mathematics

MDPI

Article

An Improved Hybrid Genetic-Hierarchical Algorithm for the Quadratic Assignment Problem

Alfonsas Misevičius *, Aleksandras Andrejevas, Armantas Ostreika, Dovilė Verenė and Gintarė Žekienė

Department of Multimedia Engineering, Kaunas University of Technology, Studentų st. 50, LT-51368 Kaunas, Lithuania; aleksandras.andrejevas@ktu.edu (A.A.); armantas.ostreika@ktu.lt (A.O.); dovile.verene@ktu.lt (D.V.); gintare.zekiene@ktu.lt (G.Ž.)

* Correspondence: alfonsas.misevicius@ktu.lt

Abstract: In this paper, an improved hybrid genetic-hierarchical algorithm for the solution of the quadratic assignment problem (QAP) is presented. The algorithm is based on the genetic search combined with the hierarchical (hierarchicity-based multi-level) iterated tabu search procedure. The following are two main scientific contributions of the paper: (i) the enhanced two-level hybrid primary (master)-secondary (slave) genetic algorithm is proposed; (ii) the augmented universalized multi-strategy perturbation (mutation process)—which is integrated within a multi-level hierarchical iterated tabu search algorithm—is implemented. The proposed scheme enables efficient balance between intensification and diversification in the search process. The computational experiments have been conducted using QAP instances of sizes up to 729. The results from the experiments with the improved algorithm demonstrate the outstanding performance of the new proposed approach. This is especially obvious for the small- and medium-sized instances. Nearly 90% of the runs resulted in (pseudo-)optimal solutions. Three new best-known solutions have been achieved for very hard, challenging QAP instances.

Keywords: combinatorial optimization; heuristic algorithms; genetic algorithms; tabu search; quadratic assignment problem

MSC: 68T20

Citation: Misevičius, A.; Andrejevas, A.; Ostreika, A.; Verenė, D.; Žekienė, G. An Improved Hybrid Genetic-Hierarchical Algorithm for the Quadratic Assignment Problem. *Mathematics* **2024**, *12*, 3726. https://doi.org/10.3390/math12233726

Academic Editor: Janez Žerovnik

Received: 24 September 2024
Revised: 20 November 2024
Accepted: 21 November 2024
Published: 27 November 2024

1. Introduction

The quadratic assignment problem (QAP) [1] can be stated as follows: Given two positive-integer quadratic matrices $\boldsymbol{A} = \left(a_{ij}\right)_{n\times n}$, $\boldsymbol{B} = \left(b_{ij}\right)_{n\times n}$, and a set Π_n of all possible permutations of the integers from 1 to n, find a permutation $p \in \Pi_n$ that minimizes the following function:

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

The quadratic assignment problem was firstly introduced by Koopmans and Beckmann [2] in 1957 but still attracts the attention of many researchers. There are many actual practical applications that can be formulated as quadratic assignment problems, including office assignment, planning a complex of buildings, design (construction) of electronic devices, finding the tightest cluster, the grey-pattern problem, the turbine-balancing problem, the arrangement of microarray layouts, configuration of an airport, scheduling parallel production lines, and assigning runners in a relay team. (A more detailed description of these particular applications can be found in [3].)

One of the concrete examples of the applications of the QAP is the placement of electronic components on printed circuit boards. In this case, the entries of the matrix $\boldsymbol{A}$ are associated with the number of connections between the pairs of components. Meanwhile, the entries of the matrix $\boldsymbol{B}$ correspond to the distances between the fixed positions on

the board. The permutation $p = (p(1), p(2), \ldots, p(n))$ can then be interpreted as a configuration for the arrangement of components in the positions, where the element $p(i)$ indicates the number of the position to which the i-th component is assigned. Then, z can be thought of as the total weighted length/cost of the connections between the components when all n components are placed into the corresponding n positions.

The QAP is also an important theoretical mathematical task. It has been proven that the QAP is NP-hard [4]. The quadratic assignment problem can be solved exactly by using the exact solution approaches (like branch and band algorithms [5], reformulation-linearization techniques [6,7], or semidefinite programming [8]), but only in the cases where the problem size is quite limited ($n < 30$). For this reason, the heuristic and metaheuristic algorithms (methods) are widely applied for the approximate solution of the QAP.

Historically, the local search (LS) (also known as neighbourhood search) principle-based algorithms were the first heuristic algorithms that were examined on the QAP in the early 1960s [9,10]. Later, several efficient improvements of the LS algorithms for the QAP were proposed (see, for example, [11–13]). It seems that the most efficient enhancement of LS algorithms is breakout local search [14,15].

The other neighbourhood search principle-based class of heuristics includes the simulated annealing [16] and tabu search algorithms [17–21]. In particular, the robust tabu search algorithm proposed by Taillard in 1991 [17] is still one of the most successful heuristic algorithms for the QAP in terms of efficacy, simplicity, and elegance.

The population-based heuristic algorithms include many algorithms that operate on sets of solutions, rather than single solutions. And this fact seems to be of crucial importance for the QAP. Within this category of algorithms, genetic algorithms have been shown to be among the most powerful heuristic algorithms for the QAP. This is especially true for the hybrid genetic (memetic) algorithms [22–33].

Distribution algorithms and differential evolution algorithms have been estimated, while population-based algorithms have been used to try to solve the QAP [34–36].

Several other types of heuristic algorithms (known as nature-/bio-inspired algorithms) have also been examined: the ant colony optimization [37], particle swarm optimization [38], artificial bee colony algorithm [39–41], hyper-heuristic method [42], electromagnetism-like optimization [43], cat swarm optimization [44], biogeography-based optimization [45], slime mould algorithm [46], firefly optimization algorithm [47,48], flower pollination algorithm [49], teaching–learning-based optimization [50], golden ball heuristic [51], chicken swarm optimization [52], optics-inspired optimization [53], antlion optimization algorithm [54,55], water-flow algorithm [56], crow search algorithm [57], artificial electric field algorithm [58], and Harris hawks optimization algorithm [59].

Also, some specific tailored heuristic approaches can be mentioned, e.g., [60–66].

The following are some of the actual articles related to the (meta)heuristic algorithms for the QAP: [27,28,32,57,59,67–69].

In [27], the authors suggest a new original principle for parent selection in genetic algorithms, which is inspired by natural biological evolutionary processes. The essence is that parents of one gender choose mates with certain characteristics favoured over others. This rule is quite simple, delicate, and can be utilized in many variants of genetic algorithms.

The authors of [28] propose the hybrid genetic algorithm that combines a so-called elite genetic algorithm and tabu search procedure. The algorithm employs two sorts of elite crossover procedures (two-exchange mutation and tabu search-based crossover), which help balance exploitation and exploration during the search process.

In [32], the researchers investigate a special hybrid heuristic algorithm, which blends three well-known metaheuristics, namely the scatter search, critical event tabu search, and random key-based genetic algorithm. This scheme results in a highly competitive algorithm, which is especially efficient for large-scale problems, providing many well-known solutions for such problems.

Then, [57] introduces the accelerated system for solving the QAP, which uses CUDA programming. A program is executed on both a CPU and a graphics processing unit (GPU). The task is divided between a CPU and a GPU. The CPU program calls the program that runs on a GPU, where many instructions are run simultaneously.

In [59], a so-called Harris hawk optimization algorithm is proposed, which is a nature (bio)-inspired heuristic optimization algorithm. Such algorithms are also known as metaphor-based since they use various metaphors, rather than the direct, definite names. For example, the Harris hawk optimization algorithm has its source of inspiration in the hunting behaviour of cooperative birds (hawks). The behaviour is imitated by adopting the corresponding mathematical formalism. In addition, the algorithm uses the classical tabu search procedure to enhance the performance of the algorithm.

One of the most recent and effective algorithms is the so-called frequent pattern-based search, which, in fact, is based on the breakout local search and, in addition, exploits the information of the frequent patterns (fragments) of the solutions [67].

In [68], a greedy genetic-hybridization approach is presented. In the proposed algorithm, an initial population is obtained using the best solution of the greedy algorithm. Standard genetic operators are then applied to the population members. The resulting solution is used as an initial assignment of the greedy algorithm. The algorithm is also applicable to the optimal network design.

We also mention [69], which investigates the heuristic evolutionary optimization algorithm. The algorithm is based on the data-mining ideas. The algorithm efficiently mines the itemsets composed of frequent items and also exploits the pseudo-utility characteristic. The algorithm balances local and global searches.

In addition, one can point out some nature-inspired algorithms (including mathematics-inspired algorithms), which, although not straightforwardly linked to the QAP, still possess certain potentialities to be applied to this problem by employing some transformation/discretization mechanisms (transfer functions) (see, for example, [70–75]).

For more thorough discussions on the heuristic algorithms for the QAP, the reader is referred to [76–78].

In this work, our main ambition is to further improve the performance of the genetic-hierarchical algorithm presented in [29] by further capitalizing on the principle of hierarchy and, at the same time, universalizing and enhancing the perturbation (mutation) mechanism. On top of that, we are trying to fight against the most severe drawbacks of the heuristics algorithms, namely the slow convergence speed and stagnation of the search processes. Overall, our genetic algorithm incorporates ideas heavily based on hierarchical principles. Hence, this algorithm is called a "hierarchical algorithm". (Notice that, generally, the concept of "hierarchy"—but within some other various backgrounds—is not fundamentally new and is considered in a few works (see, for example, [79–83]).)

The scientific contribution of the paper is twofold:

- The enhanced two-level (two-layer) hybrid primary (master)-secondary (slave) genetic algorithm is proposed, in particular, in the context of the quadratic assignment problem;
- The augmented universalized multi-strategy perturbation (mutation process)—which is integrated within a multi-level (*k*-level) hierarchical iterated tabu search algorithm (HITS)—is implemented.

In particular, in the first stage, the outstanding quality population (super population) is created by means of a cloned primary genetic algorithm—a secondary (slave) algorithm. Then, in the second phase, the created population is improved by a primary (master) genetic algorithm, which adopts the hierarchical iterated tabu search algorithm combined with the universal, versatile, and flexible perturbation procedure.

The paper is organized as follows. Section 2 covers the description of the improved hybrid genetic-hierarchical algorithm (IHGHA) and its components (parts). Section 3 presents the results of the computational experiments. Section 4 completes the paper with concluding comments.

2. An Improved Hybrid Genetic-Hierarchical Algorithm for the QAP

2.1. Preliminaries

Before describing the principle of functioning of our improved genetic-hierarchical algorithm, we provide some (basic) definitions for the sake of more clarity.

Let p ($p \in \Pi_n$) be a permutation and also let $p(v)$ ($v = 1, \ldots, n$) and $p(w)$ ($w = 1, \ldots, n$, $v \neq w$) be two elements in the permutation p. Then, we define $p^{v,w}$ as follows: $p^{v,w}(i) = \begin{cases} p(i), & i \neq v, w \\ p(v), & i = w \\ p(w), & i = v \end{cases}$; $i = 1, \ldots, n$. A two-exchange neighbourhood function Θ_2: $\Pi_n \to 2^{\Pi_n}$ assigns for each $p \in \Pi_n$ a set of neighbouring solutions $\Theta_2(p) \subseteq \Pi_n$. $\Theta_2(p)$ is defined in the following way: $\Theta_2(p) = \{p' : p' \in \Pi_n, \delta(p, p') = 2\}$, where $p \in \Pi_n$ and $\delta(p, p')$ denote the (Hamming) distance between the solutions p and p'.

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

Let p and $p^{v,w}$ be two neighbouring solutions. Then, the difference in the objective function values $\Delta_z(p, p^{v,w})$—which is obtained when the two elements $p(v)$ and $p(w)$ of the current permutation have been interchanged—is calculated according to this formula (also see [84]):

$$\begin{gathered} \Delta_z(p, p^{v,w}) = z(p^{v,w}) - z(p) = (a_{vv} - a_{ww})\left(b_{p(w)p(w)} - b_{p(v)p(v)}\right) + \\ (a_{vw} - a_{wv})\left(b_{p(w)p(v)} - b_{p(v)p(w)}\right) + \\ \sum_{k=1,k\neq v,w}^{n}\left[(a_{vk} - a_{wk})\left(b_{p(w)p(k)} - b_{p(v)p(k)}\right) + (a_{kv} - a_{kw})\left(b_{p(k)p(w)} - b_{p(k)p(v)}\right)\right]. \end{gathered} \tag{2}$$

Furthermore, one can use memory (RAM) to store the values $\Delta_z(p, p^{i,j})$ ($i, j = 1, \ldots, n$), as proposed by Frieze et al. (1989) [84] and Taillard (1991) [17]. Note that, after the exchange of the elements $p(v)$ and $p(w)$, the values $\Delta_z(p, p^{i,j})$ must be updated (new values $\Delta'_z(p, p^{i,j})$ are obtained) according to the following formula:

$$\begin{gathered} \Delta'_z(p, p^{i,j}) = \Delta_z(p, p^{i,j}) + \\ (a_{iv} - a_{iw} + a_{jw} - a_{jv})\left(b_{p(i)p(w)} - b_{p(i)p(v)} + b_{p(j)p(v)} - b_{p(j)p(w)}\right) + \\ (a_{vi} - a_{wi} + a_{wj} - a_{vj})\left(b_{p(w)p(i)} - b_{p(v)p(i)} + b_{p(v)p(j)} - b_{p(w)p(j)}\right). \end{gathered} \tag{3}$$

If $i = v$, or $i = w$, or $j = v$, or $j = w$, then the previous formula must be applied. Thus, the complexity of recalculating of all the values $\Delta_z(p, p^{i,j})$ is $O(n^2)$.

