ITC 3/47
Journal of Information Technology and Control
Vol. 47 / No. 3 / 2018
pp. 503-520
DOI 10.5755/j01.itc.47.3.20728

A New Hybrid Genetic Algorithm for the Grey Pattern Quadratic Assignment Problem

Received 2018/05/02 | Accepted after revision 2018/07/16

http://dx.doi.org/10.5755/j01.itc.47.3.20728

A New Hybrid Genetic Algorithm for the Grey Pattern Quadratic Assignment Problem

Alfonsas Misevičius

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

Evelina Stanevičienė

Kaunas University of Technology, Department of Multimedia Engineering, Studentų st. 50-416a/408, LT-51368 Kaunas, Lithuania, tel. + 370-37-300373, evelina.staneviciene@ktu.lt

Corresponding author: alfonsas.misevicius@ktu.lt

In this paper, we propose an improved hybrid genetic algorithm for the solution of the grey pattern quadratic assignment problem (GP-QAP). The novelty is the hybridization of the genetic algorithm with the so-called hierarchical iterated tabu search algorithm. Very fast exploration of the neighbouring solutions within the tabu search algorithm is used. In addition, a smart combination of the tabu search and adaptive perturbation is adopted, which enables a good balance between diversification and intensification during the iterative optimization process. The results from the experiments with the GP-QAP instances show that our algorithm is superior to other heuristic algorithms. Many best known solutions have been discovered for the large-scaled GP-QAP instances.

KEYWORDS: computational intelligence, heuristics, hybrid genetic algorithms, tabu search, combinatorial optimization, grey pattern quadratic assignment problem.

Introduction

The grey pattern quadratic assignment problem (GP-QAP) is a special case of the well-known combinatorial optimization problem, the quadratic assignment problem (QAP) [2]. GP-QAP can be formulated as follows [15]. Given two matrices $\boldsymbol{A} = (a_{ij})_{n\times n}$ and $\boldsymbol{B} = (b_{kl})_{n\times n}$ and the set Π_n of permutations of the integers from 1 to n, find a permutation $p \in \Pi_n$ that minimizes

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

where $a_{ij} = 1$ for $i, j = 1, ..., m$ $(1 \le m < n)$[1] and $a_{ij} = 0$ otherwise. The values of the matrix $(b_{kl})_{n\times n}$ may be seen as distances between every pair of n objects (elements), $b_{kl} = b_{lk}$, $b_{kk} = 0$, $k, l = 1, ..., n$. In the context of the GP-QAP, the values b_{kl} are defined according to the following rule [15]:

$$b_{kl} = b_{(r-1)n_2+s\,(t-1)n_2+u} = \omega_{rstu}, \omega_{rstu} = \max_{w_1,w_2\in\{-1,0,1\}}\{1/((r-t+w_1n_1)^2+(r-t+w_2n_2)^2)\} \quad (2)$$

where $r, t = 1, ..., n_1$, $s, u = 1, ..., n_2$, $n_1 \times n_2 = n$. The interpretation of the quantity ω_{rstu} is as follows (see also [15]). We may consider m electrons that have to be put on grid's squares. Then, ω_{rstu} may be thought of as a quantity proportional to repulsion force between two electrons i and j ($i, j = 1, ..., n$) located in the grid positions $k = p(i)$ and $l = p(j)$ with the coordinates (r, s) and (t, u). The electrons are to be arranged in such a way that the sum of the intensities of the repulsion forces is minimized.

So, there is a grid of dimensions n_1 by n_2. In the grid, there are $n = n_1 \times n$ squares: there are m black squares while the rest of the squares are white. This forms a grey pattern of density m/n. We seek to have a grey pattern where the black points are distributed in the most uniform possible way along the grid.

According to the above formulation, p denotes a permutation and $p(i)$, $p(j)$ denote the corresponding items (elements) of the permutation. The first m items of every feasible permutation may be considered as a solution of the GP-QAP. In this way, the objective is to find the best available, optimal solution, i.e., the permutation items $p(1), ..., p(m)$ $(1 \le p(i) \le n, i = 1, ..., m)$ such that the sum $\sum_{i=1}^{m}\sum_{j=1}^{m} b_{p(i)p(j)}$ (considered as an objective function of the GP-QAP) is as minimal as possible, that is:

$$z(p) = \sum_{i=1}^{m}\sum_{j=1}^{m} b_{p(i)p(j)} \rightarrow \text{minimum} \quad (1)$$

In formulation (3), only the matrix $\boldsymbol{B}$ and the values of n, m are necessary; meanwhile, the matrix $\boldsymbol{A}$ is not needed at all. In our work, we use this formulation, rather than the general formulation (1). Also, note that the GP-QAP-solution (the elements $p(1),...,p(m)$) can be associated with an m-(sub)set M such that $M = \{p(i) : i = 1,..., m\}$, $|M| = m$ (see [5]). Analogously, the elements $p(m+1),...,p(n)$ can be related to an $n-m$-(sub) set N, where $N = \{p(i) : i = m+1, ..., n\}$, $|N| = n - m$.

Exchanging two black (or two white) squares with each other does not change the value of the objective function, thus many permutations with the same objective value, $z(p)$, may exist. So, there exist at least $m!$ optimal solutions.

There can be other contexts of the problem defined by formula (1). For example, we can consider n points in the plane or n nodes of a network and we then may wish to find a cluster of m points, which minimizes the total distance between all pairs of points in the cluster. This cluster can be interpreted as the tightest cluster of m points [5]. This is similar to the max-cover problem [3] where one wishes to find the location of several facilities which cover the maximum number of points.

The other example of applications includes the selection of a group of m people out of n available people [5]. The distance between a pair of persons is a measure of compatibility (a measure of the ability to work together). The ideal group will have the most mutual compatibility and the least potential for tension.

For the solution of the GP-QAP, the computational intelligence approaches are well applicable, including the exact and heuristic algorithms. The exact algorithms are suited only for small-sized problems [5, 6]. For larger problem instances, heuristic algorithms are used: single-solution-based algorithms (local search, tabu search [5, 15]), population-based algorithms (genetic/evolutionary algorithms [5, 15, 16]. Among heuristic algorithms, hybrid genetic/evolutionary algorithms have been shown to be very effective [5, 11, 12, 14].

In this paper, we are attempting to further improve the performance of hybrid genetic algorithms by proposing some more new enhancements[2]. The main contributions are as follows.

1 The so-called hierarchical iterated tabu search (ITS) algorithm and its hybridization with the genetic algorithm are applied to the grey pattern quadratic assignment problem for the first time.

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

2 We assume that the reader is familiar with the most basic concepts of the genetic algorithms [9] and also the local search-based, tabu search-based heuristic algorithms [1, 8].

2 Very fast evaluation of the neighbouring solutions within the tabu search algorithm is used.

3 Smart combination of the tabu search and greedy adaptive perturbation is adopted. This enables to achieve the beneficial synergy of diversification and intensification during the iterative optimization process.

The paper is organized as follows. In Section 1, some preliminaries are given. In Section 2, we describe a novel hybrid genetic algorithm for the grey pattern quadratic assignment problem. The results of the computational experiments with the proposed algorithm are presented in Section 3. The paper is completed with concluding remarks.

1. Preliminaries

A neighbourhood function $\Theta: \Pi_n \to 2^{\Pi_n}$ assigns for each $p \in \Pi_n$ a set $\Theta(p) \subseteq \Pi_n$ – the set of neighbouring solutions of p. With the permutation-based problems, a common practice is to use the 2-exchange neighbourhood function Θ_2 which is defined in the following way: $\Theta_2(p) = \{p' : p' \in \Pi_n, \delta_H(p,p') = 2\}$, where $\delta_H(p,p')$ is the Hamming distance between the permutations p and p'. (Remind that the Hamming distance between two permutations p_1 and p_2 can be declared as $\delta_H(p_1,p_2) = |\{i : p_1(i) \neq p_2(i)\}|$.) However, the order of the elements $p(1), \dots, p(m)$ is not important in the GP-QAP, so we have to formulate the neighbourhood function in a more appropriate way. The 1-interchange neighbourhood function Θ_1 is defined such that every neighbouring solution $p' \in \Theta_1(p)$ is obtained from the current solution p by simply interchanging one element of $\{p(i) : i = 1, \dots, m\}$ with another element of $\{p(j) : j = m+1, \dots, n\}$. Clearly, this neighbourhood function maintains solution feasibility, i.e., $\forall p \in \Pi_n : p' \in \Theta_1(p) \Longrightarrow p' \in \Pi_n$. More formally:

$$\Theta_1(p) = \{p' : p' \in \Pi_n, \delta(p,p') = 1\} \tag{4}$$

where δ denotes the distance between solutions. The distance between two GP-QAP-solutions p_1 and p_2 can be defined in the following way:

$$\delta(p_1,p_2) = m - |\{p_1(i) : i = 1, \dots, m\} \cap \{p_2(i) : i = 1, \dots, m\}| \tag{5}$$

It can be seen that $0 \leq \delta \leq m$, $\delta(p,p) = 0$, $\delta(p_1,p_2) = \delta(p_2,p_1)$.

