Indonesian Journal of Electrical Engineering and Computer Science
Vol. 12, No. 3, December 2018, pp. 1054~1062
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v12.i3.pp1054-1062

Improved Chicken Swarm Optimization Algorithm to Solve the Travelling Salesman Problem

Fayçal Chebihi, Mohammed Essaid Riffi, Amine Agharghor, Soukaina Cherif Bourki Semlali, Abdelfattah Haily
Chouaib Doukkali University, Morocco

Article Info

Article history:

Received Apr 22, 2018
Revised Jul 16, 2018
Accepted Aug 30, 2018

Keywords:

2-OPT
CSO
DCSO
NP-HARD
TSP
TSPLIB

ABSTRACT

This paper proposes a novel discrete bio-inspired chicken swarm optimization algorithm (CSO) to solve the problem of the traveling salesman problem (TSP) which is one of the most known problems used to evaluate the performance of the new metaheuristics. This problem is solved by applying a local search method 2-opt in order to improve the quality of the solutions. The DCSO as a swarm system of the algorithm increases the level of diversification, in the same way the hierarchical order of the chicken swarm and the behaviors of chickens increase the level of intensification. In this contribution, we redefined the basic different operators and operations of the CSO algorithm. The performance of the algorithm is tested on a symmetric TSP benchmark dataset from TSPLIB library. Therefore, the algorithm provides good results in terms of both optimization accuracy and robustness comparing to other metaheuristics.

Corresponding Author:

Fayçal Chebihi,
Chouaib Doukkali University, Morocco.
Email: chebihi.f@ucd.ac.ma

1. INTRODUCTION

Traveling salesman problem (TSP) is one of the most extensively studied problems [1] in operational research. The aim of the problem is to find the shortest path linking a set of cities; the Salesman should cross each city only once and return to the city of departure in order to close the cycle.

The TSP is NP-hard problem and its complexity increase depends on the number of cities included. If we consider n is the number of cities, then the number of the possible solutions is:

$$\frac{(n-1)!}{2} \quad (1)$$

Taking into consideration that a computer finds a possible solution in 1μs time process. The total time to obtain all possible solutions is described by the following Table 1.

Table 1. The Computation Time Estimated

Number of Cities	Numbet of Tours	Time in Year
20	6,08E+16	2
25	3,1E+23	9,84E+6
30	4,4E+30	1,4E+14
35	1,48E+38	4,68E+21
40	1,02E+46	3,23E+29

This kind of problem is very common in industry business [2], [3] such as X-ray crystallography while analyzing crystals structure or even in more real life situations logistics [4] such as school transportation.

The difficulty of this problem has generated much interest to solve TSP, starting with the implementation of heuristics methods such as Tabu Search (TS) [5], Genetic Algorithm (GA) [6], Heuristic Approach [7], Greedy Randomize Adaptive Search Procedure (GRASP) [8], and Simulated Annealing (ST) [9]. Recently, several studies use of the bio-inspired algorithms using swarm intelligence methods [10] such as: ant colonies optimization (ACO) [11], particle swarm optimization (PSO) [12], [13] bee colonies optimization (BCO) [14], harmony search algorithm (HS) [15], [16], bat-inspired algorithm (BA) [17], [18],cuckoo search (CS) [19], [20] and a bio-inspired hunting search algorithm (HUS) [21].

The metaheuristic Chicken swarm optimization (CSO) is a bio-inspired behavior of chicken. It was introduced in 2014 by XIAN BING MENG [22], and proves its efficiency to solve some continued optimization problem, but it is not possible to use it to solve the combinatorial optimization problem. The aim of this paper is to propose a novel adaptation of the CSO metaheuristic to solve the combinatorial optimization problems by redefining operations and operators. To prove the efficiency of the proposed adaptation, the adapted CSO is applied to solve some benchmark instances of the traveling salesman problem. The obtained results are compared to the best solutions exiting in the literature.

The rest of the paper is organized as follows: the second section describes the Traveling Salesman Problem, the third section introduces the Chicken Swarm Optimization Algorithm, the forth section presents our proposed adaptation of the CSO algorithm to solve TSP, the fifth section shows the experimental and numerical results, and finally the last section is a conclusion.

