



# A novel multistart hyper-heuristic algorithm on the grid for the quadratic assignment problem



Tansel Dokeroglu<sup>a,\*</sup>, Ahmet Cosar<sup>b</sup>

<sup>a</sup> Computer Engineering Department, Turkish Aeronautical Association University, Ankara, Turkey

<sup>b</sup> Computer Engineering Department, Middle East Technical University, Universities Street, Ankara, Turkey

## ARTICLE INFO

### Article history:

Received 28 August 2015

Received in revised form

24 December 2015

Accepted 7 February 2016

Available online 24 February 2016

### Keywords:

Hyper-heuristics

Assignment

Simulated annealing

Tabu search

Ant colony

Breakout local search

## ABSTRACT

Hyper-heuristics introduce novel approaches for solving challenging combinatorial optimization problems by operating over a set of low level (meta)-heuristics. This is achieved by an evolutionary selection mechanism that controls and combines the strengths of the low level (meta)-heuristics. In this study, we propose a high-performance MultiStart Hyper-heuristic algorithm (MSH-QAP) on the grid for the solution of the Quadratic Assignment Problem (QAP). MSH-QAP algorithm makes use of state-of-the-art (meta)-heuristics, Simulated Annealing (SA), Robust Tabu Search (RTS), Ant Colony Optimization (FANT), and Breakout Local Search (BLS) that have been reported among the best performing algorithms for the solution of difficult QAP instances in standard benchmark libraries. In the first phase of the algorithm, the most appropriate (meta)-heuristic with its near-optimal parameter settings is selected by using a genetic algorithm optimization layer that uses a self-adaptive parameter setting method for the given problem instance. In the second phase, if an optimal solution cannot be found, selected best performing (meta)-heuristic (with its finely adjusted parameter settings) is executed on the grid using parallel processing and performing several multistarts in order to increase the quality of the discovered solution. MSH-QAP algorithm is tested on 134 problem instances of the QAPLIB benchmark and is shown to be able to solve 122 of the instances exactly. The overall deviation for the problem instances is obtained as 0.013% on the average.

© 2016 Elsevier Ltd. All rights reserved.

## 1. Introduction

The Quadratic Assignment Problem (QAP) is an NP-Hard combinatorial optimization problem introduced by Koopmans and Beckmann in 1957 to model the location selection problem of indivisible economic activities (Koopmans and Beckmann, 1957). Although facility location is the most popular form of the QAP, traveling salesman, bin-packing, maximum clique, scheduling, the graph-partitioning problem, statistical data analysis, minimum-inbreeding seed orchard layout, signal processing, transportation systems, typewriter keyboard design, layout design, backboard wiring, and data allocation are among the possible applications of the QAP (Lstiburek et al., 2015; Burkard et al., 1991; Steinberg, 1961; Rossin et al., 1999; Pfister, 1998; Dokeroglu, 2015).

The QAP is the problem of assigning facilities to locations with a varying installation costs for each location. The objective of the problem is to find an allocation such that the total cost is of installation and transporting required amounts of materials between the facilities is minimized. The QAP can be formally modeled by using three  $n \times n$  matrices,  $A$ ,  $B$ , and  $C$ :

$$A = (a_{ik}) \quad (1)$$

where  $a_{ik}$  is the flow amount from facility  $i$  to facility  $k$ .

$$B = (b_{jl}) \quad (2)$$

\* Corresponding author.

E-mail address: [tansel@ceng.metu.edu.tr](mailto:tansel@ceng.metu.edu.tr) (T. Dokeroglu).

where  $b_{jl}$  is the distance (i.e, the transportation cost) from location  $j$  to location  $l$ .

$$C = (c_{ij}) \quad (3)$$

where  $c_{ij}$  is the cost of placing facility  $i$  at location  $j$ .

The Koopmans–Beckmann form of the QAP can be written as:

$$\min_{\phi \in S_n} \left( \sum_{i=1}^n \sum_{k=1}^n a_{ik} b_{\phi(i)\phi(k)} + \sum_{i=1}^n c_{i\phi(i)} \right) \quad (4)$$

where  $S_n$  is permutation of integers  $1, 2, \dots, n$ . Each term  $a_{ik} b_{\phi(i)\phi(k)}$  is the transportation cost from facility  $i$  at location  $\phi(i)$  to facility  $k$  at location  $\phi(k)$ . Each term  $c_{i\phi(i)}$  is the total cost for installing facility  $i$ , at location  $\phi(i)$ , plus the transportation costs to all other facilities  $k$ , installed at locations  $\phi(1), \phi(2), \dots, \phi(n)$  (the range of the indexes  $i, j, k, l$ , is  $1, \dots, n$ ). The QAP  $(A, B)$  is an instance where  $A, B$ , and  $C$  are input matrices given with Eqs. (1), (2), (3). If there is no  $C$  term, we can write it as a QAP  $(A, B)$ .

The QAP instances larger than size 35 cannot be solved with exact algorithms due to the computational limitations (Loiola et al., 2007). (Meta)-heuristic approaches produce high-quality solutions under these conditions with their high performances (Tseng and Liang, 2005; Drezner, 2005; Fescioglu-Unver and Kokar, 2011; Misevičius, 2012; Duman et al., 2012). Genetic Algorithms (GA) (Tosun et al., 2013), Simulated Annealing (SA) (Connolly, 1990), Ant Colony Optimization (Dorigo and Di Caro, 1999; Gambardella et al., 1999), Tabu Search (TS) (Glover, 1990; Taillard, 1991), and Breakout Local Search (Benlic and Hao, 2013) are some of these well-known methods that have been successfully applied to the QAP.

In this study, we propose a novel two-phase high-performance Multistart Hyper-heuristic Algorithm (MSH-QAP) on the grid for the QAP. MSH-QAP makes use of an emerging approach, hyper-heuristics, a selection mechanism that controls and combines the strengths of several heuristics to find the best solution for an optimization problem (Burke et al., 2013). Due to the No Free Lunch (NFL) theorem, heuristics do not demonstrate the same performances when the domain and/or the structure of the problem is changed (Wolpert and Macready, 1997). We think that applying more than a single heuristic to the same problem with parallel computation is a good idea where most probably one of the proposed heuristics will have a better performance than the others. MSH-QAP algorithm uses the state-of-the-art (meta)-heuristics, SA, Robust Tabu Search (RTS), Fast Ant System (FAnt), and Breakout Local Search (BLS) that have been reported among the best performing algorithms for large problem instances of the QAP (Taillard, 1998; Benlic and Hao, 2013). MSH-QAP is a two phase algorithm. In the first phase of the algorithm, a GA layer selects the best heuristic and tunes the parameters of the selected heuristic adaptively while trying to find the best solution for the given QAP. If an optimal solution is not found in the first phase, then in the second phase, the selected best heuristic is run on several processors by applying a multistart technique. This phase of the algorithm behaves as a stagnation prevention mechanism and restarts the exploration of the search space from different starting points and attempts to further improve the quality of the solutions. MSH-QAP also benefits from the high performance capabilities of a parallel computing environment by running the time-consuming calculations of each heuristic on a different processor. It explores the search space and uses the delta calculation approach for the fitness evaluation of the neighbors, which is a very efficient way of reducing the computation time (Misevičius, 2012).

We can summarize the contributions of our study as follows. A novel parallel multistart hyper-heuristic algorithm is proposed for the intractable QAP. The proposed MSH-QAP algorithm significantly reduces the total execution time of the optimization while improving the solution quality of the problems. The reduction is obtained through parallel execution of the heuristics on the grid. State-of-the-art heuristics SA, RTA, FAnt, and BLS are used as low-level heuristics for the solution of the QAP. The MSH-QAP algorithm has an adaptive parameter setting mechanism for each heuristic with respect to the given problem instance. MSH-QAP algorithm obtains 122 of the benchmark problems optimally while producing only 0.013% deviation from the best known results for the remaining 12 problems.

In Section 2, related studies for the state-of-the-art hyper-heuristic algorithms and the QAP are given. Our proposed algorithm, MSH-QAP, new genetic operators, the low-level heuristics SA, RTS, FAnt, and BLS that are used in the proposed algorithm are briefly explained in Section 3. The setup of the experimental environment, obtained results, and comparison with state-of-the-art (meta)-heuristics are presented in Section 4. Concluding remarks are provided in the last section.

## 2. Related work

Several optimization problems have been reported to be successfully solved by using hyper-heuristics. In this section, we give information about the most important ones that are related with combinatorial optimization problems. We also give short information about Genetic Algorithms (GA) and the state-of-the-art QAP solution algorithms. Finally we make a comparison between the existing solutions and our proposed algorithm.

The term hyper-heuristic was first described as a technique to combine different artificial intelligence techniques for improving the performance of automated theorem proving systems in 1997 (Cowling et al., 2000). Contemporary use of hyper-heuristics involves a set of (meta)-heuristics that are used for solving NP-hard search problems. Heuristics are selected and adapted for each problem instance, automatically, by using a selection algorithm. By using hyper-heuristic techniques, general applicability of heuristic search methods is improved without requiring intervention by a human expert to adjust the parameters of employed search heuristics (Burke et al., 2013, 2003, 2010; Ochoa et al., 2012). The hyper-heuristics differ from (meta)-heuristics by performing a search within the search space of heuristics, while (meta)-heuristics search within the space of problem solutions. Thus, a hyper-heuristic works by selecting the best search method or sequence of heuristics for a given problem instance (Burke et al., 2013; Ochoa et al., 2012; Ryser-Welch and Miller, 2014).

Burke et al. report on two distinct timetabling and rostering problems using three randomly prepared sets of problem instances and investigate the performance of a tabu-search based hyper-heuristic (Burke et al., 2003). This study uses reinforcement learning principles to come up with rules that will competitively evaluate and choose the most preferable heuristic. In order to prevent some heuristics from being chosen for a certain period a tabu list of heuristics is maintained. The work successfully shows that for various problem instances acceptable quality solutions can be obtained by using a tabu-search hyper-heuristic. Burke et al. propose a study of a simple generic

hyper-heuristic on a set of constructive graph coloring heuristics for the timetabling problem. The proposed technique searches for permutations of exam and course timetabling heuristics combinations also making use of a tabu search approach (Burke et al., 2007).

The frequency assignment problem (FAP) in TD-SCDMA network of mobile communications industry is studied by Yang et al. (2014). The authors define six low-level heuristics (LLHs) search strategies based on the computation of interference, and then use genetic algorithm (GA) at a high-level to find the best combination sequence of LLH strategies to reduce interferences of the overall network. GA uses two-point crossover, uniform mutation, and Minimal Generation Gap (MGG) as the generation alternation model. This high level approach controls and combines the strengths of three well-known multi-objective evolutionary algorithms by utilizing them as the low level heuristics (Maashi et al., 2014). This study has some similarities with our approach in a sense that GA is also used in our proposed algorithm as a heuristic selection layer but the solved problems are different.

An investigation of a simple generic hyper-heuristic approach for a set of widely used constructive heuristics (graph coloring heuristics) in timetabling is proposed by Burke et al. (2007). In this study, a tabu search approach is employed to search for permutations of graph heuristics which are used for constructing timetables in exam and course timetabling problems. Beyaz et al. propose a set of novel hyper-heuristic algorithms that select/combine the state-of-the-art heuristics and local search techniques for minimizing the number of 2D bins (Beyaz et al., 2015). A historical perspective on automated algorithm design, and similarities and differences between meta-learning in the field of supervised machine learning (classification) and hyper-heuristics in the field of optimization is provided by Pappa et al. (2014).

A set of approaches used to deal with the frequency assignment problem (FAP) is proposed for the design of GSM networks (Segura et al., 2011). The formulation of FAP focuses on aspects of real-world GSM networks. A memetic algorithm including local search and variation operators is presented. A parallel hyper-heuristic-based model is used to parallelize the approach and to avoid the requirement of the adaptation step of the memetic algorithm. The model is a hybrid and combines a parallel island-based scheme with a hyper-heuristic approach. From this point of view, our approach is not an island model.

