Received: 2 October 2020 | Revised: 10 March 2021 | Accepted: 12 March 2021

DOI: 10.1002/cpe.6321

RESEARCH ARTICLE

WILEY

A performance study of meta-heuristic approaches for quadratic assignment problem

Thimershen Achary[1] | **Shivani Pillay**[1] | **Sarah M. Pillai**[1] | **Malusi Mqadi**[1] | **Emma Genders**[1] | **Absalom E. Ezugwu**[2]

[1]School of Mathematics, Statistics, and Computer Science, University of KwaZulu-Natal, Private Bag X54001, Durban, South Africa 4000,

[2]School of Mathematics, Statistics, and Computer Science, University of KwaZulu-Natal, King Edward Road, Pietermaritzburg, KwaZulu-Natal, 3201, South Africa

Correspondence
Absalom E. Ezugwu, School of Mathematics, Statistics, and Computer Science, University of KwaZulu-Natal, Pietermaritzburg 3201, South Africa.
Email: ezugwua@ukzn.ac.za

Abstract

The quadratic assignment problem (QAP) is a well-known challenging combinatorial optimization problem that has received many researchers' attention with varied real-world and industrial applications areas. It is noteworthy to mention that a plethora of nature-inspired optimization algorithms have successfully been used to solve various optimization problems, including several variants of the QAPs. In this article, a comprehensive literature review is presented to show the most relevant nature-inspired algorithms that have been used in solving the QAP. More so, extensive experiments are conducted and analyzed to show the performance of the well-known state-of-the-art nature-inspired meta-heuristic optimization algorithms in solving the QAP, including the ant colony optimization (ACO), bat algorithm, genetic algorithm (GA), particle swarm optimization (PSO), and tabu search algorithm. Besides, a modified variant of the discrete PSO algorithm is implemented and compared with existing approaches. The six selected algorithms' performances, including the modified PSO, are validated on eight commonly used QAP instances of varying complexity and size, considering the quality of solutions achieved and computational time consumed by the representative algorithms. The numerical results revealed that the most competitive algorithm was ACO, while the GA appeared to be the worst performed algorithm among the six compared meta-heuristic algorithms. However, based on the extensive analysis conducted on the tested algorithms, further improvements are suggested, including implementing new modified versions of the tested algorithms to tackle the QAP and its variant instances.

KEYWORDS

genetic algorithm, Meta-heuristic, particle swarm optimization, quadratic assignment problem

1 | INTRODUCTION

The quadratic assignment problem (QAP) was first introduced in Reference 1 based on the study presented by Koopmans and Beckmann in 1957. The QAP can be interpreted as the problem of assigning n facilities to n locations. Suppose the distances between locations and the flows between facilities are known. In that case, the goal is to find the best assignment of facilities to locations such that the sum of the product of the resultant distances and the flows is minimized. Intuitively, this interpretation encourages facilities with high flows between them to be placed at locations that are separated by a small distance. A formal representation of the QAP can be explained as follows: Given n facilities $\{F_1, F_2, \ldots, F_n\}$ and n locations

$\{L_1, L_2, \ldots, L_n\}$, distance matrix D_{nn} and the flow matrix F_{nn}, where d_{ij} is the distance between location i and location j and f_{ij} is the flow between facility i and facility j, the QAP can be defined as follows:

$$\min_{\phi \epsilon P(n)} Z_\phi = \sum_{i=1}^{n}\sum_{j=1}^{n} f_{ij} d_{\phi(i)\phi(j)}. \tag{1}$$

Here, $P(n)$ is the set of all permutations (corresponding to the assignment solutions) of the set of integers $\{1, 2, \ldots n\}$ and ϕ is the current solution. The term $f_{ij}d_{\phi(i)\phi(j)}$ describes the cost of assigning facility F_i to location $\phi(i)$ and F_j to location $\phi(j)$. A feasible solution is one where one facility is assigned to only one location, and one location is assigned to only one facility. Similarly, the QAP can also be formulated as a quadratic integer program. This formulation involves the use of a *permutation matrix*, $X = x_{ij}$ where

$$x_{ij} = \begin{cases} 1 & \text{if } \phi(i) = j \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

X can be interpreted as an assignment relation from F to L. X must meet the following *assignment constraints*:

$$x_{ij} \in \{0, 1\}, \quad i = 1, 2, \ldots, n; \quad j = 1, 2, \ldots, n, \tag{3}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, 2, \ldots, n, \tag{4}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, 2, \ldots, n. \tag{5}$$

The QAP can then be defined as:

$$\min \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{r=1}^{n}\sum_{r=1}^{n} f_{ij} d_{rs} x_{ir} x_{js}. \tag{6}$$

The term $f_{ij}d_{rs}x_{ir}x_{js}$ describes the cost of the assignment implied by X. An example of a QAP problem and solution is given as follows. The *nug5* dataset is given by the distance and flow matrix. This is a QAP instance for $n = 5$. Therefore, there are five facilities and five locations:

$$L = \begin{bmatrix} 0 & 1 & 1 & 2 & 3 \\ 1 & 0 & 2 & 1 & 2 \\ 1 & 2 & 0 & 1 & 2 \\ 2 & 1 & 1 & 0 & 1 \\ 3 & 2 & 2 & 1 & 0 \end{bmatrix}$$

$$F = \begin{bmatrix} 0 & 5 & 2 & 4 & 1 \\ 5 & 0 & 3 & 0 & 2 \\ 2 & 3 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 5 \\ 1 & 2 & 0 & 5 & 0 \end{bmatrix}$$

The distances between any two locations are given in matrix L, and the flows between any two facilities are given in matrix F. The optimal solution for this QAP instance is $\phi = \{4, 5, 1, 2, 2\}$. This assignment means that the fourth facility is assigned to the first location, fifth facility is assigned to the second location, and so forth. The cost of this solution is 50.

There are many classical combinatorial problems such as: facility layout problems, the travelling salesman problem, the bin packing problem, the graph partitioning problem, and the maximum clique problem, which can be expressed as a QAP.[2] The QAP can also be used to model a variety of problems with "real-world" applications such as backboard wiring in electrical circuits, analysis of chemical reactions, layout planning of keyboard, hospitals, campuses, and so forth.[2,3] The QAP has been proven to be NP-hard.[2] Due to its complexity and its diverse applications, a variety of approaches to the QAP have been researched. These approaches include exact, meta-heuristic, and hybrid approaches. The exact algorithm approaches include dynamic programming,[4] branch and bound,[5,6] and cutting planes.[7] The only algorithm guaranteed to find the optimal solution is

the branch and bound algorithms. However, these algorithms are generally unable to solve QAP instances where the number of facilities is greater than 20.[2] Therefore, because of this very limitation of the exact methods, researchers have often shifted their attention to using the approximate or meta-heuristic algorithms in finding near-optimal solutions for the QAP and its variants.

The several meta-heuristic approaches that have been applied to the QAP include tabu search (TS),[8,9] evolutionary strategy,[10] genetic algorithm (GA),[11-14] ant colony optimization (ACO),[15-18] particle swarm optimization (PSO),[19-22] symbiotic organisms search algorithm.[23] Local search techniques are also effective for the QAP, and so many hybrid approaches which combine meta-heuristics with local search have been developed.[24-26] In this article, we compare and contrast a variety of meta-heuristics covering both the population-based and single solution-based approaches to solve the QAP. The tested representative meta-heuristic algorithms include the ACO, bat algorithm (BA), GA, PSO, modified PSO, and the TS.

The motive of this article is to conduct a comprehensive analysis test on different nature-inspired optimization algorithms capable of finding good quality solutions for the QAP. Another aspect is to investigate which of the tested algorithms are able to conveniently alleviate the high computational cost associated with finding the best solutions for the QAP. Note that high computational cost is the main drawback of the exact methods when tackling the same problem. Although meta-heuristic algorithms cannot guarantee finding optimal solutions like the exact methods, they are able to obtain acceptable results within a reasonable time frame.[27] As such, the comparative study of this nature demonstrated in this article can serve as a practical indicator for interested domain experts in selecting more appropriate implementation methods for tackling difficult and complex QAP and its variants instances.

The comparative analysis study presented in this article is different from similar existing studies available in the literature. The six algorithms discussed here comprise of the two main classifications of meta-heuristic approaches, namely, population-based and single solution-based techniques.[28,29] Note that unlike the analysis presented in Reference 30, only two algorithms, namely, ant and bees algorithms' performance, were evaluated compared with the six algorithms tested in the current work. Similarly, the comparative analysis discussed in Reference 31 for which only the TS and simulated annealing algorithms' performance were compared differs significantly from our implemented study. Furthermore, in Reference 32, the performance of different variants of the ACO algorithms on QAP was done. However, this differs from the current study, which presents a broader range of comparative performance analysis. Moreover, the empirical validation of the obtained numerical results for the current work is performed on a highly diverse set of QAP instances.

The rest of this article is organized as follows: Section 2 provides an overview of the related work for comparison between different meta-heuristic algorithms for solving the QAP. A brief description of the representative algorithms for solving the QAP is given in Section 3. The six meta-heuristic algorithms modeling and design are described in Section 4. This section is further subdivided into five subsections, namely, ACO, BA, GA, PSO, and TS with details on the algorithmic design concepts of the selected algorithms. In Section 5, the experimentation, dataset description, results and discussion are presented. The concluding remarks and future research directions are given in Section 6.

2 | LITERATURE REVIEW

Li et al.[33] investigated the various lower bounds used for the QAP and proposed two new lower bounds through optimal reduction schemes. The paper discussed three categories of lower bounds; namely, Gilmore–Lawler bound (GLB), eigenvalue-based bounds, and other lower bounds, most of which solves the linear assignment problem. The new bounds presented better results than the GLB on the random problems and competitively on Nugent problems. The reduction schemes become the GLB when used on smaller matrices. Therefore, improvements can be made to allow them to be used with smaller matrices.

In Reference 34, Maniezzo and Colorini introduced the first ant system (AS) applied to the QAP. Inspired by the behavior of real ants, they proposed a new heuristic for the QAP to improve the global search by updating a global memory structure together with local search and lower bounds. The algorithm uses m artificial ants to construct solutions and simultaneously find better solutions. The paper utilizes the GLBs for its ratio of effectiveness and computational cost. The proposed AS was compared with another meta-heuristic GRASP. The algorithm found optimal solutions for the majority of the instances tested and achieved an average percentage error of 0.27. In addition, it performed better than the average percentage error of the GRASP, which was 0.66. The results of the work have shown that ants do not converge to just one possible solution but on multiple solutions to improve the search.

In Reference 35, Mouhoub and Wang employed two improved local search algorithms to enhance the application of the AS for the QAP. The first is the 2-opt local search, and the second is the TS. In both searches, they improved the search algorithms by using a random walk for neighborhood search to prevent getting trapped in local optima and considering more solutions in the neighborhood when stuck in local optima. This they did instead of stopping the local search like the original search methods. The improvements made on the local search methods were found to perform better than the original algorithms and were implemented with the MaxMin ant system (MMAS).

Stützlee and Dorigo[15] reviewed the different improvements of the AS for the QAP. The article presented the computational results of the original AS, MMAS, the ANTS algorithm applied to QAP, fast Ant (FANT), the HAS-QAP, robust tabu search, and a genetic hybrid (GH). These variants differ in the concepts and ideas of how the ACO works. Although GH performed well on the random graphs, MMAS however performed much better in real-life instances. A comparison of local search algorithms best suited for ACO algorithms was done, and the result showed that the local

search implemented should be considered based on the instances. It also showed that simple local searches would be adequate to high-quality solutions.

The original BA, introduced in Reference 36, was built to obtain optimal or near-optimal solutions to continuous optimization problems. To use the BA to get optimal or near-optimal solutions for the QAP (combinatorial problem), a technique to discretize position vectors is required, more specifically discretization that results in a permutation of integers. Summaries of various discretization techniques for swarm intelligence (SI) algorithms are provided in Reference 37. Similarly, the usage of the smallest position vector for a modified BA was done in Reference 38. Motivated by the fact that Random key and Smallest position vector achieved the same goal, this study uses similar implementation as presented in Reference 38 to compare with other selected meta-heuristic algorithms.

The paper that was modeled for the GA used in this experiment is Reference 39. The researchers chose a GA design with a two-point crossover operation, used a mutation probability of 0.2 and a roulette wheel selection. They tested their implementation using the eight instances from the QAPLIB dataset, namely, nug12, nug17, nug20, nug24, nug28, chr12a.dat, chr12b.dat, and chr15a.dat. Their experiments yielded good result with the worse Gap percentage being 1%. However, since their implementation was not tested on large instances, it is hard to accept their findings as a true reflection of how the algorithm would perform given any random instance from the QAPLIB dataset.

A number of discrete forms of PSO exist that includes a binary version of PSO,[40] a rank-based approach to PSO,[41] and a set-based approach.[42-44] In addition, a number of these discrete approaches have been developed specifically for the QAP.[19-22] A fuzzy approach to discretizing PSO was introduced in Reference 20. The position and velocity of each particle are represented as matrices as opposed to being represented as vectors as in the original PSO representation. The position and velocity update rules are defined in terms of matrix operations. This approach was successfully applied to a number of instances of the QAP. However, this approach was not used in this work as it was decided that representing each particle by two matrices and performing matrix operations on these matrices at each iteration was inefficient.

Hafiz et al.[21] proposed a probability-based approach for the learning in PSO. In this approach, the position of each particle is a valid permutation in the solution space. The velocity of each particle is given by a matrix of probabilities. Cognitive and social learning sets are introduced and used to make the velocity and position updates. The addition of these sets, and the method for position and velocity updating, made this approach lose the simplicity of the original PSO algorithm and so this approach was not used in this work.

Pradeepmon et al.[22] proposed another discretized PSO for the QAP. In this attempt, the position is given by a permutation in the solution space. Instead of using a velocity to update the position vector, a new position update strategy is proposed. This strategy borrows from the mutation and crossover operations defined for GA. The simplicity of the original PSO algorithm is not maintained due to the introduction of the crossover and mutation operators. Another disadvantage of this algorithm is that, at each iteration, the particle can only learn from either its current position, its own best position, the global best position or the local best position. This approach results in the lost of some learning capability.

