




Article

The Color Mix Problem

Alfonsas Misevičius *, Aleksandras Andrejevas, Armantas Ostreika , Tomas Blažauskas 
and Liudas Motiejūnas 

Department of Multimedia Engineering, Kaunas University of Technology, Studentu st. 50-400, LT-51368 Kaunas, Lithuania; aleksandras.andrejevas@ktu.edu (A.A.); armantas.ostreika@ktu.lt (A.O.); tomas.blazauskas@ktu.lt (T.B.); liudas.motiejunas@ktu.lt (L.M.)

* Correspondence: alfonsas.misevicius@ktu.lt

Abstract: In this paper, we introduce a new combinatorial optimization problem entitled the color mix problem (CMP), which is a more general case of the grey pattern quadratic assignment problem (GP-QAP). Also, we propose an original hybrid genetic-iterated tabu search algorithm for heuristically solving the CMP. In addition, we present both analytical solutions and graphical visualizations of the obtained solutions, which clearly demonstrate the excellent performance of the proposed heuristic algorithm.

Keywords: color mix problem; combinatorial optimization; heuristic algorithms; genetic algorithms; tabu search

1. Introduction

The color mix problem (CMP; in this work we consider three colors) can be formulated as a particular case of the well-known problem—the quadratic assignment problem (QAP) [1]. The formulation is as follows. Given two integer quadratic matrices $\mathbf{A} = (a_{ij})_{n \times n}$, $\mathbf{B} = (b_{ij})_{n \times n}$, and a set Π_n of all possible permutations of the integers from 0 to $n - 1$, find a permutation $p \in \Pi_n$ that minimizes the following function:

$$z(p) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_{ij} b_{p(i)p(j)}, \quad (1)$$

where $a_{ij} = 1$ for $i = 0, \dots, m_1 - 1, j = m_1, m_1 + 1, \dots, m - 1, i = m_1, m_1 + 1, \dots, m - 1, j = 0, \dots, m_1 - 1$; $a_{ij} = 2$ for $i, j = 0, \dots, m_1 - 1, m_1, m_1 + 1, \dots, m - 1$, and $a_{ij} = 0$ otherwise ($1 \leq m_1 < m, m_1 < m < n$) (see Figure 1). So, the objective can be transformed to finding the permutation elements $p(0), \dots, p(m - 1)$ ($0 \leq p(i) \leq n - 1, i = 0, \dots, m - 1$) such that the following simplified function is minimized:

$$z(p) = \sum_{i=0}^{m_1-1} \left(2 \sum_{j=0}^{m_1-1} b_{p(i)p(j)} + \sum_{j=m_1}^{m-1} b_{p(i)p(j)} \right) + \sum_{i=m_1}^{m-1} \left(\sum_{j=0}^{m_1-1} b_{p(i)p(j)} + 2 \sum_{j=m_1}^{m-1} b_{p(i)p(j)} \right). \quad (2)$$

The items in the matrix \mathbf{B} are symmetric distances between every pair of n elements. The distances are calculated by adopting the definition as in [2]:

$$b_{kl} = b_{rn_2+s,tn_2+u} = \omega_{rstu}, \quad \omega_{rstu} = \max_{w_1, w_2 \in \{-1, 0, 1\}} \left\{ \frac{1}{(r - t + w_1 n_1)^2 + (s - u + w_2 n_2)^2} \right\}, \quad (3)$$

where $k, l = 0, \dots, n - 1, r, t = 0, \dots, n_1 - 1, s, u = 0, \dots, n_2 - 1, n = n_1 \times n_2$ (in our case, $n_1 = n_2$).



Citation: Misevičius, A.; Andrejevas, A.; Ostreika, A.; Blažauskas, T.; Motiejūnas, L. The Color Mix Problem. *Appl. Sci.* **2021**, *11*, 7263. <https://doi.org/10.3390/app11167263>

Received: 13 May 2021
Accepted: 30 July 2021
Published: 6 August 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	2	2	1	1	1	1	1	0	0	0	0	0	0	0
1	2	0	2	1	1	1	1	1	0	0	0	0	0	0	0	0
2	2	2	0	1	1	1	1	1	0	0	0	0	0	0	0	0
3	1	1	1	0	2	2	2	2	0	0	0	0	0	0	0	0
4	1	1	1	2	0	2	2	2	0	0	0	0	0	0	0	0
5	1	1	1	2	2	0	2	2	0	0	0	0	0	0	0	0
6	1	1	1	2	2	2	0	2	0	0	0	0	0	0	0	0
7	1	1	1	2	2	2	2	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 1. Illustration of entries of matrix A ($n = 16, m = 8, m_1 = 3$).

In other words, we have a grid (rectangle) of dimensions $n_1 \times n_2$ (for simplicity, $n_1 = n_2$). More precisely, we have $n = n_1 \times n_2$ squares (dots, points) of identical size. For convenience, it may be recommended that n be equal to 2 raised to a power λ , i.e., 2^λ (for example, $2^6, 2^8$, and so on). The squares are evenly positioned in n_1 horizontal lines and n_2 vertical columns. There are m_1 squares of color $color_1$ (say, red). Also, there are $m_2 = m - m_1$ squares of color $color_2$ (for example, green) and also $m_3 = n - m = n - m_1 - m_2$ squares of color $color_3$ (say, blue). This forms a color mix pattern, i.e., a mixture (composition) of three colors (red, green, blue), which can be characterised by the parameters $\frac{m_1}{n}, \frac{m_2}{n}, \frac{m_3}{n}$. Then, the aim is to obtain a configuration of all the points, i.e., a color pattern, where the colors (color squares) are distributed in the most uniform possible way. In other words, we are seeking to get fine, well-balanced color combinations; that is, we wish to obtain the most uniform possible pattern for each color (see Figure 2). It should be noted that the color mix problem is a more complex case of the grey pattern problem (GPP) [2], where one has n points in the rectangle and there are m black points, while the remaining points are white. The goal is to have a grey pattern where the black points are scattered as regularly as possible on the rectangle.

The values of the elements of the analytical solution, i.e., permutation p are related to the corresponding coordinates (r, s) of the sites (locations) on the grid through the following formulas: $r = \frac{p(i)}{n_2}, s = p(i) \bmod n_2, i = 0, \dots, n - 1$. Note that the first m_1 elements in the permutation determine the sites on the grid where the red squares are placed in. The next m_2 elements define the positions of the green squares, and so on.

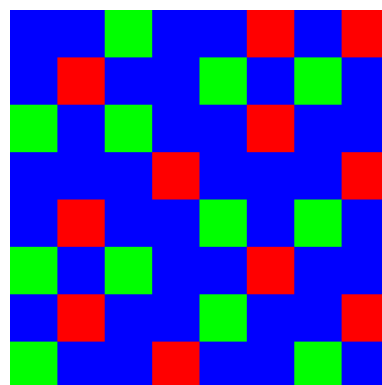


Figure 2. Illustration of the (optimal) color mix pattern ($n = 64, n_1 = n_2 = 8, m = 8, m_1 = 11, m_2 = 12, m_3 = 41$).

The CMP, thus, is a theoretical optimization problem of combinatorial character. However, we believe that the CMP can potentially have some important areas of practical applications, for example, color halftoning, fine digital textures, modern computer graphics/multimedia, computer visual arts [3–7]. Yet another application example would be the selection of groups of persons out of n available persons. In this case, the “distance” between persons would be, for example, a measure of the capability to cooperate and the ideal groups would have the least risk for conflict and the most potential for harmony.

For the solution of the color mix problem, (meta)heuristic optimization algorithms can be applied. Although heuristics do not guarantee the optimality of the found solutions, they enable us to obtain sufficiently high-quality solutions within an affordable computation time [8].

There are several heuristic algorithms—which although have not been examined on our considered problem yet—but still are thought to be potentially applicable to the CMP. As an example, single solution-based local search and simulated annealing algorithms were examined in [9] for solving the grey pattern problem [2], which is the “sibling” of the CMP. Modern tabu search methodology-based algorithms were considered in [9,10]. Population-based-evolutionary and swarm intelligence algorithms constitute another class of powerful (meta)heuristic algorithms [2,9,11]. The main feature of this kind of algorithms is that these algorithms deal with the sets of solutions, and this property seems to be of crucial importance [12]. More to this, the population-based algorithms are easy to hybridize with the single solution-based algorithms—which is a natural way to further enhance the efficiency of the population-based algorithms [13,14].

That said, the main contribution of this article is that a new combinatorial optimization problem—the color mix problem—is introduced and computationally studied. For the solution of this problem, a hybrid genetic-hierarchical iterated tabu search (HITS) algorithm is proposed. The basis of the HITS algorithm is, in turn, the multiple reuse of the (iterated) tabu search; it also encompasses a specific greedy adaptive search procedure combined with a random mutation process. (In ref. [14], it is clearly stated that “currently there is no known polynomial-time algorithm to exactly solve the grey pattern problem”. And we conjecture that this should be even more true for our considered color mix problem. So, the only rational option for obtaining high quality solutions in a reasonable amount of time is to employ heuristic algorithms. In [14], the state-of-the-art hybrid genetic-HITS algorithm was proposed for the GPP, which performed extremely well. Therefore, we have naturally chosen this sort of algorithm for the CMP. The obtained results from the computational experiments confirm the high performance of the proposed algorithm for the CMP.)

The remainder of this paper is organized as follows: In Section 2, some preliminaries (basic definitions) are given. The detailed description of the genetic-hierarchical iterated tabu search algorithm and its parts is presented in Section 3. Then, in Section 4, the results of the computational experiments are provided. The paper is completed with some concluding comments.

2. Preliminaries

Suppose that $p(v)$ ($v = 0, \dots, n - 1$) and $p(w)$ ($w = 0, \dots, n - 1, v \neq w$) are two elements in the permutation p . Then $p^{v,w}$ is defined in the following way:

$$p^{v,w}(i) = \begin{cases} p(i), & i \neq v, w \\ p(v), & i = w \\ p(w), & i = v \end{cases}; i = 0, \dots, n - 1. \quad (4)$$

A neighbourhood function $\Theta: \Pi_n \rightarrow 2^{\Pi_n}$ assigns for each $p \in \Pi_n$ a set of neighbouring solutions $\Theta(p) \subseteq \Pi_n$. In particular, the 1-exchange neighbourhood function Θ_1 is defined as follows:

$$\Theta_1(p) = \{p' : p' \in \Pi_n, (\delta_1(p, p') = 1 \vee \delta_2(p, p') = 1) \wedge \delta_3(p, p') \leq 1\}, \quad (5)$$

where $p \in \Pi_n$ and $\delta_1(p, p'), \delta_2(p, p'), \delta_3(p, p')$ denote the corresponding distance measures between the solutions p and p' . The distances $\delta_1, \delta_2, \delta_3$ between two solutions p and p' can be defined according to the following formulas:

$$\delta_1(p, p') = m_1 - |\{p(i) : i = 0, \dots, m_1 - 1\} \cap \{p'(i) : i = 0, \dots, m_1 - 1\}|, \tag{6}$$

$$\delta_2(p, p') = m_2 - |\{p(i) : i = m_1, \dots, m - 1\} \cap \{p'(i) : i = m_1, \dots, m - 1\}|, \tag{7}$$

$$\delta_3(p, p') = m - |\{p(i) : i = 0, \dots, m - 1\} \cap \{p'(i) : i = 0, \dots, m - 1\}|, \tag{8}$$

where $m_2 = m - m_1$.