A hyper-heuristic approach based on genetic programming, which is called as genetic hyper-heuristic, is introduced for evolving an enhanced version of the Sloan algorithm (Koohestani and Poli, 2014). A hyper-heuristic algorithm by Salcedo et al. is designed for solving the Jawbreaker puzzle (Salcedo-Sanz et al., 2014). Topcuoglu proposed a novel hybrid strategy by integrating memory/search algorithms with hyper-heuristic techniques on dynamic environments (Topcuoglu et al., 2014). Sim et al. proposed a continuously learning system based on hyper-heuristics in order to solve a combinatorial optimization problem. This system is inspired by artificial immune system methods and works by generating new heuristics and sampling problems in its environment (Sim et al., 2015).

A parallel hyper-heuristic approach for two-dimensional rectangular strip-packing problems is proposed by Borgulya (2014). This is an island model with a master-slave structure. All the islands run a memetic algorithm-based hyper-heuristic. The basic technique uses Extended Virtual Loser (EVL). The memory-based technique memorizes the past events. It can influence the operations of the evolutionary algorithms using the memory. The EVL technique learns the bad values of the variables based on the worst solutions of the population and computes probabilities to control the mutation steps.

GAs are used as a heuristic selection and parameter tuning layer in our proposed algorithm. GAs have been used to solve search and optimization problems (Holland, 1975). They use a computational model similar to the natural processes of selection and evolution. Individuals with better quality have more probability of surviving. GAs can perform efficient search operations in several problem areas. A random population is generated and by applying selection, crossover, and mutation operations, GAs create new solutions through the generations (Goldberg, 1989). The best individual becomes the solution of the problem. GAs select the individuals to crossover or mutate. Tournament, roulette wheel, and truncation are the most frequently used selection methods in this process.

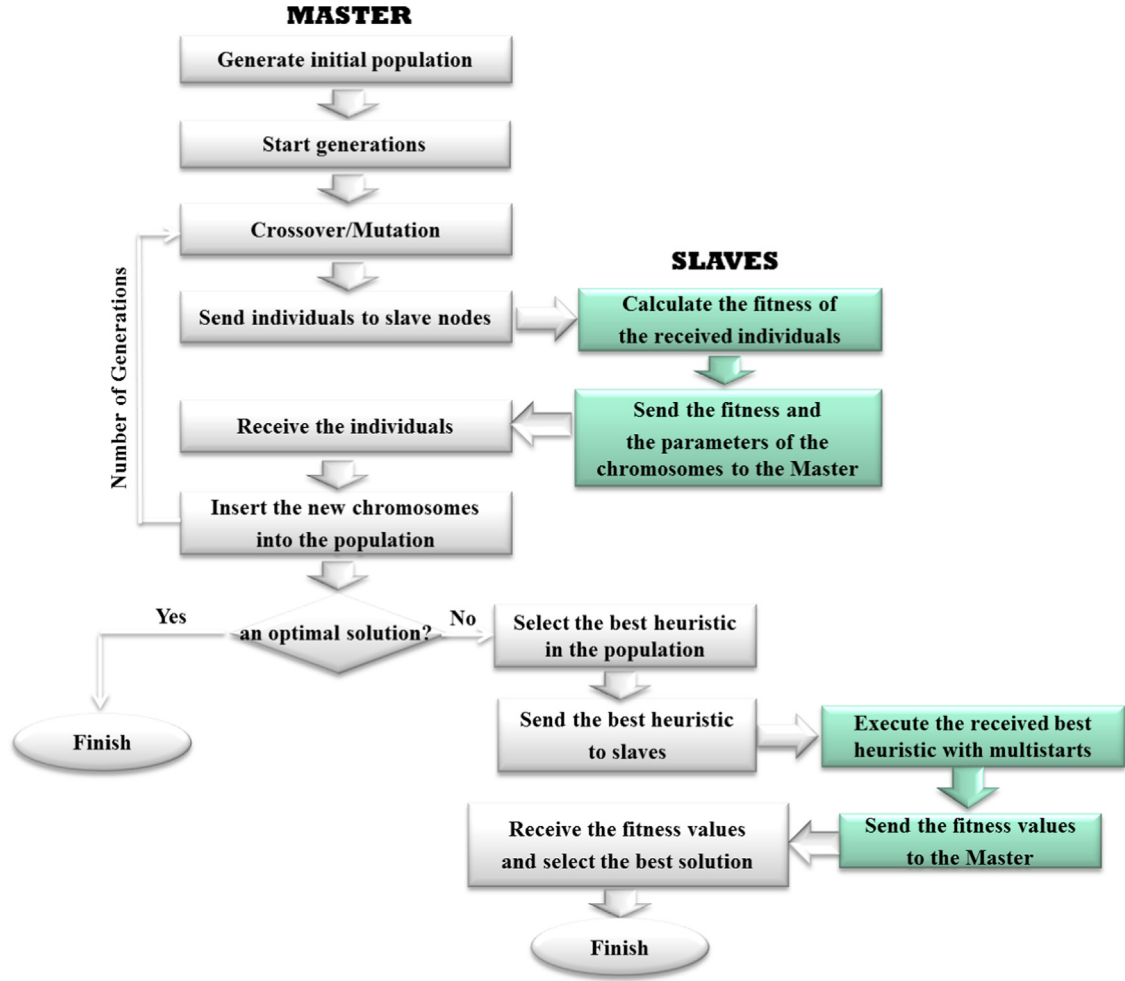
Several algorithms have been proposed for both exact and approximate solutions of the QAP. Exact algorithms are limited to solving small data sets of the QAP (up to problem size 35) with massively parallel computers, whereas (meta)-heuristics such as SA, RTS, FAnt, and BLS can provide near-optimal solutions within reasonable optimization times for larger problem instances (Dokeroglu, 2015; Benlic and Hao, 2015; Acan and Unveren, 2015). An efficient parallel hybrid genetic algorithm (HGA) is proposed by Tosun (2015). Battiti and Tecchiolli proposed a PGA for the QAP (Battiti and Tecchiolli, 1992). This PGA subdivides the population of solutions and migration of solutions among the processors occurs during the execution. Detailed information about these algorithms can be found in Crainic and Toulouse (2003). James et al. introduced a cooperative parallel TS algorithm (CPTS) for the QAP (James et al., 2009). The CPTS provides high-quality solutions for the problem instances in QAPLIB with acceptable computational times. A detailed survey about the classical QAP optimization algorithms can be found in Loiola et al. (2007).

To the best of our knowledge, our proposed algorithm (MSH-QAP) is the first parallel hyper-heuristic algorithm in the literature that has been developed for the solution of the QAP so far. We use four different low-level heuristics and implement a GA layer on the master node with a global population to select the heuristics and tune their parameters. Parallel MSH-QAP significantly reduces the overall execution time of the hyper-heuristic algorithms and provides a high performance means of exploring NP-Hard optimization problems. Several heuristics are imported into MSH-QAP with respect to the NFL theory (Wolpert and Macready, 1997). Low-level heuristics that are used by MSH-QAP algorithm are explained in the third section of our study.

### 3. Multistart hyper-heuristic algorithm (MSH-QAP)

In this section, we introduce our proposed algorithm, Multistart Hyper-heuristic Algorithm for the QAP (MSH-QAP). There exist two phases for the execution of the MSH-QAP algorithm. In the first phase, a GA is run on the grid and parameters of the heuristics are optimized and the best heuristic is decided among RTS, SA, FAnt, and BLS for the given problem instance. The solution quality is improved by using several heuristic mechanisms on a parallel computation environment. The execution of these heuristics is a time consuming process therefore, all of the fitness evaluations of the chromosomes are calculated on a separate slave node by using MPI communication paradigm. If the solution found by the first phase of the MSH-QAP is not an optimal one with respect to the solutions in the QAP benchmark library then the second phase of the MSH-QAP is started. In this phase, the best selected heuristic and its near-optimal parameter settings are given to all of the slave nodes and executed with multistarts.

The flowchart of the MSH-QAP can be found in Fig. 1. The MSH-QAP pseudocode for the master and slave nodes can be found in Algorithms 1 and 2 respectively.



**Fig. 1.** Flowchart of the MSH-QAP algorithm (green boxes are executed at the slaves). (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

**Algorithm 1.** Pseudocode for the master node of multistart hyper-heuristic algorithm (MSH-QAP).

```

1 /*First Phase*/
2  $p$ : population;
3 generate_initial_population( $p$ );
4 for  $i \leftarrow 1$  to number_of_generations do
5   for  $k \leftarrow 1$  to number_of_slaves do
6     crossover( $p$ );
7     mutation( $p$ );
8     send offspring chromosome to slave $k$  // to execute the selected heuristic at the slave side
9     receive calculated offspring chromosome from slave $k$ 
10    insert offspring chromosome into the population if better than any individual

11 /*Second Phase*/
12 if (the fitness value of the best individual is not the same with the fitness value of an optimal )
13 for  $k \leftarrow 1$  to number_of_slaves do
14   send the selected best chromosome to slave $k$  // to run the best heuristic with multistarts
15   receive the calculated chromosome from slave $k$ 
16 report the best chromosome sent by the slaves as the best solution
  
```

**Algorithm 2.** Pseudocode for the slave nodes of multistart hyper-heuristic algorithm (MSH-QAP).

```

1 /*First Phase*/
2 for  $i \leftarrow 1$  to number_of_generations do
3     receive chromosome from Master Node
4     calculate the fitness of the received chromosome
5     send the calculated chromosome to the Master Node

6 /*Second Phase*/
7 if (the fitness value of the best individual is not the same with the fitness value of an optimal )
8     receive the currently best available heuristic and its parameters from Master Node
9     iterator  $i \leftarrow 0$ ;
10 while ( $i++ < \text{number\_of\_multistarts}$ ) do
11     execute the meta-heuristic encoded in the selected chromosome with well-tuned parameters
12 send the best discovered solution to Master Node
    
```

### 3.1. Chromosome structure

The chromosome structure of the MSH-QAP algorithm consists of three segments (see Fig. 2). The leftmost segment of the chromosome keeps control of the solution vector for the QAP. The segment in the middle is the heuristic part that holds the information of search heuristic to be applied to the solution vector (SA, RTS, FAnt, and BLS are the alternatives for this segment). The rightmost segment contains the parameters of the selected heuristic. Parameters of the heuristics such as number of failures and tabu list size are the data preserved in this field in accordance with the type of the chosen heuristic. Parameters segment of the chromosome is a flexible array therefore it is adaptive and can handle different numbers of parameters depending on the type of selected heuristic.

For example, if the chromosome represents a robust tabu search heuristic then the leftmost segment is the permutation of the QAP, the second segment is a string (“robust tabu search”), and the rightmost one is an array of size four that holds parameters such as (5,000\*n, 9\*n, 11\*n, n\*n) where  $n$  is the size of the QAP.

### 3.2. Crossover and mutation operators

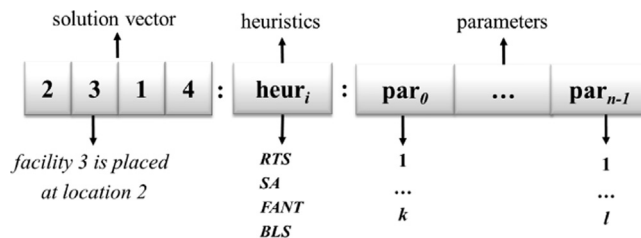
Crossover and mutation are the main operators of a GA to explore a search space of a given problem. For every chromosome, we randomly generate a solution vector before applying the heuristic. We use crossover and mutation operators for selecting the heuristics and setting the parameters. The use of the heuristics on the population is evenly distributed, which means that in a population with 24 individuals, we produce the same number of individuals for each heuristic (6 individuals for each heuristic SA, RTS, FAnt, and BLS) and keep this distribution until the end of the generations by applying selection, crossover, and mutation on the same chromosomes.

The crossover operator works on the parameters segment by cutting the rightmost 50% part of the segment. Mutation operator randomly replaces the parameters within the given boundaries. As it can be seen in Fig. 2, the parameters can be chosen between  $[1 \dots k]$ . Elitist selection is used at the end of each generation. A child chromosome with a better fitness value than an existing similar kind of heuristic in the population is replaced (see Figs. 3 and 4).

