Journal of Theoretical and Applied Information Technology
15th August 2021. Vol.99. No 15

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

SOLVING TRAVELING SALESMAN PROBLEM USING GENETIC ALGORITHM BASED ON EFFICIENT MUTATION OPERATOR

[1]AHMAD BANY DOUMI, [2]BASEL A. MAHAFZAH, [3]HAZEM HIARY
[1,2,3]Department of Computer Science, The University of Jordan, Amman 11942, Jordan

E-mail: [1]ahmad.domi.usm@gmail.com, [2]b.mahafzah@ju.edu.jo, [3]hazemh@ju.edu.jo

ABSTRACT

The Traveling Salesman Problem (TSP) is a Combinatorial Optimization Problem (COP), which belongs to NP-hard problems and is considered a typical problem for many real-world applications. Many researchers used the Genetic Algorithm (GA) for solving the TSP. However, using a suitable mutation was one of the main obstacles for GA. This paper proposes for GA an Efficient Mutation (GA-EM) for solving TSP. The efficient mutation can balance between deeply searching and preventing stuck on local optima to ensure a better convergence rate and diversity. Therefore, in this paper, a local search method based on three neighborhood structure operators; namely, transpose, shift-and-insert, and swap, is proposed to produce the efficient mutation for GA. The performance of the proposed algorithm is validated by three TSP datasets; including, TSPLIB, National TSPs, and VLSI Data Set. These datasets have different graphs' structures and sizes. The sizes of the datasets range from 150 to 18512 cities. For comparative evaluation, the results obtained from the proposed GA-EM are compared with those obtained by four relatively recent approaches using the same TSP instances. These approaches are the Modernised Genetic Algorithm for solving TSP (MGA-TSP), List-Based Simulated Annealing algorithm (LBSA), Symbiotic Organisms Search optimization algorithm based on Simulated Annealing (SOS-SA), and Multiagent Simulated Annealing algorithm with Instance-Based Sampling (MSA-IBS). The GA-EM outperformed these approaches in all used TSP instances in terms of accuracy.

Keywords: *Genetic Algorithm, Mutation Operator, Neighboring Operator, Simulated Annealing Algorithm, Traveling Salesman Problem*

1. INTRODUCTION

Traveling Salesman Problem (TSP) is a combinatorial optimization problem [1–3]. It can be categorized as an NP-hard class problem in almost all of its variations [1–4]. In TSP, given a set of cities where each city is visited exactly once by the salesman and returns to the initial city with a minimum distance tour [1–3]. When the number of cities is increased, the convergence rate to solve TSP is normally decreased [2, 5]. TSP is normally considered a good problem to evaluate the performance of newly established algorithms. TSP is very useful for real-world applications employed in military and traffic domains. Since the exact methods are not efficient for solving the large-scaled TSP due to the huge computational time consumed, the researchers in the optimization domain tend to apply the approximation approaches to solve TSP [2, 6, 7].

Metaheuristic approaches are categorized as approximation methods. Thus, two main types of algorithms belong to metaheuristic approaches, which are the local search-based algorithms and the evolutionary-based algorithms [8–10]. Local search-based algorithms start with a random solution. Then, they iteratively modify the solution using neighboring mechanisms until a local optimal solution is reached. Many local search-based methods are adapted for solving TSP; including, tabu search [11] and variable neighborhood search [12, 13]. Whereas, the Evolutionary-based Algorithms (EAs) begin with a collection of random solutions called the initial population. In each iteration, new solutions are generated using recombination and mutation operators. The parent population is normally replaced by a new offspring population if the newly generated solutions are better. This evolution process will continue until the stagnation point is reached. The Genetic Algorithm

(GA) is one of the evolutionary-based algorithms [14, 15].

In general, the GA algorithm has two drawbacks [16, 17]; first, it may get stuck in local optimum, which is not necessarily the best possible solution (i.e., global optima), because it searches in each candidate solution but without deep searching. Second, when it deals with a larger number of cities, the convergence rate will be weaker. Therefore, researchers employ local search-based algorithms due to their capabilities in local exploitation in solving TSP [7, 18].

One of the main obstacles of GA is the mutation operator. The mutation operator in GA usually produces the worst candidate solutions, where this problem can be eventually reflected in the whole population and results [6]. However, when GA runs with an efficient mutation, the convergence rate and the diversity aspect will be better [6].

In this paper, a new mutation for GA is proposed and is called an Efficient Mutation, which is denoted as GA-EM. This efficient mutation can balance between deeply searching and preventing stuck on local optima to ensure a better convergence rate and diversity. The new efficient mutation operator is based on three neighborhood structures (transpose, shift-and-insert, and swap), where it avoids the stuck state in local optima. The proposed mutation is called after a number of failed tries made to enhance the solutions rather than it is called in each GA iteration. The number of failed tries is defined and limited by threshold values. The new efficient mutation includes two steps; in the first step, we will apply the transpose step on randomly selected cities from the worst candidate solution. This transpose step is conducted without heuristic calculations to decide whether it will produce a better solution or not. In the second step, we will use the three neighborhood operators; namely, transpose, shift-and-insert, and swap to enhance the modified solution generated in the first step.

The experiments are conducted using three TSP datasets, namely TSPLIB [19, 20], National TSPs [21], and VLSI Data Sets [22] of different structures and sizes including 30 TSP instances. The size range of the TSP instances from 150 to 18512 cities. For comparative evaluation, the results obtained by the proposed GA-EM are compared with other results obtained by four recent metaheuristic algorithms using the same TSP instances. These algorithms are the Modernised Genetic Algorithm for solving TSP (MGA-TSP), List-Based Simulated Annealing (LBSA) algorithm, Symbiotic Organisms Search optimization algorithm based on Simulated Annealing (SOS-SA), and Multiagent Simulated Annealing algorithm with Instance-Based Sampling (MSA-IBS).

The rest of this paper is organized as follows: Section 2 presents the definition of the TSP problem, an overview of the genetic algorithm, an overview of simulated annealing, and their related work. Section 3 presents the proposed GA-EM with its neighboring operators. Section 4 presents the experimental results and discussions. Finally, Section 5 presents the conclusion and future work.

2. BACKGROUND AND RELATED WORK

In this paper, TSP is modeled as an optimization problem and solved using GA with efficient mutation and its performance is compared with several approaches used GA and Simulated Annealing (SA). Therefore, in this section, a background of TSP, GA, and SA is presented. Also, as related work, several approaches used GA or SA to solve TSP are briefly presented.