The solution $p^{v,w}$ can simply be obtained from the existing solution p by accomplishing the pairwise interchange move $\phi(p, v, w): \Pi_n \times N \times N \rightarrow \Pi_n$, which swaps the v th and w th elements in the given solution. Thus, $p^{v,w} = \phi(p, v, w)$.

For the neighbouring solutions p and $p^{v,w}$, the difference in the values of the objective function is calculated in $O(1)$ time by using the following formula:

$$\Delta_z(p^{v,w}, p) = z(p^{v,w}) - z(p) = \begin{cases} 2(c_1(p(w)) - c_1(p(v)) - 2b_{p(v)p(w)}), v \in \{0, \dots, m_1 - 1\}, w \in \{m, \dots, n - 1\} \\ 2(c_2(p(w)) - c_2(p(v)) - 2b_{p(v)p(w)}), v \in \{m_1, \dots, m - 1\}, w \in \{m, \dots, n - 1\} \\ 2(c_3(p(w)) - c_3(p(v)) - 2b_{p(v)p(w)}), v \in \{0, \dots, m_1 - 1\}, w \in \{m_1, \dots, m - 1\} \end{cases}, \tag{9}$$

where $c_1(x), c_2(x), c_3(x)$ are contributions of the element x (the sum of related distances):

$$c_1(x) = 2 \sum_{y=0}^{m_1-1} b_{x,p(y)} + \sum_{y=m_1}^{m-1} b_{x,p(y)}, x = 0, \dots, n - 1, \tag{10}$$

$$c_2(x) = \sum_{y=0}^{m_1-1} b_{x,p(y)} + 2 \sum_{y=m_1}^{m-1} b_{x,p(y)}, x = 0, \dots, n - 1, \tag{11}$$

$$c_3(x) = \sum_{y=0}^{m_1-1} b_{x,p(y)} - \sum_{y=m_1}^{m-1} b_{x,p(y)}, x = 0, \dots, n - 1. \tag{12}$$

Note that, after the exchange of the permutation elements $p(v)$ and $p(w)$, the contributions must be updated according to these equations:

$$c_1(x) = \begin{cases} c_1(x) + 2(b_{x,p(v)} - b_{x,p(w)}), v \in \{0, \dots, m_1 - 1\}, w \in \{m, \dots, n - 1\} \\ c_1(x) + (b_{x,p(v)} - b_{x,p(w)}), v \in \{m_1, \dots, m - 1\}, w \in \{m, \dots, n - 1\} \\ c_1(x) + (b_{x,p(v)} - b_{x,p(w)}), v \in \{0, \dots, m_1 - 1\}, w \in \{m_1, \dots, m - 1\} \end{cases}, x = 0, \dots, n - 1, \tag{13}$$

$$c_2(x) = \begin{cases} c_2(x) + (b_{x,p(v)} - b_{x,p(w)}), v \in \{0, \dots, m_1 - 1\}, w \in \{m, \dots, n - 1\} \\ c_2(x) + 2(b_{x,p(v)} - b_{x,p(w)}), v \in \{m_1, \dots, m - 1\}, w \in \{m, \dots, n - 1\} \\ c_2(x) - (b_{x,p(v)} - b_{x,p(w)}), v \in \{0, \dots, m_1 - 1\}, w \in \{m_1, \dots, m - 1\} \end{cases}, x = 0, \dots, n - 1, \tag{14}$$

$$c_3(x) = \begin{cases} c_3(x) + (b_{x,p(v)} - b_{x,p(w)}), v \in \{0, \dots, m_1 - 1\}, w \in \{m, \dots, n - 1\} \\ c_3(x) - (b_{x,p(v)} - b_{x,p(w)}), v \in \{m_1, \dots, m - 1\}, w \in \{m, \dots, n - 1\} \\ c_3(x) + 2(b_{x,p(v)} - b_{x,p(w)}), v \in \{0, \dots, m_1 - 1\}, w \in \{m_1, \dots, m - 1\} \end{cases}, x = 0, \dots, n - 1. \tag{15}$$

Using the above Equations (9)–(15), the computational complexity of complete evaluation of the neighbourhood Θ_1 is proportional to $O(mn + m_1m_2)$, where $m_2 = m - m_1$.

3. Hybrid Genetic Algorithm for the Solution of the Color Mix Problem

The principle of the used algorithm is similar to that of the hybrid genetic algorithm [15] (for more complete information on the functioning of genetic algorithms, the reader is referred to [16]), where global explorative/diversified search of the genetic operators is in teamwork with the local exploitative/intensified search. The exploitative search, in particular, is performed by the iterated hierarchical tabu search procedure (see Section 3.4.1).

In our algorithm, the permutation elements $p(0), p(1), \dots, p(n-1)$ are simply linked to the genes of individuals' chromosomes. No encoding is required. Meanwhile, the objective function value of the given solution, $z(p)$, is associated with the fitness of individuals.

The basic components of our hybrid genetic algorithm are as follows: (1) construction of initial population; (2) parent selection for crossover procedure; (3) crossover procedure; (4) improvement of the offspring by the HITS algorithm; (5) population replacement. The top-level pseudocode of the genetic algorithm is provided in Algorithm 1.

Algorithm 1 Hybrid_Genetic_Algorithm

```
// top-level pseudocode of the hybrid genetic algorithm
// input:  $B, n, m, m_1$ 
// output:  $P$ 
// parameters:  $G, PS, DT, idle\_gen\_limit$ 
begin
 $P \leftarrow \text{Initial\_Population\_Construction}(PS); i \leftarrow 0; i' \leftarrow 0;$ 
do
 $p', p'' \leftarrow \text{Selection}(P);$ 
 $p^\circ, c_1, c_2, c_3 \leftarrow \text{Greedy\_Crossover}(B, n, m, m_1, p', p'');$ 
 $p^k, z^k \leftarrow k\_Level\_Hierarchical\_Iterated\_Tabu\_Search(B, n, m, m_1, p^\circ, z(p^\circ), c_1, c_2, c_3);$ 
 $j \leftarrow 0; include\_solution \leftarrow \text{TRUE};$ 
do
if ( $(\text{Distance}(P(j), p^k) < DT \vee z^k = z(P(j))) \wedge z^k \geq z(P(0))$ )  $include\_solution \leftarrow \text{FALSE}$ 
endif;
 $j \leftarrow j + 1$ 
while ( $j \neq PS$ );
if ( $include\_solution$ )
if ( $z(P(0)) > z^k$ )  $P(0) \leftarrow p^k;$ 
elseif ( $z(P(PS - 1)) > z^k$ )
 $P(\text{Upper\_Index}(P)) \leftarrow p^k$ 
endif;
 $i' \leftarrow i;$ 
elseif ( $i - i' > idle\_gen\_limit \wedge i < G - idle\_gen\_limit$ )
 $P \leftarrow \text{Population\_Restart}(PS); i' \leftarrow i$ 
endif;
 $i \leftarrow i + 1$ 
while ( $i \neq G$ )
end.
```

Notes. 1. G, PS denote the total number of generations and the population size, respectively. 2. The distance (**Distance**) is calculated using the Equations (6) and (7). 3. DT denotes the distance threshold between solutions. 4. $idle_gen_limit$ denotes the limit of idle generations (we used $idle_gen_limit = \max\{3, 0.05G\}$). 5. The mutation process is embedded into the k -level hierarchical iterated tabu search algorithm. 6. The population restart (**Population_Restart**) is performed by applying the "clone" of the initial population construction procedure.

3.1. Construction of Initial Population

There are two stages of the construction of initial population. In the first one, the pre-initial (primordial) population is generated and improved; in the second one, the truncation

(culling) of the obtained population is performed. Firstly, $PS' = PS \times C_1$ members of the pre-initial population P are randomly generated, where PS denotes the population size, and C_1 ($C_1 \geq 1$) is the predefined parameter to control the size of the pre-initial population.

Every generated solution is subject to improvement by the HITS algorithm. There is a special condition. If the improved solution (p) is better than the best-found solution, then the improved solution substitutes the best-found solution. Otherwise, it is checked if the minimum distance ($\min_{p \in P} \{\delta_1(p, p) + \delta_2(p, p)\}$) between the solution p and the present population solutions is greater than the predetermined distance threshold, DT . If this is the case, then the new solution enters the population. Otherwise, the new solution is ignored and a random solution is included instead. The distance threshold is related to the value of m through the following equation: $DT = \max\{2, \varepsilon m\}$, where ε is the distance threshold coefficient ($0 < \varepsilon \leq 1$). In the second stage, $(C_1 - 1)PS$ worst members of the primordial population are deleted and just PS members survive for the future generations.

3.2. Parent Selection

In our algorithm, the parents are selected using a random selection rule.

3.3. Crossover Procedure

One of the main purposes of crossover is to diversify the search process by recombining the genetic information present in the parental chromosomes. In our hybrid genetic algorithm, the crossover operator takes place at each generation of GA, that is, the crossover probability is equal to exactly 1. Note that the straightforward application of the standard bit-string-based crossovers to the color mix problem may not be necessary effective, because the factual chromosomal locations of genes are irrelevant in the CMP. Still, it is rational to preserve the common genes of parents in the recombination process. Meanwhile, the other genes are formed by adopting the heuristic greedy-based approach by choosing the genes from the parents. More precisely, the genes are chosen from either the first or the second parent such that the currently selected gene contributes least to the value of the objective function. The pseudocode of the greedy crossover procedure is given in Algorithm 2. More details on this procedure can be found in [13].

It can be seen that the computational time complexity of the greedy crossover operator is proportional to $O(mn + (m_1)^2 + (m_2)^2)$.