Various parameters of the heuristics used in MSH-QAP algorithm need to be tuned well for its efficiency (Grefenstette, 1986). Therefore, an adaptive parameter setting mechanism (layer) is used for MSH-QAP algorithm in order to explore the search space in a more effective way. The crossover and mutation operators of the MSH-QAP algorithm try to tune the near-optimal parameter settings for the given heuristics while optimizing the solution of the problem. GA is applied in the first phase of the MSH-QAP algorithm.

### 3.3. Fast evaluation of new permutations

Exchanging two units of an existing solution and producing a new permutation is proved to be the best way to explore the search space of the QAP. This approach allows us to calculate the cost of the new permutation in a very fast manner by only finding the difference with the former solution (Taillard, 1991). This method is used as a performance increasing calculation method in our proposed algorithm, MSH-



**Fig. 2.** Chromosome structure for the MSH-QAP algorithm.



QAP. Starting from a solution  $\phi$ , a neighbor solution  $\pi$  is obtained by permuting units  $r$  and  $s$ :

$$\begin{aligned}\pi(k) &= \phi(k) \forall k \neq r, s \\ \pi(r) &= \phi(s) \\ \pi(s) &= \phi(r)\end{aligned}\tag{5}$$

If the matrices are symmetrical, the value of a move,  $\Delta(\phi, r, s)$  is:

$$\Delta(\phi, r, s) = \sum_{i=1}^n \sum_{j=1}^n (a_{ij} b_{\phi(i)\phi(j)} - a_{ij} b_{\pi(i)\pi(j)}) = 2 \cdot \sum_{k \neq r, s} (a_{sk} - a_{rk})(b_{\phi(s)\phi(k)} - b_{\phi(r)\phi(k)})\tag{6}$$

### 3.4. Communication topology of the algorithm

MSH-QAP algorithm uses a master and slave communication topology (star) during the generations. The master node keeps the population and sends the heuristics to be executed to the slaves. In addition to the type of the heuristic to be run on the slave, the message also keeps the parameters of the executed heuristic. After the execution of the heuristic by the slave node, the master node receives the solution and inserts the information of the new chromosome to the population if it is better than any existing solution (see also Fig. 1). Through the generations, the parameters of the heuristics and the solution quality of the problem are improved with this method. Fig. 5 presents the master–slave (star) communication topology of the MSH-QAP algorithm.

### 3.5. Low-level heuristics used in the MSH-QAP algorithm

This section gives brief information about the (meta)-heuristics used by our proposed algorithm, MSH-QAP. The (meta)-heuristics are the algorithms, Simulated Annealing (SA), Robust Tabu Search (RTS), Fast Ant System (FANT), and Breakout Local Search (BLS) that have been reported to be among the best performing ones for the solution of the QAP. Although only these four algorithms are used in the MSH-QAP algorithm, there is no constraint for the number of heuristics that MSH-QAP algorithm can import into the search process.

#### 3.5.1. Simulated annealing

Simulated Annealing (SA) is inspired from the annealing process in metallurgy (Wilhelm and Ward, 1987; Kirkpatrick et al., 1983; Chib and Greenberg, 1995). If metal cools very fast, the atoms cannot get chance to produce a powerful lattice and are settled in a random manner, resulting in a fragile metal. If the temperature is decreased very slowly, the atoms have more time to construct strong crystals and this increases the quality of the product. The SA algorithm implemented for the QAP uses a temperature reheating technique during the execution (Hussin and Sttzle, 2014). The temperature reheating is intended to allow the algorithm to escape from particular local minima regions when the temperature reaches to a very low value. This permutation is the current best solution  $\pi_{best}$  and current best evaluation is  $f(\pi_{best})$ . The initial temperature is  $T_0 = 0.005 \cdot f(\pi_{best})$ . In function *GenerateRandomNeighbor*( $\pi$ ), a neighbor  $\pi'$  of  $\pi$  is obtained by swapping the position of two items  $i$  and  $j$ . Pair-swaps are applied sequentially for each solution  $\pi$  for all  $k = 1, \dots, n$ .

All moves that improve the current solution are accepted. Moves that worsen it are accepted based on the Metropolis condition. This acceptance criterion is implemented by the function *Accept*( $T_n, \pi, \pi'$ ). For the cooling schedule, we retain the temperature level for  $c \cdot n$  consecutive swaps, where  $c$  is a parameter. Some initial experiments show that  $c = 100$  gives a nice performance. Geometric cooling is used with the temperature at iteration  $i + 1$  being set to  $T_{i+1} = \alpha \cdot T_i$  with  $\alpha = 0.9$ . When the temperature is below 1, the temperature value is set to  $T_0$  and restarted with the same cooling process. This temperature schedule is implemented in function *SetTemperature*( $T_n, Iterations, c$ ). The SA algorithm continues until the maximum computation time limit. The number of iterations and initial temperatures are the parameters to be set for the SA algorithm. Details of the SA algorithm are given in Algorithm 3.

**Algorithm 3.** Simulated annealing for the QAP.

```

1  $T_0$  is the initial temperature
2  $\pi$  is a solution vector
3 generate initial solution  $\pi$ 
4  $\pi_{best} \leftarrow \pi$ 
5  $T_0 \leftarrow$  initial value
6 while (termination condition is not satisfied) do
7    $\pi' \leftarrow$  GenerateRandomNeighbour( $\pi$ )
8    $\pi'' \leftarrow$  Accept( $T_n, \pi, \pi'$ )
9   if ( $f(\pi') < f(\pi_{best})$ ) then
10     $\pi_{best} \leftarrow \pi'$ 
11     $T_{n+1} \leftarrow$  SetTemperature ( $T_n, Iterations, c$ )
12     $\pi \leftarrow \pi''$ 
13 return  $\pi_{best}$ 
```

### 3.5.2. Robust tabu search

Glover proposed and formalized Tabu Search (meta)-heuristic as a better local search method for the optimization problems (Glover, 1990). In general, a local search method begins with a starting solution and attempts to improve the solution by visiting neighbor solutions. Neighbor solutions can be generated by changing the initial solution with minor changes. Therefore, local search method explores neighbors of the initial solution for finding a better solution. Local search method can terminate in local optima or on a landscape where all neighbors have similar quality. When this happens, local search method stops with a suboptimal solution which might be far away from the global optima. In order to remove this drawback of local search, Tabu search introduces two novel properties. To avoid getting stuck into local optima, Tabu Search can prefer a neighbor solution with a worse quality if other neighbors do not provide a better solution than the initial solution. This move can make Tabu search get rid of the local optima.

As a second property, Tabu Search does not visit previous neighbor solutions if they have been visited before or if they are marked by the user as forbidden locations. To implement this property, a taboo list is kept to store the visited solutions. Because of this property, the method name is termed as Tabu. In practice, taboo list can be implemented with various data structures held in memory. This list can hold the visited and forbidden solutions. By limiting the size of the list, it can be used as a short-term memory. When the list is filled, the oldest entry is deleted as First-In First-Out (FIFO) queue. RTS has been applied to solve QAP very effectively so far (Misevičius, 2012; Taillard, 1991). The algorithm of RTS is presented in Algorithm 4. Maximum number of failures, tabu tenure upper limit, tabu tenure lower limit, and aspiration value are the parameters to be set for the RTS heuristic.

**Algorithm 4.** Robust tabu search algorithm (Taillard, 1991).

```

1 Authorized: If a move is not tabu, it is authorized.
2 Aspired: Allow tabu moves if they are decided to be interesting.
3 Tabu List: A list to forbid reverse move.
4 Neighbor: Each location in the permutation is considered as neighbor.
5 RTS (FLOW, DIST, MaxIter, BestPerm, MinSize(<n×n/2), MaxSize(<n×n/2), Aspiration(>n×n/2));
6 TABU_LIST ← {};
7 CurCost ← QAP_Cost(BestPerm);
8 CurSol ← BestPerm;
9 Delta[i][j] ← ComputeDelta(); /* i = 0,...,n, j = 0,...,n */
10 TABU_LIST[i][j] ← - (n×i+j); /* i = 0,...,n-1, j = 0,...,n-1 */
11 for (iteration ← 1; iteration < MaxIter; iteration++) do
12   i_retained ← infinite;
13   MinDelta ← infinite;
14   Already_Aspired ← false;
15   for (each Neighbor (i, j)) do
16     current1 ← TABU_LIST[i][CurSol[j]];
17     current2 ← TABU_LIST[j][CurSol[i]];
18     Authorized ← (current1 < iteration) or (current2 < iteration);
19     Aspired ← (current1 < iteration-Aspiration) or (current2 < iteration-Aspiration) or (CurCost +
       Delta[i][j] < BestCost);
20     if ( (Aspired and Already_Aspired) or (Aspired and Delta[i][j] < MinDelta) or
       (!Aspired and !Already_Aspired and Delta[i][j] < MinDelta and Authorized) ) then
21       i_retained ← i;
22       j_retained ← j;
23       MinDelta ← Delta[i][j];
24       if (Aspired) then
25         Already_Aspired ← true;
26   if (i_retained ≠ infinite) then
27     SWAP(CurSol[i_retained], CurSol[j_retained]);
28     CurCost ← CurCost + Delta[i_retained][j_retained];
29     TABU_LIST[i_retained][CurSol[j_retained]] ← iteration + getRandom(MinSize, MaxSize);
30     TABU_LIST[j_retained][CurSol[i_retained]] ← iteration + getRandom(MinSize, MaxSize);
31     if (CurCost < BestCost) then
32       BestCost ← CurCost;
33   UPDATE_MOVE_COSTS(FLOW, DIST, CurSol, Delta, i, j, i_retained, j_retained);
34
```

### 3.5.3. Fast Ant system (FANT)

The Ant Colony Optimization (ACO) algorithms have evolved since their first introduction (Taillard, 1998, 2002). With Fast Ant System (FANT) algorithm, a general model developed from the real ant model is presented. This model is called the Ant System (meta)-heuristic and defined by a set of principles for the development of heuristics for several combinatorial optimization problems. Artificial ant systems

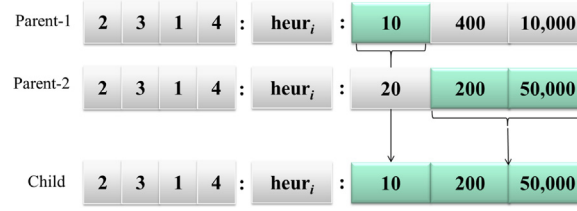


Fig. 3. Crossover operator for the MSH-QAP algorithm.

are developed by making three analogies. (1) Processes simulate real ants that are in charge of building solutions to the combinatorial problem and they are called as Artificial ants. (2) The pheromone trails of the ants corresponds to a common memory that is revised each time that a new solution is found. (3) The queen is a central process in charge of activating and coordinating artificial ants and of managing the common memory. An ant (meta)-heuristic can be described by a set of processes that work through a common memory. The first set of processes, corresponding to the ants, built solutions in a probabilistic way, with probabilities depending on information stored in memory. All these artificial Ant processes are coordinated by a Queen process that also manages the common memory.

Matrix  $M$  of size  $n \times n$  is the common memory. Matrix entry  $m_{ir}$  is the facility  $i$  placed on location  $r$  in solutions previously generated by the algorithm. The higher the quality and the larger the number of solutions generated with facility  $i$  placed on location  $r$ , the higher the  $m_{ir}$  value is.

The idea of the Fast Ant System is to design a simple method that incorporates diversification and intensification strategies. This is achieved by reinforcing the attractiveness of the  $m_{ir}$  values, the best solution found so far by the search, and clearing the common memory if the process appears to be stagnating. The Ant process builds a new solution by choosing the location  $r$  of facility  $i$  with a probability proportional to  $m_{ir}$ . Then the current solution is improved with a local search technique and sent to the Queen process (see Algorithm 5 for the pseudocode of the ANT process). While implementing intensification and diversification, the Queen process manages the memory matrix  $M$ , a variable  $v$  and the best solution  $\pi^*$  obtained by the system. Initially  $v=1$  and  $m_{ir}=v, \forall i, r$ . The Queen repeats the process given in Algorithm 6.

**Algorithm 5.** Ant process pseudocode for FAnt algorithm (Taillard, 1998).

- 1 Receive problem data, memory state and other parameters from the Queen process,
- 2 Build a new solution probabilistically
- 3 Send the new solution to Queen process.

