Conference Paper

Solving Travelling Salesman Problem by Using Optimization Algorithms

Suhair Saud, Halife Kodaz, and ismail Babaoğlu

Dept. of Computer Engineering, Faculty of Engineering, Selcuk University, Konya, Turkey

Abstract

This paper presents the performances of different types of optimization techniques used in artificial intelligence (AI), these are Ant Colony Optimization (ACO), Improved Particle Swarm Optimization with a new operator (IPSO), Shuffled Frog Leaping Algorithms (SFLA) and modified shuffled frog leaping algorithm by using a crossover and mutation operators. They were used to solve the traveling salesman problem (TSP) which is one of the popular and classical route planning problems of research and it is considered as one of the widely known of combinatorial optimization. Combinatorial optimization problems are usually simple to state but very difficult to solve. ACO, PSO, and SFLA are intelligent meta-heuristic optimization algorithms with strong ability to analyze the optimization problems and find the optimal solution. They were tested on benchmark problems from TSPLIB and the test results were compared with each other.

Keywords: Ant colony optimization, shuffled frog leaping algorithms, travelling salesman problem, improved particle swarm optimization

Corresponding Author:
Halife Kodaz
hkodaz@selcuk.edu.tr

Received: 14 November 2017
Accepted: 25 December 2017
Published: 8 January 2018

Publishing services provided by Knowledge E

Selection and Peer-review under the responsibility of the IAIT Conference Committee.

1. Introduction

The Travelling Salesman Problem, briefly TSP is a description of a large class of problems known as combinatorial optimization problems. The concept of the problem is as follows: A salesman has a number of cities to visit with a distance or time between two cities, he wants to take the shortest possible route from all these cities so that he does not pass from the same city twice and eventually return to where he left off [1]. TSP is usually simple to explain but very hard to solve, if the number of cities is small, the answer can be easily found by looking at all possible routes and choosing the shorter route, but with the increasing the number of cities, this method will become inefficient and the complexity of solving the problem also increases [2].

Computational techniques stimulated by natural phenomenon were of great interest in the recent years. The natural phenomenon of insects or large animals is studied to develop different computing techniques in last few decades [2]. Various combinatorial optimization problems such as Traveling Salesman Problem (TSP), Job-shop Scheduling

Problem (JSP), and Vehicle Routing Problem (VRP) were also approached by various modern heuristic methods, like ACO, IPSO and SFLA [3].

In this paper we give an overview of some meta-heuristic (ACO, PSO, IPSO and SFLA) algorithms to get the shortest route for the TSP. These algorithms are introduced in Section 2. Section 3 gives the experimental results of ACO, PSO, modified PSO, SFLA and modified SFLA for TSP under the MATLAB tool. Consequently, in Section 4, we conclude the paper with a summarization of results by emphasizing on the importance of this study.

2. Materials and Methods

2.1. Ant Colony Optimization (ACO)

ACO is a probabilistic technique related to artificial intelligence technology and used for designing meta-heuristic algorithms to solve combinatorial optimization problems, the first algorithm called Ant System (AS), it was first introduced and applied to TSP by [4]. Dorigo, first used ant colony algorithms based on mathematical models of ant colony's behaviors on the TSP problem and achieved positive results. ACO is a technique inspired by the way of ant colony that is secreted by pheromone to find the shortest path between food sources and nests. In addition, ant colony algorithms have been used by other researchers to solve different optimization problems [2].

The process of the ACO to solve TSP is briefly explained in the following:

Tour Construction: Initially, the artificial ants are distributed to n cities randomly and determines the ant's positions on different cities with initial value $\tau_{ij}(0)$, at each construction step, ant k applies a probabilistic decision rule. In particular, the probability rule helps ants which states in city i to decide to visit the next neighboring city j, as given in (1) [5] as follows:

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\propto}[\mu_{ij}(t)]^{\beta}}{\sum_{k}[\tau_{ik}(t)]^{\propto}[\mu_{ik}(t)]^{\beta}} & \text{if } j \in J_k(i) \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

otherwise

Where $p_{ij}^{k}(t)$ is the probability of the ant passing from node i to node j, $\tau_{ij}(t)$ is the pheromone value between nodes i and j, μ_{ij}(t) is the heuristic factor between nodes i and j at t moment, $\mu_{ij} = 1/d_{ij}$, d_{ij} is the distance between nodes i and j and $\propto$, and β are two adjustable positive parameters that control the relative weights of the edge pheromone trails and of the heuristic visibility.

