Computers & Industrial Engineering 103 (2017) 105–115

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

A stagnation-aware cooperative parallel breakout local search algorithm for the quadratic assignment problem

Yagmur Aksan [a], Tansel Dokeroglu [b,*], Ahmet Cosar [a]

[a] Computer Engineering Department of Middle East Technical University, Inonu Bulvari, 06800 Ankara, Turkey
[b] Computer Engineering Department, Turkish Aeronautical Association University, Ankara, Turkey

ARTICLE INFO

Article history:
Received 27 August 2016
Received in revised form 15 November 2016
Accepted 20 November 2016
Available online 23 November 2016

Keywords:
Quadratic assignment problem
Breakout local search
OpenMP
Optimization
Levenshtein distance

ABSTRACT

The Quadratic Assignment Problem (QAP) is one of the most challenging NP-Hard combinatorial optimization problems. Circuit-layout design, transportation/traffic engineering, and assigning gates to airplanes are some of the interesting applications of the QAP. In this study, we introduce an enhanced version of a recent local search heuristic, Breakout Local Search Algorithm (BLS), by using the Levenshtein Distance metric for checking the similarity of the new starting points to previously explored QAP permutations. The similarity-checking process prevents the local search algorithm, BLS, from getting stuck in already-explored areas. In addition, the proposed BLS Algorithm (BLS-OpenMP) incorporates multi-threaded computation using OpenMP. The stagnation-aware search for the optimal solutions of the QAP is executed concurrently on several cores with diversified trajectories while considering their similarity to already-discovered local optima. The exploration of the search space is improved by selecting the starting points intelligently and speeding up the fitness evaluations linearly with number of processors/threads. BLS-OpenMP has been tested on the hardest 59 problem instances of the QAPLIB, and it obtained 57 of the best known results. The overall deviation of the achieved solutions from the best known results is 0.019% on average, which is a significant improvement compared with state-of-the-art algorithms.

1. Introduction

The Quadratic Assignment Problem (QAP) is a well-known and challenging NP-Hard combinatorial optimization problem, that was introduced by Koopmans and Beckmann (1957) to model the positions of indivisible economic activities. In spite of the extensive academic efforts from its initial formulation to date, it remains one of the hardest combinatorial optimization problems. The purpose of the QAP is to place a number of different facilities at different locations in such a way that minimizes the total cost of the installation and operation of the given facilities as much as possible. That is, it seeks to place N facilities at N fixed locations in the most cost-effective way. The computational complexity of the QAP can be formulated using three $N \times N$ matrices, named A, B, and C.

$A = (a_{ik})$

where a_{ik} stands for the cost of material flow (per unit-distance) from facility i to facility k.

$B = (b_{jl})$

where b_{jl} stands for the distance value from location j to location l.

$C = (c_{ij})$

where c_{ij} stands for the cost value of assigning facility i to the location j.

The Koopmans-Beckmann formulation of the QAP can be presented as:

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

where S_n stands for the permutation of integers $1, 2, \ldots, n$ chosen from the set of all possible placements for n. The range of the indexes i, j, k, l is $1, \ldots, n$ where n is the problem size. Each individual product $a_{ik}b_{\phi(i)\phi(k)}$ indicates the cost of the transportation from facility i at location $\phi(i)$ to facility k at location $\phi(k)$. Moreover, $c_{i\phi(i)}$ represents the total cost for locating facility i at location i and the sum of the transportation costs to all other facilities k assigned at locations $\phi(1), \phi(2), \ldots, \phi(n)$. As a consequence, the main purpose is to seek an optimal solution that minimizes the cost function.

* Corresponding author.
E-mail addresses: yagmuraksan@gmail.com (Y. Aksan), tansel@ceng.metu.edu.tr, tdokeroglu@ceng.metu.edu.tr (T. Dokeroglu), cosar@ceng.metu.edu.tr (A. Cosar).

http://dx.doi.org/10.1016/j.cie.2016.11.023
0360-8352/

The QAP belongs to the NP-Hard class of problems, which means that there is no known algorithm proven to solve the problem in polynomial time. QAP instances with sizes larger than 35 cannot be solved in reasonable execution times. Scientists continue their efforts to solve this problem and have recently proposed several well-performing heuristic algorithms such as Simulated Annealing (Connolly, 1990; Wilhelm & Ward, 1987), Tabu Search (James, Rego, & Glover, 2009; Skorin-Kapov, 1990; Taillard, 1991), Ant Colony Optimisation (Gambardella, Taillard, & Dorigo, 1999; Stützle & Dorigo, 1999), Genetic Algorithms (Drezner, 2003), Memetic Algorithms (Carrizo, Tinetti, & Moscato, 1992), Scatter Search (Cung, Mautor, Michelon, & Tavares, 1997), and Iterated Local Search (Stützle, 2006).

The Breakout Local Search Algorithm (BLS) is a local search heuristic algorithm that was recently proposed to solve the QAP. The BLS finds local optima by using the best improvement move method and escapes from one local optimum to another solution via its adaptive perturbation mechanism that uses tabu search, recency-based, and random perturbation methods (Benlic & Hao, 2013a). The BLS has been observed to be a very effective algorithm on a set of QAPLIB benchmark instances.

In this study, we propose an improved version of the BLS using the OpenMP parallel-computation paradigm and Levenshtein Distance (LD) similarity metric (BLS-OpenMP). LD was originally defined as a distance between two strings. Briefly, it is the minimum number of editing operations (deletions, insertions, or substitutions) required to convert one string into another. In our opinion, this similarity metric can be used to detect the previously explored search spaces of the QAP instance by identifying states that are very similar states to the local optimum, and repetitive searches around the same local optimum can be avoided significantly. BLS-OpenMP saves the starting points and the local optima of the search process in which the optimization process gets stuck in. During the restarting (multistarting) process of the exploration, BLS-OpenMP produces new solutions that are strongly diversified from the previously explored areas. It is obvious that a multithreaded computation approach using OpenMP further improves the quality of the solutions by examining many more starting points in the search space compared to its single-threaded version. Therefore, we make use of OpenMP during the calculations of the fitness values. The cooperative nature of BLS-OpenMP provides a well-divided search space environment for the working threads. BLS-OpenMP is executed on the hardest 59 problem instances of the QAP library benchmark and the results show that it obtains 57 of the best known results. The overall deviation of the solutions is obtained as 0.019% on average, which is a significant improvement compared with state-of-the-art algorithms.

The outline of the study is as follows. In Section 2, we explain related works about state-of-the-art QAP algorithms. In Section 3, we give information about the components of the proposed algorithm (OpenMP, Levenshtein Distance metric, and the BLS). Section 4 presents information about the experimental setup, the QAPLIB benchmark instances and observed performances of BLS-OpenMP. Finally, we give our concluding remarks and future work.

2. Related work

In this section, we examine the exact and heuristic approaches/algorithms proposed for the optimization of the QAP and give a literature review of the state-of-the-art algorithms. The QAP is specified as the problem of finding minimum cost for assigning facilities to locations where the cost is calculated as the total sum of all distance-flow products (Loiola, de Abreu, Boaventura-Netto, Hahn, & Querido, 2007). The QAP was introduced by Koopmans and Beckmann (1957) as a mathematical model associated with economic activities. In this research, there is a discussion about problems in the assignment of independent resources interpreted as plants. Since its introduction, it has been used in various practical applications to solve real-life problems in different fields.

The QAP is one of the most challenging combinatorial optimization problems. The exact solution of a QAP with a size larger than 35 cannot be determined in an acceptable time. Because of its practical and theoretical significance and its complexity, the QAP has been an outstanding research area for scientists since its first formulation (Loiola et al., 2007). Sahni and Gonzales (1976) proved that the QAP is an NP-hard combinatorial optimization problem. Many NP-hard combinatorial optimization problems can be modelled using the QAP formulation. The travelling salesman, scheduling, transportation systems, typewriter keyboard design, graph partitioning, backboard wiring, signal processing, bin packing, maximum clique, statistical data analysis, data allocation, layout design, and minimum-inbreeding seed orchard layout are the most important of these problems (Burkard, Karisch, & Rendl, 1991; Dokeroglu, 2015; Dokeroglu & Cosar, 2016; Lstiburek, Stejskal, Misevičius, Korecky, & El-Kassaby, 2015; Steinberg, 1961).