2.1 Traveling Salesman Problem

TSP is considered one of the popular NP-Hard problems [1–4]. However, the goal of optimization algorithms is to find the shortest tour between a number of cities, but each city must be visited only once [1–3]. TSP can be expressed as a bi-directed graph $G = (C, A)$, where C is a set of cities, and A is a set of arcs (i.e., the edge between cities). Also, a cost matrix D of size $|C| \times |C|$ is used to store the distance between all pairs of cities, where the distance procedure calculates the distance between the two cities c_i and c_i+1. In general, cost matrices can be divided into two types symmetric or asymmetric. In the symmetric type, distance (i, j) equals distance (j, i), whereas distance (i, j) does not equal distance (j, i) in the asymmetric type, where the distance between cities is dependent on the direction of traversing the arcs. Mathematically, the objective function of the TSP can be formulated as shown in Eq. (1) [7], where T_d is the total closed tour length and the distance procedure (dis) is the distance between the two cities c_i and c_i+1.

$$T_d = \min(\text{dis}(c_{n-1}, c_1) + \sum_{i=1}^{n-1} \text{dis}(c_i, c_{i+1}) \qquad (1)$$

2.2 Genetic Algorithm

The Genetic Algorithm (GA) is based on a known principle called the survival of the fittest [1, 14, 23, 24]. Initially, GA starts with a set of

solutions called population. Each solution represents a vector of decision variables and each decision variable has a specific range of values [14, 25]. In this analogy, each solution is a chromosome, each decision variable is a gene, and each value of the decision variable is an allele [7, 14]. GA has a set of parameters including the size of the population, the number of iterations, also a set of operations including the selection, the crossover, and the mutation. Therefore, these solutions have to be manipulated by crossover and mutation operations through iterations. Algorithm 1 shows the main steps of GA, which initiates with an initial population X represented by random candidate solutions. Each candidate solution is evaluated and ranked based on the objective function as shown in Eq. (1). The main loop of GA (Line 3 to Line 9) strives to enhance the population by repeating Line 4 to Line 8 while a termination criterion is not met or an optimal solution is not reached.

Algorithm 1 Genetic Algorithm (GA)	
1	**X**← *Generate_Initital_Population*
2	*Evaluate*(**X**)
3	**while** (Stopping criterion is not met or optimal solution is not reached) **do**
4	**Y** ← *Selection*(**X**)
5	**W** ← *Crossover*(**Y**)
6	**Z** ← *Mutation*(**W**)
7	*Evaluate*(**Z**)
8	**X**← *Replacement*(**Z**, **X**)
9	**end while**

2.3 Simulated Annealing Algorithm

The Simulated Annealing (SA) algorithm is a generic probabilistic meta-algorithm that can be used to find an approximate solution to global optimization problems such as TSP. It is based on the idea of the cooling process of molten metal [1, 26, 27]. SA uses the temperature as a parameter to decide whether the solution will be updated or not. The temperature is decreased until the local best solution is reached. Thus, each step modifies the solution and decreases the probability to update the best solution. If the temperature is set to a high value, SA can allow some random moves. When the temperature cools or has a low value, the probability of a random move is reduced. A detailed algorithm with some of its variants is available in [27–30].

2.4 Related Work

Many metaheuristic approaches are used to solve various optimization problems including the TSP. Examples of these approaches are genetic algorithm, simulated annealing, swarm simulated annealing, discrete spider monkey optimization, harmony search, variable neighborhood search, and grey wolf optimizer [12, 27–39].

The TSP is a good problem to evaluate the performance of several algorithms [1, 2], there are many related works in solving TSP; specifically, those which used approximation algorithm with a mutation to find the shortest distance of traveling salesman through a set of cities based on TSP constraints.

The authors in [35] presented a Multi-Offspring Genetic Algorithm (MO-GA) based on biological evolutionary and mathematical ecological theory for solving TSP. In this approach, the number of children is significantly increased as compared to the basic genetic algorithm. It can increase the probability of generating an optimal solution since it was based on producing more offsprings in each iteration.

In [30], the authors proposed an adaptive hybrid metaheuristic approach that combines simulated annealing and tabu search algorithms with a dynamic neighborhood mutation for solving TSP. This approach achieved improved accuracy as compared to simulated annealing and tabu search algorithms. It can overcome the disadvantages of simulated annealing and tabu search. The hybrid approach provided a clear convergence process and a fast decrease rate. The dynamic neighborhood improved solution quality in comparison with the classical 2-opt neighborhood.

A Multiagent and Simulated Annealing with Instance-Based Sampling (MSA-IBS) was proposed by Wang et al. [29] for solving the TSP. The hybrid process exploited the learning ability of the instance-based search algorithm to enhance the simulated annealing to solve TSP. But the error rate was very high for datasets that are larger than 500 cities.

Zhan et al. [27] proposed a List-Based Simulated Annealing (LBSA) approach to solve the TSP. This approach used the parameter sensitivity and effectiveness of the list-based cooling schedule to determine temperature reduction in the simulated annealing algorithm, which is used as acceptance criteria for choosing a candidate solution. The results of the LBSA show that it performs fairly well compared to some other state-of-the-art algorithms; such as MSA-IBS. This approach achieved an error rate less than MSA-IBS, but the error rate is still high.

Ezugwu et al. [28] presented a hybrid approach of Symbiotic Organisms Search (SOS) with Simulated Annealing (SA) to solve the TSPs. The framework of SOS-SA incorporated the SA local search capability into the problem search space of the SOS algorithm. The empirical assessment results showed that the performance of the algorithm and its convergence rate in some cases produced better results than the best-known TSP benchmarked results. However, the high error rate for a large number of cities is the main challenge for this approach.

Mohsen [36] presented a hybridized algorithm, which is called an annealing elitist ant system. This algorithm combined Ant Colony Optimization (ACO), Simulated Annealing (SA), mutation operator, and local search procedure to solve TSP. The algorithm exploited the mutation operator to increase the ants' population diversity from time to time, and the local search exploits the current search area efficiently. The comparative experiments showed that this algorithm outperformed some well-known algorithms but the error rate is high when the number of cities increased.

The GA is considered one of the most successful evolutionary algorithms used for TSP [7]. Many researchers developed a new GA. Nagata et al. (2013) [33] developed Edge Assembly Crossover (EAX) for GA to solve TSP. In addition to that, Tsai et al. [37], proposed an effective algorithm to reduce the error rate. Also, in [38], the authors proposed another improved GA to solve TSP using new combination crossover operators. In [39], the authors surveyed GAs to solve TSP.

Al-Khatib et al. [7] proposed a Modernised Genetic Algorithm for solving TSP (MGA-TSP). MGA-TSP utilized an efficient crossover operator called EAX to enhance its convergence. It should be noted that this algorithm did not use an efficient mutation to enhance the solution by terminating the stuck state in local optima. MGA-TSP achieved the best results compared to some other state-of-the-art algorithms; such as LBSA, EAX, and SOS in terms of error rate.

In this paper, a new efficient mutation based on three neighborhood operators; namely, transpose, shift-and-insert, and swap, is exploited to enhance the best solution in terms of accuracy (or error rate) by preventing stuck on local optima. The efficient mutation interferes after a changed number of failed trials to enhance the solutions in the population. The proposed mutation has a small change on the shortest distance of the target solution since a small number of edges in the graph of the target solution will be replaced randomly; therefore, the efficient mutation can cause a small change on the whole population in terms of shortest distance in trying to allow for GA-EM to continue in searching deeply toward an optimal solution.