Mamaghani et al.[19] introduced a modified version of PSO as a discretized PSO for the QAP. In this approach, many of the elements such as the definition of a position vector and a velocity vector from the original PSO are retained. A sequence, which is a permutation of numbers representing feasible solutions to the QAP instance, is introduced. The sequence is decoded from the position vector using a heuristic rule, known as the smallest position value rule, so that the fitness of the solution may be evaluated. This approach was chosen since it retained the simplicity and learning ability characteristic of the original continuous PSO algorithm.

The TS algorithms can differ in respects to the tabu list and aspiration criterion. The TS can be formed as deterministic (such as a fixed tabu) or stochastic (such as the robust TS). The advantage of using a TS is that it has the ability to start with a simple implementation, then further enhanced by adding more specialized components, thereby improving the diversification and intensification to the search. Several authors have adapted the TS to the QAP, yielding promising results.[8,45,46]

Skorin-Kapov[8] proposed a fixed TS applied to the QAP. The implemented algorithm used a predetermined tabu size list that could be updated. This is conditioned to where elements in the tabu list were forbidden unless it yielded an objective function better than the current best-known solution. The solutions selected were added to the tabu list in a FIFO order. This implementation was seen as very effective for the QAP. This technique was used for the proposed TS used in this article.

Taillard[45] introduced a modified robust TS approach for the QAP. Instead of having a fixed tabu, the size of the tabu list was assigned randomly. The aspiration criteria were further modified, allowing solutions to be introduced if it had not been chosen within a fairly large number of iterations. Despite having its presence in the tabu list or a lower objective function value, another highly interesting attribute that Taillard's algorithm introduced is that the neighborhood was explored in $O(n^2)$ instead of $O(n^3)$.

Misevicius[46] proposed a new version of the TS for the QAP. Applying mutations to the best solution achieved in the search led to more diversification when tested on instance in the QAPLIB. It achieved highly promising results, yielding the best-known solution for many of the instances. Thus, showing the effectiveness of the TS as a heuristic that can be applied for the QAP. However, this mutation technique was not used in this article and it was implemented in the generic algorithm.

A comparative study of GA, TS, and SA for solving QAP problems from QAPLIB was done in Reference 47. The performance of the algorithms was compared in terms of relative difference/percentage deviation of the best cost obtained from the known best cost of the QAP instance used; this is a metric that we have adopted in our study. In the conclusion of Reference 47, it is stated that future research lies in the comparison of more different algorithms, which is what we have done in this study.

In-keeping with the future research mentioned in the conclusion of Reference 47, we selected the BA, PSO and its modified ACO and two of the algorithms tested in Reference 47, namely, TS and GA. More so, after reviewing several heuristic techniques, their advantages and implementation to the QAP, the following motivations discussed subsequently are the reasons we've chosen the aforementioned meta-heuristic techniques. For example, the ACO has a parallel construction process since ants build solutions independently and simultaneously,[48] premature convergence is eluded by distributed computation. The ACO has a guaranteed convergence and highly adaptable.

The BA can be easily implemented. BA use of hill climbing for further intensification was done in the implementation in Reference 38. The BA is a computationally efficient algorithm that has shown promising results in tests done in Reference 38. It will be interesting to see how this algorithm would compete against other meta-heuristic algorithms as it was only tested against PSO in Reference 38.

TABLE 1 An overview of the studied meta-heuristic algorithms for the QAP

S/N	Name	Algorithm	Proposed method	Dataset	Author	Year
1	EGATS	GA and TS algorithms	A hybrid algorithm that combines an elite GA and TS	QAPLIB	Zhang et al.[49]	2020
2	EOFPA	Flower pollination algorithm	A hybrid elite opposition-based learning and flower pollination algorithm	QAPLIB	Abdel-Baset et al.[50]	2017
3	ABC-QAP and PABC-QAP	ABC and parallel ABC	Hybrid artificial bee colony optimization algorithm and tabu search	QAPLIB	Dokeroglu et al.[51]	2019
4	WAITS	Whale optimization algorithm and tabu search	A memetic algorithm using the whale optimization algorithm integrated with a tabu search	QAPLIB	Abdel-Basset et al.[52]	2018
5	BMA	Memetic algorithm and BLS	Integrates breakout local search (BLS) within the population-based evolutionary computing framework	QAPLIB	Benlic and Hao[53]	2015
6	HBRKGA	GA	A hybrid approximate approach for the QAP based upon the framework of the biased random key GA	QAPLIB	Lalla-Ruiz et al.[54]	2016
7	TLBO-RTS and Parallel TLBO-RTS	TLBO and TS	Hybrid teaching-learning-based optimization (TLBO) with robust tabu search (RTS) algorithm	QAPLIB	Dokeroglu[55]	2015
8	QIEA	Quantum-inspired evolutionary algorithm	Application of the quantum-inspired evolutionary algorithm (QIEA) for determining minimal costs of the assignment in the QAP	QAPLIB	Chmiel and Kwiecien[56]	2018
9	ANT-QAP and BA-QAP	Ant algorithm and bees algorithm	A comparison of nature inspired algorithms for the QAP	QAPLIB	Chmiel et al.[30]	2017
10	TALO	Antlion optimizer (ALO)	Tournament selection based ALO algorithm with classic ALO for solving QAP	QAPLIB	Kılıç and Yüzgeç[57]	2019
11	ACO	Ant colony optimization	Worst-case bound on a simple ACO algorithm called (1+1) Max–min ant algorithm ((1+1) MMAA) for the QAP problem	QAPLIB	Xia and Zhou[32]	2018
12	C-FA	Firefly algorithm (FA)	An improved firefly algorithm based on chaos mapping strategy	QAPLIB	Guo et al.[58]	2020
13	TS-GQAP	Construction algorithms and a simple tabu search heuristic	TS heuristic is developed for generalized quadratic assignment problem (GQAP)	Continuous facility layout problem (FLP)	McKendall and Li[59]	2017
14	Discretized PSO	PSO	A probabilistic representation for the PSO particle's velocity for solving the QAP	QAPLIB	Hafiz and Abdennour[21]	2016
15	RKCS	Cuckoo search (CS) algorithm	adaptation of the random-key Cuckoo search (RKCS) algorithm for the QAP	QAPLIB	Ouaarab et al.[31]	2017

The GA can work well with discrete optimization problems. The algorithm uses probabilistic selection rules based on the candidate solution's fitness value. It is easier to model a candidate solution using a chromosome. The GA has received wide attention from optimization enthusiastic from different research domains, thereby recording several real-world and industrial based application experiences.

PSO can easily be implemented with few parameters to be fine-tuned. The algorithm has a fast convergence speed and can be computationally efficient depending on the graph size of the problem being solved. The PSO has been employed to solve several complex optimization problems with a good track record of finding near-optimal solutions. More so, several variants of the algorithms also exist. Interestingly, only a few variants of the PSO applications exist for the problem being considered in this article, that is, the QAP.

The TS can be used as both local and global search method by introducing prohibiting technique. The algorithm has been shown to be very effective and produced good quality solutions for combinatorial optimization problems. The TS utilizes intensification mechanism that accelerates the search process by exploiting patterns in good solutions spaces. The algorithm is easily implemented and can be hybridized with other global search methods to serve as an update local search process. A summary overview of some recently reported meta-heuristic algorithms implementation for the QAP is given in Table 1.

3 | META-HEURISTICS PRELIMINARIES

In this section, we give a brief overview and explanation of the representative meta-heuristic algorithms, namely, the ACO, BA, PSO, GA, and TS that were implemented in this article and applied to solve the QAP. However, the algorithmic design structures for these set of tested meta-heuristic approaches are reserved for and discussed in Section 4.

In the early 1990s, Dorigo first introduced the ACO as his PhD thesis, which based the technique on the behavior of real ant colonies.[60] This heuristic was initially created to solve the travelling salesman problem, after which it is used to solve other complex optimization problems. In 1999, Vittorio Maniezzo and Albert Colorini proposed the first application of the AS to the QAP.[15] After which, several enhancements of the AS were made, including MAX–MIN Ant system, FANT, and HAS-QAP. ASs are also commonly paired with local search algorithms such as 2-opt local search and TS.[35] The ACO is considered to be one of the best-performing algorithms, especially for structured, real-life instances,[15] and produces the best solutions for combinatorial problems.[35]

The original BA, introduced by Yang[36] in 2010, is based on the echolocation behavior of bats. This meta-heuristic algorithm was built to provide optimal or near-optimal solutions to continuous optimization problems and performed remarkably well when compared with PSO and GA as used in Reference 36. Besides, it is interesting to note that the BA has some similarities to PSO with a slight modification, that is, using loudness and pulse rate to do exploration and exploitation efficiently.[36]

For combinatorial problems, the original BA would not function well in the same problem domain. A technique of converting continuous position vectors to a discrete solution would be required. The BA variant called binary bat algorithm was introduced in Reference 61. This BA variant binarizes a given position vector using the sigmoid function. The application of this BA variant led to classification. If we go a step further in the combinatorial problem domain and say we want each solution to be a permutation, this is where the BA variant proposed in Reference 38 succeeds. It is a BA variant built specifically for the QAP. It uses the smallest position vector rule to not only discretize the solution but to create a permutation. The BA proposed in Reference 38 can be regarded as highly efficient in the sense that you would never have an invalid solution that would have to undergo punishment or reprocessing for a problem that requires a permutation in the general sense. A discrete BA was developed in Reference 26 for the QAP. To create this particular discrete BA, some modifications to the calculation of velocity and position were done to ensure discreteness. The discrete BA for QAP proposed in Reference 26 has been extensively tested with QAPLIB instances. Hence we choose the BA variant proposed in Reference 38 to test and compare in this study as we wish to see how it might fair in the QAP domain when compared with other meta-heuristic algorithms as it was only tested against PSO.

The GA is a meta-heuristic inspired by the natural process of natural selection.[62] Researchers have used GA to solve optimization problems using biologically inspired functions such as selection, crossover and mutation.[63] The GA can work well with discrete optimization problems.[64] More so, the GA and its variants are consistently used to solve and generate useful solutions for many practical optimizations problems. The GA belongs to the class of evolutionary algorithms, which generate their solutions using techniques inspired by natural evolution such as chromosomes, genotypes, and natural selection process. For example, in GA, a population of strings or chromosomes, which encode candidate solutions for an optimization problem, evolves toward better solutions.[39] The solution encoding representation for the GA, is often represented as strings of binary numbers, using 0s and 1s. The evolution usually starts from a population of randomly generated individuals over a number of generations. The fitness of every candidate individual in the population is iteratively evaluated, while multiple individuals are stochastically selected from the current population based on their fitness values. Those individuals with better fitness are consistently modified to form a new population, and after which the new population is then used in the next generation of the algorithm iterative phase. The algorithm terminates when either a maximum number of generations has been reached, or a satisfactory fitness level has been reached for the population.

A meta-heuristic approach for solving the QAP that has not received much attention is PSO which was introduced by Kennedy and Eberhart in 1995. It is a SI algorithm inspired by the social behavior patterns of animals that live and interact in large groups such as birds, fish, bees, and so forth. In PSO, the search space is explored using a swarm of particles. Each particle represents a candidate solution, and each particle is able to learn from their personal best position and other best positions of the swarm. The PSO was developed as a method for optimizing continuous nonlinear functions and so a discrete form of PSO is needed in order to apply it to the QAP. The discrete form used in this work was introduced by Mamaghani and Meybodi.[19] In order to improve the results obtained by this algorithm, an addition to the algorithm is made to promote diversity and prevent premature convergence. Furthermore, the discrete form is used to implement a modified version of the PSO algorithm, inspired by both the unified particle swarm optimization (UPSO), introduced in Reference 65, and the multiswarm particle swarm optimization (MS-PSO), introduced in Reference 66.

The TS algorithms have been widely applied to numerous combinatorial optimization problems, finding solutions within a comparatively short running time.[67] The TS was originally inspired by Fred Glover.[68] It falls under local search-based procedures. However, despite being a local meta-heuristic it is designed to explore the search space beyond the local optimum. This is achieved by imposing constraints, restricting the search to explore certain forbidden regions. The general implementation is based on a neighborhood, allowing moves to ascend to neighboring solutions. It uses a memory structure called the tabu list, which restricts moves to solutions that have already been recently visited. Thus, this approach prevents cycling and further drives the search into new unexplored regions. However, often the tabu can prohibit promising moves, which can lead to stagnation. To combat this, an aspiration criteria can be defined where it overrides the tabu thereby allowing moves to solutions with a cost value that is better than the current best-known solution. For a comprehensive background review study on optimization meta-heuristics, interested readers are referred to the work presented by Boussaïd et al.[69]

4 | META-HEURISTIC ALGORITHM DESIGN

4.1 | Ant colony optimization

The ACO uses graphs to solve computational problems probabilistically. It stimulates the behavior of real ants and how they direct each other from their colonies to food sources. They do this by first finding a good food source and leaves a trail of pheromones on the path used. The more a path is used, the more pheromones are left on that path, and the colony follows the trail with the most pheromones. The shortest path would contain the most pheromones since ants can move faster on the shorter path. The algorithm used in this article is inspired by the AS for the QAP in Reference 34.

Limitations of the ACO:

- Even though convergence is guaranteed, the time to converge is unknown[48]
- The probability distribution changes with iteration[48]

4.1.1 | Implementation to solve the QAP

For each instance, an artificial ant k assigns facility i to a location j and leaves a trace τ_{ij} on the coupling (i, j). The choice of assigning a facility i to a location j is calculated with a probability, which is a function of the quality of the coupling (i, j) and quantity of the trace of the coupling (i, j). Given by the equation:

$$P_{ij}^{k}(t) = \begin{cases} \frac{\alpha \cdot \tau_{ij}(t) + (1-\alpha) \cdot \eta_{ij}}{\sum(\alpha \cdot \tau_{ij}(t) + (1-\alpha) \cdot \eta_{ij})} & \text{if } j \epsilon\ tabu_k \\ 0 & \text{otherwise,} \end{cases} \tag{7}$$

where $0 \leq \alpha \leq 1$, and η_{ij} is the desirability of assigning facility i to location j. When a facility is assigned to a location, it will be added to a tabu list $tabu_k$. After all ants construct their solutions, the trace matrix is updated with the equation:

$$\tau_{ij}(t+1) = \rho \tau_{ij}(t) + \Delta \tau_{ij} \tag{8}$$

and $\Delta \tau_{ij}$ is,

$$\Delta \tau_{ij} = \sum_{k=1}^{m} \Delta \tau_{ij}^{k} \tag{9}$$

and $\Delta\tau_{ij}^{k}$ the quantity of the trace left from the kth ant. In order to generate the cost matrix and possible permutations, we implemented the best-known lower bound, the GLB. Denoted by:

$$z_{\mathrm{GL}} = \min z = \sum_{i,j=1}^{n} (\min \sum_{h,k=1}^{n} d_{ih} f_{jk} x_{hk}) x_{ij}. \tag{10}$$

This corresponds to solving the linear assignment using the Hungarian method (or Munkres method). A greedy algorithm is implemented to aid in the local search. Following the algorithm design presented in Reference 34, the pseudocode for the AS or ACO implemented in this article to solve the QAP is shown in algorithm listing 1.