2. TRAVELING SALESMAN PROBLEM

The traveling salesman problem [23] was first introduced by the Italian Mathematician Karl Mengre in 1930 as a given list of cities along with the cost of travel between each pair of them. The aim of this study is to find the shortest path, which allowed visiting each city once starting and ending in the same city. Given this simple formulation, it might be possible that the problem could have an equally simple solution. It appears that this is not the case even if the problem is easy to express and interpreted. To the present day there is no efficient solution to the TSP has been found. However, the problem has inspired several mathematicians, computer scientists and a host of non-professional researchers. The problem can be mathematically formulated as: Let G = (V, E, W) be a weighted graph with V = {v1, v2 … vn}, then the TSP in G can be represented as:

Let G = (V, E, W) be a weighted graph with V = {v1, v2 … vn}, then the TSP in G can be represented as;

$$\text{minimize } \sum_{e\in E} \boldsymbol{w(e).x_e} \text{ subject to: } \begin{cases} \sum_{e\in\delta(\{v_i\})} x_e = 2, \forall v_i \in V \\ \sum_{e\in\delta(S)} x_e \geq 2, S \subset V, \quad S \neq \phi \\ x_e \in \{0,1\}, \forall e \in E \end{cases} \tag{2}$$

3. CHICKEN SWARM OPTIMIZATION

The Chicken swarm optimization (CSO) offers a swarm optimization based on the chicken's natural behavior in a swarm. A hierarchical order is established in the swarm. The chickens with the highest fitness values are identified as roosters and those with the worst fitness values are chicks. Meanwhile, those in the middle are hens. The swarm is divided into groups; each group contains a rooster, a couple of hens and chicks and is created randomly as described earlier [22].

The rooster with the best fitness value can look for food in a wider range of places.

$$x_{i,j}^{t+1} = x_{i,j}^{t} \times (1 + Randn(0, \sigma^2)) \tag{3}$$

$$\sigma^2 = \begin{cases} 1, if \ f_i \leq f_j \\ \qquad\qquad k \in [1, N], k \neq i \\ e^{\left(\frac{(f_k - f_i)}{|f_i| + \varepsilon}\right)}, otherwise \end{cases} \tag{4}$$

Hens can randomly steal good food from the other chickens.

$$x_{i,j}^{t+1} = x_{i,j}^{t} + S1 \times Rand \times (x_{r1,j}^{t} - x_{i,j}^{t}) + S2 \times Rand \times (x_{r2,j}^{t} - x_{i,j}^{t}) \quad (5)$$

$$where\ S1 = e^{\left(\frac{(f_i - f_r 1)}{(|f_i| + \varepsilon)}\right)} \quad (6)$$

$$And\ S2 = e^{(f_{r2} - f_i)} \quad (7)$$

Chicks look for food around their mothers

$$x_{i,j}^{t+1} = x_{i,j}^{t} + FL \times (x_{m,j}^{t} - x_{i,j}^{t}) \quad (8)$$

Finally, the algorithm of CSO is represented as follow:

Chicken Swarm Optimization Algorithm

```
Initialize a population of N chickens and define the related parameters
Evaluate the N chickens' fitness values, t=0
While (t < Max_Generation)
  If (t % G == 0)
    Rank the chickens' fitness values and establish a hierarchical order in the swarm
    Divide the swarm into different groups, and determine the relationships between chicks and mother hens
in a group
  End IF
  For i=1 : N
    If i== rooster Update its solution/location using equation (3) EndIf
    If i== hens Update its solution/location using equation (5) EndIf
    If i== chicks Update its solution/location using equation (8) EndIf
    Evaluate the new solutions
    If the new solution is better than its previous one update it
  End for
End While
```

4. A NOVEL DISCRETE CHICKEN SWARM OPTIMIZATION TO SOLVE TRAVELING SALESMAN PROBLEM

This paper proposes a novel adaptation of the Chicken Swarm Optimization (CSO) to solve the Traveling Salesman Problem. This adaptation is established by the redefinition of operators and operations and respects the general process of the CSO algorithm. This section describes the different proposed improvements of operator and operations.

