Artificial Bee Colony optimization for the Quadratic Assignment Problem

Tansel Dokeroglu[1]*, Ender Sevinc[2], Ahmet Cosar[2]
[1] TED University, Computer Engineering Department, Ankara, TURKEY
[2] University of Turkish Aeronautical Association, Computer Engineering Department, Ankara, TURKEY

Abstract

We propose hybrid Artificial Bee Colony (ABC) optimization algorithms for the well-known Quadratic Assignment Problem (QAP). Large problem instances of the QAP are still very challenging. Scientists have not discovered any method to obtain the exact solutions for these difficult problems yet. The ABC has been reported to be an efficient meta-heuristic for the solution of many intractable problems. It has promising results making it a good candidate to obtain (near)-optimal solutions for well-known NP-Hard problems. The proposed ABC algorithm (ABC-QAP) and its parallel version (PABC-QAP) are the first applications of the ABC meta-heuristic together with Tabu search to the optimization of the QAP. The behavior of employed, onlooker and scout bees are modeled by using the distributed memory parallel computation paradigm for large problem instances of the QAP. Scout bees search for food sources, employed bees go to food source and return to hive and share their information on the dance area, onlooker bees watch the dance of employed bees and choose food sources depending on the dance. Robust Tabu search method is used to simulate exploration and exploitation processes of the bees. 125 of 134 benchmark problem instances are solved optimally from the QAPLIB library and 0.27% deviation is reported for 9 large problem instances that could not be solved optimally. The performance of the ABC optimization algorithms is competitive with state-of-the-art meta-heuristic algorithms in literature.

Keywords

Artificial Bee Colony; Quadratic Assignment; Optimization; Parallel computation; Meta-heuristic.

1 Introduction

The Quadratic Assignment Problem (QAP) was first introduced by Koopmans & Berkman in 1957 [1][2][3]. The QAP is a mathematical model for the location of indivisible economic activities. The QAP is in the class of NP-Complete problems. It is one of the most difficult combinatorial optimization problems and there's no exact algorithm that can solve problems of sizes larger than 35 locations with practical computational time.

The QAP has been studied for several years in transportation systems, telecommunications, signal-processing [4], typewriters, keyboard design, layout design, board wiring [5], layout design [6], turbine balancing [7], scheduling [8], data allocation [9], travelling salesman, bin-packing, maximum clique, linear ordering and the graph partitioning problem [10].

The QAP is the problem of assigning a set of facilities to a set of locations in such a way as to minimize the total assignment cost. The QAP can be formulated by using nxn matrices, A, B, C.

$$A = (a_{ik}), B = (b_{jl}), C = (c_{ij}) \tag{1}$$

where a_{ik} is the flow amount from the facility i to facility k, b_{jl} is the distance from location j to location l, c_{ij} is the cost of placing facility i at location j. The QAP form of Koopmans & Beckman can be given as below;

$$min_{\phi \epsilon S_n}(\sum_{i=1}^{n}\sum_{k=1}^{n} a_{ik} b_{\phi(i)\phi(k)} + \sum_{i=1}^{n} c_{i\phi(i)}) \tag{2}$$

S_n is the permutation of numbers 1,2,...,n. $a_{ik}b_{\phi(i)\phi(k)}$ is the cost of transportation from facility i at location $\phi(i)$ to facility k at location $\phi(k)$. $c_{i\phi(i)}$ is the cost of installing facility i, at location $\phi(i)$ and the transportation costs to all other facilities k, set at locations $\phi(1), \phi(2), ..., \phi(n)$. In cases where there is no C term, Lawler introduced at four-index cost array $D = (d_{ijkl})$ instead of the three matrices and obtained the general form of the QAP as [11];

$$min_{\phi \epsilon S_n}(\sum_{i=1}^{n}\sum_{k=1}^{n} d_{i\phi(i)k\phi(k)}) \tag{3}$$

The relationship with the Koopmans & Beckman problem is:

$$d_{ijkl} = a_{ik}b_{jl}(i, j, k, l = 1, 2, ..., n; i \neq k \text{ or } j \neq l) \tag{4}$$

$$d_{ijij} = a_{ii}b_{jj} + c_{ij}(i, j = 1, 2, ..., n) \tag{5}$$

Smaller problem instances of the QAP can be solved by exact algorithms in minutes/hours. However, due to its intractable behaviour, solution of the larger instances of the QAP can take even hundreds of years to complete with a single processor brute force algorithm. Therefore, many meta-heuristic algorithms have been proposed for solving the QAP. The proposed algorithms are influential such that they can discover (near)-optimal solutions in practical running times.

Artificial Bee Colony (ABC) is a recent meta-heuristic proposed by Dervis Karaboga in 2005 [12]. The ABC is inspired by the intelligent behaviour of honeybees and uses a population-based search methodology. The ABC has been reported to be efficient for the solution of intractable (NP-Complete) problems like travelling salesman [13], scheduling and constraint optimization problems [14]. The ABC algorithm uses simple common parameters such as colony size and maximum cycle (iteration) number.

In our study, the locations of the QAP (permutations) are assumed to be food sources of artificial bees. These permutations are handled by the artificial bees. The purpose of a bee is to discover food sources with the highest amount of nectar. Artificial bees explore and exploit a multidimensional search space. The employed and onlooker bees decide food sources with the experience of themselves and their hive mates. Scout bees fly around the search space randomly and look for the food sources. If the nectar amount of a new source is better than those of the previous locations, they mark the new position. The ABC algorithm combines local search methods (Robust Tabu Search is used in our study) and global search methods by modelling these processes using employed, onlooker and scout bees. With well-balanced exploration and exploitation processes, the ABC can provide significant improvements in optimization times and solution quality.

We obtain 125 optimal results for the given 134 problem instances in the QAPLIB benchmark library with ABC-QAP algorithm. In order to find better results for larger instances of the QAP, we propose a novel island parallel algorithm. There has been a growing interest in parallel applications of meta-heuristic algorithms to many combinatorial problems recently [15]. The parallel ABC algorithm (PABC-QAP) that is proposed in our study is the first parallel application of the ABC meta-heuristic to the QAP. The parallelization of the meta-heuristics can improve the efficiency of optimization algorithm significantly [16][17]. The behavior of employed, onlooker and scout bees are modeled by using the distributed memory parallel computation paradigm, Message Passing Interface (MPI). Scout bees search for food sources, employed bees fly to food source and return to hive and share their information on the dance area, onlooker bees watch the dances of employed bees and choose food sources depending on dances. Robust Tabu search method [18] is used to simulate search operations of the bees. Exploration phase of the ABC optimization is analyzed and the best performing parameter of search effort is decided. The parameters of the Tabu search method is adaptively set for the Tabu list size and aspiration values. Our algorithm adjusts these parameters during the execution of the optimization process and provides better solutions than the existing methods. We observe that the aspiration value of the Tabu search can prevent getting stuck into local optima when it is well adjusted. 0.018% deviation from the best known solutions is reported for all the problem instances. The performance of the ABC optimization is competitive with state-of-the-art meta-heuristic algorithms in literature.

In section 2, related studies for state-of-the-art QAP are presented. The details of proposed ABC algorithms for serial and parallel versions are introduced in sections 3 and 4 respectively. The experimental setup, obtained results, and comparison with state-of-the-art algorithms are reported in section 5. Concluding remarks are provided in the last section.

2 Related work

In this part, we give information about the previous studies of ABC, Tabu search and some meta-heuristic algorithms designed for the solution of the QAP [17][19]. Karaboga & Basturk develop an ABC algorithm to optimize multivariable functions. Their results are compared with Genetic, Particle Swarm Algorithm (PSO) and PSO-Evolutionary algorithms. The ABC algorithm is observed to outperform the others [20]. In another study, Karaboga & Basturk compare the ABC with differential evolution, PSO and genetic algorithm for multi-dimensional numerical problems. The obtained results show that the performance of the ABC is competitive with the other algorithms [21]. ABC Programming (ABCP) is proposed on symbolic regression problem. A set of symbolic regression benchmark problems are solved using ABCP. The simulation results indicate that the new method is feasible for the test problems of symbolic regression [22].

Tayarani-N et al. analyze the fitness landscape of the QAP. The scientists try to understand the correlations between the local optima [23]. There are some recent parallel ABC algorithms in literature. Subotic et al. propose three different parallel ABC algorithms. The parallel ABC algorithms are independent parallel runs and two variations of multiple swarms parallelization. By using independent parallel runs method, they obtain faster execution of the ABC algorithm due to the high computation capability of multicore processors. They observe better results than the sequential version of the original ABC algorithm. Several communications between hives are proposed. The communication methods improve the solution quality with different ratios between exploration and exploitation. Eleven standard benchmark functions are tested and execution speed and the quality of results are improved [24]. Benitez et al. develop two parallel ABC algorithms for protein structure prediction. They use three-dimensional hydrophobic-polar model with side-chains. Experiments are carried out for tuning the parameters of the ABC and adjusting the load balance in the computing environment. The parallel models are compared with a sequential ABC algorithm on four benchmark instances. The parallel models are observed to improve the quality of solutions [25].

Narasimhan & Harikrishna develop a parallel ABC algorithm for shared memory architectures. The bee colony is divided among the available processors. In a local memory of each processor, a set of solutions is located. Also each solution is also kept in a shared memory. The bees at a processor improve the solutions in the local memory. At the end of the execution, the solutions are moved to the shared memory and made available to the bee colony. The proposed algorithm obtains a substantial amount of speedup [26]. Parpinelli et al. propose parallel ABC algorithms: master-slave, multi-hive with migrations, and hybrid hierarchical. Statistical results show that intensive local search improves the quality of solutions [27].