Algorithm 1. Ant colony optimization for QAP

```
Calculate upper bound by multiplying the sorted distance and facility matrices;
Initialize the trace matrix;
m - number of ants;
n - number of iterations;
Initialize tabulist;
Put m ants on node 1;
for ( i in n ) {
    for ( j in m ) {
        while tabulist is not full do
            Calculate the probability of assigning facility i to location j using Equation (1);
            Add assigned facility to tabuList;
        end
        Use the solutions to compute the trace quantity of coupling(i,j);
        Update best permutation;
        for ( each coupling (i, j) ) {
            calculate Δτ_ij;
            Update the trace matrix;
        }
    }
    Clear tabulist;
}
Print the best permutation;
```

4.2 | Bat algorithm

4.2.1 | Standard BA

The BA[36] is a meta-heuristic algorithm whose creation was inspired by the echolocation behavior of bats, that is, how bats use echolocation to find prey and is built for finding optimal or near-optimal solutions to continuous optimization problems. The BA, presented in Reference 36, depends on six hyperparameters, namely: population-size, frequency bounds, loudness bounds, loudness decay rate (alpha), gamma (used in the calculation of emission rate at every iteration) and initial emission rate. The natural motion of bats (individuals) in the BA is based on the following equations:

$$f_i = f_{\min} + (f_{\max} - f_{\min})\beta, \text{where } \beta \text{ is a random number between 0 and 1,} \tag{11}$$

$$\mathbf{v}_i^t = \mathbf{v}_i^{t-1} + (\mathbf{x}_\mathbf{i}^{\mathbf{t-1}} - \mathbf{x}_*) f_i, \tag{12}$$

$$\mathbf{x}_i^t = \mathbf{x}_i^{t-1} + \mathbf{v}_i^t. \tag{13}$$

Random walk (an exploration technique) is done using the following update formula:

$$\mathbf{x}_{\text{new}} = \mathbf{x}_{\text{old}} + \epsilon A^t, \tag{14}$$

where A^t is the average loudness of all bats at iteration t and ϵ is a random vector and $\epsilon \in [-1, 1]$. It is mentioned in Reference 36 and is also backed up by depicted results that the BA is more powerful than the GA and PSO.

4.2.2 | Modified BA

A simple modification to the standard BA was done in Reference 38, main additions are the usage of smallest position value rule to discretize position vector (to convert to a solution), and hill-climbing to further increase the quality of intensification. Natural motion rules are based on the same rules as the original BA, except for Equation (2), which is as follows:

$$\mathbf{v}_i^t = \mathbf{v}_i^{t-1} + (\mathbf{x}_* - \mathbf{x}_i^{t-1}) f_i. \tag{15}$$

This equation moves the BA towards the best-known solution rather than away from it (which is what occurs in Equations (1)–(3), that is, exploitation is done rather than exploration. There is a restructuring of the BA done in Reference 38, which makes flow look a bit different from that of the original BA; this is probably done due to the slight changes in exploration and exploitation techniques used in Reference 38. This modified BA was compared with PSO in Reference 38, results looked of the modified BA looked promising. Hence the need to include it in this study. Similar to the algorithm design presented in Reference 38, the pseudocode for the modified BA that is implemented for the problem at hand is shown in the Algorithm listing 2.

4.3 | Genetic algorithm

The GA is initialized with a population of random individuals called chromosomes. Each chromosome represents a solution candidate to a given problem. The goal of the GA is to determine the fittest pair of chromosomes in the population using a fitness function. The fittest pair is used to create new solutions called the next generation or offspring. The fitter the solution candidates the most likely they are to reproduce. The algorithm loops for a predetermined set of iterations, at the end, the fittest individual in the population is presented as the most optimal solution the algorithm can generate. The main concepts for the GA are the chromosome encoding, fitness function, selection, recombination and the evolution scheme.

Chromosome The chromosome is a string representation for a candidate solution. For the QAP, the indexes represent the locations and the values at an index represent a facility assigned to that location.

Figure 1 gives an example of a Chromosome encoding for the QAP.

The indexes in the above example represent the locations, and the elements represent the facilities. The first entry is the allocation of facility 2 to location 1.

Fitness function The fitness function assigns a score for each chromosome. For our implementation, the objective function for the QAP was used as the fitness function.

Crossover The two-point crossover was used in the implementation of the algorithm. With this type of crossover, two points from two-parent chromosomes are fixed. The genetic material between those two points becomes the fixed component for the child chromosome, the rest of the genetic material is inherited from the second parent.

Parent 1: 3 5 |2 4| 1

Parent 2: 2 1 |4 5| 3

Offspring 1: 1 5 |2 4| 3

Offspring 2: 3 2 |4 5| 1

Mutation The mutation is the process that may alter the characteristics a child inherits from its parents. For the QAP chromosome, two facilities are selected at random, and their locations are swapped.

2	4	3	1	5
1	2	3	4	5

FIGURE 1 Example of a chromosome

Algorithm 2. The modified bat algorithm for the QAP

```
x = vector of position vectors;
m = number of bats;
n = number of dimensions;
v = vector of bat velocities;
f = vector of bat frequencies;
r = vector of bat emission rates;
fitness = vector of bat fitnesses;
globalBest = best solution found throughout runtime;
globalFitness = fitness of global bets;
α = loudness decay rate;
r_0 = initial emission rate;

for ( i := 1 to m ) {
    for ( j := 1 to n ) {
        x[i][j] := Random[0,4];
        v[i][j] := Random[-4,4];
    }
    A[i] := Random[1,2];
    f[i] := Random[0,1];
    r[i] := Random[0,1];
    fitness[i]:= Fitness(SPV(x[i]));
}
globaBest:= x[argmax(fitnesses)].copy();
globalBestFitness:= max(fitnesses);

for ( t:=1 to T ) {
    for ( i:=1 to m ) {
        QAPsolution:= SPV(x[i]);
        new_fitness:= Fitness(QAPsolution);
        rand:= Random[0,1];
        If rand < A[i] and new_fitness < fitnesses[i]fitnesses[i]:= new_fitness;
        A[i]:= αA[i];
        r[i]:= r_0(1 − e^{λ t});
    }
}
indexOfBestFitness := argmax(fitness);
globalBestFitness:= fitnesses[indexOfBestFitness];
globalBest:= x[indexOfBestFitness].copy();

return SPV(globalBest);
return globalBestFitness;

hillClimbingFunc(x[indexOfBestFitness]);
for ( k:= 1 to m ) {
    rand:= Random[0,1];
    if rand > r[k] then
        x[k] := x[k] + εAverage(A), where ε is a random vector;
    end
    if rand< A[i] and fitnesses[k] > globalBestFitness then
        β := Random[0,1]f[k]:= f_min + (f_max − f_max) ∗ β;
        v[k]:= v[k] + (globalBest -x[k]) * f[k];
    end
}
```

Selection In the selection process, we determine the number of offspring needed from the crossover process to complete the next generation. For this experiment, roulette wheel selection is used. Candidate solutions are given a probability of being selected for a mating that is proportional to their fitness value. The GA algorithm implementation pseudocode is shown in algorithm listing 3.

Algorithm 3. The genetic algorithm

n: is the population size;
x: is the crossover rate;
η: is the mutation rate;
Initialize generation 0;
k:=0;
P_k:= a population of n randomly selected individuals;
Evaluate P_k:
Compute the fitness(*i*) for each $i \in P_k$
while *k less than* t_i **do**
  Create generation $k+1$
  Copy:
  Select $(1-x)xn$ members of P_k and insert into P_{k+1}
  Mutate:
  Select $\eta \times n$ members from P_{k+1}; pair them up; produce offspring; insert offspring into P_{k+1}
  Evaluate P_{k+1}:
  Compute the fitness(*i*) for each $i \in P_k$
  Increment:
  $k := k+1$
end
return the fittest individual from $P_k = 0$

4.4 | Particle swarm optimization

4.4.1 | Standard particle swarm optimization

In this section, a formal description of the PSO algorithm used in this work is given. This algorithm is inspired by the algorithm introduced in Reference 19.

The PSO is a population-based search method where the population is termed "the swarm" and it consists of a number of particles. The particles move iteratively through the search space. For a QAP instance of size *n*, each particle *i* has a position, given by a continuous position vector $\vec{x}_i = \{x_{i1}, x_{i2}, \ldots, x_{in}\}$ and a velocity, given by a continuous velocity vector $\vec{v}_i = \{v_{i1}, v_{i2}, \ldots, v_{in}\}$.

The position vector of each particle *i* can be decoded into a sequence ϕ_i which is a permutation of numbers $\{\phi_{i1}, \phi_{i2}, \ldots \phi_{in}\}$ which represents a feasible solution to the QAP problem instance. The sequence associated with the position vector or particle *i* can be interpreted as follows: $\phi_{ij} = k$ means that facility *k* has been assigned to location *j* in the current solution. The cost of the solution can also be obtained from this sequence. To obtain the sequence from the continuous position vector, the smallest position value rule is applied: the index of the smallest number in the position vector is assigned the first position in the sequence, the index of the second smallest number in the position vector is assigned the second position in the sequence and so forth.

Each particle must remember its best position so far and the best position among the swarm $\vec{g_{\text{best}}} = \{g_{\text{best}1}, g_{\text{best}2}, \ldots, g_{\text{best}n}\}$. The personal and global best are defined as follows:

$$\vec{p_{\text{best}i}} = \arg\min(f(\vec{p_{\text{best}i}}(t-1)), f(\vec{p_{\text{best}i}}(t))), \tag{16}$$

$$\vec{g_{\text{best}}} = \arg\min_{i=1}^{m}(f(\vec{g_{\text{best}}}(t-1)), f(\vec{x_1}(t)), f(\vec{x_2}(t)) \ldots f(\vec{x_m}(t))). \tag{17}$$

Particles move through the search space according to the following velocity and position update vectors below:

$$\vec{v}_i(t) = w\vec{v}_i(t-1) + c_1 r_1(\vec{p_{\text{best}i}}(t-1) - \vec{x}_i(t-1)) + c_2 r_2(\vec{g_{\text{best}}}(t-1) - \vec{x}_i(t-1)), \tag{18}$$

$$\vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t), \tag{19}$$

where t is the iteration number, r_1 and r_2 are random numbers between 0 and 1, c_1 and c_2 are positive constants. c_1 is known as the coefficient of the self-recognition component and determines how much the particle's best position thus far influences its movement through the search space and c_2 is known as the coefficient of the social component and determines how much the swarm's best position thus far influences the particle's movement through the search space. w is known as the inertia factor and is usually set to vary linearly between 0 and 1 throughout the iterative process.

A scheme to increase the diversity in the latter stages of the search process in other to prevent premature convergence is also included in the algorithm. This scheme involves reinitializing the position of all the particles once stagnation is detected, while leaving the velocity of the particles unchanged.[70] This reinvigorates the particles' exploration capabilities while retaining the "experience" of the particles. This is achieved through the use of a "Refresh Gap" and the use of the "Refresh Gap" is shown in the algorithm. The pseudocode for the PSO algorithm is given below in the algorithm listing 4.

Algorithm 4. The particle swarm optimization algorithm

Initialize the size of the swarm, m Initialize the coefficient of the self-recognition component, c_1, and the coefficient of the social component, c_2 Initialize the all positions of the swarm randomly and set all the velocities to $\vec{0}$ Set the value of the refresh gap, RG and let $count = 0$ Set the maximum number of iterations, *MaximumIterations*

while $t \leq$ *MaximumIterations* **do**
  $t = t + 1$ Find the sequence represented by the position of each particle Calculate the cost f for each particle Update the global best in the swarm, if necessary **if** *the global best cost improves* **then**
    $count = 0$
  else
    $count+ = 1$
  end
  if $count == RG$ **then**
    $count = 0$ Reset the position of each particle
  for (*each particle* **)** {
    Update the personal best of the particle, if necessary Update the position and velocity of the particle
  }
end

4.4.2 | Modified particle swarm optimization

This section introduces a modified version of PSO that was used to solve QAP instances. This modified version is inspired by UPSO and MS-PSO and builds on the standard PSO algorithm described in algorithm listing 4. To modify the PSO algorithm and produce this modified version of the algorithm, the following changes to the above algorithm are made: At the start of the algorithm, the swarm is divided into k subswarms. This is also a step in MS-PSO. Each subswarm, i, has a best particle defined by:

$$\vec{\text{group}}_{\text{best}i}(t) = \text{argmin}(f(\vec{\text{group}}_{\text{best}i}(t-1)), f(\vec{x_1}(t)), f(\vec{x_2}(t)) \ \ldots \ f(\vec{x_d}(t))), \tag{20}$$

where $\vec{x_1}, \vec{x_2} \ \ldots \ \vec{x_d}$ are all in subswarm i.

A factor, $u \in [0, 1]$, is introduced. The position update rule remains the same while the velocity update rule becomes:

$$\vec{v_i}(t) = w_u \vec{v_i}(t-1) + R_1(\vec{p_{\text{best}}i}(t-1) - \vec{x_i}(t-1)) + R_2(\vec{g_{\text{best}}}(t-1) - \vec{x_i}(t-1)) + R_3(\vec{\text{group}}_{\text{best}i}(t-1) - \vec{x_i}(t-1)), \tag{21}$$

where,

$$w_u = wr_n u + w(1-u), \tag{22}$$

$$R_1 = c_1 r_1 r_3 u + c1_r 1(1-u), \tag{23}$$

$$R_2 = c_2 r_2 r_n u, \tag{24}$$

$$R_3 = (1-u)c_2 r_2 \tag{25}$$

and r_n is a random number between 0 and 1.

The velocity update rule is the same velocity update rule from UPSO except that instead of using the local best of the neighborhood of a particle to update its velocity, the best of the particles subswarm is used. The factor u controls the influence of the best particle in a particle's subswarm on its movement. For $u = 1$, the particle moves only in the direction of the global best particle. For $u = 0$, the particle moves only in the direction of the best particle in its subswarm. The pseudocode for the modified PSO algorithm is given below in algorithm listing 5.