Moreover, if matrix $\boldsymbol{A}$ and/or matrix $\boldsymbol{B}$ are symmetric, then formulas become simpler. Assume that only matrix $\boldsymbol{B}$ is symmetric. Then, one can transform the asymmetric matrix $\boldsymbol{A}$ to the symmetric matrix $\boldsymbol{A}'$ by summing up the corresponding entries in $\boldsymbol{A}$, i.e., $a'_{ij} = a_{ij} + a_{ji}$, $i = 1, \ldots, n$, $j = 1, \ldots, n$, $i \neq j$, $a'_{ii} = a_{ii}$ [20,85]. The simplified formula is as follows:

$$\Delta_z(p, p^{v,w}) = \sum_{k=1,k\neq v,w}^{n}\left(a'_{vk} - a'_{wk}\right)\left(b_{p(w)p(k)} - b_{p(v)p(k)}\right). \tag{4}$$

In a similar way, Formula (3) also becomes simpler [20,85]:

$$\Delta'_z\left(p, p^{i,j}\right) = \Delta_z\left(p, p^{i,j}\right) + \left(a\prime_{iv} - a\prime_{iw} + a\prime_{jw} - a\prime_{jv}\right)\left(b_{p(i)p(w)} - b_{p(i)p(v)} + b_{p(j)p(v)} - b_{p(j)p(w)}\right). \quad (5)$$

If $i = v$, $i = w$, $j = v$, $j = w$, then Formula (4) must be applied.

An additional time-saving trick can be used. Suppose that we dispose of three-dimensional matrices $\boldsymbol{A}'' = (a''_{lmr})_{n\times n\times n}$ and $\boldsymbol{B}'' = (b''_{stu})_{n\times n\times n}$. Also, let $a''_{lmr} = a\prime_{lr} - a'_{mr}$, and let $b''_{stu} = b_{su} - b_{tu}$, l, m, r, s, t, $u = 1, \ldots, n$. As a result, we can apply the following effective formula for calculation of the difference in the values of the objective function (Δ_z) [20,29]:

$$\Delta_z(p, p^{v,w}) = \sum_{k=1,k\neq v,w}^{n} a''_{vwk} b''_{p(w)k(v)p(k)}. \quad (6)$$

Similarly, the formula for the recalculation of the difference in the objective function values, $\Delta_z\left(p, p^{i,j}\right)$, also becomes much more compact and faster [20,29]:

$$\Delta'_z\left(p, p^{i,j}\right) = \Delta_z\left(p, p^{i,j}\right) + \left(a''_{ijv} - a''_{ijw}\right)\left(b''_{p(i)p(j)p(w)} - b''_{p(i)p(j)p(v)}\right). \quad (7)$$

2.2. (General) Structure of the Algorithm

Essentially, our genetic algorithms consist of two main parts: the primary genetic algorithm (master genetic algorithm) and secondary genetic algorithm (slave genetic algorithm), which is de facto operationalized as a clone of the primary (master) algorithm. Each of them, in turn, incorporates the hierarchical iterated tabu search algorithm integrated with the perturbation (mutation) procedure. The principal idea of the used algorithm is similar to that of the hybrid genetic algorithm [29], where global explorative/diversified search is in cooperation with the local exploitative/intensified search. The exploitative search, in particular, is performed by the iterated hierarchical tabu search procedure (see Section 2.7), and the diversification is ensured through the enhanced perturbation (mutation) mechanism (see Section 2.7.2).

In our genetic algorithm, the solution (permutation) elements $p(1), p(2), \ldots, p(n)$ are directly linked to the genes of individuals' chromosomes, so no encoding is required. Meanwhile, the objective function value z is associated with the fitness of individuals. The population of individuals, P, in our algorithm has a three-tuple list-like representation, i.e., it is organized as a structured list of size PS, where PS is the population size:

$$P = \begin{bmatrix} \left\langle p^{(1)}, z^{(1)}, \Xi^{(1)} \right\rangle \\ \vdots \\ \left\langle p^{(k)}, z^{(k)}, \Xi^{(k)} \right\rangle \\ \vdots \\ \left\langle p^{(PS)}, z^{(PS)}, \Xi^{(PS)} \right\rangle \end{bmatrix}_{PS}. \quad (8)$$

Every member of the population (i.e., the element of the list) appears as a triplet in the form $\left\langle p^{(k)}, z^{(k)}, \Xi^{(k)} \right\rangle$, where $k = 1, \ldots, PS$, $p^{(k)}$ ($p^{(k)} = \left(p^{(k)}(1), \ldots, p^{(k)}(n)\right)$) is the kth solution of the population, $z^{(k)}$ ($z^{(k)} = z^{(k)}\left(p^{(k)}\right)$) denotes the kth value of the objective function corresponding to $p^{(k)}$, and $\Xi^{(k)}$ is the kth matrix that contains the pre-computed differences in the objective function values (i.e., $\Xi^{(k)}(i,j) = \Delta_z\left(p^{(k)}, p^{(k)i,j}\right)$, $i = 1, \ldots, n-1$, $j = i+1, \ldots, n$). Storing differences $\Xi^{(k)}(i,j)$ directly in the RAM memory ensures faster execution of the genetic algorithm because there is no longer the need to re-compute the difference matrix from scratch every time. So, even if the construction of the initial

population occurs at $O(n^3)$ time, the resulting time complexity of the remaining genetic algorithm is nearly proportional to $O(n^2)$. (Notice that memory complexity is, in our algorithm, proportional to $O\left(C \times PS \times n^2\right)$, where C denotes the initial population size factor (coefficient), and PS is the population size (see Section 2.3). This fact raises no issues for modern computers.)

The basic components of the genetic algorithm are as follows: (1) construction of initial population; (2) parent selection; (3) crossover procedure; (4) improvement of the offspring by the hierarchical iterated tabu search algorithm; (5) population replacement.

The top-level pseudocode of the genetic algorithm is provided in Algorithm 1.

Remark. If there is no population change, then the current generation is recognized as an idle generation. And if the number of consecutive idle generations exceeds the predefined limit, L_{idle_gen}, then the genetic algorithm is restarted from a new population.

Algorithm 1 Top-level pseudocode of the hybrid genetic-hierarchical algorithm

```
Hybrid_Genetic_Hierarchical_Algorithm;
// input: n—problem size, A, B—data matrices
// output: p*—the best found solution
// parameters: PS—population size, G—total number of generations, DT—distance threshold, L_idle_gen—idle
generations limit,
//                CrossVar—crossover operator variant, PerturbVar—perturbation/mutation variant
begin
  create the initial population P of size PS;
  p* ← GetBestMember(P); // initialization of the best so far solution
  for i ← 1 to G do begin // main loop
    sort the members of the population P in the ascending order of the values of the
    objective function;
    select parents p', p'' ∈ P for crossover procedure;
    perform the crossover operator on the solution-parents p', p''
    and produce the offspring p''';
    apply improvement procedure Hierarchical_Iterated_Tabu_Search
    to the offspring p''', get the (improved) offspring p☆;
    if z(p☆) < z(p*) then p* ← p☆; // the best found solution is memorized
    if idle generations detected then restart from a new population
    else obtain new population P from the union of the existing parents'
         population and the offspring P ∪ {p☆} (such that |P| = PS)
  endfor;
  return p*
end.
```

Notes. 1. The subroutine GetBestMember$(\cdot)$ returns the best solution for the particular population. 2. The crossover operator is performed considering the crossover variant parameter *CrossVar*. 3. The hierarchical iterated tabu search procedure is executed with regard to the perturbation (mutation) variant parameter *PerturbVar*. 4. The population management is organized bearing in mind the distance threshold value DT.

2.3. Initial Population Creation

The genetic algorithm starts with a pre-initialization stage, i.e., the construction of the pre-initial ("primordial") population P_p of size $PS \times C$, where C ($C \geq 1$) is a predefined parameter (coefficient) that controls the size of the pre-initial population. Every solution of the "primordial" population is created by using a secondary/slave genetic algorithm (cloned master genetic algorithm) in such a way that the best-evolved solution of the slave

algorithm becomes the current incumbent solution in the initial population. The idea of doing so is simply to generate a superior-quality starting population.

The slave genetic algorithm uses, in turn, the greedy randomized adaptive search procedure (GRASP) [86] for the creation of the starting solutions. These solutions are created one at a time, such that every solution appears unique due to the random nature of GRASP. In addition, every solution created by GRASP is subjected to improvement by the hierarchical iterated tabu search algorithm. The process continues until $\left|P_p\right| = PS \times C$ solutions are constructed and improved.

After the pre-initial population is improved, the truncation (culling) of the obtained population is performed. In particular, $(C-1)PS$ worst members of the primordial population are discarded, and only PS members survive for future generations. This approach is similar to that proposed in [87,88].

There is a special trick. In particular, if the improved solution ($p^{\star}$) is better than the current best-found solution, then the improved solution replaces the best-found solution. Otherwise, it is checked if the minimum mutual distance ($\min\limits_{p \in P}\left\{\delta\left(p^{\star}, p\right)\right\}$) between the new solution $p^{\star}$ and the current population solutions is greater than the predetermined distance threshold, DT. If this is the case, then the new solution joins the population. Otherwise, the new solution is discarded and a random solution is included instead.

Remark. The distance threshold DT is connected to the size of the problem, n, through the following equation: $DT = \max\{2, \lfloor \vartheta n \rfloor\}$, where ϑ is the distance-threshold factor ($0 < \vartheta \leq 1$), which is up to the user's choice.

2.4. Parent Selection

In our genetic algorithm, the solutions' parents are selected using a rank-based selection rule [89].

2.5. Crossover Operator

In our algorithm, the crossover operator is applied to two selected parents to produce one offspring, which is subject to improvement by the HITS algorithm. The crossover operator occurs at every generation; that is, the crossover probability is equal to 1. We dispose of two types of permutation-oriented crossover operators: cohesive crossover [22] and universal crossover [90]. The first operator is problem-dependent and adopts problem-specific information, while the second one is of pure general nature. The parameter *CrossVar* is to decide which crossover operator should be used. The computational time complexity of the crossover operators is proportional to $O\left(n^2\right)$ and does not contribute much to the overall time budget of the genetic algorithm. More details on these crossover operators can be found in [22,90].

2.6. Population Replacement

We have, in particular, implemented a modified variant of the well-known "$\mu + \lambda$" update rule [91]. Our rule respects not only the quality of the solutions but also the distances between solutions. The idea is to preserve the minimum distance threshold, DT, between population members. The new replacement rule is formally denoted as "$\mu + \lambda$, ε", where $\varepsilon = DT$. (This rule is also used for the initial population construction (see Section 2.3)). So, if the minimum (mutual) distance between the newly obtained offspring and the individuals in the population is less than DT, then the offspring is omitted. (The exception is the situation where the offspring is better than the best individual population.) Otherwise, the offspring enters the current population, but only under the condition that it is better than the worst population member. The worst individual is removed in this case. Also, if the new offspring is better than the best population individual, then the offspring replaces, in particular, the best individual population. This replacement strategy ensures that only individuals who are diverse enough survive for further generations.

2.7. Improvement of Solutions—Hierarchical Iterated Tabu Search

Every produced offspring is subject to improvement by the hierarchical iterated tabu search, which can also be thought of as a multi-layered tabu search procedure where the basic idea is the cyclic (multiple) reuse of the tabu search algorithm. A similar idea of the multi-reuse of iterative/local searches was utilized in some other algorithms (like the hierarchical iterated local search [79], iterated tabu search [92], and multilevel tabu search [93]).

The k-level (k-layer) hierarchical iterated tabu search algorithm consists of three main ingredients: (1) call of the $(k-1)$-level hierarchical iterated tabu search procedure; (2) selection (acceptance) of the solution for perturbation; (3) perturbation of the selected solution. The "edge case" is $k = 0$, which means the self-contained and autonomous TS algorithm. In most cases, $k = 1$ or $k = 2$ are enough.

The perturbed solution serves as an input for the $(k-1)$-level TS procedure. This procedure returns 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 continues until a predefined number of iterations have been performed (see Algorithm 2). The best-found solution in the course of this process is regarded as the final solution of HITS. The resulting time complexity of HITS is proportional to $O(n^2)$, although the proportionality coefficient may be quite large. The overall complexity of the IHGHA algorithm can be formulated as $O(C \times PS \times n^3 + G \times Q \times \tau \times n^2)$, where $G \times Q \times \tau \gg C \times PS$; C is the initial population size factor, PS is the population size, G is the number of generations, Q is the number of iterations of hierarchical iterated tabu search, and τ is the number of iterations of the self-contained tabu search procedure. (The concrete values of C, PS, G, Q, and τ can be found in Table 1 in Section 3.1.)

Algorithm 2 Pseudocode of the multi-level (k-level) hierarchical iterated tabu search algorithm

```
Hierarchical_Iterated_Tabu_Search;
// input: p—current solution
// output: p*—the best found solution
// parameter: k—current level (k > 0), Q^⟨k⟩, Q^⟨k−1⟩, ..., Q^⟨0⟩—numbers of iterations
begin
  p* ← p;
  for q^⟨k⟩ ← 1 to Q^⟨k⟩ do begin
    apply k − 1-level hierarchical iterated tabu search algorithm to p and get p^∇;
    if z(p^∇) < z(p*) then p* ← p^∇; // the best found solution is memorized
    if q^⟨k⟩ < Q^⟨k⟩ then begin
      p ← Candidate_Acceptance(p^∇, p*);
      apply perturbation process to p
    endif
  endfor
end.
```

Note. The tabu search procedure (see Algorithm 3) is in the role of the 0-level (i.e., the self-contained) tabu search algorithm.