Loiola et al. classify some of the most important QAP algorithms according to their mathematical sources in their survey [28]. They discuss lower bounds for heuristic and exact algorithms. Talbi et al. propose a parallel algorithm for ant colonies to solve the QAP. A pheromone matrix that simulates the global memory is provided to cooperate the processes between ants. The exploration is controlled by the improvement of pheromones levels. The exploitation is enhanced by a Tabu search algorithm [29]. Zhou et al. present a hybrid frequent pattern based search approach that combines data mining and optimization paradigms. The proposed method uses a data mining procedure to obtain frequent patterns and the minded patterns are used to build new and efficient starting points. The proposed approach competes with state-of-the-art algorithms both in terms of solution quality and computing time [30]. Chmiel & Kwiecień propose a quantum-inspired evolutionary algorithm for QAP. They present how the QAP is adapted, including crossover and mutation operators and introducing quantum principles in particular procedures [31]. Mihić et al. propose a new local search approach, called randomized decomposition

(RD), for solving nonlinear, nonconvex mathematical programs. They successfully applied to over 400 instances of the quadratic assignment problem (QAP) [32].

Yagmur et al. introduce a parallel version of the Breakout Local Search (BLS) algorithm [16]. They use a Levenshtein Distance metric for checking the similarity of the new starting points. The proposed BLS Algorithm (BLS-OpenMP) combines multi-threaded computation using OpenMP. Dokeroglu proposes Teaching Learning Based (TLBO) hybrid algorithms to solve the QAP [33]. Individuals are trained with recombination operators and later a Robust Tabu Search engine processes them. Hyper-heuristics is a recent approach for solving challenging NP-Hard combinatorial optimization problems by using a set of low level meta-heuristics. Abdel Basset et al. propose a method to improve the Whale algorithm. The proposed algorithm is enhanced by a local search. The algorithm is tested on many QAP instances and it is reported to obtain near-optimal solutions with reasonable execution times [34].

Çela et al. consider new polynomial-time cases of the QAP with a special diagonal structure. They obtain a new class of polynomially solvable special cases of the QAP [35]. Bougleux et al. propose a linear assignment model to a quadratic one. This is provided by a family of edit paths induced by assignments between nodes. They show that the Graph Edit Distance is equivalent to a QAP [36]. Parallel Evolutionary algorithms are important means of improvement due to the modern parallel computer architectures. Nalepa & Blocho propose a parallel memetic algorithm to solve the vehicle routing problem with time windows. In their study, a number of populations are evolved in a parallel computation environment. Their study comprises more than 1,584,000 CPU hours and report new best solutions using the best co-operation schemes [37].

Tosun proposes a parallel hybrid algorithm (PHA) [38]. A genetic algorithm and a robust Tabu search are incorporated in his study by a parallel environment. The PHA achieves 0.05% deviation on the average from the best/optimal results. Benlic & Hao propose BLS algorithm for the QAP [39]. The BLS optimizes the solution of the QAP by the use of local search and adaptive perturbation methods. The same researchers propose a new memetic algorithm for the QAP [40]. Fescioglu et al. address the principles of the feedback and self-controlling mechanisms of Tabu search. They introduce new reaction techniques. The first strategy uses a control-theoretic approach to tune the parameters of the algorithm that affect the intensification. The second strategy adjusts the parameters based on the search history [41].

Dokeroglu & Cosar propose a parallel MultiStart Hyper-heuristic algorithm (MSH-QAP). The MSH-QAP uses meta-heuristics, Simulated Annealing, Tabu Search, Ant Colony Optimization, and BLS [17]. Duman et al. propose a new meta-heuristic based on the V flight shape of the migrating birds for the QAP. The performance of the algorithm is reported to be better than Tabu search, particle swarm optimization, genetic algorithm, simulated annealing, scatter search, differential evolution and guided evolutionary simulated annealing [42].

Tabu search optimization method was first developed by Fred W. Glover in 1986 [43]. Tabu search is a heuristic for solving optimization problems and it has a well-designed mechanism to escape from being stuck into the local optima. Tabu search can obtain (near)optimal solutions to many classical combinatorial optimization problems [44]. Taillard proposes Tabu search for the QAP in 1991 [18] and develops two new parallel algorithms. He obtains new best/optimal results in his study. To the best of our knowledge, our study is the first application of the ABC meta-heuristic optimization with Tabu search to the QAP. We introduce a single-core and an island parallel application of the ABC algorithm for the solution of the QAP.

3 Proposed ABC algorithm for the optimization of the QAP

In this section, we introduce our proposed ABC-QAP algorithm. ABC is a population based meta-heuristic optimization method. Bees constitute the population of the algorithm. Each bee looks for the optimal solution (food resource) of the given QAP instance. A solution is assumed to be a food resource and the nectar amount of each resource represents the quality of each solution. In our model, we use three types of bees: employed bees, onlookers and scouts. Scout bees search for food sources. Employed bees go to food source and return hive and share their information on the dance area. When the nectar collecting duty of an employed bee finishes, the bee becomes a scout and searches for new food sources. Onlookers

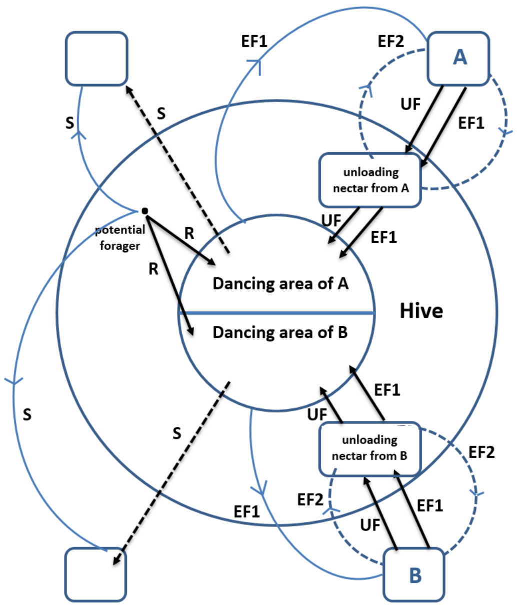

Fig. 1: The classical behaviour of honeybees looking for nectar

watch the dances of employed bees and choose food sources depending on dances. An initial population is generated at first step of the algorithm. After this process, the optimization is repeated by using the employed, onlooker, and scout bees. The scout bees start the search process.

There is a single hive in our developed model for the ABC-QAP. In the parallel version of the ABC-QAP, the number of hives is as many as the number of processors in the parallel computation environment. Figure 1 gives the fundamental characteristics of bees in the environment. A potential forager starts as an unemployed forager and has no information about the food sources around the hive. A bee can be a scout and searches the space for a food (*S* in Figure 1) or it can watch the dances and goes for finding food sources (*R* in Figure 1). The bee collects the food, returns to the hive, unloads the nectar. The bee can become an uncommitted follower (UF), recruit nest mates (EF1) or go exploiting the food source without recruiting after bees (EF2).

In our proposed model, each bee starts as a scout bee and explores the search space of the QAP. After spending some time during the exploration process, it comes back to the hive and shares its information. Evaluating the results, they go back to the best available food resources. The bees use different parameters of the Tabu search (as Tabu list size and aspiration values) and start exploiting the food resources in more detail. For the exploration and exploitation phases of the ABC-QAP, Tabu search technique is used. In the exploration phase, Tabu search roughly passes over the possible solutions. During this phase it uses smaller number of iterations and restarts the search from randomly generated initial spaces. Therefore, the Tabu search provides an efficient diversification mechanism that can evaluate the landscape of the QAP [23]. The number of scout bees that we use during the exploration phase is set to be 1,000 in our experiments. This parameter provides a good balance between exploration and exploitation phases of the optimization. The best food resource is selected after the exploration phase and it is exploited by the employed bees. Algorithm 1 gives the details of the proposed ABC-QAP algorithm.

Algorithm 1: Pseudocode of the ABC-QAP algorithm

```
1  Construct an empty database for food resources();
2  int i=0;
3  while (i++ < #iterations) do
4      // Exploration phase
5      Scout bees search for food();
6      Scout bees return to the hive and dance();
7      Onlooker bees evaluate the food sources();
8      // Exploitation phase
9      Check previously visited food resources();
10     Decide the best food resources();
11     Employeed bees travel to the food sources();
12     Tabu search for nectar collecting();
13     Return to hive();
14     Collect the solution in the hive();
15 Report the best solution();
```

3.1 Tabu search

Tabu search is a trajectory optimization technique. It uses an adaptive memory to explore the search space of the QAP. The memory prevents visiting the recently searched spaces. Tabu search can use several techniques to obtain better search spaces. Diversification and intensification are some of the techniques used during the optimization. Using different sizes of the Tabu list is a promising technique. Larger Tabu lists can provide a good stagnation avoidance mechanism, whereas a smaller Tabu list exploits local best values that may be located around local optimal value. Tabu search also uses some aspiration criteria to override a Tabu state and an aspiration criterion is allowed whenever a move may have a chance to obtain a better result. Taillard's Tabu search method has a short-term memory with multiple-levels of aspiration criteria. The short-term memory is good for high-quality solutions. However, longer term memory can have promising results. Tabu search algorithms can have a variety of parameter settings that explore the intensification and diversification strategies [45][46]. The long-term memory holds the frequency of the parts that appear in better solutions. Diversification can eliminate these long-term parts and can direct to unexplored parts. Tabu search seeks for a neighbouring solution that has the local best value. During this search, the calculation of the results forces a move that obtains the objective function most. The Tabu list forbids the reverse moves of a search process to avoid revisiting to the previously evaluated areas. Robust Tabu search uses the number of failures and the Tabu list size as parameters. The number of failures is the number of iterations in which no improvement is obtained. Taillard reports that it is possible to get better results with larger number of failures. Our proposed algorithm, ABC-QAP uses Robust Tabu search for the search processes of the scout and employed bees. Algorithm 2 presents the details of the Tabu search algorithm.