Algorithm 5. The modified particle swarm optimization algorithm

Initialize the size of the swarm, m and the number of subswarms, k Initialize the coefficient of the self-recognition component, c_1, and the coefficient of the social component, c_2 and the factor u Initialize the all positions of the swarm randomly and set all the velocities to $\vec{0}$ Set the value of the refresh gap, RG and let $count = 0$ Set the maximum number of iterations, $MaximumIterations$ **while** $t \leq MaximumIterations$ **do**
  $t = t + 1$ Find the sequence represented by the position of each particle Calculate the cost f for each particle Update the global best particle, if necessary **for (** *each subswarm* **)** {
    Update the group best in that subswarm, if necessary
  }
  if *the global best cost improves* **then**
    $count = 0$
  else
    $count+ = 1$
  end
  if $count == RG$ **then**
    $count = 0$ Reset the position of each particle
  for (*each particle* **)** {
    Update the personal best for that particle, if necessary Update the position and velocity of the particle
  }
end

4.5 | Tabu search

In this section, a formal description of the TS algorithm used in this work is given. This TS algorithm was a simplified version of the algorithm inspired by Taillard.[45]

The TS is a local search-based method. For the QAP the search space can be defined as $S = s|s = (s(1), s(2), \ldots, s(n))$, where n is the size of the solution. In the TS that was implemented the first permutation was generated randomly, forming $s0$. To generate the neighborhood, given s from S, a 2-exchange neighborhood function was used. The neighborhood can be defined as $N2(s) = s'|s'\ S, d(s, s') = 2$, where $d(s, s')$ is the distance between solutions s and s' in this case 2. A move from the current permutation solution, s, to the neighboring permutation solution, s' is calculated as follows,

$$p_{ij} : \sqcap \rightarrow \sqcap(i, j = 1, 2, \ldots, n), \tag{26}$$

where the ith and the jth elements in the current solution, s, are exchanged to form the neighboring solution s'. The following process was then repeated; for the current solution search through its neighborhood. If it finds a candidate solution that is not in the tabu list or the following aspiration criteria holds, the candidate solution has a better objective cost than the current solution. The move is performed, and the current solution is replaced by its neighbor solution. The current solution then is used as a starting point for the next trial. The tabu list is updated by adding the current solution to it. The current solution is then compared with the best-known solution. If it has a better objective cost, it then becomes the best-known solution. The best solution is then returned after a predetermined number of iterations have been performed. The TS uses short term memory, where recently visited solutions are stored in a tabu List. If the tabu list is full the first element is removed, that is, elements expire based on first in first out. Similar to the implementation algorithm presented in Reference 45, the pseudocode for the modified TS algorithm used in this study is given below in algorithm listing 6.

Where the size of the tabu list was set to n (the solution size) and the neighborhood size was set to $\frac{n(n-1)}{2}$. The process was repeated until a termination criteria was met, which was the iteration number that was set to $5n$.

5 | EXPERIMENTATION, PARAMETER SETTINGS, AND DATASET

Each algorithm discussed was implemented in Python on Google Colab (a cloud-based notebook), with two CPU, 2.2 GHz processing speed, and 13 GB Ram. Ten trials were performed for each instance by each meta-heuristic algorithm proposed. Thus, the average results achieved over the

Algorithm 6. The tabu search algorithm

```
Initialize parameters of the tabu search;
Randomly generate an initial state s_0;
currentsolution ← s_0;
bestsolution ← currentsolution;
Tabu ← [];
repeat
    Neighborhood ← getNeighbors(currentsolution);
    currentsolution ← Neighborhood[0];
    for ( Candidate in Neighborhood ) {
        If (Candidate not in Tabu) and ( z(Candidate) < z(currentsolution))
        currentsolution ← Candidate;
    }
    If z(currentsolution) < z(bestsolution)bestsolution ← currentsolution;
    Tabu.push(currentsolution);
    If Tabu.size < tabuSizeTabu.removeFirst();
until t ≤ MaximumIterations;
return bestsolution;
```

10 runs were used to get the average percentage deviations to the instances best-known solution. These results were then used to compare the performance of the different meta-heuristic algorithms, which is further discussed in the results section.

5.1 | Parameter settings

In this article, we have considered six meta-heuristic algorithms, which are investigated further on the basis of the individual algorithm capability to find good quality solutions and computational time complexity involved for the algorithms to obtain those solutions. In this section, we present all the parameter configurations for each of the six algorithms implemented to tackle the QAP. The parameter settings are shown in Tables 2–7 below. It is very important to highlight here that because the choice of parameter settings can significantly affect the quality of solutions generated by the individual algorithms, several experiments were performed in order to find the best combination of the selected parameters values that would give the desired competitive results. Therefore, following the concepts of the parameter fine-tuning presented in Reference 52, only one test problem is selected from each instance to ensure the efficient selection of the parameters values. In addition, note that in order to ensure fair and unbiased comparisons, the same parameter values for population size and maximum number of iteration were used for the common algorithms such as the PSO and its modified variant. Moreover, the final adopted number of population size and iteration are obtained after extensive experimental trials with different values. We observed that all algorithms show their best performance using the set of parameters values presented in Tables 2–7.

5.2 | Dataset description

The dataset instances used in this study came from the QAP library, QAPLIB.[3] A range of different types of problem instances were used, each having different complexity levels and solution lengths. This was done in order to test and compare how well each of the meta-heuristic algorithms, namely,

TABLE 2 Parameter settings for ANT system (or ACO)

Parameter name	Parameter value
Number of ants used, m	5
Alpha(pheromone factor)	0.5
Coefficient that represents the trace persistence, p	0.9
Iteration number	$5n$/nug28 =120

TABLE 3 Parameter settings for bat algorithm

Parameter name	Parameter value
Dimension bounds	[0,4]
Frequency bounds	[0,1]
Velocity	[1,2]
Population size	40
Initial emission rate	1.0
Alpha	0.97
Gamma	0.05
Iteration number	20,000

TABLE 4 Parameter settings for genetic algorithm

Parameter name	Parameter value
Population size	100
Mutation rate	0.1
Iteration number	1000

TABLE 5 Parameter settings for particle swarm optimization

Parameter name	Parameter value
Swarm size	120
Iteration number	600
Self-recognition coefficient c_1	2
Social coefficient c_2	2
Inertia weight w	1
Alpha	0.975
r_1, r_2	0.5
Refresh gap	10

TABLE 6 Parameter settings for modified particle swarm optimization

Parameter name	Parameter value
Swarm size	120
Number of subswarms	3
Iteration number	600
Self-recognition coefficient c_1	2
Social coefficient c_2	2
Inertia weight w	1
Alpha	0.975
r_1, r_2, r_3	0.5
Refresh gap	10

TABLE 7 Parameter settings for tabu search

Parameter name	Parameter value
Neighborhood size	$\frac{n(n-1)}{2}$
Tabu width	n
Iteration number	$5n$

TABLE 8 Instances used from the QAPLIB

Instance	*n* (solution size)	Best known solution
nug8a	8	214
nug12	12	578
nug28	28	5166
tai12a	12	224,416
tai20a	20	703,482
tai30a	30	1,818,146
esc16a	16	68
esc32e	32	2

ACO, BA, GA, PSO, modified PSO, and TS performed on each different instance. The types of QAP instances used are described as follows. Nugent's instance (nug8a, nug12, nug28) which are commonly used instances where the distance matrix is a Manhattan distance matrix. Taillard's instances (tai12a, tai20a, tai30a) were randomly generated. Eschermann's instances (esc16a, esc32e) are based on sequential circuits. The instances used are given in Table 8 below with their relative solution size and best-known solution.

5.3 | Results and discussion

In this section, comprehensive experiments are conducted to show the performance of the well-known nature-inspired optimization algorithms in solving the QAPs. All the tested algorithms were executed using the parameter settings described in Tables 2–7. The performance of each algorithm on the selected eight QAP problem instances taken from the QAPLIB is discussed. For each algorithm, the best and worst performance curves of the algorithm on the QAP instance are presented. It is important to highlight here that the problem instance in which each algorithm performed the best one is the instance for which the algorithm achieved the lowest percentage deviation of the average cost from the best-known minimum. Similarly, the instance that the algorithm performed the worst one is the instance for which the algorithm achieved the highest percentage deviation of the average cost from the known minimum. Note that the raw simulation result data from the experiments can be found in Appendix A. The following subsections present the detailed experimental results discussion for each of the tested algorithms, starting with the ACO algorithm.

5.3.1 | ACO (or ANT system)

Figure 2 shows the results obtained for the ACO algorithm on the eight instances. From the results obtained, the ACO performs exceptionally well for instances with the number of dimensions less than 20, that is, nug8a, nug12, tai12a, esc16a. This algorithm obtains optimal solutions on average according to the results of our experiment.

For instances with the number of dimensions greater than or equal to 20, ACO obtained the optimal solution for esc32e only. This could be because the complexity of the distance and flow matrices for esc32e is low, although the number of dimensions is large; and also because many values in the distance and flow matrix are 0. For nug28, tai20a, and tai30a, the algorithm performed well in terms of the solution produced as the percentage deviation of the average best cost obtained from the best-known minimum is quite low (<0.3% for nug28, <0.4% for tai20a, and <2.5% for tai30a).

The ACO had the best performance on esc32e (shown in Figure 3(A)). The first ant in the first few iterations obtains the optimal solution for all runs. The reason for this is probably that this specific QAP instance has a lower complexity. ACO performs the worst for the tai30a instance (shown in Figure 3(B)), possibly due to a combination of the high complexity of the instance and its high number of dimensions.

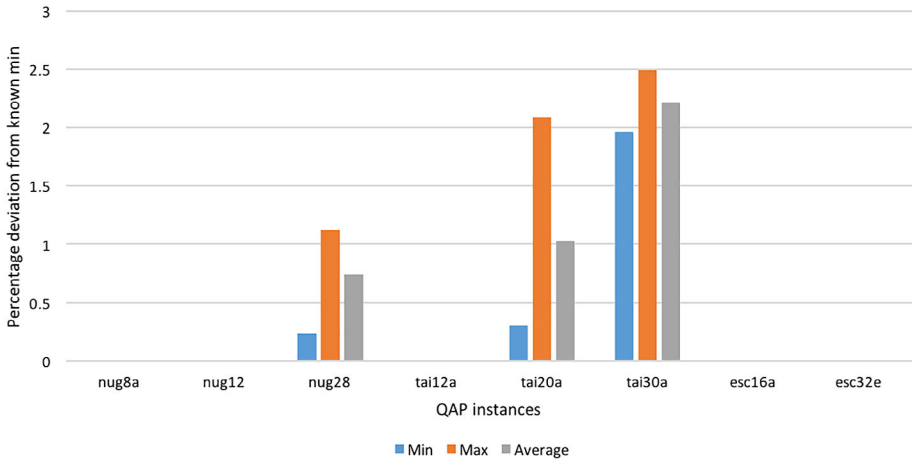
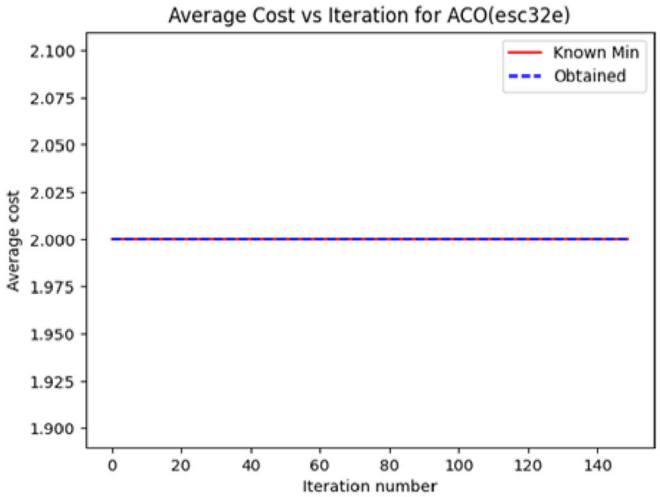
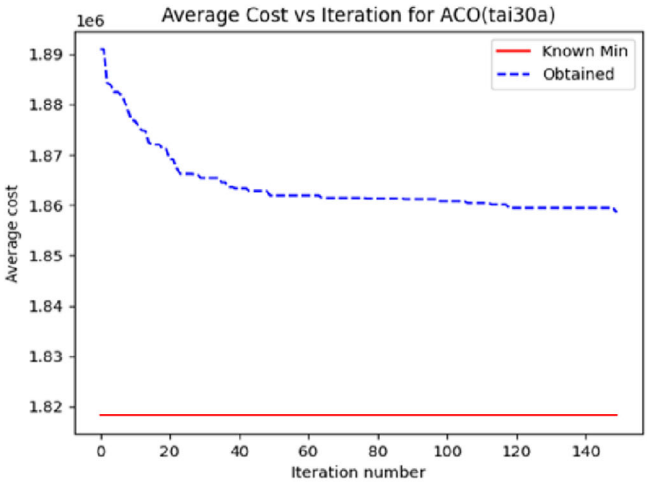


FIGURE 2 Bar graph showing the percentage deviation of minimum, maximum, and average cost value from known minimum cost value, found by ANT system (or ACO) for several QAP instances. ACO, ant colony optimization; QAP, quadratic assignment problem



(A) Graph of average cost vs. iteration number on QAP instance where ACO performed the best, which is esc32e.



(B) Graph of average cost vs. iteration number on QAP instance where ACO performed the worst, which is tai30a.

FIGURE 3 Ant colony optimization (ACO) performance curves

5.3.2 | Particle swarm optimization and modified particle swarm optimization

Figure 4 shows the results obtained by PSO on the eight problem instances. Figure 5 shows the results obtained by modified PSO on the eight problem instances. For some instances of the QAP with n<20, such as nug8, nug12, and esc16e, both PSO and modified PSO was able to obtain the optimal solutions. Furthermore, the percentage deviation of the average of best values obtained by PSO and modified PSO for each of these instances was relatively low (<1% for nug8, <5% for nug12, and <7% for esc16e). This shows that PSO and modified PSO performed well with these instances of the QAP. However, for other instances of the QAP with n<20, such as tai12a, both PSO and modified PSO were unable to obtain optimal solutions. The percentage deviation of the minimum cost obtained by PSO for tai12a was approximately 6.91%. The percentage deviation of the minimum cost obtained by modified PSO for tai12a was approximately 4.06%. The percentage deviations for maximum cost and average cost obtained by modified PSO were also lower than those obtained by PSO. This shows that modified PSO performed better than PSO for this problem instance and that modified PSO may be better than PSO for solving more complex QAP instances for n<20.

