Turkish Online Journal of Qualitative Inquiry (TOJQI)
Volume 12, Issue 7, July 2021: 3577-3588

Self-Adaptive Social Spider Algorithm optimization for Solving Travelling Salesman Problem

Dr.Ajit Kumar Raut[a], AbhisekSethy[b]

[a]Professor of Department of Information Technology, GMR Institute of Technology, Rajam, India.
[b]Assistant Professor ofDepartment of Information Technology, GMR Institute of Technology, Rajam, India.

Abstract

In this paper, a new method is presented based on Social Spider Algorithm (SSA) for solving travelling salesman problem (TSP). Since the SSA is applied to continuous problem and TSP is discrete NP-hard problem, adaptive version of the SSA is introduced that called Adaptive SSA (DSSA). The DSSA algorithm was implemented on 36 instances of TSPLIB benchmarks (a library of sample instances for the TSP) that include symmetric and asymmetric traveling salesman problems. In order to implement the DSSA, MATLAB 2017 was used. After simulation, DSSA found the optimal solution for 24 instances out of 36 datasets. The simulation results showed that DSSA is superior to other algorithms for solving both the TSP and ATSP problems. We propose the Adaptive Social Spider Algorithm (DSSA) based on SSA for solving adaptive optimization problems such as TSP problem. The DSSA is main contribution of this paper. In this section, the changes of original SSA are described to create adaptive version of SSA.

Keywords: Adaptive Social Spider Algorithm, Traveling Salesman Problem, Optimization.

1. Introduction

The Traveling Salesman Problem is one of the most intractable problem in NP-Hard class. The TSP can be modeled as adaptive optimization problem. Studies show that many evolutionary computing and swarm intelligence algorithms have been used for solving optimization problem and in particular TSP problem. For example, genetic algorithm (GA), bat algorithm (BA), firefly algorithm (FA) and so on. From the last 50 years so far, The TSP has been important research topic in the world [1]. The purpose of this problem is to find a Hamiltonian cycle in the connected graph while the cost of the tour is minimized. The Hamilton cycle is a path that visits each vertex of the graph exactly once and returns to the origin vertex. Social Spider Algorithm (SSA) is a type of swarm intelligence algorithm that developed by James J.q. Yu and Victor O.K. Li in 2015 [2] and is developed for solving continuous optimization problem. In this work, we proposed DSSA algorithm and then applied it to solve adaptive optimization problems. The main contribution of this work is DSSA algorithm that adaptive type of SSA to solve TSP and ATSP (Asymmetric TSP) problems. The DSSA algorithm was implemented on 36 instances of TSPLIB benchmarks (a library of sample instances for the TSP). After simulation, the results are compared to IBA, ESA, GA, IDGA, DFA and DICA algorithms that implemented in this problem in [3].This paper is continued as follows: Related

research is presented in Section 2. The proposed algorithm is explained in Section 3. In addition, the results are presented in Section4. Finally, conclusions are explained in Section 5.

2. Related Work

There are various techniques for solving TSP problem. Investigation and study of various references indicate that evolutionary algorithms have been widely applied in optimization problem and in particular TSP problem that in this part we mentioned a number of them as examples. In [3] authors presented aadaptive Bat algorithm for solving routing problems. They implemented the proposed algorithm on 37 instances of TSPLIB. In [1] [4] applied simulated annealing and tabu search algorithm for solving TSP problem. In [5], ant colony optimization algorithms are used for solving traveling salesman problem. In another article [6], authors proposed a hybrid evolutionary fuzzy learning algorithm to solve large scale TSP problem. They results demonstrated that their algorithm is suitable for routing problem and has acceptable computing time. In another study [7], proposed a generalized heuristic method based on ant colony algorithm. The proposed method was applied to TSP problem and achieved acceptable results. In [8] authors proposed a Hybrid algorithm for solving Probabilistic TSP based on Particle Swarm Optimization. Hybrid evolutionary algorithms proposed in [9] and applied to solve Multi objective TSP problem. Firefly algorithm and ant colony algorithm were used for solving TSP problem in [10], [11], respectively. In [12], adaptive invasive weed optimization algorithm proposed to solve TSP and in [11] a Tabu Search Approach is designed for the Prize Collecting Traveling Salesman Problem.