Algorithm 2 Greedy_Crossover

```
// pseudocode of the greedy crossover procedure
// input:  $B, n, m, m_1, p', p''$ 
// output:  $p^\circ, c_1, c_2, c_3$ 
begin
   $p^\circ \leftarrow \{p'(i) | i = 0, \dots, m_1 - 1\} \cap \{p''(i) | i = 0, \dots, m_1 - 1\}$ ;  $common\_elements \leftarrow |p^\circ|$ ;
   $i \leftarrow 0$ ;
  do
     $j \leftarrow 0$ ;  $sum \leftarrow 0$ ; while ( $j \neq common\_elements$ )  $sum \leftarrow sum + b_{i,p(j)}$ ;  $j \leftarrow j + 1$  endwhile;
     $c_1(i) \leftarrow 2 \times sum$ ;  $c_2(i) \leftarrow sum$ ;  $c_3(i) \leftarrow sum$ ;  $i \leftarrow i + 1$ 
  while ( $i \neq n$ );
  while ( $common\_elements \neq m_1$ )
     $\#\_of\_points \leftarrow m_1$ ;
   $p', common\_elements, c_1, c_2, c_3 \leftarrow$ 
    Greedy_Offspring_Creation( $B, n, m, m_1, p', c_1, common\_elements, \#\_of\_points$ );
   $p'', common\_elements, c_1, c_2, c_3 \leftarrow$ 
    Greedy_Offspring_Creation( $B, n, m, m_1, p'', c_1, common\_elements, \#\_of\_points$ )
  endwhile;
```

```

 $p^\circ \leftarrow p^\circ \cup (\{p'(i) | i = m_1, \dots, m-1\} \cap \{p''(i) | i = m_1, \dots, m-1\});$   $common\_elements \leftarrow |p^\circ|;$ 
 $i \leftarrow 0;$ 
do
   $j \leftarrow m_1;$   $sum \leftarrow 0;$  while ( $j \neq common\_elements$ )  $sum \leftarrow sum + b_{i,p(j)};$   $j \leftarrow j + 1$  endwhile;
   $c_1(i) \leftarrow c_1(i) + sum;$   $c_2(i) \leftarrow c_2(i) + 2 \times sum;$   $c_3(i) \leftarrow c_3(i) - sum;$   $i \leftarrow i + 1$ 
while ( $i \neq n$ );
while ( $common\_elements \neq m$ )
   $\#\_of\_points \leftarrow m;$ 
   $p', common\_elements, c_1, c_2, c_3 \leftarrow$ 
    Greedy_Offspring_Creation( $B, n, m, m_1, p', c_2, common\_elements, \#\_of\_points$ );
   $p'', common\_elements, c_1, c_2, c_3 \leftarrow$ 
    Greedy_Offspring_Creation( $B, n, m, m_1, p'', c_2, common\_elements, \#\_of\_points$ )
endwhile;
 $p^\circ \leftarrow$  Filling_Remaining_Elements( $p^\circ, p', p''$ )
end.

```

Notes. 1. The pseudocode of the offspring creation procedure (**Greedy_Offspring_Creation**) is presented in Algorithm 3. 2. The (unoccupied) positions $p(m), p(m+1), \dots, p(n-1)$ are filled in with the remaining unselected elements using the procedure **Fill_Remaining_Elements**.

Algorithm 3 Greedy_Offspring_Creation

```

// pseudocode of the greedy offspring creation sub-procedure
// input:  $B, n, m, m_1, p, this\_c, common\_elements, \#\_of\_points, c_1, c_2, c_3$ 
// output:  $p, common\_elements, c_1, c_2, c_3$ 
begin
  if ( $common\_elements \neq \#\_of\_points$ )
     $minimum\_contribution \leftarrow \infty;$   $j \leftarrow 0;$ 
    do
      if (element  $p(j)$  is not yet selected)
        if ( $this\_c(p(j)) < minimum\_contribution$ )  $minimum\_contribution \leftarrow this\_c(p(j));$   $k \leftarrow j$ 
    endif;
     $j \leftarrow j + 1$ 
  while ( $j \neq \#\_of\_points$ );
  if ( $minimum\_contribution \neq \infty$ )
    if ( $\#\_of\_points = m_1$ )
       $i \leftarrow 0;$ 
      do
         $c_1(i) \leftarrow c_1(i) + 2 \times b_{i,p(k)};$   $c_2(i) \leftarrow c_2(i) + b_{i,p(k)};$   $c_3(i) \leftarrow c_3(i) + b_{i,p(k)};$   $i \leftarrow i + 1$ 
      while ( $i \neq n$ );
    elseif ( $\#\_of\_points = m$ )
       $i \leftarrow 0;$ 
      do
         $c_1(i) \leftarrow c_1(i) + b_{i,p(k)};$   $c_2(i) \leftarrow c_2(i) + 2 \times b_{i,p(k)};$   $c_3(i) \leftarrow c_3(i) - b_{i,p(k)};$   $i \leftarrow i + 1$ 
      while ( $i \neq n$ );
    endif;
     $p(common\_elements) \leftarrow p(k);$   $common\_elements \leftarrow common\_elements + 1$ 
  endif
endif
end.

```

3.4. Offspring Improvement

3.4.1. Hierarchical Iterated Tabu Search Algorithm

Every produced offspring is subject to improvement by the hierarchical iterated tabu search procedure which can also be seen as a multi-level tabu search [17] procedure, where the basic idea is the cyclic (multiple) reuse of the tabu search algorithm. The k -level

hierarchical iterated tabu search algorithm is consisting of three main ingredients: (1) call of the $k - 1$ -level hierarchical iterated tabu search procedure; (2) selection (acceptance) of the solution for perturbation, i.e., mutation; (3) perturbation of the selected solution.

The perturbed solution serves as an input for the autonomous (self-contained) TS procedure. The TS algorithm gives back an optimized solution, and so on. The solution acceptance rule is very simple: we always accept the recently found improved solution.

The overall iterative process is continued until a given total number of iterations have been accomplished (see Algorithm 4). The time complexity of HITS is proportional to $O(mn + m_1m_2)$, although the proportionality coefficient may be quite large.

We recall that we have used the 2-level hierarchical tabu search algorithm ($k = 2$).

Algorithm 4 k _Level_Hierarchical_Iterated_Tabu_Search

```
// pseudocode of the  $k$ -level hierarchical iterated tabu search algorithm
// input:  $B, n, m, m_1, p, z, c_1, c_2, c_3$ 
// output:  $p^{<k>}, z^{<k>}$ 
// parameters:  $Q^{<k>}$ 
begin
   $p^{<k>} \leftarrow p; z^{<k>} \leftarrow z; c_1^{<k>} \leftarrow c_1; c_2^{<k>} \leftarrow c_2; c_3^{<k>} \leftarrow c_3; q^{<k>} \leftarrow 0;$ 
  do
     $p^{<k-1>}, z^{<k-1>}, c_1^{<k-1>}, c_2^{<k-1>}, c_3^{<k-1>} \leftarrow k -$ 
    1_Level_Hierarchical_Iterated_Tabu_Search( $B, n, m, m_1, p^{<k>}, z^{<k>}, c_1^{<k>}, c_2^{<k>}, c_3^{<k>}$ );
    if ( $z^{<k>} > z^{<k-1>}$ )  $p^{<k>} \leftarrow p^{<k-1>}; z^{<k>} \leftarrow z^{<k-1>}; c_1^{<k>} \leftarrow c_1^{<k-1>}; c_2^{<k>} \leftarrow c_2^{<k-1>};$ 
     $c_3^{<k>} \leftarrow c_3^{<k-1>}$  endif;
     $q^{<k>} \leftarrow q^{<k>} + 1;$ 
    if ( $q^{<k>} \neq Q^{<k>}$ ) // perturbation of the solution  $p^{<k>}$ 
       $p^{<k>} \leftarrow$  Mutation_Procedure( $p^{<k>}$ );
     $p^{<k>}, c_1^{<k>}, c_2^{<k>}, c_3^{<k>} \leftarrow$  Greedy_Adaptive_Search_Procedure( $B, n, m, m_1, p^{<k>}, c_1^{<k>}, c_2^{<k>}, c_3^{<k>}$ )
    endif
  while ( $q^{<k>} \neq Q^{<k>}$ )
end.
```

Note. The (zero-level) 0-level HITS procedure is correspondent to the iterated tabu search (ITS) procedure (see Algorithm 5).

Algorithm 5 Iterated_Tabu_Search

```
// pseudocode of the iterated tabu search algorithm
// input:  $B, n, m, m_1, p, z, c_1, c_2, c_3$ 
// output:  $p^{<0>}, z^{<0>}, c_1^{<0>}, c_2^{<0>}, c_3^{<0>}$ 
// parameters:  $Q^{<0>}$ 
begin
   $p^{<0>} \leftarrow p; z^{<0>} \leftarrow z; c_1^{<0>} \leftarrow c_1; c_2^{<0>} \leftarrow c_2; c_3^{<0>} \leftarrow c_3; q^{<0>} \leftarrow 0;$ 
  do
     $p^{\bullet}, z^{\bullet}, c_1^{\bullet}, c_2^{\bullet}, c_3^{\bullet} \leftarrow$  Tabu_Search( $B, n, m, m_1, p^{<0>}, z^{<0>}, c_1^{<0>}, c_2^{<0>}, c_3^{<0>}$ );
    if ( $z^{<0>} > z^{\bullet}$ )  $p^{<0>} \leftarrow p^{\bullet}; z^{<0>} \leftarrow z^{\bullet}; c_1^{<0>} \leftarrow c_1^{\bullet}; c_2^{<0>} \leftarrow c_2^{\bullet}; c_3^{<0>} \leftarrow c_3^{\bullet}$  endif;
     $q^{<0>} \leftarrow q^{<0>} + 1;$ 
    if ( $q^{<0>} \neq Q^{<0>}$ ) // perturbation of the solution  $p^{<0>}$ 
       $p^{<0>} \leftarrow$  Mutation_Procedure( $p^{<0>}$ );
     $p^{<0>}, c_1^{<0>}, c_2^{<0>}, c_3^{<0>} \leftarrow$  Greedy_Adaptive_Search_Procedure( $B, n, m, m_1, p^{<0>}, c_1^{<0>}, c_2^{<0>}, c_3^{<0>}$ )
    endif
  while ( $q^{<0>} \neq Q^{<0>}$ )
end.
```

3.4.2. Tabu Search Algorithm

The 0-level HITS algorithm (the ITS algorithm) uses a self-contained (“kernel”) tabu search (TS) procedure (a more detailed discussion of the principles of TS algorithms can be found in [18]).

It is this procedure that is wholly responsible for the improvement of a given solution. At the same time, this procedure is in the role of the intensification of the search process, while the perturbation process rather serves as the search diversification mechanism. Briefly speaking, the TS procedure analyses the neighbourhood of the incumbent solution and accepts the non-tabu (non-prohibited) move that most improves the objective function. In order to avoid cycling search trajectories, the return to recently visited solutions is forbidden for some time period. The tabu list (list of prohibitions), T , is operationalized as a matrix of size $n \times n$. In this case, the entry t_{ij} memorizes the sum of the current iteration number and the tabu tenure, h . The value of h depends on the problem size, n (we have chosen $h = 0.5n$).

It should be noted that the tabu status is disregarded at accident moments with a very small probability α ($\alpha < 0.1$). This strategy enables to increase the number of non-prohibited solutions and not to confine the search directions too much. The tabu status is also ignored if an aspiration criterion is satisfied, i.e., the currently attained solution appears better than the best obtained solution.

Moreover, our TS algorithm utilizes an additional memory known as a secondary memory, SM . The goal of this memory is to archive high-quality solutions, which although are evaluated as quite good, but are not elected. Thus, if the best solution stays unchanged for more than $idle_iter_limit = \gamma\tau$ iterations, then the tabu list is wiped out and the search is reset to the one of the “second” solutions in SM (here, τ denotes the number of iterations of the TS algorithm, and γ is a futile (idle) iterations limit factor in such a way that $1 \leq \gamma\tau \leq \tau$).

The overall TS process is finished as soon as the maximum number of TS iterations, τ , have been executed. The pseudo-code of the tabu search algorithm is shown in Algorithm 6. After finishing the tabu search procedure, the obtained solution is subjected to perturbation, which is described in the next section.