Steinberg (1961) focused on the QAP to decrease the connections between components. Geoffrion and Graves (1976) applied the QAP to scheduling problems. Dickey and Hopkins (1972) used it to allocate construction in a university campus. Duman and Ilhan (2007) applied it to the allocation of electronic components on a printed circuit board or on a microchip. Ben-David and Malah (2005) studied the index assignment problem on which the effect of channel errors on the coding-system performance depends.

There exist several heuristics that use the tabu search algorithm to solve the QAP. Drezner proposed an algorithm to improve the concentric tabu search for the QAP (Drezner, 2005). The Iterated Tabu Search (ITS) algorithm was proposed by Misevičius (2012) to practically solve the medium- and large-scale QAP. ITS is a combination of an intensification mechanism to search for good solutions in the neighbourhood and a diversification mechanism in an attempt to escape from local optima and continue searching the area. Moreover, it raises a novel formula for quick computation of the objective function so that the results can be obtained more quickly. The results of the tests for the ITS algorithm show promising efficiency particularly for random QAP instances. Fescioglu-Unver and Kokar proposed a self controlling TS algorithm for the QAP (Fescioglu-Unver & Kokar, 2011) and came up with novel mechanisms for the TS algorithm. These mechanisms aim to adjust the algorithm parameters for intensification and diversification. Thanks to the self-controlling mechanisms of the algorithm, good results can be obtained on different QAP instances. Acan and Unveren proposed an algorithm called GDA, which is a combination of great deluge and tabu search algorithms, for the QAP (Acan & Unveren, 2015). GDA benefits from the experience accumulated in memory. When a better solution is obtained in a specific iteration, GDA is applied to the level-based acceptance criterion and the first stage of external memory is updated. Furthermore, for a determined number of times, the second stage is updated after the first stage. The elements in the second stage are chosen to be dissimilar according to their similarity degree. Therefore, it stores promising elements from different regions of the exploration area. According to the promising experimental results, it is apparent that GDA can be an alternative way to solve the QAP.

ANGEL, which combines the Ant Colony Optimization (ACO), the Genetic Algorithm (GA) and a Local Search method, was proposed as a hybrid metaheuristic to solve the QAP by Tseng and Liang (2005). It consists of ACO and GA phases. The BLS executes a local search algorithm that finds local optima with the best improvement move method and escapes from these local optima to other local optima with its adaptive perturbation mechanism, which uses tabu search, recency-based and random perturbation

methods (Benlic & Hao, 2013a). The BLS has even been applied to various complex combinatorial problems, including maximum clique (both weighted and unweighted cases) (Benlic & Hao, 2013b), maximum cut (Benlic & Hao, 2013c), and minimum sum colouring (Benlic & Hao, 2012). Pretty good success is achieved in these experiments on the well known problem instances of the QAPLIB benchmark. The results show that good performance is achieved on this problem set, for which the current best known solutions are achieved in acceptable computation time. Benlic and Hao (2015) proposed a population-based memetic algorithm (BMA), which is incorporated with the BLS, a powerful local optimization algorithm. BMA consists of a uniform crossover, a fitness-based pool updating methodology and an adjustable mutation strategy.

Because it offers speed-up opportunity that outperforms current multi-core processors, Tsutsui and Fujimoto (2009) applied GPU computation with compute unified device architecture (CUDA) to solve the QAP. In combination with the tabu search algorithm, they propose ant colony optimization for the QAP. In their study, tabu search moves create two groups according to their computing costs. To compute of these groups' moving costs, threads of CUDA are applied. To minimize the disabling time, Move-Cost Adjusted Thread Assignment (MATA) is used. Moreover, Cunning Ant System is used for the ACO algorithm. The result of this study is that a GPU computation with MATA achieves sufficient acceleration to outperform with the speed of computation in CPU. Dokeroglu and Cosar (2016) proposed a novel Parallel Multi-start Hyper-heuristic algorithm (MSH-QAP) for the solution of the QAP. With the help of hyper-heuristics, the appropriate heuristic for the given instance is dynamically determined along with the parameters of the heuristic. In this research, MSH-QAP benefits from the state-of-the-art (meta)-heuristics which have been declared the best performing algorithms for solving the QAP. These algorithms consist of Simulated Annealing (SA), Robust Tabu Search (RTS), and Ant Colony Optimisation-based Fast Ant System (FANT). Overall, 119 problems from the benchmark set are solved optimally in terms of reaching best known results and solutions to the remaining 15 problems have an average deviation of 0.07%.

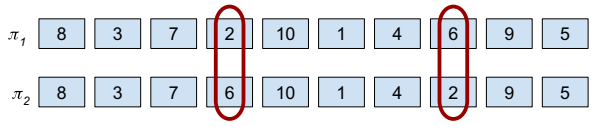
Harris, Berretta, Inostroza-Ponta, and Moscato (2015) proposed a Memetic Algorithm (MA) that uses a ternary tree structure for its population and the TS algorithm, which runs simultaneously, for its local search mechanism. In fact, MA is an improvement of the algorithm mentioned in the study of Meneses and Inostroza-Ponta (2011). The algorithm shows great performance in the sense of not only lower computation time but also higher quality solutions. Czapiński (2013) proposed a Parallel Multistart Tabu Search (PMTS) algorithm, which is a remarkable rapid heuristic for the QAP. Furthermore, it is implemented on a highly powerful GPU hardware intended for high-performance computing with the CUDA platform. Therefore, PMTS is shown to perform competitively with a single-core or a parallel CPU implementation on a high-end six-core CPU. These noteworthy results from experiments are due to the neighbourhood evaluation in parallel architecture, effective memory design, sensible access patterns and almost entirely GPU Tabu Search.

A parallel hybrid algorithm (PHA) with three phases was proposed by Tosun (2015). PHA initially benefits from a genetic algorithm to obtain a high-quality initial seed on which a diversification mechanism will be run. Finally, this modified solution is used for a robust tabu search to find a near-optimal result. Moreover, PHA uses parallel computing; therefore, it obtains considerable speed-up.

To the best of our knowledge, there is no existing heuristic approach that takes advantage of threads and cooperative similarity checking of previously explored areas using the Levenshtein Distance metric to solve the QAP. Our proposed heuristic search algorithm, BLS-OpenMP, uses a log-based similarity checking approach for previously searched permutations of the QAP problem instances, which is one of the first implementations in this area. Recent approaches have attempted to get rid of stagnation only by randomly generating new starting points on mostly a single core and most frequently with similar permutations.

3. Proposed algorithm (BLS-OpenMP)

In this section, BLS-OpenMP algorithm is introduced. BLS-OpenMP makes use of parallel computation by using OpenMP and shows a promising speed-up compared to its single-CPU version. Speed-up not only increases the number of fitness calculations but also makes the algorithm search the environment more



π_2 \ π_1	#	8	3	7	2	10	1	4	6	9	5
#	0	1	2	3	4	5	6	7	8	9	10
8	1	0	1	2	3	4	5	6	7	8	9
3	2	1	0	1	2	3	4	5	6	7	8
7	3	2	1	0	1	2	3	4	5	6	7
6	4	3	2	1	1	2	3	4	4	5	6
10	5	4	3	2	2	1	2	3	4	5	6
1	6	5	4	3	3	2	1	2	3	4	5
4	7	6	5	4	4	3	2	1	2	3	4
2	8	7	6	5	4	4	3	2	2	3	4
9	9	8	7	6	5	5	4	3	3	2	3
5	10	9	8	7	6	6	5	4	4	3	2

Fig. 1. A new candidate starting point that violates the minimum similarity measure.

effectively. BLS-OpenMP can obtain competitive results on a wide range of well-known problem instances of the QAPLIB by reaching the best known results in reasonable computation times. We present the details of BLS-OpenMP in Algorithm 1. It starts by generating intelligent initial candidate solutions. To prevent starting with the same/similar candidate solutions, explored areas (permutations) are saved in the memory and considered during the multi-start process. Diversified new solutions are produced using the Levenshtein Distance (LD) similarity metric. The algorithm proceeds with the descent procedure, which aims to locate the local optimum. BLS-OpenMP explores its neighbours in a steepest descent process. As soon as the local optimum is reached, the number of perturbation moves and jump magnitude are determined. Eventually, it applies the perturbation stage, which consists of tabu search, recency-based and random perturbation methods to escape from the local optimum with a new starting point.

Algorithm 1. BLS-OpenMP Algorithm (Benlic & Hao, 2013a).