For n>=20, the only QAP instance that both PSO and modified PSO obtained the optimal solution for is esc32e. In addition, for this problem instance, both PSO and modified PSO achieved a 0% percentage deviation of the average cost values. This makes esc32e the instance that both PSO and modified PSO performed the best on. However, for other QAP instances with n>=20, both PSO and modified PSO performed poorly. Neither PSO nor modified PSO was able to obtain the optimal solution for tai20a, nug28, or tai30a. However, it should be noted that modified PSO performed better than PSO for these problem instances. The minimum cost values obtained by modified PSO for tai20a, nug28, and tai30a were lower than

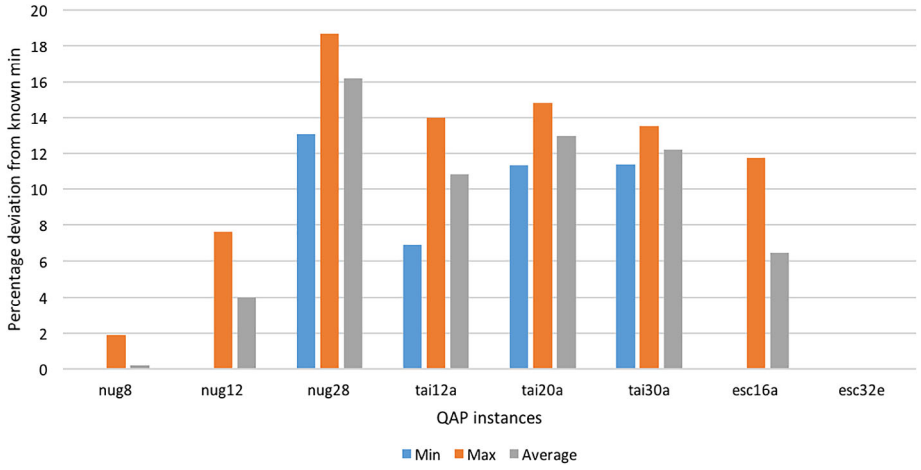


FIGURE 4 Triple bar graph showing percentage deviation of minimum, maximum, and average cost value from known minimum cost value, found by PSO for QAP for several QAP instances. PSO, particle swarm optimization; QAP, quadratic assignment problem

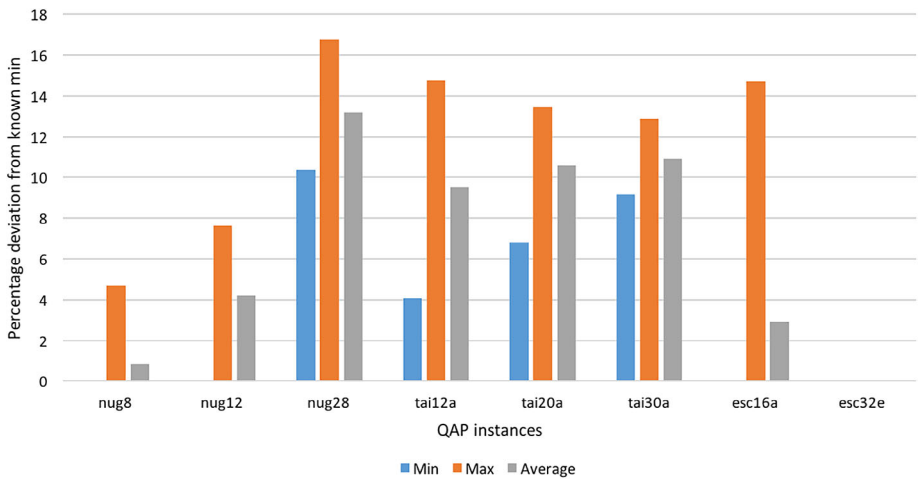


FIGURE 5 Triple bar graph showing percentage deviation of minimum, maximum, and average cost value from known minimum cost value, found by modified PSO for QAP for several QAP instances. PSO, particle swarm optimization; QAP, quadratic assignment problem

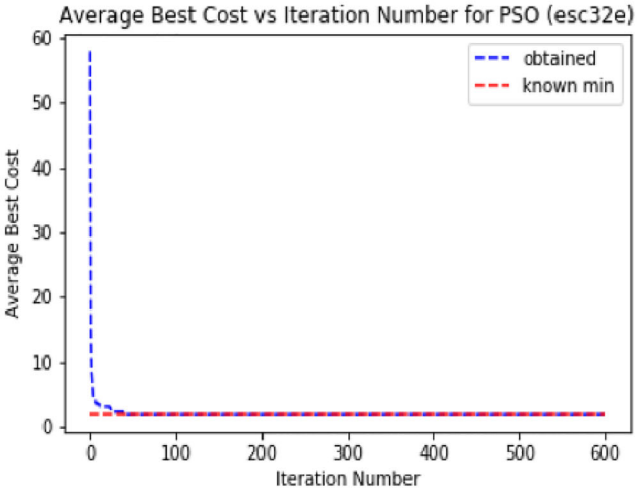
the minimum cost values obtained by PSO for these problem instances. This is best demonstrated by the tai20a QAP instance where PSO obtained a percentage deviation for the minimum cost value of 11.33% while PSO obtained a percentage deviation of the minimum cost value of 6.82%. Furthermore, the percentage deviations for maximum cost values and average cost values obtained by modified PSO were also lower than those obtained by PSO for tai20a, nug28, and tai30a.

For the instance that PSO performed the best on (esc32e, shown in Figure 6(A)), PSO was able to find the optimal solution very quickly in the iteration process. This could be a result of the low complexity of this problem instance.

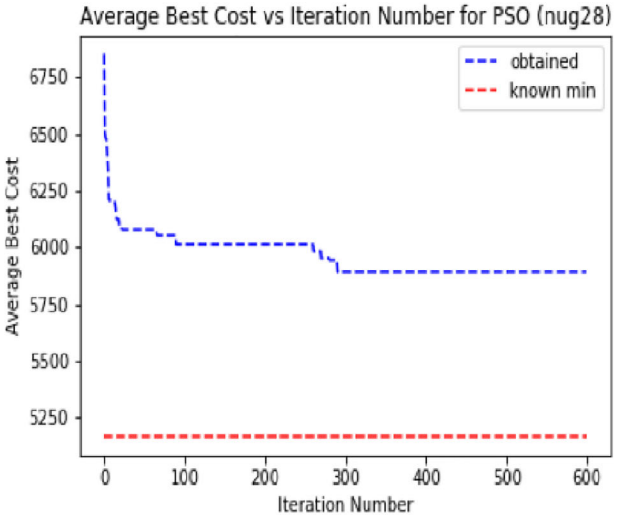
For the instance that PSO performed the worst on (nug28, shown in Figure 6(B)), PSO was, on average, not able to find the optimal solution. PSO was prone to get stuck in local optima. From the stagnation of the average best cost in the late iterations, it can be deduced that not enough better solutions were discovered at these iterations. This is a disadvantage of the PSO algorithm and is likely because of a lack of diversity in the population at these iterations. This means that the "Refresh Gap" strategy used did not adequately address the issue of the lack of diversity in the population at late iterations. It is possible that, even though the positions of the particles were reset, the velocity maintained the trajectory to the local optima.

For the instance that modified PSO performed the best on (esc32e, shown in Figure 7(A) above), modified PSO was able to find the global optimal solution relatively quickly in the iteration process. This could be a result of the low complexity of this problem instance.

For the instance that modified PSO performed the worst on nug28 (shown in Figure 7(B)), it was, on average, not able to discover the global optimal solution. Modified PSO, like PSO, was prone to get stuck in local optima. However, unlike PSO, modified PSO does not suffer from the stagnation of the average best cost in the late iterations. Better solutions were discovered at late iterations. This could be because the velocity of the particles leads the particles to promising local best solutions and not only the one promising global best solution. Leading the particles to promising local best solutions could be what allows diversity in the population at late iterations, and this diversity could be what enables the discovery of better solutions.

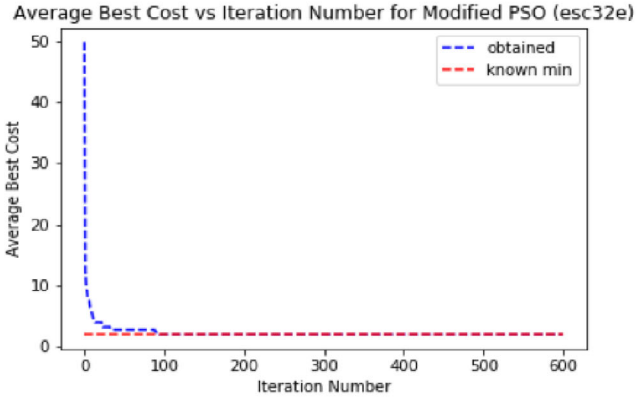


(A) Graph of average cost vs. iteration number on QAP instance where PSO performed the best, which is esc32e.

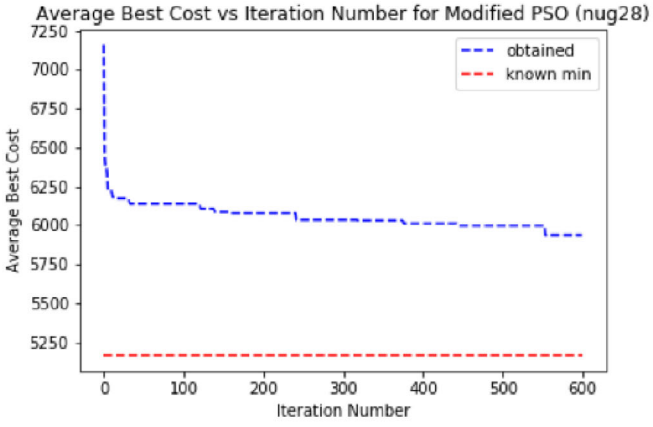


(B) Graph of average cost vs. iteration number on QAP instance where PSO performed the worst, which is nug28.

FIGURE 6 Particle swarm optimization (PSO) performance curves



(A) Graph of average cost vs. iteration number on QAP instance where Modified PSO performed the best, which is esc32e.



(B) Graph of average cost vs. iteration number on QAP instance where Modified PSO performed the worst, which is nug28.

FIGURE 7 Modified particle swarm optimization (PSO) performance curves

5.3.3 | Modified BA

Figure 8 shows the results obtained by the modified BA on the eight problem instances. For QAP instances, with the number of dimensions less than 20, tested on the performance of the modified BA on nug8a was very good as it obtained a percentage deviation on the average of the best value of 0% from the known minimum for nug8a. Performance on nug12 and esc16a were relatively good based on the same metric used to judge performance on nug8a; they were in the range of 2.5%–5%. Performance on tail12a was poor as compared with the rest of the instances with the number of dimensions less than 20, as the algorithm obtained a percentage deviation on the average best value of 9% from the known minimum from of tai12a.

For QAP instances, with the number of dimensions greater than or equal to 20, tested on the performance of the modified BA on esc32e was excellent as it obtained a percentage deviation on the average of the best value of 0% from the known minimum for esc32e, even though this instance used 32 dimensions. Performances on tai20a and tai30a were relatively poor on the same metric used to judge performance on esc32e (they were in the range of 11.5%–12.5%). This could be due to a combination of high complexity and number of dimensions of tai20a and tai30a. Performance on nug28 was extremely poor as compared with the rest of the instances with the number of dimensions greater than or equal to 20, as the algorithm obtained a percentage deviation on the average best value of around 16% from known minimum from of nug28.

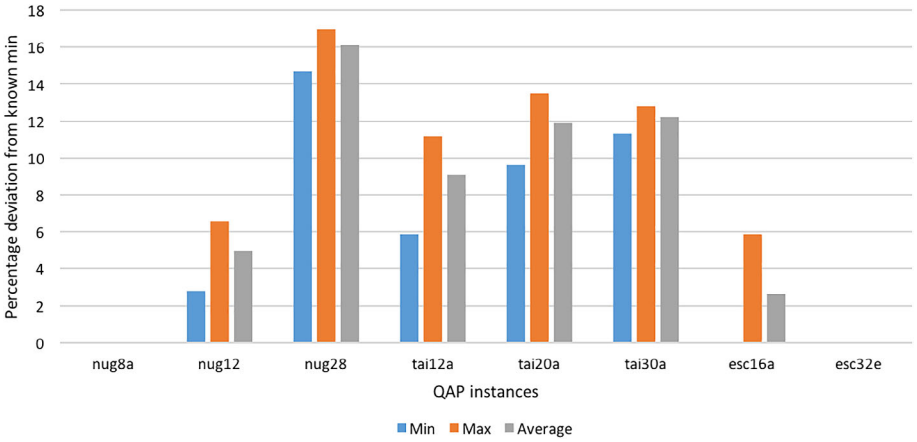
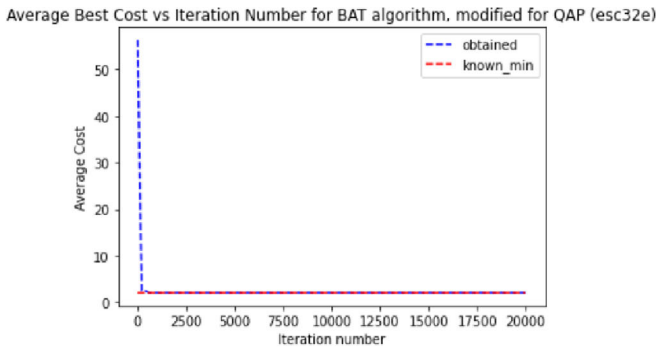
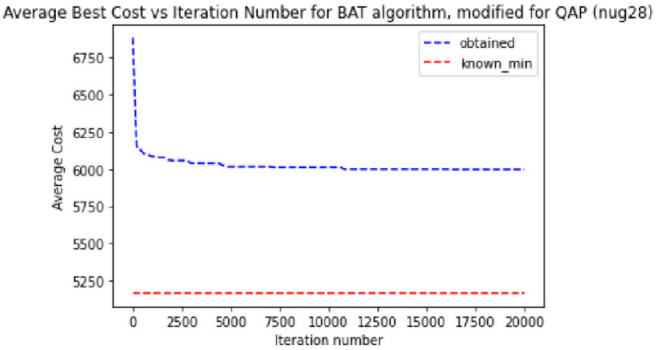


FIGURE 8 Triple bar graph showing percentage deviation of minimum, maximum, and average cost value from known minimum cost value, found by modified BA for QAP for several QAP instances. BA, bat algorithm; QAP, quadratic assignment problem



(A) Graph of average cost vs iteration number on QAP instance where BA performed the best, which is esc32e.