Algorithm 6 Tabu_Search

```
// pseudocode of the tabu search algorithm
// input:  $B, n, m, m_1, p, z, c_1, c_2, c_3$ 
// output:  $p^\bullet, z^\bullet, c_1^\bullet, c_2^\bullet, c_3^\bullet$ 
// parameters:  $\tau, h, \alpha, \beta, idle\_iter\_limit$ 
begin
  clear tabu list  $TabuList$ ; clear hash table  $HashTable$ ;
   $p^\bullet \leftarrow p; z^\bullet \leftarrow z; c_1^\bullet \leftarrow c_1; c_2^\bullet \leftarrow c_2; c_3^\bullet \leftarrow c_3; q \leftarrow 0; q' \leftarrow 0; SM\_Size \leftarrow 0;$ 
  do
     $\Delta'_{min} \leftarrow \infty; \Delta''_{min} \leftarrow \infty;$ 
     $\Delta'_{min}, \Delta''_{min}, v', v'', w', w'', flag', flag'' \leftarrow \text{Neighbourhood\_Search}(0, m_1, m, n, q, p, z, z^\bullet, c_1, "1");$ 
     $\Delta'_{min}, \Delta''_{min}, v', v'', w', w'', flag', flag'' \leftarrow \text{Neighbourhood\_Search}(m_1, m, m, n, q, p, z, z^\bullet, c_2, "2");$ 
     $\Delta'_{min}, \Delta''_{min}, v', v'', w', w'', flag', flag'' \leftarrow \text{Neighbourhood\_Search}(0, m_1, m_1, m, q, p, z, z^\bullet, c_3, "3");$ 
    if ( $\Delta''_{min} \neq \infty$ )  $SM(SM\_Size) \leftarrow p, z + \Delta''_{min}, c_1, c_2, c_3, v'', w'', flag''; SM\_Size \leftarrow SM\_Size + 1$ 
  endif;
  if ( $\Delta'_{min} \neq \infty$ )
     $p \leftarrow \phi(p, v', w');$  recalculate  $c_1, c_2, c_3$  depending on  $v', w', flag'$ ;
     $TabuList(p(v'), p(w')) \leftarrow q + h; TabuList(p(w'), p(v')) \leftarrow q + h;$ 
     $z \leftarrow z + \Delta'_{min}; HashTable(z) \leftarrow \text{FALSE};$ 
    if ( $z^\bullet > z$ )  $p^\bullet \leftarrow p; z^\bullet \leftarrow z; c_1^\bullet \leftarrow c_1; c_2^\bullet \leftarrow c_2; c_3^\bullet \leftarrow c_3; q' \leftarrow q$  endif
  endif;
```

```

if  $(q - q' > \text{idle\_iter\_limit} \wedge q < \tau - \text{idle\_iter\_limit})$ 
  clear tabu list TabuList;
   $p, z, c_1, c_2, c_3, v'', w'', \text{flag}'' \leftarrow \text{SM}(\text{Random}(\beta \times \text{SM\_Size}, \text{SM\_Size}));$ 
   $p \leftarrow \phi(p, v'', w'');$  recalculate  $c_1, c_2, c_3$  depending on  $v'', w'', \text{flag}''$ ;
   $\text{TabuList}(p(v''), p(w'')) \leftarrow q + h$ ;  $\text{TabuList}(p(w''), p(v'')) \leftarrow q + h$ ;
   $\text{HashTable}(z) \leftarrow \text{FALSE}$ ;
   $q' \leftarrow q$ 
endif;
 $q \leftarrow q + 1$ 
while  $(q \neq \tau)$ 
end.

```

Notes. 1. τ denotes the total number of iterations. 2. h denotes the tabu tenure (prohibition period). 3. *idle_iter_limit* denotes the limit of idle search iterations (we used $\text{idle_iter_limit} = \max\{3, 0.2\tau\}$). 4. *SM* denotes the secondary memory for archiving “second” solutions, i.e., solutions that are left second after the evaluation of the neighbourhood Θ_1 ; *SM_Size* denotes the size of the secondary memory. 5. α denotes a randomization coefficient (we used $\alpha = 0.02$), and β is a random access parameter (we used $\beta = 0.8$). 6. The pseudocode of the neighbourhood search procedure (**Neighbourhood_Search**) is presented in Algorithm 7. 7. The function **Random**(x_1, x_2) returns a random number in the interval $[x_1, x_2)$.

Algorithm 7 Neighbourhood_Search

```

// pseudocode of the neighbourhood search sub-procedure
// input: neighbourhood_index1, neighbourhood_index2
//       neighbourhood_index3, neighbourhood_index4
//       TabuList, HashTable, q, p, z, z*, c, flag ( $\text{flag} = \text{"1"|"2"|"3"}$ )
// output:  $\Delta'_{min}, \Delta''_{min}, v', v'', w', w'', \text{flag}', \text{flag}''$ 
// parameters:  $\alpha$ 
begin
   $\text{flag}' \leftarrow \text{"empty"}$ ;  $i \leftarrow \text{neighbourhood\_index1}$ ;
  do
     $j \leftarrow \text{neighbourhood\_index3}$ ;
    do
       $\Delta \leftarrow 2 \times (c(p(j)) - c(p(i)) + 2 \times b_{p(i), p(j)})$ ;
      if  $((\Delta < \Delta'_{min} \wedge \text{HashTable}(z + \Delta) \wedge (\text{TabuList}(p(i), p(j)) < q \vee \text{Random}() < \alpha)) \vee z + \Delta < z^*)$ 
        if  $(\Delta < \Delta'_{min})$ 
           $\Delta''_{min} \leftarrow \Delta'_{min}$ ;  $v'' \leftarrow v'$ ;  $w'' \leftarrow w'$ ;  $\text{flag}'' \leftarrow \text{flag}'$ ;  $\Delta'_{min} \leftarrow \Delta$ ;  $v' \leftarrow i$ ;  $w' \leftarrow j$ ;  $\text{flag}' \leftarrow \text{flag}$ ;
        elseif  $(\Delta < \Delta''_{min})$ 
           $\Delta''_{min} \leftarrow \Delta$ ;  $v'' \leftarrow i$ ;  $w'' \leftarrow j$ ;  $\text{flag}'' \leftarrow \text{flag}$ ;
        endif
      endif
       $j \leftarrow j + 1$ 
    while  $(j \neq \text{neighbourhood\_index4})$ 
     $i \leftarrow i + 1$ 
  while  $(i \neq \text{neighbourhood\_index2})$ 
end.

```

Note. The function **Random**() returns a random number in the interval $[0, 1)$.

3.4.3. Perturbation Process

The perturbation procedure contains only two steps: (1) mutation (random shuffling) and (2) re-construction (recreation) of mutated solution.

Let p^\sim be the candidate solution to the mutation process. Then, after shuffling the elements $p^\sim(0), \dots, p^\sim(m_1 - 1), p^\sim(m_1), \dots, p^\sim(m - 1)$, the elements $p^\sim(m_1 - r_1), \dots, p^\sim(m_1 - 1), p^\sim(m - r_2), \dots, p^\sim(m - 1)$ are abandoned (no action is required). In our algorithm, $r_1 = \max\{1, \zeta m_1\}$, $r_2 = \max\{1, \zeta m_2\}$. The parameter ζ ($0 < \zeta \leq 1$) is

called a mutation rate coefficient. The pseudocode of the mutation procedure is given in Algorithm 8.

Algorithm 8 Mutation_Procedure

```
// pseudocode of the mutation procedure
// input:  $m, m_1, p$ 
// output:  $p^\sim$ 
begin
   $p^\sim \leftarrow p$ ;
   $i \leftarrow 0$ ; while ( $i \neq m_1 - 1$ )  $p^\sim \leftarrow \phi(p^\sim, i, \text{Random}(i, m_1))$ ;  $i \leftarrow i + 1$  endwhile;
   $i \leftarrow m_1$ ; while ( $i \neq m - 1$ )  $p^\sim \leftarrow \phi(p^\sim, i, \text{Random}(i, m))$ ;  $i \leftarrow i + 1$  endwhile
end.
```

The re-construction is performed by a so-called greedy adaptive search procedure (GASP) (see Algorithm 9). It is deterministic and also adaptive since it respects the selected solution elements. At every step of the first phase of procedure, $step$ ($step = 0, \dots, r_1 - 1$), an element ($j_{min} \in \{p(m_1 - r_1 + step), \dots, p(m_1 - 1)\} \cup \{p(m - r_2), \dots, p(n - 1)\}$) is picked up, one at a time, so that the sum $\sum_{i=0}^{m_1 - r_1 + step - 1} c_1(p(i)) + c_1(j_{min})$ is minimized (see Algorithm 10). After adding the element, the contributions are recomputed accordingly. The second phase is executed in a similar fashion (see Algorithm 10). All the calculations take approximately $O(\xi mn)$ time. This results in a quite fast execution of GASP when the value of ξ is not large.

Algorithm 9 Greedy_Adaptive_Search_Procedure

```
// pseudocode of the greedy adaptive search procedure
// input:  $B, n, m, m_1, p, c_1, c_2, c_3$ 
// output:  $p, c_1, c_2, c_3$ 
// parameters:  $\xi$ 
begin
   $i \leftarrow 0$ ;  $r_1 = \max\{1, \xi m_1\}$ ;  $r_2 = \max\{1, \xi m_2\}$ ;
  do
     $j \leftarrow m_1 - l_1$ ;  $sum_1 \leftarrow 0$ ; while ( $j \neq m_1$ )  $sum_1 \leftarrow sum_1 + b_{i,p(j)}$ ;  $j \leftarrow j + 1$  endwhile;
     $j \leftarrow m - l_2$ ;  $sum_2 \leftarrow 0$ ; while ( $j \neq m$ )  $sum_2 \leftarrow sum_2 + b_{i,p(j)}$ ;  $j \leftarrow j + 1$  endwhile;
     $c_1(i) \leftarrow c_1(i) - 2 \times sum_1 - sum_2$ ;  $c_2(i) \leftarrow c_2(i) - sum_1 - 2 \times sum_2$ ;
   $c_3(i) \leftarrow c_3(i) - sum_1 + sum_2$ ;
   $i \leftarrow i + 1$ 
while ( $i \neq n$ );
   $z \leftarrow 0$ ;  $i \leftarrow 0$ ; while ( $i \neq m_1 - r_1$ )  $z \leftarrow z + c_1(p(i))$ ;  $i \leftarrow i + 1$  endwhile;
   $i \leftarrow m_1$ ; while ( $i \neq m - r_2$ )  $z \leftarrow z + c_2(p(i))$ ;  $i \leftarrow i + 1$  endwhile;
   $i \leftarrow m_1 - r_1$ ;
while ( $i \neq m_1$ )
     $p, z, c_1, c_2, c_3 \leftarrow \text{Greedy\_Solution\_Construction}(B, n, m, m_1, i, c_1)$ ;  $i \leftarrow i + 1$ 
  endwhile;
   $i \leftarrow m - r_2$ ;
while ( $i \neq m$ )
     $p, z, c_1, c_2, c_3 \leftarrow \text{Greedy\_Solution\_Construction}(B, n, m, m_1, i, c_2)$ ;  $i \leftarrow i + 1$ 
  endwhile;
   $p \leftarrow \text{Filling\_Remaining\_Elements}(p)$ 
end.
```

Algorithm 10 Greedy_Solution_Construction

```

// pseudocode of the greedy solution construction sub-procedure
// input:  $B, n, m, m_1, i, this\_c, c_1, c_2, c_3$ 
// output:  $p, z, c_1, c_2, c_3$ 
begin
   $minimum\_contribution \leftarrow \infty; j \leftarrow 0;$ 
  do
    if (element  $j$  is not yet selected)
      if ( $this\_c(j) < minimum\_contribution$ )  $minimum\_contribution \leftarrow this\_c(j); j_{min} \leftarrow j$  endif;
       $j \leftarrow j + 1$ 
    while ( $j \neq n$ );
     $p(i) \leftarrow j_{min};$ 
    if ( $this\_c = c_1$ )
       $j \leftarrow 0;$ 
      do
         $c_1(j) \leftarrow c_1(j) + 2 \times b_{i,j_{min}}; c_2(j) \leftarrow c_2(j) + b_{i,j_{min}}; c_3(j) \leftarrow c_3(j) + b_{i,j_{min}}; j \leftarrow j + 1$ 
      while ( $j \neq n$ );
    elseif ( $this\_c = c_2$ )
       $j \leftarrow 0;$ 
      do
         $c_1(j) \leftarrow c_1(j) + b_{i,j_{min}}; c_2(j) \leftarrow c_2(j) + 2 \times b_{i,j_{min}}; c_3(j) \leftarrow c_3(j) - b_{i,j_{min}}; j \leftarrow j + 1$ 
      while ( $j \neq n$ )
    endif;
     $z \leftarrow z + minimum\_contribution$ 
  end.