3. GA-EM APPROACH

In this section, the proposed GA-EM approach is discussed and presented in Algorithm 2. This paper focuses on the proposed efficient mutation in terms of how and when it works. In general, the solutions that are exposed to the classical mutation, often become worse than they were before. The proposed efficient mutation takes to account this issue by using the proposed mutation after a specific time defined by a threshold. This threshold takes into consideration the deep search in the proposed approach as well as the proposed efficient mutation performs an operation that can be described as a precise mutation for the candidate solution and then an improvement process for the generated solution. Briefly, the proposed mutation picks one of the solutions randomly after failed tries to improve the population, then it modifies the picked solution. This mutation is based on three neighboring operators; namely, transpose, shift-and-insert, and swap discussed in the following sections. The main motivation of the efficient mutation is to terminate the stuck state in local optima. The proposed GA-EM is initiated with an initial population of candidate solutions that are generated randomly and then improved by a 2-opt neighboring algorithm which is shown in Algorithm 3. The GA-EM uses random selection, EAX crossover, and the proposed efficient mutation operators to enhance the population. Thus, GA-EM is terminated after checking that there is no possible improvement in the population. However, each step of the proposed GA-EM, as shown in Algorithm 2, is discussed in the following sections.

3.1 TSP Initialization

In TSP, each candidate solution for GA is represented by all cities of TSP as $H = (c_1, c_2, ..., c_N)$ of N cities, where each city should appear one time only. Each candidate solution is evaluated by the fitness function formulated in Eq. (1). The fitness values are used to rank the candidate solutions in the population from best to worst solutions.

3.2 Improving the Initial Population

The 2-opt neighboring algorithm, which is presented in Algorithm 3, is widely used to enhance

the initial population in the approximation algorithms, such as GA in [18, 33, 40]. In Algorithm 3, the Neighbour[c][0] and Neighbour[c][1] represent the cities that precede and follow city c, respectively, in the candidate solution, near[c][j] array indicates the jth nearest city-to-city c, d represents the distance value between the two cities and H is a set of cities of the candidate solution. In general, the 2-opt algorithm works as in the following instance: the edge $E_{1,2}$ = (c_1, c_2) and the edge $E_{3,4}$ = (c_3, c_4) will be changed to the edge $E_{1,3}$ = (c_1, c_3) and the edge $E_{2,4}$ = (c_2, c_4) based on the heuristic equation which is shown in Line 10 of Algorithm 3. The heuristic equation can simply decide whether the two edges, that are connected between the cities in the current graph to be replaced, have a fitness value in terms of the shortest distance worse than the fitness value of the two edges to be added or not. Thus, the 2-opt is a local search and improvement mechanism for neighboring cities in TSP [18, 40].

Algorithm 2 Efficient Mutation for GA (GA-EM)	
1	**X**← *Generate_Initital_Population by 2-opt neighboring algorithm* // See Algorithm 3.
2	*Evaluate*(**X**)
3	**while** (Stopping criterion is checking that the algorithm has no possible improvement on population) **do** // Number of iterations of GA-EM is denoted by $Itr_{GA\text{-}EM}$
4	**Y** ← *Random_Selection*(**X**)
5	**W** ← *EAX_Crossover*(**Y**)
6	**Z** ← *Efficient_Mutation*(**W**) // See Algorithm 4
7	*Evaluate*(**Z**)
8	**X** ← *Replacement*(**Z**, **X**)
9	**end while**

3.3 Selection Operator

In the Selection procedure, for all candidate solutions in the population, every two solutions will be randomly selected to produce the next solutions. The generated solutions will formulate the new population in each generation.

3.4 Edge Assembly Crossover Operator

The Edge Assembly Crossover (EAX) is an efficient crossover operator used by GA-based TSP [33]. EAX merges two candidate solutions from the population according to the selection operator in one graph G. The redundant edges of graph G will be modified by the AB-cycles method [33]. Further, one edge (say a) is taken from first parent (say A); and another edge (say b) is taken from the second parent (say B) alternatively and continually (a-b-a-b-a-...-a-b). Each graph G can be divided into a set of AB-cycles. There is an E-set for AB-cycles which represents the number of generated AB-cycles. E-set is exploited to cut one candidate solution into sub-cycles. Then, the sub-cycles are merged using greedy criteria based on the Hamiltonian cycle approach, where the cycle of the shorter number of edges and not in both parents (A and B) will be exploited. EAX is presented in more detail in [33].

Algorithm 3 2-opt Neighboring Algorithm	
1	Randomly **generate** a solution H = {1, ..., N};
2	**repeat**
3	Randomly **select** $c_1 \in H$;
4	**for** i = 0 to 1 **do**
5	c_2 = *Neighbour*[c_1][i];
6	**for** j = 1 to *ConstNum* **do**
7	c_3 = *near*[c_1][j]; // { *ConstNum* < N}
8	**if** ($-d(c_1, c_2) + d(c_1, c_3)) \geq 0$ **then break**;
9	c_4 = *Neighbour*[c_3][(i + 1) mod 2];
10	**if** ($(-d(c_1, c_2) - d(c_3, c_4) + d(c_1, c_3) + d(c_2, c_4)) < 0$) **then**
11	**Update** the candidate solution and H. Go to Line 3;
12	**end if**
13	**end for**
14	**end for**
15	**until** H becomes empty

3.5 Efficient Mutation Operator

The mutation operator usually follows the crossover operator to avoid stuck on local optima. In general, this operator will increase the possibility for GA to find the optimal solution. Thus, Algorithm 4 shows the efficient mutation operator. In the efficient mutation operator, the positions of two cities are selected randomly and the positions of cities which are located between these selected two cities will be transposed as shown in Line 7 of Algorithm 4. Then, the three neighboring operators will try to improve the modified solution as shown in Lines 9–14 of Algorithm 4. Finally, the improved solution will be submitted to the current population. Normally, the proposed mutation will run after enough failed tries to improve the population as shown in Line 1 of Algorithm 4. The following steps describe the efficient mutation operator, as shown in Algorithm 4:

Step 1: Efficient mutation operator initiates when GA fails in improving the population during a

specific number of trials. The specific number of trials is bounded by minimum and maximum threshold values; initially, the specific number of trials is defined by the maximum threshold value and decreased after each entry. When the minimum threshold value is reached, the value is redefined to the maximum threshold value. Thus, the best distance of the solution within the previous population computed by Eq. (1) will be used to be compared with the best distance of the solution within the current population.