4.1. Operator Improvement

The position X_i of a selected chicken$_i$ is the solution represented by a Hamiltonians cycle of the n cities $V = \{v_1, v_2, \dots, v_n\}$.$X_i = [v_1, v_2, \dots, v_n]$ $where$ $(i \in [1, \dots, n])$

4.2. Operations Improvement

4.2.1. The Chicken Movement

There are 3 types of chickens (hens, chicks and roosters). Each one has a determined process of movement that can be determined as follows:

a) Roosters

Roosters that have better fitness value can look for food in a larger space. This concept is represented by Equation (3). Based on these preps, we can define this movement as:

$$x_i^{t+1} = x_i^t \otimes Randn(0 - \sigma^2) \quad (9)$$

Where $\otimes$ means a self-permutation and $Randn(O, \sigma^2)$ defines a random number of permutations. An example of this operation is represented as follows:

Let's X_i^t = [1,2,3,4,5] and Rand = 2

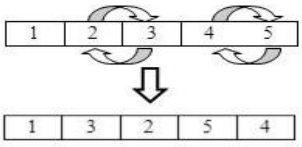

Figure 1. Example of movement of a rooster

The new solution becomes: X_i^{t+1} = [1,3,2,5,4]

b) Hens

Hens follow their group-mate rooster to look for food. However, they randomly steal food from other chickens. These movements are defined in Equation (2). In this adaptation, we represent the movement of hens following the group-mate rooster as a local search around the rooster. In the same way, we represent the movement of hens while stealing food from other chickens as a local search around the chicken. The movement equation of hens become:

$$x_i^{t+1} = x_i^t \oplus Rand \otimes (x_{r1}^t \,!\; x_i^t) \oplus Rand \otimes (x_{r2}^t \,!\; x_i^t) \quad (10)$$

Where A⊖B presents a list of permutations to go from solution B to solution A.
Example: Let's A= [1,2,3,4,5] and B= [2,4,3,1,5]

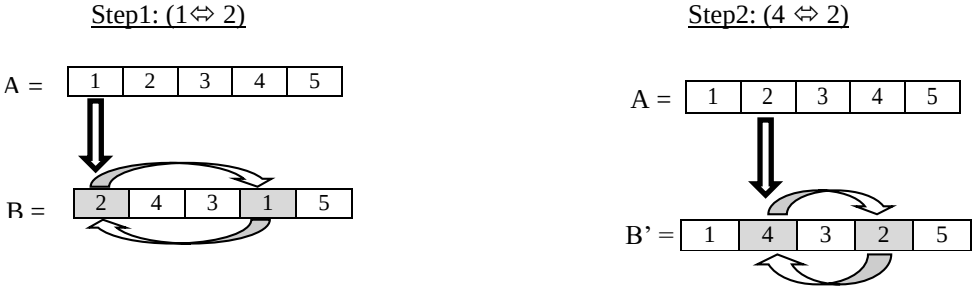

Figure 2. Example of movement of a hens step 1 and step 2

Finally, to go from solution A to solution B the list of permutation is: A⊖B = [{1,2}, {4,2}].
The ⊕ operator indicates that we add the next operation to the new solution created by the last operation.

c) Chicks

The chicks find the food around their mother. This concept presents a fatal disadvantage because the chicks can only learn from their mothers. Therefore, the chicks can easily fall into local minimum. Dinghui [24] proposes a new equation to go through this problem by adding the probability of learning from the main rooster and a self-learning operation. The new equation becomes:

$$x_i^{t+1} = w \otimes x_i^t \oplus FL \otimes (x_m^t \,!\; x_i^t) \oplus C \otimes (x_r^t \,!\; x_i^t) \quad (11)$$

Where w is a self-permutation parameter that indicates the number of permutations and FL is a learning factor which means that the chick learn from its mother in the same way C is a learning factor from the rooster.

4.3. The Neighborhood

To improve the solution quality, neighborhood methods are required. This article proposes a 2-opt local search to improve the quality of the solution proposed by the DSCO algorithm.

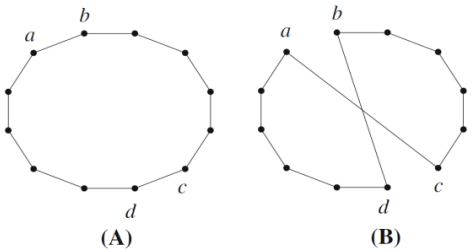

Figure 3. Move from solution (B) to solution (A) with a simple permutation

he 2-opt movement causes a small perturbation to the solution in order to find a good solution in its neighborhood. This operation is demonstrated in Figure 3.