3.2 Tuning the Tabu list size and aspiration value

The size of a Tabu list is an important parameter for finding the optimal values. For every unit and location in the QAP, the last iteration that occupies this location is saved in the list. Small size Tabu lists can cause the search to process in the same areas and get stuck into local optima, whereas larger lists can prevent better moves to be explored and lead to lower quality solutions. The optimization may execute more iterations than it is needed due to the excessive Tabu list size. The lower and upper limits of the Tabu list size are decided to be between [0.9 x n - 1.1 x n] by Taillard (where n is the problem size).

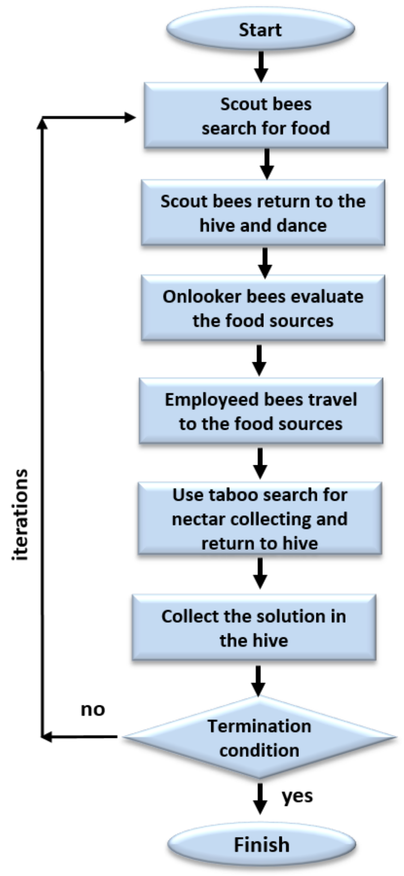
Start
Scout bees search for food
Scout bees return to the hive and dance
Onlooker bees evaluate the food sources
Employeed bees travel to the food sources
Use taboo search for nectar collecting and return to hive
Collect the solution in the hive
Termination condition
no
yes
iterations
Finish

Fig. 2: Flowchart of the ABC-QAP algorithm

These parameters are observed to be the mean values that provides reasonable execution times during the optimization process. A dynamic size Tabu list between these value provides a very effective way for the optimization process and the aspiration value [47]. Therefore, we apply a technique that changes the size of the Tabu list dynamically in this study.

Aspiration value of Tabu search allows a move to be executed if it finds a better move than the existing solutions. The aspiration value of Tabu search process is generally decided as recommended by classical methods. However, this value can change depending on the structure of the problem. Therefore, we propose a dynamic aspiration value in our study. The value changes at each 100,000 iteration of Tabu search. The aspiration value can be a good way of local search with smaller values. However, when the search process is stuck into local optima, optimization needs to get out of this space while preserving its previously obtained experiences (by not destroying the existing permutation of the current solution). A well-tuned aspiration value can provide such a mechanism. Our dynamic aspiration value changes between [n - (n x n x 10)]. Smaller values of the aspiration can search the closer spaces of the current solution while larger values are providing an escape mechanism from local optima. With this way, a diversified space search is constructed with hundreds of processors that are optimizing the same problem instance with different Tabu list sizes and aspiration values.

3.3 Quick evaluation of neighbouring solutions

Swapping (exchanging) two different locations of a current QAP solution and generating a new permutation is a very effective approach to traverse the solution search space of the QAP. This approach allows us to calculate the cost of the new permutation quickly by only finding the cost of difference [18]. Formally, calculating the fitness value of a QAP permutation is $O(n^2)$ for a calculation to be made from scratch, whereas it becomes *O(n)* when the difference of two permutations is calculated. This method

Algorithm 2: Tabu Search [18][33]

```
Authorized: If a move that is not Tabu.
Aspired: Permit Tabu moves if they are interesting.
Tabu List: Vector of moves to avoid backward moves.
Neighbor: Each location of the permutation is considered as a neighbor.
Tabu Search (Flow, Dist, MaxIter, BestPerm, MinSize: minimum Tabu list size, Maxsize: maximum
  Tabu list size, Aspiration value);
Tabu_list = empty;
Calculate_current_Cost(BestPerm);
Cur_Sol = BestPerm;
$\Delta$[j][k] = Compute_$\Delta$(); /* j,k = 0,...,n */
Tabu_list[j][k] = - (n$\times$j+k);
for (int i = 1; i < MaxIter; i++) do
    j_retained = $\infty$;
    Min_$\Delta$ = $\infty$;
    Already_Aspired = false;
    for (all neighbors (j, k)) do
        $cur_1$ = Tabu_list[j][Cur_Sol[k]];
        $cur_2$ = Tabu_list[k][Cur_Sol[j]];
        Authorized = ($cur_1$ < i) || ($cur_2$ < i);
        Aspired = ($cur_1$ < i - Aspiration)|| ($cur_2$ < i-Aspiration)|| (Cur_Cost + $\Delta$[j][k] < BestCost);
        if ( (Aspired && Already_Aspired) || (Aspired && $\Delta$[j][k] < Min_$\Delta$) ||
        (!Aspired && !Already_Aspired && $\Delta$[j][k] < Min_$\Delta$ && Authorized) ) then
            j_retained = j;
            k_retained = k;
            Min_$\Delta$ = $\Delta$[j][k];
            if (Aspired) then
                Already_Aspired = true;
    if (j_retained != $\infty$)) then
        Exchange(Cur_Sol[j_retained], Cur_Sol[k_retained]);
        Cur_Cost = Cur_Cost + $\Delta$[j_retained][k_retained];
        Tabu_list[j_retained][Cur_Sol[k_retained]] = j + getRandom(MinSize, MaxSize);
        Tabu_List[k_retained][Cur_Sol[j_retained]] = j + getRandom(MinSize, MaxSize);
        if (Cur_Cost < BestCost) then
            BestCost = Cur_Cost;
    Update_Move_Costs(Flow, Dist, Cur_Sol, $\Delta$, j, k, j_retained, k_retained);
```

is used as a performance increasing calculation method in our proposed algorithm, PABC-QAP. Robust Tabu search uses a matrix to store the cost of each possible exchange and these costs are added to obtain the cost of the new solution. Starting from a solution ϕ, a neighbour solution π is obtained by permuting units r and s:

$$\begin{aligned} \pi(k) &= \phi(k) \forall k \neq r, s \\ \pi(r) &= \phi(s) \\ \pi(s) &= \phi(r) \end{aligned} \tag{6}$$

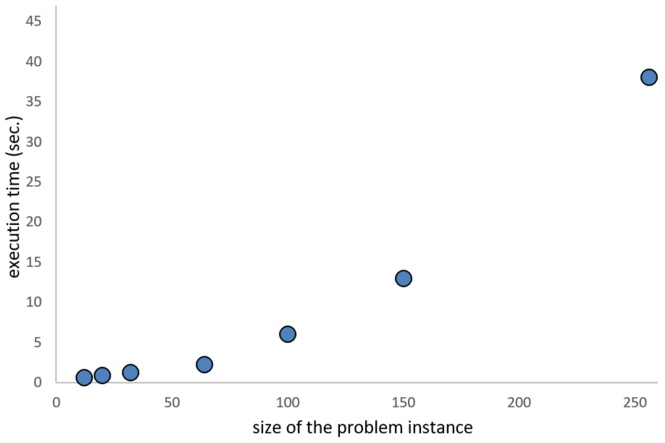

Fig. 3: The execution time of the problem instances with respect to their sizes

The value of a move $\triangle(\phi, r, s)$ is as given below when the matrices are symmetrical;

$$\begin{aligned} \pi(k) = \phi(k) \forall k \neq r, s \\ \pi(r) = \phi(s) \\ \pi(s) = \phi(r) \end{aligned} \tag{7}$$

In Figure 3, the execution times of problem instances ranging from size 12 to 256 are given. 10,000 neighbors are searched for the problem instances with the given quick evaluation method introduced above. A linear rise in time can be observed during the executions. This technique provides a big advantage during the exploration and exploitation phases of the ABC optimization.

4 Parallel ABC algorithm for the QAP

For hard problem instances of the QAP, we need to develop a more powerful and scalable parallel algorithm. Therefore, we propose an island parallel ABC optimization algorithm for the QAP. In this algorithm, there are a *master* node (processor) and many *slave* nodes in the parallel computation environment. The name of the proposed algorithm is PABC-QAP. Each one of the slave nodes in the computation environment works on a separate hive and with a different set of bees (scout, onlooker and employeed). The PABC-QAP algorithm starts a diverse exploration phase at each processor. This is provided by randomly generated initial starting permutations of the QAP. The seeding mechanism at each processor is initialized with (current time x # processor). This seeding value is able to provide well-diversified permutations for each processor. During the exploration phase, Tabu search algorithm is run with a small number of number of failures (100) and 1,000 restarts. This way of executing the Tabu search with smaller number of iterations provides a good way of exploring the search space of the QAP. 1,000 explorations are executed with 255 processors. Totally, 255,000 exploration processes are run in parallel. Due to the minimum amount of communication between processors, the algorithm is observed to be scalable and works almost with a linear speed-up. In some of the problem sets that we cannot obtain the best/optimal values with ABC-QAP, it was possible to obtain the best values even in the exploration phase of the parallel PABC-QAP algorithm. For very hard problem instances like *tai100a* and *tai256c*, we still need to spend most of our optimization time in the exploitation phase of the algorithm. Since a good balance between exploration and exploitation phases ensures a time-efficient optimization process, we carry out some experiments to understand the behaviour of the exploration in terms of time and devitaion from the best results. Its effect to the exploitation phase is analyzed in the experiments section of our study.

After the exploration phase of the scout bees is completed, the best permutation is sent to the exploitation phase. Then, Tabu search starts optimizing the current solution with higher number of restarts and failures given in Table 1. After the optimization process is completed at each slave processor, the results and the execution time of the optimizations are sent to the master node. Slave nodes may spend different execution times because of the diversified optimization process at each node. The master node receives the results from the slave nodes and reports the best one as the result of the PABC-QAP algorithm. Figure 4 and Algorithm 3 present the flowchart and the details of the PABC-QAP algorithm respectively.