Require:
Distance and flow matrices: d and f of size $n \times n$.
The number of thread: $count_{thread}$
The size of loop for the BLS processes: $size_{bls}$
The vector keeping the previous solutions: $previousSol$
Ensure:
A permutation π over a set of facility locations.
$\pi \leftarrow$ random permutation of $\{1, ..., n\}$
$\pi_{initDF} \leftarrow GenerateSolutionsByDFMatrices()$
/* π_{initDF} randomly generated initial solution */
/* Generate initial solutions for remaining threads*/
$\pi_{initRs} \leftarrow GenerateSolutions$
$\pi_{inits} \leftarrow \pi_{initDF} + \pi_{initRs}$
/* π_{inits} is the list of all initial solutions */
while $i < size_{bls}$ **do**
  /* Generate different initial solutions */
  $\pi_{initRs} \leftarrow GenerateSolutions(previousSol)$
  $\pi_{inits} \leftarrow \pi_{initRs}$
  Write initial solutions to $previousSol$
  Send π_{inits} to the threads
  $c \leftarrow C(\pi)$ /* c is objective value of the curr. sol. */
  Compute the initial n x n matrix δ of move gains
  $\pi_{best} \leftarrow \pi$ /* π_{best} the best solution so far */
  $c_{best} \leftarrow c$ /* c_{best} the best objective value so far */
  $c_p \leftarrow c$ /* c_p the best value of the last descent */
  $w \leftarrow 0$ /* w non-improving local optima counter*/
  $L \leftarrow L_0$ /* set perturb. moves L to default value L_0*/
  while *not terminated by all of the threads* **do**
    /* Identify the best improving move */
    /*apply it to the neighbours */
    $\pi \leftarrow \text{SteepestDes}(\pi, \pi_{best}, c, c_{best}, c_p, H, Iter, \delta, w)$
    /* Determine L to apply to π */
    $L \leftarrow \text{DetermineJumpMagnitude}(c, c_p, w)$
    /* Perturb π with L perturb. moves */
    $c_p \leftarrow c$ /* Update value of local optimum */
    $\pi \leftarrow \text{Perturbation}(\pi, L, H, Iter, \delta, w)$
  Obtain solutions from threads
  Write solutions/local optima to $previousSol$
Report the overall best solution

Generating candidate solutions: BLS-OpenMP creates initial candidates based on the previously explored solutions. Candidates do not attempt to explore the same/similar problem landscapes. Because it is not possible to exhaustively explore all permutations π of $\{1, \ldots, n\}$ solutions, this technique has a significant effect on the optimization process. Candidate solutions are produced in two phases: smart beginning, in which the distance and flow matrices are considered, followed by the intelligent consideration of previously explored local optima.

Initialization with respect to the already explored areas/local optima: BLS-OpenMP needs a number of candidates equal to the number of threads in the OpenMP environment for each multi-start. To guarantee that all candidates are different from one another, we impose the constraint that randomly generated solutions cannot be the same as or/similar to the previously examined ones, based on a defined similarity ratio. According to the previously determined minimum percentage of difference between candidate solutions, it is determined whether the generated new permutations are acceptable or not. Unless these new permutations are suitable, BLS-OpenMP generates new candidate solutions with enough diversity. The similarity checking mechanism is derived from a Minimum Edit-Distance algorithm called LD (Ristad & Yianilos, 1998). LD is a distance similarity metric for comparing two strings. It is defined as the number of editing operations (deletion, insertion, or substitution) needed to convert one string into the other. LD has been successfully applied to several real-life applications. Spell correction, speech recognition, computational biology, DNA analysis, machine translation, information extraction, and plagiarism detection are some applications of LD. The LD between two strings, a and b is calculated according to the following formula:

$$lev_{a,b}(i,j) = \begin{cases} max(i,j) & \text{if } min(i,j) = 0, \\ min \begin{cases} lev_{a,b}(i-1,j) + 1 \\ lev_{a,b}(i,j-1) + 1 \\ lev_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

where $1_{(a_i \neq b_j)}$ is equal to 0 when $a_i = b_j$ and equal to 1otherwise.

As shown in Fig. 1, there exist two candidate solutions, π_1 and π_2. We assume that π_1 is used in the first permutation of BLS-OpenMP and that we need to generate another candidate to execute our algorithm on this solution, which is different from the first one at least by 30%. Therefore, we use LD to calculate the minimum difference similarity. We obtain the distance value $lev(\pi_1, \pi_2) = 2$. This means that two substitutions are sufficient to transform π_1 into π_2. Hence, the second candidate, π_2, violates the similarity ratio constraint because it has only 20% difference (2/10). It is discarded by our algorithm and another candidate permutation needs to be generated.

In BLS-OpenMP, all candidate solutions have the same length, so two solutions can be transformed into one another using only substitution operations. The cost of each substitution is evaluated as 1. Searching for a sequence of edits from the first permutation value to the second permutation value requires a large amount of memory and navigating from these sequences of edits requires much effort. This calculation is performed by comparing all values in the first solution with the corresponding values at the same indices of the second one. We can calculate the minimum difference size in a cost-effective way. The candidate solution π is randomly generated for each thread to execute the remaining steps of the algorithm. The size of the generated candidates must be the same as the number of threads specified at the beginning of the algorithm. Later, the algorithm performs the similarity checking on these candidates. The

similarity checking mechanism determines whether solution π violates the minimum difference constraint. The minimum percentage of difference is determined at the beginning of the algorithm. To provide this control, π is compared with the solutions in the vector *previousSol* in which BLS-OpenMP keeps track of the previously experimented solutions. All values of π are compared with the corresponding values at the same indices of the solution from the vector *previousSol*. If π satisfies the minimum difference constraint, it is added to the list of candidates. Otherwise, another new candidate is generated, and the similarity checking mechanism proceeds until the suitable difference constraint is provided.

Improving local optima: To take full advantage of parallel computation, this process and the following processes of BLS-OpenMP are executed on several threads. Before the local search procedure, we consider two different matrices. The first one is the delta matrix, into which we put the result from the computation of the cost difference if two elements are swapped in candidate solution π. The second one is the last swapped matrix which is used for the tabu search mechanism in the perturbation strategy. To improve the local optima, BLS-OpenMP uses the steepest descent procedure. The values in candidate permutation π are iteratively transposed to minimize the cost and reach the local optima. Each iteration of this procedure searches the neighbours to discover the best adjacent solution. Furthermore, it continues with the iterations as long as an improved solution in the neighbouring space can be obtained. When the algorithm cannot find a better adjacent solution, BLS-OpenMP proceeds to the determining Jump Magnitude section of algorithm to escape from the current local optimum.

Determining the jump magnitude: The role of deciding the measure of perturbation moves, jump magnitude is an important issue for perturbation stage of BLS-OpenMP. The magnitude of jump is determined according to the present state of neighbouring exploration. With the benefit of diversifying jump magnitude, the quality of investigation is increased. Unless the best recorded solution is improved during the predetermined number of descent phases, strong perturbation mechanism is required to get rid of the local optima. Nevertheless, if the candidate solution is enhanced, the number of perturbation moves is determined accordingly so that it can escape from the previous local optimum. However, if the cost of current solution is the same as the previous one, exploration returns to the previous local optimum. By the time BLS-OpenMP reaches to local optimum and determines the Jump Magnitude, it executes the last part of the algorithm, the perturbation phase. This phase has a very significant role in terms of diversifying the exploration areas and escaping from the current local optimum.

Diversifying search area by a distributed perturbation strategy: The last and the most significant phase of BLS-OpenMP algorithm is the distributed perturbation phase. After the jump magnitude is assigned, it tries to escape from the local optimum by selecting among three separate types of moves for perturbation according to the present state of the exploration. The adaptive perturbation mechanism makes use of tabu search, recency-based and random perturbation methods. The perturbation mechanism has an intensive diversification impact on BLS-OpenMP. It is significant to determine the most suitable perturbation type not to deteriorate the solution too much. This determination is made probabilistically by threads and the possibility of determining a specific type is identified dynamically according to the present search state and also to the current number of iteration w. When the best permutation is obtained through the enhancing exploration or by the time the number of iteration surpasses the specified threshold T when the local optima is not enhanced, w takes the value of zero. Moreover, it is practical to assure to apply tabu search perturbation type in minimum degree (Benlic & Hao, 2013a). Consequently, there is a constraint that at least the threshold value P_0 must be assigned to the possibility P of the application of Tabu Search Perturbation. P is calculated according to the following formulation:

$$P = \begin{cases} e^{-w/T}, & \text{if } e^{-w/T} > P_0 \\ P_0, & \text{otherwise} \end{cases}$$

In this formulation, T is the maximum number of iterations required to determine that no enhanced solution exists. In this case, BLS-OpenMP becomes a stronger perturbation mechanism. If the value of w increases, the possibility of applying tabu search perturbation reduces, whereas the possibility of applying the other two methods increases to obtain more powerful diversification. Apart from the possibility of applying tabu search perturbation, the probability of recency based perturbation is calculated as $(1 - P)$. Q and for the random perturbation, the probability is calculated as $(1 - P)$. $(1 - Q)$ where Q is a value between 0 and 1.

