Enhancing Particle Swarm Optimization Using Opposite Gradient Search for Travelling Salesman Problem

Thirachit Saenphon*
Faculty of Information and Communication Technology, Silpakorn University, Nonthaburi 11120, Thailand.

* Corresponding author. Tel.: +668-4950-5101; email: thirachits@gmail.com
Manuscript submitted June 14, 2018; accepted September 10, 2018.
doi: 10.17706/ijcce.2018.7.4.167-177

Abstract: The evolutionary computing based on Particle Swarm Optimization (PSO) technique has been proposed to obtain better performance for solving travelling salesman problems. Basically, the original PSO encounters a problem of convergence before tackling the best among local optimal solutions. To eliminate such problem, this paper presents an enhanced PSO algorithm called FOGS-PSO, which is a combination of PSO and Fast Opposite Gradient Search (FOGS) under benefits from the exploration ability of PSO and the ability to generate effective candidate solutions of FOGS. This algorithm is divided into two phases. Firstly, FOGS is applied to generate the best candidate solutions locating on the manifold of objective. Secondly, PSO is then applied to improve the searching result and speed. Travelling salesman problem was experimented as well as the objective function according to Hopfield-Tanks network. The proposed algorithm is compared with a variety of algorithms based on PSO techniques. The results of the test problems show that the algorithm performs well in terms of distance and number of generations.

Key words: Opposite gradient search, optimization, particle swarm optimization, travelling salesman problem.

1. Introduction

One of the well-studied combinatorial problem is the Travelling salesman problem (TSP). The problem has been given this name because it can be described concern in a salesman who has to travel a long distance on one tour to visit his customers. The salesman start from his home and he bids to inflict all the customers in different cities exactly once before turning back to his home as the solution to minimize the entire length of the tour. Two interesting issues of travelling salesman problem are the shortest travelling distance and the order of cities which are traversed. To solve the problem, most evolutionary algorithm used random technique to generate different sets of solutions and filter only solutions which minimal value of the objective function. These algorithms are not interested in the geometrical structure of the objective function as a part of the solution finding process. Considering the geometric structure of the objective function can reduce the search area and find the results quickly.

TSP concerns the sequence of cities and the total travelling distance. The sequence of the cities is important and involves the shortest distance. Hence, a set of travelling sequences must be used as a set of generating points scattered throughout the manifold of Hopfield-Tank's energy function [1], [2] that is the objective function for optimizing the function [3]. There are several algorithms to solve travelling salesman problems. Some of the techniques are Ant Colony Optimization (ACO) [3], [4], Particle Swarm Optimization (PSO) [5]-[7], Genetic Algorithm (GA) [8]. GPSO was proposed to solve TSP [7]. This algorithm includes two

phases, the first phase is applied Fuzzy C-Means clustering, and a rule-based route permutation, a random swap strategy and a cluster merge procedure. This approach firstly generates an initial non-crossing route. The second phase combined Genetic-based PSO procedure to solve the TSP with better efficiency. An efficient method based on hybrid genetic algorithm-particle swarm optimization (GA-PSO) is presented for various types of economic dispatch (ED) problem [9]. Despite PSO having been successfully applied to some complex problems such as TSP, there are still some problems. For instance, PSO might fall into local optimal solutions because of the faster loss of diversity on some problems [10].

To overcome PSO problems, this approach emphasized on how to define the objective function involving the sequences of travelling as well as the total distance of each sequence and how to apply the manifold of the objective function for searching the better solution in PSO technique. FOGS was a manifold search algorithm to find the locations with zero gradients and minimum values of the objective function which first introduced in [3]. Therefore, combining FOGS with PSO is unlike other PSO based method. The paper has four sections. In section II, a new method named novel two-stage hybrid FOGS-PSO algorithm is presented. In section III, TSP examples are used to experimental results with the proposed algorithms that have been used to solve the problems. Finally, the conclusions are given in Section IV.

2. The Framework of Hybrid Fogs-Pso Algorithm

In the previous parts, the PSO algorithm should be used as a powerful technique for managing various forms of optimization problems. Original PSO is based on social adaptation of knowledge for working, and all individuals in the population are considered for generating a new population in the next generation. For generating new population, I combined the concept of searching for the best solution on the manifold of objective function or FOGS with PSO. The algorithm of FOGS-PSO is described in the following sections.