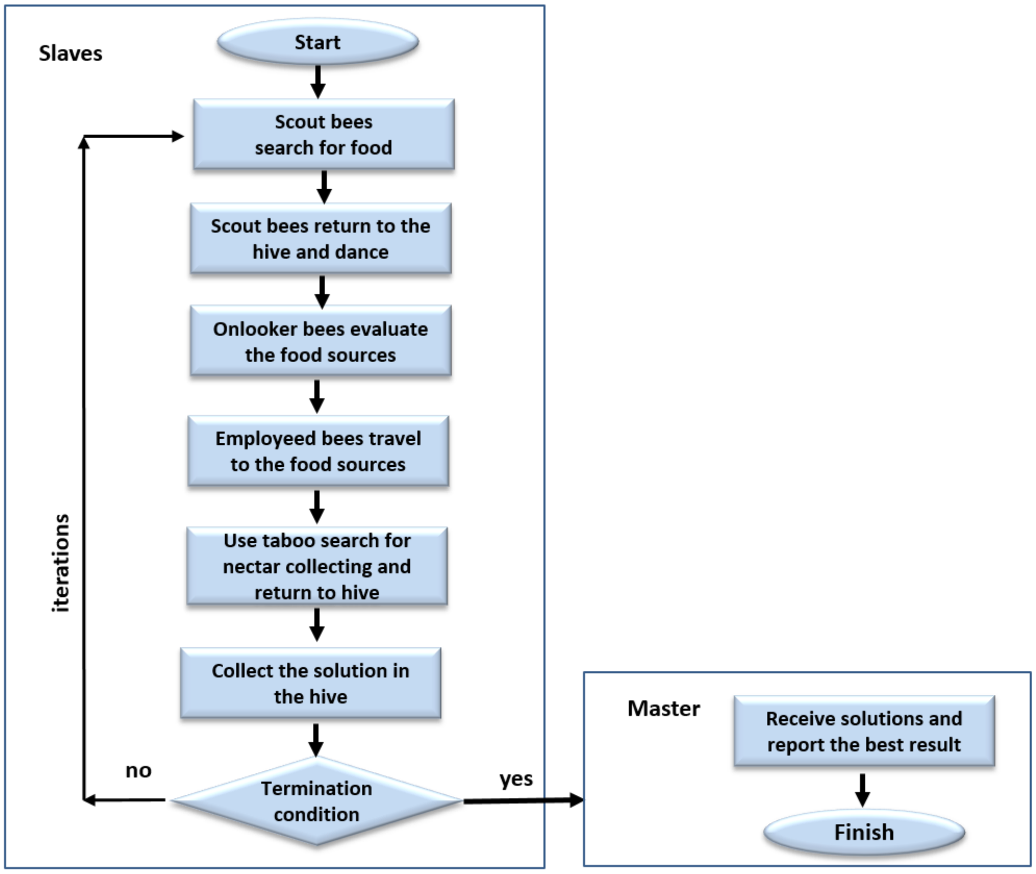
Slaves
Start
Scout bees search for food
Scout bees return to the hive and dance
Onlooker bees evaluate the food sources
Employeed bees travel to the food sources
Use taboo search for nectar collecting and return to hive
Collect the solution in the hive
iterations
no
Termination condition
yes
Master
Receive solutions and report the best result
Finish

Fig. 4: Flowchart of the parallel PABC-QAP algorithm

5 Performance evaluation of the proposed algorithms

In this section, we present the experimental setup and the results obtained by the proposed ABC algorithms, ABC-QAP and PABC-QAP. During our experiments, we use QAPLIB (the benchmark problem library of the QAP) [48]. 134 problem instances are solved from QAPLIB benchmark problem instances [4]. Most of the state-of-the-art algorithms use this library. Therefore, it provides a fair platform for the evaluation of new algorithms. Problem instances of this library are generated from real life applications or randomly (such as Manhattan distances of rectangular grids (*Head12*), hospital layout (*kra30*), and the backboard wiring (*Ste36a*)). During our experiments, each problem instance is tested 30 times and the average/best results of the tests are reported. C++ programming language is used for the implementation of ABC-QAP and PABC-QAP algorithms.

We carry out our experiments on HP ProLiant DL585 G7 that has AMD Opteron 6212 CPU running at 2.6 GHz and having 8 cores. It is possible to create 8 threads at each core (providing 64 possible cores simultaneously). Each CPU has 64-bit computing capacity and AMD SR5690 chipset. The server uses 256 GB Pc:3-10600 RAM and 1.5 TB.

Algorithm 3: Pseudocode of the PABC-QAP algorithm

```
1  if (I am a slave processor) then
2      Construct an empty database for food resources();
3      int i=0;
4      while (i++ < #iterations) do
5          // Exploration phase
6          Scout bees search for food();
7          Scout bees return to the hive and dance();
8          Onlooker bees evaluate the food sources();
9          // Exploitation phase
10         Check previously visited food resources();
11         Decide the best food resources();
12         Employeed bees travel to the food sources();
13         Tabu search for nectar collecting();
14         Return to hive();
15         Collect the solution in the hive();
16         Send the best result to the master node();
17 if (I am the master processor) then
18     Receive the solutions from the slaves();
19     Report the best solution();
```

5.1 The effect of increasing the number of processors

The number of processors has a crucial impact on the performance of the proposed parallel algorithm, PABC-QAP. In Figure 5, we give the deviation of our experiments on *tai50a* problem instance with increasing number of processors (Tabu search uses 1,000,000 number of failures and ABC uses 100 scout bees for the exploration phase). The experiments are repeated 30 times and their average value is reported. x-axis is the number of explorations and y-axis gives the deviation of the optimization from the best results in the library. If a good diversification mechanism can be provided at each processor and as many as possible number of processors are used, then the probability of finding the optimal value is significantly increased. Certainly, providing well-tuned parameters is another big advantage during the optimization. Our main purpose in this part of our tests is to ensure that parallel environment can provide a better tool while finding the optimal solution. The first thing that comes to mind when considering parallel algorithms is their speed-up. However, island parallel ABC algorithm that we propose in this study works on several hives and optimizes independent solution candidates instead of parallelizing the calculation of a single solution's optimization effort. The latter can be a very difficult job most of the time and can not provide (near)-linear speed-ups, whereas the coarsely grained island parallel ABC can efficiently consume its computation time. The result has 1.1% deviation with a single processor for the given configuration. With 255 threads (that work on different hives that are located on each processor's memory) is 0.57%. A clear progress is observed during the experiments. The performance increases as the number of failures and iterations of the Tabu search becomes higher.

5.2 Tuning the number of explorations

One of our biggest concerns is setting the right balance between exploration and exploitation phases of the ABC algorithm in terms of time and deviation from the optimal results. The number of explorations has a crucial impact on the performance of the optimization process. In Figure 6, we give the deviation of

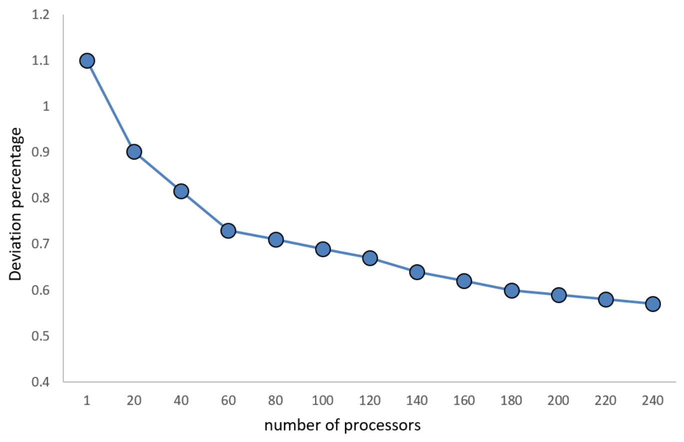

Fig. 5: The effect of increasing number of processors for the PABC-QAP algorithm

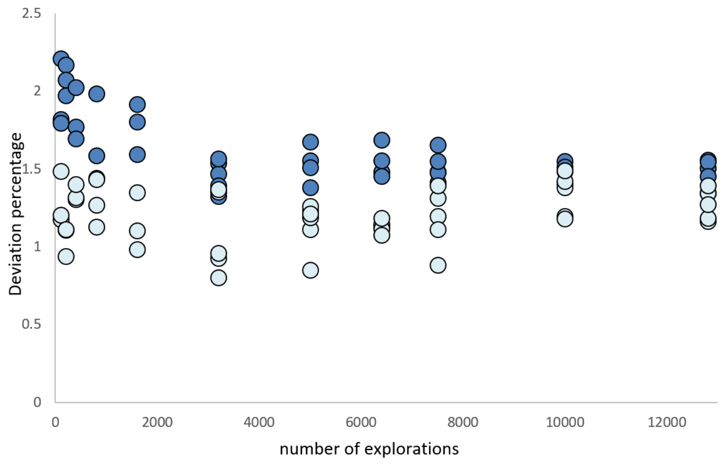

Fig. 6: The effect of the number of explorations on problem instance *tai50a*.

some results for *tai50a* problem instance with increasing number of explorations. We run each exploration 30 times and report the results on Figure 6. The problem instance, *tai50a*, is a quite difficult and moderate size problem in the library. x-axis is the number of explorations and y-axis gives the deviation of the optimization. Starting from 1 up to 130,000 explorations, we observe the performance of the exploration process on the optimization. Dark markers on Figure 6 are the exploration results, whereas the white markers are the results that are obtained after the exploitation phase. Even the number of exploration is run with 130,000 iterations to get better starting permutations for the exploitation phase, we decide that 1,000 iteration is a nice value with its reasonable execution time for the exploration. No big advantage is observed with higher as the number of explorations is increased more than 2,000 iterations. Therefore, we set the number of exploration activity as 1,000 for all the problem instance. Of course this is not the optimal value for all the problem instances but a simple parameter tuning method provided for the balance between exploration and exploitation phases. In the exploitation phase of the ABC-QAP, as it is reported by Taillard, higher number of failures provides better results when Tabu list size and aspiration values are tuned well.