```

3.5. Population Replacement

For the replacement of a population, we apply a modified replacement rule to respect not only the quality of the solutions, but also the distances (differences) between solutions. In particular, we have implemented an enhanced version of the “ $\mu + 1$ ” update rule. The idea is to preserve the minimum distance (DT) between population members. So, if the newly created offspring violates the minimum distance criterion, then it is simply omitted. The exception is the situation where the produced offspring is better than the best population individual. Note that the offspring is included into the population only if it is better than the worst population individual.

4. Computational Experiments

The hybrid genetic algorithm is coded by using C# programming language. For converting the analytical solutions to graphical images, however, Java programming language is used. The computational experiments have been conducted using an $\times 86$ series personal computer equipped with an Intel (Intel Corp., Santa Clara, CA, USA) 2 GHz 4 cores processor, 8 GB RAM and the 64-bit MS Windows operating system.

We experimented with the distance matrix B of size 64 ($n = 64$), which is available at the public electronic library of the benchmark data instances of the QAP—QAPLIB (<http://www.seas.upenn.edu/qaplib> (accessed on 30 June 2021), also see [19]). The grid size is 8×8 ($n_1 = n_2 = 8$). The value of the parameter m varies between 2 and $n - 1$, while the parameter m_1 varies between 1 and $m - 1$ (in fact, $\frac{m}{2}$ is enough since solutions with $\frac{m}{2} + 1, \frac{m}{2} + 2, \dots$ can be obtained by symmetry from solutions with $\frac{m}{2} - 1, \frac{m}{2} - 2, \dots$). We recall that $m_2 = m - m_1, m_3 = n - m$, where m_1, m_2, m_3 denote the numbers of dots (squares) of first, second and third color, respectively.

As main algorithm performance criteria, we adopt the success rate (SR) of the algorithm and the average deviation (AD) of the objective function. The success rate is calculated by the following formula: $SR = \frac{N_{bkv}}{R} \times 100[\%]$, where N_{bkv} is the total number of the best-known solutions (BKSs) over $R = 10$ independent runs of the algorithm. The average deviation is derived from this formula: $AD = \frac{\bar{z} - BKV}{BKV} \times 100[\%]$, where \bar{z} is the

average objective function value over R runs of the algorithm and BKV denotes the best-known value of the objective function.

At every run, the algorithm is applied to the given particular values of n , m and m_1 . Each time, the genetic algorithm starts from a new, distinct random initial population. The current run of the algorithm is terminated if the total number of generations, G , has been reached.

The particular values of the control parameters of the genetic algorithm are shown in Table 1.

Table 1. Values of the control parameters of the hybrid genetic algorithm used in the experiments.

Parameter	Value	Remarks
Population size, PS	20	
Number of generations, G	50	
Distance threshold, DT	$\max\{2, \lfloor 0.1m \rfloor\}$	$0 < DT \leq m$
Idle generations limit, $idle_gen_limit$	$\max\{3, \lfloor 0.05G \rfloor\}$	$0 < idle_gen_limit \leq G$
Number of iterations of hierarchical iterated tabu search, Q_{hier}	100	$Q_{hier} = Q^{(0)} \times Q^{(1)} \times Q^{(2)} \dagger$
Number of iterations of tabu search, τ	30	
Idle iterations limit, $idle_iter_limit$	$\max\{3, \lfloor 0.2\tau \rfloor\}$	$0 < idle_iter_limit \leq \tau$
Tabu tenure, h	$\lfloor 0.5n \rfloor$	$h > 0$
Randomization coefficient for tabu search, α	0.02	$0 \leq \alpha < 1$
Mutation rate factor, ξ	0.25	$0 < \xi \leq 1$
Number of runs of the algorithm, R	10	

$\dagger Q^{(0)} = 10, Q^{(1)} = 5, Q^{(2)} = 2.$

The results of the conducted experiments are presented in Table 2. All these results were achieved without fine-tuning of the parameters—only using the prescribed, default values of the control parameters of the genetic algorithm.

Table 2. Results of the computational experiments (part I).

n	m	m_1	BKV	BFV	AD [%]	SR [%]	t [sec]	n	m	m_1	BKV	BFV	AD [%]	SR [%]	t [sec]
64	2	1	6250	6250	0.000	100	0.773	64	16	8	4,020,000	4,020,000	0.000	100	4.016
64	3	1	37,500	37,500	0.000	100	1.434	64	17	1	6,544,448	6,544,448	0.000	100	3.678
64	4	1	93,750	93,750	0.000	100	1.524	64	17	2	6,180,016	6,180,016	0.000	100	3.845
64	4	2	75,000	750,00	0.000	100	1.431	64	17	3	5,837,720	5,837,720	0.000	100	3.617
64	5	1	225,000	225,000	0.000	100	1.908	64	17	4	5,498,552	5,498,552	0.000	100	3.925
64	5	2	207,384	207,384	0.000	100	1.900	64	17	5	5,288,324	5,288,324	0.000	100	3.886
64	6	1	423,248	423,248	0.000	100	2.014	64	17	6	5,120,416	5,120,416	0.000	100	3.782
64	6	2	337,500	337,500	0.000	100	1.862	64	17	7	4,994,172	4,994,172	0.000	100	3.788
64	6	3	341,022	341,022	0.000	100	2.101	64	17	8	4,926,672	4,926,672	0.000	100	3.949
64	7	1	662,500	662,500	0.000	100	2.180	64	18	1	7,743,646	7,743,646	0.000	100	3.797
64	7	2	559,448	559,448	0.000	100	2.166	64	18	2	7,161,396	7,161,396	0.000	100	4.036
64	7	3	487,500	487,500	0.000	100	2.217	64	18	3	6,812,246	6,812,246	0.000	100	3.884
64	8	1	918,750	918,750	0.000	100	2.327	64	18	4	6,452,128	6,452,128	0.000	100	3.992
64	8	2	800,000	800,000	0.000	100	2.326	64	18	5	6,217,116	6,217,116	0.000	100	4.062
64	8	3	718,750	718,750	0.000	100	2.410	64	18	6	5,991,716	5,991,716	0.000	100	3.777
64	8	4	650,000	650,000	0.000	100	2.484	64	18	7	5,854,952	5,854,952	0.000	100	4.039
64	9	1	1,290,000	1,290,000	0.000	100	2.888	64	18	8	5,768,624	5,768,624	0.000	100	3.867
64	9	2	1,154,188	1,154,188	0.000	100	3.344	64	18	9	5,761,814	5,761,814	0.000	100	4.099
64	9	3	1,057,730	1,057,730	0.000	100	3.801	64	19	1	8,958,414	8,958,414	0.000	100	3.654
64	9	4	976,980	976,980	0.000	100	3.676	64	19	2	8,394,606	8,394,606	0.000	100	4.358
64	10	1	1,748,280	1,748,280	0.000	100	3.104	64	19	3	7,815,844	7,815,844	0.000	100	4.482
64	10	2	1,542,500	1,542,500	0.000	100	3.241	64	19	4	7,481,512	7,481,512	0.000	100	4.727

Table 2. Cont.