Tabu-search perturbation: Tabu search has been considered as an outstanding efficient approach for solving hard combinatorial problems such as travelling salesman, scheduling, product delivery and routing, transportation systems and manufacturing cell design. As indicated by the name, this type of perturbation is based on TS fundamentals. In fact, TS essentially looks for whether a move is applied throughout a specific number of iterations (γ). The value of γ indicates the strength of diversification. Furthermore, it considers the historic information about the last time the move was performed and it attaches importance to not perturbing the solution too much. Appropriate moves are chosen according to the constraint, as in the following formulation for set A:

$$A = \{swap(u, v) | min\{\delta(\pi, u, v)\}, (H_{uv} + \gamma) < Iter \text{ or } (\delta(\pi, u, v) + c) < c_{best},\ u \neq v \text{ and } 1 \leqslant u,\ v \leqslant n\}$$

H is defined as the matrix into which the iteration values when the move was last executed are placed, *Iter* is the current iteration number, c is the cost of the current permutation and c_{best} is the cost of the best permutation that has been found up to that moment.

Recency-based distributed perturbation: Apart from the TS perturbation type, there is a recency based perturbation mechanism which benefits from some of the historic information in the H matrix. Although the move causes a decrease in the cost, this perturbation type only considers the least recently executed moves. Indeed, the eligible moves are determined according to the constraint presented in the following formulation by set B:

$$B = \{swap(u, v) | min\{H_{uv}\},\ u \neq v \text{ and } 1 \leqslant u,\ v \leqslant n\}$$

Random distributed perturbation: The random perturbation method determines the moves to be performed. More precisely, the most suitable relocations are selected according to the constraint presented in the following formulation by set C:

$$C = \{swap(u, v) | u \neq v \text{ and } 1 \leqslant u,\ v \leqslant n\}$$

Once the perturbation genre is found, corresponding relocations determined by sets A, B and C are appropriately applied. Therefore, at the next iteration of the steepest descent algorithm, this resulting solution is accepted as the new candidate permutation.

Fast fitness calculation of the new neighbours: When the solution in the neighbouring space is obtained in the steepest descent or perturbation procedures of the algorithm, BLS-OpenMP starts its updating mechanism for this improved solution. Swapping two indexes of a current permutation and creating a different candidate may lead to a better solution. We update the cost of new solution by using the fitness value of the former one. Therefore, we can calculate the cost of the improved solution in a very fast manner (Taillard, 1991). In BLS-OpenMP, this approach is used as a performance improving computation method. Starting from a candidate

solution Φ, a neighbouring solution π is obtained by changing units r and s:

$$\pi(k) = \phi(k) \quad \forall k \neq r, s$$
$$\pi(r) = \phi(s)$$
$$\pi(s) = \phi(r)$$

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

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

The number of computations in the algorithm is a significant criterion when evaluating the performance of the heuristics. BLS-OpenMP uses a fast updating mechanism that solely computes the delta part of the novel permutation. It is not necessary to calculate each neighbouring permutation from scratch, which is a very costly process for large QAP instances.

Open Multi-Processing (OpenMP): OpenMP libraries are used to realize parallel programming on a multi-core platform. OpenMP is an API for shared-memory parallel computations and works on multi-core computers. It provides a portable and scalable model for shared memory parallel application developers. The variables can be defined as shared (global) or private. Private variables provide better performances in a shared-memory, whereas global variables may cause the threads to queue. If we classify the data as shared, it can be accessed by all threads. Hence, threads read from and write on shared variables. OpenMP follows the Fork and Join Model (see Fig. 2). Programs are initialized with a single thread, the master thread (Thread 0). At the beginning, the master thread creates a number of parallel worker threads (named as FORK). Instructions in blocks are executed in parallel by each thread. In the end, all threads are synchronized and join the master thread (named JOIN).

OpenMP provides an outstanding acceleration in computation by benefiting from the parallelization of the several cores on a computer. BLS-OpenMP divides the workloads of the search process among the threads in a coarse-grained way. Local and global variables are declared and the algorithm is designed to produce the most effective speed-up by preventing starvation. BLS-OpenMP provides almost linear speed-up during the optimization of the QAP instances.

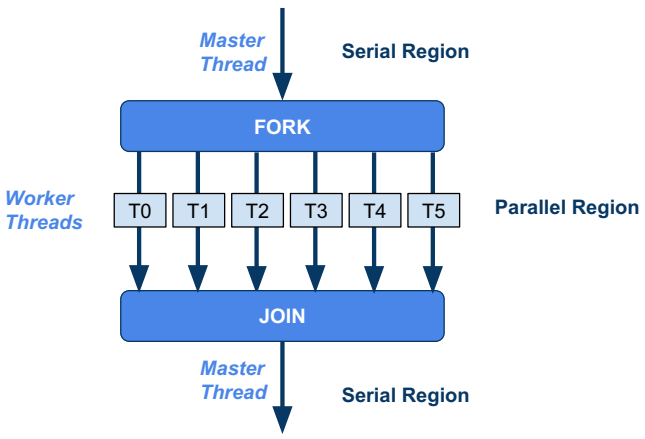


Fig. 2. The OpenMP fork and join execution model.

4. Performance evaluation of the proposed algorithm BLS-OpenMP

In this section, we explain our experimental environment and give information about the benchmark library, QAPLIB. Later, the procedure for determining the most efficient parameters for the proposed algorithm BLS-OpenMP is given. The results of our experiments on the hardest 59 QAPLIB problem instances are reported and compared with those of state-of-the-art algorithms. A personal computer with an Intel Core i7-6700 CPU 3.40 GHz with 4 cores, 16 GB Memory (RAM), and Windows 8.1 64-bit Operating System is used during the experiments. It is possible to execute eight logical processors on this computer. Four sets of the problem instances given in the QAPLIB are used during the experiments to measure the performance of BLS-OpenMP (Burkard et al., 1991; Stützle, 2006). Type 1 problems (unstructured, randomly generated instances) have distance and flow matrices that are randomly generated based on a uniform distribution. Type 2 problems (instances with grid-based distances) contain instances in which the distances are the Manhattan distances between points on a grid, whereas the flows are randomly generated. Type 3 problems (real-life instances) are produced from real-life practical QAP applications. Type 4 problems (real-life-like instances) are generated instances that resemble real-life QAP instances.

4.1. Setting the parameters of BLS-OpenMP

Working with the most appropriate parameter settings is an important issue for the performance of a heuristic algorithm. The parameters that most significantly affect BLS-OpenMP are the similarity ratio (percentage) of the starting permutations, the number of threads provide the highest speed-up, the number of iterations, and the number of multi-starts. The best performing parameters are determined on the tai60a problem instance. Later, the rest of the experiments are executed with these settings and the performance of BLS-OpenMP is compared with those of state-of-the-art algorithms.

Setting the similarity ratio: Experiments are performed to find the most effective similarity ratio on the problem set of the *tai60a* instance. While performing these experiments, the other parameters (number of threads, number of multi-starts, and number of iterations) are kept constant. These experiments are executed with 16 threads and the number of iterations is set to 10,000. The algorithm is restarted 100 times. The results of the similarity ratios are presented in Fig. 3. In the plot, the *x-axis* is the similarity ratio percentage, and the *y-axis* is the average percentage deviation (APD) from the best known solution (BKS), which is calculated according to the minimum cost obtained when the execution is terminated. The rest of the experiments are performed with the similarity ratio of 30%.