To be more precise, let $p(v)$ $(u = 1, \dots, m)$ and $p(w)$ $(v = m+1, \dots, n)$ be two items to be swapped. Then, a short notation of the form $p^{v,w}$ can be used such that $p^{v,w}(i) = \begin{cases} p(i), & i \neq v, w \\ p(v), & i = w \\ p(w), & i = v \end{cases}$. This means that $p^{v,w}$ is obtained from p by interchanging the items $p(v)$ and $p(w)$ (p is said to move to $p^{v,w}$). Of course, $\delta(p, p^{v,w}) = 1$, $p^{v,v} = p$, $(p^{v,w})^{v,w} = p$. It is of high importance to efficiently calculate the difference in the objective values when interchanging the items $p(v)$ and $p(w)$. The difference is calculated in $O(1)$ time by this formula:

$$\Delta(p^{v,w}, p) = z(p^{v,w}) - z(p) = \\ = 2\big(c\big(p(w)\big) - c\big(p(v)\big) - b_{p(v)p(w)}\big) \tag{6}$$

where $c(x)$ is a contribution of the element x (the sum of related distances):

$$c(x) = \sum_{y=1}^{m} b_{xp(y)},\ x = 1, \dots, n \tag{7}$$

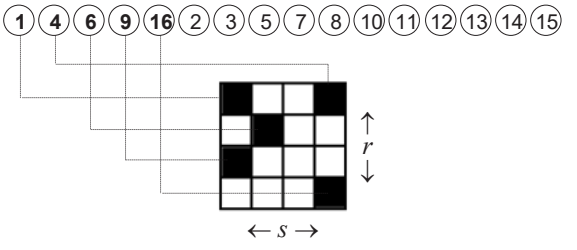
After the exchange, the contributions are updated according to the expression:

$$c(x) = \begin{cases} c(x) + b_{xp(v)}, & x = p(w) \\ c(x) - b_{xp(w)}, & x = p(v) \\ c(x) + b_{xp(v)} - b_{xp(w)}, & x \neq p(v), x \neq p(w) \end{cases}, \\ x = 1, \dots, n \tag{8}$$

The elements of the found solution determine the locations in the grid where the black squares have to be placed in. The coordinates (r,s) of the black squares are derived according to these formulas: $r = \lfloor (p(i)-1)/n_2 \rfloor + 1$, $s = \big((p(i)-1) \bmod n_2\big) + 1$ (see also Figure 1).

Let us also introduce the concept of an opposition-based solution (opposite solution), which is in connection with what is known as an opposition-based learning (OBL) [17]. The rationale of opposition-based solutions is based on an assumption that it is more advantageous to consider an opposite solution with respect to the current solution from

Figure 1
A graphical illustration of correspondence of the analytical solution to the graphical image



a search space, rather than a pure random solution generated in a blind random way. The helpfulness of using the opposition-based solutions has been confirmed by solving the maximum diversity problem (MDP) [19], which may be seen as a "sister problem" of the GP-QAP.

Definition 1. The GP-QAP-solution $\overline{p} \in \Pi_n$ is an opposition-based solution (opposite solution) with respect to the solution p if $\delta(p, \overline{p}) = m$.

Finally, we are defining a backbone solution (see also [19]).

Definition 2. The GP-QAP-solution $p^{\divideontimes} \in \Pi_n$ is a backbone solution (with respect to two underlying solutions p_1, p_2) if simultaneously $\delta(p^{\divideontimes}, p_1) \leq \lceil m/2 \rceil$ and $\delta(p^{\divideontimes}, p_2) \leq \lceil m/2 \rceil$.

Roughly speaking, the backbone solution shares information with its both underlying solutions and is close enough to both of them (or possibly "equivalent" to the underlying solution(s) in the sense that $\delta(p^{\divideontimes}, p_1) = 0$ and/or $\delta(p^{\divideontimes}, p_2) = 0$).

Within the genetic algorithm, the solution $p = (p(1), \ldots p(m))$ is associated with an individual (in fact, the individual's chromosome). Then, the single element $p(i)$ corresponds to a gene of the current chromosome.

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

Our algorithm is based on the general hybrid genetic algorithm framework where the population-based evolutionary search (i.e., explorative search) is combined with the local improvement of the offspring (i.e., exploitative search) to enhance the overall search performance.

The hybrid genetic algorithm consists of six main components: a) creation of an initial (starting) population; b) parent selection; c) a crossover operator; d) an improvement of the produced offspring; e) updating of the population; and f) restart from the new population (if necessary). The high-level description of the hybrid genetic algorithm is presented in Figure 2.

The algorithm starts with creation of the initial population of fixed size *PS* by paying attention to both the quality (fitness) of the population individuals and the mutual variability (i.e., mutual distance) between all the population members. The algorithm then performs iterations called generations until the pre-defined number of generations, N_{gen}, has been accomplished. At every generation, the standard genetic operations — selection, crossover, population replacement — take place (with the exception of mutation). The mutation procedure is integrated into the improvement algorithm — the hierarchical iterated tabu search (HITS). Our genetic algorithm also incorporates the restart mechanism in the cases of observed stagnation of the evolutionary process.

The components of the hybrid genetic algorithm are discussed in more detail in the subsequent sections.

2.1. Initial Population Construction

The initial population is generated as follows.

1 Let $flag$ = 'OFF', $P = \emptyset$, $k = 1$, $l = 1$. (l is the lexicographic index of the generated solution.)
2 If $flag$ = 'OFF', then generate a random permutation (solution) p_1; otherwise, generate an opposition-based random permutation p_l. $l = l + 1$.
3 Apply the hierarchical iterated tabu search algorithm to the generated solution and get the improved solution p_l^*.
4 If ($flag$ = 'OFF') and ($k = 1$), then: a) include the solution p_1^* into the population P; b) $flag$ = 'ON'; c) go to Step 2.
5 If ($flag$ = 'ON') and ($k = 1$) and ($z(p_l^*) < z(p)$: $p \in P$), then: a) replace the 1st member of the population by the solution p_l^* ; b) go to Step 2.
6 If $\left((z(p_l^*) \neq z(p)\text{: } \forall p \in P) \text{ and } \left(\min_{p \in P}\{\delta(p_l^*, p)\} \geq \right.\right.$

$DT\Big)\Big)$ or $\Big(z(p_l^*) < \min_{p\in P}\{z(p)\}\Big)$, then include the solution p_l^* into the population P. Otherwise, include the random solution p_l into the population P.

7 $k = k + 1$. If $k \leq PS$, then go to Step 2; otherwise, the initial population formation is finished.

Regarding the generation of opposition-based solutions, an easy way to operationalize this is to maintain a long-term frequency array (memory) f, where $f(i)$ records the number of times the item i has appeared in a solution. The array f is operated in a very simple way: all one needs is to initialize it with zeros and update its values each time a new solution is generated, i.e., $f\big(p_l(i)\big) = f\big(p_l(i)\big) + 1$, where p_l is the currently generated solution. To obtain the opposition-based random solution, it is sufficient to pick up m items with the smallest frequency (ties are broken randomly). Obviously, this ensures that $\delta(p_l, p_{l-1}) = m \;\; (l = 2, 3, ...)$, where p_l, p_{l-1} are solutions consecutively generated one after another.

Each generated solution is subject to improvement by the hierarchical ITS. After improvement, it is checked if the distance between the improved solution p_l^* and the population $P(\delta(p_l^*, P) = \min_{p\in P}\{\delta(p_l^*, p)\})$ 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 ensures both the quality and genetic variance of the initial population.

2.2. Parent Selection

At each generation, two solutions (permutations) p' and p'' are randomly selected in the population P to serve as parents for reproduction.