<i>n</i>	<i>m</i>	<i>m</i> ₁	<i>BKV</i>	<i>BFV</i>	<i>AD</i> [%]	<i>SR</i> [%]	<i>t</i> [sec]	<i>n</i>	<i>m</i>	<i>m</i> ₁	<i>BKV</i>	<i>BFV</i>	<i>AD</i> [%]	<i>SR</i> [%]	<i>t</i> [sec]
64	10	3	1,416,954	1,416,954	0.000	100	3.234	64	19	5	7,198,882	7,198,882	0.000	100	4.057
64	10	4	1,304,924	1,304,924	0.000	100	3.011	64	19	6	6,961,460	6,961,460	0.000	100	3.868
64	10	5	1,297,932	1,297,932	0.000	100	3.693	64	19	7	6,767,378	6,767,378	0.000	100	4.052
64	11	1	2,225,738	2,225,738	0.000	100	3.782	64	19	8	6,629,500	6,629,500	0.000	100	4.235
64	11	2	2,023,044	2,023,044	0.000	100	3.759	64	19	9	6,578,410	6,578,410	0.000	100	4.808
64	11	3	1,832,500	1,832,500	0.000	100	3.400	64	20	1	10,226,448	10,226,448	0.000	100	4.246
64	11	4	1,708,508	1,708,508	0.000	100	3.465	64	20	2	9,639,732	9,639,732	0.000	100	4.222
64	11	5	1,666,674	1,666,674	0.000	100	3.452	64	20	3	9,055,990	9,055,990	0.000	100	4.664
64	12	1	2,772,894	2,772,894	0.000	100	3.382	64	20	4	8,482,792	8,482,792	0.000	100	5.065
64	12	2	2,530,300	2,530,300	0.000	100	3.440	64	20	5	8,242,654	8,242,654	0.000	100	4.332
64	12	3	2,332,450	2,332,450	0.000	100	3.627	64	20	6	7,947,380	7,947,380	0.000	100	4.206
64	12	4	2,135,000	2,135,000	0.000	100	3.541	64	20	7	7,733,348	7,733,348	0.000	100	4.096
64	12	5	2,094,542	2,094,542	0.000	100	3.160	64	20	8	7,541,536	7,541,536	0.000	100	4.441
64	12	6	2,058,808	2,058,808	0.000	100	3.282	64	20	9	7,452,098	7,452,098	0.000	100	4.763
64	13	1	3,391,510	3,391,510	0.000	100	3.191	64	20	10	7,419,404	7,419,404	0.000	100	4.697
64	13	2	3,117,606	3,117,606	0.000	100	3.213	64	21	1	11,537,304	11,537,304	0.000	100	4.605
64	13	3	2,883,878	2,883,878	0.000	100	3.350	64	21	2	10,944,262	10,944,262	0.000	100	4.390
64	13	4	2,682,128	2,682,128	0.000	100	3.367	64	21	3	10,359,224	10,359,224	0.000	100	4.364
64	13	5	2,575,000	2,575,000	0.000	100	3.354	64	21	4	9,791,688	9,791,688	0.000	100	5.261
64	13	6	2,535,680	2,535,680	0.000	100	3.292	64	21	5	9,287,240	9,287,240	0.000	100	5.036
64	14	1	4,073,442	4,073,442	0.000	100	3.354	64	21	6	9,030,702	9,030,702	0.000	100	5.125
64	14	2	3,767,164	3,767,164	0.000	100	3.334	64	21	7	8,795,182	8,795,182	0.000	100	5.358
64	14	3	3,504,318	3,504,318	0.000	100	3.387	64	21	8	8,571,828	8,571,828	0.000	100	5.686
64	14	4	3,248,604	3,248,604	0.000	100	3.364	64	21	9	8,442,646	8,442,646	0.000	100	5.311
64	14	5	3,131,906	3,131,906	0.000	100	3.471	64	21	10	8,349,142	8,349,142	0.000	100	5.732
64	14	6	3,027,500	3,027,500	0.000	100	3.493	64	22	1	12,922,754	12,922,754	0.000	100	5.163
64	14	7	3,020,756	3,020,756	0.000	100	3.650	64	22	2	12,252,800	12,252,800	0.000	100	5.106
64	15	1	4,803,016	4,803,016	0.000	100	3.453	64	22	3	11,671,958	11,671,958	0.000	100	5.248
64	15	2	4,467,502	4,467,502	0.000	100	3.363	64	22	4	11,113,084	11,113,084	0.000	100	4.511
64	15	3	4,167,234	4,167,234	0.000	100	3.623	64	22	5	10,602,386	10,602,386	0.000	100	5.285
64	15	4	3,909,108	3,909,108	0.000	100	3.658	64	22	6	10,104,188	10,104,188	0.000	100	5.329
64	15	5	3,746,844	3,746,844	0.000	100	3.689	64	22	7	9,872,662	9,872,662	0.000	100	5.306
64	15	6	3,609,580	3,609,580	0.000	100	3.621	64	22	8	9,618,052	9,618,052	0.000	100	5.380
64	15	7	3,517,500	3,517,500	0.000	100	3.679	64	22	9	9,483,714	9,483,714	0.000	100	5.865
64	16	1	5,568,750	5,568,750	0.000	100	3.711	64	22	10	9,344,104	9,344,104	0.000	100	6.292
64	16	2	5,210,000	5,210,000	0.000	100	3.786	64	22	11	9,342,556	9,342,556	0.000	100	5.886
64	16	3	4,888,750	4,888,750	0.000	100	4.207	64	23	1	14,411,076	14,411,076	0.000	100	5.790
64	16	4	4,580,000	4,580,000	0.000	100	4.013	64	23	2	13,718,236	13,718,236	0.000	100	5.500
64	16	5	4,408,750	4,408,750	0.000	100	4.390	64	23	3	13,051,236	13,051,236	0.000	100	5.568
64	16	6	4,250,000	4,250,000	0.000	100	4.241	64	23	4	12,471,980	12,471,980	0.000	100	6.146
64	16	7	4,128,750	4,128,750	0.000	100	3.981	64	23	5	11,942,532	11,942,532	0.000	100	5.976

In Table 2, we provide the following information: *n*—the total number of dots (squares), *m*, *m*₁, *BKV*—best known value of the objective function, *BFV*—best found objective function value, *AD*—average deviation of the objective function, *SR*—success rate, *t*—the average time (CPU time) per one run of the algorithm. Only the part of the results are shown for the sake of brevity. More results of the computational experiments (including the graphical images) will be available at <https://www.personalas.ktu.lt/~alfmise/>, accessed on 30 June 2021. Also, electronic copies of the additional results can be freely requested by contacting the authors.

The results in Table 2 demonstrate the astonishingly excellent performance (and reliability) of the proposed genetic algorithm from both the quality of solutions and the computational budget point of view. 100-percentage success rate was achieved in all 126 examined cases (scenarios). The cumulative average CPU time per run is equal to approximately 3.904 s.

The lasting, steady stability of the results and the almost linear character of the run time of the algorithm can be observed (see Table 2 and Figure 3).

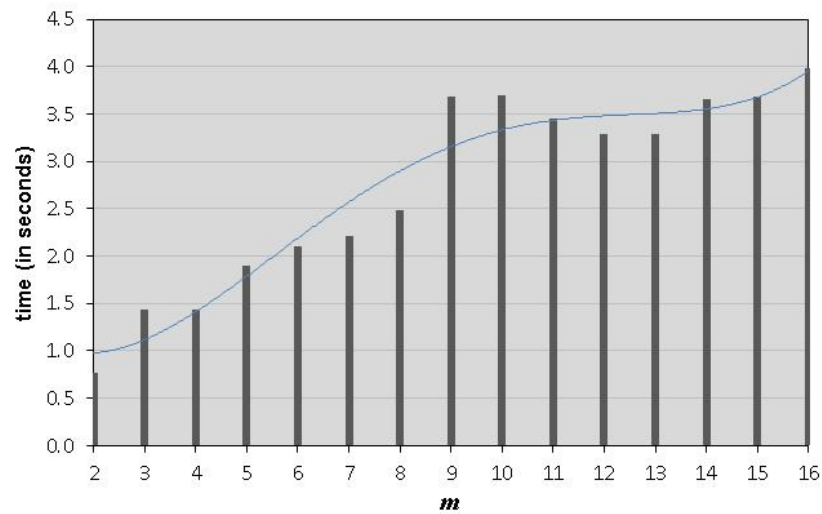


Figure 3. Illustration of run time behaviour of the hybrid genetic algorithm for different values of *m*.

In order to reveal the importance and usefulness of the hierarchical iterated tabu search used within the hybrid genetic algorithm, we carried out some additional experiments using the pure genetic algorithm without the HITS procedure (under equal other conditions). The obtained results are shown in Table 3. The results clearly demonstrate that: (i) the performance of the hybrid genetic algorithm heavily depends on the HITS procedure used; (ii) the dominance of hybrid GA over pure GA is strongly pronounced both in terms of the average deviation of the objective function and the success rate of the algorithm—thus our designed HITS procedure is very well suited for implementation in the hybrid genetic algorithm.

Table 3. Results of the computational experiments (part II).

<i>n</i>	<i>m</i>	<i>m</i> ₁	BKV	BFV	AD [%]	SR [%]	<i>t</i> [sec]	<i>n</i>	<i>m</i>	<i>m</i> ₁	BKV	BFV	AD [%]	SR [%]	<i>t</i> [sec]
64	2	1	6250	6250	2.800	90.000	0.001	64	16	8	4,020,000	4,486,464	14.254	0.000	0.001
64	3	1	37,500	37,500	11.063	10.000	0.000	64	17	1	6,544,448	6,992,588	8.553	0.000	0.001
64	4	1	93,750	93,750	14.326	10.000	0.000	64	17	2	6,180,016	6,584,496	7.851	0.000	0.001
64	4	2	75,000	80,528	29.246	0.000	0.000	64	17	3	5,837,720	6,220,054	8.219	0.000	0.001
64	5	1	225,000	237,240	12.263	0.000	0.000	64	17	4	5,498,552	5,857,500	8.062	0.000	0.001
64	5	2	207,384	210,254	6.547	0.000	0.000	64	17	5	5,288,324	5,608,208	8.586	0.000	0.001
64	6	1	423,248	441,200	6.896	0.000	0.000	64	17	6	5,120,416	5,403,878	7.521	0.000	0.001
64	6	2	337,500	364,444	12.436	0.000	0.000	64	17	7	4,994,172	5,308,534	7.945	0.000	0.001
64	6	3	341,022	350,208	5.958	0.000	0.000	64	17	8	4,926,672	5,144,976	7.986	0.000	0.001
64	7	1	662,500	675,148	4.839	0.000	0.000	64	18	1	7,743,646	7,979,294	5.802	0.000	0.001
64	7	2	559,448	593,832	9.039	0.000	0.000	64	18	2	7,161,396	7,560,614	7.821	0.000	0.001
64	7	3	487,500	530,464	13.574	0.000	0.000	64	18	3	6,812,246	7,132,094	7.490	0.000	0.001
64	8	1	918,750	1,003,102	10.586	0.000	0.000	64	18	4	6,452,128	6,808,512	7.302	0.000	0.001
64	8	2	800,000	871,472	11.057	0.000	0.000	64	18	5	6,217,116	6,502,736	6.601	0.000	0.001
64	8	3	718,750	799,078	13.877	0.000	0.000	64	18	6	5,991,716	6,295,004	7.190	0.000	0.001
64	8	4	650,000	757,616	19.520	0.000	0.000	64	18	7	5,854,952	6,120,370	6.599	0.000	0.001
64	9	1	1,290,000	1,358,246	8.438	0.000	0.000	64	18	8	5,768,624	6,079,670	6.638	0.000	0.001
64	9	2	1,154,188	1,207,806	8.683	0.000	0.000	64	18	9	5,761,814	5,964,462	5.299	0.000	0.001

Table 3. Cont.