**Algorithm 6.** Queen process pseudocode for FAnt algorithm (Taillard, 2002).

- 1 Start an Ant Process
- 2 Wait for a solution  $\pi$  from an Ant process
- 3 **if** ( $\pi = \pi^*$ ) **then**
- 4     set  $v \leftarrow v + 1$
- 5     set  $m_{ir} = v \forall i, r$
- 6 **if** ( $\pi$  is better than  $\pi^*$ ) **then**
- 7     set  $\pi^* \leftarrow \pi$
- 8     set  $v \leftarrow 1$
- 9     set  $m_{ir} \leftarrow v \forall i, r$
- 10 set  $m_{i\pi_i} \leftarrow m_{i\pi_i} + v, \forall i$
- 11 set  $m_{i\pi_i^*} \leftarrow m_{i\pi_i^*} + R, \forall i$  //  $R$  is a parameter used by Queen process

### 3.5.4. Breakout local search (BLS)

Breakout local search (BLS) implements a technique similar to Iterated Local Search (ILS) algorithm (Malucelli, 2003; Benlic and Hao, 2013). It changes between a local search phase and a dedicated perturbation phase iteratively to obtain new promising solution areas. BLS starts from an initial solution,  $\pi_0$ , and applies the steepest descent procedure to  $\pi_0$  for reaching a local optimum  $\pi$ . Each round of the steepest descent explores the neighbors and selects the best one. If there is no solution in the neighborhood, it means that a local optima is reached. BLS attempts to escape from this local optimum  $\pi$  by applying dedicated perturbation moves to  $\pi$  (with a suitable number  $L$ ).  $\pi$  is said to be perturbed with these moves. This perturbed solution becomes a new starting point.

A successful local search depends on the right degree of diversification applied to the search. An optimal diversification degree at a certain solution may not be necessarily optimal for another one. The diversification degree that is applied by a perturbation mechanism depends on the number of perturbation moves (*jump magnitude*) and the type of moves. In a weak diversification environment, the local search has a higher probability of getting stuck into local optimal solutions. This is called stagnation. Conversely, a strong diversification may lead to a random restart with a low probability of obtaining better solutions. The perturbation mechanism of BLS tries to find the



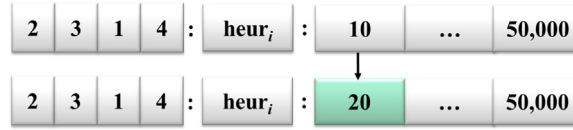


Fig. 4. Mutation operator for the MSH-QAP algorithm.

most suitable degree of diversification required at a certain stage of the search. BLS does this by dynamically determining the number  $L$  of perturbation moves and by adaptively choosing between three types of perturbation moves of different intensities.

Algorithm 7 presents the solution of BLS for the QAP. Details of the algorithm can be found in Benlic and Hao (2013). Starting solution of the BLS is randomly chosen and uses the descent procedure to reach a local optimum  $\pi$ . The jump magnitude  $L$  is decided depending on the search escaped or returned to the previous local optimum. BLS uses  $L$  perturbation moves to  $\pi$  in order to get a new starting point. Initial jump magnitude, maximal jump magnitude, maximum number of non-improving attractors visited before strong perturb, tabu tenure, smallest probability for applying directed perturbation, and probability for applying random over recency-based perturb are the parameters to be set for the BLS heuristic.

**Algorithm 7.** Breakout local search heuristic algorithm (Benlic and Hao, 2013).

```

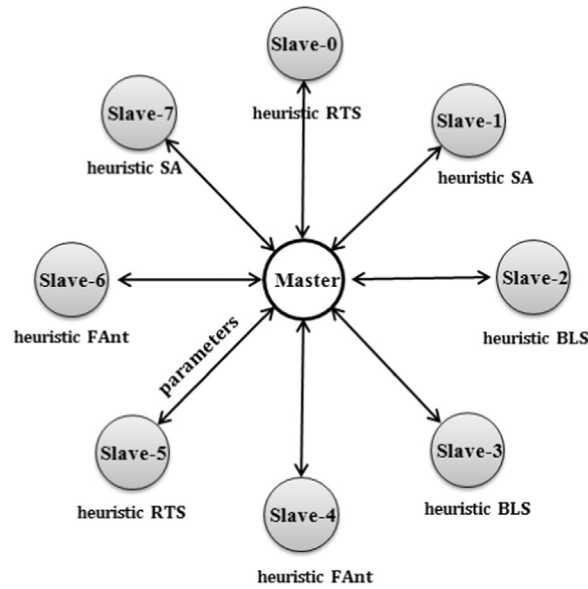
1 distance and flow matrices  $d$  and  $f$  of size  $n \times n$ .
2 a permutation  $\pi$  over a set of facility locations.
3  $\pi \leftarrow$  random permutation of  $\{1, \dots, n\}$ 
4  $c \leftarrow C(\pi)$  // is the objective value of the current solution
5 Compute the initial  $n \times n$  matrix  $\delta$  of move gains
6  $\pi_{best} \leftarrow \pi$  //  $\pi_{best}$  is the best solution found so far
7  $c_{best} \leftarrow c$  //  $c_{best}$  is the best objective value reached so far
8  $c_p \leftarrow c$  //  $c_p$  is the best objective value of the last descent
9  $\omega \leftarrow 0$  //  $\omega$  is the counter for consecutive non-improving local optima
10  $L \leftarrow L_0$  // set the number of perturbation moves  $L$  to its default value  $L_0$ 
11 while the termination condition is not satisfied do
12   while  $\exists \text{ swap}(u, v)$  such that  $(c + \delta(\pi, u, v)) < c$  do
13      $\pi_{best} \leftarrow \pi \oplus \text{swap}(u, v)$  // perform the best-improving move
14      $c \leftarrow c + \delta(\pi, u, v)$ 
15      $H_{uv} \leftarrow \text{Iter}$  // update iteration number when move  $uv$  was last performed
16     Update Matrix  $\delta$ 
17      $\text{Iter} \leftarrow \text{Iter} + 1$ 
18   if  $(c < c_{best})$  then
19      $\pi_{best} \leftarrow \pi$ ;  $c_{best} \leftarrow c$ ; // update the recorded best solution
20      $\omega \leftarrow 0$ ; // reset counter for consecutive non-improv. local optima
21   if  $(c \neq c_p)$  then
22      $\omega \leftarrow \omega + 1$ ;
23   // determine the perturbation strength  $L$  to be applied to  $\pi$ 
24   if  $(\omega > T)$  then
25     // search seems to be stagnating, set  $L$  to a large value
26      $L \leftarrow L_{max}$ ;  $\omega \leftarrow 0$ ;
27   if  $(c = c_p)$  then
28      $L \leftarrow L + 1$  // search returned to the previous local optimum, increase jump magnitude by one
29   else
30      $L \leftarrow L_0$  // search escaped from the previous local optimum, reinitialize jump magnitude
31   // perturb the current local optimum  $\pi$  with  $L$  perturb. moves
32    $c_p < c$ ; // update the objective value of the previous local optimum
33    $\pi \leftarrow \text{Perturbation}(\pi, L, H, TER, \delta, \omega)$ ;

```

#### 4. Performance evaluation of experimental results

In this section, we present the results of our experiments that are obtained with the MSH-QAP algorithm. 134 problem instances given in the QAP benchmark, QAPLIB, are solved during the experiments (Burkard et al., 1991). Since most of the state-of-the-art QAP algorithms are evaluated on these problem instances, QAPLIB provides a fair ground to compare the solutions with the other algorithms in the literature. The QAP instances are analyzed on four types of problem instances categorized by Stützle (2006). The classified instances given by Stützle are:

Type 1. *Unstructured, randomly generated instances* have distance matrix that is randomly generated based on a uniform distribution.



**Fig. 5.** Communication topology of the algorithm (master node sends the information of the heuristics and gets back the solution of the problem and the execution information of the heuristic). There are slaves and a master in the communication topology. Each slave is executing a different heuristic and sending the evaluated chromosome to the master node.

**Table 1**  
Parameter settings for the MSH-QAP algorithm.

| Parameter             | Setting |
|-----------------------|---------|
| Population size       | 24      |
| Number of generations | 10      |
| Number of processors  | 64      |
| Number of multistarts | 5       |
| Crossover ratio       | 50%     |
| Mutation ratio        | 1%      |

**Table 2**  
Ranges of the parameters for the low-level heuristics that are optimized during the experiments. The last column of the table shows the optimized parameter ranges for the heuristics ( $n$  is the QAP size).

| Heuristic | Parameter name                                    | Range                                  | Observed best range                   |
|-----------|---|--|---------------------------------------|
| SA        | Initial temperature                               | [1, 200]                               | [45, 70]                              |
|           | Number of iterations                              | $[100 \times n, 3000 \times n]$        | $[2500 \times n, 3000 \times n]$      |
| RTS       | Max. number of failures                           | $[100 \times n, 10000 \times n]$       | $[9000 \times n, 10000 \times n]$     |
|           | Tabu tenure lower limit                           | $[2 \times n, 10 \times n]$            | $[8 \times n, 9 \times n]$            |
|           | Tabu tenure upper limit                           | $[11 \times n, 20 \times n]$           | $[11 \times n, 12 \times n]$          |
|           | Aspiration value                                  | $[n \times n, 10 \times (n \times n)]$ | $[n \times n, 2 \times (n \times n)]$ |
| FAnt      | Number of iterations                              | $[10 \times n, 5000 \times n]$         | $[4000 \times n, 5000 \times n]$      |
|           | Parameter for managing the traces ( $R$ )         | [1, $n$ ]                              | $[1, 0.2 \times n]$                   |
| BLS       | Parameter for managing the traces ( $increment$ ) | [1, $n$ ]                              | $[1, 0.3 \times n]$                   |
|           | Initial jump magnitude                            | $[0.01 \times n, 0.5 \times n]$        | $[0.04 \times n, 0.18 \times n]$      |
|           | Maximal jump magnitude                            | $[0.1 \times n, 0.9 \times n]$         | $[0.5 \times n, 0.7 \times n]$        |
|           | Maximum number of non-improving attractors        | [100, 5000]                            | [2500, 2900]                          |
|           | Tabu tenure                                       | $[2 \times n, 18 \times n]$            | $[8 \times n, 12 \times n]$           |

**Table 3**  
Deviation percentages of the heuristics from the best known solutions through generations. Initial population (having randomly selected parameters for the heuristics) and optimized parameters after 20-generations are presented. The experiments are carried out 30 times for nug30 problem instances.

| Heuristic | Initial population deviation (%) | Deviation after 20-generations (%) | Improvement percentage (%) |
|-----------|----------------------------------|------------------------------------|----------------------------|
| SA        | 164.40                           | 0.50                               | 99.7                       |
| RTS       | 267.71                           | 0.04                               | 99.9                       |
| FAnt      | 51.01                            | 0.05                               | 98.9                       |
| BLS       | 142.42                           | 0.03                               | 99.9                       |

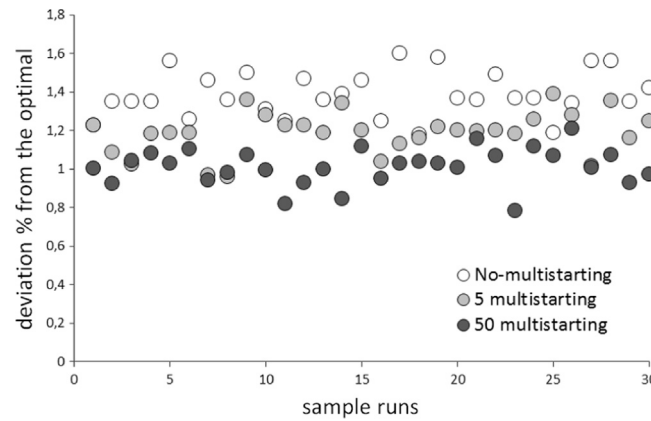


Fig. 6. The effect of multistart for tai100a (having the same parameter settings for each run of RTS).

Type 2. *Instances with Grid-based distances* contain instances in which the distances are the Manhattan distance between points on a grid.