After n iterations of this process, every ant completes a tour, a feasible solution of TSP represented by the route of ants which visited all nodes are appeared as a sequence including all serial numbers of cities [4]. At every iteration, all ants should be estimated by means of the objective function. Then, two high-quality ants are chosen for the pheromone deposition on their edges: the first is the best ant in the current iteration, and the second is the global best ant found so far. The pheromone updating is calculated by (2) [5] as follows:

$$\tau_{ij}(t+1) = (1-p)\cdot\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\,\tau_{ij}^{k}(t+1) \tag{2}$$

Where $\Delta\,\tau_{ij}^{k}$(t+1) is calculated by (3),

$$\Delta\,\tau_{ij}^{k}(t+1) = \begin{cases} 1/L^{k}(t+1) & \text{if ant k travels on edge}(i,j) \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Where τ_{ij}(t) is the accumulated pheromone level at iteration counter, $\Delta\,\tau_{ij}^{k}$ is the increase of trail level on edge (i, j) caused by ant k, $L^{k}(t+1)$ is the total length of the ant's tours, and $(1-p)$ is the pheromone decay parameter, $(p) \in (0,1)$, it illustrates the pheromone trail evaporation when the ant selects a city and decides to move. Computing of the optimal path is completed by the algorithm iterates in a certain number of iterations, after the pheromone trail updating process and k ants have travelled through all the cities, the next iteration is $(t+1)$ will update the distances between cities as illustrated by result later.

2.2. Shuffled Frog Leaping Algorithm (SFLA)

Shuffled Frog Leaping Algorithm (SFLA) is one of the evolutionary algorithms, it has proposed firstly by Eusuff, Lansey, and Pasha [6]. It was designed as a real coded population based on meta-heuristic optimization method for combinatorial optimization to achieve an informed heuristic searching a heuristic function (any mathematical function) to seek for the location that has the maximum amount of available food to solve a combinatorial optimization problem. It is based on an evolution of memes carried out by responsive individuals and global exchange of information among the population. Although this algorithm is easy, it has high performance [6]. SFLA is a method that simulates the memetic evolution of a group of frogs leaping in a swamp. The SLFA framework is illustrated in details as follows [6].

2.3. Cycle Crossover and Order Crossover with inversion mutation shuffled frog leaping algorithm (CX-OX-IMSFLA)

This algorithm combines the advantages of genetic-based memetic algorithm (MA) and PSO.This algorithm has few parameters, prompt formation, great capability in global search and easy realization. The SFLA was proved to solve discrete as well as continuous optimization problems but when the SFLA is used to solve the discrete problems, the leaping step needs to be modified and some disadvantages are also appeared in the original algorithm, such as non-uniform initial population, slow convergent rate, limitations in local searching ability and adaptive ability, and premature convergence [7]. For this reason, SFLA is modified and applied to solve TSP as discussed below:

Proposed method considers a feasible travel path as a frog and displays an improved shuffled frog leaping algorithm (SFLA) for solving TSP. To improve the quality of the position of the worst frog, the CXIM operator is applied. CXIM operator synthetically takes advantages of the features of the cycle crossover (CX) and the inversion mutation (IM) during the local search method. To measure the performance of the modified SFLA, CXIM operator is employed in proposed algorithm to solve symmetric TSP and compared with Basic SFLA.

2.3.1. Modified SFLA for TSP

Basic SFLA losses the exploring ability of divergent field and sometimes trapped within a local optima, also the TSP has its own traits, to make a balance between the convergent and divergent property, based on the description of the weaknesses of the basic SFLA which stated above. A modified SFLA to solve TSP is presented, which includes the genetic crossover and mutation property of divergent category.