Setting the number of threads: OpenMP is a shared-memory parallelization paradigm. The performance of the threads can be affected while accessing the global shared data. We examine the most appropriate number of threads to speed-up the execution of the algorithm with respect to the number of processors. We increase the number of threads from 1 to 32, and the other parameters of the algorithm remain constant. A 30% similarity ratio, 10,000 iterations, and 100 multi-starts are the parameters used. The most appropriate number of threads is determined to be 16. The initialization of the algorithm with 16 threads can be realized with almost the same execution times as with 8 threads, and it is possible to use two times more multi-starts. Further, these experiments validate the fact that executing the algorithm on a number of threads in excess of the number of cores of computer will not speed-up the performance because

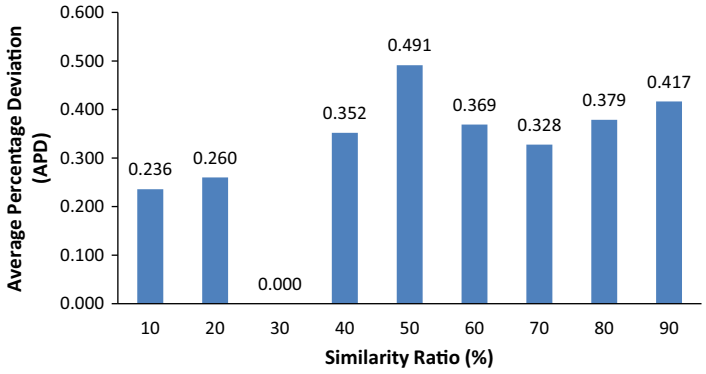


Fig. 3. Similarity ratio analysis on tai60a problem instance. APD is the average percentage deviation from the BKS. 30% similarity ratio provides minimum deviation.

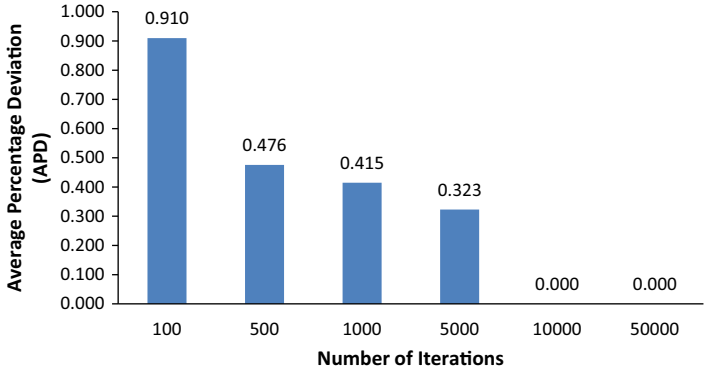


Fig. 4. APD analysis with the number of iterations on tai60a problem instance. APD is the average percentage deviation from the BKS. Execution with 10,000 and 50,000 iterations does not provide any deviation.

the logical cores operating on the same physical core share resources with one another.

Setting the number of iterations: During the experiments, we keep the other parameters of the algorithm (similarity ratio, number of threads and number of multi-starts) constant as we increase the number of iterations from 100 to 50,000. The experiments are initialized with a 30% similarity ratio and 16 threads. The process is restarted 100 times for each experiment. The computational results of these experiments are presented in Fig. 4 (see Table 1).

We define the most appropriate number of iterations for making BLS-OpenMP effective as 10,000. Although the larger number of iterations might be better, we prefer shorter execution times to ensure time efficiency. Therefore, the value of the number of iterations is set to 10,000.

Setting the number of multi-starts: We observe the performances of the algorithm multi-starts ranging from 10 to 500. These tests are initialized with 16 threads and the number of iterations is set to 10,000. The computational results of these experiments are presented in Fig. 5. When the number of multi-starts increases, the results are improved significantly. Nevertheless, each restart increases the execution time. We set the number of multi-starts with respect to the size of the problem (n) and execute BLS-OpenMP in fewer multi-starts for smaller problems and increase the number of restarts for larger problem instances.

Table 1
Execution time analysis of the number of threads on tai60a problem instance. The times are given in minutes. APD is the average percentage deviation from the BKS.

# of threads	Time (min)	APD (%)
1	30.85	0.686
2	33.13	0.279
4	37.42	0.298
8	54.84	0.331
16	107.89	0.000
24	246.29	0.257
32	>1440.00	–

Comparing OpenMP and CPU versions: In this part, experiments are separated into four parts according to the versions: the implementations of CPU, Multi-start CPU, Multi-start with similarity check CPU, and Multi-start with similarity check using OpenMP.

The computational results of these experiments are presented in Table 2 and Fig. 6. By using the advantages of multi-threads and applying a multi-start mechanism with a log-based approach for the previously searched permutations of the QAP instances, significant improvements are observed in the solution quality.

The computational results of these experiments are presented in Table 3 and Fig. 7.

Overhead of similarity checking mechanism: The similarity checking mechanism makes a significant improvement while finding the best known solutions by using a log-based approach for the previously searched permutations of the QAP. However, it has an overhead of execution time that is spent on checking the similarity. To observe this overhead, we perform experiments on the problem set, *tai60a*. In the experiments, we observe the time spent controlling the similarity check constraint is approximately 1 min, adding 0.8% to the total execution time. This running time overhead becomes larger as the problem size, number of multi-starts, and size of the recorded local optima are increased.

4.2. Comparison with state-of-the-art algorithms

We compare BLS-OpenMP with state-of-the-art algorithms in terms of the solution quality and running time efficiency. The

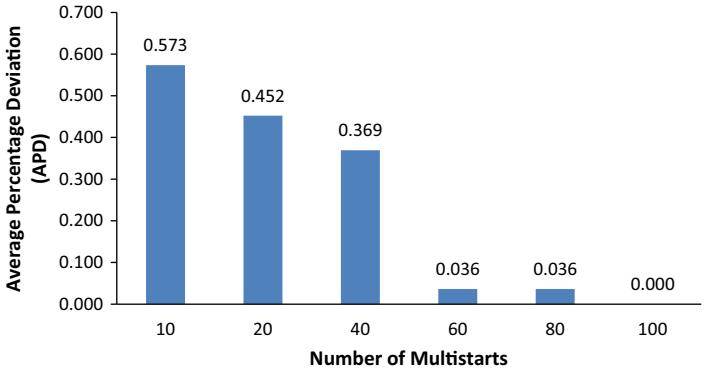


Fig. 5. APD analysis with the number of Multi-starts on tai60a problem instance. Execution with 100 multi-starts does not provide any deviation.

Table 2
Comparison of OpenMP and CPU versions of the proposed algorithms on tai60a problem instance. The execution times are given in minutes. Implementation of multi-start similarity check OpenMP is observed to provide no deviation.

Name of the algorithm	Ratio (%)	# of threads	# of iterations	# of multi-starts	Time (min)	APD (%)
CPU	–	1	3,255,266	1	107.883	0.465
Multi-start CPU	–	1	10,000	331	107.884	0.397
Multi-start with similarity check CPU	30	1	10,000	347	108.347	0.165
Multi-start with similarity check OpenMP	30	16	10,000	100	107.891	0.000

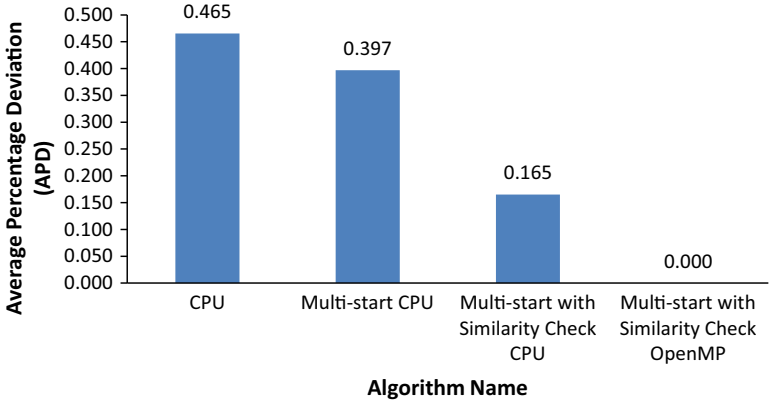


Fig. 6. Comparison of OpenMP and CPU versions of the proposed algorithms on tai60a problem instance. Implementation of multi-start similarity check OpenMP algorithm does not provide any deviation.

Table 3
Comparison of OpenMP and CPU versions on Tai100a problem instance. The times are given in minutes. Implementation of multi-start similarity check OpenMP provides the minimum deviation among them.