Algorithm 4 Efficient Mutation Operator	
1	**Define** *minimum_threshold* and *maximum_threshold* values
2	**Store** the value of *maximum_threshold* in *initial_value*;
3	*number_of_trials* = *maximum_threshold*;
4	**if** *previous_best_distance* = *current_best_distance* after *number_of_trials* **then**
5	**A** ← *get_worst_candidate_solution*;
6	**for** *j* = 0 to $Itr_{TRANSPOSE}$ **do**
7	**B** ← *transpose*(**A**); // Select two cities randomly (without heuristic calculation)
8	**end for**
9	**for** *i* = 0 to Itr_{SM} **do**
10	**C** ← *transpose*(**B**);
11	**D** ← *shift-and-insert*(**C**);
12	**E** ← *swap*(**D**);
13	**A** ← *short_tour*(**B, C, D, E**);
14	**end for**
15	*maximum_threshold* = *maximum_threshold* – 1;
16	**if** *maximum_threshold* = *minimum_threshold* **then**
17	*maximum_threshold* = *initial_value*;
18	**end if**
19	**end if**

Step 2: Getting the worst solution, A, from the current population in terms of worst distance based on Eq. (1).

Step 3: Manipulating the worst solution A by selecting two cities randomly, then applying the transpose procedure without heuristic calculations to come up with solution B. This step can be repeated a small number of times.

Step 4: The improvement loop of efficient mutation (Lines 9 to 14 of Algorithm 4), tries to enhance solution B by repeating the neighboring operators (transpose, shift-and-insert, and swap) based on heuristic calculations while a termination criterion is not met.

Step 5: Identifying a new value for the number of failed trials.

The transpose procedure (*C_Sol*; *i*, *j*) transposes the cities between location *i* and location *j* from the candidate solution *C_Sol*. The proposed GA-EM works on a deep search where it strives to reach an optimal solution. In general, the mutation operator prevents access to the optimal solution [6]. So, the proposed efficient mutation operator will consider this conflict. The efficient mutation is based on applying the transpose operator on cities selected randomly as the initial step and then three operators (transpose, shift-and-insert, and swap) are executed to enhance the modified solution using heuristic equations. Thus, Algorithm 4 shows the efficient mutation process. Then the modified solution will be added to the current population as a new candidate solution. The transpose operation is the best choice to be the initial step in Algorithm 4 since the transpose operation replaces only two edges in the graph in each transpose step. Furthermore, the transpose operation causes a small change on the candidate solution compared to swap and shift-and-insert operations. The transpose operator is shown in Algorithm 5, where *C_Sol* represents a candidate solution, c_i indicates to the city *i* in *C_Sol*, c_j indicates to city *j* in *C_Sol*, and *d* represents the distance value between the two cities c_i and c_j. In Algorithm 5, initially, the next cities c_i+1 and c_j+1 are selected, which are neighbors to c_i and c_j respectively in the candidate solution, from c_i and c_j to be used in the heuristic equation shown in Line 3, which is based on distance values between the cities to decide whether new edges can be added between (c_i and c_j) and (c_i+1 and c_j+1) or not after removing the current edges between (c_i, c_i+1) and (c_j, c_j+1) from the candidate solution.

The swap procedure (*C_Sol*; *i*, *j*) swaps the two cities in location *i* and location *j* from the candidate solution *C_Sol*. This operator can change four edges. Algorithm 6 shows how the swap operator works. In Algorithm 6, initially, the cities c_i+1, c_j+1, c_i+2, and c_j-1 are selected from c_i and c_j in the candidate solution to be used in the heuristic equation shown in Line 5, which is based on distance values between the cities to decide whether new four edges can be added between the corresponding cities or not after removing the current edges. The heuristic condition decides whether replacing the current edges in the graph of the candidate solution (c_i, c_i+1), (c_i+1, c_i+2), (c_j-1, c_j) and (c_j, c_j+1) with new four edges between (c_i,

c_j), (c_j, c_i+2), (c_j-1, c_i+1) and (c_i+1, c_j+1) increases the fitness value in terms of shortest distance or not.

The shift-and-insert procedure (*C_Sol*; *i*, *j*) shifts the city in location *j* to location *i* consecutively from the candidate solution *C_Sol*. This operator can change three edges. Algorithm 7 shows how the shift-and-insert operator works. In Algorithm 7, initially, the cities c_i+1, c_j+1, and c_j-1 are selected from c_i and c_j in the candidate solution to be used in the heuristic equation shown in Line 4, which is based on distance values between the cities to decide whether new three edges can be added after removing the current edges between the corresponding cities or not. The heuristic condition decides whether replacing the current edges in the graph of the candidate solution (c_i, c_i+1), (c_j-1, c_j) and (c_j, c_j+1) with new three edges between (c_i, c_j), (c_j, c_i+1), and (c_j-1, c_j+1) increases the fitness value in terms of shortest distance or not.

Algorithm 5 Transpose(C_Sol, c_i, c_j)	
1	c_{i+1} ← store the city which is after c_i in *C_Sol*
2	c_{j+1} ← store the city which is after c_j in *C_Sol*
3	**if** (($d(c_i, c_j) + d(c_{i+1}, c_{j+1})) < (d(c_i, c_{i+1}) + d(c_j, c_{j+1}))$)
4	**if** ($c_{i+1} < c_j$)
5	$temp = c_{i+1}$
6	$c_{i+1} = c_j$
7	$c_j = temp$
8	**while** ($c_{i+1} > c_j$)
9	$temp = C\_Sol[c_{i+1}]$
10	$C\_Sol[c_{i+1}] = C\_Sol[c_j]$
11	$C\_Sol[c_j] = temp$
12	$c_j = c_j + 1$
13	$c_{i+1} = c_{i+1} - 1$
14	**end while**
15	**end if**
16	**end if**

3.6 Evaluation Operator

The proposed GA-EM algorithm calls the objective function for each candidate solution to calculate the fitness value. Then, if the child solutions are fittest, will replace the parent solutions in the population. Since TSP aims to find the shortest path (or shortest distance) and GA is based on the survival of the fittest, GA-EM uses an objective function based on Eq. (1) to find the fitness values for candidate solutions of the population in each iteration of GA.

3.7 Termination Condition

There are different criteria to terminate the GA algorithm such as the number of iterations, reaching an optimal solution, or other criteria. The termination condition for the proposed GA-EM algorithm is checking that the algorithm has no possible improvement in the population. Therefore, GA-EM computes the termination criterion by calculating the average distance of all solutions in the population and then comparing the average distance with the distance of the best solution in the population. If the result equals or less than a threshold value, then the termination criterion is met. The final result of the proposed algorithm will be the fittest solution. The fittest solution means the solution with the highest objective function value compared to the rest of the solutions.