<i>n</i>	<i>m</i>	<i>m</i> ₁	BKV	BFV	AD [%]	SR [%]	<i>t</i> [sec]	<i>n</i>	<i>m</i>	<i>m</i> ₁	BKV	BFV	AD [%]	SR [%]	<i>t</i> [sec]
64	9	3	1,057,730	1,104,352	8.168	0.000	0.000	64	19	1	8,958,414	9,156,896	4.063	0.000	0.001
64	9	4	976,980	1,044,920	9.989	0.000	0.000	64	19	2	8,394,606	8,607,328	5.324	0.000	0.001
64	10	1	1,748,280	1,788,460	4.324	0.000	0.000	64	19	3	7,815,844	8,222,316	7.125	0.000	0.001
64	10	2	1,542,500	1,634,212	6.838	0.000	0.000	64	19	4	7,481,512	7,875,526	6.679	0.000	0.001
64	10	3	1,416,954	1,501,536	7.997	0.000	0.000	64	19	5	7,198,882	7,574,816	6.539	0.000	0.001
64	10	4	1,304,924	1,392,388	9.950	0.000	0.000	64	19	6	6,961,460	7,309,520	6.388	0.000	0.001
64	10	5	1,297,932	1,332,112	8.434	0.000	0.000	64	19	7	6,767,378	7,064,918	5.981	0.000	0.001
64	11	1	2,225,738	2,290,748	5.288	0.000	0.000	64	19	8	6,629,500	6,952,552	6.787	0.000	0.001
64	11	2	2,023,044	2,076,684	5.514	0.000	0.000	64	19	9	6,578,410	6,830,782	5.401	0.000	0.001
64	11	3	1,832,500	1,915,170	6.952	0.000	0.000	64	20	1	10,226,448	10,517,434	3.699	0.000	0.001
64	11	4	1,708,508	1,795,422	8.413	0.000	0.000	64	20	2	9,639,732	9,910,520	4.274	0.000	0.001
64	11	5	1,666,674	1,753,082	7.841	0.000	0.000	64	20	3	9,055,990	9,416,676	5.563	0.000	0.001
64	12	1	2,772,894	2,854,242	5.750	0.000	0.000	64	20	4	8,482,792	8,888,376	6.915	0.000	0.001
64	12	2	2,530,300	2,610,040	6.448	0.000	0.000	64	20	5	8,242,654	8,594,084	5.827	0.000	0.001
64	12	3	2,332,450	2,432,710	7.464	0.000	0.000	64	20	6	7,947,380	8,334,310	5.762	0.000	0.001
64	12	4	2,135,000	2,244,260	8.971	0.000	0.000	64	20	7	7,733,348	8,129,728	6.040	0.000	0.001
64	12	5	2,094,542	2,199,812	7.457	0.000	0.000	64	20	8	7,541,536	7,959,008	6.239	0.000	0.001
64	12	6	2,058,808	2,185,722	9.072	0.000	0.000	64	20	9	7,4520,98	7,695,586	5.576	0.000	0.001
64	13	1	3,391,510	3,516,412	5.260	0.000	0.000	64	20	10	7,419,404	7,681,794	5.535	0.000	0.001
64	13	2	3,117,606	3,247,566	6.789	0.000	0.000	64	21	1	11,537,304	11,779,998	3.440	0.000	0.001
64	13	3	2,883,878	3,054,264	8.015	0.000	0.000	64	21	2	10,944,262	11,233,234	3.737	0.000	0.001
64	13	4	2,682,128	2,883,948	9.364	0.000	0.000	64	21	3	10,359,224	10,691,492	4.562	0.000	0.001
64	13	5	2,575,000	2,763,746	9.017	0.000	0.000	64	21	4	9,791,688	10,301,198	5.978	0.000	0.001
64	13	6	2,535,680	2,645,876	7.006	0.000	0.001	64	21	5	9,287,240	9,813,644	6.822	0.000	0.001
64	14	1	4,073,442	4,124,748	6.906	0.000	0.001	64	21	6	9,030,702	9,414,508	5.731	0.000	0.001
64	14	2	3,767,164	3,938,116	6.487	0.000	0.001	64	21	7	8,795,182	9,193,420	5.686	0.000	0.001
64	14	3	3,504,318	3,739,058	8.121	0.000	0.001	64	21	8	8,571,828	8,927,104	5.878	0.000	0.001
64	14	4	3,248,604	3,460,336	9.205	0.000	0.001	64	21	9	8,442,646	8,770,372	5.575	0.000	0.001
64	14	5	3,131,906	3,357,136	8.918	0.000	0.001	64	21	10	8,349,142	8,770,990	5.904	0.000	0.001
64	14	6	3,027,500	3,218,330	8.926	0.000	0.001	64	22	1	12,922,754	13,137,430	2.686	0.000	0.001
64	14	7	3,020,756	3,155,456	8.318	0.000	0.001	64	22	2	12,252,800	12,637,734	3.935	0.000	0.001
64	15	1	4,803,016	4,998,654	7.116	0.000	0.001	64	22	3	11,671,958	11,983,748	4.157	0.000	0.001
64	15	2	4,467,502	4,630,860	7.929	0.000	0.001	64	22	4	11,113,084	11,539,630	5.017	0.000	0.001
64	15	3	4,167,234	4,451,480	8.768	0.000	0.001	64	22	5	10,602,386	10,990,008	4.841	0.000	0.001
64	15	4	3,909,108	4,226,464	10.262	0.000	0.001	64	22	6	10,104,188	10,547,594	6.131	0.000	0.001
64	15	5	3,746,844	3,960,880	8.556	0.000	0.001	64	22	7	9,872,662	10,275,716	5.915	0.000	0.001
64	15	6	3,609,580	3,919,934	9.932	0.000	0.001	64	22	8	9,618,052	10,004,572	5.390	0.000	0.001
64	15	7	3,517,500	3,697,084	9.693	0.000	0.001	64	22	9	9,483,714	9,902,300	5.608	0.000	0.001
64	16	1	5,568,750	6,023,910	9.766	0.000	0.001	64	22	10	9,344,104	9,782,968	5.666	0.000	0.001
64	16	2	5,210,000	5,656,588	9.616	0.000	0.001	64	22	11	9,342,556	9,675,646	4.871	0.000	0.001
64	16	3	4,888,750	5,203,500	9.443	0.000	0.001	64	23	1	14,411,076	14,606,470	2.797	0.000	0.001
64	16	4	4,580,000	5,013,788	10.901	0.000	0.001	64	23	2	13,718,236	14,031,986	3.435	0.000	0.001
64	16	5	4,408,750	4,827,872	10.779	0.000	0.001	64	23	3	13,051,236	13,353,810	3.397	0.000	0.001
64	16	6	4,250,000	4,540,838	10.744	0.000	0.001	64	23	4	12,471,980	12,812,272	4.138	0.000	0.001
64	16	7	4,128,750	4,575,034	12.484	0.000	0.001	64	23	5	11,942,532	12,371,092	4.570	0.000	0.001

The analytical solutions can be quite easily transformed to their “visual counter-parts” and it is for the readers to judge about the quality of different generated color frames (see Figures 4 and 5).

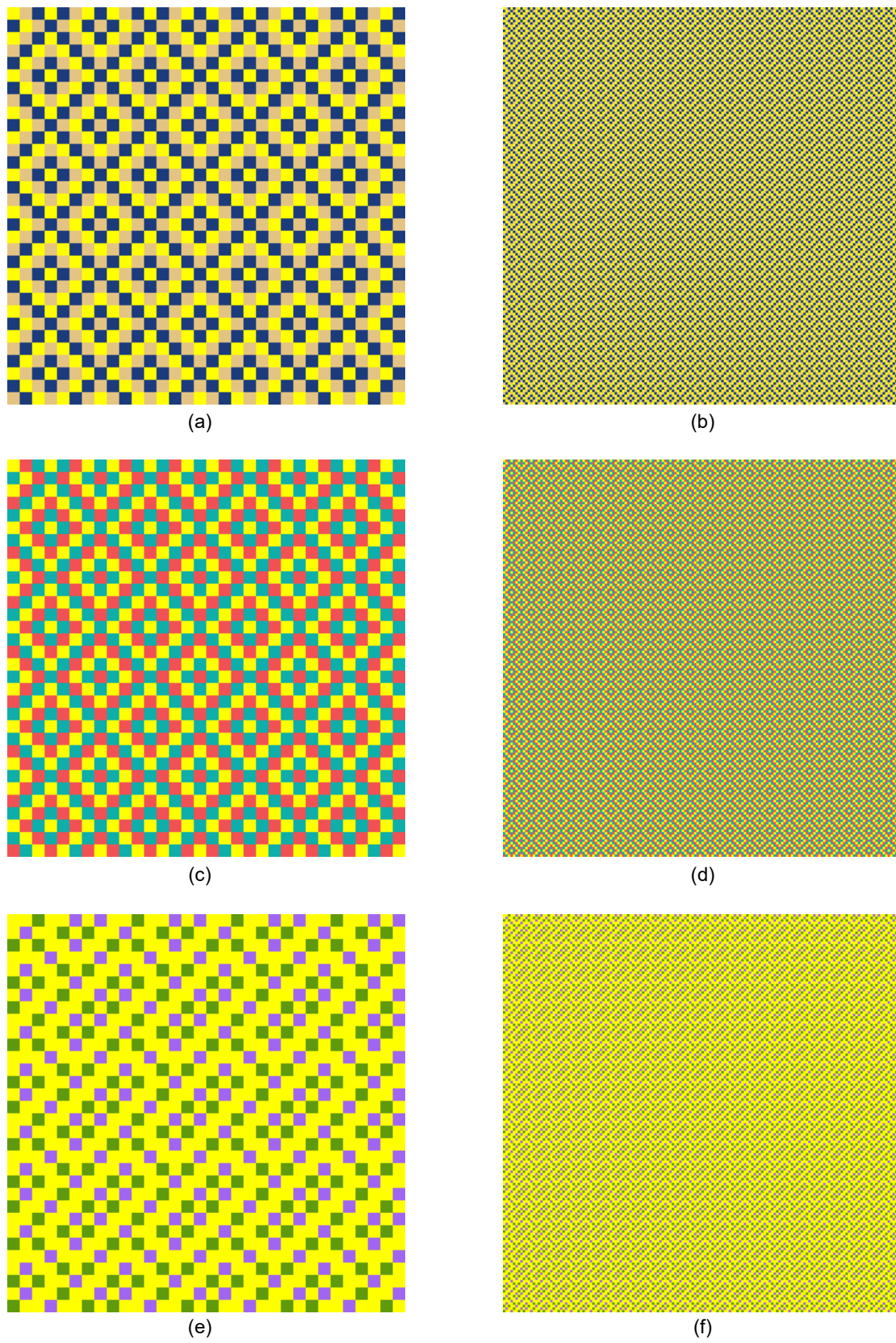
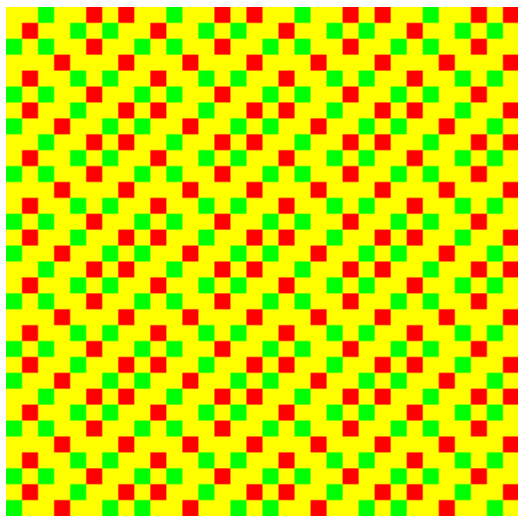
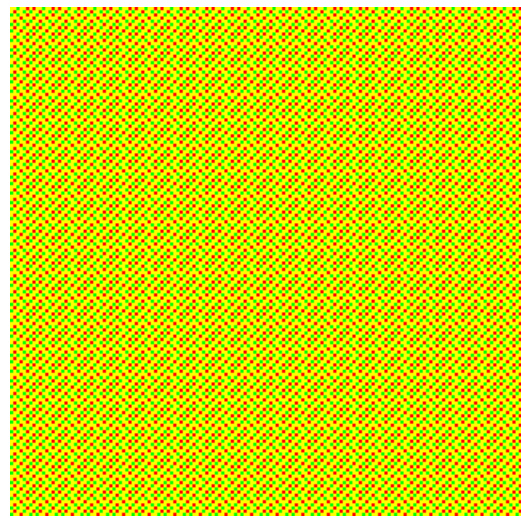


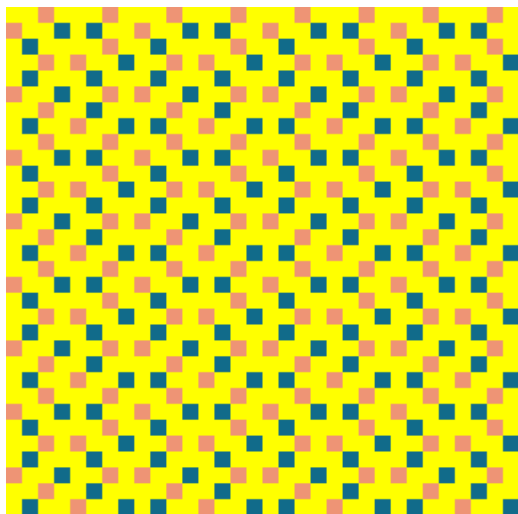
Figure 4. Illustrations of color patterns: (a) pattern-1, (b) pattern-2, (c) pattern-3, (d) pattern-4, (e) pattern-5, (f) pattern-6.