2.3. Crossover Operator

The goal of crossover (recombination) operator is to produce an offspring from a pair of parents. The principle of functioning of our crossover is based on two concepts: backbone solution and opposition-based (opposite) solution (see also [19]). This allows both to preserve the common elements (genes) in two selected parents and introduce completely new genes. Note that, in addition, the backbone solution is partially optimized to ensure a higher quality of the off-

Figure 2
Pseudo-code of the hybrid genetic algorithm

```
procedure Hybrid_Genetic_Algorithm;  //hybrid genetic algorithm for the grey pattern quadratic assignment problem
//input: n, m, B
//output: p* – the best found solution
//parameters: PS – population size, N_gen – number of generations, L_idle_gen – idle generations limit, DT – distance threshold
begin
  get data, parameters; initialize algorithm variables;
  create initial population P of size PS;  //see Section 2.1
  p* := argmin_{p∈P}{z(p)};  //the best so far solution is memorized
  for gen_index := 1 to N_gen do begin
    randomly select the parents p', p'' ∈ P for reproduction;
    produce the offspring p°, p°° ∈ Π_n;
    apply Hierarchical_Iterated_Tabu_Search to the offspring p°, get an (improved) solution p☆;
    if z(p☆) < z(p*) then p* := z(p☆);  //the best so far solution is updated
    update the current population P;  //see Section 2.5
    apply Hierarchical_Iterated_Tabu_Search to the offspring p°°, get an (improved) solution p☆;
    if z(p☆) < z(p*) then p* := z(p☆);  //the best so far solution is updated
    update the current population P;
    if idle generation limit L_idle_gen is exceeded then begin
       restart from the random population;
       if min_{p∈P}{z(p)} < z(p*) then p* := argmin_{p∈P}{z(p)}
    endif
  endfor
end.
```

spring. A so-called greedy adaptive procedure (GAP) is applied for this purpose. Thus, two offspring solutions are generated: the optimized offspring solution and its counterpart (the opposite offspring solution). The crossover is called the backbone and opposition-based crossover and its high-level description is given in Figure 3. Some specific details are as follows.

The values of the short-term array (gene frequencies) f_{ST}^{cross} are calculated by this expression:

$$f_{ST}^{cross}(i) = \left|\left\{j: j \in \{p'^{(k)}: k = 1, \dots, m\}, j \in \{p''^{(k)}: k = 1, \dots, m\}\right\}\right|, \quad i = 1, \dots n \tag{9}$$

where p', p'' are the corresponding parental solutions. The m genes with the most frequency are then chosen to form the backbone solution $p^{\divideontimes}$. After this, we apply the greedy adaptive procedure, which respects only $m/2$ genes with the largest frequency. So, the GAP receives a partial solution $p^{\divideontimes}$ (the elements $p^{\divideontimes}(1)$, ...,

Figure 3

Pseudo-code of the backbone and opposition-based crossover

```
procedure Backbone_and_Opposition_Based_Crossover;  //backbone and opposition-based crossover for the GP-QAP
//input: n, m,
//        p', p'' – parents
//output: p° – partially optimized offspring solution, p°°– opposition-based offspring solution
begin
  //construction of the backbone solution with respect to the parental solutions p', p''
  calculate the values of the short-term array (gene frequencies) f_ST^cross(i), i = 1, ..., n;  //see formula(9)
  pick up m genes with the largest frequency (break ties randomly),
  the chosen m genes form the backbone solution p^*;
  //partial optimization of the backbone solution
  disregard m/2 genes of the backbone solution p^*
  (the genes with the smallest frequency are disregarded (ties are resolved randomly));
  apply Greedy_Adaptive_Procedure to the partial backbone solution p^*,
  get a partially optimized complete offspring solution p°;
  //construction of the opposition-based solution
  generate an opposition-based offspring solution p°° with respect to p°, i.e., p°° = complement(p°)
end.
```

Figure 4

Pseudo-code of the greedy adaptive procedure

```
procedure Greedy_Adaptive_Procedure;  //greedy adaptive procedure for the GP-QAP
//input: n, m, B,
//        p – partial solution, where the elements p(m/2 + 1), ..., p(m) are disregarded
//output: p – feasible (complete) solution
begin
  for i := 1 to n do begin c(i) := 0; Selected(i) := FALSE endfor;
  for i := 1 to n do for j := 1 to m/2– 1 do c(i) := c(i) + B(i, p(j));  //calculation of contributions (c)
  for i := 1 to m/2 do Selected(p(i)) := TRUE;  //initialization of Selected
  i := m/2; k := p(m/2);
  for q := 1 to m/2 do begin  //cycle is repeated until the solution has been completed
    minimum_contribution := ∞;
    for j := 1 to n do
      if Selected(j) = FALSE then begin
          c(j) := c(j) + B(j, k); if c(j) < minimum_contribution then begin minimum_contribution := c(j); j_min := j endif
      endif;
    i := i + 1; p(i) := j_min;  //the element with the minimum contribution is added to the solution
    Selected(p(i)) := TRUE; k := j_min
  endfor;
  i := m + 1; for j := 1 to n do  //assigning values to the elements p(m + 1), ..., p(n)
              if Selected(j) = FALSE then begin p(i) := j; i := i + 1 endif
end.
```

$p^{*}(m/2)$) as an input. 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 ($q = 1, \ldots, m/2$), the element from the set of unselected elements $\{j: j = 1, \ldots, n\} \setminus \{p(i): i = 1, \ldots, m/2 + q - 1\}$ with the minimum contribution value (see formula (7)) across all the unselected elements, i.e., $j = \underset{j \in \{j: j=1,\ldots,n\} \setminus \{p(i): i=1,\ldots,m/2+q-1\}}{\operatorname{argmin}} \{c(p(j))\}$. This is continued until the solution has been completed (see Figure 4). Note that the objective function value z can be obtained from the c values by the equation: $z = 2\sum_{j=1}^{m} c(p(j))$. The complexity of the GAP algorithm is $O(mn)$.

The principle of greedy adaptive algorithm is not new and it is in its nature similar to "greedy randomized adaptive search procedures" (GRASP) [7]. GAP is adaptive since it selects the current element with respect to the already selected elements and the set of selected elements is updated at every iteration. The greedy component of GAP is that it always chooses the element with the minimum possible contribution. However, randomization is absent in GAP.

For the generation of opposite solution, we utilize a long term frequency array f_{LT}^{cross}. The initialization of f_{LT}^{cross} is done before running the genetic algorithm. The values of f_{LT}^{cross} are updated each time the new optimized backbone solution is constructed, i.e., $f_{LT}^{cross}(p^{\circ}(i)) = f_{LT}^{cross}(p^{\circ}(i)) + 1$, where p° is the optimized backbone solution. To get the opposition-based solution, it is sufficient to pick up m items with the smallest frequency.

2.4. Improvement of the Offspring: A Hierarchical Iterated Tabu Search Algorithm

2.4.1. Hierarchical Iterated Tabu Search Algorithm

Our proposed hierarchical iterated tabu search algorithm follows the hierarchical iterated local search paradigm [10]. The central idea is that the further enhancement of local search-based algorithms is achieved by intelligently developing the inner structure (architecture) of the algorithms and creating hierarchically structured (hierarchical) algorithms (HAs). The basic principle behind HA is the multiple utilization (reuse) of the well-known heuristic algorithm (like the local search (LS) or tabu search (TS)). In the case of TS, we firstly obtain an iterated tabu search — ITS — by combining tabu search and some perturbations. Further, the ITS algorithm itself is combined with the other ITS algorithm, which results in the "ITS-ITS" algorithm. This can be further iterated. Thus, we can define a hierarchy of many "copies" of the ITS algorithms embedded within each other. Each copy contains the three main ingredients: 1) invocation of the iterated tabu search procedure; 2) candidate acception; 3) perturbation procedure.

A. Tabu search

The tabu search procedure plays the essential role in the hierarchical ITS algorithm. In the simplest way, the TS procedure uses the 1-exchange neighbourhood Θ_1. In particular, TS starts with the current solution and iteratively swaps an element of the set $M = \{p(i): i = 1, ..., m\}$ with an element of the set $N = \{p(i): i = m + 1, ..., n\}$ such that the objective function value is minimized taking into account the tabu condition and aspiration criterion.

To reduce the computational time, we use the modified neighbourhood Θ_1^*, which is defined as follows (see also [18]):

$$\Theta_1^*(p) = \{p': p' \in \{p(i): i = 1, \ldots, m\} \setminus \{p(v)\} \cup \{p(w)\}, p(v) \in M', p(w) \in N'\} \quad (10)$$

where $v = 1, \ldots, |M'|$, $w = m + 1, ..., m + |M'|$. Sets M', N' are formed in the following way:

$$M' = \{p(i): c(p(i)) \geq threshold_1,\ i = 1, \ldots, m\},$$
$$N' = \{p(i): c(p(i)) \leq threshold_2,\ i = m + 1, \ldots, n\} \quad (11)$$

where c is the contribution array, $threshold_1 = \max\{c(p(i)): i = 1, \ldots, m\} - \rho BMax$, $threshold_2 = \min\{c(p(i)): i = m + 1, \ldots, n\} + \rho BMax$, $BMax = \max\{b_{kl}: k = 1, \ldots n, l = 1, \ldots n\}$, ρ ($\rho > 0$) is a parameter (a neighbourhood size factor).