Algorithm 6 Swap(C_Sol, c_i, c_j)	
1	c_{i+1} ← store the city which is after c_i in *C_Sol*
2	c_{i+2} ← store the city which is after c_{i+1} in *C_Sol*
3	c_{j+1} ← store the city which is after c_j in *C_Sol*
4	c_{j-1} ← store the city which is before c_j in *C_Sol*
5	**if** (($d(c_i, c_j) + d(c_j, c_{i+2}) + d(c_{j-1}, c_{i+1}) + d(c_{i+1}, c_{j+1})) < (d(c_i, c_{i+1}) + d(c_{i+1}, c_{i+2}) + d(c_{j-1}, c_j) + d(c_j, c_{j+1}))$) **AND** ($c_{i+2} <> c_j$))
6	$temp_1 = C\_Sol[c_{i+1}]$
7	$temp_2 = C\_Sol[c_j]$
8	$C\_Sol[c_{i+1}] = temp_2$
9	$C\_Sol[c_j] = temp_1$
10	**end if**

4. EXPERIMENTAL ENVIRONMENT AND RESULTS

In this section, the experimental environment and the obtained results by GA-EM, MGA-TSP, SOS-SA, LBSA, and MSA-IBS are presented. The datasets of 30 instances are used to test the proposed GA-EM algorithm and the other mentioned four metaheuristic algorithms, where these 30 instances are carefully selected from TSP datasets: TSPLIB [19, 20], National TSPs [21], and VLSI data sets [22]. These datasets have different numbers of cities and graph structures as shown in Table 1.

To evaluate the performance of the proposed GA-EM algorithm, the obtained results of the GA-EM are compared with the other four metaheuristic algorithms. Some of these algorithms showed the best-known results for the used TSP instances. Thus,

one of these algorithms used GA as a primary algorithm for solving TSP and others used hybrid algorithms such as simulated annealing with other algorithms designed for TSP specifically. For evaluation purposes, the experimental testing platform for the proposed GA-EM algorithm was conducted on a 2.53 GHz CPU Desktop with 3GB RAM, while the programming language is C# console on Microsoft Visual Studio 2010.

Algorithm 7 Shift-and-Insert(C_Sol, c_i, c_j)	
1	c_{i+1} ← store the city which is after c_i in *C_Sol*
2	c_{j+1} ← store the city which is after c_j in *C_Sol*
3	c_{j-1} ← store the city which is before c_j in *C_Sol*
4	**if** $((d(c_i, c_j) + d(c_j, c_{i+1}) + d(c_{j-1}, c_{j+1})) < (d(c_i, c_{i+1}) + d(c_{j-1}, c_j) + d(c_j, c_{j+1})))$
5	*temp* = *C_Sol*[c_j]
6	**if** ($c_j < c_{i+1}$)
7	**for** ($i = c_j + 1$; $i <= c_{i+1}$; i++)
8	*C_Sol*[i-1] = *C_Sol* [i]
9	**end for**
10	**end if**
11	**if** ($c_j >= c_{i+1}$)
12	**for** ($i = c_j$; $i > c_{i+1}$; i--)
13	*C_Sol* [i] = *C_Sol* [i-1]
14	**end for**
15	**end if**
16	*C_Sol* [c_{i+1}] = *temp*
17	**end if**

The parameter values have a significant influence on the solution's quality of each algorithm, all these algorithms shared the same population size which equals 100 candidate solutions except MSA-IBS where it is not defined [29]. The mean results of 20 runs are calculated for each of the 9 TSP instances, as shown in Tables 2 and 3.

As mentioned previously, efficient mutation involves two phases. The first phase is the random mutation step "without heuristic calculation" as shown in Line 7 of Algorithm 4 and the second phase is the enhancement step "with heuristic calculation" for a solution as shown in Lines 9–14 of Algorithm 4.

Table 2 shows the results of GA-EM using transpose, shift-and-insert, and swap operators where each operator has one random mutation "without heuristic calculation". Therefore, the second phase of the efficient mutation "with heuristic calculation" is not applied. The number of steps of transpose, shift-and-insert, and swap operators in each GA iteration is one step. The OPT-BKS column is the Optimal Best-Known Solution results for each TSP instance.

Table 2 shows the results of GA-EM using the classical mutation in each GA iteration. According to the results shown in Table 2, the transpose operator is better to use the random mutation in finding the shortest distance. So that, the efficient mutation employs the transpose operator in the first phase.

Since the proposed GA-EM does not execute the efficient mutation in each iteration, the threshold values used to enter the efficient mutation must be selected, which is, as a minimum threshold value equals 1 and as a maximum threshold value equals 10. Thus, using a threshold value larger than 10 can lead to vanishing the mutation role. However, the number of initial transpose steps executed on cities that are selected randomly and without heuristic calculations is validated in terms of mean accuracy as shown in Table 3.

As borne out by the results shown in Table 3, almost all the best results are obtained when the number of transpose steps equals 2, this means the first phase of efficient mutation operation replaces 4 edges in the candidate solution in each entry for efficient mutation operation. The results show a small number of mutations after a specific number of failed tries can produce more efficient enhancement than executing a large number of mutations since the GA-EM searches deeply toward the best solution.

Table 4 presents the abbreviations of the four comparative algorithms. However, Table 5 shows the mean accuracy results. These results are obtained by the proposed GA-EM algorithm and the other four metaheuristic algorithms, namely MGA-TSP, SOS-SA, LBSA, and MSA-IBS. Also, Table 5 shows the mean results of 20 runs by each algorithm for all TSP instances. In Table 5, the best results in terms of accuracy are highlighted in bold font.

According to Table 5, the proposed GA-EM algorithm achieved better accuracy than the other four comparative algorithms in 11 datasets. Also, GA-EM achieved the best-known optimal solutions in 19 datasets, as well as the other four algorithms, where they have obtained the same best results in terms of mean accuracy. The proposed algorithm has achieved these results since it can ensure a diversity aspect in the population during the GA-EM running process. Thus, the population with a diverse aspect can achieve better results, where the candidate solutions selected to crossover can

produce new child has more capabilities to reach optimal results. However, the MGA-TSP algorithm achieved the second rank. The results show that preserving diversity aspects in the population during the execution of an algorithm used to solve TSP such as in the proposed GA-EM, can achieve better results than others such as MGA-TSP which offering the diversity aspect in the initial population step only. In summary, the proposed GA-EM has achieved better results in terms of accuracy since GA-EM can ensure a diversity aspect in the population until the termination condition is met.

Table 1: Datasets of TSP Instances.

Instance Number	Instance Name	Optimal Solution	General Description Name
1	Ch150	6,528	150 city problem (Churritz)
2	U159	42,080	Drilling problem (Reinelt)
3	Rat195	2,323	Rattled grid (Pulleyblank)
4	Kroa200	29,368	200 city problem A (Krolak/Felts/Nelson)
5	Ts225	126,643	225 city problem (Juenger)
6	Gil262	2,378	262 city problem (Gillet/Johnson)
7	Pr299	48,191	299 city problem (Padberg/Rinaldi)
8	Lin318	42,029	318 city problem (Lin/Kernighan)
9	Rd400	15,281	400 city random TSP(Reinelt)
10	Fl417	11,861	Drilling problem (Reinelt)
11	Pr439	107,217	439 city problem (Padberg/Rinaldi)
12	U574	36,905	Drilling problem (Reinelt)
13	Rat575	6,773	Rattled grid (Pulleyblank)
14	U724	41,910	Drilling problem (Reinelt)
15	Rat783	8,806	Rattled grid (Pulleyblank)
16	Pr1002	259,045	1,002 city problem (Padberg/Rinaldi)
17	Pcb1173	56,892	Drilling problem (Juenger/Reinelt)
18	D1291	50,801	Drilling problem (Reinelt)
19	Rl1323	270,199	1,323 city TSP (Reinelt)
20	Fl1400	20,127	Drilling problem (Reinelt)
21	D1655	62,128	Drilling problem (Reinelt)
22	Vm1748	336,556	1,784 city problem (Reinelt)
23	U2319	234,256	Drilling problem (Reinelt)
24	Pcb3038	137,694	Drilling problem (Junger/Reinelt)
25	Fnl4461	182,566	Die 5 neuen Laender Deutschlands (ExDDR) (Bachem/Wottawa)
26	Rl5934	556,045	5,934 city TSP (Reinelt)
27	Pla7397	23,260,728	Programmed logic array (Johnson)
28	Usa13509	19,982,859	Cities with pop. at least 500 in the continental US
29	Brd14051	469,385	BR Deutschland in den Grenzen von 1989 (Bachem/Wottawa)
30	D18512	645,238	Bundesrepublik Deutschland (Bachem)