Algorithm 3 Pseudocode of the tabu search algorithm

Tabu_Search;
// input: n—problem size,
// p—current solution, Ξ—difference matrix
// output: $p^{\bullet}$—the best found solution (along with the corresponding difference matrix)
// parameters: τ—total number of tabu search iterations, h—tabu tenure, α—randomization coefficient,
// L_{idle_iter}—idle iterations limit, *HashSize*—maximum size of the hash table
begin
clear tabu list *TabuList* and hash table *HashTable*;
$p^{\bullet} \leftarrow p$; $q \leftarrow 1$; $q' \leftarrow 1$; *secondary_memory_index* $\leftarrow 0$; *improved* $\leftarrow$ FALSE;
while ($q \leq \tau$) **or** (*improved* = TRUE) **do begin** // main cycle
$\Delta'_{min} \leftarrow \infty$; $\Delta''_{min} \leftarrow \infty$; $v' \leftarrow 1$; $w' \leftarrow 1$;
for $i \leftarrow 1$ **to** $n-1$ **do**
for $j \leftarrow i+1$ **to** n **do begin** // $n(n-1)/2$ neighbours of p are scanned
$\Delta \leftarrow \Xi(i, j)$;
forbidden $\leftarrow$ **iif**(((*TabuList*$(i, j) \geq q$) **or** (*HashTable*$((z(p) + \Delta)$ **mod** *HashSize*) = TRUE) **and**
(random() $\geq \alpha$)), TRUE, FALSE);
aspired $\leftarrow$ **iif**$(z(p) + \Delta < z(p^{\bullet})$, TRUE, FALSE);
if $((\Delta < \Delta'_{min})$ **and** (*forbidden* = FALSE)) **or** (*aspired* = TRUE) **then begin**
if $\Delta < \Delta'_{min}$ **then begin** $\Delta''_{min} := \Delta'_{min}$; $v'' := v'$; $w'' := w'$; $\Delta'_{min} := \Delta$; $v' := i$; $w' := j$ **endif**
else if $\Delta < \Delta''_{min}$ **then begin** $\Delta''_{min} := \Delta$; $v'' := i$; $w'' := j$ **endif**
endif
endfor;
if $\Delta''_{min} < \infty$ **then begin** // archiving second solution, Ξ, v'', w''
secondary_memory_index $\leftarrow$ secondary_memory_index + 1; SM(secondary_memory_index) $\leftarrow$ p, Ξ, v″, w″
endif;
if $\Delta'_{min} < \infty$ **then begin** // replacement of the current solution and recalculation of the values of Ξ
$p \leftarrow \phi(p, v', w')$;
recalculate the values of the matrix Ξ;
if $z(p) < z(p^{\bullet})$ **then begin** $p^{\bullet} \leftarrow p$; $q' \leftarrow q$ **endif**; // the best so far solution is memorized
TabuList$(v', w') \leftarrow q + h$; // the elements $p(v')$, $p(w')$ become tabu
HashTable$((z(p) + \Delta'_{min})$ **mod** *HashSize*$) \leftarrow$ TRUE
endif;
improved $\leftarrow$ **iif**$(\Delta'_{min} < 0$, TRUE, FALSE);
if (*improved* = FALSE) **and** $(q - q' > L_{idle\_iter})$ **and** $(q < \tau - L_{idle\_iter})$ **then begin**
// retrieving solution from the secondary memory
random_access_index $\leftarrow$ random($\beta \times$ secondary_memory_index, secondary_memory_index);
p, Ξ, v″, w″ $\leftarrow SM$(random_access_index);
$p \leftarrow \phi(p, v'', w'')$;
recalculate the values of the matrix Ξ;
clear tabu list *TabuList*;
TabuList$(v'', w'') \leftarrow q + h$; // the elements $p(v'')$, $p(w'')$ become tabu
$q' \leftarrow q$
endif;
$q \leftarrow q + 1$
endwhile
end.

Notes. 1. The immediate 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 $[x_1, x_2]$. 4. The values of the matrix Ξ are recalculated according to the formula (7). 5. β denotes the random access parameter (we used $\beta = 0.8$).

2.7.1. Tabu Search Algorithm

The 1-level HITS algorithm (the ITS algorithm) uses a self-contained tabu search (TS) procedure. It is this procedure that is wholly responsible for the direct improvement of a particular solution and is in the role of the intensification of the search process. Briefly speaking, the TS procedure analyses the whole neighbourhood of the incumbent solution and accepts the non-tabu (i.e., non-prohibited) move that most improves (least degrades) the objective function. In order to avoid the cycling process, the return to the recently visited solutions is forbidden for some time (tenure).

Going into more detail, the tabu list (list of prohibitions), T, is operationalized as a matrix of size $n \times n$, where n is the problem size. Suppose that in the course of the algorithm, the v-th and w-th elements in the permutation p have been interchanged. Then, the tabu list (matrix) entry t_{vw} memorizes the current iteration number, q ($q \geq 1$), plus the tabu tenure, h ($h \geq 1$), i.e., the number of the future iteration starting at which the corresponding elements may again be interchanged. (Thus, $t_{vw} = q + h$. Of course, initially, all the values of T are equal to zero. The value of h depends on the problem size, n (we have used the values between $h = \lfloor 0.05n \rfloor$ and $h = \lfloor 1.0n \rfloor$).) We also use the hash table, HT. So, the formalistic tabu criterion, TC, can specifically be defined as provided in this expression: $TC_{ij} = \begin{cases} \text{TRUE}, & \left(t_{ij} \geq q\right) \text{ OR } \left(HT\left(\left(z(p) + \Delta_z\left(p, p^{i,j}\right) \text{ MOD } HTS\right)\right) = \text{TRUE}\right) \\ \text{FALSE}, & \text{otherwise} \end{cases}$, where p is the current solution, i, j are the element indices ($i = 1, \ldots, n-1, j = i+1, \ldots, n$), and HTS denotes the hash table size. It should be noted that, in our algorithm, the tabu status is disregarded at random moments with a (very) small probability. This strategy enables an increase in the number of non-prohibited solutions and does not suppress the search directions too much. In this case, the modified tabu criterion, $TC^{\ddagger}$, is defined in the following way: $TC^{\ddagger}_{ij} = \begin{cases} \text{TRUE}, & \left(t_{ij} \geq q\right) \text{ AND } \left(\varsigma \geq \alpha\right) \\ \text{FALSE}, & \text{otherwise} \end{cases}$, where ς is a uniform random real number within the interval $[0, 1]$, α ($\alpha \in (0, 1)$) denotes the randomization parameter (we used $\alpha = 0.01$).

The aspiration criterion, AC, is used alongside the tabu criterion. This means that the tabu status is ignored if the predefined condition is satisfied. In particular, the currently attained solution appears better than the best solution found so far. The formalized aspiration criterion, AC, is defined according to the following equation: $AC_{ij} = \begin{cases} \text{TRUE}, & z(p) + \Delta_z\left(p, p^{i,j}\right) < z^* \\ \text{FALSE}, & z(p) + \Delta_z\left(p, p^{i,j}\right) \geq z^* \end{cases}$, where z^* denotes the best value of the objective function found so far.

Having the tabu criterion and aspiration criterion defined, a move (acceptance) criterion, MC, is defined as follows: $MC_{ij} = \begin{cases} \text{TRUE}, & \left(\left(z(p) + \Delta_z\left(p, p^{i,j}\right) < z^{\bullet}\right) \text{AND} \left(TC^{\ddagger}_{ij} = \text{FALSE}\right)\right) \text{ OR } \left(AC_{ij} = \text{TRUE}\right) \\ \text{FALSE}, & \text{otherwise} \end{cases}$, where $z^{\bullet}$ denotes the minimum value of the objective function found at the current iteration. (At the beginning, $z^{\bullet} = \infty$(max integer).)

The best move is determined by the use of the following formula: $(v, w) = \underset{i,j}{\text{argmin}}\left\{MC_{ij} = \text{TRUE}\right\}$, where p is the current solution, $i = 1, \ldots, n-1, j = i+1, \ldots, n$.

More to this, our TS algorithm utilizes an additional memory—a secondary memory, SM (a similar approach is used in [94]). The purpose of using this memory is to archive high-quality solutions, which, although they are evaluated as very good, are not selected. Thus, if the best solution stays unchanged for more than $L_{idle\_iter} = \lfloor \gamma\tau \rfloor$ iterations, then the entire tabu list is wiped out and the search is reset to one of the "second" solutions in SM. (Here, τ denotes the number of iterations of the TS algorithm, and γ is an idle iterations limit factor such that $1 \leq \lfloor \gamma\tau \rfloor \leq \tau$.)

The pseudo-code of the tabu search algorithm is shown in Algorithm 3.

Remark 1. *The TS procedure is finished as soon as the total number of TS iterations, τ, have been executed.*

Remark 2. *After finishing the tabu search procedure, the obtained solution is subjected to perturbation, which is described in the next section.*

2.7.2. Perturbation (Mutation) Process

Roughly speaking, the perturbation is a stochastic, smaller, or larger move, or consecutive moves within a particular neighbourhood. The well-known examples of perturbations include classical random swap moves (shuffling) and complex destruction (-reconstruction) moves.

It is strongly believed that the perturbation component of the iterative heuristic algorithms has a very significant impact on the overall efficacy of these algorithms. This component constitutes one of the crucial factors in revealing new, undiscovered regions and directions in the solution space during the course of the search process, and this fact is acknowledged in several important research works in prestigious sources [95–106]. For example, in [96], it is ascertained that "perturbation is an effective strategy used to jump out of a local optimum and to search a new promising region". In [97], it is confirmed that perturbations can really help escape traps in (deep) local optima, which can occur in tabu search algorithms. A combination of several perturbation schemes was also tried [99–104]. For example, authors of [102,103] use both weak perturbations and strong perturbations because it was observed that weak perturbations are not sufficient for the algorithms to continue their search successfully. Therefore, the authors propose employing strong perturbations to jump out of local optima traps and bring the search to unexplored, distant regions in the search space. In [105], the authors use another term ("shake procedure") for the perturbation procedure, but the essence remains the same.

The perturbation is characterized in terms of a strength (mutation rate) parameter—denoted as ξ—which may be static or dynamic, and which can be defined as the number of moves during the perturbation process. This parameter is, in fact, the most influential quantitative factor in the perturbation process, and the size of perturbation strength, ξ, greatly affects the search progress. The larger the value of ξ, the larger number of elements is affected in the perturbation process (the more disrupted element becomes the solution, and the larger element is the distance between an unaffected solution and a permuted one), and vice versa. However, perturbation strength that is too large will cause the random–restart-like search. Meanwhile, perturbation strength that is too small will not be able to jump out of local optima, or will simply fall into a cyclic process [106].

Our perturbation process is, therefore, adaptive. It is adaptive in the sense that the perturbation strength depends on the particular problem (instance) size. That is, instances of different sizes will have different perturbation strengths. In order to adapt the perturbation strength to the instances of different sizes, the actual perturbation strength, ξ, is set to $\max\{2, \lfloor \omega n \rfloor\}$, where ω ($0 < \omega \leq 1$) is the perturbation strength factor. It is obvious that $2 \leq \xi \leq n$.

Regarding the formal definition of the perturbation procedure, it can be described by using a two-tuple operator $\phi(p, \xi)$: $\Pi_n \times N \to \Pi_n$, such that, if $p^{\sim} = \phi(p, \xi)$, then $p^{\sim} \in \Pi_n$ and $p^{\sim} \neq p$. So, $\phi(p, \xi)$ transforms the incumbent solution p into a new solution $p^{\sim}$ from the neighbourhood $\Theta_{\xi}(p)$, such that $\delta(p, p^{\sim}) = \delta(p, \phi(p)) = \xi$.

In our genetic algorithm, the perturbation process is integrated within the hierarchical iterated tabu search algorithm. In particular, the perturbation is applied to a chosen—improved—solution, which is picked in accordance with the predetermined candidate solution-acceptance rule. (In our algorithm, we used two rules: (1) accept every new (last) improved solution; (2) accept the best-so-far-found solution.) The permuted solution serves as an input for the tabu search procedure.

We dispose two main sorts of perturbation procedures: (1) random perturbation, (2) quasi-greedy (greedy-random) perturbation. In turn, two types of random perturba-

tions are proposed: (i) uniform (steady) random perturbation; (ii) Lévy perturbation. In both cases, the perturbation process is based on the random pairwise interchanges of the elements of the permutation. So, in the first case, the value of the perturbation rate factor ω is fixed (immutable) ($\omega = \text{const}$). Meanwhile, in the second case, the perturbation rate factor ω follows the Lévy distribution law [107,108]. In this case, the factor ω is as follows: $\omega^{(i)} = \text{wrap}\left(\omega^{(i-1)} + L(\eta)\right)$, where i denotes the current iteration number, $\omega^{(i)}$ refers to the perturbation rate factor at ith iteration, $\omega^{(0)}$ is from the interval $(0,\ 1]$, and the function (operator) $\text{wrap}(\cdot)$ "wraps" the particular number within the interval $(0,\ 1]$. $L(\eta)$ (step size) is calculated using Lévy distribution. It can be evaluated using the following formula: $L(\eta) = \frac{u}{v^{1/\eta}}$, where u and v are zero mean Gaussian random variables of variances σ and 1. Here, $\sigma = \left(\frac{\Gamma(1+\eta)\sin\left(\frac{\eta\pi}{2}\right)}{\Gamma\left(\frac{1+\eta}{2}\right)\eta 2^{(\eta-1)/2}}\right)^{1/\eta}$, where η is a parameter (we used $\eta = 1.5$) [109,110]. $\Gamma(\cdot)$ is a Gamma function that can practically be expressed as $\Gamma(\text{x}) \approx \sqrt{\frac{2\pi}{x}}\left(\frac{1}{e}\left(x + \frac{1}{12x - \frac{1}{10x}}\right)\right)^{x}$ [111]. Note that all random perturbations are quite aggressive and vigorous and introduce a considerable extent of diversification to the search process.