The tabu list $TabuList$ is organized as a matrix, where the tabu list entry $TabuList(p(v), p(w))$ stores the current iteration number plus the tabu tenure h, i.e., the number of the iteration starting at which the corresponding elements $(p(v), p(w))$ may again be interchanged. The tabu tenure is fixed at the pre-defined value ($h = \lfloor 0.3m \rfloor$). The interchange of elements $p(v), p(w)$ is not allowed if the value of $TabuList(p(v), p(w))$ is equal or greater than the current iteration number. The tabu status is ignored if the aspiration criterion is met, i.e., the interchange results in a solution that is better than the best so far solution. In addition, we disregard the tabu status with a small probability α, ($\alpha = 0.02$) even if the aspiration criterion does not hold. This slightly increases the number of accepted moves and helps avoid-

ing potential stagnation of the search.

In addition to the tabu list, we also use a long-term memory like mechanism to maintain an archive of good solutions that were evaluated but not chosen [4]. The goal is to diversify the search process and explore more regions of the search space. To implement this mechanism, a list called an archive (*Archive*) is used, which is composed of so-called "second" solutions. In particular, at each iteration, the best chosen solution becomes the current solution, while the second best solution $p^{\sim}$ is included in *Archive*. The tabu search restarts from one of these solutions when the best found solution is not improved for L_{idle_iter} iterations (L_{idle_iter} is called an idle iterations limit). In our algorithm, $L_{idle\_iter} = \lfloor 0.2\tau \rfloor$, where τ is the number of tabu search iterations. Note that it is required to save the whole information, i.e., the current solution p, the contributions c, and the indices of the elements which generate $p^{\sim}$. *Archive* is emptied after finishing the TS procedure. The pseudo-code of the tabu search procedure is shown in Figure 5.

Figure 5

Pseudo-code of the tabu search algorithm

```
procedure Tabu_Search;  //tabu search algorithm for the GP-QAP
//input: n, m, B,
//       p – current solution
//output: p• – the best found solution (along with the corresponding contributions)
//parameters: τ – number of tabu search iterations, h – tabu tenure, α – randomization coefficient,
//            ρ – parameter used to regulate the neighbourhood size,
//            L_idle_iter – idle iterations limit
begin
  clear tabu list TabuList;
  p• := p; k := 1; k' := 1; archive_counter := 0; improved := FALSE;
  while (k ≤ τ) or (improved = TRUE) then begin //main cycle
    determine sets M', N'; m' := |M'|; n' := m + |N'| (rearrange p accordingly);
    Δ'min := ∞; Δ''min := ∞; v' := 1; w' := m + 1;
    for i := 1 to m' do
      for j := m + 1 to n' do begin //m'(n' − m) neighbours of p are scanned
        Δ := 2(c(p(j)) − c(p(i)) − B(p(i), p(j)));
        forbidden := iif((TabuList(p(i), p(j)) ≥ k) and (random() ≥ α)), TRUE, FALSE);
        aspired := iif(z(p) + Δ < z(p•), TRUE, FALSE);
        if ((Δ < Δ'min) and (forbidden = FALSE)) or (aspired = TRUE) then begin
          if Δ < Δ'min then begin Δ''min := Δ'min; v'' := v'; w'' := w'; Δ'min := Δ; v' := i; w' := j endif
          else if Δ < Δ''min then begin Δ''min := Δ; v'' := i; w'' := j endif
        endif
      endfor;
    if Δ''min < ∞ then begin //archiving second solution, c, v'', w''
      archive_counter := archive_counter + 1; Archive(archive_counter) ← p, c, v'', w''
    endif;
    if Δ'min < ∞ then begin //replacement of the current solution and recalculation of c
      p := p^(v',w'); for i := 1 to n do c(i) := c(i) + B(i, p(v')) − B(i, p(w'));
      if z(p) < z(p•) then begin p• := p; k' := k endif; //the best so far solution is memorized
      TabuList(p(v'), p(w')) := k + h; TabuList(p(w'), p(v')) := k + h //the elements p(v'), p(w') become tabu
    endif;
    improved := iif(Δ'min < 0, TRUE, FALSE);
    if (improved = FALSE) and (k − k' > L_idle_iter) and (k < τ − L_idle_iter) then begin //retrieving solution from the archive
      random_access_index := random(archive_counter * 0.8, archive_counter);
      p, c, v'', w'' ← Archive(random_access_index);
      p := p^(v'',w''); for i := 1 to n do c(i) := c(i) + B(i, p(v'')) − B(i, p(w''));
      clear tabu list TabuList;
      TabuList(p(v''), p(w'')) := k + h; TabuList(p(w''), p(v'')) := k + h; //the elements p(v''), p(w'') become tabu
      k' := k
    endif;
    k := k + 1
  endwhile
end.
```

Notes. 1. The immediate if function **iif**(x, y_1, y_2) returns y_1 if x = TRUE, otherwise it returns y_2. 2. The function **random**() returns a pseudo-random number uniformly distributed in [0, 1]. 3. The function **random**(x_1, x_2) returns a pseudo-random number in [x_1, x_2].

B. Iterated tabu search

In the iterated tabu search algorithm, the self-contained tabu search described above is combined with some sort of perturbations (see Section 2.4.3). The TS procedure transforms the current solution into the optimized solution. Perturbation is applied to chosen optimized candidate solution that is selected by a defined candidate acception rule (see Section 2.4.2). The perturbed solution serves as an input for the self-contained TS procedure, which starts immediately after the perturbation procedure has been executed. TS again returns an improved solution. This solution (or possibly some other previously optimized solution), in turn, is perturbed, and so on. The best found solution is regarded as the resulting solution of ITS. The overall process continues until a pre-defined number of iterations have been performed (see Figure 6).

C. Hierarchical iterated tabu search

The 1-level hierarchical iterated tabu search (1-HITS) algorithm can be obtained from the ITS algorithm. The structure of the algorithm remains practically unchanged, except that the ITS algorithm (instead of the TS algorithm) is used for the solution improvement (see Figure 7).

It is possible to further extend the 1-HITS algorithm in a very gentle way. New extension is entitled as 2-HITS. The pseudo-code of 2-HITS is almost identical to

Figure 6
Pseudo-code of the iterated tabu search algorithm

```
procedure Iterated_Tabu_Search;  //iterated tabu search algorithm for the GP-QAP
//input: p – current solution
//output: p^∇ – the best found solution (along with the corresponding contributions)
//parameter: Q – number of iterations
begin
  p^∇ := p;
  for q := 1 to Q do begin
    apply Tabu_Search to p and get p•;
    if z(p•) < z(p^∇) p^∇ := p•;  //the best found solution is memorized
    if q < Q then begin
       p := Candidate_Acceptance(p•, p^∇); apply Perturbation to p
    endif
  endfor
end.
```

Figure 7
Pseudo-code of the 1-level hierarchical iterated tabu search algorithm

```
procedure 1-Hierarchical_Iterated_Tabu_Search;  //1-level hierarchical iterated tabu search algorithm for the GP-QAP
//input: p – current solution
//output: p^(1) – the best found solution (along with the corresponding contributions)
//parameters: Q1 – number of iterations
begin
  p^(1) := p;
  for q1 := 1 to Q1 do begin
    apply Iterated_Tabu_Search to p and get p^∇;
    if z(p^∇) < z(p^(1)) p^(1) := p^∇;  //the best found solution is memorized
    if q1 < Q1 then begin
       p := Candidate_Acceptance(p^∇, p^(1)); apply Perturbation to p
    endif
  endfor
end.
```

Figure 8
Template of the seven-level hierarchical iterated tabu search algorithm

```
procedure Hierarchical_Iterated_Tabu_Search;  //seven-level hierarchical iterated tabu search for the GP-QAP
//input: n, m, B,
//       p – current solution
//output: p* – the best found solution
//parameters: Q, Q1, . . . , Q7 – numbers of iterations
begin
  for i := 1 to n do begin c(i) := 0;
  for i := 1 to n do for j := 1 to m do c(i) := c(i) + B(i, p(j));  //initialization of contributions (c)
  p(7) := p;
  for q7 := 1 to Q7 do begin
    . . .
    p(1) := p;
    for q1 := 1 to Q1 do begin
      p∇ := p;
      for q := 1 to Q do begin
        apply Tabu_Search to p and get p•;
        if z(p•) < z(p∇) then p∇ := p•;  //new better solution is memorized
        if q < Q then begin
          p := Candidate_Acceptance(p•, p∇); apply Perturbation to p
        endif
      endfor;                                          Iterated_Tabu_Search
      if z(p∇) < z(p(1)) then p(1) := p∇;  //new better solution is memorized
      if q1 < Q1 then begin
        p := Candidate_Acceptance(p∇, p(1)); apply Perturbation to p
      endif
    endfor;                                  1-Hierarchical_Iterated_Tabu_Search
    . . .
    if z(p(6)) < z(p(7)) then p(7) := p(6);  //the best solution is memorized
    if q7 < Q7 then begin
      p := Candidate_Acceptance(p(6), p(7)); apply Perturbation to p
    endif
  endfor;
  p* := p(7)
end.                                         7-Hierarchical_Iterated_Tabu_Search
```

the one of 1-HITS, except that the invocation of the ITS procedure is substituted by the invocation of the 1-HITS procedure. Continuing in the above manner, one can create a cascade of self-similar algorithms: 3-HITS, 4-HITS, and so on. It is not difficult, only one should be careful and patient. Our most latest version of HITS is, in particular, 7-HITS. The description of 7-HITS is presented in Figure 8. It is a high level template of the algorithm, rather than a detailed pseudo-code. Some parts are omitted for the sake of brevity.