Table 2: Mean Accuracy of GA-EM Using Three Variations of Mutation for 9 TSP Instances.

S/N	Instance Name	OPT-BKS	Transpose Operator	Shift-and-Insert Operator	Swap Operator
1	Ch150	6,528	6,533	6,537	6,538
2	Lin318	42,029	42,037	42,042	42,045
3	Pr1002	259,045	259,055	259,064	259,069
4	D1655	62,128	62,149	62,156	62,162
5	U2319	234,256	234,371	234,378	234,390
6	Pcb3038	137,694	137,743	137,764	137,783
7	Fnl4461	182,566	182,609	182,654	182,675
8	Rl5934	556,045	556,121	556,142	556,174
9	Pla7397	23,260,728	23,261,816	23,261,853	23,261,877

Table 3: Mean Accuracy of GA-EM Using Three Variations of Number of Transpose Steps for 9 TSP Instances.

S/N	Instance Name	OPT-BKS	Number of Transpose Steps		
			= 2	= 5	= 10
1	Ch150	6,528	6,528	6,528	6,528
2	Lin318	42,029	42,029	42,040	42,077
3	Pr1002	259,045	259,045	259,062	259,089
4	D1655	62,128	62,134	62,145	62,159
5	U2319	234,256	234,350	234,372	234,382
6	Pcb3038	137,694	137,695	137,711	137,720
7	Fnl4461	182,566	182,570	182,588	182,595
8	Rl5934	556,045	556,054	556,074	556,092
9	Pla7397	23,260,728	23,261,749	23,261,782	23,261,796

Table 4: The Abbreviations of the Four Comparative Algorithms.

Abbreviation	Algorithm Name	Reference
MGA-TSP	Modernised genetic algorithm for the travelling salesman problem	[7]
SOS-SA	Simulated annealing based symbiotic organisms search optimization algorithm	[28]
LBSA	List-based simulated annealing algorithm	[27]
MSA-IBS	Multiagent simulated annealing algorithm with instance-based sampling	[29]

5. CONCLUSION AND FUTURE WORK

In this paper, an efficient mutation for a genetic algorithm is presented and used to solve TSP. The efficient mutation is based on three operators (transpose, shift-and-insert, and swap) to avoid getting stuck on local optima and allow for deep search. The proposed efficient mutation is called after a specific number of failed tries to improve the population of candidate solutions. The specific number of trials is bounded by minimum and maximum threshold values. Thus, initially, the specific number of trials is defined by the maximum threshold value and decreased after each entry. When the minimum threshold value is reached, the value is redefined to the maximum threshold value. The proposed GA-EM is applied for a small number of tries to mutate the population which led to reduce the number of failed tries to enhance the population during the enhancement process. The initial step of efficient mutation is based on selecting two cities randomly from the selected solution, then applying the transpose

procedure without heuristic calculations, this step is repeated a specific number of times, and then applying heuristic steps by the three operators (transpose, shift-and-insert, and swap). The results obtained from the proposed GA-EM were compared with those obtained by four relatively recent efficient algorithms; namely, MGA-TSP, SOS-SA, LBSA, and MSA-IBS using the same TSP instances. The proposed algorithm can outperform these algorithms in all tested TSP instances in terms of accuracy.

In general, solving TSP for complex and large TSP datasets requires a lot of running time on sequential computers. Therefore, improving the running time can be accomplished using a parallel approach as shown in [41, 42]. So, improving the running time of the proposed GA-EM can be achieved using a parallel approach, which is considered as future work. Parallel approaches can be achieved on various parallel architectures and interconnection networks. Examples of these architectures and interconnection networks are OTIS hyper hexa-cell [43–45], OTIS-Hypercube [46], chained-cubic tree interconnection network [47, 48], optical chained-cubic tree [49], OTIS-Mesh interconnection network [46, 50], and hyper hexa-cell interconnection network [51, 52]. Accordingly, as future work, other complex and larger TSP datasets can be solved to further ensure the validity of the GA-EM algorithm.

Table 5: The Mean Accuracy Results of the GA-EM in Comparison to the Other Four Metaheuristic Algorithms.

S/N	Instance Name	Mean Accuracy					
		OPT-BKS	MGA-TSP (2019) [7]	SOS-SA (2017) [28]	LBSA (2016) [27]	MSA-IBS (2015) [29]	GA-EM
1	Ch150	6,528	6,528	6,530	6,530	6,529	6,528
2	U159	42,080	42,080	42,081	42,080	42,080	42,080
3	Rat195	2,323	2,323	2,327	2,328	2,330	2,323
4	Kroa200	29,368	29,368	29,371	29,374	29,378	29,368
5	Ts225	126,643	126,643	126,701	126,643	126,643	126,643
6	Gil262	2,378	2,378	2,382	2,379	2,379	2,378
7	Pr299	48,191	48,191	48,228	48,221	48,226	48,191
8	Lin318	42,029	42,029	42,180	42,196	42,184	42,029
9	Rd400	15,281	15,281	15,452	15,350	15,430	15,281
10	Fl417	11,861	11,861	11,878	11,868	11,876	11,861
11	Pr439	107,217	107,217	107,561	107,465	107,407	107,217
12	U574	36,905	36,905	37,164	37,165	37,156	36,905
13	Rat575	6,773	6,773	6,840	6,837	6,840	6,773
14	U724	41,910	41,910	42,262	42,252	42,212	41,910
15	Rat783	8,806	8,806	8,900	8,888	8,893	8,806
16	Pr1002	259,045	259,045	261,802	261,805	261,482	259,045
17	Pcb1173	56,892	56,892	57,570	57,432	57,562	56,892
18	D1291	50,801	50,801	51,291	51,199	51,344	50,801
19	Rl1323	270,199	270,199	271,711	271,714	271,818	270,199
20	Fl1400	20,127	20,132	20,231	20,249	20,375	**20,130**
21	D1655	62,128	62,134	64,112	63,001	62,893	**62,134**
22	Vm1748	336,556	336,558	336,719	339,711	339,618	**336,557**
23	U2319	234,256	234,356	235,338	235,975	235,236	**234,350**
24	Pcb3038	137,694	137,696	139,702	139,635	139,706	**137,695**
25	Fnl4461	182,566	182,571	185,546	185,509	185,535	**182,570**
26	Rl5934	556,045	556,056	566,212	566,053	566,167	**556,054**
27	Pla7397	23,260,728	23,261,751	23,800,000	23,800,000	23,800,000	**23,261,749**
28	Usa13509	19,982,859	19,984,989	21,400,000	20,400,000	20,400,000	**19,984,987**
29	Brd14051	469,385	469,396	478,099	478,010	478,610	**469,394**
30	D18512	645,238	645,463	659,457	657,457	658,149	**645,460**