2.1. The Traveling Salesman Problem

The Traveling Salesman Problem is one of the most intractable problem, and it takes a lot of time to solve it. There are a number of cities in the TSP problem. The distance between each pair of cities in this problem is fixed and constant. The aim of this problem is to find a Hamilton cycle with minimum cost. In a directed graph, the Hamiltonian cycle is a path from one vertex to itself. In this cycle, the salesman passes from all the other cities exactly once. Mathematically, the problem is determined by finding the number of permutations and then evaluating each state.

This problem can be simulated as follows [1]:

If G = (V, A) is a complete graph, the Hamilton cycle with minimum cost can be formulated as:

$$Minimize\ z = \sum_{i \epsilon V} \sum_{j \epsilon V} l_{ij} x_{ij} \qquad (1)$$

In this Equation, lij indicate the distance between the cities i and j. If this distance is equal for every pair of cities, TSP is symmetrical TSP and otherwise is Asymmetrical TSP.

2.2.Social Spider Algorithm

The original Social Spider Algorithm (SSA) is introduced here, SSA is a type of swarm intelligence algorithm that introduced in 2015 for solving continuous optimization problems [2]. According to reference [2], spiders have a web that formulated the search space of problem. When spiders move on the web, vibration is produced. The vibration is propagated in the web, and another spider

receives it. In fact, spiders shared information by the web. Moreover, each spider has a position and fitness of the position on the web. Spiders can move from one position to another position on the web. As mentioned in [2], the pseudo-code of Social Spider Algorithm is presented as:

Algorithm-1: Social spider algorithm

1. Assign values to the parameters of SSA
2. Create the population of spiders' pop and assign memory for them
3. Initialize v_s^{tar} for each spider
4. While stopping criteria not met do
5. for each spider s in pop do
6. Evaluate the fitness value of s
7. Generate a vibration at the position of s
8. End for
9. for each spider s in pop do
10. Calculate the intensity of the vibrations V, generated by all spiders
11. Select the strongest vibration v_s^{best} from V
12. if the intensity of v_s^{best} is larger than v_s^{tar} then
13. Store v_s^{best} as v_s^{tar}
14. End if
15. Update CS
16. Generate a random number r from [0,1)
17. if $r > p_c^{cs}$ then
18. Update the dimension mask $\boldsymbol{m_s}$
19. End if
20. Generate p_s^{fo}.
21. Perform a random walk.
22. Address any violated constraints
23. End for
24. End while

25. Output the best solution found

This pseudo-code has 5 important steps: fitness evaluation, vibration generation, mask changing, random walks and constraints handling. Details of each step are described below.

In fitness evaluation step, fitness function is calculated for each spider. Since the SSA is designed for minimization problem, each spider with minimum fitness is known as the best spider in swarm.

Let **f**(**Ps**) be a spider fitness in its position and c is a small constant. Each spider produces a vibration in accordance with the **f**(**Ps**) as [2, 14, 15, 16]

$$I(P_s, P_s, t) = \log\left(\frac{1}{f(P_s) - c} + 1\right) \quad (2)$$

The vibration attenuation is defined in accordance with distance as [2]:

$$I(P_a, P_b, t) = I(P_a, P_a, t) \times \exp\left(-\frac{D(P_a, P_b)}{\bar{\sigma} \times r_a}\right) \quad (3)$$

Where D(Pa, Pb) is the Manhattan distance between the spiders a and b, σ is the standard deviation of spider positions, and $r_a \in (0, \infty)$ is the user-defined parameter that controls the attenuation rate [14][16].After this step, SSA propagates these vibrations and each spider receives the produced vibration from another spider. Then spiders find the strongest received intensity.