5.3 The execution results on the QAPLIB problem instances

In this part of our study, we present the results of our ABC-QAP and parallel PABC-QAP algorithms in terms of deviation percentage from the best known solutions and execution times. First, we solve a problem instance with ABC-QAP to obtain the minimum deviation from the best known solutions reported in the QAPLIB. In case we cannot have the best results, we apply different Tabu list and aspiration settings for the ABC-QAP algorithm. Later, we execute the PABC-QAP algorithm with 255 processors and given settings in Tables 2 to 8. The name of the problem instance, Best Known Solution (BKS) value of the problem instance reported by the QAPLIB, how many times the best result is (found), Average Percentage Deviation from the best known solution (APD), Best Percentage Deviation of our result from the BKS (BPD), execution time of the algorithm in seconds (sec.), number of processors used during the optimization (#proc.), the aspiration value of the Tabu search process (aspiration) are given in the Tables. If the problem instance is optimized with PABC-QAP algorithm then the number of processors is reported to be more than 255 in the Tables. The harder problem instances that are not optimally solved by the ABC-QAP algorithm are tested with its parallel version and 255 processors. The average execution time and deviations are reported at the Tables.

Table 1 gives four configurations for the Tabu search algorithm that we use in our experiments. For small problem instances that are easily solved, we apply configuration 1. For harder problem instances we use configuration 2. Due to the execution time limitations, we need to use configuration 3 for the *tai256c* problem instance. The Tabu list size parameter used in our experiments is proposed by Taillard [18]. Aspiration value range is applied dynamically for the first time in our study in the ranges given in Table 1. Setting 4 is used for the exploration phase of the ABC-QAP algorithm.

All the problems in the library of the QAPLIB are solved during the experiments. Tables 2 to 8 give our experimental results on 134 problem instances in the QAPLIB. We spend nearly 339,599.5 hours of CPU time with parallel computation (14,150.0 days) during our experiments. 125 of the problems are solved optimally with respect to the results given in the benchmark library. 105 of these results are obtained by ABC-QAP and 21 of them are solved with parallel PABC-QAP algorithm. The problem instances, *bur*, *had*, *chr*, *els*, *esc*, *had*, *kra*, *lipa*, *nug*, *Rou*, *Scr*, *Sko* (except *sko100a*) and *Ste* are solved optimally by the ABC algorithms. The larger problem instances of *taia* and *taib* are the hardest instances that we have to deal with during the experiments. The average of best reported deviations is observed to be 0.092% for *tai* problem instances. Specially, *tai50a*, *tai60a*, *tai80a*, and *tai100a* have the largest deviations among all the problem instances. For *tai256c*, we report one of the best results in literature, 0.082% deviation. The ABC algorithms spend most of their execution time on these problem instances. *tai256c*, tai150b, and *tai100a* spend 71,129, 16,665, and 10,379 seconds during the optimization respectively.

TABLE 1: Tabu search parameter settings

configuration	maximum # failures	Tabu list size	range of aspiration value
1	50,000xn	lower limit=(9xn/10) – upper limit=(11xn/10)	[n - nxn]
2	50,000xn	lower limit=(9xn/10) – upper limit=(11xn/10)	[n - nxnx10]
3	20,000xn	lower limit=(9xn/10) – upper limit=(11xn/10)	[n - nxnx10]
4	1,000xn	lower limit=(9xn/10) – upper limit=(11xn/10)	[n - nxn]

5.4 Comparison with the state-of-the-art meta-heuristic algorithms

In this part of our experiments, we compare our proposed ABC algorithms with state-of-the-art algorithms in literature. These algorithms are (JRG-DivTS) [49], which is developed by James et al. It is an advanced version of the Tabu search algorithm that is a multi-start TS algorithm. SC-Tabu [41], TLBO-RTS [33], Iterated Tabu Search (ITS) by Misevicius [50], Lagrangian Smoothing Algorithm (LagSA) [51], GA/SD [52] and ACO/GA/LS [53]. Table 9 gives the results of the algorithms for *taia* problem instances. These problems are the hardest problems in literature. Overall deviation of the ABC algorithms is 0.248% for the problem instances. This is the second best performance among the reported algorithms. For *tai40a*, the best result is reported by the ABC algorithm. For *tai100a*, our algorithm is the second best one. Only the performance of the ABC on *tai80a* is not among the first three algorithms. For *taib* problem instances, TLBO-RTS, JRG-DivTS, ITS, SC-TABU, and ABC have deviation results 0.006, 0.07, 0.051, 0.009, and 0.00 respectively.

TABLE 2: Results obtained for the problem instances *bur*, *had*, *chr*, and *els*. BKS is the best known result for the problem instance. APD is average percentage deviation. BPD is the best percentage deviation. sec. is the execution time of the algorithm.

Instance	BKS	found	APD	BPD	sec.	#proc.	aspiration
Bur26a	5426670	10	0	0	2	1	[n - n x n]
Bur26b	3817852	10	0	0	1	1	[n - n x n]
Bur26c	5426795	10	0	0	7	1	[n - n x n]
Bur26d	3821225	10	0	0	12	1	[n - n x n]
Bur26e	5386879	10	0	0	0.1	1	[n - n x n]
Bur26f	3782044	10	0	0	0.2	1	[n - n x n]
Bur26g	10117172	10	0	0	1	1	[n - n x n]
Bur26h	7098658	10	0	0	1	1	[n - n x n]
Chr12a	9552	10	0	0	0.1	1	[n - n x n]
Chr12b	9742	10	0	0	0.1	1	[n - n x n]
Chr12c	11156	10	0	0	0.1	1	[n - n x n]
Chr15a	9896	10	0	0	0.2	1	[n - n x n]
Chr15b	7990	10	0	0	0.1	1	[n - n x n]
Chr15c	9504	10	0	0	2	1	[n - n x n]
Chr18a	11098	10	0	0	5	1	[n - n x n]
Chr18b	1534	10	0	0	0.1	1	[n - n x n]
Chr20a	2192	10	0	0	15	1	[n - n x n]
Chr20b	2298	10	0	0	8	1	[n - n x n]
Chr20c	14142	10	0	0	2	1	[n - n x n]
Chr22a	6156	10	0	0	8	1	[n - n x n]
Chr22b	6194	10	0	0	6	1	[n - n x n]
Chr25a	3796	10	0	0	13	1	[n - n x n]
Els19	17212548	10	0	0	0.10	1	[n - n x n]
Average	-	10	0.0	0.0	3.65	-	-

TABLE 3: Results obtained for the problem instance *esc*.

Instance	BKS	found	APD	BPD	sec.	#proc.	aspiration
Esc16a	68	10	0	0	0.1	1	[n - n x n]
Esc16b	292	10	0	0	0.1	1	[n - n x n]
Esc16c	160	10	0	0	0.1	1	[n - n x n]
Esc16d	16	10	0	0	0.1	1	[n - n x n]
Esc16e	28	10	0	0	0.1	1	[n - n x n]
Esc16f	0	10	0	0	0.1	1	[n - n x n]
Esc16g	26	10	0	0	0.1	1	[n - n x n]
Esc16h	996	10	0	0	0.1	1	[n - n x n]
Esc16i	14	10	0	0	0.1	1	[n - n x n]
Esc16j	8	10	0	0	0.1	1	[n - n x n]
Esc32a	130	10	0	0	0.1	1	[n - n x n x 10]
Esc32b	168	10	0	0	0.1	1	[n - n x n]
Esc32c	642	10	0	0	0.1	1	[n - n x n]
Esc32d	200	10	0	0	0.1	1	[n - n x n]
Esc32e	2	10	0	0	0.1	1	[n - n x n]
Esc32g	6	10	0	0	0.1	1	[n - n x n]
Esc32h	438	10	0	0	0.1	1	[n - n x n]
Esc64a	116	10	0	0	0.1	1	[n - n x n]
Esc128	64	10	0	0	1	1	[n - n x n]
Average	-	10	0.0	0.0	0.15	-	-

TABLE 4: Results obtained for the problem instances *had*, *kra* and *lipa*.

Instance	BKS	found	APD	BPD	sec.	#proc.	aspiration
Had12	1652	10	0	0	0.1	1	[*n* - *n* x *n*]
Had14	2724	10	0	0	0.1	1	[*n* - *n* x *n*]
Had16	3720	10	0	0	0.1	1	[*n* - *n* x *n*]
Had18	5358	10	0	0	0.1	1	[*n* - *n* x *n*]
Had20	6922	10	0	0	0.1	1	[*n* - *n* x *n*]
Kra30a	88900	10	0	0	7	1	[*n* - *n* x *n*]
Kra30b	91420	10	0	0	1	1	[*n* - *n* x *n* x 10]
Kra32	88700	10	0	0	0.2	1	[*n* - *n* x *n*]
Lipa20a	3683	10	0	0	0.1	1	[*n* - *n* x *n*]
Lipa20b	27076	10	0	0	0.1	1	[*n* - *n* x *n*]
Lipa30a	13178	10	0	0	1	1	[*n* - *n* x *n*]
Lipa30b	151426	10	0	0	0.1	1	[*n* - *n* x *n*]
Lipa40a	31538	10	0	0	5	1	[*n* - *n* x *n*]
Lipa40b	476581	10	0	0	0.1	1	[*n* - *n* x *n*]
Lipa50a	62093	10	0	0	14	1	[*n* - *n* x *n*]
Lipa50b	1210244	10	0	0	0.1	1	[*n* - *n* x *n*]
Lipa60a	107218	10	0	0	19	1	[*n* - *n* x *n* x 10]
Lipa60b	2520135	10	0	0	18	1	[*n* - *n* x *n* x 10]
Lipa70a	169755	10	0	0	31	1	[*n* - *n* x *n* x 10]
Lipa70b	4603200	10	0	0	3	1	[*n* - *n* x *n* x 10]
Lipa80a	253195	10	0	0	64	1	[*n* - *n* x *n* x 10]
Lipa80b	7763962	10	0	0	33	1	[*n* - *n* x *n* x 10]
Lipa90a	360630	10	0	0	102	1	[*n* - *n* x *n* x 10]
Lipa90b	12490441	10	0	0	45	1	[*n* - *n* x *n* x 10]
Average	-	10	0.0	0.0	14.3	-	-

TABLE 5: Results obtained for the problem instance *nug*.