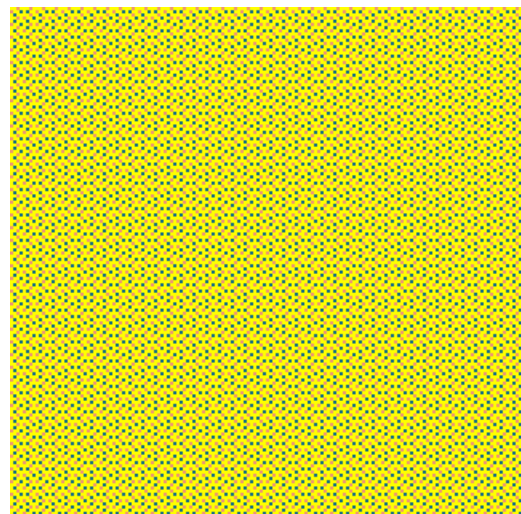
(a)



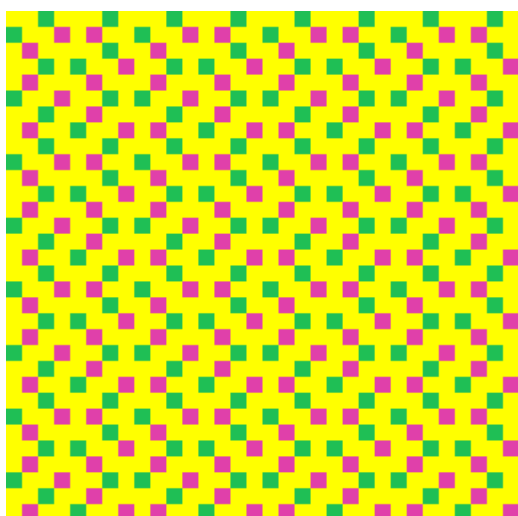
(b)



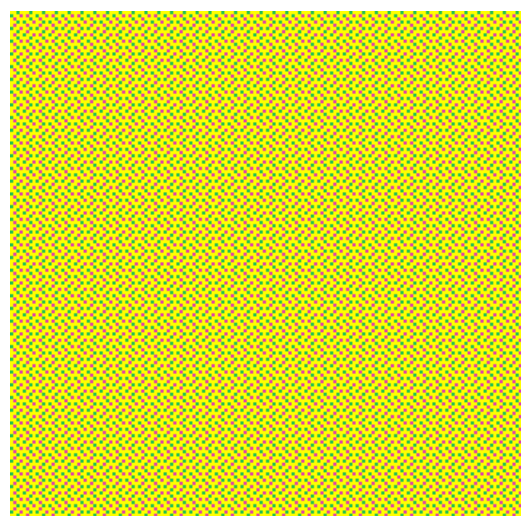
(c)



(d)



(e)



(f)

Figure 5. Illustrations of color patterns: (a) pattern-7, (b) pattern-8, (c) pattern-9, (d) pattern-10, (e) pattern-11, (f) pattern-12.

Every square (dot) of the depicted graphical images is consisting of $100 = 10 \times 10$ pixels in Figures 4a,c,e and 5a,c,e and $4 = 2 \times 2$ pixels in Figures 4b,d,f and 5b,d,f. $64 = 8 \times 8$ squares constitute a single “template” and each 64-square-template is replicated times horizontally and times vertically in patterns pattern-1, pattern-3, pattern-5, pattern-7, pattern-9, pattern-11 and 20 times horizontally and 20 times vertically in patterns pattern-2, pattern-4, pattern-6, pattern-8, pattern-10, pattern-12 (see Figures 4 and 5). Quantitative characteristics (specifications) for each color pattern are summarized in Table 4.

Table 4. Quantitative characteristics (specifications) of color patterns.

Color Pattern ID	m_1	m_2	m_3	Color ₁			Color ₂			Color ₃			# Pixels Per Square
				R	G	B	R	G	B	R	G	B	
pattern-1	21	22	21	E4	C4	82	1B	3B	7D	FF	FF	00	100
pattern-2	21	22	21	E4	C4	82	1B	3B	7D	FF	FF	00	4
pattern-3	21	22	21	F1	51	55	0E	AE	AA	FF	FF	00	100
pattern-4	21	22	21	F1	51	55	0E	AE	AA	FF	FF	00	4
pattern-5	11	12	41	A0	66	F1	5F	99	0E	FF	FF	00	100
pattern-6	11	12	41	A0	66	F1	5F	99	0E	FF	FF	00	4
pattern-7	11	12	41	FF	00	00	00	FF	00	FF	FF	00	100
pattern-8	11	12	41	FF	00	00	00	FF	00	FF	FF	00	4
pattern-9	10	10	44	11	6B	8A	EE	94	75	FF	FF	00	100
pattern-10	10	10	44	11	6B	8A	EE	94	75	FF	FF	00	4
pattern-11	10	10	44	E0	40	AA	1F	BF	55	FF	FF	00	100
pattern-12	10	10	44	E0	40	AA	1F	BF	55	FF	FF	00	4

Remarks: 1. $m_2 = m - m_1, m_3 = n - m; m_1, m_2, m_3$ denote the numbers of squares (dots) of colors $color_1, color_2, color_3$, respectively. 2. The values of the colors’ components R (red), G (green), B (blue) are presented in a hexadecimal form (in accordance with RGB color model) [20]. 3. The values of the components R, G, B of colors $color_1, color_2$ are generated randomly, while the third color $color_3$ is always yellow (FF FF 00).

5. Conclusions

In this paper, we have introduced a new combinatorial optimization problem entitled the color mix problem, which has practical potential applications in modern computer graphics, multimedia, as well as the visual arts.

We have also proposed the hybrid genetic-iterated tabu search algorithm for heuristically solving the color mix problem. The most important feature of the algorithm used is that the genetic algorithm operators are hybridized with the hierarchical iterated tabu search procedure, which, in turn, incorporates the efficient tabu search algorithm combined with mutations of solutions and fast greedy adaptive search procedure. This results in a smart hybridization scheme with effective optimizer of the offspring solutions.

The genetic algorithm was examined by using various data instances. The results from the experiments confirm the promising performance of the proposed algorithm for the solution of the color mix problem.

As to the future research direction, it is worthwhile to study the behaviour of the proposed algorithm on bigger data instances and also to adapt our algorithm for problems with larger numbers of colors.

Author Contributions: Conceptualization, A.M. and A.A.; methodology, A.M. and A.A.; software, A.A.; validation, A.M., A.A., A.O., T.B. and L.M.; formal analysis, A.M., A.A., A.O., T.B. and L.M.; investigation, A.M.; resources, A.M. and A.O.; data curation, A.M. and A.A.; writing—original draft preparation, A.M. and A.A.; writing—review and editing, A.M., A.A., A.O., T.B. and L.M.; visualization, A.A.; supervision, A.M.; project administration, A.M.; funding acquisition, A.M. and A.O. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Çela, E. *The Quadratic Assignment Problem: Theory and Algorithms*; Kluwer: Dordrecht, The Netherlands, 1998.
2. Taillard, E.D. Comparison of iterative searches for the quadratic assignment problem. *Locat. Sci.* **1995**, *3*, 87–105. [[CrossRef](#)]
3. Kang, H.R. *Digital Color Halftoning*; The SPIE Optical Engineering Press: Bellingham, WA, USA; IEEE Press: Piscataway, NJ, USA, 1999.
4. Kuznetsov, Y.V. *Principles of Image Printing Technology*; Springer: Cham, Switzerland, 2021.
5. Lau, D.L.; Arce, G.R. *Modern Digital Halftoning*; Marcel Dekker: New York, NY, USA; Basel, Switzerland, 2008.
6. Ulichney, R. *Digital Halftoning*; MIT Press: London, UK, 1987.
7. Wong, P.W.; Memon, N.D. Image processing for halftones. *IEEE Sig. Proc. Magaz.* **2003**, *20*, 59–70.
8. Martí, R.; Pardalos, P.M.; Resende, M.G.C. (Eds.) *Handbook of Heuristics*; Springer: Cham, Switzerland, 2018.
9. Drezner, Z. Finding a cluster of points and the grey pattern quadratic assignment problem. *OR Spectrum* **2006**, *28*, 417–436. [[CrossRef](#)]
10. 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*; Voß, S., Martello, S., Osman, I.H., Roucairol, C., Eds.; Kluwer: Boston, MA, USA, 1999; pp. 345–358.
11. Taillard, E.D.; Gambardella, L.M. *Adaptive Memories for the Quadratic Assignment Problem*; Technical Report IDSIA-87-97; CiteSeerX: Lugano, Switzerland, 1997.
12. Misevičius, A. Experiments with hybrid genetic algorithm for the grey pattern problem. *Informatica* **2006**, *17*, 237–258. [[CrossRef](#)]
13. Misevičius, A.; Stanevičienė, E. A new hybrid genetic algorithm for the grey pattern quadratic assignment problem. *Inf. Technol. Control* **2018**, *47*, 503–520. [[CrossRef](#)]
14. Misevičius, A.; Palubeckis, G.; Drezner, Z. Hierarchicality-based (self-similar) hybrid genetic algorithm for the grey pattern quadratic assignment problem. *Memet. Comput.* **2021**, *13*, 69–90. [[CrossRef](#)]
15. Misevičius, A.; Verenė, D. A hybrid genetic-hierarchical algorithm for the quadratic assignment problem. *Entropy* **2021**, *23*, 108. [[CrossRef](#)] [[PubMed](#)]
16. Goldberg, D.E. *Genetic Algorithms in Search, Optimization and Machine Learning*; Addison-Wesley: Reading, MA, USA, 1989.
17. Mehrdoost, Z.; Bahrainian, S.S. A multilevel tabu search algorithm for balanced partitioning of unstructured grids. *Int. J. Numer. Meth. Engng.* **2016**, *105*, 678–692. [[CrossRef](#)]
18. Glover, F.; Laguna, M. *Tabu Search*; Kluwer: Dordrecht, The Netherlands, 1997.
19. Burkard, R.E.; Karisch, S.; Rendl, F. QAPLIB—A quadratic assignment problem library. *J. Global Optim.* **1997**, *10*, 391–403. Available online: <http://anjos.mgi.polymtl.ca/qaplib/> (accessed on 30 June 2021).
20. Hexadecimal Color. Available online: <https://cgtweb1.tech.purdue.edu/courses/cgt215/141/Htmlhowto/hexcolor.html> (accessed on 30 June 2021).