The modified shuffled frog leaping algorithm by using Cycle Crossover with Inversion Mutation (CXIM) and Order Crossover with Inversion Mutation (OXIM) are discussed in this section in full details.

2.3.2. Expression of the Frog and Strategy for Initial Population

TSP is popularization of the sorting problem which sorts a set of given cities. A possible solution of TSP explains a possible travel path that is normally represented as a series

of characters, each character represents a real city, so the frogs stand for the travel routed here.

The initial population is created randomly to achieve enough diversification, frog i can be represented as $P_i = P_{i1}, P_{i2}, P_{i3}, \ldots P_{iD}$, where D is the quantity of total cities, the decision variable $P_{ij}\ (i = 1,\ 2,\ \cdots,\ F),\ \ (j = 1,\ 2,\ \cdots,\ D)$ is the number corresponding to a real city, then, calculate the fitness (performance) for each frog. Here, the performance function of a frog can be noted as the mutual of the route length corresponding to this frog [8].

2.3.3. Partition

Arrange the frogs in ascending order based upon their fitness, the P Frogs of a sorted population is divided into m subsets (memeplexes) each subset contains n frogs such that $P = m \times n$. Put the first frog into the first memeplex, the second frog into the second memeplex, the m^{th} frog into the m^{th} memeplex, and the $(m+1)^{th}$ frog back into the first memeplex. All the frogs of a population are placed into m different memeplexes, the various memeplexes describe different cultures of the designated frogs [8]. Then P_{best} and P_{worst}, are determined individually at each subset.

2.3.4. Improvement Strategy for the Worst Frog (Local Search)

In the local search, each frog should pass through D different cities, but the new frog will come out as real number and maybe contain repetition, so this research first, uses cycle crossover and inversion mutation, second, two point crossover and inversion mutation operators to generate a new frog as integer number to solve TSP which is an integer programming problem.

2.3.5. Path representation

There are many different procedures to represent the path to optimize TSP, such as binary representation and path representation. In this research, in order to represent a tour of TSP, path representation is used. As mentioned above, a tour is represented as a list of n cities [9]. To solve the combination of TSP with the path representation, crossover and mutation operators are defined:

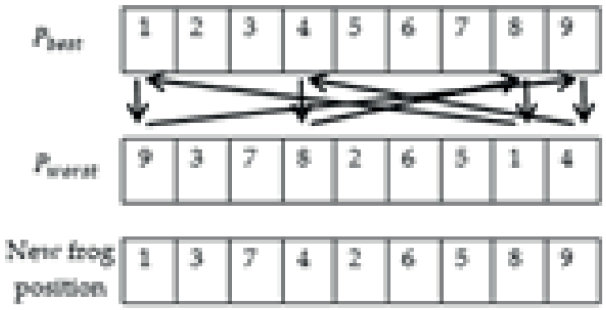

Figure 1: An example of CX method.

2.3.6. Cycle Crossover

There are many types of crossovers, in this research Cycle Crossover (CX) is used to improve the position of worst frog, CX is in common use when processing the strings with integer coding. It was proposed by [10]. It attempts to create an offspring from the parents, each element comes from one parent together with its position, it involves three main steps [11]:

1. Make a cycle of elements from best local frog P_{best}as follows:

a. From P_{best}, start with the first element.

b. Look at the element at the same position in P_{worst}.

c. Go to the position with the same element in P_{best}.

d. Add this element to the cycle.

e. Repeat step (b) through (d) until reaching the first element of P_{best}.

2. Put the elements of the cycle in the blank string on the positions that they are in the P_{best}

3. Take next cycle from P_{worst}.

An example of the CX method is shown in Fig. 1.

So after creating a new frog with CX operator, another frog with mutation operator can be obtained. In this research inversion mutation (IM) operator is used.