We have also designed three variants of the quasi-greedy (greedy-random) perturbations. All of them are based on "softly" (partially) randomized tabu search, where the tabu search process is intertwined with the random moves (the terms "tabu search-driven perturbation" and "tabu shaking" [98] can also be used). The tabu-based perturbation is very similar to the ordinary tabu search procedure. Two solutions—the best available non-tabu solution and the second-best non-tabu solution—are found by exploring the whole neighbourhood of the current solution. After this, only exactly one of these solutions is accepted with the probability, called switch probability, P_s ($P_s \in (0,\ 1)$), where the precise value of P_s is assigned by the user. The following are the three distinct particular variants: (1) $P_s = 0.1$ (quasi-greedy perturbation 1); (2) $P_s = 0.5$ (quasi-greedy perturbation 2); (3) $P_s = 0.9$ (quasi-greedy perturbation 3). Notice that the tabu-based perturbations are rather weak and allow for a small amount of diversification. Still, they are quite promising because they take the solutions' quality into consideration by avoiding too much deterioration of the solutions.

All the above-mentioned five perturbation procedures can be juxtaposed in many different ways to achieve the most relevant balance between exploration and exploitation in the search process. We argue that these procedures are among the most influential and sensitive factors for the efficiency of our genetic algorithm (as can be seen in the next section). The top-level pseudocode (in Backus–Naur-like syntax form) of the perturbation process is presented in Algorithm 4, which is a template for a family of multi-type perturbation procedures. It can be perceived that there altogether exists at least 92 different potential variations of Algorithm 4: $5 + 2 \times 3 + 3 \times 2 + 2 \times 3 \times 2 = 29$ variations if $N_{perturb} = 1$ and $3 + 2 \times 3 + 2 \times 3 + 3 \times 2 + 3 \times 2 + 2 \times 3 \times 2 + 2 \times 3 \times 2 + 2 \times 3 \times 2 = 63$ variations if $N_{perturb} > 1$ (also see Table A1 in Appendix A). So, this indeed offers a very large degree of flexibility and versatility to the perturbation process, and the researchers can choose a variation that suits their individual demands to the highest level. Note that such an approach is in connection to what is known as multi-strategy algorithms [112] and automatic design of algorithms (see, for example, [113,114]).

Algorithm 4 High-level (abstract) pseudocode of the universalized multi-type (multi-strategy) perturbation procedure

```
Universalized_Multi-Type_Perturbation;
// input: p—current solution
// output: p ∼—(best) obtained perturbed/reconstructed solution
// parameters: ω—perturbation strength (mutation rate) factor, N_perturb—number of perturbation iterations
//             PerturbVar—perturbation variant
begin
  p^∈ ← EMPTY_SOLUTION;
  [apply uniform random perturbation | Lévy perturbation; get permuted solution p∼;
   p ← p∼;]
  for i ← 1 to N_perturb do begin
    [apply uniform random perturbation | Lévy perturbation; get permuted solution p∼;
     p ← p∼;]
    [apply quasi-greedy perturbation 1 | 2 | 3; get permuted solution p∼;
     memorize best permuted solution as p^∈;]
    [apply uniform random perturbation | Lévy perturbation; get permuted solution p∼;
     p ← p∼]
  endfor;
  [apply uniform random perturbation | Lévy perturbation get permuted solution p∼;]
  if p^∈ is not EMPTY_SOLUTION then p∼ ← p^∈
end.
```

Notes. 1. The uniform random perturbation and Lévy perturbation procedures are applied to the incumbent solution p considering the particular perturbation strength (mutation rate) factor ω. Remark. 2. All the scenarios of Algorithm 4 are managed by using the switch parameter *PerturbVar*.

3. Computational Experiments

3.1. Experiment Setup

The improved hybrid genetic-hierarchical algorithm is coded by using C# programming language. The computational experiments have been conducted using a x86 series personal computer with Intel 3.1 GHz 4 cores processor, 8 GB RAM, and 64-bit MS Windows operating system. No parallel processing is used, and only one core is assigned to a separate algorithm.

In the experiments, we have used the test (benchmark) data instances from the electronic public library of the QAP instances—QAPLIB [115,116], as well as the papers by De Carvalho et al. (2006) [117] and Drezner et al. (2005) [118] (see also [119]). (The best-known solutions are from QAPLIB and [26,27,32,119–123].) Overall, 140 data instances were examined. The sizes of the instances vary between 10 and 729.

As the main algorithm quantitative performance criteria, we adopt the average deviation ($\overline{\theta}$) of the objective function and the number of best-known (pseudo-optimal) found solutions (N_{best}). The average deviation is calculated by the following formula: $\overline{\theta} = \frac{\overline{z}-BKV}{BKV} \times 100[\%]$, where $\overline{z}$ is the average objective function value and BKV denotes the best-known value of the objective function. The average deviation and the number of best-known solutions are calculated over R independent runs of the algorithm (where $R = 10$).

At every separate run, the algorithm with the fixed set of control parameters is applied to the particular benchmark data instance. Each time, the genetic algorithm is starting from a new, distinct random initial population. The execution of the algorithm is terminated if the total number of generations, G, has been reached, or if the best-known (pseudo-optimal) solution has been found.

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

Table 1. Values (ranges of values) of the control parameters of the improved hybrid genetic algorithm used in the experiments.

Parameters	Values	Remarks
Population size, PS	$[2, 40]$	
Initial population size factor, C	$[2, 4]$	$C \geq 1$
Number of generations, G	$[1, 100]$	
Distance threshold, DT	$\max\{2, \lfloor 0.5n \rfloor\}$	$2 \leq DT \leq n$
Idle generations limit, L_{idle_gen}	$\max\{2, \lfloor 0.05G \rfloor\}$	
Total number of iterations of hierarchical iterated tabu search, Q	$[1, 1000]$	$Q = Q^{(1)} \times Q^{(2)} \times \ldots$
Number of iterations of tabu search, τ	$[1, 500]$	
Tabu tenure, h	$[\lfloor 0.1n \rfloor, \lfloor 1.0n \rfloor]$	$h > 0$
Randomization coefficient for tabu search, α	0.01	$0 \leq \alpha < 1$
Idle iterations limit, L_{idle_iter}	$\max\{3, \lfloor 0.5\tau \rfloor\}$	$0 < L_{idle\_iter} \leq \tau$
Perturbation (mutation) rate factor, ω	$[0.1, 0.9]$	$0 < \omega \leq 1$
Switch probability, P_s	0.1|0.5|0.9	$0 < P_s < 1$
Perturbation variant (variation), *PerturbVar*	1.92	
Number of runs of the algorithm, R	10	

3.2. Main Results: Comparison of Algorithms and Discussion

The results of the conducted experiments are presented in Table 2. In this table, we provide the following information: BKV—best-known value of the objective function, $\bar{\theta}$—average deviation of the objective function, Time—the average time (CPU time) per one run of the algorithm.

These results were taken by properly adjusting the most suitable variations of the perturbation process (the sensitivity of results on the variations of perturbation procedure obtained in the preliminary experiments can be observed in Figure A1 in Appendix A—the represented results indicate that the performance of IHGHA heavily depends on perturbation process). Only the best-attained selected results are presented in Table 2, while the results of preparatory experimentation are omitted for convenience and brevity's sake. It can be viewed that the results from Table 2 evidently demonstrate the excellent performance and reliability of the proposed genetic algorithm from both the quality of solutions and the computational resources point of view.

Table 2. Results of IHGHA for the set of 140 instances of QAPLIB [115–117,119].

Instance	BKV	$\bar{\theta}$	Time (s)	Instance	BKV	$\bar{\theta}$	Time (s)
bl36	3296	0.000	7.608	**sko100b**	153,890	0.000	217.000
bl49	4548	0.000	649.900	**sko100c**	147,862	0.000	347.800
bl64	5988	0.000	1650.000	**sko100d**	149,576	0.000	346.500
bl81	7532	0.000	50,470.000	**sko100e**	149,150	0.000	315.700
bl100	9256	0.099	60,740.000	**sko100f**	149,036	0.000	591.900
bl121	11,396	0.126	125,300.000	**ste36a**	9526	0.000	0.245
bl144	13,432	0.229	178,300.000	**ste36b**	15,852	0.000	0.048
chr25a	3796	0.000	1.744	**ste36c**	8,239,110	0.000	0.078
ci36	168,611,971	0.000	1.175	**tai10a**	135,028	0.000	0.003
ci49	236,355,034	0.000	4.586	**tai10b**	1183,760	0.000	0.002
ci64	325,671,035	0.000	46.110	**tai12a**	224,416	0.000	0.003
ci81	427,447,820	0.000	236.500	**tai12b**	39,464,925	0.000	0.003
ci100	523,146,366	0.000	4562.000	**tai15a**	388,214	0.000	0.006
ci121	653,409,588	0.000	117,300.000	**tai15b**	51,765,268	0.000	0.005
ci144	794,811,636	0.003	199,400.000	**tai17a**	491,812	0.000	0.008
dre15	306	0.000	0.003	**tai20a**	703,482	0.000	0.103
dre18	332	0.000	0.028	**tai20b**	122,455,319	0.000	0.008
dre21	356	0.000	0.033	**tai25a**	1167,256	0.000	0.226
dre24	396	0.000	0.125	**tai25b**	344,355,646	0.000	0.031
dre28	476	0.000	0.393	**tai27e1**	2558	0.000	0.114
dre30	508	0.000	0.629	**tai27e2**	2850	0.000	0.207
dre42	764	0.000	6.351	**tai27e3**	3258	0.000	0.075

Table 2. *Cont.*

Instance	BKV	$\bar{\theta}$	Time (s)	Instance	BKV	$\bar{\theta}$	Time (s)
dre56	1086	0.000	53.610	**tai27e4**	2822	0.000	0.089
dre72	1452	0.000	160.000	**tai27e5**	3074	0.000	0.072
dre90	1838	0.000	1341.000	**tai30a**	1818,146	0.000	0.261
dre110	2264	0.000	7458.000	**tai30b**	637,117,113	0.000	0.117
dre132	2744	0.000	51,840.000	**tai35a**	2422,002	0.000	1.392
els19	17,212,548	0.000	0.009	**tai35b**	283,315,445	0.000	0.492
esc32a	130	0.000	0.119	**tai40a**	3139,370	0.000	963.200
esc32b	168	0.000	0.009	**tai40b**	637,250,948	0.000	0.395
esc32c	642	0.000	0.003	**tai45e1**	6412	0.000	0.604
esc32d	200	0.000	0.006	**tai45e2**	5734	0.000	0.797
esc32e	2	0.000	0.002	**tai45e3**	7438	0.000	0.839
esc32f	2	0.000	0.003	**tai45e4**	6698	0.000	0.863
esc32g	6	0.000	0.002	**tai45e5**	7274	0.000	0.504
esc32h	438	0.000	0.006	**tai50a**	4938,796	0.000	2704.000
esc64a	116	0.000	0.019	**tai50b**	458,821,517	0.000	3.875
esc128	64	0.000	0.178	**tai60a**	7,205,962	0.000	8965.000
had20	6922	0.000	0.008	**tai80a**	13,499,184	0.153	68,420.000
kra30a	88,900	0.000	0.175	**tai100a**	21,043,560	0.201	97,600.000
kra30b	91,420	0.000	0.278	**tai60b**	608,215,054	0.000	4.632
kra32	88,700	0.000	0.077	**tai64c**	1,855,928	0.000	0.019
lipa20a	3683	0.000	0.008	**tai75e1**	14,488	0.000	18.710
lipa20b	27,076	0.000	0.002	**tai75e2**	14,444	0.000	19.480
lipa30a	13,178	0.000	0.030	**tai75e3**	14,154	0.000	23.590
lipa30b	151,426	0.000	0.008	**tai75e4**	13,694	0.000	34.270
lipa40a	31,538	0.000	0.184	**tai75e5**	12,884	0.000	17.400
lipa40b	476,581	0.000	0.017	**tai80b**	818,415,043	0.000	23.860
lipa50a	62,093	0.000	0.373	**tai100b**	1,185,996,137	0.000	66.760
lipa50b	1,210,244	0.000	0.050	**tai125e1**	35,426	0.000	1610.000
lipa60a	107,218	0.000	3.647	**tai125e2**	36,178	0.000	2156.000
lipa60b	2,520,135	0.000	0.137	**tai125e3**	30,498	0.000	1414.000
lipa70a	169,755	0.000	4.485	**tai125e4**	33,084	0.000	2283.000
lipa70b	4,603,200	0.000	0.212	**tai125e5**	37,210	0.000	2299.000
lipa80a	253,195	0.000	18.550	**tai150b**	498,896,643	0.000	2384.000
lipa80b	7,763,962	0.000	0.499	**tai343e1**	141,048	0.098	13,110.000
lipa90a	360,630	0.000	72.840	**tai343e2**	148,584	0.112	12,980.000
lipa90b	12,490,441	0.000	0.746	**tai343e3**	142,092	0.232	13,070.000
nug28	5166	0.000	0.039	**tai343e4**	152,966	0.134	13,440.000
nug30	6124	0.000	0.075	**tai343e5**	139,114	0.143	13,280.000
rou20	725,522	0.000	0.064	**tai729e1**	416,260	0.901	112,300.000
scr20	110,030	0.000	0.017	**tai729e2**	422,570	0.899	111,900.000
sko42	15,812	0.000	0.415	**tai729e3**	405,004	0.945	112,400.000
sko49	23,386	0.000	5.763	**tai729e4**	412,910	0.929	113,500.000
sko56	34,458	0.000	5.620	**tai729e5**	418,018	0.980	112,000.000
sko64	48,498	0.000	6.424	**tho30**	149,936	0.000	0.056
sko72	66,256	0.000	25.500	**tho40**	240,516	0.000	2.948
sko81	90,998	0.000	65.390	**tho150**	8,133,398	0.000	73,000.000
sko90	115,534	0.000	170.200	**wil50**	48,816	0.000	2.839
sko100a	152,002	0.000	254.200	**wil100**	273,038	0.000	548.600

Notes. Time denotes the average CPU time per one run. 123 runs succeeded in finding (pseudo-) optimal solutions with 100% success rate.