(b) Graph of average cost vs. iteration number on QAP instance where BA performed the best, which is nug28.

FIGURE 9 Modified bat algorithm (BA) performance curves

The modified BA performed the best on the esc32e instance as the average best cost line obtained merged with the best-known minimum line quickly (shown in Figure 9(A)). esc32e appears to be a simple QAP instance, hence the results. We see somewhat of the same results with the other algorithms tested.

By looking at Figure 9(B), the average best cost obtained versus iteration number starts to decrease early (favorable), but then reached a plateau a significantly great distance away from the best-known minimum line. There was no further improvement after the plateau was reached. This could be due to the high complexity of nug28.

5.3.4 | Tabu search

Figure 10 shows the results obtained by TS on the eight problem instances.

For the QAP instances nug8, tai12, esc16e, esc32e, TS was able to obtain the optimal solutions. With relatively low average percentage deviations (<3% for nug8, <6% for nug12, <2% for esc16e, and 0% for esc32e). TS performed the best on esc32. Furthermore, the instance TS performed the worst on nug12 and tai12 with the highest average deviation of 6% and 7.63%, respectively. Other instances where the number of dimensions was greater than or equal to 20, that is (nug28, tai20a, tai30a) although the optimal solution was not found they still had relatively low average percentage deviation, even smaller than that of instances where the number of dimensions is less than 20 (4% for nug28, 5% for tai20a, and 5% for tai30a). Thus, showing that TS performed reasonably well for these QAP instances.

TS performed the best on the esc32e instance (performance is shown in Figure 11(A) above) and was able to find the optimal solution very quickly in the iteration process. This could be as a result of the low complexity of this problem instance. This shows that TS has the potential to obtain promising results despite the dimension of the problem, so long as the dimension of the instance is low.

For the instance that the TS performed the worst on (nug12 and tai12a, see performance of TS on nug12 in Figure 11(B)), this could be a result of the higher complexity of the problems instances. It can be seen on the graph that it was not able to find the optimal solution and was prone to get

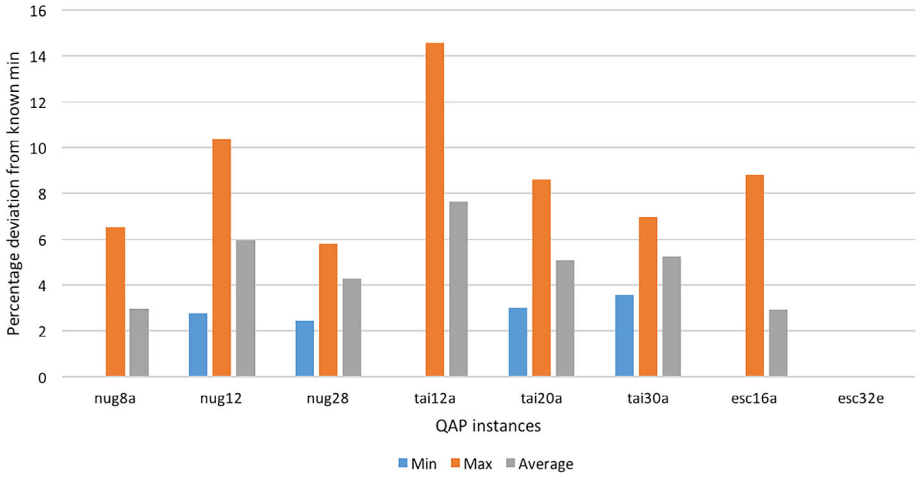
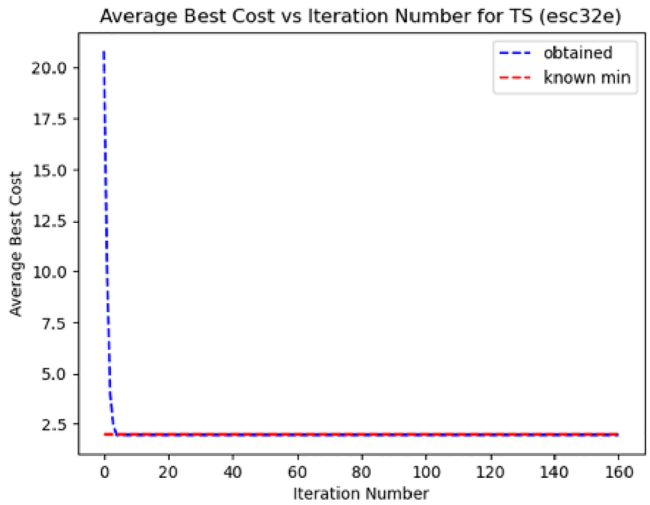
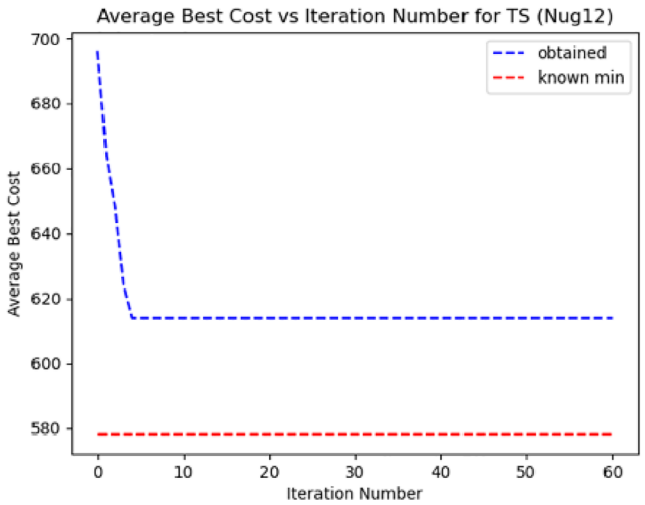


FIGURE 10 Triple bar graph showing percentage deviation of minimum, maximum, and average cost value from known minimum cost value, found by TS for several QAP instances. QAP, quadratic assignment problem; TS, tabu search



(A) Graph of average cost vs. iteration number on the instance where TS performed the best, which is esc32e.



(B) Graph of average cost vs iteration number on QAP instance

FIGURE 11 Tabu search performance curves

stuck in local optima. From the stagnation of the average best cost in the later iterations, showing that not enough better solutions were discovered or explored at these iterations. Thus, it can be seen that more diversification methods are needed. Thereby allowing the search to explore other parts of the solution search space (to try find a global optimum). This could be done using a frequency-based memory, applying penalties to solutions that are not improving also by adding other diversification techniques such as mutations presented in Reference 46 to enhance the search leading to more optimal results.

5.3.5 | Genetic algorithm

Figure 12 shows the results obtained by the GA on the eight problem instances. It should be noted that the percentage deviation of the maximum value obtained by the GA for esc32e is shown to be 100% when the actual percentage deviation is 31.10%. This is done to maintain the legibility of the graph. The GA performed best for the nug8a dataset, obtaining an average percentage deviation of 0.42%. The algorithm struggled with larger instances tai20a, tai30a, and esc32e, obtaining an average standard deviation percentage of 13.30%, 13.22%, and 17.5%, respectively. It never found the best solution for tai20a and tai30a. The algorithm was, on average able to obtain the best solution for esc32e.

The GA performed the best on the nug8a QAP instance (performance is shown in Figure 13(A)). The slope of the average cost line obtained was very steep over the first few iterations and began to even move out towards the end, merging with the best-known minimum line. The GA performed the worst on the tai20a QAP instance (performance is shown in Figure 13(B)). The slope of the average cost obtained line was very steep over the

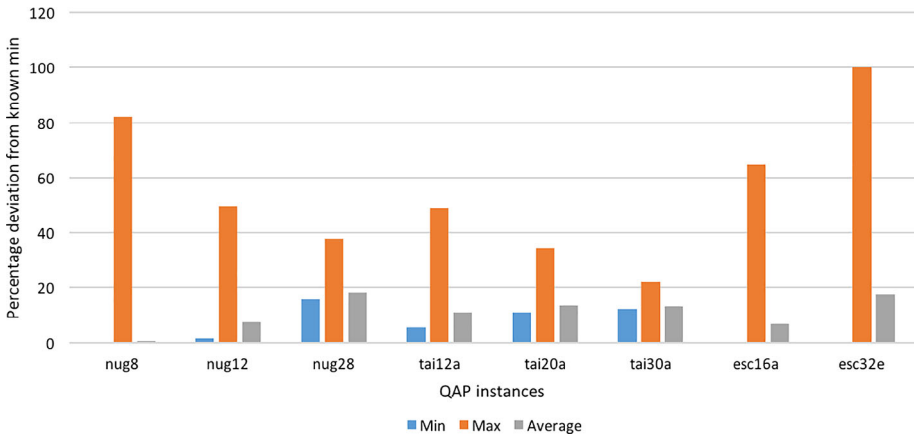
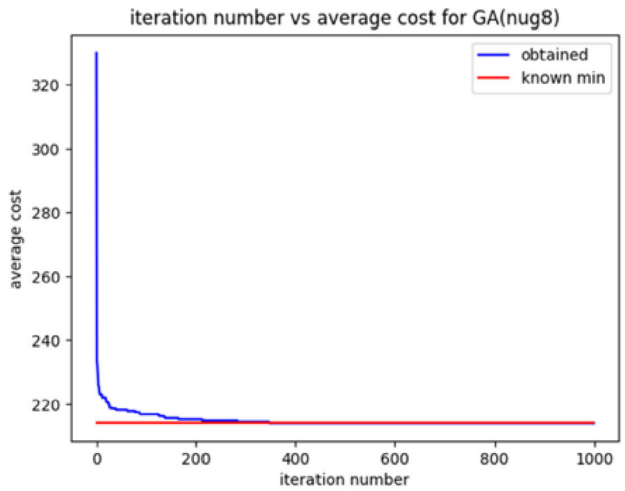
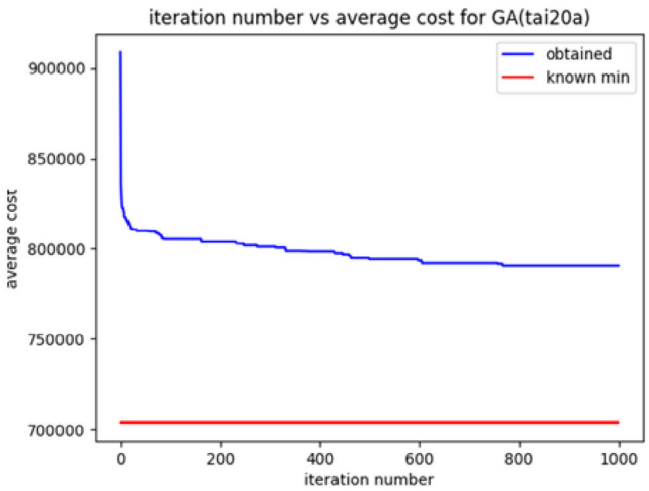


FIGURE 12 Triple bar graph showing percentage deviation of minimum, maximum, and average cost value from known minimum cost value, found by GA for several QAP instances. GA, genetic algorithm; QAP, quadratic assignment problem



(A) Graph of average cost vs iteration number on QAP instance where GA algorithm performed the best, which is nug8a.



(B) Graph of average cost vs iteration number on QAP instance where the GA algorithm performed the worst, which is tai20a.

FIGURE 13 Genetic algorithm (GA) performance curves

first few iterations and began to even move out towards the end, however the distance between the average cost line obtained and the best-known minimum line is very great towards the end of the iterations.

5.3.6 | Ranking of the algorithms

Based on the results displayed in Figure 14, it can be seen that for the problem instances with $n < 20$, all the algorithms perform relatively well. The highest percentage deviation of the average value from the known minimum for an instance with $n < 20$ is obtained by the GA for tai12a: (11.11%). It should be noted that all the algorithms performed better on esc16a than tai12a and this shows that the complexity of the problem instance may influence the performance of the algorithm more than the size of the problem instance. For $n >= 20$, it should be noted that most of the algorithms performed worse than they did for $n < 20$. However, all the algorithms performed very well for esc32e as all algorithms achieved a percentage deviation of the average value from the known minimum of 0%. This could again be attributed to the low complexity of the problem instance. Only GA did not perform well on this instance and actually achieved one of the highest percentage deviations of the average value from the known minimum. Furthermore, based on these results, it can be seen that the ACO outperforms all the other algorithms across all instances. In the same manner, the GA mostly yields the worst results.

The algorithms are ranked according to the following procedure: We constructed the following Tables 9 and 10 showing QAP instances tested, the selected algorithms for this study, and the percentage deviation of the average value of best solution found, produced by algorithms, from the known minimum of QAP instance.

FIGURE 14 Bar chart showing the percentage deviation of the average of best cost value found from known minimum cost value for each of the algorithms tested

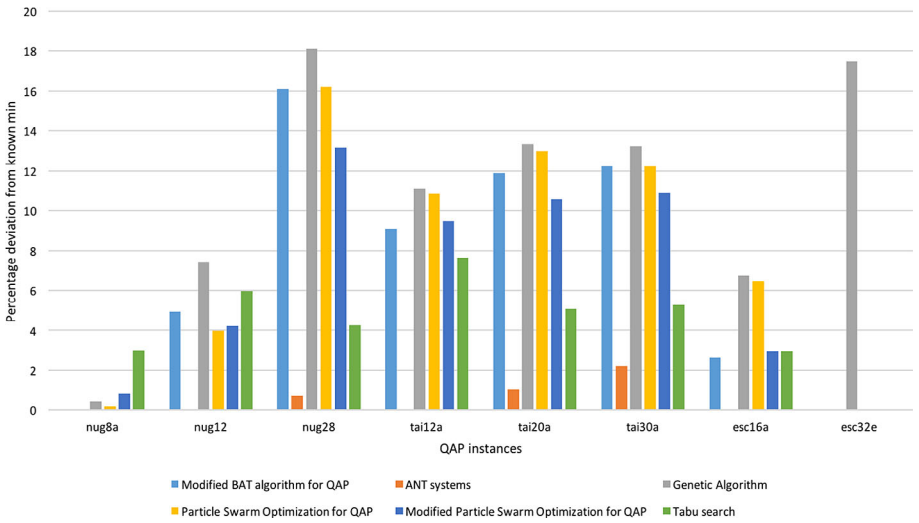


TABLE 9 The percentage deviation of the average value of best solution found from the known minimum of various QAP instances for the tested algorithms

Instance	ACO	BA	GA	PSO	Modified PSO	TS
nug8a	0.00	0.00	0.42	0.19	0.84	3.00
nug12	0.00	4.95	7.44	3.98	4.22	5.96
tai12a	0.00	9.07	11.11	10.86	9.49	7.63
esc16a	0.00	2.65	6.76	6.47	2.94	2.94
tai20a	1.02	11.90	13.36	13.00	10.58	5.07
nug28	0.74	16.10	18.11	16.21	13.16	4.26
tai30a	2.22	12.24	13.22	12.21	10.91	5.27
esc32e	0.00	0.00	17.50	0.00	0.00	0.00

Abbreviations: ACO, ant colony optimization; BA, bat algorithm; GA, genetic algorithm; PSO, particle swarm optimization; QAP, quadratic assignment problem; TS, tabu search.