2.3.7. Inversion Mutation

The inversion mutation was proposed by [12]. The purpose of mutation is to preserve the genetic diversity of frogs in order to prevent the algorithms from being trapped in a local optima and preventing the population of frogs from becoming too similar to each other. The inversion mutation performs inversion of the substring (route) between two selected cities randomly, removed it from the tour and inserts it randomly in a selected

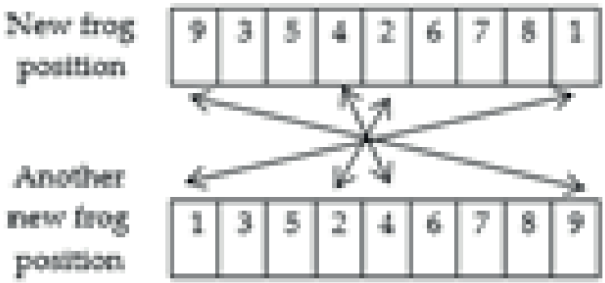

Figure 2: An example of IM operator for mutate the CX method.

position. This means that the sub tour is inserted in reversed order. Figure (2) illustrates the inversion mutation concept [9, 13].

Finally according to their performance (fitness), the best frog can be picked up as appropriate result.

The steps of IM operator are specified as follows:

1. In the new frog, identify the substring randomly.

2. Create another frog that is same as the new frog but the substring is inversed.

2.3.8. Order Crossover

This operator was first achieved by Davis in 1985 [14]. The OX utilizes a property of path representation with important cities ordering [9]. This method constructs a new position by two cut points that are randomly chosen from parents. Here to produce the offspring, the cities between the cut points are exchanged by the cities in the second parent [15].

OX involves three main steps [11]:

1. Choose an arbitrary sub-tour from the P_{best}.

2. Copy this sub-tour to the blank string but sub-tour's position is the same as that in the $P_{best}$to generate a new frog.

3. Copy the rest of the cities from P_{worst} to the blank string, this excluding the cities that were taken at the sub-tour.

- Choosing the cities that follow the sub-tour end,

- Insert them in their order from worst frog vector,

- Once the end of the new frog position vector reached, fill it with the rest of the cities that are at the worst frog vector.

An example of the OX is shown in Fig. 3.

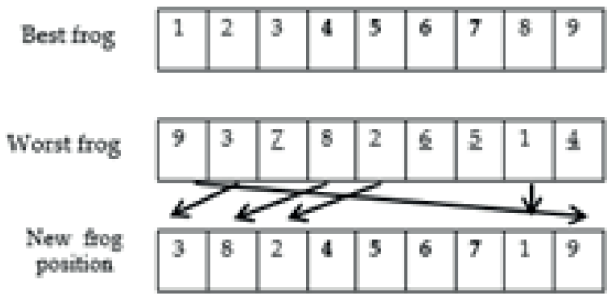

Figure 3: An example of OX method.

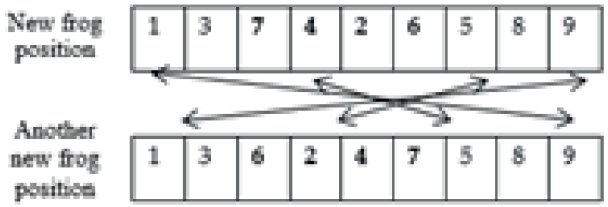

Figure 4: An example of IM operator for mutate OX method.

2.3.9. Inversion Mutation

Another new frog positions with inversion mutation (IM) operator can be obtained to prevent the algorithms from being trapped in a local optima and preventing the population of frogs from being too similar to each other [9]. This will give appropriate result.

The steps of IM operator are specified as follows:

1. In the new frog position, select the sub-tour that inherits from the best frog P_{best}.

2. Create another frog that is same as the new frog but the sub-tour has been inversed.

The inversion mutation concept is illustrated in Figure (4).

2.4. Particle Swarm Optimization (PSO)

Particle swarm optimization has appeared as a powerful optimization developed by Kennedy and Eberhart in 1995 [16, 17], They influenced by the work of Heppner and Grenard, which included the analogy of corn quests in the 1990 [18]. PSO inspired by the social behavior of bird flocking and fish schooling. Birds, fish and animal flocks; impressive production, their movement is synchronized and they move without collision. Birds fly and they maintain in a certain distance between neighbors [19].