Type 3. *Real-life instances* are produced from real-life QAP applications.

Type 4. *Real-life-like instances* are generated instances that are similar to real-life QAP problems.

We compare our results with those of the recent state-of-the-art (meta)-heuristics in terms of solution quality and computational effort and discuss the robustness and scalability issues of the proposed algorithm. Experiments are performed on a High Performance Cluster (HPC) computer which has 46 nodes, each with 2 CPUs giving 92 CPUs. Each CPU has 4 cores giving a total of 368 cores. Each node has 16 GB of RAM giving 736 GB of total memory. High-bandwidth communication is available among nodes, using two 24 port Gbps ethernet switches, and one 24 port infiniband switch, providing very low latency messaging with a capacity of 8 Gbps. C++ and the MPI libraries are used during the development (Novoa et al., 2015; Kumar, 2002). Tests are performed 10 times for each problem instance and the average values are reported. Table 1 presents the parameters used for the GA phase of the MSH-QAP algorithm during the experiments.

Table 2 presents the ranges of the parameters for the low-level heuristics that are optimized during the experiments. The last column of the table gives the overall best ranges of the parameters that are used by the heuristics. Some of the parameters increase with respect to the problem size  $n$ .

Settings the right parameters has a significant impact on the performance of a heuristic algorithm (Shi and Eberhart, 1998; Lobo et al., 2007). There is no exact method defined for this process and previously prepared parameters may not produce good performance for every problem instance. In the first phase of our algorithm, the parameters used for the SA, RTS, FAnT, and BLA algorithms are tuned adaptively by using crossover and mutation operators. For the SA algorithm, the acceptance probability, the annealing schedule temperature, and initial temperatures are set. For the RTS algorithm, maximum number of failures, tabu tenure, and aspiration values are optimized and for the FAnT algorithm, number of FAnT iterations, ant memory, parameter for managing the traces, and iteration counters are optimized. For BLS, initial jump magnitude, maximal jump magnitude, maximum number of non-improving attractors visited before strong perturb, tabu tenure, smallest probability for applying directed perturbation, and probability for applying random over recency-based perturb are tuned. How the heuristic algorithms perform through the generations (initial population and 20 generations) is given in Table 3. The table shows the deviation percentages of the heuristics.

#### 4.1. The effect of multistart process

Multistarting a heuristic from a different point is an efficient technique that has been used for most of the state-of-the-art algorithms (James et al., 2005, 2006, 2009). It helps the algorithm prevent from sticking into local optima by restarting the exploration with a new beginning solution randomly. In Fig. 6, we can observe its positive effect on the average of the solution quality when the number of multistarts is increased for tai100a problem instance. The experiments are performed with a single, 5, and 50 multistarts respectively for 30 times. 27.1% improvement in the average deviation of the obtained results is observed with 50 multistarts with respect to a non-multistarting solution method. The results are given with respect to the Best Known Solution (BKS) of the tai100a problem instance in the QAPLIB.

#### 4.2. Fitness distance landscape analysis of the QAP instances

Fitness Distance Correlation (FDC) defines an efficient analysis of the QAP landscape and local optima with respect to the fitness value and similarities to the best known solutions (Jones and Forrest, 1995; Tavarani-N and Prugel-Bennett, 2014, 2012). FDC is a parameter to decide which type of heuristic to apply to a problem instance depending on its FDC analysis. The QAP instances are reported to have very unstructured landscapes (Merz and Freisleben, 2000). The local optima of the QAP instances are not restricted to small search spaces and they are not very correlated with each other (tai100b is an exception to this evaluation). Therefore, most of the QAP instances have no exploitable structure with respect to the distribution of the local optima. For that reason, we use a random number generator for different starting points of the QAP instances during GA and multistart phases of the algorithm.

**Table 4**

122 optimal solutions found by our proposed algorithm, MSH-QAP.

| Instance | BKS       | Found | APD | BPD | Min. | Instance | BKS        | Found | APD | BPD | Min. |
|----------|-----------|-------|-----|-----|------|----------|------------|-------|-----|-----|------|
| bur26a   | 5426670   | 10    | 0   | 0   | 1.3  | lipa60b  | 2520135    | 10    | 0   | 0   | 3.0  |
| bur26b   | 3817852   | 10    | 0   | 0   | 1.3  | lipa70a  | 169755     | 10    | 0   | 0   | 3.5  |
| bur26c   | 5426795   | 10    | 0   | 0   | 1.3  | lipa70b  | 4603200    | 10    | 0   | 0   | 3.5  |
| bur26d   | 3821225   | 10    | 0   | 0   | 1.3  | lipa80a  | 253195     | 10    | 0   | 0   | 4.0  |
| bur26e   | 5386879   | 10    | 0   | 0   | 1.3  | lipa80b  | 7763962    | 10    | 0   | 0   | 4.0  |
| bur26f   | 3782044   | 10    | 0   | 0   | 1.3  | lipa90a  | 360630     | 10    | 0   | 0   | 4.5  |
| bur26g   | 10117172  | 10    | 0   | 0   | 1.3  | lipa90b  | 12490441   | 10    | 0   | 0   | 4.5  |
| bur26h   | 7098658   | 10    | 0   | 0   | 1.3  | nug14    | 1014       | 10    | 0   | 0   | 0.7  |
| chr12a   | 9552      | 10    | 0   | 0   | 0.6  | nug15    | 1150       | 10    | 0   | 0   | 0.8  |
| chr12b   | 9742      | 10    | 0   | 0   | 0.6  | nug16a   | 1610       | 10    | 0   | 0   | 0.8  |
| chr12c   | 11156     | 10    | 0   | 0   | 0.6  | nug16b   | 1240       | 10    | 0   | 0   | 0.8  |
| chr15a   | 9896      | 10    | 0   | 0   | 0.8  | nug17    | 1732       | 10    | 0   | 0   | 0.9  |
| chr15b   | 7990      | 10    | 0   | 0   | 0.8  | nug18    | 1930       | 10    | 0   | 0   | 0.9  |
| chr15c   | 9504      | 10    | 0   | 0   | 0.8  | nug20    | 2570       | 10    | 0   | 0   | 1.0  |
| chr18a   | 11098     | 10    | 0   | 0   | 0.9  | nug21    | 2438       | 10    | 0   | 0   | 1.1  |
| chr18b   | 1534      | 10    | 0   | 0   | 0.9  | nug22    | 3596       | 10    | 0   | 0   | 1.1  |
| chr20a   | 2192      | 10    | 0   | 0   | 1.0  | nug24    | 3488       | 10    | 0   | 0   | 1.2  |
| chr20b   | 2298      | 10    | 0   | 0   | 1.0  | nug25    | 3744       | 10    | 0   | 0   | 1.3  |
| chr20c   | 14142     | 10    | 0   | 0   | 1.0  | nug27    | 5234       | 10    | 0   | 0   | 1.4  |
| chr22a   | 6156      | 10    | 0   | 0   | 1.1  | nug28    | 5166       | 10    | 0   | 0   | 1.4  |
| chr22b   | 6194      | 10    | 0   | 0   | 1.1  | nug30    | 6124       | 10    | 0   | 0   | 1.5  |
| chr25a   | 3796      | 10    | 0   | 0   | 1.3  | rou12    | 235528     | 10    | 0   | 0   | 0.6  |
| els19    | 17212548  | 10    | 0   | 0   | 1.0  | rou15    | 354210     | 10    | 0   | 0   | 0.8  |
| esc16a   | 68        | 10    | 0   | 0   | 0.8  | rou20    | 725522     | 10    | 0   | 0   | 1.0  |
| esc16b   | 292       | 10    | 0   | 0   | 0.8  | scr12    | 31410      | 10    | 0   | 0   | 0.6  |
| esc16c   | 160       | 10    | 0   | 0   | 0.8  | scr15    | 51140      | 10    | 0   | 0   | 0.8  |
| esc16d   | 16        | 10    | 0   | 0   | 0.8  | scr20    | 110030     | 10    | 0   | 0   | 1.0  |
| esc16e   | 28        | 10    | 0   | 0   | 0.8  | sks42    | 15812      | 10    | 0   | 0   | 2.1  |
| esc16f   | 0         | 10    | 0   | 0   | 0.8  | sks49    | 23386      | 10    | 0   | 0   | 2.8  |
| esc16g   | 26        | 10    | 0   | 0   | 0.8  | sks56    | 34458 28   | 10    | 0   | 0   | 2.8  |
| esc16h   | 996       | 10    | 0   | 0   | 0.8  | sks64    | 48498      | 10    | 0   | 0   | 3.2  |
| esc16i   | 14        | 10    | 0   | 0   | 0.8  | sks72    | 66256      | 10    | 0   | 0   | 3.6  |
| esc16j   | 8         | 10    | 0   | 0   | 0.8  | sks81    | 90998      | 10    | 0   | 0   | 4.1  |
| esc32a   | 130       | 10    | 0   | 0   | 1.6  | sks90    | 115534     | 10    | 0   | 0   | 4.5  |
| esc32b   | 168       | 10    | 0   | 0   | 1.6  | sks100e  | 149150     | 10    | 0   | 0   | 75.0 |
| esc32c   | 642       | 10    | 0   | 0   | 1.6  | sks100f  | 149036     | 10    | 0   | 0   | 75.0 |
| esc32d   | 200       | 10    | 0   | 0   | 1.6  | ste36a   | 9526       | 10    | 0   | 0   | 1.8  |
| esc32e   | 2         | 10    | 0   | 0   | 1.6  | ste36b   | 15852      | 10    | 0   | 0   | 1.8  |
| esc32f   | 2         | 10    | 0   | 0   | 1.6  | ste36c   | 8239110    | 10    | 0   | 0   | 1.8  |
| esc32g   | 6         | 10    | 0   | 0   | 1.6  | tai12a   | 224416     | 10    | 0   | 0   | 0.6  |
| esc32h   | 438       | 10    | 0   | 0   | 1.6  | tai12b   | 39464925   | 10    | 0   | 0   | 0.6  |
| esc64    | 116       | 10    | 0   | 0   | 3.2  | tai15a   | 388214     | 10    | 0   | 0   | 0.8  |
| esc128   | 64        | 10    | 0   | 0   | 6.4  | tai15b   | 51765268   | 10    | 0   | 0   | 0.8  |
| had12    | 1652      | 10    | 0   | 0   | 0.6  | tai17a   | 491812     | 10    | 0   | 0   | 0.9  |
| had14    | 2724      | 10    | 0   | 0   | 0.7  | tai20a   | 703482     | 10    | 0   | 0   | 1.0  |
| had16    | 3720      | 10    | 0   | 0   | 0.8  | tai20b   | 122455319  | 10    | 0   | 0   | 1.0  |
| had18    | 5358      | 10    | 0   | 0   | 0.9  | tai25a   | 1167256    | 10    | 0   | 0   | 1.3  |
| had20    | 6922      | 10    | 0   | 0   | 1.0  | tai25b   | 344355646  | 10    | 0   | 0   | 1.3  |
| kra30a   | 88900     | 10    | 0   | 0   | 1.5  | tai30a   | 1818146    | 10    | 0   | 0   | 1.5  |
| kra30b   | 91420     | 10    | 0   | 0   | 1.5  | tai30b   | 637117113  | 10    | 0   | 0   | 1.5  |
| kra32    | 88700     | 10    | 0   | 0   | 1.6  | tai35a   | 2422002    | 10    | 0   | 0   | 1.8  |
| lipa20a  | 3683      | 10    | 0   | 0   | 1.0  | tai35b   | 283315445  | 10    | 0   | 0   | 1.8  |
| lipa20b  | 27076     | 10    | 0   | 0   | 1.0  | tai40b   | 637250948  | 10    | 0   | 0   | 2.0  |
| lipa30a  | 13178     | 10    | 0   | 0   | 1.5  | tai50b   | 458821517  | 10    | 0   | 0   | 3.0  |
| lipa30b  | 151426    | 10    | 0   | 0   | 1.5  | tai60b   | 608215054  | 10    | 0   | 0   | 3.2  |
| lipa40a  | 31538     | 10    | 0   | 0   | 2.0  | tai64c   | 1855928    | 10    | 0   | 0   | 3.3  |
| lipa40b  | 476581    | 10    | 0   | 0   | 4.0  | tai80b   | 818415043  | 10    | 0   | 0   | 4.0  |
| lipa50a  | 62093     | 10    | 0   | 0   | 2.5  | tai100b  | 1185996137 | 10    | 0   | 0   | 5.0  |
| lipa50b  | 1210244   | 10    | 0   | 0   | 2.5  | tho30    | 149936     | 10    | 0   | 0   | 1.5  |
| lipa60a  | 107218    | 10    | 0   | 0   | 3.0  | tho40    | 240516     | 10    | 0   | 0   | 2.0  |
| tai50b   | 818415043 | 10    | 0   | 0   | 4.0  | wil50    | 8133398    | 10    | 0   | 0   | 2.5  |

#### 4.3. Comparison with the state-of-the-art (meta)-heuristics

Optimal results obtained with MSH-QAP algorithm for 122 QAPLIB problem instances are presented in Table 4. Although the number of generations is 10 for the MSH-QAP algorithm and there exists a second phase, optimal results of these instances are found at the very early generations (mostly at the first generation) and terminated the execution of the algorithm for the next generations. The second phase of the MSH-QAP algorithm is not started for these instances. The execution time of these results is shorter than the other problem instances of whose optimal solutions are not found by MSH-QAP algorithm. SA and FAnt algorithms are observed to outperform RTS and BLS for the small problem instances (up to 30 problem size) with their performances however, RTS and BLS outperform both these algorithms as the problem size increases and dominates the population of the genetic phase of the algorithm.