2.4.2. Candidate Acception

The function **Candidate_Acceptance** can be implemented in many different ways. We utilize the so-called "where-you-are" rule, which means that we always choose the first candidate from the function parameters' list. For example, in the case of **Candidate_Acceptance**($p^{\bullet}, p^{\Delta}$), $p^{\bullet}$ is accepted.

2.4.3. Perturbation

The perturbation procedure is very simple in its structure and it consists of two parts: a) random mutation (shuffling) and b) re-construction of the mutated solution by fast greedy adaptive procedure (see Figure 9).

Firstly, the accepted candidate solution undergoes a random mutation process; in particular, the solution is

"disintegrated" by disregarding (removing) μ elements from the current solution (μ is a parameter called mutation rate). The μ elements are chosen in a random way (see Figure 10). The value of μ is relatively small in our algorithm (μ=⌊0.15m⌋), so only a minor fraction of elements is involved in the mutation procedure.

Secondly, the mutated partial solution is subject to re-construction (partial optimization) by the fast greedy adaptive procedure (FGAP), which is identical to that used in the crossover operator, except that the more effective calculation of the contributions is applied (see Figure 11). The calculation takes $O(\mu n)$ time. The overall complexity of the FGAP algorithm is also $O(\mu n)$. This results in a very fast execution of both FGAP and HITS as long as the value of μ is not large.

2.5. Population Management

After the offspring is improved by HITS, it is tested if the new solution ($p^{\star}$) differs from the other solutions in population. If it is the case, it is checked if the new solution is better than the best solution in the population or the distance between the new solution and population ($\delta(p^{\star}, P) = \min_{p \in P}\{\delta(p^{\star}, p)\}$) is greater than or equal to the distance threshold DT. If this is true, then the new solution replaces the worst solution in the current population ($P = P \cup \{p^{\star}\} \setminus \{p_{worst}\}$, where $p_{worst} = \text{argmax}_{p \in P}\{z(p)\}$). (Otherwise, the population remains unaltered and the algorithm continues with the next generation.) This rule is to maintain both the high-quality and sufficient diversity of the members of population.

Figure 9
Pseudo-code of the perturbation procedure

```
procedure Perturbation;  //perturbation procedure for the GP-QAP
//input: p – current solution
//output: p – the resulting (perturbed) solution
//parameter: μ – mutation rate (number of disregarded elements)
-----------------------------------------------------------------
begin
   apply Mutation to p using mutation rate μ, get partial (mutated) solution p⁻;
   apply Fast_Greedy_Adaptive_Procedure to the partial solution p⁻,
   get a perturbed complete solution p
   //the solution p is then to be improved by tabu search
end.
```

Figure 10
Pseudo-code of the random mutation procedure

```
procedure Mutation;  //random mutation procedure for the GP-QAP
//input: m,
//       p – current (complete) solution
//output: p⁻ – resulting (partial) solution
-----------------------------------------------------------------
begin
   for i := 1 to m − 1 do begin  //shuffling the items p(1), …, p(m)
      generate an integer j randomly, uniformly, i ≤ j ≤ m;
      p := p^{i,j}  //the items p(i) and p(j) are interchanged
   endfor;
   //after shuffling, the items p(m − μ + 1), …, p(m) are disregarded (no action is needed)
   p⁻ := p
end.
```

Figure 11
Pseudo-code of the fast greedy adaptive procedure

```
procedure Fast_Greedy_Adaptive_Procedure;  //fast greedy adaptive procedure for the GP-QAP
//input: n, m, B,
//       p – partial solution, where the elements p(m − μ + 1), …, p(m) are disregarded
//output: p – feasible (complete) solution
//parameter: μ – mutation rate (1 ≤ μ < m)
---------------------------------------------------------------------------------------
begin
  for i := 1 to n do for j := m − μ + 1 to m do c(i) := c(i) − B(i, p(j));  //fast (re)calculation of contributions (c)
  for i := 1 to n do Selected(i) := FALSE; for i := 1 to m − μ do Selected(p(i)) := TRUE;  //initialization of Selected
  i := m − μ; k := p(m − μ);
  for q := 1 to μ do begin  //cycle is repeated until the solution has been completed
    minimum_contribution := ∞;
    for j := 1 to n do
      if Selected(j) = FALSE then begin
        c(j) := c(j) + B(j, k); if c(j) < minimum_contribution then begin minimum_contribution := c(j); j_min := j endif
      endif;
    i := i + 1; p(i) := j_min;  //the element with the minimum contribution is added to the solution
    Selected(p(i)) := TRUE; k := j_min
  endfor;
  i := m + 1; for j := 1 to n do if Selected(j) = FALSE then begin p(i) := j; i := i + 1 endif
end.
```

2.6. Restart

The restart of the genetic algorithm takes place if the solutions of the population are not improved for L_{idle_gen} generations (L_{idle_gen} is an idle generations limit, which is set to $\lfloor 0.15N_{gen} \rfloor$, N_{gen} is the number of generations). The restart is performed by simply constructing the new population (see Section 2.1).

3. Results of Computational Experiments

Our new hybrid genetic algorithm (NHGA) was implemented by using C# programming language. The computational experiments have been carried out on a 3 GHz personal computer running Windows 7 Enterprise.

We have tested our algorithm on the medium and large-scaled GP-QAP instances with n = 256 and n = 1024, respectively. The instances are generated according to the method described in [15][3] . The grids are of dimensions 16×16 ($n_1 = n_2 = 16$) and 32×32 ($n_1 = n_2 = 32$), respectively. The grey density parameter m varies from 2 to 128 and from 2 to 512.

The values of the control parameters of NHGA used in the experiments are shown in Table 1. (Note that the calibration of the parameters was not performed.)

Firstly, we have experimented with the problems of size 256 and we have compared our algorithm with the improved genetic-evolutionary algorithm (IGEA) presented in [12]. To our knowledge, IGEA seems very likely to be the most efficient (to date) heuristic algorithm for the problems of this size. As the algorithms IGEA and NHGA constantly find the best known (pseudo-optimal) solutions (BKSs), we compare the run time performance, rather than the quality of solutions.

Thus, the experimentation was designed in such a way that the algorithms IGEA and NHGA were run 10 and 100 times, respectively. At every run, the algorithms are applied to a given m, each time starting from new random solutions. The current run is finished as soon as BKS has been found (even without

3 These instances can also be found at the website: http://www.personalas.ktu.lt/~alfmise/.

Table 1
Values of the control parameters of the hybrid genetic algorithm

Parameter	Value	Remarks
Population size, PS	20	
Number of generations, N_{gen}	40	
Idle generations limit, L_{idle_gen}	$\lfloor 0.15N_{gen} \rfloor$	$0 < L_{idle\_gen} \leq N_{gen}$
Distance threshold, DT	$\lfloor 0.25m \rfloor$	$0 \leq DT \leq m$
Number of hierarchical tabu search iterations, Q_{HIER}	384	$Q_{HIER} = Q \times Q_1 \times Q_2 \times Q_3 \times Q_4 \times Q_5 \times Q_6 \times Q_7$†
Number of tabu search iterations, τ	80	
Tabu tenure, h	$\lfloor 0.3m \rfloor$	$h > 0$
Idle iterations limit, L_{idle_iter}	$\lfloor 0.2\tau \rfloor$	$0 < L_{idle\_iter} \leq \tau$
Neighbourhood size factor, ρ	0.4	$\rho > 0$
Randomization coefficient, α	0.02	$0 < \alpha < 1$
Mutation rate, μ	$\lfloor 0.15m \rfloor$	$0 < \mu < m$

† $Q = Q_1 = Q_2 = Q_3 = Q_4 = Q_5 = Q_6 = 2$, $Q_7 = 3$.

reaching the limit of generations N_{gen}). The obtained run times (CPU times) of the algorithms to achieve the BKS for every m are reported in Table 2. For the algorithm IGEA, the run time averaged over 10 runs is presented. For the algorithm NHGA, the run time of the shortest run out of 100 runs is given. The comparison thus seems to be slightly unfair — we just wanted to bring to the light how fast our algorithm can run.