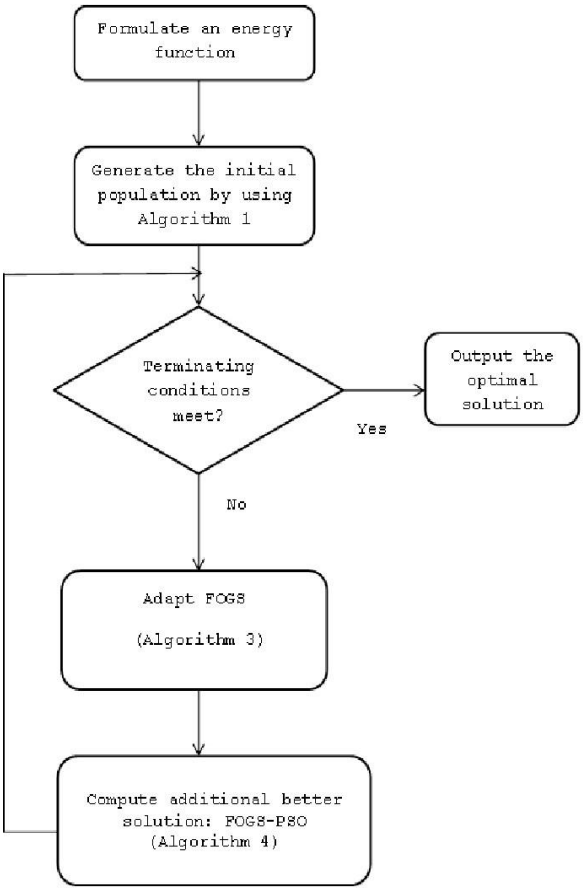


Fig. 1. A flowchart of FOGS-PSO described the combining of FOGS and PSO algorithm.

Many candidate solutions are generated by the FOGS algorithm. These solutions are spread throughout the surface of the energy function $E(S^{(*)})$ in a first part of proposed algorithm. The generated candidate solutions and their locations may not be evenly distributed enough to find the best solution because the structure and the geometric properties of the energy function use in high-dimensional space can not be visualized easily. To solve the defect of this distributed problem, the other evolutionary algorithm was combined with FOGS for enhancing the better solution. In this paper, the PSO algorithm was combined to generate some additional solutions besides the candidate solutions already generated by FOGS algorithm. The flowchart of the total algorithm named FOGS-PSO is shown in Fig. 1.

2.1. Generate the Initialize Swarm Population

The initialization swarm is an important population for the PSO algorithm to solve an optimization problem. In this proposed algorithm, the distribution of generated populations along the constraints stated as above- mentioned equations are concerned. To resolve these aspects, the values of some $S_{i,j}^{(*)}$ are set to zeros. Suppose each $S^{(*)}$ is of size $\times n$, where n is the number of cities and N is the number of populations. Given G^+ and $G^-$to represent gradients of energy function are positive and negative value respectively. After this step is processed, the output is the first generation N swarm population that divided into two different sign gradient set. The algorithm 1 is described the generating new population of swarm.

Algorithm 1. The proposed algorithm begins with generating an initial population of swarm.

1: $G^+ = \emptyset$ and $G^- = \emptyset$
2: For $1 \le k \le 2N$ do
3: Generate $S^{(k)}$ such that $\forall i,j: S_{i,j}^{(k)} = 1$
4: End for
5: For $1 \le k \le 2N$do
6: For $1 \le q \le 2N$do
7: Randomly set the values of i and j such that $1 \le i,j \le n$
8: $S_{i,j}^{(k)} = 0$
9: End for
10: End for
11: Sort all $S^{(k)}$, $1 \le k \le 2N$, in ascending order according to $E(S^{(k)})$
12: Select the first N populations of $S^{(k)}$
13: For $1 \le k \le N$ do
14: If $\nabla E\left(S^{(k)}\right) \ge 0$ then
15: Insert $S^{(k)}$ into G^+
16: Else
17: Insert $S^{(k)}$ into G^-
18: End if
19: End for