REFERENCES:

[1] Y. Deng, J. Xiong, Q. Wang, "A Hybrid Cellular Genetic Algorithm for the Traveling Salesman Problem", *Mathematical Problems in Engineering*, Vol. 2021, (2021).

[2] D. Applegate, R. Bixby, V. Chvátal, W.J. Cook, "The Traveling Salesman Problem: A Computational Study", *Princeton University Press,* 2007.

[3] K. Yang, X. M. You, S. Liu, H. Pan, "A novel ant colony optimization based on game for traveling salesman problem", *Applied Intelligence* 50 (12), (2020), 4529–4542.

[4] M.R. Garey, D.S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", *Series of Books in the Mathematical Sciences*, first ed., W. H. Freeman & Co., New York, 1979. ISBN: 978-0-7167-1045-5.

[5] K. Helsgaun, "An effective implementation of the Lin–Kernighan traveling salesman heuristic", *European Journal of Operational Research* 126 (1), (2000), 106–130.
https://doi.org/10.1016/S0377-2217(99)00284-2

[6] M. Albayrak, N. Allahverdi, "Development a new mutation operator to solve the traveling salesman problem by aid of genetic algorithms", *Expert Systems with Applications* 38 (3), (2011), 1313–1320. https://doi.org/10.1016/j.eswa.2010.07.006

[7] R.E.M. Al-Khatib, M.A. Al-Betar, M.A. Awadallah, K.M. Nahar, M.M.A. Shquier, A.M. Manasrah, A.B. Doumi, "MGA-TSP: Modernised genetic algorithm for the travelling salesman problem", *International Journal of Reasoning-Based Intelligent Systems* 11 (3), (2019), 215–226.
https://doi.org/10.1504/IJRIS.2019.10019776

[8] A.A. Heidari, S. Mirjalili, H. Faris, I. Aljarah, M. Mafarja, H. Chen, "Harris hawks optimization: Algorithm and applications", *Future Generation Computer Systems* 97, (2019), 849–872. https://doi.org/10.1016/j.future.2019.02.028

[9] S.M. Chen, C.Y. Chien, "Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques", *Expert Systems with Applications* 38 (12), (2011), 14439–14450.
https://doi.org/10.1016/j.eswa.2011.04.163

[10] S. Ahmed, K. K. Ghosh, L. Garcia-Hernandez, A. Abraham, R. Sarkar" Improved coral reefs optimization with adaptive b-hill climbing for feature selection", *Neural Computing and Applications*, (2020), pp. 1-20.

[11] M. Gendreau, G. Laporte, F. Semet, "A tabu search heuristic for the undirected selective travelling salesman problem", *European Journal of Operational Research* 106 (2-3), (1998), 539–545. https://doi.org/10.1016/S0377-2217(97)00289-0

[12] S. Hore, A. Chatterjee, A. Dewanji, "Improving variable neighborhood search to solve the traveling salesman problem", *Applied Soft Computing* 68, (2018), 83–91. https://doi.org/10.1016/j.asoc.2018.03.048

[13] N. Mladenović, R. Todosijević, D. Urošević, "An efficient general variable neighborhood search for large travelling salesman problem with time windows", *Yugoslav Journal of Operations Research* 23 (1), (2013), 19–30. https://doi.org/10.2298/YJOR120530015M

[14] M. Mitchell, "An Introduction to Genetic Algorithms", *The MIT Press*, 1996. ISBN: 9780262631853.

[15] C. Changdar, G.S. Mahapatra, R.K. Pal, "An efficient genetic algorithm for multi-objective solid travelling salesman problem under fuzziness", *Swarm and Evolutionary Computation* 15, (2014), 27–37. https://doi.org/10.1016/j.swevo.2013.11.001

[16] M. Yousefikhoshbakht, "Solving the Traveling Salesman Problem: A Modified Metaheuristic Algorithm", *Complexity*, Vol. 2021. (2021).

[17] G. Vahdati, S.Y. Ghouchani, M. Yaghoobi, "A hybrid search algorithm with Hopfield neural network and genetic algorithm for solving traveling salesman problem", *Proceedings of 2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE)*, Singapore, Vol. 1, February 2010, pp. 435–439. https://doi.org/10.1109/ICCAE.2010.5451917

[18] D.S. Johnson, L.A. McGeoch, "The traveling salesman problem: A case study in local optimization", *E.H.L. Aarts, J.K. Lenstra (Eds.)*, Local Search in Combinatorial Optimization, John Wiley and Sons, London, 1997, pp. 215–310.

[19] G. Reinelt, "TSPLIB – A traveling salesman problem library", *ORSA Journal on Computing* 3 (4), (1991), 376–384. https://doi.org/10.1287/ijoc.3.4.376

[dataset] [20] G. Reinelt, "TSPLIB". http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

[dataset] [21] "National TSPs". http://www.math.uwaterloo.ca/tsp/world/countries.html

[dataset] [22] R. Andri, "VLSI Data Sets". http://www.math.uwaterloo.ca/tsp/vlsi/index.html

[23] J.H. Holland, "Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence", *MIT Press*, 1992. ISBN-10: 0262581116.

[24] O. Belghazi, M. Cherkaoui, "Pitch angle control for variable speed wind turbines using genetic algorithm controller", *Journal of Theoretical and Applied Information Technology* 39 (1), (2012), 6-10.

[25] D.E. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning", *Addison-Wesley Longman Publishing Co*., Boston, 1989. ISBN: 978-0-201-15767-3.

[26] M. Alssager, Z.A. Othman, "Simulated Annealing Algorithm Using Iterative Component Scheduling Approach for Chip Shooter Machines", *Journal of Theoretical & Applied Information Technology* 65 (2), (2014), 480-490.