According to [2] [16], mask is a sparse matrix that spiders use it as a guide to walk. In mask changing step, each spider generates a new mask. P_m and P_c are user-defined parameter in (0,1). SSA uses these parameters to changing mask. As mentioned in [2] [16], probability of changing mask is $1 - P_c^{cs}$. P_m represents the probability of being 1 in each cell of the mask, when the mask is decided to be changed. It is worth noting that each bit of a mask is changed independently.

Then a new following position generated as [2]:

$$P_{s,i}^{fo} = \begin{cases} P_{s,i}^{tar} & m_{s,i} = 0 \\ P_{s,i}^{r} & m_{s,i} = 1 \end{cases} \quad (4)$$

A random walk is represented as [2] [14] [16]:

$$P_s(t+1) = P_s + (P_s - P_s(t-1)) \times r + P_s^{fo} - P_s) \odot R \quad (5)$$

Where R is a vector of random floating-point numbers between 0 to 1 and Θ is element-wise multiplication, respectively [2].

The final step is the constraint handling. In this step, SSA is the consideration of optimization problem's limitation and testing the position of each spider. If the spider was out of the search space, the position is modified as follows [2]:

$$P_{s,i}(t+1) = \begin{cases} (\bar{x}_i - P_{s,i}) \times r & if \quad P_{s,i}(t+1) > \bar{x}_i \\ (P_{s,i} - \underline{x}_i) \times r & if \quad P_{s,i}(t+1) < \underline{x}_i \end{cases} \quad (6)$$

As mentioned in [2] [16], $\overline{x}_i$, $\underline{x}_i$ are the upper bound and the lower bound of the search space in problem and r is a random floating point number between 0 and 1.

3. The proposed method

Motivated by [3], we propose the Adaptive Social Spider Algorithm (DSSA) based on SSA for solving adaptive optimization problems such as TSP problem. The DSSA is main contribution of this paper. In this section, the changes of original SSA are described to create adaptive version of SSA.

3.1.Controlling Randomness

SSA has two user-controlled parameters for adjusting randomness in order to control exploration and exploitation of the algorithm ,i.e.$P_c \in (0,1)$ is a user-defined parameter[2] [16] that represents "the probability of changing mask", and P_m, that determines the probability of being 1 in each cell of the mask matrix. Unlike SSA that used fixed parameter scheme [17] [18], DSSA used the parameters in a deterministic [17] [19] manner to control the randomness of the algorithm in such a way that the algorithm gains maximum randomness in the beginning of the search and achieves maximum exploration. The randomness then decreases toward the end of the algorithm in order to converge on the solutions.

In doing so, DSSA calculates $P_{c,i}$ (P_c at iteration i) and $P_{m,i}$ ($P_m$at iteration i) as follows:

$$P_{c,i} = P_{c,i-1} \times (1 + i \times \frac{C_c}{M^2}) \quad (7)$$

and

$$P_{m,i} = P_{m,i-1} \times (1 - \left(\frac{C_m \times (M - i)}{M^2}\right)) \quad (8)$$

where $i$is the iteration number, $M$is the number of iterations, $C_c$and $C_m$are control parameters determined by user.Figure1 show how the parameters Pc and Pm change during the execution of the algorithm in which the horizontal coordinates represent the iteration number and vertical coordinates represent the value of these parameters.