4.4. Discrete CSO Algorithm

Step 1: Initialize a population of N chickens and define related parameters (N the number of chickens in the swarm, Rand [0, 1], r1 [1… N] is an index of rooster r2 r1 [1, …, N] is an index of chickens, FL [0,1], C and w.

Step 2: Use a 2-opt local search to improve the quality of solutions

Step 3: Evaluate The N chicken's fitness values at t=0 and save the global best solution.

Step 4: Rank the chickens and establish a hierarchal order in the swarm and improve the solution using 2-opt local search.

Step 5: Randomly divide the swarm into different groups and determine the relationship between the chicks and the mother-hens in a group.

Step 6: Find a new solution by updating the position of each rooster, hens and chicks using the new Equations (9), (10) and (11).

Step 7: Update the new solution when it is better than the previous one.

Step 8: Return to step 4 until the maximum number of iterations is reached.

5. EXPERIMENTAL RESULTS

This section illustrates performance tests of CSO algorithm on Euclidean instances of TSPLIB. All experiments are performed on an Intel computer processor (R) Core (TM) i7-6500 CPU @ 2.5GHz @ 2.60 GHz and 16 GB of RAM. The program is coded in the programming language C # Visual Studio 2015 and for each instance TSPLIB we test 100 times.

To run the program, a list of parameters has been set. Table 2 shows the values of the parameters used:

Table 2. The Parameters Values

Parameters	Values
PS (population size)	100
RN (Number of roosters)	2%
HN (Number of hens)	20%
CN (Number of chicks)	78%
G (Number of tours to update the algorithm)	2
C (Rooster learning factor)	0,4
FL (Hens learning factor)	0,4
W (self-learning factor)	0.9

5.1. Parametric Analysis

There are eight parameters in the DCSO algorithm. The purpose of the algorithm is that the hens must look for a new solution around a good one represented by the rooster. As a result, the number of roosters must be higher than the number of hens (NR > NH). Similarly, the chicks are seeking a new solution

around their mothers, and for a good intensification, the number of chicks must be higher than the number of hens (NC > NH). If the value of G is very high, the algorithm cannot converge quickly to the optimal solution. Otherwise, if the value of G is very low, the algorithm may fall into a local optimal. After several tests, G=2 may achieve a good result in a much reduced time. As for G, the value of this parameter has a significant impact on the result. Furthermore, to avoid the problem of falling into minimum optimal in the chick's movement, the self-learning parameter gives to the chick the possibility to find a good solution in a bigger space of solutions by the value of w=0,9. The FL parameters give the chicks the possibility to learn from the mother. Moreover, the best value must be a random number between 0,4 and 1 (FL∈ [0.4,1]). In order to give more robustness to the algorithm, the chicks can also learn from roosters using a C parameter with a learning factor randomly chosen between 0.4 and 1 (C∈ [0.4,1]).

Figure 4 shows the evolution of the average running time for 100 executions while varying the parameter G and the size of the population using TSPLIB instance with a different number of the city (ST70 and KroA100).

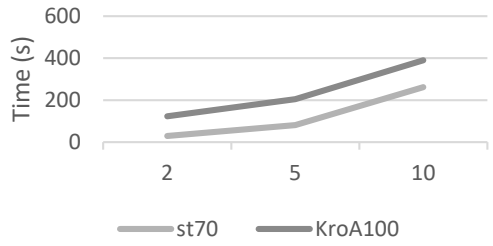

Figure 4. Run time obtained by varying G parameter

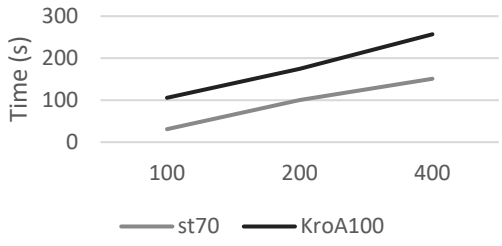

Figure 5. Run time obtained by varying population size

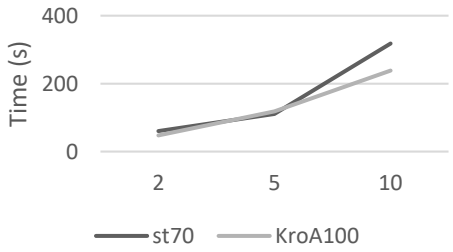

Figure 6. Run time obtained by varying the number of roosters

The results reveal that there is an increase in the execution time when increasing the value of the parameter G on Figure 2 and the size of the population on Figure 5.