2.2. Brief Concept of Particle Swarm Optimization

The original Particle Swarm Optimization (PSO) algorithm was first presented by Kennedy and Eberhart in 1995. This optimization algorithm was inspired by the behaviors of a flock of birds or the sociological behavior of a group of people. Nowadays, the PSO is widely used in many fields, such as the continuous

optimization problems, the discrete optimization problems, Fuzzy system control, etc. PSO algorithm has been widely used for applying with other techniques to solve TSP problem [9], [11]. Suppose that the number of dimensions of search space is D and m particles from the colony. The i^{th} particle is represented by a D-dimensional $x_i\,(i = 1, 2, \dots, m)$ vector which means that the particle locates at $x_i = (x_{i1}, x_{i2}, x_{i3}, \dots, x_{iD})$ in the search space. The fitness of particle is calculated by putting its position into an objective function. When the fitness is lower in minimum optimization problem, the corresponding x_i become better. The velocity of i^{th} particle is also a D-dimensional vector, $V_i = (v_{i1}, v_{i2}, \dots, v_{iD})(i = 1,2, \dots, m)$. The best position of the i^{th} particle is also a defined by $P_i = (p_{i1}, p_{i2}, \dots, p_{iD})$ while the best position of the colony of the current generation is denoted by $P_g = (p_{g1}, p_{g2}, \dots, p_{gD})$. The PSO algorithm could be performed by the following equations:

$$x_i(k+1) = x_i(k) + V_i(k+1) \tag{1}$$

$$V_i(k+1) = \omega V_i(k) + c_1 r_1\big(P_i - x_i(k)\big) + c_2 r_2\left(P_g - x_i(k)\right) \tag{2}$$

where $i = 1,2, \dots, m$; ω is the inertia coefficient which is a constant in [0, 1]. It represents how much the current velocity at $x_i(k)$ c_1 *and* c_2 are learning rates; r_1 *and* r_2 are random numbers uniformly distributed [0, 1].

2.3. Adapt Fast Opposite Gradient Search

From the second generation, two new vectors are putting down in between two old vectors of opposite gradients in the first generation are computed to reduce the searching area when one of them gives a better result. And so this new vector along with another element in the first generation whose value of energy function is in an acceptable scope and its gradient that is opposite to the new vector, so the selected vector in the first generation is used to get a new vector in the third generation. Otherwise, any two vectors in the first generation, whose values of energy function are in an acceptable scope and their gradients are opposite to each other, are chosen to generate two new vectors in the second generation.

Given $G^+$and G^- to represent the vectors which their gradients are positive and negative value respectively. All vectors in G^+ and G^- are already sorted in ascending order by their energy cost function values. Given Max be the maximum number of iterations. When the algorithm terminates after the iteration criteria Max is found, the locations with zero gradients will be obtained. This location is a global optimum solution for the problem. The description of this step is shown in Algorithm 3. Details of how to calculate the weight value (ω) for iteration t of M, the maximum number of iterations, is described in Algorithm 2 bellow.

Algorithm 2. The algorithm show how to calculate weight value, ω.

1: Initialization: $\omega = 1, F_1 = 0, F_2 = 0$
2: Find a new vector using FOGS algorithm (algorithm 3)
3: If no city of $S^{(\theta)}$ is travelled in any step then
4: $F_1 = 1$
5: End if
6: If no city of $S^{(\gamma)}$ is travelled in any step then
7: $F_2 = 1$
8: End if
9: If $F_1 = 1, F_2 = 1$ then

10: Reduce ω, $\omega = \omega \times \frac{M}{t} \times 0.05$
11: A new vector that out of the scope is ignored and discarded.
12: Go to step 2 for finding other new vector with the new ω
13: Else
14: Let ω be a current weight value
15: Go to step 2 for finding a new vector again
16: End if

Algorithm 3. Adapted Fast Opposite Gradient Search.