On the whole, the results demonstrate that NHGA clearly dominates IGEA. IGEA was able to slightly outperform NHGA in very few cases only (m=26,101,102,103). Figure 12 illustrates the incredible overall speed improvement of NHGA for m's varying from 30 to 100. It can be observed that, for some instances, the computation time is reduced by a factor of over 100 (!). Such small run times of NHGA indicate that NHGA is capable to obtain pseudo-optimal solutions at very early stages of the construction of the initial population with the help of the stand-alone hierarchical tabu search only. Very probably, the efficiency of NHGA could be improved even more by an accurate tuning of the values of the control parameters.

During the additional extensive, long-lasting experimentation, we were examining the algorithm NGHA on the large-sized problems (n=1024), which are much more difficult and time-consuming. It should be stressed that, nevertheless, we were successful in discovering new record breaking solutions for more than 190 values of m. The results are presented in Table 3. The CPU times are omitted. The new BKVs are in bold face. All the remaining values are from [13].

The further thorough experiments are needed to show (verify) the pseudo-optimality of the newly obtained best known solutions.

We also provide several visual representations (grey frames) corresponding to some of the new best known solutions (in particular, m=401,402,403,404,405,406,407,408) (see Figure 13). In the graphical illustrations, each

1024-square-grid is replicated 8 times horizontally and 8 times vertically for the visibility convenience.

Table 2

Results of the experiments with the medium-sized GP-QAP instances (n = 256)

m	Best known value (BKV)	Dev. from BKV	CPU time (sec)		m	Best known value (BKV)	Dev. from BKV	CPU time (sec)		m	Best known value (BKV)	Dev. from BKV	CPU time (sec)	
			IGEA	NHGA				IGEA	NHGA				IGEA	NHGA
2	1562	0	***0.0***	***0.00***	**45**	8674910	0	*150*	***5.91***	**87**	39389054	0	*25.0*	***0.19***
3	7810	0	***0.0***	***0.00***	**46**	9129192	0	*64*	***11.38***	**88**	40416536	0	*23.0*	***0.15***
4	15620	0	***0.0***	***0.00***	**47**	9575736	0	*3.1*	***0.29***	**89**	41512742	0	*183*	***6.98***
5	38072	0	***0.0***	***0.00***	**48**	10016256	0	*2.0*	***0.16***	**90**	42597626	0	*165*	***4.76***
6	63508	0	***0.0***	***0.00***	**49**	10518838	0	*3.4*	***0.56***	**91**	43676474	0	*224*	***17.64***
7	97178	0	***0.0***	***0.00***	**50**	11017342	0	*2.8*	***0.56***	**92**	44759294	0	*157*	***7.94***
8	131240	0	***0.0***	***0.00***	**51**	11516840	0	*7.5*	***0.83***	**93**	45870244	0	*214*	***19.18***
9	183744	0	***0.0***	***0.00***	**52**	12018388	0	*6.3*	***0.39***	**94**	46975856	0	*190*	***21.52***
10	242266	0	***0.0***	***0.00***	**53**	12558226	0	*4.6*	***0.54***	**95**	48081112	0	*169*	***2.14***
11	304722	0	*0.1*	***0.00***	**54**	13096646	0	*4.0*	***0.03***	**96**	49182368	0	*216*	***6.58***
12	368952	0	*0.1*	***0.00***	**55**	13661614	0	*10.1*	***0.12***	**97**	50344050	0	*213*	***23.88***
13	457504	0	*0.1*	***0.00***	**56**	14229492	0	*2.8*	***0.15***	**98**	51486642	0	*188*	***63.40***
14	547522	0	*0.1*	***0.00***	**57**	14793682	0	*2.2*	***0.37***	**99**	52660116	0	*201*	***50.53***
15	644036	0	*0.1*	***0.00***	**58**	15363628	0	*2.3*	***0.09***	**100**	53838088	0	*117*	***48.18***
16	742480	0	*0.1*	***0.00***	**59**	15981086	0	*3.5*	***0.71***	**101**	55014262	0	***84***	*156*
17	878888	0	*0.2*	***0.00***	**60**	16575644	0	*2.4*	***0.95***	**102**	56202826	0	***40.0***	*115*
18	1012990	0	*0.1*	***0.00***	**61**	17194812	0	*2.2*	***0.01***	**103**	57417112	0	***73***	*84.00*
19	1157992	0	*0.2*	***0.00***	**62**	17822806	0	*3.6*	***0.01***	**104**	58625240	0	*62*	***51.14***
20	1305744	0	*0.3*	***0.08***	**63**	18435790	0	*1.9*	***0.00***	**105**	59854744	0	*38.0*	***32.41***
21	1466210	0	*0.5*	***0.00***	**64**	19050432	0	*2.3*	***0.00***	**106**	61084902	0	*33.0*	***10.85***
22	1637794	0	*0.3*	***0.00***	**65**	19848790	0	*3.1*	***0.00***	**107**	62324634	0	*21.0*	***0.73***
23	1820052	0	*0.2*	***0.00***	**66**	20648754	0	*4.5*	***0.02***	**108**	63582416	0	*12.6*	***0.65***
24	2010846	0	*0.6*	***0.02***	**67**	21439396	0	*9.7*	***0.08***	**109**	64851966	0	*11.1*	***1.02***
25	2215714	0	*3.2*	***0.31***	**68**	22234020	0	*18.0*	***0.23***	**110**	66120434	0	*10.7*	***0.41***
26	2426298	0	***16.5***	*22.98*	**69**	23049732	0	*27.0*	***0.64***	**111**	67392724	0	*8.2*	***0.46***
27	2645436	0	*1.1*	***0.02***	**70**	23852796	0	*26.0*	***0.98***	**112**	68666416	0	*7.7*	***0.17***
28	2871704	0	*0.9*	***0.03***	**71**	24693608	0	*78*	***0.43***	**113**	69984758	0	*10.2*	***0.13***
29	3122510	0	*0.7*	***0.03***	**72**	25522408	0	*490*	***64.80***	**114**	71304194	0	*6.3*	***0.18***
30	3373854	0	*0.5*	***0.00***	**73**	26375828	0	*298*	***5.81***	**115**	72630764	0	*5.1*	***0.37***
31	3646344	0	*0.6*	***0.00***	**74**	27235240	0	*304*	***2.50***	**116**	73962220	0	*5.3*	***0.21***
32	3899744	0	*0.5*	***0.02***	**75**	28114952	0	*41.0*	***0.85***	**117**	75307424	0	*4.0*	***0.03***
33	4230950	0	*0.7*	***0.03***	**76**	29000908	0	*121*	***1.30***	**118**	76657014	0	*3.6*	***0.06***
34	4560162	0	*2.6*	***0.36***	**77**	29894452	0	*145*	***4.81***	**119**	78015914	0	*2.3*	***0.03***
35	4890132	0	*3.2*	***0.41***	**78**	30797954	0	*117*	***1.15***	**120**	79375832	0	*1.7*	***0.05***
36	5222296	0	*2.0*	***0.44***	**79**	31702182	0	*11.6*	***0.81***	**121**	80756852	0	*1.6*	***0.07***
37	5565236	0	*1.8*	***0.34***	**80**	32593088	0	*3.3*	***0.47***	**122**	82138768	0	*1.4*	***0.03***
38	5909202	0	*0.9*	***0.14***	**81**	33544628	0	*3.9*	***0.26***	**123**	83528554	0	*1.0*	***0.04***
39	6262248	0	*1.1*	***0.08***	**82**	34492592	0	*70*	***3.33***	**124**	84920540	0	*0.7*	***0.01***
40	6613472	0	*0.9*	***0.02***	**83**	35443938	0	*57.0*	***1.41***	**125**	86327812	0	*0.4*	***0.00***
41	7002794	0	*0.6*	***0.11***	**84**	36395172	0	*61*	***2.77***	**126**	87736646	0	*0.3*	***0.00***
42	7390586	0	*0.7*	***0.16***	**85**	37378800	0	*151*	***1.20***	**127**	89150166	0	*0.2*	***0.00***
43	7794422	0	*3.2*	***0.20***	**86**	38376438	0	*94*	***0.32***	**128**	90565248	0	*0.2*	***0.00***
44	8217264	0	*16.0*	***0.87***										

The deviation from BKV (Dev. from BKV) is calculated as the ratio $(z^* - BKV)/BKV$, where z^* denotes the algorithms' best found solution. The best known values of the objective function corresponding to the best known solutions are from [12].