For more details, other experiments have been performed to detect the best value of the parameters RN, HN and CN.

The experiments in Figure 6 show that the percentage of the rooster should be low to ensure faster convergence.

Table 3 shows the numerical results obtained when applying the DCSO to the TSP using some TSPLIB instances. The first column contains the name of the instance, the second column contains the number of nodes in the instance, the third column contains the optimal solution found in TSPLIB Library, and the fourth column contains the best solution obtained by the DCSO algorithm. Moreover, the fifth column contains the worst solution, the sixth column contains the average solution, the seventh column denotes the standard deviation of solution obtained over 30 independent runs, the eighth column contains the percentage deviation of the average solution over 30 independent runs, the ninth column contains the percentage deviation of the best solution in 30 independent runs, the tenth column is represented by tow parameters. The number for optimal solution (C_{opt}) and the number of solution (over 30 runs) for which the deviation from optimal solution is less than or equal to 1, and the last column represents the best time

obtained by the DCSO algorithm in 30 independent runs. The algorithm stops when the best solution is found or if the execution time exceeds 3000s.

Table 3. Numerical Results Obtained by DCSO Applied to Some TSP Instances of TSPLIB

Instance	Size	Opt	Best Sol	Worst Sol	Average	PDav (%)	PDbest (%)	$C_{1\%}/C_{opt}$	Time (s)
Eil51	*51*	*426*	***426***	***426***	***426***	***0.00***	***0.00***	***30/30***	*0,32*
Berlin52	*52*	*7542*	***7542***	***7542***	***7542***	***0.00***	***0.00***	***30/30***	*0,09*
St70	*70*	*675*	***675***	***675***	***675***	***0.00***	***0.00***	***30/30***	*0,04*
Eil76	*76*	*538*	***538***	*541*	*239.5*	*0.27*	***0.00***	*30/21*	*5,78*
Pr76	*76*	*108159*	***108159***	***108159***	***108159***	***0.00***	***0.00***	***30/30***	*8,35*
Rat99	*99*	*1211*	***1211***	***1211***	***1211***	***0.00***	***0.00***	***30/30***	*11,79*
KroA100	*100*	*21282*	***21282***	***21282***	***21282***	***0.00***	***0.00***	***30/30***	*0,05*
KroB100	*100*	*22141*	***22141***	***22141***	***22141***	***0.00***	***0.00***	***30/30***	*3,17*
KroC100	*100*	*20749*	***20749***	***20749***	***20749***	***0.00***	***0.00***	***30/30***	*2,68*
KroD100	*100*	*21294*	***21294***	***21294***	***21294***	***0.00***	***0.00***	***30/30***	*7,49*
KroE100	*100*	*22068*	***22068***	*22156*	*22112*	*0.19*	***0.00***	*30/05*	*11,52*
Rd100	*100*	*7910*	***7910***	***7910***	***7910***	***0,00***	***0.00***	***30/30***	*10,55*
Eil101	*100*	*629*	***629***	*637*	*632.43*	*0.54*	***0.00***	*30/05*	*16.9*
Lin105	*105*	*14379*	***14379***	***14379***	***14379***	***0.00***	***0.00***	***30/30***	*2,2*
Pr107	*107*	*44303*	***44303***	*44326*	*44314,5*	*0,02*	***0,00***	*30/25*	*20,02*
Pr124	*124*	*59030*	***59030***	***59030***	***59030***	***0,00***	***0,00***	***30/30***	*1,9*
Bier127	*127*	*118282*	***118282***	*118657*	*118469,5*	*0,15*	***0,00***	*30/7*	*64,62*
Ch130	*130*	*6110*	***6110***	*6155*	*6124,1*	*0,23*	***0,00***	*30/5*	*15,05*
Pr136	*136*	*96772*	***96772***	*97468*	*96995*	*0,23*	***0,00***	*30/4*	*20,75*
Pr144	*144*	*58537*	***58537***	***58537***	***58537***	***0,00***	***0,00***	***30/30***	*2,01*
Ch150	*150*	*6528*	***6528***	*6584*	*6550,3*	*0,34*	***0,00***	*30/2*	*23,7*
KroA150	*150*	*26524*	***26524***	*26649*	*26560,2*	*0,13*	***0,00***	*30/7*	*20,02*
KroB150	*150*	*26130*	***26130***	*26266*	*26146,63*	*0,06*	***0,00***	*30/7*	*21,21*
Pr152	*152*	*73682*	***73682***	*73818*	*73759,06*	*0,10*	***0,00***	*30/20*	*13,4*
Rat195	*195*	*2323*	*2324*	*2360*	*2340,7*	*0,76*	*0,04*	*20/0*	-
D198	*198*	*15780*	***15780***	*15870*	*15802,83*	*0,14*	***0,00***	*30/3*	*45,07*
KroA200	*200*	*29368*	***29368***	*29740*	*29449,23*	*0,27*	***0,00***	*30/2*	*49,05*
KroB200	*200*	*29437*	*29448*	*29819*	*29542.49*	*0.29*	*0.03*	*28/0*	-
Gil262	*262*	*2378*	*2382*	*2410*	*2390,7*	*0,53*	*0,08*	*26/0*	-
A280	*280*	*2579*	***2579***	*2611*	*2586,83*	*0,30*	***0,00***	*30/6*	*102,17*
Pr299	*299*	*48191*	***48191***	*48552*	*48311,7*	*0,25*	***0,00***	*30/2*	*125,11*
Lin318	*318*	*42029*	*42154*	*42713*	*42462.16*	*1.03*	*0.29*	*12/0*	-
Rd400	*400*	*15281*	*15336*	*15574*	*15465.3*	*1.20*	*0.35*	*7/10*	-
Nrw1379	*1379*	*56638*	*58951*	*59837*	*59349.53*	*4.78*	*4.08*	*0/0*	-