Name of the algorithm	Ratio (%)	# of threads	# of iterations	# of multi-starts	time (min)	APD (%)
CPU	–	1	2,736,285	1	448.517	0.732
Multi-start CPU	–	1	10,000	287	448.520	0.727
Multi-start with similarity check CPU	30	1	10,000	278	451.170	0.686
Multi-start with similarity check OpenMP	30	16	10,000	100	448.529	0.617

QAPLIB benchmark library provides a fair ground to make comparisons with the other heuristic algorithms from the literature. The state-of-the-art heuristic algorithms in the literature are the Multi-Start TS Algorithm JRG-DivTS by James et al. (2009), Iterated Tabu Search (ITS) by Misevičius (2012), Self Controlling Tabu Search (SC-Tabu) by Fescioglu-Unver and Kokar (2011), Ant Colony Optimization GA/Local Search Hybrid ACO/GA/LS by Tseng and Liang (2005), GA Hybrid with Concentric TS Operator GA/C-TS and GA Hybrid with a Strict Descent Operator GA/SD by Drezner (2005), Parallel Hybrid Algorithm (PHA) by Tosun (2015), Memetic search for the QAP (BMA) by Benlic and Hao (2015), and Great Deluge and Tabu Search (GDA) by Acan and Unveren (2015).

Table 4 presents the parameters used during the experiments. Tables 5–9 present the results of our experiments with respect to the four categories specified by Stützle (2006). The three best performing results are given in bold face.

Experiments are conducted on the hardest 59 problem instances of the QAPLIB benchmark and BKS results are obtained for 57 of the instances. Because the other QAP instances in the library are easy and can be solved optimally by the classical BLS algorithm, they

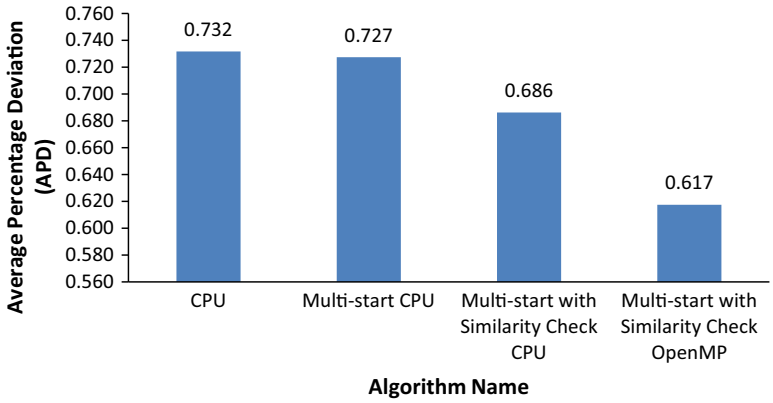


Fig. 7. Comparison between OpenMP and CPU versions on Tai100a problem instance. Implementation of multi-start similarity check OpenMP algorithm provides the minimum deviation.

Table 4
Parameter settings for BLS-OpenMP.

Problem type	Parameter	Value
Type 1–4	Similarity ratio	30%
Type 1	Number of threads	16
Type 2–4 and Nug	Number of threads	8
Type 1–4	Number of iterations	10,000
Type 1–4	Number of multi-starts	100

Table 5
Best known solutions found by BLS-OpenMP on *Nug* problem instances. BPD is the best percentage deviation from the best known solution. The execution times are given in seconds. All of the *Nug* problem instances are solved exactly.

Instance	BKS	APD (%)	BPD	Time (sec)
nug14	1014	0	0	0.016
nug15	1150	0	0	0.016
nug16a	1610	0	0	0.610
nug16b	1240	0	0	0.015
nug17	1732	0	0	1.093
nug18	1930	0	0	0.687
nug20	2570	0	0	0.046
nug21	2438	0	0	0.484
nug22	3596	0	0	0.015
nug24	3488	0	0	0.046
nug25	3744	0	0	1.640
nug27	5234	0	0	0.016
nug28	5166	0	0	1.031
nug30	6124	0	0	1.156
Average		**0**	**0**	0.491

are not used/reported in our experiments. The overall deviation for the problem instances from the BKS is obtained as 0.019% on average. The overall deviation percentage of BLS-OpenMP for the problems classified as Type 1, 2, 3, or 4 by Stützle (2006) is 0.025%. BLS-OpenMP is among the best three performing algorithms according to the obtained results. For Type 1 problem instances, BLS-OpenMP is among the best algorithms. It has only 0.125% deviation from the BKS, whereas the other two algorithms (BMA and PHA) have deviations of 0.129% and 0.131% respectively. The deviations of BLS-OpenMP for the well-known problem instances *tai*60*a* and *tai*80*a* place it among the best three best performing algorithms with 0.0% and 0.504% deviations, respectively. We verify that multi-starting a heuristic from a different exploration point is an efficient technique. It has been used in most of the state-of-the-art algorithms (James et al., 2009). In our implementation, it helps BLS-OpenMP escape from local optima by restarting the exploration with a new and well-diversified candidate through a smart beginning mechanism. In fact, we can observe its positive effect in Fig. 5 on the solution quality when the number of multi-starts is increased for the *tai*60*a* problem instance.

Speed-up and scalability are significant issues for the performance evaluation of BLS-OpenMP. Multi-core architectures (thread programming) are used by BLS-OpenMP. It makes use of threads by using the OpenMP library; therefore, the workload of optimization is shared among the threads and a higher performance is obtained. The total execution time of the algorithm is reduced in proportion to the number of threads. The more threads (with respect to the number of processors) that are used, the more the execution time is decreased until we reach the number of cores supported by the chip. The performance of the algorithm can be further improved by

Table 6
Comparison of BLS-OpenMP with state-of-the-art algorithms on Type-1 problem instances. The times are given in minutes. 7 of the Type 1 problem instances are solved exactly by BLS-OpenMP.

Instance	BKS	BLS-OpenMP		JRG-DivTS		ITS		SC-Tabu		ACO/GA/LS		GDA		BMA		PHA		CPTS	
		APD	time	APD	time	APD	time	APD	time	APD	time	APD	time	APD	time	APD	time	APD	time
tai20a	703482	**0**	0.09	**0**	0.2	**0**	0.0	0.246	0.01	–	–	**0**	2.10	**0**	0.0	**0**	0.37	**0**	0.1
tai25a	1167256	**0**	0.13	**0**	0.6	**0**	0.1	0.239	0.03	–	–	**0**	15.82	**0**	0.0	**0**	0.55	**0**	0.3
tai30a	1818146	**0**	0.20	**0**	1.3	**0**	0.2	0.154	0.07	0.341	1.4	0.091	20.29	**0**	0.0	**0**	0.97	**0**	1.6
tai35a	2422002	**0**	0.32	**0**	4.4	**0**	0.5	0.280	0.18	0.487	3.5	0.153	24.99	**0**	0.0	**0**	1.28	**0**	2.3
tai40a	3139370	**0**	32.16	0.222	5.2	0.22	1.3	0.561	0.20	0.593	13.1	0.261	27.78	**0.059**	8.1	**0**	10.60	0.148	3.5
tai50a	4938796	**0**	68.16	0.725	10.2	0.41	5.5	0.889	0.23	0.901	29.7	0.276	41.14	**0.131**	42.0	**0**	12.74	0.440	10.3
tai60a	7205962	**0**	107.89	0.718	25.7	0.45	12.5	0.940	0.41	1.068	58.5	0.448	78.86	**0.144**	67.5	**0**	19.58	0.476	26.4
tai80a	13499184	**0.504**	235.95	0.753	52.7	**0.36**	60.0	0.648	1.01	1.178	152.2	0.832	111.34	**0.426**	65.8	0.644	39.97	0.570	94.8
tai100a	21052466	0.617	448.53	0.825	142.1	**0.30**	200.0	0.977	1.99	1.115	335.6	0.874	138.32	**0.405**	44.1	**0.537**	71.89	0.558	261.2
Average		**0.125**	99.27	0.360	26.9	0.19	31.1	0.548	0.46	0.812	84.9	0.326	51.18	**0.129**	25.3	**0.131**	17.55	0.244	44.5

Table 7
Comparison of BLS-OpenMP with state-of-the-art algorithms on Type-2 problem instances. All of the Type 2 problem instances are solved exactly by BLS-OpenMP.