Table 3
Results of the experiments with the large-sized GP-QAP instances (n = 1024)

m	BKV	m	BKV	m	BKV	m	BKV	m	BKV	m	BKV	m	BKV	m	BKV
2	390	66	5132250	130	23460170	194	57086766	258	106260632	322	176518174	386	**262819150**	450	363665156
3	1954	67	5312762	131	23897592	195	57739124	259	107273354	323	177756358	387	**264317294**	451	365361310
4	3908	68	5493398	132	24335656	196	58392270	260	108286116	324	178959204	388	**265791714**	452	**367056444**
5	9488	69	5675784	133	24773832	197	**59055298**	261	109299348	325	180158842	389	267231448	453	**368754728**
6	15882	70	5868614	134	25213730	198	**59710314**	262	**110313464**	326	181377560	390	**268698768**	454	**370451410**
7	24290	71	6061636	135	25653062	199	60357328	263	111323160	327	182598340	391	**270205824**	455	372157104
8	32808	72	6253544	136	26091040	200	61005880	264	**112349256**	328	183826382	392	**271688930**	456	373863658
9	45844	73	6451748	137	26536474	201	61656140	265	**113366358**	329	185039942	393	**273203922**	457	375575430
10	60310	74	6658646	138	**26980064**	202	62313154	266	114381534	330	186245648	394	**274660438**	458	377289052
11	75878	75	6866464	139	27426740	203	62979360	267	**115403394**	331	187480294	395	**276180114**	459	379005274
12	91852	76	7077272	140	27873238	204	63648372	268	**116415772**	332	188702624	396	**277670456**	460	380724964
13	114040	77	7287952	141	28319430	205	64329116	269	**117454596**	333	189934056	397	**279172978**	461	382449898
14	136706	78	7497962	142	28761578	206	65021762	270	**118462682**	334	**191134986**	398	**280677404**	462	**384174962**
15	160770	79	7708934	143	29211334	207	65721964	271	**119472414**	335	**192370030**	399	**282179032**	463	385902750
16	185552	80	7919112	144	29649520	208	66422364	272	**120505424**	336	**193596576**	400	**283712492**	464	387628048
17	218392	81	8147012	145	30118164	209	67136312	273	**121574572**	337	**194844496**	401	**285211916**	465	389368514
18	251618	82	8363950	146	30588480	210	67852496	274	**122629730**	338	196104848	402	**286746076**	466	391105794
19	288006	83	8600584	147	31065948	211	68585494	275	**123626322**	339	**197349714**	403	**288238662**	467	392843892
20	324794	84	8839620	148	31546098	212	**69315338**	276	**124716364**	340	**198600114**	404	**289784468**	468	394587900
21	365546	85	9079818	149	32025690	213	**70050648**	277	**125741472**	341	**199870596**	405	**291260654**	469	396332818
22	407406	86	9322672	150	32508848	214	**70789536**	278	**126807112**	342	201181102	406	**292833208**	470	398083462
23	451448	87	9563920	151	32992712	215	**71527862**	279	**127842588**	343	202519622	407	**294387042**	471	399837320
24	496888	88	9818424	152	33479148	216	**72259864**	280	**128937048**	344	203834574	408	**295934120**	472	401592882
25	549180	89	10074140	153	33968988	217	72990332	281	**130003342**	345	205167090	409	**297474388**	473	403351666
26	603368	90	10331422	154	34461110	218	73726832	282	**131077338**	346	206544390	410	**298998414**	474	**405112104**
27	659044	91	10600710	155	**34955468**	219	**74466810**	283	**132132040**	347	207925194	411	**300564200**	475	406875344
28	716280	92	10871062	156	**35450196**	220	75201458	284	**133187012**	348	209234904	412	**302129606**	476	**408637620**
29	777436	93	11138470	157	35944108	221	**75953890**	285	**134287484**	349	210612178	413	**303688980**	477	410409038
30	837798	94	11411510	158	36437606	222	76713866	286	**135364426**	350	211922934	414	**305224788**	478	412182568
31	907090	95	11679880	159	36933614	223	**77465610**	287	**136441394**	351	213304876	415	**306845268**	479	**413959340**
32	975008	96	11944352	160	37426912	224	**78218352**	288	**137549224**	352	214686716	416	**308393388**	480	415733856
33	1050792	97	12237102	161	37947342	225	78977922	289	**138637260**	353	216044260	417	**309969764**	481	417519180
34	1125558	98	12523996	162	38464394	226	**79744456**	290	**139677068**	354	**217376574**	418	**311420318**	482	419302686
35	1203646	99	**12813836**	163	**38982592**	227	80520900	291	**140801272**	355	**218738658**	419	**313094242**	483	421092758
36	1281132	100	**13103420**	164	**39500208**	228	81287994	292	**141875610**	356	**220109066**	420	**314652760**	484	422883164
37	1368444	101	**13398254**	165	40025416	229	82061894	293	**142916356**	357	221526988	421	**316172166**	485	424678088
38	1456842	102	13691306	166	**40550006**	230	82837128	294	**144026694**	358	222888300	422	**317825280**	486	426473544
39	1547598	103	13988062	167	**41078930**	231	83613898	295	**145175322**	359	224268836	423	**319428868**	487	428272184
40	1638808	104	14288780	168	**41606240**	232	84406568	296	**146290048**	360	225646838	424	**321022500**	488	430071632
41	1736236	105	14593444	169	**42140968**	233	**85225404**	297	**147454448**	361	227020504	425	**322637088**	489	431876322
42	1834074	106	14899130	170	**42673974**	234	**86030804**	298	**148527002**	362	228390592	426	**324266210**	490	433683572
43	1935946	107	15216394	171	**43219476**	235	86829778	299	**149672540**	363	229827232	427	**325898680**	491	435492454
44	2042792	108	**15537796**	172	**43787404**	236	87618540	300	**150827224**	364	231201722	428	**327457994**	492	437303524

Table 3 (continued)

m	BKV	*m*	BKV	*m*	BKV	*m*	BKV	*m*	BKV	*m*	BKV	*m*	BKV	*m*	BKV
45	2147200	109	**15857934**	173	**44361196**	237	88445972	301	**151952048**	365	**232578308**	429	**328993270**	493	439118116
46	2260650	110	16177106	174	**44933388**	238	89239062	302	**153122860**	366	**233960416**	430	**330616714**	494	440933678
47	2373506	111	**16504524**	175	**45511224**	239	**90075414**	303	**154282700**	367	**235302078**	431	**332246332**	495	442752278
48	2482832	112	16837956	176	**46091442**	240	90875504	304	**155417120**	368	**236712932**	432	**333874768**	496	444570032
49	2607474	113	17174378	177	**46680202**	241	91698908	305	**156547208**	369	**238200964**	433	**335514106**	497	446397066
50	2730510	114	17508602	178	**47274350**	242	92523578	306	**157684310**	370	**239582944**	434	**337154026**	498	448224550
51	2857088	115	17849756	179	47871440	243	93371894	307	**158836480**	371	**241044336**	435	**338796402**	499	450053654
52	2988998	116	18191920	180	**48462430**	244	94187252	308	159999648	372	**242479798**	436	340435998	500	451883116
53	3120248	117	18535442	181	**49056670**	245	95044544	309	161157378	373	**243893396**	437	**342076542**	501	453718668
54	3257234	118	18902942	182	**49654614**	246	95865322	310	162344014	374	**245361204**	438	**343718980**	502	455554546
55	3398018	119	19272770	183	**50258968**	247	96720682	311	163515706	375	**246783270**	439	**345362184**	503	457391626
56	3535048	120	19631156	184	**50876864**	248	97531736	312	164641724	376	**248246874**	440	**347009592**	504	459230104
57	3684478	121	20001764	185	**51494526**	249	98361638	313	165838280	377	**249710800**	441	**348669786**	505	461073188
58	3829950	122	20370638	186	**52115066**	250	99225594	314	**167015058**	378	**251141622**	442	350325606	506	462916382
59	3984538	123	20746696	187	**52731636**	251	100062350	315	**168180928**	379	**252535872**	443	**351981700**	507	464761614
60	4136400	124	21117234	188	**53348334**	252	100898116	316	**169354960**	380	**254011358**	444	**353638456**	508	466607612
61	4291962	125	21484868	189	**53959660**	253	101733670	317	**170529852**	381	**255486362**	445	**355296090**	509	468457260
62	4447434	126	21852518	190	**54571808**	254	102566006	318	**171723964**	382	**256914322**	446	356954184	510	470307298
63	4604860	127	22218924	191	55185346	255	103399158	319	**172910768**	383	**258421702**	447	358612252	511	472158510
64	4762688	128	22581376	192	55788864	256	104232704	320	**174113066**	384	**259861698**	448	360270272	512	474010112
65	4949042	129	23021790	193	56452088	257	105247082	321	175343494	385	**261377866**	449	361968220		

Figure 12

Illustration of the run time improvement. In the y-axis, we present the ratio of CPU_{IGEA} to CPU_{NHGA}, where $CPUI_{GEA}$, CPU_{NHGA} denote the run times of IGEA and NHGA, respectively