1: Set $\omega = 1$ and $count = 1$
2: While $count \leq Max$ do
3: Let $S^{(\theta)}$ and $S^{(\gamma)}$ are the first vector of $G^+$and G^- respectively
4: Compute two new vectors $S^{(1)}$ from $S^{(\theta)}$ and $S^{(2)}$ from $S^{(\gamma)}$
5: Calculate the new weight value ω using algorithm 2
6: For $1 \leq i, j \leq n$ do
7: If new $S_{i,j}^{(1)} < 0$ then
8: Set $S_{i,j}^{(1)} = 0$
9: Else
10: Set $S_{i,j}^{(1)} = |S_{i,j}^{(1)} - 1|$
11: End if
12: If new $S_{i,j}^{(2)} < 0$ then
13: Set $S_{i,j}^{(2)} = 0$
14: Else
15: Set $S_{i,j}^{(2)} = |S_{i,j}^{(2)} - 1|$
16: End if
17: End for
18: For $1 \leq i \leq n$ do
19: Given $j = arg_{1 \leq k \leq n} \min(S_{i,k}^{(1)})$
20: Set $S_{i,j}^{(1)} = 1;\ \forall k \neq j: S_{i,k}^{(1)} = 0;\ \forall k \neq i: S_{k,j}^{(1)} = 0$
21: End for
22: For $1 \leq i \leq n$ do
23: Let $j = arg_{1 \leq k \leq n} \min(S_{i,k}^{(2)})$
24: Set $S_{i,j}^{(1)} = 1;\ \forall k \neq j: S_{i,k}^{(2)} = 0;\ \forall k \neq i: S_{k,j}^{(2)} = 0$
25: End for

2.4. Compute Additional Better Solution: FOGS-PSO

The PSO algorithm is based on vector updates and supports itself well for optimization in continuous vector spaces.

Therefore, when the PSO used to find the solution in the TSP problem which is a combinatorial problem, this algorithm still had limitation. The aim of combining PSO algorithm with FOGS in this paper is to find the better solution searching process. The possibly best solution can be found out in a few generations. A sequence of cities in term of vector $S^{(\theta)}$ obtained from Algorithm 3 is a path for some particle to travel during the optimization process.

PSO was used to generate additional paths $S^{(\gamma)}$ to find better solutions. The sequence obtained from FOGS will accordingly set to the best position for a first particle. The beginning positions of the particles corresponding to travelling sequence $S^{(\theta)}$are set the best position of the first $0.1 \times M$ particles.

Given M to represent the maximum number of iterations and $S^{(i)}$ be the i^{th} travelling sequence. The travelling sequence obtained from Algorithm 3 is denoted by $S^{(\theta)}$. Suppose m particles are given. The detail of combining the PSO algorithm is shown in Algorithm 4.

Algorithm 4. Combining Particle Swarm Optimization Algorithm with FOGS.

1: Set generation $t = 0$
2: While $t < M$ do
3: Initialize the first $0.1 \times M$ of particles P using its initial position from R computed in Algorithm 3
4: Set learning factors C_1, C_2
5: If $P_{best} < P_{gbest}$ then
6: $P_{gbest} = P_{best}$
7: End if
8: Calculating the current inertia weight coefficient ω and r_1, r_2 randomly
9: Updating the velocity and position according equation (1) and (2) from [8]
10: Updating for the P_{gbest}
11: $t = t + 1$
12: End while
13: P_{gbest} is the best solution

2.5. Enhance Other Evolutionary Algorithm with FOGS

By working to reduce the solution space quickly and easily create new generation points of FOGS nearby the optimum solution over another evolutionary which a new population created by a random method. I study that FOGS can combine other evolutionary algorithms to enhance the performance of them. A following flowchart in Fig. 2 shows the step of combining FOGS with alternative evolutionary algorithms. FOGS will replace the random initialize population and generate the new population in the second forwards convergence to the optimum solution quickly.

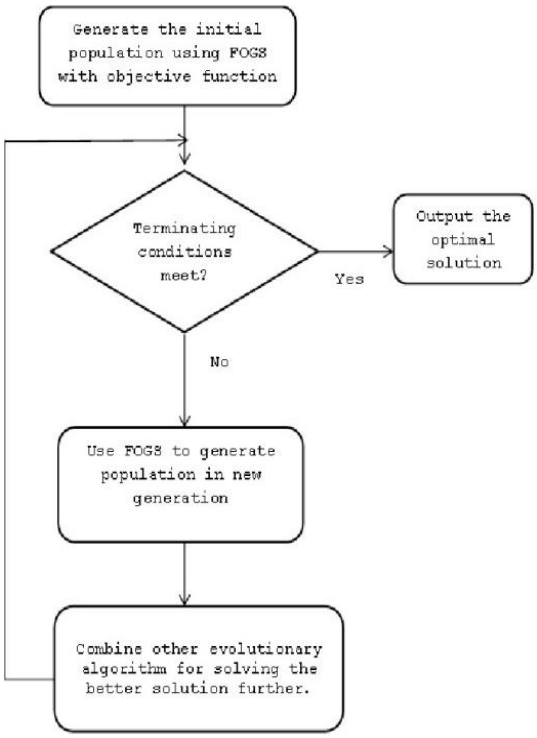


Fig. 2. A flowchart represents combination of FOGS with other evolutionary algorithm.