Instance	BKS	BLS-OpenMP		JRG-DivTS		SC-Tabu		ACO/GA/LS		GA/SD		GA/C-TS		GDA		BMA		PHA		CPTS	
		APD	Time	APD	Time	APD	Time	APD	Time	APD	Time	APD	Time	APD	Time	APD	Time	APD	Time	APD	Time
sko42	15812	**0**	0.37	**0**	4.0	0.016	0.14	**0**	0.7	0.014	0.16	**0**	1.2	**0**	2.24	**0**	0.002	**0**	1.60	**0**	5.3
sko49	23386	**0**	0.56	0.008	9.6	0.085	0.22	0.056	7.6	0.107	0.28	0.009	2.1	0.005	3.85	**0**	0.01	**0**	4.03	**0**	11.4
sko56	34458	**0**	0.81	0.002	13.2	0.069	0.34	0.012	9.1	0.054	0.42	0.001	3.2	0.001	14.72	**0**	0.02	**0**	16.24	**0**	21.0
sko64	48498	**0**	1.20	**0**	22.0	0.074	0.51	0.004	17.4	0.051	0.73	**0**	5.9	**0**	29.39	**0**	0.03	**0**	23.10	**0**	42.9
sko72	66256	**0**	1.82	0.006	38.0	0.159	0.73	0.018	70.8	0.112	0.93	0.014	8.4	0.007	37.99	**0**	3.50	**0**	33.63	**0**	69.6
sko81	90998	**0**	2.40	0.016	56.4	0.076	1.05	0.025	112.3	0.087	1.44	0.014	13.3	0.019	57.14	**0**	4.30	**0**	39.87	**0**	121.4
sko90	115534	**0**	3.25	0.026	89.6	0.134	1.44	0.042	92.1	0.139	2.31	0.011	22.4	0.031	93.83	**0**	15.30	**0**	40.53	**0**	193.7
sko100a	152002	**0**	29.80	0.027	129.2	0.094	1.99	0.021	171.0	0.114	3.42	0.018	33.6	0.029	153.17	**0**	22.30	**0**	41.72	**0**	304.8
sko100b	153890	**0**	8.51	0.008	106.6	0.059	1.99	0.012	192.4	0.096	3.47	0.011	34.1	0.015	164.27	**0**	6.50	**0**	42.32	**0**	309.6
sko100c	147862	**0**	4.28	0.006	126.7	0.039	1.99	0.005	220.6	0.075	3.22	0.003	33.8	0.013	154.51	**0**	12.00	**0**	42.19	**0**	316.1
sko100d	149576	**0**	12.87	0.027	123.5	0.091	1.99	0.029	209.2	0.137	3.45	0.049	33.9	0.017	148.86	0.006	20.90	**0**	41.91	**0**	309.8
sko100e	149150	**0**	4.33	0.009	108.8	0.037	1.99	0.002	208.1	0.071	3.31	0.002	30.7	0.016	146.15	**0**	11.90	**0**	42.48	**0**	309.1
sko100f	149036	**0**	17.11	0.023	110.3	0.122	1.99	0.034	210.9	0.148	3.55	0.032	35.7	0.013	153.38	**0**	23.00	**0**	41.98	0.003	310.3
Average		**0**	6.72	0.012	72.1	0.081	1.26	0.020	117.1	0.093	2.05	0.013	19.9	0.013	89.19	**0**	9.21	**0**	31.66	**0**	178.8

Table 8
Comparison of BLS-OpenMP with state-of-the-art algorithms on Type-3 problem instances. All of the Type 3 problem instances are solved exactly by BLS-OpenMP.

Instance	BKS	BLS-OpenMP		ACO/GA/LS		ITS		SC-TABU		GDA		BMA		PHA		CPTS	
		APD	Time	APD	Time	APD	Time	APD	Time	APD	Time	APD	Time	APD	Time	APD	Time
kra30a	88900	**0**	0.0037	**0**	0.1208	0.01	0.025	0.714	0.048	**0**	0.947	**0**	0.040	–	–	–	–
kra30b	91420	**0**	0.0330	**0**	0.6703	**0**	0.025	0.178	0.048	**0**	1.392	**0**	0.020	–	–	–	–
kra32	88700	**0**	0.0122	–	–	–	–	–	–	**0**	0.626	**0**	0.018	–	–	–	–
ste36a	9526	**0**	0.1990	**0**	0.6247	0.04	0.093	0.761	0.085	**0**	2.173	**0**	0.082	**0**	1.37	–	–
ste36b	15852	**0**	0.1170	**0**	0.1062	**0**	0.092	0.761	0.085	**0**	0.406	**0**	0.022	–	–	–	–
ste36c	8239110	**0**	0.1821	**0**	0.4547	**0**	0.092	0.761	0.085	**0**	1.395	**0**	0.043	**0**	1.37	**0**	2.5
esc32b	168	**0**	0.0004	**0**	0.0070	**0**	0.032	–	–	**0**	0.598	**0**	0.002	–	–	–	–
esc32c	642	**0**	0.0003	**0**	0.0005	**0**	0.035	–	–	**0**	0.005	**0**	0.002	–	–	–	–
esc32d	200	**0**	0.0003	**0**	0.0028	**0**	0.033	–	–	**0**	0.026	**0**	0.002	–	–	–	–
esc32e	2	**0**	0.0003	**0**	0.0003	**0**	0.032	–	–	**0**	0.004	**0**	0.002	–	–	–	–
esc32g	6	**0**	0.0002	**0**	0.0003	**0**	0.032	–	–	**0**	0.005	**0**	0.000	–	–	–	–
esc32h	438	**0**	0.0024	**0**	0.0048	**0**	0.035	–	–	**0**	0.005	**0**	0.002	–	–	–	–
esc64a	116	**0**	0.0012	**0**	0.0043	**0**	0.183	1.207	0.001	**0**	0.038	**0**	0.003	–	–	–	–
esc128	64	**0**	0.0052	**0**	0.0378	0.01	1.000	14.271	0.124	**0**	2.135	**0**	0.022	–	–	–	–
Average		**0**	0.0398	**0**	0.1565	**0**	0.131	2.665	0.068	**0**	0.697	**0**	0.018	**0**	1.37	**0**	2.5

Table 9
Comparison of BLS-OpenMP with state-of-the-art algorithms on Type-4 problem instances. All of the Type 4 problem instances are solved exactly by BLS-OpenMP.

Instance	BKS	BLS-OpenMP		JRG-DivTS		ITS		ACO/GA/LS		SC-TABU		GDA		BMA		PHA		CPTS	
		APD	Time	APD	Time	APD	Time	APD	Time	APD	Time	APD	Time	APD	Time	APD	Time	APD	Time
tai20b	122455319	**0**	0.072	**0**	0.2	**0**	0.01	–	–	0.335	0.003	**0**	0.074	**0**	0.000	**0**	0.37	**0**	0.1
tai25b	344355646	**0**	0.023	**0**	0.5	**0**	0.02	–	–	0.702	0.010	**0**	0.327	**0**	0.000	**0**	0.57	**0**	0.4
tai30b	637117113	**0**	0.221	**0**	1.3	0.010	0.04	**0**	0.3	0.313	0.035	**0**	1.148	**0**	0.000	**0**	0.81	**0**	1.2
tai35b	283315445	**0**	0.295	**0**	2.4	0.020	0.08	**0**	0.3	–	–	**0**	6.387	**0**	0.000	**0**	1.11	**0**	2.4
tai40b	637250948	**0**	0.303	**0**	3.2	0.010	0.21	**0**	0.6	0.219	0.092	**0**	4.877	**0**	0.003	**0**	1.57	**0**	4.5
tai50b	458821517	**0**	0.707	**0**	8.8	0.020	0.55	**0**	2.9	0.281	0.235	0.005	10.249	**0**	1.200	**0**	5.82	**0**	13.8
tai60b	608215054	**0**	18.622	**0**	17.1	0.040	1.07	**0**	2.8	0.886	0.415	**0**	33.639	**0**	5.200	**0**	9.49	**0**	30.4
tai80b	818415043	**0**	218.053	0.006	58.2	0.230	3.00	**0**	60.3	0.798	1.004	0.025	0.005	**0**	31.300	**0**	27.70	**0**	110.9
tai100b	1185996137	**0**	160.797	0.056	118.9	0.140	6.67	0.010	698.9	0.553	1.988	0.028	72.601	**0**	13.600	**0**	42.50	0.001	241.0
Average		**0**	44.344	0.007	23.4	0.052	1.29	0.001	109.4	0.511	0.473	0.006	14.367	**0**	5.700	**0**	9.99	**0**	45.0

initializing a larger number of threads with a computer that has a more powerful CPU. This will reduce the execution time and give a greater chance to improve the number of multi-starts and search the landscape of the QAP more effectively.