Regarding the comparison of our algorithm and other heuristic algorithms for the QAP, it can be seen, first of all, that our results (see Table 2) are obviously better than those reached in the previous version of the hybrid genetic-hierarchical algorithm (see Table 9 in [29]).

We have also compared our algorithm and the other state-of-the-art heuristic algorithms for the QAP, in particular, the frequent pattern-based search algorithm [67] and the newest elaborated versions of the hybrid genetic algorithm [27,118]. The results of comparisons are shown in Tables 3–6. As can be seen, these results again illustrate the superior efficacy of IHGHA. It appears that the results are seemingly in our favour with respect to all the other competitors considered here. This is very evident in our algorithm's run times, especially for the small- and medium-scaled problems. (Notice that the difference in run time between algorithms can exceed a factor of 100 for some instances; see, for example,

the results of comparison of IHGHA and HGA for the instances tai27e∗, tai45e∗, tai75e∗, and tai125e∗.)

In the final analysis, our main observations are as follows:

1. Based on Table 2, we were able to achieve 100% success rate for almost all examined instances (in particular, for 123 instances out of 140). These instances are solved to (pseudo-)optimality within very reasonable computation times, which are, to our knowledge, record-breaking in many cases. The exception is only a handful of instances (namely, bl100, bl121, bl144, ci144, tai80a, tai100a, tho150, tai343e∗, and tai729e∗). Among these instances, the instances bl100, bl121, bl144, tai100a, tai343e∗, and tai729e∗ are overwhelmingly difficult for the heuristic algorithms and still need new revolutionizing algorithmic solutions.
2. The best-known solution was, in total, found in 1253 runs out of 1400 runs (89.5% of the runs). We also found the best-known solution at least once out of 10 runs for 129 instances out of 140 (92.14% of the instances). And we achieved an average deviation of less than 0.1% for 127 instances out of 140 (90.71% of the instances). The cumulative average deviation over 140 instances is equal to 0.044%.
3. On top of this, we were successful in achieving three new best-known solutions for the instances bl100, bl121, and bl144, which are presented in Tables 7–9.

Overall, the results from Tables 2–9 demonstrate that our algorithm is quite successful with respect to the three essential aspects: (i) effectiveness in obtaining the zero percentage average deviation of the yielded solutions for most of the examined problems; (ii) ability to reaching out for the (pseudo-)optimal or near-optimal solutions within extremely small running times for the small- and medium-scaled instances; (iii) potential to achieving the best-known/record-breaking solutions for the hard instances. Overall, it could be argued that the obtained results confirm the usefulness of the proposed two-layer architecture of the hybrid genetic algorithm and the relevance of our multi-type perturbation process, which is integrated within the hierarchical tabu search algorithm and which seems to be very well-suited for our hybrid genetic algorithm.

It should be noted that there is still room for further improvements. In this regard, the parameter adjustment/calibration is one of the key factors in obtaining the increased efficacy of the algorithm. In this respect, two main ways are manual and automatic calibration.

In the first case, the designer's and/or user's acquired experience and competence play an extremely important role. On the other hand, the determination of the most sensitive and influential parameters is also of the highest importance.

As a representative example, we demonstrate how, in our case, the performance of the algorithm is increased by simply manipulating the number of iterations of the algorithm. The results are presented in Figure A2 in Appendix A.

Regarding the automatic parameter adaptation, we believe that this could be one of the very promising future research directions.

Last but not least, the performance of IHGHA can be improved even more by applying an elementary, straightforward approach of parallelization in a distributed-like manner. Where autonomous, independent clones of the algorithm are assigned to separate cores of the computer. So, instead of running a single copy of the algorithm R times, the algorithm is run only 1 time on R cores (with different values of the random number generator's seed). The anticipated effect is a decrease in the total run time by a factor of R.

Table 3. Comparison of the results of IHGHA and frequent pattern-based search (FPBS) algorithm [67] (part I, part II).

(I)					
Instance	**BKV**	**IHGHA**		**FPBS**	
		$\bar{\theta}$	**Time (s)**	$\bar{\theta}$	**Time (s)**
sko72	66,256	**0.000(10)**	**25.500**	**0.000(10)**	288.000
sko81	90,998	**0.000(10)**	**65.390**	**0.000(10)**	198.000
sko90	115,534	**0.000(10)**	170.200	0.010(n/a)	**144.000**
sko100a	152,002	**0.000(10)**	**254.200**	**0.000(10)**	510.000
sko100b	153,890	**0.000(10)**	**217.000**	**0.000(10)**	348.000
sko100c	147,862	**0.000(10)**	**347.800**	**0.000(10)**	522.000
sko100d	149,576	**0.000(10)**	**346.500**	**0.000(10)**	972.000
sko100e	149,150	**0.000(10)**	**315.700**	**0.000(10)**	732.000
sko100f	149,036	**0.000(10)**	591.900	0.003(n/a)	**240.000**
wil100	273,038	**0.000(10)**	**548.600**	**0.000(10)**	984.000
tho150	8,133,398	**0.000(7)**	73,000.000	0.006(n/a)	**3444.000**
(II)					
Instance	**BKV**	**IHGHA**		**FPBS**	
		$\bar{\theta}$	**Time (s)**	$\bar{\theta}$	**Time (s)**
tai40a	3,139,370	**0.000(10)**	**963.200**	0.037(n/a)	3150.000
tai50a	4,938,796	**0.000(10)**	**2704.000**	0.106(n/a)	4068.000
tai60a	7,205,962	**0.000(10)**	8965.000	0.189(n/a)	**3600.000**
tai80a	13,499,184	**0.153(3)**	68,420.000	0.467(0)	**3112.000**
tai100a	21,043,560	**0.201**(0)	97,600.000	0.390(0)	**2166.000**
tai50b	458,821,517	**0.000(10)**	**3.875**	**0.000(10)**	12.000
tai60b	608,215,054	**0.000(10)**	**4.632**	**0.000(10)**	24.000
tai80b	818,415,043	**0.000(10)**	**23.860**	**0.000(10)**	84.000
tai100b	11,185,996,137	**0.000(10)**	**66.760**	**0.000(10)**	174.000
tai150b	498,896,643	**0.000(10)**	**2384.000**	0.092(n/a)	2784.000

Notes. (I) Time denotes the average CPU time per one run. In parentheses, we present the number of times that the BKS (best-known solution) has been found. (II) Time denotes the average CPU time per one run. In parentheses, we present the number of times that the BKS has been found. The best-known value for the instance tai100a is from [120].

Table 4. Comparison of the results of IHGHA and hybrid genetic algorithm (HGA) [118] (part I, part II).

(I)					
Instance	**BKV**	**IHGHA**		**HGA**	
		$\bar{\theta}$	**Time (s)**	$\bar{\theta}$	**Time (s)**
dre30	508	**0.000(10)**	**0.629**	**0.000(10)**	143.400
dre42	764	**0.000(10)**	**6.351**	1.340(n/a)	547.800
dre56	1086	**0.000(10)**	**53.610**	17.460(n/a)	1810.800
dre72	1452	**0.000(10)**	**160.000**	27.280(n/a)	5591.400
dre90	1838	**0.000(10)**	**1341.000**	33.880(n/a)	11,557.800
dre110	2264	**0.000(10)**	**7458.000**	n/a	n/a
dre132	2744	**0.000(10)**	**51,840.000**	n/a	n/a
(II)					
Instance	**BKV**	**IHGHA**		**HGA**	
		$\bar{\theta}$	**Time (s)**	$\bar{\theta}$	**Time (s)**
tai27e1	2558	**0.000(10)**	**0.114**	**0.000(10)**	~60.000
tai27e2	2850	**0.000(10)**	**0.207**	**0.000(10)**	~60.000
tai27e3	3258	**0.000(10)**	**0.075**	**0.000(10)**	~60.000
tai45e1	6412	**0.000(10)**	**0.604**	**0.000(10)**	~300.000
tai45e2	5734	**0.000(10)**	**0.797**	**0.000(10)**	~300.000
tai45e3	7438	**0.000(10)**	**0.839**	**0.000(10)**	~300.000
tai75e1	14,488	**0.000(10)**	**18.710**	**0.000(10)**	~2220.000
tai75e2	14,444	**0.000(10)**	**19.480**	0.339(n/a)	~2220.000
tai75e3	14,154	**0.000(10)**	**23.590**	**0.000(10)**	~2220.000
tai125e1	35,426	**0.000(10)**	**1610.000**	n/a	n/a
tai125e2	36,178	**0.000(10)**	**2156.000**	n/a	n/a
tai125e3	30,498	**0.000(10)**	**1414.000**	n/a	n/a

Notes. (I) Time denotes the average CPU time per one run. In parentheses, we present the number of times that the BKS has been found. (II) Time denotes the average CPU time per one run. In parentheses, we present the number of times that the BKS has been found. The best-known values for the instances tai125e1, tai125e2, and tai125e3 are from [32].

Table 5. Comparison of the results of IHGHA and hybrid genetic algorithm with biologically inspired parent selection (HGA-BIPS) [27].

Instance	BKV	IHGHA		HGA-BIPS	
		$\bar{\theta}$	Time (s)	$\bar{\theta}$	Time (s)
bl36	3296	**0.000(10)**	**7.608**	**0.000(10)**	135.600
bl49	4548	**0.000(10)**	**649.900**	0.193(n/a)	959.400
bl64	5988	**0.000(10)**	**1650.000**	0.084(n/a)	2758.200
bl81	7532	**0.000(10)**	50,470.000	0.154(n/a)	**7316.400**
bl100	9256	**0.099(3)**	60,740.000	0.164(0)	**24,119.000**
bl121	11,396	**0.126(1)**	125,300.000	0.281(0)	**69,892.000**
bl144	13,432	**0.229(1)**	**178,300.000**	0.459(0)	189,168.000

Notes. Time denotes the average CPU time per one run. In parentheses, we present the number of times that the BKS has been found. The best-known values for the instances bl36, bl49, and bl64 are from [121,122]. The best-known value for the instance bl81 is from [123]. The best-known values for the instances bl100 and bl121 are from [26]. The best-known value for the instance bl144 is from [27]. The new best-known solutions have been achieved in this paper for the instances bl100, bl121, and bl144, which are presented in Tables 7–9.

Table 6. Comparison of the results of IHGHA and hybrid genetic algorithm with biologically inspired parent selection (HGA-BIPS) [27].

Instance	BKV	IHGHA		HGA-BIPS	
		$\bar{\theta}$	Time (s)	$\bar{\theta}$	Time (s)
ci36	168,611,971	**0.000(10)**	**1.175**	**0.000(10)**	142.800
ci49	236,355,034	**0.000(10)**	**4.586**	**0.000(10)**	614.400
ci64	325,671,035	**0.000(10)**	**46.110**	**0.000(10)**	2285.000
ci81	427,447,820	**0.000(10)**	**236.500**	**0.000(10)**	7456.000
ci100	523,146,366	**0.000(10)**	**4562.000**	**0.000(10)**	22,753.000
ci121	653,409,588	**0.000(10)**	117,300.000	0.005(n/a)	**65,011.000**
ci144	794,811,636	**0.003(8)**	199,400.000	0.020(n/a)	**177,469.000**

Notes. Time denotes the average CPU time per one run. In parentheses, we present the number of times that the BKS has been found. The best known values for the instances ci36, ci49, ci64, ci81, and ci100 are from [122]. The best-known values for the instances ci121 and ci144 are from [123].

Table 7. History of discovering the best-known solutions for the QAP instance bl100.

Objective Function Value	Algorithm	Authors	Year	References
9432	GA-TS	J.M. Rodriguez et al.	2004	[124]
9272	HGA	Z. Drezner, G. Marcoulides	2008/9	[121,122]
9264	HGA-DI	Z. Drezner, A. Misevičius	2013	[123]
9256	HGA-AM	Z. Drezner, T.D. Drezner	2019	[26]
9248	IHGHA	A. Misevičius et al.	2024	this paper

Notes. GA-TS—genetic algorithm with tabu search, HGA—hybrid genetic algorithm, HGA-DI—hybrid genetic algorithm with differential improvement, HGA-AM—alpha male hybrid genetic algorithm (hybrid genetic algorithm using alpha male).

Table 8. History of discovering the best-known solutions for the QAP instance bl121.

