221	75953890	285	<b>134274040</b>	349	<b>210604866</b>	413	303688980	477	410409038
30	837798	94	11411510	158	36437606	222	76713866	286	135364426	350	<b>211889464</b>	414	305224788	478	412182568
31	907090	95	11679880	159	36933614	223	77465610	287	136441394	351	<b>213270308</b>	415	306845268	479	413959340
32	975008	96	11944352	160	37426912	224	78218352	288	137549224	352	<b>214657946</b>	416	308393388	480	415733856
33	1050792	97	12237102	161	37947342	225	78977922	289	138637260	353	<b>216003554</b>	417	309969764	481	417519180

<i>m</i>	BKV	<i>m</i>	BKV	<i>m</i>	BKV	<i>m</i>	BKV	<i>m</i>	BKV	<i>m</i>	BKV	<i>m</i>	BKV	<i>m</i>	BKV
34	1125558	98	12523996	162	38464394	226	79744456	290	139677068	354	<b>217373862</b>	418	311420318	482	419302686
35	1203646	99	12813836	163	38982592	227	80520900	291	140801272	355	<b>218705420</b>	419	313094242	483	421092758
36	1281132	100	13103420	164	39500208	228	81287994	292	141875610	356	<b>220094764</b>	420	314652760	484	422883164
37	1368444	101	13398254	165	40025416	229	82061894	293	142916356	357	<b>221480240</b>	421	316172166	485	424678088
38	1456842	102	13691306	166	40550006	230	82837128	294	144026694	358	<b>222855502</b>	422	317825280	486	426473544
39	1547598	103	13988062	167	41078930	231	83613898	295	145175322	359	<b>224213630</b>	423	319428868	487	428272184
40	1638808	104	14288780	168	41606240	232	84406568	296	146290048	360	<b>225564514</b>	424	321022500	488	430071632
41	1736236	105	14593444	169	42140968	233	85225404	297	147454448	361	<b>226984558</b>	425	322637088	489	431876322
42	1834074	106	14899130	170	42673974	234	86030804	298	148527002	362	<b>228376202</b>	426	324266210	490	433683572
43	1935946	107	15216394	171	43219476	235	86829778	299	149672540	363	<b>229795484</b>	427	325898680	491	435492454
44	2042792	108	15537796	172	43787404	236	87618540	300	150827224	364	<b>231191072</b>	428	327457994	492	437303524
45	2147200	109	15857934	173	44361196	237	88445972	301	151952048	365	<b>232452308</b>	429	328993270	493	439118116
46	2260650	110	16177106	174	44933388	238	89239062	302	153122860	366	<b>233891082</b>	430	330616714	494	440933678
47	2373506	111	16504524	175	45511224	239	90075414	303	154282700	367	<b>235279264</b>	431	332246332	495	442752278
48	2482832	112	16837956	176	46091442	240	90875504	304	155417120	368	<b>236710772</b>	432	333874768	496	444570032
49	2607474	113	17174378	177	46680202	241	91698908	305	156547208	369	<b>238187632</b>	433	335514106	497	446397066
50	2730510	114	17508602	178	47274350	242	92523578	306	157637358	370	<b>239560518</b>	434	337154026	498	448224550
51	2857088	115	17849756	179	47871440	243	93371894	307	158826642	371	<b>241007278</b>	435	338796402	499	450053654
52	2988998	116	18191920	180	48462430	244	94187252	308	159975500	372	242479798	436	340435998	500	451883116
53	3120248	117	18535442	181	49056670	245	95044544	309	161114056	373	<b>243861204</b>	437	342076542	501	453718668
54	3257234	118	18902942	182	49654614	246	95865322	310	162278886	374	<b>245342850</b>	438	343718980	502	455554546
55	3398018	119	19272770	183	50258968	247	96720682	311	163452700	375	<b>246762156</b>	439	345362184	503	457391626
56	3535048	120	19631156	184	50876864	248	97531736	312	164632658	376	<b>248177836</b>	440	347009592	504	459230104
57	3684478	121	20001764	185	51494526	249	98361638	313	165822918	377	<b>249642256</b>	441	348669786	505	461073188
58	3829950	122	20370638	186	52115066	250	99225594	314	166988238	378	<b>251101810</b>	442	350325606	506	462916382
59	3984538	123	20746696	187	52731636	251	100062350	315	168178906	379	252535872	443	351981700	507	464761614
60	4136400	124	21117234	188	53348334	252	100898116	316	169340652	380	254011358	444	353638456	508	466607612
61	4291962	125	21484868	189	53959660	253	101733670	317	170523972	381	255486362	445	355296090	509	468457260
62	4447434	126	21852518	190	54571808	254	102566006	318	171723584	382	256914322	446	356954184	510	470307298
63	4604860	127	22218924	191	55185346	255	103399158	319	172868988	383	258421702	447	358612252	511	472158510
64	4762688	128	22581376	192	55788864	256	104232704	320	174034368	384	259861698	448	360270272	512	474010112
65	4949042	129	23021790	193	56452088	257	105247082	321	175343090	385	261377866	449	361968220		