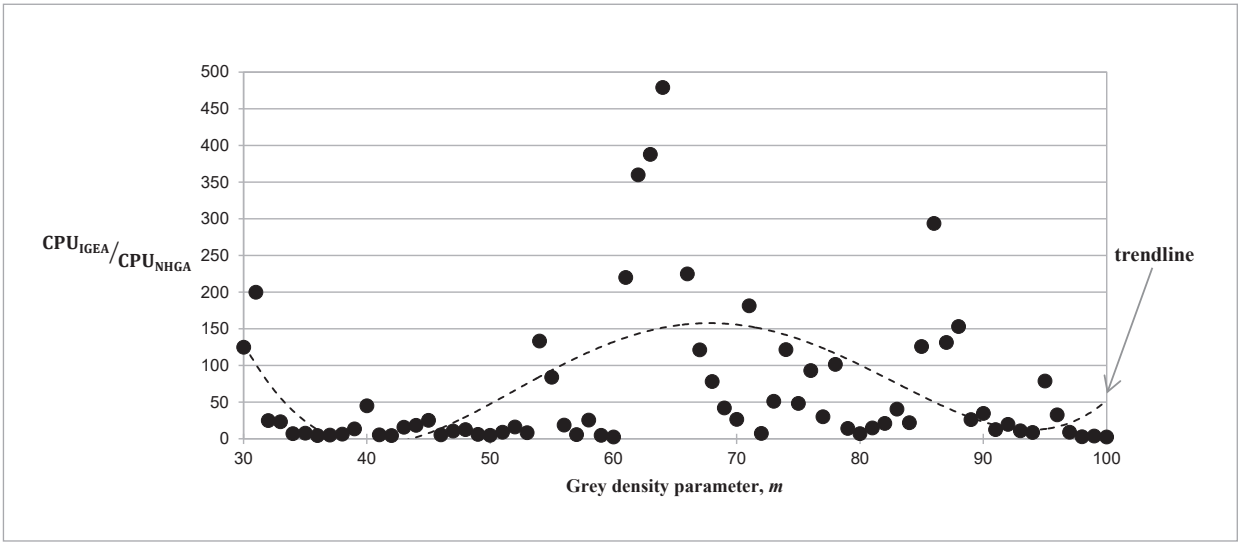
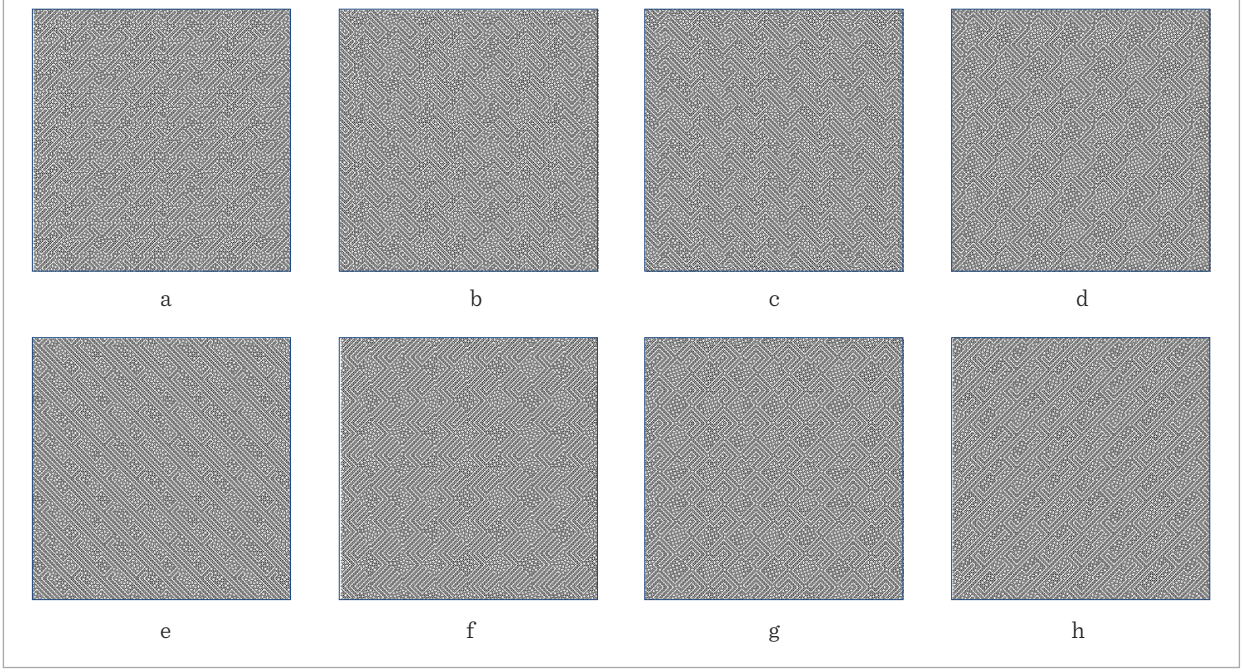


Figure 13
Examples of (pseudo-)optimal grey frames (n = 1024): (a) m =4 01, (b) m =4 02, (c) m = 403, (d) m = 404, (e) m = 405, (f) m = 406, (g) m =4 07, (h) m = 408



4. Concluding Remarks

In this paper, we have proposed a new improved hybrid genetic algorithm for solving the grey pattern quadratic assignment problem. In particular, the genetic algorithm is hybridized with the hierarchical iterated tabu search and this is adopted for the GP-QAP for the first time. The compacted, reduced neighbourhood is used. This enables very fast execution of the hierarchical ITS algorithm and the hybrid GA. In addition, we apply a smart combination of the iterated tabu search and the greedy adaptive perturbations. This allows beneficial balance between diversification and intensification during the iterative search process.

Our algorithm has been computationally tested on the medium and large-sized instances of the GP-QAP, where the instances size is equal to 256 and 1024, respectively. The results obtained from the conducted experiments demonstrate that the new hybridized GA is extremely effective and outperforms other the-state-of-the-art heuristic algorithms. The high efficiency is confirmed by the numerous best known solutions achieved for the challenging GP-QAP instances of size 1024.

Regarding the future work, it might be worthy to improve the population initialization and management mechanisms within hybrid GA to avoid presumable stagnation of the genetic process. It is also worth to try to further enhance the performance of HGA by developing new architecture of the genetic algorithm itself.

Additionally, our proposed new HGA might be adapted for other types of combinatorial optimization problems.

References

1. Aarts, E. H. L., Lenstra, J.K. (Eds.). Local Search in Combinatorial Optimization, Wiley, 1997.
2. Çela, E. The Quadratic Assignment Problem: Theory and Algorithms, Kluwer, 1998. https://doi.org/10.1007/978-1-4757-2787-6
3. Daskin, M. Network and Discrete Location: Models, Algorithms and Applications, John Wiley, 1995. https://doi.org/10.1002/9781118032343
4. Dell'Amico, M., Trubian, M. Solution of Large Weighted Equicut Problems. European Journal of Operational Research, 1998, 106(2-3), 500-521. https://doi.org/10.1016/S0377-2217(97)00287-7
5. Drezner, Z. Finding a Custer 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
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. Feo, T. A., Resende, M. G. C. Greedy Randomized Adaptive Search Procedures. Journal of Global Optimization, 1995, 6(2), 109-133. https://doi.org/10.1007/BF01096763
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. Hussin, M. S., Stützle, T. Hierarchical Iterated Local Search for the Quadratic Assignment Problem. In: Blesa, M. J., Blum, C., Di Gaspero, L., Roli, A., Sampels, M., Schaerf, A. (Eds.), Hybrid Metaheuristics, HM 2009, Lecture Notes in Computer Science, Springer, 2009, 5818, 115-129. https://doi.org/10.1007/978-3-642-04918-7_9
11. Misevičius, A. Experiments with Hybrid Genetic Algorithm for the Grey Pattern Problem. Informatica, 2006, 17(2), 237-258.
12. 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
13. Misevičius, A., Guogis, E., Stanevičienė, E. Computational Algorithmic Generation of High-Quality Colour Patterns. In: Skersys, T., Butleris, R., Butkienė, R. (Eds.), Information and Software Technologies (ICIST 2013), Communications in Computer and Information Science, Springer, Berlin, Heidelberg, 2013, 403, 285-296. https://doi.org/10.1007/978-3-642-41947-8_24
14. Misevičius, A., Rubliauskas, D. Performance of Hybrid Genetic Algorithm for the Grey Pattern Problem. Information Technology and Control, 2005, 34(1), 15-24.
15. 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
16. Taillard, E., Gambardella, L. M. Adaptive Memories for the Quadratic Assignment Problem. Tech. Report IDSIA-87-97, Lugano, Switzerland, 1997.
17. Tizhoosh, H. R. Opposition-Based Learning: A New Scheme for Machine Intelligence. In: Mohammadian, M. (Ed.), Proceedings of International Conference on Computational Intelligence for Modeling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA/IAWTIC), IEEE Press, 2005, 2, 695-701. https://doi.org/10.1109/CIMCA.2005.1631345
18. 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
19. 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