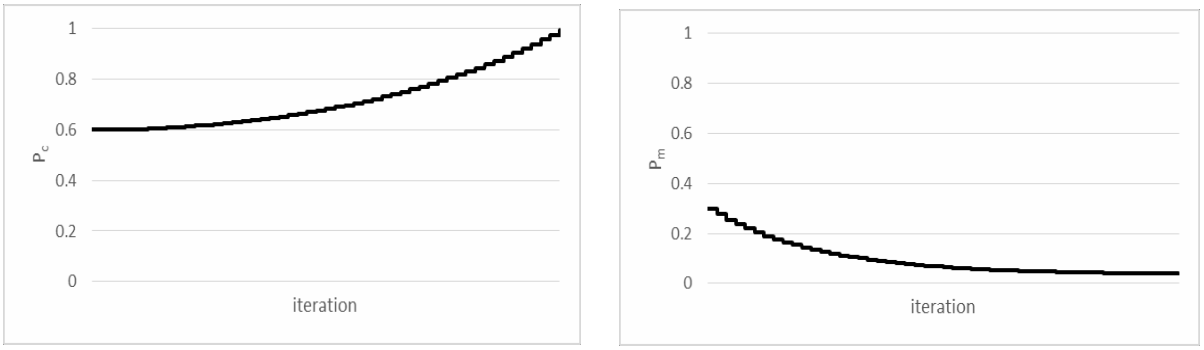

Figure 1. The Changes of Pc and Pm throughout the algorithm

As depicted in these Figures, the trade-off between exploration and exploitation is established in DSSA algorithm.

3.2.**Distance Function**

DSSA uses Hamming distance (HD) [3] to measure the distance between two spiders, instead of Manhattan distance as in original SSA. Each permutation of cities represents a spider. To calculate the HD between two permutations, consider two spiders as a two permutation consist five nodes, e.g., $P_1 = [3\ 5\ 1\ 4\ 2]$, and $P_2 = [3\ 4\ 1\ 2\ 5]$, then $HD(P_1, P_2) = 3$that is the result of comparing peer-to-peer elements in two permutations. Firstly, $HD(P_1, P_2) = 0$. If any element of P_1 doesn't match to corresponding element in P_2,$HD(P_1, P_2) = HD(P_1, P_2) + 1$.

3.3.Follow Position

As stated earlier, in order to increase randomness of the algorithm, SSA generates follow position based on a little deviation from the target position. SSA performs this by some arithmetic calculations on numbers, and a random sparse matrix m_s. Because DSSA aims to solve adaptive optimization problems, the real number-based calculation used in SSA does not address the requirements of such problems. DSSA utilizes a novel method to gain following position based on target position and mask matrix as follows:

$$P_s^{fo} = \begin{cases} P_s^{tar} & m_{s,i} = 0, \quad \forall i \\ \text{swap}(P_{s,i}^{tar}, P_{s,j}^{tar}) & m_{s,i} = 1\ , m_{s,j} = 1 \end{cases} \tag{9}$$

For example, suppose P_s^{tar}=[12345678910],andm_s=[0100000100] (i.e.,i=2,j=8),

then P_s^{fo}= $\text{swap}(P_{s,i}^{tar}, P_{s,j}^{tar})$ = [1 8 3 4 5 6 7 2 910].

3.4.Movement

Inspired from [3] [20], DSSA utilizes a combined method to perform a random walk for each spider towards the following position. In the proposed method, a combination of five well- known successor operators have been used to determine new position of the spider s at step t+1 based on its position at step t:

$$P_s(t+1) = \begin{cases} \text{Vinsert}(P_{s,i}(t), j) \\ \text{randswap}(P_{s,i}(t), P_{s,j}(t)) \\ \text{swap}(P_{s,i}(t), P_{s,j}(t)) \\ 2-opt(P_s(t), \delta_s^t) \\ 3-opt(P_s(t), \delta_s^t) \end{cases} \tag{10}$$

These operators are defined as follows:

Vertex insertion ($\text{Vinsert}(P_{s,i}(t), j)$): This operator randomly selects a node of the tour in position i($P_{s,i}(t)$) ,removes it and reinserts in another random position (position j) [20]. Random Swapping (rand $\text{swap}(P_{s,i}(t), P_{s,j}(t))$): This operator selects two nodes randomly ($P_{s,i}(t), P_{s,j}(t)$at position s, i and j) and swap their positions [20]. Swapping (s $\text{wap}(P_{s,i}(t), P_{s,j}(t))$): This operator selects one

node of $P_s(t)$randomly (for example $P_{s,i}(t)$), find its corresponding index in follow position ($P_{s,i}(t) = P_{s,j}^{fo}(t)$) and swap the nodes($P_{s,i}(t), P_{s,j}(t)$). 2-opt ($2-opt(P_s(t), \delta_s^t)$): This operator was defined by Lin in 1965 [21]. The action of this function is: selecting two random edges from the path and remove them, then generating two new edges. The function doesn't create sub tour when generate new edges [3]. In this case, $\delta_s^t$is a random number between 1, and the Hamming distance of current position of spider s and its follow position [3]:

$$\delta_s^t = Random\ [\ 1, HD\left(P_s(t), P_s^{fo}\right)] \quad (11)$$

3-opt ($3-opt(P_s(t), \delta_s^t)$): The 3-opt operator was defined by Lin [21]. The action of this function is: selecting three random edges from the path and remove them, then generating three new edges. Similar to 2-opt, the function doesn't create sub tour [3]. It's worth noting that the complexity of this function is much greater than the 2-opt function [22] [23].

3.5. Fitness Function

DSSA similar to TSP is designed for minimization problem. Therefore, the sum of the cost of all arcs of the solution has been used as the fitness function in this implementation.

3.6. Adaptive Social Spider Algorithm (DSSA)

DSSA needs some changes to the SSA algorithm to be able to solve adaptive optimization problems such as TSP and ATSP. The pseudo-code of the proposed DSSA is depicted in Algorithm 2.

Assign values to the parameters of DSSA.

Algorithm 2: Adaptive Social Spider Algorithm

1. Create the population of spiders pop and memory to them
2. Initialize VStar for each spider
3. while stopping criteria not met do
4. for each spider s in pop do
5. Evaluate the fitness value of s
6. Generate a vibration at the position of s
7. End for
8. for each spider s in pop do
9. Calculate the intensity of the vibrations V generated by all spiders
10. Select the strongest vibration VSbest from V
11. if the intensity of VSbest is larger than VStar then
12. Store VSbest as VStar
13. End if
14. Update CS
15. Generate a random number r from [0,1)
16. if r > CS then
17. Update the dimension mask mS
18. End if

19. for each spider s in pop do
20. if mS = 0 then
21. Psfo = Psstar
22. Else
23. Select two index i, j where mS,i = 0 and mS = 0
24. Psfo = swap(Ps,istar , Ps,jstar)
25. End if
26. Select at random a combination of these operators
27. PS(t+1) <- Vinsert (PS,i(t),j)
28. Swap(PS,i(t), PS,j(t))
29. 3-opt(PS(t), δS(t))
30. End for
31. Select one solution among the best ones
32. Generate a new spider selecting the best neighbor around the chosen spider using the mentioned operators
33. End for
34. End while , Output the best solution found

4. Results and discussion

In this work, 36 instances of TSPLIB with 17 to 439 nodes are used to evaluate the performance of the proposed DSSA algorithm. According to [3], For TSP problem, 21 instances and for ATSP problem, 15 instances have been chosen. MATLAB 2017 is used on a Windows 10 operating system for this simulation. The experiments are based on 50 spiders which move on the web based on five successive operators with ra = 5. Number of iterations is variable depending on the length of the tour (>=5000), and Pm is changed based on Eq. (7) and Eq. (8) as demonstrated in Figure 6. A summary of parameter setting is shown in Table1.