Considering the search spaces of the combinatorial optimization problems is a new and interesting research area (Tayarani-N & Prugel-Bennett, 2014). Analysing the distribution of the search space, determining correlation between the local optima, and keeping track of the most promising areas are reported to be new efficient tools for local search techniques. In a recent study, Porumbel, Hao, and Kuntz (2010) reported a distance measure for the graph colouring problem. They introduce the clustering hypothesis that high quality solutions are not randomly scattered throughout the search space but are instead grouped into clusters within spheres of specific diameter. Our approach is unique and was applied for the first time on the QAP. It diversifies each new solution by considering the similarities of the previously-discovered areas.

5. Conclusions and future work

In this study, we apply a similarity checking mechanism to a recently proposed local search heuristic (BLS) by using the Levenshtein Distance metric to solve of the QAP. This approach makes significant improvements on the BLS algorithm while searching for the best known solutions (by using a log-based approach) and prevents the redundant exploration of previously searched similar permutations of a QAP instance. Furthermore, BLS-OpenMP uses multi-threaded programming (OpenMP). A stagnation-aware search for the best known solutions of the QAP is executed concurrently on several cores with diversified trajectories while considering the similarity percentages of the previously visited local optima. The exploration of the search space is improved by selecting new starting points intelligently and speeding-up the fitness evaluations linearly with the number of the processors/threads. BLS-OpenMP is executed on the hardest 59 problem instances of the QAP library benchmark and the obtained results show that it is possible to find the best known solutions for 57 of the instances. The overall deviation for the rest of the solutions is 0.019% on average, which is a significant improvement compared with the results of state-of-the-art algorithms. In future work, we plan to apply other similarity measures, such as the Hamming, Chebyshev, Manhattan, and Euclidian distances to solve the QAP. We believe that the performances of the other heuristics, such as tabu search and simulated annealing algorithms, can also be improved by adding a similarity checking mechanism. Further improvements can be obtained by running the algorithm on a high performance computer with thousands of processors.

References

Acan, A., & Unveren, A. (2015). A great deluge and tabu search hybrid with two-stage memory support for quadratic assignment problem. *Applied Soft Computing, 36*, 185–203.

Ben-David, G., & Malah, D. (2005). Bounds on the performance of vector-quantizers under channel errors. *IEEE Transactions on Information Theory, 51*(6), 2227–2235.

Benlic, U., & Hao, J. K. (2012). A study of breakout local search for the minimum sum coloring problem. In *Asia-Pacific conference on simulated evolution and learning* (pp. 128–137).

Benlic, U., & Hao, J. K. (2013a). Breakout local search for the quadratic assignment problem. *Applied Mathematics and Computation, 219*(9), 4800–4815.

Benlic, U., & Hao, J. K. (2013b). Breakout local search for maximum clique problems. *Computers & Operations Research, 40*(1), 192–206.

Benlic, U., & Hao, J. K. (2013c). Breakout local search for the max-cut problem. *Engineering Applications of Artificial Intelligence, 26*(3), 1162–1173.

Benlic, U., & Hao, J. K. (2015). Memetic search for the quadratic assignment problem. *Expert Systems with Applications, 42*(1), 584–595.

Burkard, R. E., Karisch, S. E., & Rendl, F. (1991). QAPLIB—A quadratic assignment problem library. *European Journal of Operational Research, 55*(1), 115–119.

Carrizo, J., Tinetti, F. G., & Moscato, P. (1992). A computational ecology for the quadratic assignment problem. In *Proceedings of the 21st meeting on informatics and operations research*. .

Connolly, D. T. (1990). An improved annealing scheme for the QAP. *European Journal of Operational Research, 46*(1), 93–100.

Cung, V. D., Mautor, T., Michelon, P., & Tavares, A. I. (1997). *Scatter search based approach for the quadratic assignment problem*.

Czapiński, M. (2013). An effective parallel multistart tabu search for quadratic assignment problem on CUDA platform. *Journal of Parallel and Distributed Computing, 73*(11), 1461–1468.

Dickey, J. W., & Hopkins, J. W. (1972). Campus building arrangement using TOPAZ. *Transportation Research, 6*(1), 59–68.

Dokeroglu, T. (2015). Hybrid teaching-learning-based optimization algorithms for the quadratic assignment problem. *Computers & Industrial Engineering, 85*, 86–101.

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

Drezner, Z. (2003). A new genetic algorithm for the quadratic assignment problem. *INFORMS Journal on Computing, 15*(3), 320–330.

Drezner, Z. (2005). The extended concentric tabu for the quadratic assignment problem. *European Journal of Operational Research, 160*(2), 416–422.

Duman, E., & Or, I. (2007). The quadratic assignment problem in the context of the printed circuit board assembly process. *Computers & Operations Research, 34*(1), 163–179.

Fescioglu-Unver, N., & Kokar, M. M. (2011). Self controlling tabu search algorithm for the quadratic assignment problem. *Computers & Industrial Engineering, 60*(2), 310–319.

Gambardella, L. M., Taillard, E. D., & Dorigo, M. (1999). Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society, 50*(2), 167–176.

Geoffrion, A. M., & Graves, G. W. (1976). Scheduling parallel production lines with changeover costs: Practical application of a quadratic assignment/LP approach. *Operations Research, 24*(4), 595–610.

Harris, M., Berretta, R., Inostroza-Ponta, M., & Moscato, P. (2015). A memetic algorithm for the quadratic assignment problem with parallel local search. In *2015 IEEE congress on evolutionary computation (CEC)* (pp. 838–845).

James, T., Rego, C., & Glover, F. (2009). A cooperative parallel tabu search algorithm for the QAP. *European Journal of Operational Research, 195*(3), 810–826.

Koopmans, T. C., & Beckmann, M. J. (1957). Assignment problems and the location of economic activities. *Econometrica, 25*(1), 53–76.

Loiola, E. M., de Abreu, N. M. M., Boaventura-Netto, P. O., Hahn, P., & Querido, T. (2007). A survey for the quadratic assignment problem. *European Journal of Operational Research, 176*(2), 657–690.

Lstiburek, M., Stejskal, J., Misevičius, A., Korecky, J., & El-Kassaby, Y. A. (2015). Expansion of the minimum-inbreeding seed orchard design to operational scale. *Tree Genetics & Genomes, 11*(1), 1–8.

Meneses, H., & Inostroza-Ponta, M. (2011). Evaluating memory schemas in a memetic algorithm for the quadratic assignment problem. In *2011 30th international conference of the Chilean Computer Science Society (SCCC)* (pp. 14–18).

Misevičius, A. (2012). An implementation of the iterated tabu search algorithm for the quadratic assignment problem. *OR Spectrum, 34*(3), 665–690.

Porumbel, D. C., Hao, J. K., & Kuntz, P. (2010). A search space cartography for guiding graph coloring heuristics. *Computers & Operations Research, 37*(4), 769–778.

Ristad, E. S., & Yianilos, P. N. (1998). Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 20*(5), 522–532.

Sahni, S., & Gonzales, T. (1976). P-complete approximation algorithms. *Journal of the Association for Computing Machinery, 23*, 555–565.

Skorin-Kapov, J. (1990). Tabu search applied to the quadratic assignment problem. *ORSA Journal on Computing, 2*(1), 33–45.

Steinberg, L. (1961). The backboard wiring problem: A placement algorithm. *SIAM Review, 3*(1), 37–50.

Stützle, T. (2006). Iterated local search for the quadratic assignment problem. *European Journal of Operational Research, 174*(3), 1519–1539.

Stützle, T., & Dorigo, M. (1999). ACO algorithms for the quadratic assignment problem. *New Ideas in Optimization*, 33–50.

Taillard, E. (1991). Robust taboo search for the quadratic assignment problem. *Parallel Computing, 17*(4–5), 443–455.

Tayarani-N, M. H., & Prugel-Bennett, A. (2014). On the landscape of combinatorial optimization problems. *IEEE Transactions on Evolutionary Computation, 18*(3), 420–434.

Tosun, U. (2015). On the performance of parallel hybrid algorithms for the solution of the quadratic assignment problem. *Engineering Applications of Artificial Intelligence, 39*, 267–278.

Tseng, L., & Liang, S. (2005). A hybrid metaheuristic for the quadratic assignment problem. *Computational Optimization and Applications, 34*(1), 85–113.

Tsutsui, S., & Fujimoto, N. (2009). *Solving quadratic assignment problems by genetic algorithms with GPU computation: A case study*. Montreal Quebec, Canada: GECCO.

Wilhelm, M. R., & Ward, T. L. (1987). Solving quadratic assignment problems by simulated annealing. *IIE Transactions, 19*(1), 107–119.