Instance	BKS	found	APD	BPD	sec.	#proc.	aspiration
Nug12	578	10	0	0	0.1	1	[*n* - *n* x *n*]
Nug14	1014	10	0	0	1	1	[*n* - *n* x *n*]
Nug15	1150	10	0	0	0.2	1	[*n* - *n* x *n*]
Nug16a	1610	10	0	0	1	1	[*n* - *n* x *n*]
Nug16b	1240	10	0	0	0.1	1	[*n* - *n* x *n*]
Nug17	1732	10	0	0	1	1	[*n* - *n* x *n*]
Nug18	1930	10	0	0	2	1	[*n* - *n* x *n*]
Nug20	2570	10	0	0	2	1	[*n* - *n* x *n*]
Nug21	2438	10	0	0	3	1	[*n* - *n* x *n*]
Nug22	3596	10	0	0	0.1	1	[*n* - *n* x *n*]
Nug24	3488	10	0	0	3	1	[*n* - *n* x *n*]
Nug25	3744	10	0	0	4	1	[*n* - *n* x *n*]
Nug27	5234	10	0	0	2	1	[*n* - *n* x *n*]
Nug28	5166	10	0	0	3	1	[*n* - *n* x *n*]
Nug30	6124	10	0	0	2	1	[*n* - *n* x *n*]
Average	-	10	0.0	0.0	1.6	-	-

Table 10 gives the comparison of our results with *sko* problem instances. The overall deviation of the ABC algorithms is 0.0%. The performance of our algorithm can be reported to be the best one. The only problem instance that we cannot solve optimally is *sko100a* (with 0.008% deviation). In Table 11, proposed parallel PABC-QAP algorithm is compared with other parallel algorithms in literature. The results are obtained from their original studies. TLBO-RTS algorithm is executed on 50 processors, PABC-QAP works with 255 processors, QAP-IPGA uses 240 processors, COSEARCH uses 150 workstations and CPTS uses 10 processors. The results of PABC-QAP is observed to be among the best performing parallel algorithms. The number of processors used during the optimization is a crucial parameter that the largest number of processors is used by the PABC-QAP algorithm. The CPTS has a nice performance in addition to the small number of processors it uses during the experiments. CPTS consumes the longest execution times for the optimization. TLBO-RTS, CORESEARCH, CPTS, and PABC-QAP are the best performing parallel algorithms in literature for the solution of the QAP.

TABLE 6: Results obtained for the problem instances *Rou*, *Scr*, *Sko* and *Ste*.

Instance	BKS	found	APD	BPD	sec.	#proc.	aspiration
Rou12	235528	10	0	0	0.1	1	[*n* - *n* x *n*]
Rou15	354210	10	0	0	0.1	1	[*n* - *n* x *n*]
Rou20	725522	10	0	0	2	1	[*n* - *n* x *n*]
Scr12	31410	10	0	0	0.1	1	[*n* - *n* x *n*]
Scr15	51140	10	0	0	0.1	1	[*n* - *n* x *n*]
Scr20	110030	10	0	0	2	1	[*n* - *n* x *n*]
Sko42	15812	10	0	0	13	255	[*n* - *n* x *n* x 10]
Sko49	23386	10	0	0	452	255	[*n* - *n* x *n* x 10]
Sko56	34458	10	0	0	54	255	[*n* - *n* x *n* x 10]
Sko64	48498	10	0	0	144	255	[*n* - *n* x *n* x 10]
Sko72	66256	10	0	0	421	255	[*n* - *n* x *n* x 10]
Sko81	90998	10	0	0	912	255	[*n* - *n* x *n* x 10]
Sko90	115534	10	0	0	882	255	[*n* - *n* x *n* x 10]
Sko100a	152002	-	0.008	0.006	10233	255	[*n* - *n* x *n* x 10]
Sko100b	153890	10	0	0	7665	255	[*n* - *n* x *n* x 10]
Sko100c	147862	10	0	0	2898	255	[*n* - *n* x *n* x 10]
Sko100d	149576	10	0	0	2755	255	[*n* - *n* x *n* x 10]
Sko100e	149150	10	0	0	2525	255	[*n* - *n* x *n* x 10]
Sko100f	149036	10	0	0	1699	255	[*n* - *n* x *n* x 10]
Ste36a	9526	10	0	0	8	1	[*n* - *n* x *n* x 10]
Ste36b	15852	10	0	0	2	1	[*n* - *n* x *n* x 10]
Ste36c	8239110	10	0	0	5	1	[*n* - *n* x *n* x 10]
Average	-	9.54	0.0	0.0	1394.2	-	-

TABLE 7: Results obtained for the problem instances *Taia* and *Taib*.

Instance	BKS	found	APD	BPD	sec.	#proc.	aspiration
Tai12a	224416	10	0	0	0.1	1	[*n* - *n* x *n* x 10]
Tai12b	39464925	10	0	0	0.1	1	[*n* - *n* x *n* x 10]
Tai15a	388214	10	0	0	0.1	1	[*n* - *n* x *n* x 10]
Tai15b	51765268	10	0	0	0.1	1	[*n* - *n* x *n* x 10]
Tai17a	491812	10	0	0	0.1	1	[*n* - *n* x *n* x 10]
Tai20a	703482	10	0	0	2	1	[*n* - *n* x *n* x 10]
Tai20b	122455319	10	0	0	0.1	1	[*n* - *n* x *n* x 10]
Tai25a	1167256	10	0	0	4	1	[*n* - *n* x *n* x 10]
Tai25b	344355646	10	0	0	0.1	1	[*n* - *n* x *n* x 10]
Tai30a	1818146	10	0	0	10	1	[*n* - *n* x *n* x 10]
Tai30b	637117113	10	0	0	6	1	[*n* - *n* x *n* x 10]
Tai35a	2422002	10	0	0	26	1	[*n* - *n* x *n* x 10]
Tai35b	283315445	10	0	0	0.1	1	[*n* - *n* x *n* x 10]
Tai40a	3139370	10	0	0	242	255	[*n* - *n* x *n* x 10]
Tai40b	637250948	10	0	0	9	255	[*n* - *n* x *n* x 10]
Tai50a	4938796	0	0.312	0.286	1190	255	[*n* - *n* x *n* x 10]
Tai50b	458821517	10	0	0	94	255	[*n* - *n* x *n* x 10]
Tai60a	7205962	0	0.449	0.422	2039	255	[*n* - *n* x *n* x 10]
Tai60b	608215054	10	0	0	42	255	[*n* - *n* x *n* x 10]
Tai64c	1855928	10	0	0	9	255	[*n* - *n* x *n* x 10]
Tai80a	13499184	0	0.817	0.783	5012	255	[*n* - *n* x *n* x 10]
Tai80b	818415043	10	0	0	482	255	[*n* - *n* x *n* x 10]
Tai100a	21044752	0	0.614	0.578	10379	255	[*n* - *n* x *n* x 10]
Tai100b	1185996137	10	0	0	9879	255	[*n* - *n* x *n* x 10]
Tai150b	498896643	0	0.074	0.065	16665	255	[*n* - *n* x *n* x 10]
Tai256c	44759294	0	0.096	0.082	71129	255	[*n* - *n* x *n* x 10]
Average	-	7.69	0.092	0.085	4508.5	-	-

TABLE 8: Results obtained for the problem instances *Tho* and *Wil*.

Instance	BKS	found	APD	BPD	sec.	#proc.	aspiration
Tho30	149936	10	0	0	0.1	1	[n - n x n]
Tho40	240516	10	0	0	0.1	1	[n - n x n x 10]
Tho150	8133398	0	0.014	0.009	41817	255	[n - n x n x 10]
Wil50	48816	10	0	0	89	255	[n - n x n x 10]
Wil100	273038	0	0.036	0.020	621	255	[n - n x n x 10]
Average	-	6.0	0.010	0.008	8516.6	-	-

There are some studies that apply ABC to special assignment problems. None of these algorithms use Tabu search or solve the benchmark problems given in the QAPLIB. There is no parallel implementation of these algorithms and none of them is hybrid like our algorithm. Baykasoğlu et al. propose a bee algorithm for generalized assignment problems [54]. Behzadi & Sundarakani propose an ABC algorithm and applied to minimize the total cost of the developed quadratic model including weighted distance and fixed locating cost by adopting optimal assignment decision. This study does not report any solution for the problems in QAPLIB but results for some synthetic problems [55]. Sultan et al. propose an ABC algorithm to optimize the QAP (A Hospital Case Study). The main idea is to use different crossover techniques for employee and onlooker bee stages and use exchange position operator for scout bee stage [56].

When developing deterministic parallel algorithms, the first thing that comes to mind is to try to get the same results faster by accelerating the working times. However, in addition to providing acceleration, meta-heuristic algorithms are able to find (near)-optimal results in shorter times than the deterministic algorithms with their intelligent way of searching. For very large problems, this may not be always possible. Parallelization of the meta-heuristics is interesting that the main purpose becomes increasing the possibility of finding the optimal value rather than speeding-up the execution time of the algorithms only. With deterministic algorithms, computers can spend years for discovering an optimal value, whereas this may be provided by parallel meta-heuristics in a few hours with well-coordinated work of a few hundred of processors. From this perspective, the PABC-QAP algorithm is a very powerful parallel meta-heuristic algorithm that balances the exploration and exploitation of the Tabu search [57][58][59]. The scalability property of the PABC-QAP algorithm is good that with the additional processors it does not spend more time because of its well-granularity. The software architecture of the algorithm is flexible that there is no limitation on the number of processors that can be added to the parallel computation environment.