Table 1: Parameter setting DSSA

Parameter	Value
Population size	50
Movement functions	Eq. 10
Maximum iteration	<=5000
Ra	5
Pc	0.8

The results of experiments of DSSA as well as the comparison with other algorithms is shown in Table 2, where the Optima column shows optimal tour length reported in TSPLIB, and Best columns show the tour length of the best solutions obtained from DSSA and other mentioned algorithms. Each row of Table 2 also shows one instance of TSPLIB [3]. In this comparison, we

used the results of experiments for other algorithms were reported in [3]. As mentioned in [3] the algorithms are: "dynamic bat algorithm (IBA), evolutionary simulated annealing (ESA), genetic algorithm (GA), an island based distributed genetic algorithm (IDGA), a dynamic firefly algorithm (DFA) and an imperialist competitive algorithm (DICA)".

Table 2: Comparing the results of DSSA and IBA, ESA, GA, IDGA, DFA, DICA for the TSP and ATSP

Instance		DSSA	IBA	ESA	GA	IDGA	DFA	DICA
Name	Optima	Best	Best	Best	Best	Best	Best	Best
Oliver30	420	420	420	420	420	420	420	420
Eilon50	425	425	425	427	426	425	425	425
Eil51	426	426	426	426	427	426	426	426
Berlin52	7542	7542	7542	7542	7542	7542	7542	7542
St70	675	675	675	675	675	675	675	675
Eilon75	535	535	535	545	550	544	535	537
Eil76	538	538	539	546	545	545	543	544
KroA100	21,282	21,282	21,282	21,282	21,350	21,345	21,282	21,282
KroB100	22,140	22,140	22,140	22,202	22,176	22,208	22,183	22,180
KroC100	20,749	20,749	20,749	20,749	20,861	20,830	20,756	20,756
KroD100	21,294	21,294	21,294	21,500	21,492	21,582	21,408	21,399
KroE100	22,068	22,068	22,068	22,099	22,150	22,110	22,079	22,083
Eil101	629	629	634	650	655	650	643	644
Pr107	44,303	44,303	44,303	44,413	44,392	44,428	44,303	44,303
Pr124	59,030	59,030	59,030	59,030	59,030	59,072	59,030	59,030
Pr136	96,772	97,022	97,547	98,499	98,432	98,532	97,716	97,736
Pr144	58,537	58,537	58,537	58,574	58,599	58,581	58,546	58,563
Pr152	73,682	73,682	73,921	74,172	74,520	74,249	74,033	74,052
Pr264	49,135	49,235	49,756	51,603	51,712	51,653	50,491	50,553
Pr299	48,191	48,765	48,310	49,242	49,659	49,572	48,579	48,600
Pr439	107,217	113,148	111,538	113,497	113,576	113,207	111,967	111,983
Similarly								
br17	39	39	39	39	39	39	39	39
ftv33	1286	1286	1286	1286	1290	1286	1286	1286
ftv35	1473	1473	1473	1473	1490	1498	1473	1473
ftv38	1530	1530	1530	1530	1565	1560	1530	1530
p43	5620	5620	5620	5620	5620	5620	5620	5620
ftv44	1613	1613	1613	1645	1649	1645	1620	1622
ftv47	1776	1777	1796	1795	1820	1822	1795	1799

ry48p	14,422	14,422	14,422	14,485	14,545	14,530	14,453	14,463
ft53	6905	6987	7001	6990	7270	7076	6993	7002
ftv55	1608	1618	1608	1725	1700	1842	1628	1630
ftv64	1839	1909	1879	1955	2014	2080	1903	1900
ftv70	1950	2046	2111	2200	2184	2135	2173	2167
ft70	38,673	39,785	39,901	39,650	39,407	39,241	39,668	39,660
kro124p	36,230	36,580	37,538	40,019	39,265	39,099	39,438	39,400
rbg323	1326	1591	1615	1620	1514	1510	1599	1600

Table 2 shows all the cases for which the algorithms found the optimal solutions in bold font, where DSSA found the optimal solution for 24 instances out of 36 dataset. As it can be seen, DSSA generally shows better results than IBA, ESA, GA, IDGA, DFA and DICA. Additionally, in 12 cases for which the algorithms could not find the best solutions, DSSA achieved a solution closer to the optimal ones as compared to the other algorithms. Specially, the comparison between DSSA and IBA shows that DSSA found the optimal solution in 24 cases while IBA found the optimal solutions in 22 cases. It can be said that DSSA has performed better than IBA. The Comparison of the results of DSSA and IBA, ESA, GA, IDGA, DFA, DICA for the TSP and ATSP is demonstrated in Figure 2.