Objective Function Value	Algorithm	Authors	Year	References
11,640	GA-TS	J.M. Rodriguez et al.	2004	[124]
11,412	HGA	Z. Drezner, G. Marcoulides	2008/9	[121,122]
11,400	HGA-DI	Z. Drezner, A. Misevičius	2013	[123]
11,396	HGA-AM	Z. Drezner, T.D. Drezner	2019	[26]
11,392	IHGHA	A. Misevičius et al.	2024	this paper

Notes. GA-TS—genetic algorithm with tabu search, HGA—hybrid genetic algorithm, HGA-DI—hybrid genetic algorithm with differential improvement, HGA-AM—alpha male hybrid genetic algorithm (hybrid genetic algorithm using alpha male).

Table 9. History of discovering the best-known solutions for the QAP instance bl144.

Objective Function Value	Algorithm	Authors	Year	References
13,832	GA-TS	J.M. Rodriguez et al.	2004	[124]
13,472	HGA	Z. Drezner, G. Marcoulides	2008/9	[121,122]
13,460	HGA-DI	Z. Drezner, A. Misevičius	2013	[123]
13,432	HGA-BIPS	Z. Drezner, T.D. Drezner	2020	[27]
13,428	IHGHA	A. Misevičius et al.	2024	this paper

Notes. GA-TS—genetic algorithm with tabu search, HGA—hybrid genetic algorithm, HGA-DI—hybrid genetic algorithm with differential improvement, HGA-BIPS—hybrid genetic algorithm with biologically inspired parent selection.

4. Concluding Remarks

In this paper, we have presented the improved version of the hybrid genetic-hierarchical algorithm (IHGHA) for the solution of the well-known combinatorial optimization problem—the quadratic assignment problem. The algorithm was extensively examined on 140 QAP benchmark instances of various categories.

The following are the main essential points with respect to the presented algorithm:

- Two-level scheme of the hybrid primary (master)-secondary (slave) genetic algorithm is proposed;
- The multi-strategy perturbation process—which is integrated within the hierarchical iterated tabu search algorithm—is introduced.

This multi-type perturbation process plays an extremely significant role in our hybrid genetic algorithm due to its flexibility and versatility, and we think that it will be very interesting and helpful for researchers to contribute to furthering the progress of heuristic algorithms.

The obtained results confirm the excellent efficiency of the proposed algorithm. The set of more than a hundred QAP instances with sizes up to 729 is solved very effectively. This is especially noticeable for the small- and medium-sized instances. Three new record-breaking solutions have been achieved for the extraordinarily hard QAP instances, and we hope that our algorithm can serve as a landmark for the new heuristic algorithms for the QAP.

As to future work, it would be worthy to investigate the automatic (adaptive) selection/determination of the perturbation procedure variants in the hierarchical tabu search of the genetic algorithm.

Author Contributions: Conceptualization, A.M.; Methodology, A.M.; Software, A.M. and A.A.; Validation, A.O. and D.V.; Formal analysis, A.M.; Investigation, A.M. and A.A.; Resources, A.M.; Data curation, A.M.; Writing—original draft, A.M.; Writing—review & editing, A.O., D.V. and G.Ž.; Visualization, D.V.; Supervision, A.M.; Project administration, A.M.; Funding acquisition, A.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

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

Appendix A

Table A1. Variations of the perturbation procedure.

Perturbation Variations			
1—URP	24—LP, QGP 1, URP	47—URP, MQGP 3, URP	70—M(URP, QGP 2), LP
2—LP	25—LP, QGP 1, LP	48—URP, MQGP 1, LP	71—M(URP, QGP 3), LP
3—QGP 1	26—LP, QGP 2, URP	49—URP, MQGP 2, LP	72—M(LP, QGP 1), LP
4—QGP 2	27—LP, QGP 2, LP	50—URP, MQGP 3, LP	73—M(LP, QGP 2), LP
5—QGP 3	28—LP, QGP 3, URP	51—LP, MQGP 1, URP	74—M(LP, QGP 3), LP
6—URP, QGP 1	29—LP, QGP 3, LP	52—LP, MQGP 2, URP	75—M(QGP 1, URP)
7—URP, QGP 2	30—MQGP 1	53—LP, MQGP 3, URP	76—M(QGP 2, URP)
8—URP, QGP 3	31—MQGP 2	54—LP, MQGP 1, LP	77—M(QGP 3, URP)
9—LP, QGP 1	32—MQGP 3	55—LP, MQGP 2, LP	78—M(QGP 1, LP)
10—LP, QGP 2	33—URP, MQGP 1	56—LP, MQGP 3, LP	79—M(QGP 2, LP)
11—LP, QGP 3	34—URP, MQGP 2	57—M(URP, QGP 1)	80—M(QGP 3, LP)
12—QGP 1, URP	35—URP, MQGP 3	58—M(URP, QGP 2)	81—URP, M(QGP 1, URP)
13—QGP 1, LP	36—LP, MQGP 1	59—M(URP, QGP 3)	82—URP, M(QGP 2, URP)
14—QGP 2, URP	37—LP, MQGP 2	60—M(LP, QGP 1)	83—URP, M(QGP 3, URP)
15—QGP 2, LP	38—LP, MQGP 3	61—M(LP, QGP 2)	84—URP, M(QGP 1, LP)
16—QGP 3, URP	39—MQGP 1, URP	62—M(LP, QGP 3)	85—URP, M(QGP 2, LP)
17—QGP 3, LP	40—MQGP 2, URP	63—M(URP, QGP 1), URP	86—URP, M(QGP 3, LP)
18—URP, QGP 1, URP	41—MQGP 3, URP	64—M(URP, QGP 2), URP	87—LP, M(QGP 1, URP)
19—URP, QGP 1, LP	42—MQGP 1, LP	65—M(URP, QGP 3), URP	88—LP, M(QGP 2, URP)
20—URP, QGP 2, URP	43—MQGP 2, LP	66—M(LP, QGP 1), URP	89—LP, M(QGP 3, URP)
21—URP, QGP 2, LP	44—MQGP 3, LP	67—M(LP, QGP 2), URP	90—LP, M(QGP 1, LP)
22—URP, QGP 3, URP	45—URP, MQGP 1, URP	68—M(LP, QGP 3), URP	91—LP, M(QGP 2, LP)
23—URP, QGP 3, LP	46—URP, MQGP 2, URP	69—M(URP, QGP 1), LP	92—LP, M(QGP 3, LP)

Notes. URP—uniform (steady) random perturbation, LP—Lévy perturbation, QGP—quasi-greedy perturbation, MQGP—multiple (cyclic) quasi-greedy perturbation, M(URP, QGP)—multiple (cyclic) uniform (steady) random and quasi-greedy perturbation, M(LP, QGP)—multiple (cyclic) Lévy and quasi-greedy perturbation, M(QGP, URP)—multiple (cyclic) quasi-greedy and uniform (steady) random perturbation, M(QGP, LP)—multiple (cyclic) quasi-greedy and Lévy perturbation.

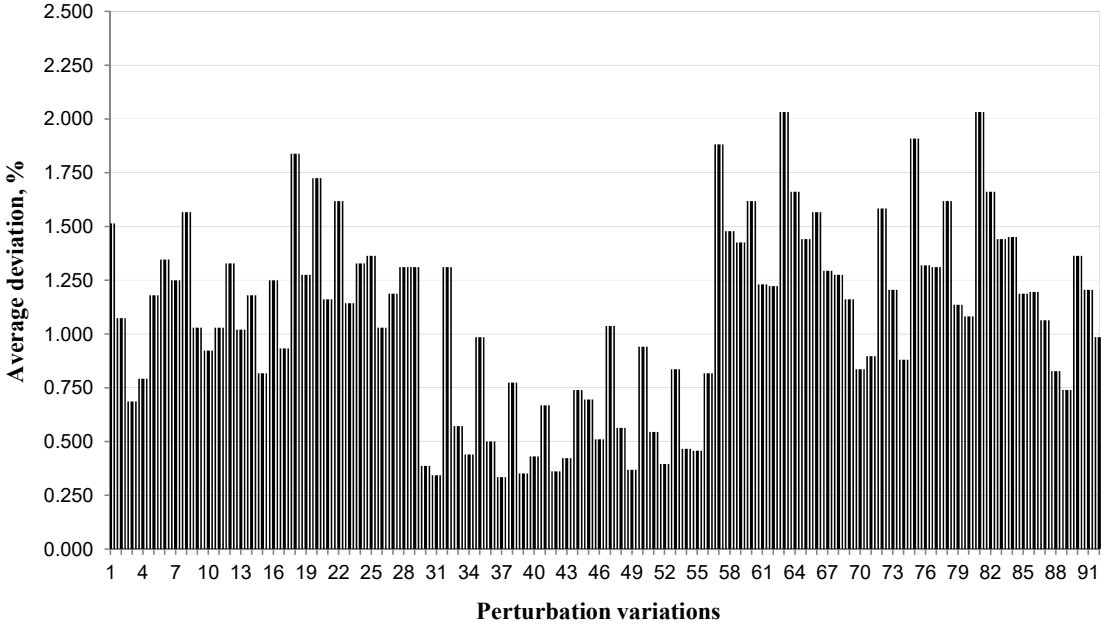

Figure A1. Percentage average deviation of solutions vs. perturbation variations for the QAP instance bl49 using short runs of IHGHA.

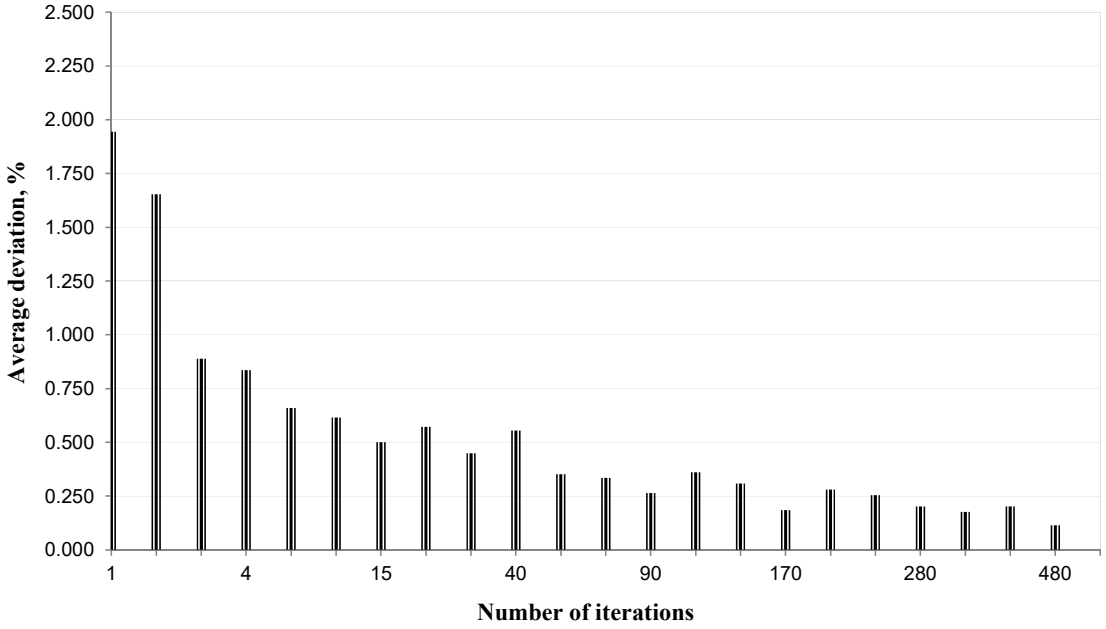

Figure A2. Percentage average deviation of solutions versus number of iterations for the QAP instance bl49.

References

1. Çela, E. *The Quadratic Assignment Problem: Theory and Algorithms*; Kluwer: Dordrecht, The Netherland, 1998.
2. Koopmans, T.; Beckmann, M. Assignment problems and the location of economic activities. *Econometrica* **1957**, *25*, 53–76. [CrossRef]
3. Drezner, Z. The quadratic assignment problem. In *Location Science*; Laporte, G., Nickel, S., Saldanha da Gama, F., Eds.; Springer: Cham, Switzerland, 2015; pp. 345–363. [CrossRef]
4. Sahni, S.; Gonzalez, T. P-complete approximation problems. *J. ACM* **1976**, *23*, 555–565. [CrossRef]
5. Anstreicher, K.M.; Brixius, N.W.; Gaux, J.P.; Linderoth, J. Solving large quadratic assignment problems on computational grids. *Math. Program.* **2002**, *91*, 563–588. [CrossRef]
6. Hahn, P.M.; Zhu, Y.-R.; Guignard, M.; Hightower, W.L.; Saltzman, M.J. A level-3 reformulation-linearization technique-based bound for the quadratic assignment problem. *INFORMS J. Comput.* **2012**, *24*, 202–209. [CrossRef]
7. Date, K.; Nagi, R. Level 2 reformulation linearization technique-based parallel algorithms for solving large quadratic assignment problems on graphics processing unit clusters. *INFORMS J. Comput.* **2019**, *31*, 771–789. [CrossRef]
8. Ferreira, J.F.S.B.; Khoo, Y.; Singer, A. Semidefinite programming approach for the quadratic assignment problem with a sparse graph. *Comput. Optim. Appl.* **2018**, *69*, 677–712. [CrossRef]
9. Armour, G.C.; Buffa, E.S. A heuristic algorithm and simulation approach to relative location of facilities. *Manag. Sci.* **1963**, *9*, 294–304. [CrossRef]
10. Buffa, E.S.; Armour, G.C.; Vollmann, T.E. Allocating facilities with CRAFT. *Harvard Bus. Rev.* **1964**, *42*, 136–158.
11. Murthy, K.A.; Li, Y.; Pardalos, P.M. A local search algorithm for the quadratic assignment problem. *Informatica* **1992**, *3*, 524–538.
12. Pardalos, P.M.; Murthy, K.A.; Harrison, T.P. A computational comparison of local search heuristics for solving quadratic assignment problems. *Informatica* **1993**, *4*, 172–187.
13. Angel, E.; Zissimopoulos, V. On the quality of local search for the quadratic assignment problem. *Discret. Appl. Math.* **1998**, *82*, 15–25. [CrossRef]
14. Benlic, U.; Hao, J.-K. Breakout local search for the quadratic assignment problem. *Appl. Math. Comput.* **2013**, *219*, 4800–4815. [CrossRef]
15. Aksan, Y.; Dokeroglu, T.; Cosar, A. A stagnation-aware cooperative parallel breakout local search algorithm for the quadratic assignment problem. *Comput. Ind. Eng.* **2017**, *103*, 105–115. [CrossRef]
16. Misevičius, A. A modified simulated annealing algorithm for the quadratic assignment problem. *Informatica* **2003**, *14*, 497–514. [CrossRef]
17. Taillard, E.D. Robust taboo search for the QAP. *Parallel Comput.* **1991**, *17*, 443–455. [CrossRef]
18. Misevicius, A. A tabu search algorithm for the quadratic assignment problem. *Comput. Optim. Appl.* **2005**, *30*, 95–111. [CrossRef]
19. Fescioglu-Unver, N.; Kokar, M.M. Self controlling tabu search algorithm for the quadratic assignment problem. *Comput. Ind. Eng.* **2011**, *60*, 310–319. [CrossRef]
20. Misevicius, A. An implementation of the iterated tabu search algorithm for the quadratic assignment problem. *OR Spectrum* **2012**, *34*, 665–690. [CrossRef]