The results in Table 3 confirm that the DCSO algorithm can solve most of the TSPLIB instances in a very fast execution time.

To prove the robustness of the algorithm, Table 4 compares the average solution proposed by the DSCO algorithm with other methods recently used and which are applied to solve TSP using TSPLIB library. Table 5 compares the best execution time achieved by the algorithm compared to new bio-inspired algorithm that solves the TSP.

Table 4. The Average Results Obtained by Some Methods on Some TSPLIB Instances

Instance	Opt	DSCO	ACO [25] [26]	PSO [13]	GA [27] [28]
Eil51	426	**426**	430	436,9	429
Berlin52	7542	**7542**	7594	7832	7738
St70	675	**675**	750	697,5	-
Eil76	538	**538**	552,6	560,4	-
KroA100	21282	**21282**	21475	-	21445

According to the results in Table 4, the average time obtained by the DSCO algorithm gives good results than the other methods. Table 5, Figure 7, and Figure 8 compare the average time obtained by the DSCO algorithm compared to new bio-inspired algorithms.

Table 5. The Average Time Results Obtained by Bio-inspired Methods on Some TSPLIB Instances

Instance	Opt	DSCO	CS [20]	BA [18]	HUS [21]
Eil51	426	0,32	1,16	**0,20**	0,51
Berlin52	7542	0,09	0,09	**0,03**	0,16
St70	675	**0,04**	1,56	0,43	1,42
KroA100	21282	**0,05**	2,70	1,36	7,18
KroB100	22141	**3,17**	8,74	3,35	11,25
KroC100	20749	**2,50**	3,36	2,51	6,48

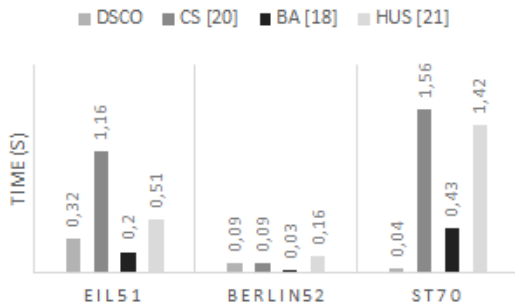

Figure 7. Comparing execution time of bio-inspired algorithm in a small TSP instances