**Figure 2**

Examples of (pseudo-)optimal grey frames ( $n = 1024$ ): (a)  $m = 300$ , (b)  $m = 301$ , (c)  $m = 302$ , (d)  $m = 303$ , (e)  $m = 304$ , (f)  $m = 305$ , (g)  $m = 306$ , (h)  $m = 307$



most effective variants of the restart procedures. New types of restart techniques are needed. The attention should also be paid to the other factors linked to restarts, for example, restarts criteria and/or rates.

Overall, the conducted experiments exhibit that the quality of the obtained results is very encouraging.

We report that, for one of the best designed component combinations – **S3-C8-NOG3-LI3-PRS4** – the best known solution was found at least 1 time out of 10 runs for every examined value of  $m$  for  $n = 256$ .

On the whole, the BKV was found at least 1 time in 10 runs 183 times. In summary, the best known solution was found 878 times. Our algorithm found the BKS in about 40% of the runs with  $m < 100$ . The newly designed algorithmic configurations achieved the BKS in nearly 85% of the cases with  $m < 100$ , while the particular variant **S3-C1-NOG3-LI3-PRS4** found the BKS in 100% of these cases.

For  $n = 1024$ , the BKV was found at least 1 time in 10 runs 66 times. Overall, the best known value was achieved 455 times.

The average relative deviation from BKV for all tested

values of  $m$  was only 0.048% for  $n = 256$ ; while the deviation for  $n = 1024$  was 0.166%.

For  $n = 256$ , we were able to find the best known solution in all 10 runs for  $m = 95, 96, 97, 98, 99$ . For  $n = 1024$ , we found the BKS at least once in 10 runs for  $m = 50, 70, 80, 120, 130$ . We found the BKS in all 10 runs for  $m = 50$ .

For the promising ensemble of options **S3-C8-NOG4-LI3-PRS4**, the average deviation never was larger than 0.027% for  $n = 256$ , while the cumulative average deviation of this ensemble for all 10 tested values of  $m$  is equal to 0.01% for  $n = 256$ . For our favourite assemblage **S3-C8-NOG3-LI3-PRS4**, the deviation never was larger than 0.223% for  $n = 1024$ , while the cumulative deviation of this assemblage is equal to 0.06% for  $n = 1024$ .

We used the configuration **S3-C8-NOG3-LI3-PRS4** in additional extensive experiments with the large-sized problem instances ( $n = 1024$ ). The enlarged size of the initial population was used. We also used the increased number of generations and increased number of iterations of the local optimizer (HITS algorithm). The number of runs of HGA was increased,

as well. During the experiments, we were successful to achieve new record solutions for 91 values of  $m$ . The results are reported in Table 11. The new BKVs are in bold face. All the remaining best known values are from [20]. It is very likely that with the greater run-time resources, the outcome may be improved with possibility of uncovering more best-known solutions.

In Figure 2, we provide some graphical representations (color frames) corresponding, in particular, to  $m = 300, 301, 302, 303, 304, 305, 306, 307$ . In the graphical illustrations, the 1024-square-grids are replicated 8 times horizontally and 8 times vertically for the visibility convenience. The squares' color is blue (instead of black). The color code is #0000FF (in accordance with the RGB color model), where "red"=0, "green"=0, "blue"= 255.