21. Shylo, P.V. Solving the quadratic assignment problem by the repeated iterated tabu search method. *Cybern. Syst. Anal.* **2017**, *53*, 308–311. [CrossRef]
22. Drezner, Z. A new genetic algorithm for the quadratic assignment problem. *INFORMS J. Comput.* **2003**, *15*, 320–330. [CrossRef]
23. Misevicius, A. An improved hybrid genetic algorithm: New results for the quadratic assignment problem. *Knowl.-Based Syst.* **2004**, *17*, 65–73. [CrossRef]
24. Benlic, U.; Hao, J.-K. Memetic search for the quadratic assignment problem. *Expert Syst. Appl.* **2015**, *42*, 584–595. [CrossRef]
25. Ahmed, Z.H. A hybrid algorithm combining lexisearch and genetic algorithms for the quadratic assignment problem. *Cogent Eng.* **2018**, *5*, 1423743. [CrossRef]
26. Drezner, Z.; Drezner, T.D. The alpha male genetic algorithm. *IMA J. Manag. Math.* **2019**, *30*, 37–50. [CrossRef]
27. Drezner, Z.; Drezner, T.D. Biologically inspired parent selection in genetic algorithms. *Ann. Oper. Res.* **2020**, *287*, 161–183. [CrossRef]
28. Zhang, H.; Liu, F.; Zhou, Y.; Zhang, Z. A hybrid method integrating an elite genetic algorithm with tabu search for the quadratic assignment problem. *Inf. Sci.* **2020**, *539*, 347–374. [CrossRef]
29. Misevičius, A.; Verenė, D. A hybrid genetic-hierarchical algorithm for the quadratic assignment problem. *Entropy* **2021**, *23*, 108. [CrossRef]
30. Ryu, M.; Ahn, K.-I.; Lee, K. Finding effective item assignment plans with weighted item associations using a hybrid genetic algorithm. *Appl. Sci.* **2021**, *11*, 2209. [CrossRef]
31. Silva, A.; Coelho, L.C.; Darvish, M. Quadratic assignment problem variants: A survey and an effective parallel memetic iterated tabu search. *Eur. J. Oper. Res.* **2021**, *292*, 1066–1084. [CrossRef]
32. Wang, H.; Alidaee, B. A new hybrid-heuristic for large-scale combinatorial optimization: A case of quadratic assignment problem. *Comput. Ind. Eng.* **2023**, *179*, 109220. [CrossRef]
33. Ismail, M.; Rashwan, O. A Hierarchical Data-Driven Parallel Memetic Algorithm for the Quadratic Assignment Problem. Available online: https://ssrn.com/abstract=4517038 (accessed on 22 January 2024).
34. Arza, E.; Pérez, A.; Irurozki, E.; Ceberio, J. Kernels of Mallows models under the Hamming distance for solving the quadratic assignment problem. *Swarm Evol. Comput.* **2020**, *59*, 100740. [CrossRef]
35. Pradeepmon, T.G.; Panicker, V.V.; Sridharan, R. A variable neighbourhood search enhanced estimation of distribution algorithm for quadratic assignment problems. *OPSEARCH* **2021**, *58*, 203–233. [CrossRef]
36. Hameed, A.S.; Aboobaider, B.M.; Mutar, M.L.; Choon, N.H. A new hybrid approach based on discrete differential evolution algorithm to enhancement solutions of quadratic assignment problem. *Int. J. Ind. Eng. Comput.* **2020**, *11*, 51–72. [CrossRef]
37. Gambardella, L.M.; Taillard, E.D.; Dorigo, M. Ant colonies for the quadratic assignment problem. *J. Oper. Res. Soc.* **1999**, *50*, 167–176. [CrossRef]
38. Hafiz, F.; Abdennour, A. Particle swarm algorithm variants for the quadratic assignment problems—A probabilistic learning approach. *Expert Syst. Appl.* **2016**, *44*, 413–431. [CrossRef]
39. Dokeroglu, T.; Sevinc, E.; Cosar, A. Artificial bee colony optimization for the quadratic assignment problem. *Appl. Soft Comput.* **2019**, *76*, 595–606. [CrossRef]
40. Samanta, S.; Philip, D.; Chakraborty, S. A quick convergent artificial bee colony algorithm for solving quadratic assignment problems. *Comput. Ind. Eng.* **2019**, *137*, 106070. [CrossRef]
41. Peng, Z.-Y.; Huang, Y.-J.; Zhong, Y.-B. A discrete artificial bee colony algorithm for quadratic assignment problem. *J. High Speed Netw.* **2022**, *28*, 131–141. [CrossRef]
42. Adubi, S.A.; Oladipupo, O.O.; Olugbara, O.O. Evolutionary algorithm-based iterated local search hyper-heuristic for combinatorial optimization problems. *Algorithms* **2022**, *15*, 405. [CrossRef]
43. Wu, P.; Hung, Y.-Y.; Yang, K.-J. A Revised Electromagnetism-Like Metaheuristic For.pdf. Available online: https://www.researchgate.net/profile/Peitsang-Wu/publication/268412372_A_REVISED_ELECTROMAGNETISM-LIKE_METAHEURISTIC_FOR_THE_QUADRATIC_ASSIGNMENT_PROBLEM/links/54d9e45f0cf25013d04353b9/A-REVISED-ELECTROMAGNETISM-LIKE-METAHEURISTIC-FOR-THE-QUADRATIC-ASSIGNMENT-PROBLEM.pdf (accessed on 22 January 2024).
44. Riffi, M.E.; Bouzidi, A. Discrete cat swarm optimization for solving the quadratic assignment problem. *Int. J. Soft Comput. Softw. Eng.* **2014**, *4*, 85–92. [CrossRef]
45. Lim, W.L.; Wibowo, A.; Desa, M.I.; Haron, H. A biogeography-based optimization algorithm hybridized with tabu search for the quadratic assignment problem. *Comput. Intell. Neurosci.* **2016**, *2016*, 5803893. [CrossRef] [PubMed]
46. Houssein, E.H.; Mahdy, M.A.; Blondin, M.J.; Shebl, D.; Mohamed, W.M. Hybrid slime mould algorithm with adaptive guided differential evolution algorithm for combinatorial and global optimization problems. *Expert Syst. Appl.* **2021**, *174*, 114689. [CrossRef]
47. Guo, M.-W.; Wang, J.-S.; Yang, X. An chaotic firefly algorithm to solve quadratic assignment problem. *Eng. Lett.* **2020**, *28*, 337–342.
48. Rizk-Allah, R.M.; Slowik, A.; Darwish, A.; Hassanien, A.E. Orthogonal Latin squares-based firefly optimization algorithm for industrial quadratic assignment tasks. *Neural Comput. Appl.* **2021**, *33*, 16675–16696. [CrossRef]
49. Abdel-Baset, M.; Wu, H.; Zhou, Y.; Abdel-Fatah, L. Elite opposition-flower pollination algorithm for quadratic assignment problem. *J. Intell. Fuzzy Syst.* **2017**, *33*, 901–911. [CrossRef]
50. Dokeroglu, T. Hybrid teaching–learning-based optimization algorithms for the quadratic assignment problem. *Comp. Ind. Eng.* **2015**, *85*, 86–101. [CrossRef]

51. Riffi, M.E.; Sayoti, F. Hybrid algorithm for solving the quadratic assignment problem. *Int. J. Interact. Multimed. Artif. Intell.* **2017**, *5*, 68–74. [CrossRef]
52. Semlali, S.C.B.; Riffi, M.E.; Chebihi, F. Parallel hybrid chicken swarm optimization for solving the quadratic assignment problem. *Int. J. Electr. Comput. Eng.* **2019**, *9*, 2064–2074. [CrossRef]
53. Badrloo, S.; Kashan, A.H. Combinatorial optimization of permutation-based quadratic assignment problem using optics inspired optimization. *J. Appl. Res. Ind. Eng.* **2019**, *6*, 314–332. [CrossRef]
54. Kiliç, H.; Yüzgeç, U. Tournament selection based antlion optimization algorithm for solving quadratic assignment problem. *Eng. Sci. Technol. Int. J.* **2019**, *22*, 673–691. [CrossRef]
55. Kiliç, H.; Yüzgeç, U. Improved antlion optimization algorithm for quadratic assignment problem. *Malayas. J. Comput. Sci.* **2021**, *34*, 34–60. [CrossRef]
56. Ng, K.M.; Tran, T.H. A parallel water flow algorithm with local search for solving the quadratic assignment problem. *J. Ind. Manag. Optim.* **2019**, *15*, 235–259. [CrossRef]
57. Kumar, M.; Sahu, A.; Mitra, P. A comparison of different metaheuristics for the quadratic assignment problem in accelerated systems. *Appl. Soft Comput.* **2021**, *100*, 106927. [CrossRef]
58. Yadav, A.A.; Kumar, N.; Kim, J.H. Development of discrete artificial electric field algorithm for quadratic assignment problems. In Proceedings of the 6th International Conference on Harmony Search, Soft Computing and Applications. ICHSA 2020. Advances in Intelligent Systems and Computing, Istanbul, Turkey, 16–17 July 2020; Nigdeli, S.M., Kim, J.H., Bekdaş, G., Yadav, A., Eds.; Springer: Singapore, 2021; Volume 1275, pp. 411–421. [CrossRef]
59. Dokeroglu, T.; Ozdemir, Y.S. A new robust Harris Hawk optimization algorithm for large quadratic assignment problems. *Neural Comput. Appl.* **2023**, *35*, 12531–12544. [CrossRef]
60. Acan, A.; Ünveren, A. A great deluge and tabu search hybrid with two-stage memory support for quadratic assignment problem. *Appl. Soft Comput.* **2015**, *36*, 185–203. [CrossRef]
61. Chmiel, W.; Kwiecień, J. Quantum-inspired evolutionary approach for the quadratic assignment problem. *Entropy* **2018**, *20*, 781. [CrossRef]
62. Drezner, Z. Taking advantage of symmetry in some quadratic assignment problems. *INFOR Inf. Syst. Oper. Res.* **2019**, *57*, 623–641. [CrossRef]
63. Dantas, A.; Pozo, A. On the use of fitness landscape features in meta-learning based algorithm selection for the quadratic assignment problem. *Theor. Comput. Sci.* **2020**, *805*, 62–75. [CrossRef]
64. Öztürk, M.; Alabaş-Uslu, Ç. Cantor set based neighbor generation method for permutation solution representation. *J. Intell. Fuzzy Syst.* **2020**, *39*, 6157–6168. [CrossRef]
65. Alza, J.; Bartlett, M.; Ceberio, J.; McCall, J. Towards the landscape rotation as a perturbation strategy on the quadratic assignment problem. In Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '21, Lille, France, 10–14 July 2021; Chicano, F., Ed.; ACM: New York, NY, USA, 2021; pp. 1405–1413. [CrossRef]
66. Amirghasemi, M. An effective parallel evolutionary metaheuristic with its application to three optimization problems. *Appl. Intell.* **2023**, *53*, 5887–5909. [CrossRef]
67. Zhou, Y.; Hao, J.-K.; Duval, B. Frequent pattern-based search: A case study on the quadratic assignment problem. *IEEE Trans. Syst. Man Cybern. Syst.* **2022**, *52*, 1503–1515. [CrossRef]
68. Baldé, M.A.M.T.; Gueye, S.; Ndiaye, B.M. A greedy evolutionary hybridization algorithm for the optimal network and quadratic assignment problem. *Oper. Res.* **2021**, *21*, 1663–1690. [CrossRef]
69. Ni, Y.; Liu, W.; Du, X.; Xiao, R.; Chen, G.; Wu, Y. Evolutionary optimization approach based on heuristic information with pseudo-utility for the quadratic assignment problem. *Swarm Evol. Comput.* **2024**, *87*, 101557. [CrossRef]
70. Abdel-Basset, M.; Mohamed, R.; Saber, S.; Hezam, I.M.; Sallam, K.M.; Hameed, I.A. Binary metaheuristic algorithms for 0–1 knapsack problems: Performance analysis, hybrid variants, and real-world application. *J. King Saud Univ.-Comput. Inf. Sci.* **2024**, *36*, 102093. [CrossRef]
71. Alvarez-Flores, O.A.; Rivera-Blas, R.; Flores-Herrera, L.A.; Rivera-Blas, E.Z.; Funes-Lora, M.A.; Nino-Suárez, P.A. A novel modified discrete differential evolution algorithm to solve the operations sequencing problem in CAPP systems. *Mathematics* **2024**, *12*, 1846. [CrossRef]
72. El-Shorbagy, M.A.; Bouaouda, A.; Nabwey, H.A.; Abualigah, L.; Hashim, F.A. Advances in Henry gas solubility optimization: A physics-inspired metaheuristic algorithm with its variants and applications. *IEEE Access* **2024**, *12*, 26062–26095. [CrossRef]
73. Zhang, J.; Ye, J.-X.; Lin, J.; Song, H.-B. A discrete Jaya algorithm for vehicle routing problems with uncertain demands. *Syst. Sci. Control Eng.* **2024**, *12*, 2350165. [CrossRef]
74. Zhang, Y.; Xing, L. A new hybrid improved arithmetic optimization algorithm for solving global and engineering optimization problems. *Mathematics* **2024**, *12*, 3221. [CrossRef]
75. Zhao, S.; Zhang, T.; Cai, L.; Yang, R. Triangulation topology aggregation optimizer: A novel mathematics-based meta-heuristic algorithm for engineering applications. *Expert Syst. Appl.* **2024**, *238 Pt B*, 121744. [CrossRef]
76. Loiola, E.M.; De Abreu, N.M.M.; Boaventura-Netto, P.O.; Hahn, P.; Querido, T. A survey for the quadratic assignment problem. *Eur. J. Oper. Res.* **2007**, *176*, 657–690. [CrossRef]
77. Abdel-Basset, M.; Manogaran, G.; Rashad, H.; Zaied, A.N.H. A comprehensive review of quadratic assignment problem: Variants, hybrids and applications. *J. Amb. Intel. Hum. Comput.* **2018**, *9*, 1–24. [CrossRef]