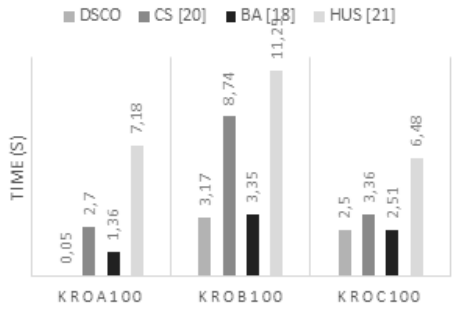

Figure 8. Comparing execution time of bio-inspired algorithm in a large TSP instances

The experimental results show that execution time of DSCO algorithm is better than the other bio-inspired algorithms in most of TSPLIB instances used to test the performance of the algorithm. Especially in large instance, our algorithm can find the best solution in the best time, which will be used to solve other NP-hard combinatory problems like TSP.

6. CONCLUSION

This paper presents a new discrete Chicken swarm optimization (DCSO) algorithm to solve the symmetric Traveling salesman problem (TSP) by applying a local search to improve the quality of the solutions. The adaptation of the algorithm is based on the behaviors of chickens in a swarm. The algorithm has been tested on a set of benchmark instances of TSPLIB. Its performance exceeds the recent methods used to solve the TSP such as ACO, PSO and GA, the effectiveness of the algorithm is due to the diversification of operations and operators based on the different kind of chickens (roosters, hens and chicks).

In the future researches we will improve the DCSO algorithm in order to obtain a better results by applying an hybrid approaches. Moreover, the DCSO robustness and rapidity encourage the use of algorithm to solve other combinatorial optimization problems.

REFERENCES

[1] S. Arora, "Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems," J. ACM, vol. 45, no. 5, pp. 753–782, Sep. 1998.
[2] R. G. Bland and D. F. Shallcross, "Large travelling salesman problems arising from experiments in X-ray crystallography: A preliminary report on computation," Oper. Res. Lett., vol. 8, no. 3, pp. 125–128, Jun. 1989.
[3] H. D. Ratliff and A. S. Rosenthal, "Order-Picking in a Rectangular Warehouse: A Solvable Case of the Traveling Salesman Problem," Oper. Res., vol. 31, no. 3, pp. 507–521, Jun. 1983.
[4] J. K. Lenstra and A. H. G. R. Kan, "Some Simple Applications of the Travelling Salesman Problem," J. Oper. Res. Soc., vol. 26, no. 4, pp. 717–733, Dec. 1975.
[5] M. Zachariasen and M. Dam, "Tabu Search on the Geometric Traveling Salesman Problem," in Meta-Heuristics, I. H. Osman and J. P. Kelly, Eds. Boston, MA: Springer US, 1996, pp. 571–587.
[6] J.-Y. Potvin, "Genetic algorithms for the traveling salesman problem," Ann. Oper. Res., vol. 63, no. 3, pp. 337–370, Jun. 1996.
[7] Abid, M. M., & Muhammad, I. (2015). Heuristic Approaches to Solve Traveling Salesman Problem. Indonesian Journal of Electrical Engineering and Computer Science, 15(2), 390-396.
[8] Y. Marinakis, A. Migdalas, and P. M. Pardalos, "Expanding Neighborhood GRASP for the Traveling Salesman Problem," Comput. Optim. Appl., vol. 32, no. 3, pp. 231–257, Dec. 2005.
[9] X. Geng, Z. Chen, W. Yang, D. Shi, and K. Zhao, "Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search," Appl. Soft Comput., vol. 11, no. 4, pp. 3680–3689, Jun. 2011.