TABLE 10 Illustration of the percentage deviation of average values of best solution found from the known minimum of various QAP instances

Instance	ACO	BA	GA	PSO	Modified PSO	TS
nug8a	0.00	0.00	0.42	0.19	0.84	3.00
nug12	0.00	4.95	7.44	3.98	4.22	5.96
tai12a	0.00	9.07	11.11	10.86	9.49	7.63
esc16a	0.00	2.65	6.76	6.47	2.94	2.94
tai20a	1.02	11.90	13.36	13.00	10.58	5.07
nug28	0.74	16.10	18.11	16.21	13.16	4.26
tai30a	2.22	12.24	13.22	12.21	10.91	5.27
esc32e	0.00	0.00	17.50	0.00	0.00	0.00

Abbreviations: ACO, ant colony optimization; BA, bat algorithm; GA, genetic algorithm; PSO, particle swarm optimization; QAP, quadratic assignment problem; TS, tabu search.

One may assume that a higher number of dimensions of the QAP problem may result in worse performance for the algorithms, but this is not the case. For example, TS performs better in esc32e (32-dimensions) than in nug8a (eight-dimensions). Thus, given the afore-mentioned statement, we assign a uniform weighting to the QAP instances used for scoring the algorithms to rank them.

We move row by row, assigning scores to each algorithm per row based on how well they performed on a QAP instance of a specific row relative to the other algorithms. Essentially, we are obtaining an overall rank of the algorithms tested by aggregating their ranks for each QAP instance used. Green is a score of 6, blue is a score of 5, yellow is a score of 4, purple is a score of 3, orange is a score of 2, and red is a score of 1. Table 10 illustrates

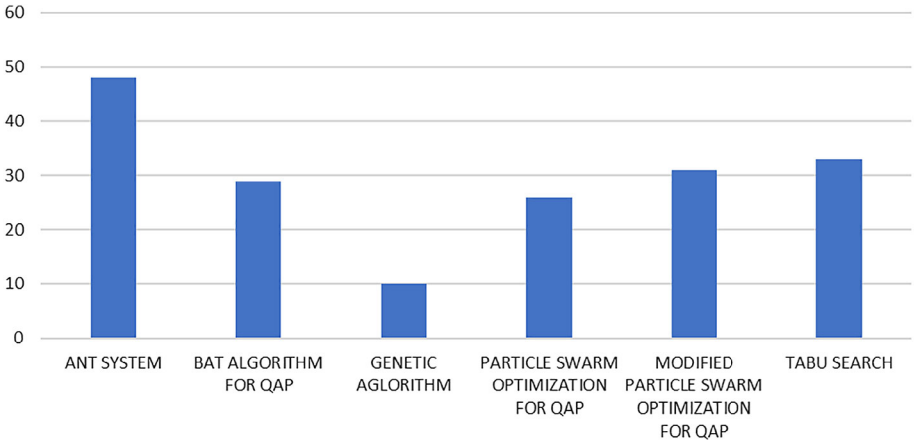


FIGURE 15 Total score obtained for each algorithm based on ranking from Table 10

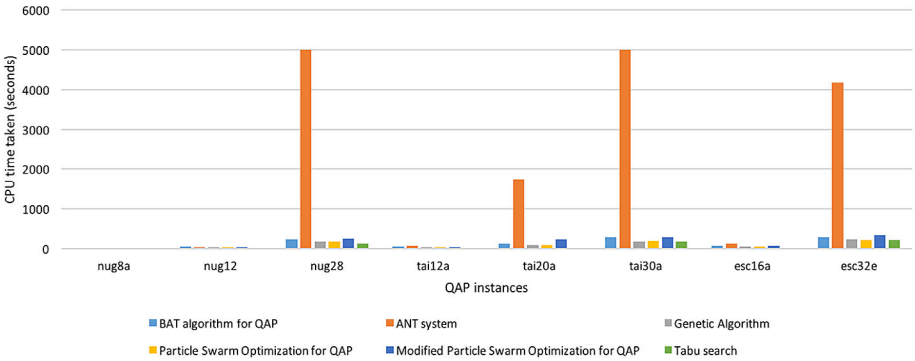


FIGURE 16 Average CPU time consumed by the individual algorithm on each quadratic assignment problem instance

the percentage deviation of the average value of best solutions found by the algorithms from the known minimum of various QAP instances, as well as colored cells indicating the algorithms performance rank position relative to the executed QAP instances.

In Figure 15, the total score obtained by each algorithm from Table 10 is plotted; this makes ranking the algorithms easier.

The ranks of the algorithms are as follows:

1. ACO
2. TS
3. Modified PSO
4. BA
5. PSO
6. GA

The algorithms are ranked in descending order as shown from the list above, that is, from 1 to 6 (in that precedence, 1 is ranked above 2 and 2 above 3, and 6 being the least value). Therefore, from the ranks, we can conclude that ACO performed the best, however GA performed the worst.

Figure 16 shows the average CPU time taken by each algorithm on each instance. Note that in this chart, the CPU Time taken for ACO on nug28 is shown as 5000 s but the actual value is 6494.5 s. The CPU time taken for ACO on esc32e is shown to be 5000 s, but the actual CPU time taken is 21,704.75. The actual values have not been represented to maintain the legibility of the graph.

We see that for BA, GA, PSO, modified PSO, and TS, the average CPU time required scales with the number of dimensions of the QAP instance. However, ANT systems differs from this trend, being that it is scaling not only with the number of dimensions but with the complexity of the instance as well. For example, ANT systems had its best performance on esc32e and its worst performance on tai30a, yet CPU time required for esc32e is 4186.64 s whilst CPU time required for tai30a is 21,704.75 s. A full comparison of CPU time between the meta-heuristic algorithms tested could not be done as their number of iterations differed.

6 | CONCLUSION

Presented in this article is a comparative study of six meta-heuristic algorithms for solving the QAP. Each of the implemented algorithm was tested on eight QAP benchmark instances taken from the QAPLIB. These instances have varying dimensions and complexity. The six meta-heuristic

algorithms studied in this article include the modified BA for QAP from Reference 38, GA from Reference 63, ACO from Reference 34, TS from Reference 8, PSO for QAP,[19] and modified PSO for QAP. Among all the results obtained, we deemed the following fit for comparative uses: percentage deviation of the average value of the best solution from the known minimum of QAP instance and average CPU time consumed by the algorithms in finding the best solutions. From the results obtained, we concluded that the ACO was the best performed algorithm, and that the GA was the worst-performed algorithm. It should be noted that even though the ACO algorithm does in some cases require significantly higher CPU times than the other algorithms, other algorithms reach Plateaus from which they do not escape local optima during their runtimes. Although ACO took longer CPU time, however with less than 120 iterations, it was able to get close to the known minimum of the QAP instances tested on. This reinforces that the ACO is the best performed algorithm when compared with the other algorithms tested.

With regards to CPU time, all algorithms tested, apart from ACO, followed the trend of requiring more CPU time as the dimensions of the QAP instance increases. The efficiency of the ACO algorithm seems to depend on both the number of dimensions and complexity of the QAP instance, but more so on the complexity of the QAP instance, as huge jumps in CPU time required by the ACO were recorded for the QAP instances that were estimated with some degree of confidence to be very complex. For real-world usage, an algorithm that performs well and that is time-efficient would be required. Thus, we conclude that even though ACO is best performing theoretically, the TS (second-best theoretically) is the best for real-world usage as it gets good results and scales with the number of dimensions of the QAP instance only.

Future research lies in running experiments with various hybrids, adaptive, and parallel versions of the tested and new meta-heuristic algorithms.[71] Similarly, one can also run the experiments using complex datasets with large graph size to explore if the performance of the ACO with respect to the other algorithms remains consistent. Furthermore, it would be interesting to investigate the performance of the six tested algorithms in other related assignment problems. Therefore, in this regard, it is necessary also to study how different parameter configurations impact the performance of the algorithms, respectively.

DECLARATION OF COMPETING INTEREST

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this article.

ORCID

Absalom E. Ezugwu https://orcid.org/0000-0002-3721-3400

REFERENCES

1. Koopmans TC, Beckmann M. Assignment problems and the location of economic activities. *Econometr J Econometr Soc*. 1957;53–76.
2. Loiola EM, de Abreu NMM, Boaventura-Netto PO, Hahn P, Querido T. A survey for the quadratic assignment problem. *Eur J Oper Res*. 2007;176(2):657-690.
3. Burkard RE, Karisch SE, Rendl F. Qaplib–a quadratic assignment problem library. *J Glob Optim*. 1997;10(4):391-403.
4. Christofides N, Benavent E. An exact algorithm for the quadratic assignment problem on a tree. *Oper Res*. 1989;37(5):760-768.
5. Lawler EL. The quadratic assignment problem. *Manag Sci*. 1963;9(4):586-599.
6. Pardalos PM, Crouse JV. A parallel algorithm for the quadratic assignment problem. Paper presented at: Proceedings of the 1989 ACM/IEEE Conference on Supercomputing Supercomputing'89, New York; 1989:351-360; IEEE.
7. Bazaraa MS, Sherali HD. Benders' partitioning scheme applied to a new formulation of the quadratic assignment problem. *Naval Res Logist Quart*. 1980;27(1):29-41.
8. Skorin-Kapov J. Tabu search applied to the quadratic assignment problem. *ORSA J Comput*. 1990;2(1):33-45.
9. Skorin-Kapov J. Extensions of a tabu search adaptation to the quadratic assignment problem. *Comput Oper Res*. 1994;21(8):855-865.
10. Nissen V. Solving the quadratic assignment problem with clues from nature. *IEEE Trans Neural Netw*. 1994;5(1):66-72.
11. Bui TN, Moon BR. A genetic algorithm for a special class of the quadratic assignment problem. *DIMACS Ser Discr Math Theoret Comput Sci*. 1994;16:99-116.
12. Ahmed ZH. A genetic algorithm for a special class of the quadratic assignment problem. *Opsearch*. 2015;52(4):714-732.
13. Ahuja RK, Orlin JB, Tiwari A. A greedy genetic algorithm for the quadratic assignment problem. *Comput Operat Res*. 2000;27(10):917-934.
14. Drezner Z. A new genetic algorithm for the quadratic assignment problem. *INFORMS J Comput*. 2003;15(3):320-330.
15. Stützle T, Dorigo M. ACO algorithms for the quadratic assignment problem. *New Ideas Optim*. McGraw-Hill Ltd., UK; 1999;(C50).
16. Ariyasingha IDID, Fernando TGI. A new multi-objective ant colony optimisation algorithm for solving the quadratic assignment problem. *Vidyodaya J Sci*. 2019;22(1):1–11.
17. Oliveira S, Hussin MS, Roli A, Dorigo M, Stützle T. Analysis of the population-based ant colony optimization algorithm for the TSP and the QAP. Paper presented at: Proceedings of the 2017 IEEE Congress on Evolutionary Computation (CEC), Donostia, Spain; 2017:1734-1741; IEEE.
18. Peras M, Ivkovic N. Channel assignment with ant colony optimization. *Swarm, Evolutionary, and Memetic Computing and Fuzzy and Neural Computing*. New York, NY: Springer; 2019:31-42.
19. Mamaghani AS, Meybodi MR. Solving the quadratic assignment problem with the modified hybrid PSO algorithm. Paper presented at: Proceedings of the 2012 6th International Conference on Application of Information and Communication Technologies (AICT), Tbilisi, Georgia; 2012:1-6; IEEE.
20. Liu H, Abraham A, Zhang J. A particle swarm approach to quadratic assignment problems. *Soft Computing in Industrial Applications*. New York, NY: Springer; 2007:213-222.
21. Hafiz F, Abdennour A. Particle swarm algorithm variants for the quadratic assignment problems-a probabilistic learning approach. *Expert Syst Appl*. 2016;44:413-431.