The basic PSO algorithm has several phases including Initialization, Evolutionary phases, Velocity Updating and Position Updating. These phases are described in details as follows:

1. Assign the random position and velocity values to the vectors for the population of

size D.

2. Calculate the fitness function for each particle.

3. Identify the best fitness (least objective) among the neighbors of the whole swarm and keep it as a G_{best}.

4. Determine the objective fitness of each particle that it has ever possessed and set it as a P_{best}.

5. Update the position and velocity information for each particle at each time step according to (4) as follows:

$$V_i^{k+1} = w.v_i^k + c_1 r_1 \left(P_{ibest} - x_i^k\right) + c_2 r_2 \left(G_{best} - x_i^k\right)$$
$$X_i^{k+1} = X_i^k + V_i^{k+1} \quad (4)$$

6. The current fitness value is compared with its own best value (P_{best}). If the current fitness value is better than P_{best}, it will be replaced with current objective value and position vector respectively.

7. G_{best} are updated, that is, if the current best fitness of the whole swarm is fitter than G_{best}, then it will be replaced with current best objective and it's corresponding position vector respectively.

8. Repeat steps 2 – 7.

9. If stopping criterion is reached or if the number of iterations has reached it's maximum, the loop is terminated. If not, go back to step 5

10. Loop end.

2.5. Improved Particle Swarm Optimization (IPSO)

As mentioned above, PSO algorithm is a nature-inspired technique originally designed as a simulation of a simplified social system. PSO is an evolutionary computation technique developed successfully in recent years and can be used to continuous and discrete optimization problems to find optimal solutions through local and global models.

The method described above is suitable for problems of continuous value but it can't be used directly to problems of discrete value such as TSP. In [20], they redefined the

basic PSO algorithm by suggesting new concepts inspired by the 'Swap operator' and 'Swap sequence', therefore in this paper, the TSP tried to be solved by PSO in another way.

The suggested algorithm is one of the most important algorithm that has the automatic equilibrium ability between global and local searching capabilities and achieves better convergence with increase the probabilities of finding a minimum solution of TSP. In the initial phase of the algorithm, it starts with generating the population of particles randomly to compose the particle swarm. For a D-dimensional search space, the position of the i^{th}particle is represented as X_i = (X_{i1}, X_{i2}, ... X_{iD}), i.e in proposed method the process of particle formation occurs on the traveling salesman strategy as follows: each particle is represented as a series of nodes such as 1, 2, 3, 4 and 5, this series means salesman visits node 1 then node 2 then node 3 then node 4 then node 5 and go back node 1. The arrangement of node visiting done by PSO can be obtained by calculating a distance of nodes sequence which is sorted out to obtain the least distance [21]. For each particle, the cost of TSP mentioned above is calculated by objective function. Each particle preserves a memory of its previous best position P_{best}= (P_{i1}, P_{i2}, ... P_{iD}) and the best one among all the particles in whole population is represented as P_{gbest}= (P_{g1}, P_{g2}, ... P_{gD}).Also the position of P_{best} and P_{gbest} must be kept. Then the iteration starts and increased until it reaches the maximum value. At each iteration, the fitness of each particle is calculated and compared it's value with the fitness of current P_{best}. If $F(X_i)$ is less than $F(P_{best})$, then update the cost function and position, as well as for P_{gbest}. Later four different hybrid approaches have to be done to find the shortest path, these are:

- By P_{best} : The position of each particle is shuffled by assigning two integer parameters (pointer1) and (pointer2) to select randomly two cities from the P_{best} vector to get l sub-vector. The new P_{best} contains an arrangement of cities in the same manner with shuffling the new sub-vector starting with pointer1 and ending with pointer2.This step should ensure that cities aren't repeated.

- By P_{gbest} : By two integer parameters (pointer1) and (pointer2), a sub-vector extracted from P_{gbest} randomly starting with (pointer1) and ended with (pointer2) and then it can be inserted at the end of the main vector. This step should ensure that cities aren't repeated.

- By employing shuffling process by P_{gbest}, information can be swapped between two particles to have the ability to fly to the new search area.