**Table 5**

Comparison of the MSH-QAP algorithm with state-of-the-art algorithms on Type-1 problem instances.

| Instance | BKS      | MSH-QAP      |      | JRG-DivTS |       | ITS          |       | SC-Tabu |       | ACO/GA/LS |          | GDA          | BMA          | PHA          | CPTS         |
|----------|----------|--------------|------|-----------|-------|--------------|-------|---------|-------|-----------|----------|--------------|--------------|--------------|--------------|
|          |          | APD          | Min. | APD       | Min.  | APD          | APD   | Min.    | APD   | Min.      | APD      | APD          | APD          | APD          | APD          |
| tai20a   | 70382    | <b>0</b>     | 1.0  | <b>0</b>  | 0.2   | <b>0</b>     | 0.246 | 0.001   | –     | –         | <b>0</b> | <b>0</b>     | <b>0</b>     | <b>0</b>     | <b>0</b>     |
| tai25a   | 1167256  | <b>0</b>     | 1.3  | <b>0</b>  | 0.2   | <b>0</b>     | 0.239 | 0.03    | –     | –         | <b>0</b> | <b>0</b>     | <b>0</b>     | <b>0</b>     | <b>0</b>     |
| tai30a   | 1818146  | <b>0</b>     | 1.5  | <b>0</b>  | 1.3   | <b>0</b>     | 0.154 | 0.07    | 0.341 | 1.4       | 0.091    | <b>0</b>     | <b>0</b>     | <b>0</b>     | <b>0</b>     |
| tai35a   | 2422002  | <b>0</b>     | 1.8  | <b>0</b>  | 4.4   | <b>0</b>     | 0.280 | 0.18    | 0.487 | 3.5       | 0.153    | <b>0</b>     | <b>0</b>     | <b>0</b>     | <b>0</b>     |
| tai40a   | 3139370  | 0.261        | 30.0 | 0.222     | 5.2   | 0.220        | 0.561 | 0.20    | 0.593 | 13.1      | 0.261    | <b>0.059</b> | <b>0</b>     | <b>0.148</b> | <b>0.148</b> |
| tai50a   | 4941410  | <b>0.165</b> | 37.5 | 0.725     | 10.2  | 0.410        | 0.889 | 0.23    | 0.901 | 29.7      | 0.276    | <b>0.131</b> | <b>0</b>     | <b>0.440</b> | <b>0.440</b> |
| tai60a   | 7205962  | <b>0.270</b> | 45.0 | 0.718     | 25.7  | 0.450        | 0.940 | 0.41    | 1.068 | 58.5      | 0.448    | <b>0.144</b> | <b>0</b>     | <b>0.476</b> | <b>0.476</b> |
| tai80a   | 13499184 | 0.530        | 60.0 | 0.753     | 52.7  | <b>0.360</b> | 0.648 | 1.0     | 1.178 | 152.2     | 0.832    | <b>0.426</b> | 0.644        | <b>0.570</b> | <b>0.570</b> |
| tai100a  | 21059006 | <b>0.338</b> | 75.0 | 0.825     | 142.1 | <b>0.300</b> | 0.977 | 1.99    | 1.115 | 335.6     | 0.874    | <b>0.405</b> | 0.537        | <b>0.558</b> | <b>0.558</b> |
| Average  |          | <b>0.174</b> | 28.1 | 0.360     | 26.88 | 0.193        | 0.548 | 0.45    | 0.812 | 84.9      | 0.326    | <b>0.129</b> | <b>0.131</b> | <b>0.243</b> | <b>0.243</b> |

**Table 6**

Comparison of the MSH-QAP algorithm with state-of-the-art algorithms on Type-2 problem instances.

| Instance | BKS    | MSH-QAP      |      | JRG-DivTS |       | ACO/GA/LS |       | GA/SD |      | GA/C-TS  |      | GDA      | BMA      | PHA          | CPTS     |
|----------|--------|--------------|------|-----------|-------|-----------|-------|-------|------|----------|------|----------|----------|--------------|----------|
|          |        | APD          | Min. | APD       | min.  | APD       | Min.  | APD   | Min. | APD      | Min. | APD      | APD      | APD          | APD      |
| sko42    | 15812  | <b>0</b>     | 2.1  | <b>0</b>  | 4.0   | <b>0</b>  | 0.7   | 0.014 | 0.16 | <b>0</b> | 1.2  | <b>0</b> | <b>0</b> | <b>0</b>     | <b>0</b> |
| sko49    | 23386  | <b>0</b>     | 2.5  | 0.008     | 9.6   | 0.056     | 7.6   | 0.107 | 0.28 | 0.009    | 2.1  | 0.005    | <b>0</b> | <b>0</b>     | <b>0</b> |
| sko56    | 34458  | <b>0</b>     | 2.8  | 0.002     | 13.2  | 0.012     | 9.1   | 0.054 | 0.42 | 0.001    | 3.2  | 0.001    | <b>0</b> | <b>0</b>     | <b>0</b> |
| sko64    | 48498  | <b>0</b>     | 3.2  | <b>0</b>  | 22.0  | 0.004     | 17.4  | 0.051 | 0.73 | <b>0</b> | 5.9  | <b>0</b> | <b>0</b> | <b>0</b>     | <b>0</b> |
| sko72    | 66256  | <b>0</b>     | 3.6  | 0.006     | 38.0  | 0.018     | 70.8  | 0.112 | 0.93 | 0.014    | 8.4  | 0.007    | <b>0</b> | <b>0</b>     | <b>0</b> |
| sko81    | 90998  | <b>0</b>     | 4.1  | 0.016     | 56.6  | 0.025     | 112.3 | 0.087 | 1.44 | 0.014    | 13.3 | 0.019    | <b>0</b> | <b>0</b>     | <b>0</b> |
| sko90    | 115534 | <b>0</b>     | 4.5  | 0.026     | 89.6  | 0.042     | 92.1  | 0.139 | 2.31 | 0.011    | 22.4 | 0.031    | <b>0</b> | <b>0</b>     | <b>0</b> |
| sko100a  | 152002 | 0.003        | 75.0 | 0.027     | 129.2 | 0.021     | 171.0 | 0.114 | 3.42 | 0.018    | 33.6 | 0.029    | <b>0</b> | <b>0</b>     | <b>0</b> |
| sko100b  | 153890 | 0.004        | 75.0 | 0.008     | 106.6 | 0.012     | 192.4 | 0.096 | 3.47 | 0.011    | 34.1 | 0.015    | <b>0</b> | <b>0</b>     | <b>0</b> |
| sko100c  | 147862 | 0.003        | 75.0 | 0.006     | 126.7 | 0.005     | 220.6 | 0.075 | 3.22 | 0.003    | 33.8 | 0.013    | <b>0</b> | <b>0</b>     | <b>0</b> |
| sko100d  | 149576 | 0.004        | 75.0 | 0.027     | 123.5 | 0.029     | 209.2 | 0.137 | 3.45 | 0.049    | 33.9 | 0.013    | <b>0</b> | <b>0.006</b> | <b>0</b> |
| sko100e  | 149150 | <b>0</b>     | 75.0 | 0.009     | 108.8 | 0.002     | 208.1 | 0.071 | 3.31 | 0.002    | 30.7 | <b>0</b> | <b>0</b> | <b>0</b>     | <b>0</b> |
| sko100f  | 149036 | <b>0</b>     | 75.0 | 0.023     | 110.3 | 0.034     | 210.9 | 0.148 | 3.55 | 0.032    | 35.7 | 0.013    | <b>0</b> | <b>0</b>     | 0.003    |
| Average  |        | <b>0.001</b> | 36.4 | 0.012     | 72.1  | 0.020     | 117.1 | 0.093 | 2.1  | 0.013    | 19.9 | 0.011    | <b>0</b> | <b>0</b>     | <b>0</b> |

**Table 7**

Comparison of the MSH-QAP algorithm with state-of-the-art algorithms on Type-3 problem instances.

| Instance | BKS     | MSH-QAP  |      | JRG-DivTS | ACO/GA/LS | GA/SD    | ITS      | GA-C/Ts  | SC-TABU | GDA      | BMA      | PHA      | CPTS     |
|----------|---------|----------|------|-----------|-----------|----------|----------|----------|---------|----------|----------|----------|----------|
|          |         | APD      | Min. | APD       | BPD       | APD      | APD      | APD      | APD     | APD      | APD      | APD      | APD      |
| kra30a   | 88900   | <b>0</b> | 1.5  | <b>0</b>  | <b>0</b>  | –        | <b>0</b> | –        | 0.714   | –        | <b>0</b> | <b>0</b> | –        |
| kra30b   | 91420   | <b>0</b> | 1.5  | <b>0</b>  | <b>0</b>  | 0.253    | <b>0</b> | <b>0</b> | 0.178   | –        | <b>0</b> | <b>0</b> | –        |
| kra32    | 88700   | <b>0</b> | 1.6  | <b>0</b>  | <b>0</b>  | 0.037    | –        | <b>0</b> | –       | –        | <b>0</b> | <b>0</b> | –        |
| ste36a   | 9526    | <b>0</b> | 1.8  | <b>0</b>  | <b>0</b>  | –        | 0.04     | –        | –       | –        | <b>0</b> | <b>0</b> | <b>0</b> |
| ste36b   | 15852   | <b>0</b> | 1.8  | <b>0</b>  | <b>0</b>  | 0.246    | <b>0</b> | 0.005    | –       | –        | <b>0</b> | <b>0</b> | –        |
| ste36c   | 8239110 | <b>0</b> | 1.8  | <b>0</b>  | <b>0</b>  | 0.015    | <b>0</b> | <b>0</b> | –       | –        | <b>0</b> | <b>0</b> | <b>0</b> |
| esc32b   | 168     | <b>0</b> | 1.6  | <b>0</b>  | <b>0</b>  | <b>0</b> | <b>0</b> | 0.039    | –       | –        | <b>0</b> | <b>0</b> | –        |
| esc32c   | 642     | <b>0</b> | 1.6  | <b>0</b>  | <b>0</b>  | <b>0</b> | <b>0</b> | <b>0</b> | –       | –        | <b>0</b> | <b>0</b> | –        |
| esc32d   | 200     | <b>0</b> | 1.6  | <b>0</b>  | <b>0</b>  | <b>0</b> | <b>0</b> | <b>0</b> | –       | –        | <b>0</b> | <b>0</b> | –        |
| esc32e   | 2       | <b>0</b> | 1.6  | <b>0</b>  | <b>0</b>  | <b>0</b> | <b>0</b> | <b>0</b> | –       | <b>0</b> | <b>0</b> | <b>0</b> | –        |
| esc32g   | 6       | <b>0</b> | 1.6  | <b>0</b>  | <b>0</b>  | <b>0</b> | <b>0</b> | –        | –       | <b>0</b> | <b>0</b> | <b>0</b> | –        |
| esc32h   | 438     | <b>0</b> | 1.6  | <b>0</b>  | <b>0</b>  | <b>0</b> | <b>0</b> | –        | –       | <b>0</b> | <b>0</b> | <b>0</b> | –        |
| esc64a   | 116     | <b>0</b> | 3.2  | <b>0</b>  | <b>0</b>  | <b>0</b> | <b>0</b> | <b>0</b> | –       | <b>0</b> | <b>0</b> | <b>0</b> | –        |
| esc128   | 64      | <b>0</b> | 6.4  | <b>0</b>  | <b>0</b>  | <b>0</b> | 0.01     | <b>0</b> | –       | –        | <b>0</b> | <b>0</b> | –        |
| Average  |         | <b>0</b> | 2.1  | <b>0</b>  | <b>0</b>  | 0.045    | <b>0</b> | 0.004    | 0.446   | <b>0</b> | <b>0</b> | <b>0</b> | <b>0</b> |