[10] Al-Obaidi, A. T. S., Abdullah, H. S., & Ahmed, Z. O. (2018). Meerkat Clan Algorithm: A New Swarm Intelligence Algorithm. Indonesian Journal of Electrical Engineering and Computer Science, 10(1), 354-360.
[11] M. Manfrin, M. Birattari, T. Stützle, and M. Dorigo, "Parallel Ant Colony Optimization for the Traveling Salesman Problem," in Ant Colony Optimization and Swarm Intelligence, vol. 4150, M. Dorigo, L. M. Gambardella, M. Birattari, A. Martinoli, R. Poli, and T. Stützle, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 224–234.
[12] M. Clerc, "Discrete Particle Swarm Optimization, illustrated by the Traveling Salesman Problem," in New Optimization Techniques in Engineering, vol. 141, Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 219–239.
[13] X. H. Shi, Y. C. Liang, H. P. Lee, C. Lu, and Q. X. Wang, "Particle swarm optimization-based algorithms for TSP and generalized TSP," Inf. Process. Lett., vol. 103, no. 5, pp. 169–176, Aug. 2007.
[14] D. Teodorovic, P. Lucic, G. Markovic, and M. D. Orco, "Bee Colony Optimization: Principles and Applications," 2006, pp. 151–156.
[15] Zong Woo Geem, Joong Hoon Kim, and G. V. Loganathan, "A New Heuristic Optimization Algorithm: Harmony Search," SIMULATION, vol. 76, no. 2, pp. 60–68, Feb. 2001.
[16] M. BOUZIDI and M. E. RIFFI, "ADAPTATION OF THE HARMONY SEARCH ALGORITHM TO SOLVE THE TRAVELLING SALESMAN PROBLEM," Journal of Theoretical and Applied Information Technology, 04-Oct-2014.
[17] X.-S. Yang, "A New Metaheuristic Bat-Inspired Algorithm," in Nature Inspired Cooperative Strategies for Optimization (NICSO 2010), vol. 284, J. R. González, D. A. Pelta, C. Cruz, G. Terrazas, and N. Krasnogor, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 65–74.
[18] Saji, Y., & Riffi, M. E. (2016). "A novel discrete bat algorithm for solving the travelling salesman problem," Neural Comput. Appl., vol. 27, no. 7, pp. 1853–1866, Oct. 2016.
[19] X.-S. Yang and Suash Deb, "Cuckoo Search via Lévy flights," 2009, pp. 210–214.
[20] A. Ouaarab, B. Ahiod, and X.-S. Yang, "Discrete cuckoo search algorithm for the travelling salesman problem," Neural Comput. Appl., vol. 24, no. 7–8, pp. 1659–1669, Jun. 2014.
[21] amine AGHARGHOR and M. E. RIFFI, "HUNTING SEARCH ALGORITHM TO SOLVE THE TRAVELING SALESMAN PROBLEM.," Journal of Theoretical and Applied Information Technology, 10-Apr-2015.
[22] X. Meng, Y. Liu, X. Gao, and H. Zhang, "A New Bio-inspired Algorithm: Chicken Swarm Optimization," in Advances in Swarm Intelligence, vol. 8794, Y. Tan, Y. Shi, and C. A. C. Coello, Eds. Cham: Springer International Publishing, 2014, pp. 86–94.
[23] J. Monnot and S. Toulouse, "The Traveling Salesman Problem and its Variations," in Paradigms of Combinatorial Optimization, V. Th. Paschos, Ed. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2013, pp. 173–214.
[24] D. Wu, F. Kong, W. Gao, Y. Shen, and Z. Ji, "Improved chicken swarm optimization," 2015, pp. 681–686.
[25] C. Tsai, "A new hybrid heuristic approach for solving large traveling salesman problem*1," Inf. Sci., vol. 166, no. 1–4, pp. 67–81, Oct. 2004.
[26] A.Puris, R. Bello, Y. Martínez, and A. Nowe, "Two-Stage Ant Colony Optimization for Solving the Traveling Salesman Problem," in Nature Inspired Problem-Solving Methods in Knowledge Engineering, vol. 4528, J. Mira and J. R. Álvarez, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 307–316.
[27] S.-M. Soak and B.-H. Ahn, "New Genetic Crossover Operator for the TSP," in Artificial Intelligence and Soft Computing - ICAISC 2004, vol. 3070, L. Rutkowski, J. H. Siekmann, R. Tadeusiewicz, and L. A. Zadeh, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 480–485.
[28] I.-H. Kuo, S.-J. Horng, T.-W. Kao, T.-L. Lin, C.-L. Lee, Y.-H. Chen, Y. Pan, and T. Terano, "A hybrid swarm intelligence algorithm for the travelling salesman problem: A hybrid swarm intelligence algorithm for the travelling salesman problem," Expert Syst., vol. 27, no. 3, pp. 166–179, Jun. 2010.