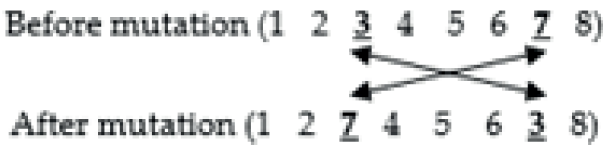

Figure 5: Before and after the PSM.

- Improving the whole position of population: After shuffling process, apply mutation operator, in general, there are different types of mutations operators used in evolution strategy, in our proposed algorithm pair-wise swap mutation (PSM) or exchange mutation (EM) were used [22-24].

PSM operator invented by Banzhaf in 1990. The concept of this operator is to selects two cities randomly from the tour and swapped them [24]. Sometimes, this technique is also called interchange mutation, or exchange mutation or random swap. For example, consider the tour represented by [25]:

In this case, the location of city 3 and city 7 will be swapped as in Fig. 5.

After that, at each generation, the distance is calculated and updated if the new cost function is less than the previous one.

The city which yields the shortest distance is selected, the process continues for many generations until the tour is completed.

3. Experimental Results and Discussion

Following the above mentioned algorithms, all the simulations were completed on a Windows XP 32-bit operating system with a 1.70 GHz processor and 4.00 GB RAM memory, all the algorithms were developed with the help of MATLAB 6.5 tool. All the mentioned heuristics algorithms above were tested on well-known datasets from the TSPLIB that is found at [26]. Our approach was tested on paths of sizes n = (30, 51, 52, 70, 76, 100), where n is the cities number. These cities were chosen from small dataset to large dataset to enable us of making comparisons related to scientific literature. For ant colony, the chosen parameters are α= 1, β= 2, rho (pheromone evaporation rate)= 0.1 and the maximum number of iterations was set to 1000. However, if the algorithm doesn't give updated result specially when the tour contains large amount of cities, the number of ants may be increased and taking last updated result as starting one. Ants number can be (m = 100, 500, 1000). It can be seen easily that the optimized result is changed when number of ants increased above 100.

For particle swarm optimization algorithm, the basic version gave us an approximate solution to the problem but it wasn't an optimal solution because TSP have the discrete

feature reverse PSO algorithm that solved continues optimization problem. For this reason, we tried to modify basic PSO to evaluate the optimal solution. For basic PSO, set the number of particles (birds) equal to the number of cities.Increasing or decreasing the number of birds doesn't affect the result.

Choosing the algorithm parameters as follows, w (inertia weight) = 0.7, c1 (cognitive parameter) = 1.8, c_1 (social parameter)= 4-c_2, c_1 + c_2 must equal to 4, maximum number of iterations equal to 1000 and defining the maximum and minimum position value in order to limit the velocity value.

For improved PSO, the parameters involved in IPSO include the maximum number of iterations to be 1000 and it may be increased according to the cities tour size. The population number equal to number of city, c_1 and c_2 randomly selected and must be generated as discrete uniform random numbers. In SFLA, for each N-city TSP problem, the parameters are set as follows: there are 10 memeplexes (m), n is the number of frogs within each memeplex, so the number of population (P) is m*n, each memeplex will be divided into submemplexes and q represents the number of frogs in a submemeplex. All results were obtained by running the program over 1000 independent runs. Table 1 shows the comparison test results between proposed algorithms. According to equation (5) given below, error values were calculated to show the efficiency of the algorithms used. In table 1, the bold results in mean values represent the best performance among others, the best result were obtained by using IPSO on TSP instances Oliver30, St70, Eil76 and KroA100, about Eil51 and Berlin52 the results were reasonable.

$$\text{Error} = \frac{\text{Best Value-Optimal Value}}{\text{Optimal Value}} \tag{5}$$

Table 1: The comparison of ACO, PSO, IPSO, SFLA, OXIMSFLA and CXIMSFLA with each other.