## 4. Concluding Remarks

In this paper, we present the results of the extensive computational experiments with the hybrid genetic algorithm for the grey pattern quadratic assignment problem. The experiments are based on analysing of various algorithmic features, i.e., components and their influence on the overall performance of HGA. The following essential components were examined: initial population, selection of parents, crossover procedures, number of offspring, local improvement, population replacement and population restart.

It has been ascertained that various components play different roles and they are not equally important. The results of the computational experiments demonstrate the high importance of the initial population and local improvement components. Starting from the improved initial population is indeed much more advantageous than using the random population. Overall, maintaining the superior-quality population is an essential aspect for HGA.

Note that the aggressive, intensive local improvement of the offspring is preferable to fast, inexpensive local improvement; this is especially true for the large-sized problem instances. In these cases, the results obviously demonstrate the dominance of the local improvement (i.e., "exploitative") component over "explorative" components (for example, crossover procedures).

The experimental analysis indicates that it is of great importance to operate with highly diversified populations and make use of the proper restart mechanism for avoiding the loss of genetic variability and the premature convergence of the genetic algorithm. The factor of the strength of restart of the stagnated population is very important.

We emphasize that the large number of best known solutions were achieved for the GP-QAP instances of size 1024 by applying our found favourite configuration of the hybrid genetic algorithm.

Our used methodology might be adapted for other types of heuristic algorithms. Also, it may be interesting to design an algorithmic framework for automatic configuring of the different components.

## References

1. Boussaïd, I., Lepagnot, J., Siarry, P. A Survey on Optimization Metaheuristics. *Information Sciences*, 2013, 237, 82-117. <https://doi.org/10.1016/j.ins.2013.02.041>
2. Çela, E. *The Quadratic Assignment Problem: Theory and Algorithms*. Kluwer, 1998. <https://doi.org/10.1007/978-1-4757-2787-6>
3. Drezner, Z. Compounded Genetic Algorithms for the Quadratic Assignment Problem. *Operations Research Letters*, 2005, 33(5), 475-480. <https://doi.org/10.1016/j.orl.2004.11.001>
4. Drezner, Z. Finding a Cluster of Points and the Grey Pattern Quadratic Assignment Problem. *OR Spectrum*, 2006, 28(3), 417-436. <https://doi.org/10.1007/s00291-005-0010-7>
5. Drezner, Z., Misevičius, A. Enhancing the Performance of Hybrid Genetic Algorithms by Differential Improvement. *Computers & Operations Research*, 2013, 40(4), 1038-1046. <https://doi.org/10.1016/j.cor.2012.10.014>
6. Drezner, Z., Misevičius, A., Palubeckis, G. Exact Algorithms for the Solution of the Grey Pattern Quadratic Assignment Problem. *Mathematical Methods of Operations Research*, 2015, 82(1), 85-105. <https://doi.org/10.1007/s00186-015-0505-1>
7. Erkut, E. The Discrete p-Dispersion Problem. *Europe-*