TABLE 9: Comparison of the ABC optimization algorithms with state-of-the-art algorithms on *taia* problem instances. Three best results of the algorithms are given as bold numbers.

		TLBO-RTS		JRG-DivTS		ITS	SC-Tabu		**ABC**	
Instance	BKS	APD	min.	APD	min.	APD	APD	min.	APD	min.
Tai20a	70382	**0.0**	5.2	**0.0**	0.2	**0.0**	0.246	0.001	**0.0**	0.03
Tai25a	1167256	**0.0**	8.3	**0.0**	0.2	**0.0**	0.239	0.03	**0.0**	0.06
Tai30a	1818146	**0.0**	12.1	**0.0**	1.3	**0.0**	0.154	0.07	**0.0**	0.16
Tai35a	2422002	**0.0**	16.7	**0.0**	4.4	**0.0**	0.280	0.18	**0.0**	0.043
Tai40a	3139370	**0.074**	57.5	0.222	5.2	**0.220**	0.561	0.20	**0.0**	4.03
Tai50a	4938796	**0.550**	127.6	0.725	10.2	**0.410**	0.889	0.23	**0.312**	19.83
Tai60a	7205962	**0.643**	128.5	0.718	25.7	**0.450**	0.940	0.41	**0.449**	33.98
Tai80a	13499184	0.771	244.8	**0.753**	52.7	**0.360**	**0.648**	1.0	0.827	172.98
Tai100a	21052466	1.045	385.7	**0.825**	142.1	**0.300**	0.977	1.99	**0.644**	335.6
Average		**0.342**	109.5	0.360	26.88	**0.193**	0.548	0.45	**0.248**	35.2

TABLE 10: Comparison of the ABC optimization algorithms with state-of-the-art algorithms on *sko* problem instances. Three best results of the algorithms are given as bold numbers..

		TLBO-RTS		JRG-DivTS		ACO/GA/LS		GA/SD		LagSA		**ABC**	
Instance	BKS	APD	min.	APD	min.	APD	min.	APD	min.	APD	min.	APD	min.
Sko42	15812	**0.0**	7.7	**0.0**	4.0	**0.0**	0.7	0.014	0.16	0.42	3.71	**0.0**	0.21
Sko49	23386	**0.014**	4.1	**0.008**	9.6	0.056	7.6	0.107	0.28	0.14	6.96	**0.0**	7.5
Sko56	34458	**0.003**	18.5	**0.002**	13.2	0.012	9.1	0.054	0.42	0.12	13.6	**0.0**	0.9
Sko64	48498	**0.003**	27.5	**0.0**	22.0	0.004	17.4	0.051	0.73	0.12	24.15	**0.0**	2.4
Sko72	66256	0.022	39.2	**0.006**	38.0	**0.018**	70.8	0.112	0.93	0.26	43.1	**0.0**	7.01
Sko81	90998	**0.023**	56.6	**0.016**	56.6	0.025	112.3	0.087	1.44	0.11	66.2	**0.0**	15.2
Sko90	115534	**0.029**	79.4	**0.026**	89.6	0.042	92.1	0.139	2.31	0.16	116.9	**0.0**	14.7
Sko100a	152002	0.040	109.5	0.027	129.2	**0.021**	171.0	0.114	3.42	0.13	170.8	**0.008**	170.5
Sko100b	153890	0.027	109.4	**0.008**	106.6	**0.012**	192.4	0.096	3.47	-	-	**0.0**	127.7
Sko100c	147862	0.016	109.6	*0.006*	126.7	**0.005**	220.6	0.075	3.22	-	-	**0.0**	48.3
Sko100d	149576	0.035	109.3	**0.027**	123.5	**0.029**	209.2	0.137	3.45	-	-	*0.0*	45.91
Sko100e	149150	0.020	109.7	**0.009**	108.8	**0.002**	208.1	0.071	3.31	-	-	**0.0**	42.08
Sko100f	149036	**0.024**	109.5	**0.023**	110.3	0.034	210.9	0.148	3.55	-	-	**0.0**	28.31
Average		**0.019**	69.06	**0.012**	72.1	0.020	117.1	0.093	2.1	0.183	55.7	**0.0**	39.3

TABLE 11: Comparison with state-of-the-art parallel algorithms.

Instance	BKS	TLBO-RTS		TB-MTS	COSEARCH	CPTS		QAP-IPGA		**PABC-QAP**	
		APD	min.	APD	APD	APD	min.	APD	min.	APD	min.
Bur26d	3821225	**0.0**	9.2	**0.0**	**0.0**	**0.0**	0.4	**0.0**	14.7	**0.0**	0.2
Nug30	6124	**0.0**	14.8	**0.0**	**0.0**	**0.0**	1.7	**0.0**	28.6	**0.0**	0.03
Tai35b	283315445	**0.0**	22.4	**0.0**	**0.0**	**0.0**	2.4	0.820	33.3	**0.0**	0.01
Ste36c	8239110	**0.0**	24.1	**0.0**	**0.0**	**0.0**	2.5	**0.0**	34.7	**0.0**	0.08
Lipa50a	62093	**0.0**	68.8	**0.0**	**0.0**	**0.0**	11.2	0.840	35.5	**0.0**	0.23
Sko64	48498	**0.0**	119.3	0.004	0.003	**0.0**	42.9	0.350	44.1	**0.0**	2.4
Tai64c	1855928	**0.0**	117.7	**0.0**	**0.0**	**0.0**	20.6	**0.0**	44.2	**0.0**	0.08
Tai100a	21052466	**0.596**	483.3	0.814	**0.544**	**0.589**	261.2	0.893	52.1	0.644	172.9
Tai100b	1185996137	**0.0**	508.2	0.397	0.135	**0.001**	241.0	**0.0**	52.3	**0.0**	164.7
Sko100a	152002	**0.003**	594.3	0.073	0.054	**0.0**	304.8	0.290	52.4	**0.008**	170.5
Wil100	273038	**0.0**	482.6	0.035	0.009	**0.0**	316.6	**0.0**	52.7	0.036	10.35
Tai150b	498896643	**0.015**	428.5	1.128	0.439	**0.076**	1,549.4	0.790	57.3	**0.074**	277.7
Tho150	8133398	**0.030**	556.6	**0.012**	0.065	**0.013**	1,991.7	0.94	57.3	**0.014**	696.9
Tai256c	44759294	-	-	**0.270**	-	0.136	7377.8	-	-	**0.096**	1185.5
Average		**0.049**	263.8	0.195	0.098	**0.058**	866.1	0.385	43.1	**0.059**	191.5

5.5 Theoretical analysis of the proposed algorithms

ABC algorithms proposed in our study are hybrid meta-heuristic algorithms. These algorithms are non-deterministic and they make use of probabilistic intelligent methods to search and obtain the best/optimal solutions. Theoretically, there is no guarantee that the meta-heuristic algorithms will always return the optimal solutions. However, the meta-heuristic algorithms are one of the best tools to deal with NP-Hard problems in literature. With respect to the No Free Lunch Theorem (NFL), a particular meta-heuristic may have promising results for a set of problems but the same algorithm may have poor performance on a different set of problems [59]. Therefore, we know that developing a good algorithm for a specific problem is not going to be a generalized solution for all classes of problems. In this sense, we propose hybrid algorithms that can perform better and provide higher probabilities for obtaining the optimal results.

The main drawback of the meta-heuristic algorithms is that they execute numerous evaluations for each new solution that is generated during the optimization process. Increasing the speed of these calculations by using dynamic programming and parallel computation techniques ensure that the proposed algorithm will explore/exploit more of the search space and thus have a higher chance of finding better results in the same amount of time.

Termination criterion and the parameter settings of the proposed algorithms are crucial issues to be discussed in our study. Of course, running the algorithms with higher number of iterations provides a higher chance for finding better solutions. However, being stuck around local optima and the effect of well-tuned parameter settings are the most important factors of meta-heuristic algorithms when increasing the number of iterations. In order to provide a mechanism to avoid being stuck at the local optima, we develop a parallel version of the algorithm. By this way, processors are working on diversified parts of the problem. This technique reduces the possibility of our algorithm getting stuck at local optima. A substantial improvement is observed in our solutions. Tuning the parameters can be considered as another important problem of meta-heuristic optimization algorithms. Simple parameter tuning techniques can provide significant improvements for these algorithms.

6 Conclusions and future work

In this study, we present novel ABC optimization algorithms for the optimization of the QAP. ABC meta-heuristic is observed to perform well for the solution of the QAP. Large problem instances of the QAP are still challenging. Therefore, we need to develop a parallel (MPI) version of the ABC algorithm. A distributed memory parallel version of the ABC is proposed. A state-of-the-art trajectory optimization method, Tabu search, is adapted for the exploration and exploitation of the ABC algorithms. Tabu list and aspiration values of this local search method are tuned during our experiments for better optimization results. Majority of the benchmark problem instances are solved optimally with the new ABC algorithms. By providing a good balance between exploration and exploitation phases, better results are reported.

There are still good opportunities to obtain better results with higher number of processors, better tuned parameters or new introduced meta-heuristic techniques. As future work, we plan to apply ABC for the other well-known combinatorial problem instances. There are many meta-heuristics introduced recently. Applying parallel computation and tuning their parameters in run-time can be interesting and has potential to improve the solution of existing NP-Hard combinatorial problems. Black box optimization function is an effective tool to evaluate the performance of the new algorithms. It consists of a wide range of benchmark problems. We plan to use this tool to evaluate the performance of our new hybrid algorithm on different problem domains.

References

[1] T. C. Koopmans, M. Beckmann, Assignment problems and the location of economic activities, Econometrica: journal of the Econometric Society (1957) 53–76.