Problem	Oliver30 Optimal: 423			Eil51Optimal: 426			Berlin 52Optimal: 7542			St70Optimal: 675			Eil76Optimal: 538			KroA100Optimal: 21282		
Method	Best	Mean	Error	Best	Mean	Error	Best	Mean	Error	Best	Mean	Error	Best	Mean	Error	Best	Mean	Error
ACO	426	525	0.007	443	516	0.040	7549	9385	0.001	707	888	0.047	573	659	0.065	22388	28655	0.052
PSO	763	892	0.804	908	1313	1.131	17296	22206	1.293	2009	3411	1.976	1662	1975	2.089	113174	191394	4.318
IPSO	424	441	0.002	464	543	0.089	7816	8723	0.036	755	871	0.119	584	641	0.086	24596	28385	0.156
SFLA	741	1357	0.752	1169	1703	1.168	19865	30598	1.634	2615	3759	2.874	1904	2580	2.539	128520	175413	5.039
OXIMSFLA	434	517	0.026	534	593	0.254	8362	9987	0.109	892	1004	0.321	733	819	0.362	37212	41371	0.749
CXIMSFLA	556	699	0.314	671	812	0.575	12266	15109	0.626	1355	1734	1.007	1072	1326	0.993	58069	78451	1.729

Following the above mentioned success tests of the application, the running time consumed by algorithm is presented for the data sets in different sizes. Oliver30 as a small size data group, Eil51, Berlin52 as a medium data group, these groups converges

quickly and requires less computational time due to their small complexity while St70 and KroA100 as a large data group, it requires more computational time due to it's complexity. Table 2 shows that the average running time of the applications of each data set.It can be seen that time is increased when increasing the size of the tour.

Table 2: Running times for ACO, PSO, IPSO, SFLA, OXIMSFLA and CXIMSFLA implementation.

Problem	Time (Second)					
	ACO	PSO	IPSO	SFLA	OXIMSFLA	CXIMSFLA
Oliver30	19	9	104	442	1338	706
Eil51	194	9	259	1136	16241	5147
Berlin52	276	15	469	1150	21907	5732
St70	1678	28	1058	1859	68119	17638
Eil76	2035	28	3036	1771	99104	28007
KroA100	4516	45	9426	3042	197264	52365

4. Conclusions

This paper presents the performance of different types of optimization techniques used in artificial intelligence (AI) by making a comparison of six evolutionary-based search methods. These are ACO, PSO, IPSO, SFLA, OX-IMSFLA and CX-IMSFLA which using a crossover and mutation operator.

OX-IMSFLA and CX-IMSFLA are a new trial that get the approximate solution by using SFLA, these methods achieved near-optimal result in symmetric small-sized TSP instances such as Oliver30, Eil51, and Berlin52, but in big-sized TSP instances such as St70, Eil76 and KroA100, large execution time was consumed.

These algorithms were used to solve TSP which is one of the popular and classical route planning problems of research and is accounted as one of the widely known of combinational optimization. Matlab program were written to implement each algorithm to solve our problem.

Finally, from this paper, it can be concluded that for NP-hard optimization problems and complicated search problems, meta-heuristic methods are very good choices for solving these problems. This conclusion based on solution quality and run time comparison.The comparison is meaningful and done between meta-heuristics methods. Algorithms were tested on benchmark problems for TSPLIB and the test results were compared with each other.

Acknowledgement

We like to express our sincere thanks to Scientific Research Project of Selçuk University.

References

[1] F. Greco, "Travelling Salesman Problem, I-Tech," ed: Croatia, 2008.

[2] H. Dikmen, H. Dikmen, A. Elbir, Z. Ekşi, and F. Çelik, "Gezgin Satici Probleminin Karinca Kolonisi ve Genetik Algoritmalarla Eniyilemesi ve Karsilastirilmasi," Journal of Natural & Applied Sciences, vol. 18, 2014.

[3] X. Yan, C. Zhang, W. Luo, W. Li, W. Chen, and H. Liu, "Solve traveling salesman problem using particle swarm optimization algorithm," International Journal of Computer Science, vol. 9, pp. 264-271, 2012.

[4] M. Dorigo, V. Maniezzo, and A. Colorni, "The ant system: Optimization by a colony of cooperative agents," 1996.