22. Pradeepmon T, Sridharan R, Panicker V. Development of modified discrete particle swarm optimization algorithm for quadratic assignment problems. *Int J Ind Eng Comput*. 2018;9(4):491-508.
23. Abdel-Basset M, Rashad H, Zhou Y. Solving quadratic assignment problem by symbiotic organisms search algorithm. *Int J Intell Enterprise*. 2019;6(1):77-91.
24. Stützle T. Iterated local search for the quadratic assignment problem. *Eur J Oper Res*. 2006;174(3):1519-1539.
25. Aksan Y, Dokeroglu T, Cosar A. A stagnation-aware cooperative parallel breakout local search algorithm for the quadratic assignment problem. *Comput Ind Eng*. 2017;103:105-115.
26. Riffi ME, Saji Y, Barkatou M. Incorporating a modified uniform crossover and 2-exchange neighborhood mechanism in a discrete bat algorithm to solve the quadratic assignment problem. *Egypt Inform J*. 2017;18(3):221-232.
27. Munien C, Mahabeer S, Dzitiro E, Singh S, Zungu S, Ezugwu AE. Metaheuristic approaches for one-dimensional bin packing problem: a comparative performance study. *IEEE Access*. 2020;8:227438–227465.
28. Dokeroglu T, Sevinc E, Kucukyilmaz T, Cosar A. A survey on new generation metaheuristic algorithms. *Comput Ind Eng*. 2019;137:106040.
29. Ezugwu A.E, Shukla A.K, Nath R. et al. Metaheuristics: a comprehensive overview and classification along with bibliometric analysis. *Artif Intell Rev*. 2021;1-56. https://doi.org/10.1007/s10462-020-09952-0.
30. Ezugwu AE, Shukla AK, Nath R, Akinyelu AA, Agushaka JO, Chiroma H, Muhuri PK. Metaheuristics: a comprehensive overview and classification along with bibliometric analysis. *Artif Intell Rev*. 2021. https://doi.org/10.1007/s10462-020-09952-0.
31. Aktel A, Yagmahan B, Özcan T, Yenisey MM, Sansarcı E. The comparison of the metaheuristic algorithms performances on airport gate assignment problem. *Transp Res Proc*. 2017;22:469-478.
32. Xia X, Zhou Y. Performance analysis of aco on the quadratic assignment problem. *Chin J Electron*. 2018;27(1):26-34.
33. Li Y, Pardalos PM, Ramakrishnan KG, Resende MGC. Lower bounds for the quadratic assignment problem. *Ann Oper Res*. 1994;50(1):387-410.
34. Maniezzo V, Colorni A. The ant system applied to the quadratic assignment problem. *IEEE Trans Knowl Data Eng*. 1999;11(5):769-778.
35. Mouhoub M, Wang Z. Improving the ant colony optimization algorithm for the quadratic assignment problem. Paper presented at: Proceedings of the 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), Hong Kong; 2008:250-257; IEEE.
36. Yang X-S. *A New Metaheuristic Bat-Inspired Algorithm*. Vol 284. Berlin/Heidelberg, Germany: Springer; 2010:65-74.
37. Krause J, Cordeiro J, Parpinelli RS, Lopes HS. A survey of swarm algorithms applied to discrete optimization problems. *Swarm Intelligence and Bio-Inspired Computation*. Elsevier; 2013:169-191. https://doi.org/10.1016/C2012-0-02754-8.
38. Shukla A. A modified bat algorithm for the quadratic assignment problem. Paper presented at: Proceedings of the 2015 IEEE Congress on Evolutionary Computation (CEC), Sendai, Japan; May 2015:486-490; IEEE.
39. Azarbonyad H, Babazadeh R. A genetic algorithm for solving quadratic assignment problem (qap); 2014. arXiv preprint arXiv:1405.5050.
40. Kennedy J, Eberhart RC. A discrete binary version of the particle swarm algorithm. Paper presented at: Proceedings of the 1997 IEEE International Conference on Systems, Man, and cybernetics. Computational Cybernetics and Simulation, Orlando, FL; vol 5, 1997:4104-4108; IEEE.
41. Liu B, Wang L, Jin Y-H. An effective PSO-based memetic algorithm for flow shop scheduling. *IEEE Trans Syst Man Cybern B (Cybern)*. 2007;37(1):18-27.
42. Clerc M. Discrete particle swarm optimization, illustrated by the traveling salesman problem. *New Optimization Techniques in Engineering*. New York, NY: Springer; 2004:219-239.
43. Neethling M, Engelbrecht AP. Determining RNA secondary structure using set-based particle swarm optimization. Paper presented at: Proceedings of the 2006 IEEE International Conference on Evolutionary Computation, Vancouver, BC, Canada; 2006:1670-1677; IEEE.
44. Chen W-N, Zhang J, Chung HSH, Zhong W-L, Wu W-G, Shi Y-H. A novel set-based particle swarm optimization method for discrete optimization problems. *IEEE Trans Evolut Comput*. 2009;14(2):278-300.
45. Taillard É. Robust taboo search for the quadratic assignment problem. *Parallel Comput*. 1991;17(4-5):443-455.
46. Misevicius A. A tabu search algorithm for the quadratic assignment problem. *Comput Optim Appl*. 2005;30(1):95-111.
47. Abd G, Abeer M, El-Sayed M. A comparative study of meta-heuristic algorithms for solving quadratic assignment problem. *Int J Adv Comput Sci Appl*. 2014;5(1):1–6.
48. Ezugwu AE, Adeleke OJ, Akinyelu AA, Viriri S. A conceptual comparison of several metaheuristic algorithms on continuous optimisation problems. *Neural Comput Appl*. 2020;32(10):6207-6251.
49. Zhang H, Liu F, Zhou Y, Zhang Z. A hybrid method integrating an elite genetic algorithm with tabu search for the quadratic assignment problem. *Inf Sci*. 2020;539:347-374.
50. Abdel-Baset M, Wu H, Zhou Y, Abdel-Fatah L. Elite opposition-flower pollination algorithm for quadratic assignment problem. *J Intell Fuzzy Syst*. 2017;33(2):901-911.
51. Dokeroglu T, Sevinc E, Cosar A. Artificial bee colony optimization for the quadratic assignment problem. *Appl Soft Comput*. 2019;76:595-606.
52. Abdel-Basset M, Manogaran G, El-Shahat D, Mirjalili S. Integrating the whale algorithm with tabu search for quadratic assignment problem: a new approach for locating hospital departments. *Appl Soft Comput*. 2018;73:530-546.
53. Benlic U, Hao J-K. Memetic search for the quadratic assignment problem. *Expert Syst Appl*. 2015;42(1):584-595.
54. Lalla-Ruiz E, Exposito-Izquierdo C, Melián-Batista B, Moreno-Vega JM. A hybrid biased random key genetic algorithm for the quadratic assignment problem. *Inf Process Lett*. 2016;116(8):513-520.
55. Dokeroglu T. Hybrid teaching–learning-based optimization algorithms for the quadratic assignment problem. *Comput Ind Eng*. 2015;85:86-101.
56. Chmiel W, Kwiecień J. Quantum-inspired evolutionary approach for the quadratic assignment problem. *Entropy*. 2018;20(10):781.
57. Kılıç H, Yüzgeç U. Tournament selection based antlion optimization algorithm for solving quadratic assignment problem. *Eng Sci Technol Int J*. 2019;22(2):673-691.
58. Guo M-W, Wang J-S, Xue Y. An chaotic firefly algorithm to solve quadratic assignment problem. *Eng Lett*. 2020;28(2):337-342.
59. McKendall A, Li C. A tabu search heuristic for a generalized quadratic assignment problem. *J Ind Product Eng*. 2017;34(3):221-231.
60. Gambardella LM, Taillard ÉD, Dorigo M. Ant colonies for the quadratic assignment problem. *J Operat Res Soc*. 1999;50(2):167-176.
61. Nakamura RYM, Pereira LAM, Costa KA, Rodrigues D, Papa JP, Yang XS. Bba: a binary bat algorithm for feature selection. Paper presented at: Proceedings of the 2012 25th SIBGRAPI conference on graphics, Patterns and Images, Ouro Preto, Brazil; 2012:291-297; IEEE.
62. Owens AD. Charles Darwin and the theory of natural selection. *Sci Scope*. 2015;39(2):89.
63. Melanie Mitchell. *An Introduction to Genetic Algorithms*, Complex Adaptive Series, Elsevier; 1996.

64. Z Wu and A Simpson. *An Efficient Genetic Algorithm Paradigm for Discrete Optimisation of Pipeline Networks*. 1997.
65. Parsopoulos KE, Vrahatis MN. Unified particle swarm optimization for solving constrained engineering optimization problems. Paper presented at: Proceedings of the International Conference on Natural Computation; 2005:582-591; Springer, New York, NY.
66. Blackwell T, Branke J. Multi-swarm optimization in dynamic environments. Paper presented at: Proceedings of the Workshops on Applications of Evolutionary Computation; 2004:489-500; Springer, New York, NY.
67. Pirim H, Bayraktar E, Eksioglu B. Tabu search: a comparative study; 2008.
68. Glover F. Tabu search—Part I. *ORSA J Comput*. 1989;1(3):190-206.
69. Boussaïd I, Lepagnot J, Siarry P. A survey on optimization metaheuristics. *Inf Sci*. 2013;237:82-117.
70. James T, Rego C, Glover F. Multistart tabu search and diversification strategies for the quadratic assignment problem. *IEEE Trans Syst Man Cybern A Syst Humans*. 2009;39(3):579-596.
71. Tosun U, Dokeroglu T, Cosar A. A robust island parallel genetic algorithm for the quadratic assignment problem. *Int J Prod Res*. 2013;51(14):4117-4133.

How to cite this article: Achary T, Pillay S, Pillai SM, Mqadi M, Genders E, Ezugwu AE. A performance study of meta-heuristic approaches for quadratic assignment problem. *Concurrency Computat Pract Exper*. 2021;33:e6321. https://doi.org/10.1002/cpe.6321

APPENDIX A

This Appendix shows the raw simulation result data obtained from the experiments (Figures A1–A7).

Ant System							
	Solution Values				Percentage Deviation from Best Known		
Dataset	Best Known	Min	Max	Average	Min	Max	Average
nug8a	214	214	214	214	0	0	0
nug12	578	578	578	578	0	0	0
nug28	5166	5178	5224	5204,4	0,232288037	1,122725513	0,743321719
esc16a	68	68	68	68	0	0	0
esc32e	2	2	2	2	0	0	0
tai12a	224416	224416	224416	224416	0	0	0
tai20a	703482	705622	718178	710682,4	0,304201103	2,089037104	1,023537205
tai30a	1818146	1853836	1863492	1858509,1	1,962988671	2,494079133	2,220014234

FIGURE A1 Table showing the results obtained by the ANT system

BAT Algorithm							
	Solution Values		Percentage Deviation from known Min				
Dataset	Best Known Value	Min	Max	Average	Min	Max	Average
nug8a	214	214	214	214	0	0	0
nug12	578	594	616	606,6	2,76816609	6,574394464	4,948096886
nug28	5166	5924	6042	5997,8	14,67286101	16,95702671	16,10143244
tai12a	224416	237560	249494	244777,2	5,856979894	11,17478255	9,072971624
tai20a	703482	771094	798288	787173,4	9,611049039	13,47667744	11,89673652
tai30a	1818146	2024320	2050346	2040620	11,33979339	12,77125159	12,23631106
esc16a	68	68	72	69,8	0	5,882352941	2,647058824
esc32e	2	2	2	2	0	0	0

FIGURE A2 Table showing the results obtained by the bat system

Genetic Algorithm							
	Solution Values				Percentage Deviation from Known Min (%)		
Dataset	Best Known Value	Min	Max	Average	Min	Max	Average
nug8	214	214	390	214,9	0	82,24	0,42
nug12	578	586	864	621	1,38	49,48	7,44
nug28	5166	5986	7116	6101,99	15,87	37,74	18,11
tai12a	224416	237292	334392	249346,87	5,74	49	11,11
tai20a	703482	781742	946316	797434,73	11,12	34,52	13,36
tai30a	1818146	2039546	2221064	2058572,69	12,18	22,16	13,22
esc16a	68	68	122	72,6	0	64,7	6,76
esc32e	2	2	64	2,35	0	3110	17,5

FIGURE A3 Table showing the results obtained by the genetic algorithm

Particle Swarm Optimization							
	Solution Values				Percentage Deviation from Known Min (%)		
Dataset	Best Known Value	Min	Max	Average	Min	Max	Average
nug8	214	214	218	214.4	0.0	1.8691588785046727	0.18691588785046995
nug12	578	578	622	601.0	0.0	7.612456747404845	3.9792387543252596
nug28	5166	5842.0	6132.0	6003.4	13.085559427022842	18.69918699186992	16.209833526906692
tai12a	224416	239924	255824	248785.6	6.910380721517183	13.995437045486952	10.85911877940967
tai20a	703482	783182	807756	794797.4	11.329358817993922	14.82255409520073	12.980488484424624
tai30a	1818146	2024702.0	2063802.0	2040217.6	11.360803807835014	13.511346173519618	12.214178619318805
esc16a	68	68	76	72.4	0.0	11.76470588235294	6.470588235294127
esc32e	2	2.0	2.0	2.0	0.0	0.0	0.0

FIGURE A4 Table showing the results obtained by particle swarm optimization

Modified Particle Swarm Optimization							
	Solution Values				Percentage Deviation from Known Min (%)		
Dataset	Best Known Value	Min	Max	Average	Min	Max	Average
nug8	214	214	224	215.8	0.0	4.672897196261682	0.841121495327108
nug12	578	578	622	602.4	0.0	7.612456747404845	4.221453287197228
nug28	5166	5700	6032	5853.0	10.336817653890824	16.7634533488192	13,15911731
tai12a	224416	233524	257512	245711.8	4.058534150862684	14.747611578497077	9.489430343647506
tai20a	703482	751464	798014	777900.2	6.8206435985568925	13.437728328514448	10.578550694971579
tai30a	1818146	1984546	2052242	2016425.2	9.15218029795187	12.875533648012865	10.90557083974554
esc16a	68	68	78	70.0	0.0	14.705882352941178	2.941176470588235
esc32e	2	2	2	2.0	0.0	0.0	0.0

FIGURE A5 Table showing the results obtained by the modified particle swarm optimization

Tabu Search							
	Solution Values				Percentage Deviation from known Min		
Dataset	Best Known Value	Min	Max	Average	Min	Max	Average
nug8a	214	214	228	220,4	0	6,542056075	2,990654206
nug12	578	594	638	612,4	2,76816609	10,38062284	5,951557093
nug28	5166	5292	5466	5386,2	2,43902439	5,807200929	4,262485482
tai12a	224416	224416	257124	241541,2	0	14,57471838	7,631006702
tai20a	703482	724558	764178	739163,2	2,99595441	8,627939308	5,072084289
tai30a	1818146	1883148	1944906	1913968	3,575180431	6,971937347	5,270313825
esc16a	68	68	74	70	0	8,823529412	2,941176471
esc32e	2	2	2	2	0	0	0

FIGURE A6 Table showing the results obtained by tabu search

Dataset	BAT algorithm for QAP	ANT systems	Genetic algorithm	Particle Swarm Optimization for QAP	Modified Particle Swarm Optimization for QAP	Tabu search
nug8a	28.25	4.45	17	14	19.2	0.24
nug12	51.75	45.4	36	30	42.85	1.7
nug28	242	6494.5	175.5	175.5	257.5	118.7
esc16a	79	122.86	61	53.5	77	7.1
esc32e	300	4186.64	240	235.5	346	230.9
tai12a	53.25	17.01	38.6	30.75	42.75	1.83
tai20a	130	1748.5	88	85	249	21.7
tai30a	291.25	21704.75	186	205	299.5	167.3

FIGURE A7 Table showing average CPU time (in seconds) taken by each algorithm for each instance