[2] R. E. Burkard, Quadratic assignment problems, Handbook of combinatorial optimization (2013) 2741–2814.

[3] E. Cela, The quadratic assignment problem: theory and algorithms, Vol. 1, Springer Science & Business Media, 2013.

[4] F. R. R. E. Burkard, S. Karisch, Qaplib-a quadratic assignment problem library, European Journal of Operational Research 55 (1) (1991) 115–119.

[5] L. Steinberg, The backboard wiring problem: A placement algorithm, Siam Review 3 (1) (1961) 37–50.

[6] D. F. Rossin, M. C. Springer, B. D. Klein, New complexity measures for the facility layout problem: an empirical study using traditional and neural network analysis, Computers & Industrial Engineering 36 (3) (1999) 585–602.

[7] G. F. Pfister, In search of clusters, Vol. 2, Prentice Hall PTR Englewood Cliffs, 1998.

[8] M. Lim, Y. Yuan, S. Omatu, Efficient genetic algorithms using simple genes exchange local search policy for the quadratic assignment problem, Computational Optimization and Applications 15 (3) (2000) 249–268.

[9] R. K. Adl, S. M. T. R. Rankoohi, A new ant colony optimization based algorithm for data allocation problem in distributed databases, Knowledge and Information Systems 20 (3) (2009) 349–373.

[10] D. W. Pentico, Assignment problems: A golden anniversary survey, European Journal of Operational Research 176 (2) (2007) 774–793.

[11] E. L. Lawler, The quadratic assignment problem, Management science 9 (4) (1963) 586–599.

[12] D. Karaboga, An idea based on honey bee swarm for numerical optimization, Tech. rep., Technical report-tr06, Erciyes university, engineering faculty, computer engineering department (2005).

[13] D. Karaboga, B. Gorkemli, A combinatorial artificial bee colony algorithm for traveling salesman problem, Innovations in intelligent systems and applications (2011) 50–53.

[14] B. B. LD. Karaboga, Artificial bee colony (abc) optimization algorithm for solving constrained optimization problems, in: International fuzzy systems association world congress, Springer, 2007, pp. 789–798.

[15] T. Kucukyilmaz, H. E. Kiziloz, Cooperative parallel grouping genetic algorithm for the one-dimensional bin packing problem, Computers & Industrial Engineering 125 (2018) 157–170.

[16] Y. Aksan, T. Dokeroglu, A. Cosar, A stagnation-aware cooperative parallel breakout local search algorithm for the quadratic assignment problem, Computers & Industrial Engineering 103 (2017) 105–115.

[17] T. Dokeroglu, A. Cosar, A novel multistart hyper-heuristic algorithm on the grid for the quadratic assignment problem, Engineering Applications of Artificial Intelligence 52 (2016) 10–25.

[18] É. Taillard, Robust taboo search for the quadratic assignment problem, Parallel computing 17 (4-5) (1991) 443–455.
[19] G. A. E.-N. A. Said, A. M. Mahmoud, E.-S. M. El-Horbaty, A comparative study of meta-heuristic algorithms for solving quadratic assignment problem, arXiv preprint arXiv:1407.4863.
[20] D. Karaboga, B. Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm, Journal of global optimization 39 (3) (2007) 459–471.
[21] D. Karaboga, B. Basturk, On the performance of artificial bee colony (abc) algorithm, Applied soft computing 8 (1) (2008) 687–697.
[22] N. K. D. Karaboga, C. Ozturk, B. Gorkemli, Artificial bee colony programming for symbolic regression, Information Sciences 209 (2012) 1–15.
[23] M.-H. Tayarani-N, A. Prugel-Bennett, On the landscape of combinatorial optimization problems, IEEE Transactions on Evolutionary Computation 18 (3) (2014) 420–434.
[24] N. S. M. Subotic, M. Tuba, Different approaches in parallelization of the artificial bee colony algorithm, International Journal of mathematical models and methods in applied sciences 5 (4) (2011) 755–762.
[25] C. M. V. Benítez, H. S. Lopes, Parallel artificial bee colony algorithm approaches for protein structure prediction using the 3dhp-sc model, in: Intelligent Distributed Computing IV, Springer, 2010, pp. 255–264.
[26] H. Narasimhan, Parallel artificial bee colony (pabc) algorithm, in: Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on, IEEE, 2009, pp. 306–311.
[27] R. S. Parpinelli, C. M. V. Benitez, H. S. Lopes, Parallel approaches for the artificial bee colony algorithm, in: Handbook of Swarm Intelligence, Springer, 2011, pp. 329–345.
[28] E. M. Loiola, N. M. M. de Abreu, P. O. Boaventura-Netto, P. Hahn, T. Querido, A survey for the quadratic assignment problem, European journal of operational research 176 (2) (2007) 657–690.
[29] C. F. E.G. Talbi, O. Roux, D. Robillard, Tabu search a tutorial, Future Generation Computer Systems 17 (4) (2001) 441–449.
[30] Y. Zhou, J.-K. Hao, B. Duval, When data mining meets optimization: A case study on the quadratic assignment problem, arXiv preprint arXiv:1708.05214.
[31] W. Chmiel, J. Kwiecień, Quantum-inspired evolutionary approach for the quadratic assignment problem, Entropy 20 (10) (2018) 781.
[32] K. Mihić, K. Ryan, A. Wood, Randomized decomposition solver with the quadratic assignment problem as a case study, INFORMS Journal on Computing 30 (2) (2018) 295–308.
[33] T. Dokeroglu, Hybrid teaching-learning-based optimization algorithms for the quadratic assignment problem, Computers & Industrial Engineering 85 (2015) 86–101.
[34] M. Abdel-Baset, M. Gunsekaran, D. El-Shahat, S. Mirjalili, Integrating the whale algorithm with tabu search for quadratic assignment problem: A new approach for locating hospital departments, Applied Soft Computing.
[35] E. Çela, V. Deineko, G. J. Woeginger, New special cases of the quadratic assignment problem with diagonally structured coefficient matrices, European journal of operational research 267 (3) (2018) 818–834.
[36] S. Bougleux, L. Brun, V. Carletti, P. Foggia, B. Gaüzère, M. Vento, Graph edit distance as a quadratic assignment problem, Pattern Recognition Letters 87 (2017) 38–46.
[37] J. Nalepa, M. Blocho, Co-operation in the parallel memetic algorithm, International Journal of Parallel Programming 43 (5) (2015) 812–839.
[38] U. Tosun, On the performance of parallel hybrid algorithms for the solution of the quadratic assignment problem, Engineering Applications of Artificial Intelligence 39 (2015) 267–278.
[39] J. H. U. Benlic, Select breakout local search for the quadratic assignment problem, Applied Mathematics and Computation 219 (9) (2013) 4800–4815.
[40] J. H. U. Benlic, Memetic search for the quadratic assignment problem, Expert Systems with Applications 42 (1) (2015) 584–595.
[41] M. N.Fescioglu-Unver, Self controlling tabu search algorithm for the quadratic assignment problem, Computers & Industrial Engineering 60 (0) (2011) 310–319.
[42] A. F. A. E. Duman, M. Uysal, Migrating birds optimization: A new metaheuristic approach and its performance on quadratic assignment problem, Information Sciences 217 (2012) 65–77.
[43] F. Glover, Future paths for integer programming and links to artificial intelligence, Computers & Operations Research 13 (3) (1986) 533–549.
[44] F. Glover, Tabu search: A tutorial, Interfaces 20 (4) (1990) 74–94.
[45] J. P. Kelly, M. Laguna, F. Glover, A study of diversification strategies for the quadratic assignment problem, Computers and Operations Research 21 (8) (1994) 885–894.
[46] M. Laguna, R. Marti, V. Campos, Intensification and diversification with elite tabu search solutions for the linear ordering problem, Computers & Operations Research 26 (12) (1999) 1217–1230.
[47] F. Glover, Tabu searchpart ii, ORSA Journal on computing 2 (1) (1990) 4–32.
[48] R. E. Burkard, S. E. Karisch, F. Rendl, Qaplib a quadratic assignment problem library, Journal of Global optimization 10 (4) (1997) 391–403.
[49] T. James, C. Rego, F. Glover, Multistart tabu search and diversification strategies for the quadratic assignment problem, IEEE TRANSACTIONS ON SYSTEMS, Man, And Cybernetics-part a: systems and humans 39 (3) (2009) 579–596.
[50] A. Misevicius, An implementation of the iterated tabu search algorithm for the quadratic assignment problem, OR spectrum 34 (3) (2012) 665–690.
[51] Y. Xia, An efficient continuation method for quadratic assignment problems, Computers & Operations Research 37 (6) (2010) 1027–1032.
[52] Z. Drezner, The extended concentric tabu for the quadratic assignment problem, European Journal of Operational Research 160 (2) (2005) 416–422.

[53] L.-Y. Tseng, S.-C. Liang, A hybrid metaheuristic for the quadratic assignment problem, Computational Optimization and Applications 34 (1) (2006) 85–113.
[54] A. Baykasoğlu, L. Özbakır, P. Tapkan, Artificial bee colony algorithm and its application to generalized assignment problem, in: Swarm intelligence, focus on ant and particle swarm optimization, InTech, 2007.
[55] G. Behzadi, B. Sundarakani, Practical abc intelligence solution for quadratic assignment problems.
[56] J. A. Sultan, D. A. Matrood, Z. M. Khaleel, Artificial bee colony for quadratic assignment problem: A hospital case study, Journal of University of Human Development/Vol 2 (3).
[57] R. C. E. Y. Shi, Parameter selection in particle swarm optimization, in: International conference on evolutionary programming, Springer, 1998, pp. 591–600.
[58] C. F. L. F. Lobo, Z. Michalewicz, Parameter setting in evolutionary algorithms, Springer Science & Business Media 54.
[59] W. G. M. D. H. Wolpert, No free lunch theorems for optimization, IEEE transactions on evolutionary computation 1 (1) (1997) 67–82.