[5] Z. C. S. S. Hlaing and M. A. Khine, "Solving traveling salesman problem by using improved ant colony optimization algorithm," International Journal of Information and Education Technology, vol. 1, p. 404, 2011.

[6] M. Eusuff, K. Lansey, and F. Pasha, "Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization," Engineering optimization, vol. 38, pp. 129-154, 2006.

[7] X.-h. Luo, Y. Yang, and X. Li, "Solving TSP with shuffled frog-leaping algorithm," in Intelligent Systems Design and Applications, 2008. ISDA'08. Eighth International Conference on, 2008, pp. 228-232.

[8] M. Wang and W. Di, "A modified shuffled frog leaping algorithm for the traveling salesman problem," in Natural Computation (ICNC), 2010 Sixth International Conference on, 2010, pp. 3701-3705.

[9] P. Larranaga, C. M. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic, "Genetic algorithms for the travelling salesman problem: A review of representations and operators," Artificial Intelligence Review, vol. 13, pp. 129-170, 1999.

[10] I. Oliver, D. Smith, and J. R. Holland, "Study of permutation crossover operators on the traveling salesman problem," in Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms: July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA, 1987.

[11] A. Umbarkar and P. Sheth, "Crossover Operators In Genetic Algorithms: A Review," ICTACT journal on soft computing, vol. 6, 2015.

[12] D. B. Fogel and J. W. Atmar, "Comparing genetic operators with Gaussian mutations in simulated evolutionary processes using linear systems," Biological Cybernetics, vol. 63, pp. 111-114, 1990.

[13] H. H. Chieng, "A genetic simplified swarm algorithm for optimizing n-cities open loop travelling salesman problem," Universiti Tun Hussein Onn Malaysia, 2016.

[14] L. Davis, "Job shop scheduling with genetic algorithms," in Proceedings of an international conference on genetic algorithms and their applications, 1985.

[15] N. Kumar and R. K. Karambir, "A comparative analysis of pmx, cx and ox crossover operators for solving traveling salesman problem," International journal of Latest Research in science and technology, vol. 1, 2012.

[16] F. Huilian, "Discrete particle swarm optimization for TSP based on neighborhood," Journal of Computational Information Systems, vol. 6, pp. 3407-3414, 2010.

[17] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on, 1995, pp. 39-43.

[18] F. Heppner and U. Grenander, "A stochastic nonlinear model for coordinated bird flocks," The ubiquity of chaos, pp. 233-238, 1990.

[19] I. Yaman, "Portf□y optimizasyonunda değiştirilmiş parçacık sürü optimizasyonu yaklaşımı," Master, Statistics Department, Hacettepe University, 2014.

[20] K.-P. Wang, L. Huang, C.-G. Zhou, and W. Pang, "Particle swarm optimization for traveling salesman problem," in Machine Learning and Cybernetics, 2003 International Conference on, 2003, pp. 1583-1585.

[21] K. Premalatha and A. Natarajan, "Hybrid PSO and GA for global maximization," Int. J. Open Problems Compt. Math, vol. 2, pp. 597-608, 2009.

[22] D. Tang, S. Dong, X. Cai, and J. Zhao, "A two-stage quantum-behaved particle swarm optimization with skipping search rule and weight to solve continuous optimization problem," Neural Computing and Applications, vol. 27, pp. 2429-2440, 2016.

[23] C. Ratanavilisagul and A. Kruatrachue, "A modified particle swarm optimization with mutation and reposition," Int J Innov Comput Inform Control, vol. 10, pp. 2127-2142, 2014.

[24] W. Banzhaf, "The "molecular" traveling salesman," Biological Cybernetics, vol. 64, pp. 7-14, 1990.

[25] H. H. Chieng and N. Wahid, "A Performance Comparison of Genetic Algorithm's Mutation Operators in n-Cities Open Loop Travelling Salesman Problem," in Recent Advances on Soft Computing and Data Mining, ed: Springer, 2014, pp. 89-97.

[26] G. Reinelt, "TSPLIB http://www.iwr.uni-heidelberg.de/groups/comopt/software," TSPLIB95, 1995.