- an *Journal of Operational Research*, 1990, 46(1), 48-60. [https://doi.org/10.1016/0377-2217\(90\)90297-O](https://doi.org/10.1016/0377-2217(90)90297-O)
8. Glover, F., Laguna, M. *Tabu Search*, Kluwer, 1997. <https://doi.org/10.1007/978-1-4615-6089-0>
  9. Goldberg, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.
  10. Hoos, H. H. *Programming by Optimization*. *Communications of the ACM*, 2012, 55(2), 70-80. <https://doi.org/10.1145/2076450.2076469>
  11. Kuby, M. J. *Programming Models for Facility Dispersion: The p-Dispersion and Maximum Dispersion Problems*. *Geographical Analysis*, 1987, 19(4), 315-329. <https://doi.org/10.1111/j.1538-4632.1987.tb00133.x>
  12. Lim, M. H., Yuan, Y., Omatu, S. *Efficient Genetic Algorithms Using Simple Genes Exchange Local Search Policy for the Quadratic Assignment Problem*. *Computational Optimization and Applications*, 2000, 15(3), 249-268. <https://doi.org/10.1023/A:1008743718053>
  13. Misevičius, A. *Experiments with Hybrid Genetic Algorithm for the Grey Pattern Problem*. *Informatica*, 2006, 17(2), 237-258.
  14. Misevičius, A. *Restart-Based Genetic Algorithm for the Quadratic Assignment Problem*. In *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*, (Eds.) Bramer, M., Coenen, F., Petridis, M., Springer, 2008, 91-104. [https://doi.org/10.1007/978-1-84882-171-2\\_7](https://doi.org/10.1007/978-1-84882-171-2_7)
  15. Misevičius, A. *Generation of Grey Patterns Using an Improved Genetic-Evolutionary Algorithm: Some New Results*. *Information Technology and Control*, 2011, 40(4), 330-343. <https://doi.org/10.5755/j01.itc.40.4.983>
  16. Misevičius, A., Guogis, E., Stanevičienė, E. *Computational Algorithmic Generation of High-Quality Colour Patterns*. In *Information and Software Technologies, 19th International Conference, ICIST 2013*, (Eds.) Skersys, T., Butleris, R., Butkienė, R., *Proceedings, Communications in Computer and Information Science (CCIS)*, 403, Springer, 2013, 285-296. [https://doi.org/10.1007/978-3-642-41947-8\\_24](https://doi.org/10.1007/978-3-642-41947-8_24)
  17. Misevičius, A., Kuznecovaitė, D. *Investigating Some Strategies for Construction of Initial Populations in Genetic Algorithms*. *Computational Science and Techniques*, 2017, 5(1), 560-573. <https://doi.org/10.15181/cs.t.v5i1.1277>
  18. Misevičius, A., Rubliauskas, D. *Performance of Hybrid Genetic Algorithm for the Grey Pattern Problem*. *Information Technology and Control*, 2005, 34(1), 15-24.
  19. Misevičius, A., Rubliauskas, D. *Testing of Hybrid Genetic Algorithms for Structured Quadratic Assignment Problems*. *Informatica*, 2009, 20(2), 255-272.
  20. Misevičius, A., Stanevičienė, E. *A New Hybrid Genetic Algorithm for the Grey Pattern Quadratic Assignment Problem*. *Information Technology and Control*, 2018, 47(3), 503-520. <https://doi.org/10.5755/j01.itc.47.3.20728>
  21. Pavai, G., Geetha, T. V. *A Survey on Crossover Operators*. *ACM Computing Surveys*, 2017, 49(4), 1-43. <https://doi.org/10.1145/3009966>
  22. Sivanandam, S. N., Deepa, S. N. *Introduction to Genetic Algorithms*, Springer, 2008.
  23. Taillard, E. *Comparison of Iterative Searches for the Quadratic Assignment Problem*. *Location Science*, 1995, 3(2), 87-105. [https://doi.org/10.1016/0966-8349\(95\)00008-6](https://doi.org/10.1016/0966-8349(95)00008-6)
  24. Taillard, E., Gambardella, L. M. *Adaptive Memories for the Quadratic Assignment Problem*. *Tech. Report ID-SIA-87-97*, Lugano, Switzerland, 1997.
  25. Talbi, E.-G., Hafidi, Z., Geib, J.-M. *Parallel Tabu Search for Large Optimization Problems*. In *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, (Eds.) Voß, S., Martello, S., Osman, I. H., Roucairol, C., Kluwer, 1998, 345-358. [https://doi.org/10.1007/978-1-4615-5775-3\\_24](https://doi.org/10.1007/978-1-4615-5775-3_24)
  26. Tate, D. M., Smith, A. E. *A Genetic Approach to the Quadratic Assignment Problem*. *Computers & Operations Research*, 1995, 22(1), 73-83. [https://doi.org/10.1016/0305-0548\(93\)E0020-T](https://doi.org/10.1016/0305-0548(93)E0020-T)
  27. Wu, Q., Hao, J.-K. *A Hybrid Metaheuristic Method for the Maximum Diversity Problem*. *European Journal of Operational Research*, 2013, 231(2), 452-464. <https://doi.org/10.1016/j.ejor.2013.06.002>
  28. Zhou, Y., Hao, J.-K., Duval, B. *Opposition-Based Memetic Search for the Maximum Diversity Problem*. *IEEE Transactions on Evolutionary Computation*, 2017, 21(5), 731-745. <https://doi.org/10.1109/TEVC.2017.2674800>