Figure 2. Comparing results of DSSA and IBA, ESA, GA, IDGA, DFA, DICA for the TSP and ATSP

Figure 3 compares DSSA to other algorithms based on the number of cases that they obtained the optimal results. The main conclusion that we can made from the experiments is that DSSA is the best algorithm for solving TSP problem

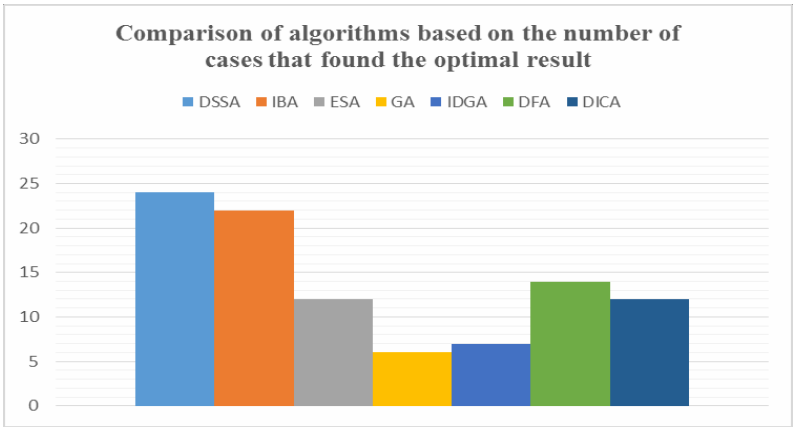

Figure 3. Comparison of algorithms based on the number of cases that found the optimal result.

5. Conclusion

In this paper, the Adaptive Social Spider Algorithm (DSSA) is proposed based on original SSA for solving adaptive optimization problem such as TSP and ATSP. DSSA is implemented with some changes in original SSA which consists of distance function and movement method. The DSSA results to solve TSP and ATSP have been compared to "dynamic bat algorithm, evolutionary simulated annealing, genetic algorithm, an island based distributed genetic algorithm, a dynamic firefly algorithm and an imperialist competitive algorithm" that reported in [3]. The results prove that the proposed DSSA achieves the best performance to solve both the TSP and ATSP. The proposed DSSA in this work can be applied for any adaptive optimization problem such as routing problem and any problem that formulated by complex graph.

References

1. Yu, L., Zheyong, B., Xiang, L., 2016. Developing a discrete neighborhood structure for an adaptive hybrid simulated annealing – tabu search algorithm to solve the symmetrical traveling salesman problem. Appl. Soft Comput. 49, 937-952.
2. James, J.Q. Yu., Victor, O.K. Li., 2015. A social spider algorithm for global optimization. Appl. Soft Comput. 30,614-627.
3. Osaba, E., Yang, X., Diaz, F., Lopez-Garcia, P., Carballedo, R., 2016. An improved discrete bat algorithm for symmetric and asymmetric Traveling Salesman Problems. EngApplArtif Intell.48,59-71.
4. Geng, X., Chen, Z., Yang, W., Shi, D., Zhao, K., 2011. Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search. Appl. Soft Comput. 11(4),3680-3689.
5. Ariyasingha, I.D.I.D., Fernando, T.G.I., 2015. Performance analysis of the multi –objective ant colony optimization algorithms for the traveling salesman problem. Swarm EvolComput. 23,11-26.
6. Feng, H-M., Liao, K-L., 2014. Hybrid evolutionary fuzzy learning scheme in the applications of traveling salesman problems. Inf Sci. 270,204-225.
7. Abd Aziza, Z., 2015. Ant Colony Hyper-heuristics for Travelling Salesman Problem. In: IEEE International Symposium on Robotics and Intelligent Sensors,534-538.
8. Marinakis, Y., Marinaki, M., 2010. A Hybrid Multi-Swarm Particle Swarm Optimization algorithm for the Probabilistic Traveling Salesman Problem. ComputOper Res. 37,432-442.
9. Psychas, I., Delimpasi, E., Marinakis, Y., 2015. Hybrid evolutionary algorithms for the Multi objective Traveling Salesman Problem. Expert Syst. 42,8956-8970.
10. Kumbharana, Sh., Pandey, G., 2013. Solving Travelling Salesman Problem using FireflyAlgorithm.