78. Achary, T.; Pillay, S.; Pillai, S.M.; Mqadi, M.; Genders, E.; Ezugwu, A.E. A performance study of meta-heuristic approaches for quadratic assignment problem. *Concurr. Comput. Pract. Exp.* **2021**, *33*, 1–29. [CrossRef]
79. Hussin, M.S.; Stützle, T. Hierarchical iterated local search for the quadratic assignment problem. In *Hybrid Metaheuristics, HM 2009, Lecture Notes in Computer Science*; Blesa, M.J., Blum, C., Di Gaspero, L., Roli, A., Sampels, M., Schaerf, A., Eds.; Springer: Berlin/Heidelberg, Germany, 2009; Volume 5818, pp. 115–129. [CrossRef]
80. Battarra, M.; Benedettini, S.; Roli, A. Leveraging saving-based algorithms by master–slave genetic algorithms. *Eng. Appl. Artif. Intell.* **2011**, *24*, 555–566. [CrossRef]
81. Liu, S.; Xue, J.; Hu, C.; Li, Z. Test case generation based on hierarchical genetic algorithm. In Proceedings of the 2014 International Conference on Mechatronics, Control and Electronic Engineering, MEIC 2014, Shenyang, China, 15–17 November 2014; Atlantis Press: Dordrecht, The Netherland, 2014; pp. 278–281. [CrossRef]
82. Ahmed, A.K.M.F.; Sun, J.U. A novel approach to combine the hierarchical and iterative techniques for solving capacitated location-routing problem. *Cogent Eng.* **2018**, *5*, 1463596. [CrossRef]
83. Misevičius, A.; Palubeckis, G.; Drezner, Z. Hierarchicity-based (self-similar) hybrid genetic algorithm for the grey pattern quadratic assignment problem. *Memet. Comput.* **2021**, *13*, 69–90. [CrossRef]
84. Frieze, A.M.; Yadegar, J.; El-Horbaty, S.; Parkinson, D. Algorithms for assignment problems on an array processor. *Parallel Comput.* **1989**, *11*, 151–162. [CrossRef]
85. Merz, P.; Freisleben, B. Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Trans. Evol. Comput.* **2000**, *4*, 337–352. [CrossRef]
86. Li, Y.; Pardalos, P.M.; Resende, M.G.C. A greedy randomized adaptive search procedure for the quadratic assignment problem. In *Quadratic Assignment and Related Problems. DIMACS Series in Discrete Mathematics and Theoretical Computer Science*; Pardalos, P.M., Wolkowicz, H., Eds.; AMS: Providence, RI, USA, 1994; Volume 16, pp. 237–261.
87. Drezner, Z. Compounded genetic algorithms for the quadratic assignment problem. *Oper. Res. Lett.* **2005**, *33*, 475–480. [CrossRef]
88. Berman, O.; Drezner, Z.; Krass, D. Discrete cooperative covering problems. *J. Oper. Res. Soc.* **2011**, *62*, 2002–2012. [CrossRef]
89. Tate, D.M.; Smith, A.E. A genetic approach to the quadratic assignment problem. *Comput. Oper. Res.* **1995**, *22*, 73–83. [CrossRef]
90. Misevičius, A.; Kuznecovaitė, D.; Platužienė, J. Some further experiments with crossover operators for genetic algorithms. *Informatica* **2018**, *29*, 499–516. [CrossRef]
91. Sivanandam, S.N.; Deepa, S.N. *Introduction to Genetic Algorithms*; Springer: Heidelberg, Germany; New York, NY, USA, 2008.
92. Smyth, K.; Hoos, H.H.; Stützle, T. Iterated robust tabu search for MAX-SAT. In *Advances in Artificial Intelligence: Proceedings of the 16th Conference of the Canadian Society for Computational Studies of Intelligence, Halifax, NS, Canada, 11–13 June 2003.*; Lecture Notes in Artificial Intelligence; Xiang, Y., Chaib-Draa, B., Eds.; Springer: Berlin/Heidelberg, Germany, 2003; Volume 2671, pp. 129–144. [CrossRef]
93. 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]
94. Dell'Amico, M.; Trubian, M. Solution of large weighted equicut problems. *Eur. J. Oper. Res.* **1998**, *106*, 500–521. [CrossRef]
95. Durmaz, E.D.; Şahin, R. An efficient iterated local search algorithm for the corridor allocation problem. *Expert Syst. Appl.* **2023**, *212*, 118804. [CrossRef]
96. Polat, O.; Kalayci, C.B.; Kulak, O.; Günther, H.-O. A perturbation based variable neighborhood search heuristic for solving the vehicle routing problem with simultaneous pickup and delivery with time limit. *Eur. J. Oper. Res.* **2015**, *242*, 369–382. [CrossRef]
97. Lu, Z.; Hao, J.-K.; Zhou, Y. Stagnation-aware breakout tabu search for the minimum conductance graph partitioning problem. *Comput. Oper. Res.* **2019**, *111*, 43–57. [CrossRef]
98. Sadati, M.E.H.; Çatay, B.; Aksen, D. An efficient variable neighborhood search with tabu shaking for a class of multi-depot vehicle routing problems. *Comput. Oper. Res.* **2021**, *133*, 105269. [CrossRef]
99. Qin, T.; Peng, B.; Benlic, U.; Cheng, T.C.E.; Wang, Y.; Lü, Z. Iterated local search based on multi-type perturbation for single-machine earliness/tardiness scheduling. *Comput. Oper. Res.* **2015**, *61*, 81–88. [CrossRef]
100. Canuto, S.A.; Resende, M.G.C.; Ribeiro, C.C. Local search with perturbations for the prize-collecting Steiner tree problem in graphs. *Networks* **2001**, *38*, 50–58. [CrossRef]
101. Shang, Z.; Zhao, S.; Hao, J.-K.; Yang, X.; Ma, F. Multiple phase tabu search for bipartite boolean quadratic programming with partitioned variables. *Comput. Oper. Res.* **2019**, *102*, 141–149. [CrossRef]
102. Lai, X.; Hao, J.-K. Iterated maxima search for the maximally diverse grouping problem. *Eur. J. Oper. Res.* **2016**, *254*, 780–800. [CrossRef]
103. Ren, J.; Hao, J.-K.; Rodriguez-Tello, E.; Li, L.; He, K. A new iterated local search algorithm for the cyclic bandwidth problem. *Knowl.-Based Syst.* **2020**, *203*, 106136. [CrossRef]
104. Avci, M. An effective iterated local search algorithm for the distributed no-wait flowshop scheduling problem. *Eng. Appl. Artif. Intell.* **2023**, *120*, 105921. [CrossRef]
105. Lai, X.; Hao, J.-K. Iterated variable neighborhood search for the capacitated clustering problem. *Eng. Appl. Artif. Intell.* **2016**, *56*, 102–120. [CrossRef]
106. Li, R.; Hu, S.; Wang, Y.; Yin, M. A local search algorithm with tabu strategy and perturbation mechanism for generalized vertex cover problem. *Neural Comput. Appl.* **2017**, *28*, 1775–1785. [CrossRef]
107. Lévy, P. *Théorie de l'Addition des Variables Aléatoires*; Gauthier-Villars: Paris, France, 1937.

108. Pavlyukevich, I. Lévy flights, non-local search and simulated annealing. *J. Comput. Phys.* **2007**, *226*, 1830–1844. [CrossRef]
109. Chambers, J.M.; Mallows, C.L.; Stuck, B.W. A method for simulating stable random variables. *J. Am. Stat. Assoc.* **1976**, *71*, 340–344. [CrossRef]
110. Mantegna, R.N. Fast, accurate algorithm for numerical simulation of Lévy stable stochastic processes. *Phys. Rev. E* **1994**, *49*, 4677–4683. [CrossRef]
111. Nemes, G. New asymptotic expansion for the Gamma function. *Arch. Math.* **2010**, *95*, 161–169. [CrossRef]
112. Li, W.; Li, H.; Wang, Y.; Han, Y. Optimizing flexible job shop scheduling with automated guided vehicles using a multi-strategy-driven genetic algorithm. *Egypt. Inform. J.* **2024**, *25*, 100437. [CrossRef]
113. López-Ibáñez, M.; Mascia, F.; Marmion, M.-E.; Stützle, T. A template for designing single-solution hybrid metaheuristics. In Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation, GECCO Comp '14, Vancouver, BC, Canada, 12–16 July 2014; Igel, C., Ed.; ACM: New York, NY, USA, 2014; pp. 1423–1426. [CrossRef]
114. Meng, W.; Qu, R. Automated design of search algorithms: Learning on algorithmic components. *Expert Syst. Appl.* **2021**, *185*, 115493. [CrossRef]
115. Burkard, R.E.; Karisch, S.; Rendl, F. QAPLIB—A quadratic assignment problem library. *J. Glob. Optim.* **1997**, *10*, 391–403. [CrossRef]
116. QAPLIB—A Quadratic Assignment Problem Library—COR@L. Available online: https://coral.ise.lehigh.edu/data-sets/qaplib/ (accessed on 10 November 2023).
117. De Carvalho, S.A., Jr.; Rahmann, S. Microarray layout as a quadratic assignment problem. In *German Conference on Bioinformatics, GCB 2006, Lecture Notes in Informatics—Proceedings*; Huson, D., Kohlbacher, O., Lupas, A., Nieselt, K., Zell, A., Eds.; Gesellschaft für Informatik: Bonn, Germany, 2006; Volume P-83, pp. 11–20.
118. Drezner, Z.; Hahn, P.M.; Taillard, E.D. Recent advances for the quadratic assignment problem with special emphasis on instances that are difficult for metaheuristic methods. *Ann. Oper. Res.* **2005**, *139*, 65–94. [CrossRef]
119. Taillard—QAP. Available online: http://mistic.heig-vd.ch/taillard/problemes.dir/qap.dir/qap.html (accessed on 30 November 2023).
120. Misevičius, A. Letter: New best known solution for the most difficult QAP instance "tai100a". *Memet. Comput.* **2019**, *11*, 331–332. [CrossRef]
121. Drezner, Z. Tabu search and hybrid genetic algorithms for quadratic assignment problems. In *Tabu Search*; Jaziri, W., Ed.; In-Tech: London, UK, 2008; pp. 89–108. [CrossRef]
122. Drezner, Z.; Marcoulides, G.A. On the range of tabu tenure in solving quadratic assignment problems. In *Recent Advances in Computing and Management Information Systems*; Athens Institute for Education and Research: Athens, Greece, 2009; pp. 157–168.
123. Drezner, Z.; Misevičius, A. Enhancing the performance of hybrid genetic algorithms by differential improvement. *Comput. Oper. Res.* **2013**, *40*, 1038–1046. [CrossRef]
124. Rodriguez, J.M.; MacPhee, F.C.; Bonham, D.J.; Horton, J.D.; Bhavsar, V.C. Best permutations for the dynamic plant layout problem. In *Efficient and Experimental Algorithms: Proceedings of the 12th International Conference on Advances in Computing and Communications (ADCOM 2004), Ahmedabad, India, 15–18 December 2004*; Dasgupta, A.R., Iyengar, S.S., Bhatt, H.S., Eds.; Allied Publishers Pvt. Ltd.: New Delhi, India; Ahmedabad, India, 2004; pp. 173–178.