The state-of-the-art algorithms in the literature, Multi-Start TS Algorithm JRG-DivTS (James et al., 2006), Iterated Tabu Search (ITS) (Misevičius, 2012), Self Controlling Tabu Search (SC-Tabu) (Fescioglu-Unver and Kokar, 2011), Ant Colony Optimization GA/Local Search Hybrid ACO/GA/LS (Tseng and Liang, 2005), GA Hybrid with Concentric TS Operator GA/C-TS (Drezner, 2005), GA Hybrid with a Strict Descent Operator GA/SD (Drezner, 2005), Parallel Hybrid Algorithm (PHA) (Tosun, 2015), Memetic search for the QAP (BMA) (Benlic and Hao, 2015), Great Deluge and Tabu Search (GDA) (Acan and Unveren, 2015) are selected to be compared with MSH-QAP algorithm during our experiments. Tables 5–8 present the results of our experiments with respect to the four categories defined by Stützle respectively (the best performing first three results are given in bold face).

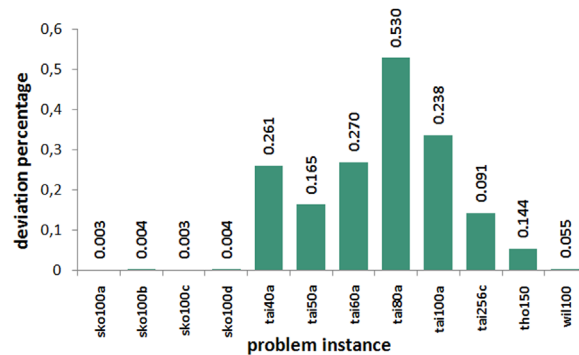
The overall deviation percentage of the MSH-QAP algorithm for the problems categorized by Stützle is 0.035% which is among the best three algorithms that are reported in this study. This is a high performance result and proves the robustness of the algorithm when



**Table 8**

Comparison of the MSH-QAP algorithm with state-of-the-art algorithms on Type-4 problem instances.

| Instance | BKS        | MSH-QAP |      | JRG-DivTS |       | ITS   |      | ACO/GA/LS   |       | SC-Tabu      |       | BMA | PHA | CPTS  |
|----------|------------|---------|------|-----------|-------|-------|------|-------------|-------|--------------|-------|-----|-----|-------|
|          |            | APD     | Min. | APD       | Min.  | APD   | Min  | APD         | Min   | APD          | Min   | APD | APD | APD   |
| tai20b   | 122455319  | 0       | 1.0  | 0         | 0.2   | 0     | 0.01 | –           | –     | 0            | 0.002 | 0   | 0   | 0     |
| tai25b   | 344355646  | 0       | 1.3  | 0         | 0.5   | 0     | 0.01 | –           | –     | 0.007        | 0.010 | 0   | 0   | 0     |
| tai30b   | 637117113  | 0       | 1.5  | 0         | 1.3   | 0     | 0.01 | 0           | 0.3   | 0            | 0.034 | 0   | 0   | 0     |
| tai35b   | 283315445  | 0       | 1.8  | 0         | 2.4   | 0.02  | 0.08 | 0           | 0.3   | 0.059        | –     | 0   | 0   | 0     |
| tai40b   | 637250948  | 0       | 2.0  | 0         | 3.2   | 0.01  | 0.2  | 0           | 0.6   | 0            | 0.092 | 0   | 0   | 0     |
| tai50b   | 458821517  | 0       | 3.0  | 0         | 8.8   | 0.02  | 0.5  | 0           | 2.9   | <b>0.002</b> | 0.23  | 0   | 0   | 0     |
| tai60b   | 608215054  | 0       | 3.2  | 0         | 17.1  | 0.04  | 1.7  | 0           | 2.8   | 0            | 0.41  | 0   | 0   | 0     |
| tai80b   | 818415043  | 0       | 4.0  | 0.006     | 58.2  | 0.23  | 3.0  | 0           | 60.3  | <b>0.003</b> | 1.0   | 0   | 0   | 0     |
| tai100b  | 1185996137 | 0       | 5.0  | 0.056     | 118.9 | 0.14  | 6.66 | <b>0.01</b> | 698.9 | <b>0.014</b> | 1.98  | 0   | 0   | 0.001 |
| Average  |            | 0       | 2.5  | 0.07      | 23.4  | 0.051 | 1.35 | 0.001       | 109.4 | 0.009        | 0.47  | 0   | 0   | 0     |

**Fig. 7.** The deviation percentages of MSH-QAP algorithm for the solutions whose BKS are not found.

compared with the other the state-of-the-art algorithms in the literature. For Type 1, MSH-QAP is the third algorithm with respect to the performance of the other algorithms. The average deviation of the first two algorithms (BMA and PHA) is 0.130% on the average, whereas MSH-QAP has only 0.174% deviation from the BKS. The deviation of MSH-QAP for the well-known problem instance *tai100a* is the best performing one among the selected algorithms. For Type 2 problems, algorithms BHA, BMA, and CPTS perform 0.0% deviation, whereas MSH-QAP has only 0.001% deviation. For problem Types 3 and 4, MSH-QAP is among the best performing algorithms in literature having 0.0% deviation with respect to the BKS results reported in the QAPLIB.

The number of evaluations is an important criterion for the heuristics while judging their performances. For example, MSH-QAP and TLBO-RTS (Dokeroglu, 2015) use the same fast evaluation technique that only calculates the delta part of the new solution. The study given in Tosun et al. (2013) uses a calculation that evaluates every new solution from scratch. This is a very costly operation and therefore the number of evaluations in this paper is lower than our study and its performance is worse than our solution quality. The authors report that even the number of evaluations is increased, their stagnation mechanism does not work as well as our local search techniques. MSH-QAP and TLBO-RTS have almost the same number of evaluations. The number of processors used by TLBO-RTS is larger than that of our experiments. Even with these experimental setups, MSH-QAP outperforms the TLBO-RTS. The number of evaluation performed by MSH-QAP can be reported as having a good performance when compared with the state-of-the-art heuristics.

#### 4.4. Robustness and scalability

MSH-QAP algorithm involves randomization therefore, achieving solutions within a desired solution quality range is an important aspect. In order to prove this issue, each problem instance in the experiments is solved 10 times and the average of the results are reported. The overall average deviation of the MSH-QAP is 0.013% for 134 problem instances in the QAPLIB while achieving optimal solutions of 122 instances. The MSH-QAP algorithm is capable of handling larger number of heuristics and can improve its solution quality by this way.

Another important aspect of the parallel algorithms is the speed-up and scalability obtained by the proposed algorithm. MSH-QAP algorithm uses a dynamic load balancing method. Whenever a processor finishes the execution of its current task, it sends a message to the master node to request a new task. This is necessary because the execution of the heuristics take varying amounts of CPU time. The total execution time of the algorithm is reduced proportionally with the given number of processors. As the number of processors is increased, the execution time decreases within a ratio very close to processor number. The architecture of the software is a master and slave paradigm. Slaves send their results to the master and receive further chromosomes to work on. This architecture is suitable for an HPC environment like ours. When the number of processors is increased from 20 to 60 (three times more processors are used) only an overhead of 8.7% is observed in the execution time due to the messaging. Therefore, MSH-QAP algorithm is a scalable algorithm for HPC environments with moderate number of CPUs up to 1000. The performance of the algorithm can be further improved by giving larger number of processors. This will reduce the calculation time of the fitness evaluations and give a higher chance to the user to improve the number of multistarts and explore

the landscape of the QAP more effectively. Fig. 7 presents the deviation percentages for the problem instances where our algorithm cannot find the BKS.

#### 4.5. Overall evaluation of the algorithm

No Free Lunch Theorem (NFL) proposes that if a good performance is demonstrated by an algorithm on a certain class of problems, it will have a trade-off and the performance of the algorithm will decrease for other class of problems (Wolpert and Macready, 1997). The (meta)-heuristic approaches can have high performances depending on the method they implement for the solution of a combinatorial optimization problem. However, due to the assertions of NFL theorem, they may not demonstrate the same performances when the domain and/or the structure of the problem is changed. From this perspective, applying many heuristics to the solution of a problem with parallel computation is a good idea where most probably one of them will have better performance than the others. With this idea, MSH-QAP algorithm proposes a framework that can import several heuristics into its population-based infrastructure and outperform a single heuristic.

The MSH-QAP algorithm is capable of combining several well-established (meta)-heuristics for the solution of the QAP. The same approach can also be applied for the other combinatorial optimization problems. The exploration process of MSH-QAP algorithm for different problem instances is more powerful than the (meta)-heuristic approaches. This is proved experimentally on a QAP benchmark with an overall deviation 0.013% for all the instances.

The MSH-QAP algorithm has several diversification and intensification techniques with respect to the given heuristics. This provides a more efficient use of the available computing power for the search. The intelligent parameter setting property of MSH-QAP algorithm is another advantage over the other heuristics and it has a significant impact on the optimization of the problems (Shi and Eberhart, 1998; Lobo et al., 2007). The biggest advantage of modifying the parameters generally comes from increasing the number of iterations of the heuristics. As it might be expected, higher quality solutions are received by larger number of iterations that the heuristics use during the optimization process. However, it has been observed that the best success is achieved by using the most suitable heuristic for a given problem instance, and not by a better selection of parameters.

The parallel computation of the MSH-QAP with a dynamic load balancing is another big advantage the algorithm. Because within an era of multi-core processors, it is not wise using several cores during the optimization. In our opinion, the parallelization of the state-of-the-art heuristic algorithm will be a new promising area for the researcher. This will either be performed on the instruction level of the computation or with coarse-grained parallel processes like our approach.

## 5. Conclusions and future work

In this study, we propose a novel parallel MultiStart Hyper-heuristic algorithm (MSH-QAP) on the grid for the solution of the QAP. MSH-QAP is the first proposed parallel hyper-heuristic algorithm in the literature for the solution of the QAP. The (meta)-heuristic approaches can have high performances depending on the method they implement for the solution of a combinatorial optimization problem. However, as the No Free Lunch (NFL) theorem suggests, they may not demonstrate the same performance when the domain and/or the structure of the problem is changed. At this point, we think that applying many heuristics to the same problem with parallel computation can be a good idea where most probably one of the proposed heuristics will have a better performance than others.

By using hyper-heuristics, we decide the right heuristic for a given QAP instance (because each instance can have a very different structure) and adaptively set the parameters of the heuristic. MSH-QAP algorithm makes use of the state-of-the-art (meta)-heuristics, Simulated Annealing (SA), Robust Tabu Search (RTS), Ant Colony Optimization-based (FANT: Fast Ant System), and Breakout Local Search (BLS) that have been reported among the best performing algorithms for the solution of large QAP instances. The experiments are executed on a HPC with several processors by multistarting the heuristics. The proposed technique significantly increased the quality of the solutions on the problem instances of the QAPLIB benchmark. It is reported to obtain 122 of the problems exactly while producing only 0.013% average deviation (among the first three algorithms in the literature) from the best known results.