[27] S.H. Zhan, J. Lin, Z.J. Zhang, Y.W. Zhong, "List-based simulated annealing algorithm for traveling salesman problem", *Computational Intelligence and Neuroscience 2016*, (2016), 1712630. https://doi.org/10.1155/2016/1712630

[28] A.E.S. Ezugwu, A.O. Adewumi, M.E. Frîncu, "Simulated annealing based symbiotic organisms search optimization algorithm for traveling salesman problem", *Expert Systems with Applications* 77, (2017), 189–210. https://doi.org/10.1016/j.eswa.2017.01.053

[29] C. Wang, M. Lin, Y. Zhong, H. Zhang, "Solving travelling salesman problem using multiagent simulated annealing algorithm with instance-based sampling", *International Journal of Computing Science and Mathematics* 6 (4), (2015), 336–353. https://doi.org/10.1504/IJCSM.2015.071818

[30] Y. Lin, Z. Bian, X. Liu, "Developing a dynamic neighborhood structure for an adaptive hybrid simulated annealing – tabu search algorithm to solve the symmetrical traveling salesman problem", *Applied Soft Computing* 49, (2016), 937–952. https://doi.org/10.1016/j.asoc.2016.08.036

[31] M.A.H. Akhand, S.I. Ayon, S.A. Shahriyar, N. Siddique, H. Adeli, "Discrete spider monkey optimization for travelling salesman problem", *Applied Soft Computing* 86, (2020), 105887. https://doi.org/10.1016/j.asoc.2019.105887

[32] M. M. J. Jurjee, H. M. Sarim, N. H. A. Al-Dabbagh, & E. B. Nababan, "A multi-population harmony search algorithm for the dynamic travelling salesman problem with traffic factors", *Journal of Theoretical and Applied Information Technology 95* (2), (2017), 265.

[33] Y. Nagata, S. Kobayashi, "A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem", *INFORMS Journal on Computing* 25 (2), (2013), 346–363. https://doi.org/10.1287/ijoc.1120.0506

[34] H. Faris, I. Aljarah, M.A. Al-Betar, S. Mirjalili, "Grey wolf optimizer: A review of recent variants and applications", *Neural Computing and Applications* 30 (2), (2018), 413–435. https://doi.org/10.1007/s00521-017-3272-5

[35] J. Wang, O.K. Ersoy, M. He, F. Wang, "Multi-offspring genetic algorithm and its application to the traveling salesman problem", *Applied Soft Computing* 43, (2016), 415–423.https://doi.org/10.1016/j.asoc.2016.02.021

[36] A.M. Mohsen, "Annealing ant colony optimization with mutation operator for solving TSP", *Computational Intelligence and Neuroscience 2016*, (2016), 8932896. https://doi.org/10.1155/2016/8932896

[37] C.W. Tsai, S.P. Tseng, M.C. Chiang, C.S. Yang, T.P. Hong, "A high-performance genetic algorithm: Using traveling salesman problem as a case", *The Scientific World Journal 2014*, (2014), 178621.

[38] P.D. Thanh, H.T.T. Binh, B.T. Lam, "New mechanism of combination crossover operators in genetic algorithm for solving the traveling salesman problem", *V.H. Nguyen, A.C. Le, V.N. Huynh (Eds.)*, Knowledge and Systems Engineering, Advances in Intelligent Systems and Computing, Springer, Champ. Vol. 326, 2015, pp. 367–379. https://doi.org/10.1007/978-3-319-11680-8_29

[39] V. Dwivedi, T. Chauhan, S. Saxena, P. Agrawal, "Travelling salesman problem using genetic algorithm", *National Conference on Development of Reliable Information Systems, Techniques and Related Issues (DRISTI 2012), Proceedings published in International Journal of Computer Applications*, 2012, pp. 25–30.

[40] G.A. Croes, "A method for solving traveling salesman problems", *Operations Research* 6, (1958), 791–812.
https://doi.org/10.1287/opre.6.6.791

[41] Y. Zhou, F. He, Y. Qiu, "Dynamic strategy based parallel ant colony optimization on GPUs for TSPs", *Science China Information Sciences* 60 (6), (2017), 068102. https://doi.org/10.1007/s11432-015-0594-2

[42] Y. Zhou, F. He, Y. Qiu, "Optimization of parallel iterated local search algorithms on graphics processing unit", *The Journal of Supercomputing* 72 (6), (2016), 2394–2416.
https://doi.org/10.1007/s11227-016-1738-3

[43] B.A. Mahafzah, A. Sleit, N.A. Hamad, E.F. Ahmad, T.M. Abu-Kabeer, "The OTIS hyper hexa-cell optoelectronic architecture", *Computing* 94 (5), (2012), 411–432.
https://doi.org/10.1007/s00607-011-0177-5

[44] A. Al-Adwan, A. Sharieh, B.A. Mahafzah, "Parallel heuristic local search algorithm on OTIS hyper hexa-cell and OTIS mesh of trees optoelectronic architectures", *Applied Intelligence* 49 (2), (2019), 661–688. https://doi.org/10.1007/s10489-018-1283-2

[45] A. Al-Adwan, R. Zaghloul, B.A. Mahafzah, A. Sharieh, "Parallel quicksort algorithm on OTIS hyper hexa-cell optoelectronic architecture", *Journal of Parallel and Distributed Computing* 141 (2020), 61–73.
https://doi.org/10.1016/j.jpdc.2020.03.015

[46] A. Al-Adwan, B.A. Mahafzah, A. Sharieh, "Solving traveling salesman problem using parallel repetitive nearest neighbor algorithm on OTIS-Hypercube and OTIS-Mesh optoelectronic architectures", *The Journal of Supercomputing* 74 (1), (2018), 1–36. https://doi.org/10.1007/s11227-017-2102-y

[47] M. Abdullah, E. Abuelrub, B.A. Mahafzah, "The chained-cubic tree interconnection network", *International Arab Journal of Information Technology* 8 (3), (2011), 334–343.

[48] S.W. Al-Haj Baddar, B.A. Mahafzah, "Bitonic sort on a chained-cubic tree interconnection network", *Journal of Parallel and Distributed Computing* 74 (1), (2014), 1744–1761.
https://doi.org/10.1016/j.jpdc.2013.09.008

[49] B.A. Mahafzah, M. Alshraideh, T.M. Abu-Kabeer, E.F. Ahmad, N.A. Hamad, "The optical chained-cubic tree interconnection network: Topological structure and properties", *Computers & Electrical Engineering* 38 (2), (2012), 330–345. https://doi.org/10.1016/j.compeleceng.2011.11.023

[50] B.A. Mahafzah, R.Y. Tahboub, O.Y. Tahboub, "Performance evaluation of broadcast and global combine operations in all-port wormhole-routed OTIS-Mesh interconnection networks", *Cluster Computing* 13 (1), (2010), 87–110.
https://doi.org/10.1007/s10586-009-0117-8

[51] B.A. Mahafzah, I.O. Al-Zoubi, "Broadcast communication operations for hyper hexa-cell interconnection network", *Telecommunication Systems* 67 (1), (2018), 73–93. https://doi.org/10.1007/s11235-017-0322-3

[52] A. Al-Adwan, B.A. Mahafzah, A. Aladwan, "Load balancing problem on hyper hexa cell interconnection network", *International Journal of Advanced Computer Science and Applications* 11 (10), (2020), 373–379.