11. International Journal for Research in Science & Advanced Technologies. 2, 053-057.
12. Ghafurian, S., Javadian, N., 2011. An ant colony algorithm for solving fixed destination multi-depot multiple traveling salesman problems. Appl. Soft Comput. 11,1256-1262.
13. Zhou, Y., Luo, Q., Chen, H., He, A., Wu, J., 2015. A discrete invasive weed optimization algorithm for solving traveling salesman problem. Neurocomputing. 151,1227-1236.
14. Pedro, O., Saldanha, R., 2013. A Tabu Search Approach for the Prize Collecting Traveling Salesman Problem. Electron Notes Discrete Math. 41,261-268.
15. Ochoa, A., Juarez-Casimiro, K., Olivier, T., Camarena, R., Vazquez, I., 2017. Social Spider Algorithm to Improve Intelligent Drones Used in Humanitarian Disasters Related to Floods. In: Melin, P., Castillo, O., Kacprzyk, J. (eds). Nature-Inspired Design of Hybrid Intelligent Systems. Studies in Computational Intelligence, Springer, Cham. 667,457-476.
16. Agarwal, P., Singh, R., Kumar, S., Bhattacharya, M. 2016. Social Spider Algorithm Employed Multi-level Thresholding Segmentation Approach. In: Satapathy, S., Das, S. (eds) Proceedings of First International Conference on Information and Communication Technology for Intelligent Systems: Volume 2. Smart Innovation, Systems and Technologies, Springer, Cham. 51,249-259.
17. James, J. Q. Yu., Victor, O. K. Li., 2015 "Parameter sensitivity analysis of Social Spider Algorithm" IEEE Congress on Evolutionary Computation (CEC)
18. A. Y. S. Lam, V. O. K. Li, J. J. Q. Yu, 2012, "Real-coded chemical reaction optimization", IEEE Trans. Evol. Comput. 16 (3), 339–353.
19. K. Price, R.M. Storn, J.A. Lampinen, 2005. Differential Evolution – A Practical Approach to Global Optimization, Springer.
20. W.-N. Chen, J. Zhang, Y. Lin, N. Chen, Z.-H. Zhan, H.S.-H. Chung, Y. Li, Y.-H. Shi, 2013. Particle swarm optimization with an aging leader and challengers, IEEE Trans. Evol. Comput. 17(2), 241–258
21. Osaba, E., Diaz, F., Oniva, E., 2014. Golden ball: a novel meta-heuristic to solve combinatorial optimization problems based on soccer concepts. Appl Intell.41, 145-166.
22. Lin, S., 1965. Computer solutions of the traveling salesman problem. Bell Syst. Tech. J. 44, 2245- 2269.
23. Alfa, A., Heragu, S., Chen, M., 1991. A 3-opt based simulated annealing algorithm for vehicle routing problems. Comput. Ind. Eng. 21,635–639.
24. Rocki, K., Suda, R., 2012. Accelerating 2-opt and 3-opt local search using GPU in the Travelling salesman problem. In: IEEE International Conference on High Performance Computing and Simulation, 489–495.