Larger problem instances of the QAP are still very challenging and demand more computation resources/techniques by diversified search techniques therefore, as future work we plan to add more low-level heuristics (BMA and GDA) and apply well-known machine learning techniques such as reinforcement learning and support vector machines. Furthermore, we plan to execute MSH-QAP on a more powerful Cloud computing environment with different scalable communication topologies that are more appropriate for larger number of processors up to tens of thousands.

## Acknowledgments

We thank to our anonymous referees for their valuable support and comments that helped us improve our study significantly.

## References

- Acan, A., Unveren, A., 2015. A great deluge and tabu search hybrid with two-stage memory support for quadratic assignment problem. *Appl. Soft Comput.* 36, 185–203.
- Battiti, R., Tecchiolli, G., 1992. Parallel biased search for combinatorial optimization: genetic algorithms and tabu. *Microprocess. Microsyst.* 16 (7), 351–367.
- Benlic, U., Hao, J.K., 2013. Breakout local search for the quadratic assignment problem. *Appl. Math. Comput.* 219 (9), 4800–4815.
- Benlic, U., Hao, J.K., 2015. Memetic search for the quadratic assignment problem. *Expert Syst. Appl.* 42 (1), 584–595.
- Beyaz, M., Dokeroglu, T., Cosar, A., 2015. Robust hyper-heuristic algorithms for the offline oriented/non-oriented 2D bin packing problems. *Appl. Soft Comput.* 36, 236–245.
- Borgulya, I., 2014. A parallel hyper-heuristic approach for the two-dimensional rectangular strip-packing problem. *CIT. J. Comput. Inf. Technol.* 22 (4), 251–265.
- Burkard, R.E., Karisch, S.E., Rendl, F., 1991. QAPLIB a quadratic assignment problem library. *Eur. J. Oper. Res.* 55 (1), 115–119.

- Burke, E.K., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S., 2003. Hyper-heuristics: an emerging direction in modern search technology. In: Glover, F., Kochenberger, G. (Eds.), *Handbook of Metaheuristics*. Kluwer, pp. 457–474.
- Burke, E.K., McCollum, B., Meisels, A., Petrovic, S., Qu, R., 2007. A graph-based hyper-heuristic for educational timetabling problems. *Eur. J. Oper. Res.* 176 (1), 177–192.
- Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Woodward, J.R., 2010. A classification of hyper-heuristic approaches. In: *Handbook of Metaheuristics*. Springer, US, pp. 449–468.
- Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Zcan, E., Qu, R., 2013. Hyper-heuristics: a survey of the state of the art. *J. Oper. Res. Soc.* 64 (12), 1695–1724.
- Burke, Edmund K., Graham Kendall, Eric Soubeiga, 2003. A tabu-search hyperheuristic for timetabling and rostering. *J. Heurist.* 9.6, 451–470.
- Chib, S., Greenberg, E., 1995. Understanding the metropolis-hastings algorithm. *Am. Stat.* 49 (4), 327–335.
- Connolly, D.T., 1990. An improved annealing scheme for the QAP. *Eur. J. Oper. Res.* 46 (1), 93–100.
- Cowling, P., Kendall, G., Soubeiga, E., 2000. A hyperheuristic approach for scheduling a sales summit. In: *Selected Papers of the Third International Conference on the Practice And Theory of Automated Timetabling*, PATAT, Konstanz, Germany, Lecture Notes in Computer Science, pp. 176–190.
- Crainic, T.G., Toulouse, M., 2003. Parallel strategies for metaheuristics. In: *Handbook of metaheuristics*. Kluwer Academic, Dordrecht, pp. 475–513.
- Dokeroglu, T., 2015. Hybrid teaching-learning-based optimization algorithms for the quadratic assignment problem. *Comput. Ind. Eng.* 85, 86–101.
- Dorigo, M., Di Caro, G., 1999. Ant colony optimization: a new metaheuristic. In: *Proceedings of the 1999 Congress on Evolutionary Computation*, 1999. CEC 99, vol. 2.
- Drezner, Z., 2005. The extended concentric tabu for the quadratic assignment problem. *Eur. J. Oper. Res.* 160 (2), 416–422.
- Duman, E., Uysal, M., Alkaya, A.F., 2012. Migrating birds optimization: a new metaheuristic approach and its performance on quadratic assignment problem. *Inf. Sci.* 217, 65–77.
- Fescioglu-Unver, N., Kokar, M.M., 2011. Self controlling tabu search algorithm for the quadratic assignment problem. *Comput. Ind. Eng.* 60 (2), 310–319.
- Gambardella, L.M., Taillard, E.D., Dorigo, M., 1999. Ant colonies for the quadratic assignment problem. *J. Oper. Res. Soc.* 50 (2), 167–176.
- Glover, F., 1975. Tabu search—Part II. *ORSA J. Comput.* 2 (1), 4–32.
- Goldberg, D., 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, New York.
- Grefenstette, J.J., 1986. Optimization of control parameters for genetic algorithms. *IEEE Trans. Syst. Man Cybern.* 16 (1), 122–128.
- Holland, J.H., 1975. *Adaptation in Natural and Artificial Systems*. The MIT Press.
- Hussin, M.S., Stützle, T., 2014. Tabu search vs. simulated annealing as a function of the size of quadratic assignment problem instances. *Comput. Oper. Res.* 43, 286–291.
- James, T., Rego, C., Glover, F., 2005. Sequential and parallel path relinking algorithms for the quadratic assignment problem. *IEEE Intell. Syst.* 20 (4), 58–65.
- James, T., Rego, C., Glover, F., 2006. Multi-Start Tabu Search and Diversification Strategies for the Quadratic Assignment Problem. Working Paper, Virginia Tech.
- James, T., Rego, C., Glover, F., 2009. A cooperative parallel tabu search algorithm for the QAP. *Eur. J. Oper. Res.* 195 (3), 810–826.
- Jones, T., Forrest, S., 1995. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In: *ICGA'95*, pp. 184–192.
- Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P., 1983. Optimization by simulated annealing. *Science* 220 (4598), 671–680.
- Koohestani, B., Poli, R., 2014. Evolving an improved algorithm for envelope reduction using a hyper-heuristic approach. *IEEE Trans. Evol. Comput.* 18 (4), 543–558.
- Koopmans, T.C., Beckmann, M.J., 1957. Assignment problems and the location of economic activities. *Econometrica* 25 (1), 53–76.
- Kumar, V., 2002. *Introduction to Parallel Computing*. Addison-Wesley.
- Lobo, F.G., Lima, C.F., Michalewicz, Z., 2007. *Parameter Setting in Evolutionary Algorithms*, vol. 54. Springer Science & Business Media.
- Loiola, E.M., de Abreu, N.M.M., Boaventura-Netto, P.O., Hahn, P., Querido, T., 2007. A survey for the quadratic assignment problem. *Eur. J. Oper. Res.* 176 (2), 657–690.
- Lstiburek, M., Stejskal, J., Misevicius, A., Korecky, J., El-Kassaby, Y.A., 2015. Expansion of the minimum-inbreeding seed orchard design to operational scale. *Tree Genet. Genomes* 11 (1), 1–8.
- Maashi, M., Ozcan, E., Kendall, G., 2014. A multi-objective hyper-heuristic based on choice function. *Expert Syst. Appl.* 41 (9), 4475–4493.
- Malucelli, F., 2003. *Quadratic Assignment Problems: Solution Methods and Applications* (Ph.D. thesis). Dipartimento di Informatica, Università di Pisa, Italy.
- Merz, P., Freisleben, B., 2000. Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Trans. Evol. Comput.* 4 (4), 337–352.
- Misevicius, A., 2012. An implementation of the iterated tabu search algorithm for the quadratic assignment problem. *OR Spect.* 34 (3), 665–690.
- Novoa, C., Qasem, A., Chaparala, A., 2015. A SIMD tabu search implementation for solving the quadratic assignment problem with GPU acceleration. In: *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure*, p. 13.
- Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J.A., Walker, J., Gendreau, M., Kendall, G., McCollum, B., Parkes, A.J., Petrovic, S., Burke, E.K., 2012. Hyflex: A Benchmark Framework for Cross-Domain Heuristic Search. *Evolutionary Computation in Combinatorial Optimization*. Springer, Berlin, Heidelberg, pp. 136–147.
- Pappa, G.L., Ochoa, G., Hyde, M.R., Freitas, A.A., Woodward, J., Swan, J., 2014. Contrasting meta-learning and hyper-heuristic research: the role of evolutionary algorithms. *Genet. Prog. Evol. Mach.* 15 (1), 3–35.
- Pfister, G.F., 1998. *In Search of Clusters*, 2nd edition. Prentice-Hall.
- Rossin, D.F., Springer, M.C., Klein, B.D., 1999. New complexity measures for the facility layout problem: an empirical study using traditional and neural network analysis. *Comput. Ind. Eng.* 36 (3), 585–602.
- Ryser-Welch, P., Miller, J.F., 2014. A review of hyper-heuristic frameworks. In: *Proceedings of the 50th Anniversary Convention of the AISB*.
- Salcedo-Sanz, S., Matias-Roman, J.M., Jimenez-Fernandez, S., Portilla-Figueras, A., Cuadra, L., 2014. An evolutionary-based hyper-heuristic approach for the Jawbreaker puzzle. *Appl. Intell.* 40 (3), 404–414.
- Segura, C., Miranda, G., Leon, C., 2011. Parallel hyperheuristics for the frequency assignment problem. *Memet. Comput.* 3 (1), 33–49.
- Shi, Y., Eberhart, R.C., 1998. Parameter selection in particle swarm optimization. In: *Evolutionary Programming VII*. Springer, Berlin, Heidelberg, pp. 591–600.
- Sim, K., Hart, E., Paechter, B., 2015. A lifelong learning hyper-heuristic method for bin packing. *Evol. Comput.* 23 (1), 37–67.
- Stützle, T., 2006. Iterated local search for the quadratic assignment problem. *Eur. J. Oper. Res.* 174 (3), 1519–1539.
- Steinberg, L., 1961. The backboard wiring problem: a placement algorithm. *SIAM Rev.* 3 (1), 37–50.
- Taillard, E., 1991. Robust taboo search for the quadratic assignment problem. *Parallel Comput.* 17 (4–5), 443–455.
- Taillard, E.D., 1998. FANT: Fast Ant System. *Istituto Dalle Molle Di Studi Sull'Intelligenza Artificiale*.
- Taillard, E.D., 2002. Ant systems. *Handbook of Applied Optimization*, 130–137.
- Tayarani-N, M.H., Prugel-Bennett, A., 2012. Quadratic assignment problem: a landscape analysis. *Evol. Intell.*, 1–20.
- Tayarani-N, M.H., Prugel-Bennett, A., 2014. On the landscape of combinatorial optimization problems. *IEEE Trans. Evol. Comput.* 18 (3), 420–434.
- Topcuoglu, H.R., Ucar, A., Altin, L., 2014. A hyper-heuristic based framework for dynamic optimization problems. *Appl. Soft Comput.* 19, 236–251.
- Tosun, U., Dokeroglu, T., Cosar, A., 2013. A new robust island parallel genetic algorithm for the quadratic assignment problem. *Int. J. Prod. Res.* 51 (14), 4117–4133.
- Tosun, U., 2015. On the performance of parallel hybrid algorithms for the solution of the quadratic assignment problem. *Eng. Appl. Artif. Intell.* 39, 267–278.
- Tseng, L., Liang, S., 2005. A hybrid metaheuristic for the quadratic assignment problem. *Comput. Optim. Appl.* 34 (1), 85–113.
- Wilhelm, M.R., Ward, T.L., 1987. Solving quadratic assignment problems by simulated annealing. *IIE Trans.* 19 (1), 107–119.
- Wolpert, D.H., Macready, W.G., 1997. No free lunch theorems for optimization. *IEEE Trans. Evol. Comput.* 1 (1), 67–82.
- Yang, C., Peng, S., Jiang, B., Wang, L., Li, R., 2014. Hyper-heuristic genetic algorithm for solving frequency assignment problem in TD-SCDMA. In: *Proceedings of the 2014 Conference Companion on Genetic and Evolutionary Computation Companion*, pp. 1231–1238.