3. Experimental Studies

3.1. Parameter Set-up

In order to evaluate the performance of proposed algorithm, ten TSP test problems were selected: Att48, Eil51, Berlin52, St70, Eil76, Pr76, KroA100, Rd100, Eil101, and KroA200. These test cases and their results were compared with the other four techniques (PSO, ESTPSO [11], IESTPSO [12], FOGSACO[3]) based on the benchmark problems from TSP library(TSPLIB). The parameters in PSO, and ESTPSO were adopted from the report of Yan *et al*. [11] and IESTPSO were adopted from Zhang et al. [12]. The following parameters were taken after exhaustive testing. The chosen ones were those that made the best computing solutions concerning both the quality of the solutions and the computational time. The optimal solution can be obtained within 500 iterations as same as the previous works of PSO, ESTPSO, IESTPSO and FOGSACO. Therefore, the parameters selected for FOGS-PSO are summarized in Table 1.

Table 1. Parameter Setting of FOGS-PSO Algorithm

Parameters	Values
β_1	500
β_2	500
β_3	200
β_4	1
N in algorithm 1	100
Max in algorithm 3	500
c_1	0.08
c_2	0.12
M population size in algorithm 4	100
Archive R size in Algorithm 3	$0.1 \times M$
Number of experimental runs in each benchmark	50

3.2. Experimental Results

In the experiments, each of these algorithms, i.e. PSO, ESTPSO and IESTPSO was run 50 times for each benchmark problem. Our performance evaluation emphasized the total travelling distance, the execution time, and the mean number of generations used to find the possibly best solutions.

The possibly best travelling distances of the test cases found by the algorithms are shown in Tables 2-11. The number of cities in these test cases varies from 48 to 200. Each number denotes the total distance found for each problem by each algorithm. From Tables 2-11, these results are better than the results from the other compared especially traditional PSO algorithm. Especially, the results show that all algorithms combined with FOGS yield better results than original algorithms.

Table 2. For Problem Att48 with 48 Cities. The Optimal Result Is 33,522

Algorithms	**Best Result**	**Average**	**S.D.**	**Error**
PSO	34,810	36,058.82	1330.07	3.84
ESTPSO	34,286	35,090.44	297.77	2.28
IETPSO	33,842	34,741.95	299.53	0.95
FOGS-ACO	33,561	34,205.04	282.09	0.12
FOGS-PSO	**33,561**	**34,321.62**	**289.53**	**0.12**

Table 3. For Problem Eil51 with 51 Cities. The Optimal Result Is 426

Algorithms	**Best Result**	**Average**	**S.D.**	**Error**
PSO	450	467.85	20.19	5.76
ESTPSO	429	444.56	6.37	0.82

IETPSO	428	441.76	5.94	0.70
FOGS-ACO	426	436.25	5.31	0.00
FOGS-PSO	**426**	**434.52**	**5.27**	**0.00**

Table 4. For Problem Berlin52 with 52 Cities. The Optimal Result Is 7,542

Algorithms	Best Result	Average	S.D.	Error
PSO	8,157	8,288.44	136.60	8.16
ESTPSO	7,544	7,804.20	172.70	0.03
IETPSO	7,544	7,879.60	200.93	0.03
FOGS-ACO	7,546	7,581.68	47.59	0.06
FOGS-PSO	**7,542**	**7,562.82**	**36.63**	**0.00**

Table 5. For Problem St70 with 70 Cities. The Optimal Result Is 675

Algorithms	Best Result	Average	S.D.	Error
PSO	719	768.08	37.36	6.52
ESTPSO	687	709.71	16.10	1.73
IETPSO	684	716.11	18.74	1.32
FOGS-ACO	679	694.39	8.34	0.58
FOGS-PSO	**677**	**701.32**	**8.85**	**0.29**

Table 6. For Problem Pr76 with 76 Cities. The Optimal Result Is 108,159

Algorithms	Best Result	Average	S.D.	Error
PSO	118,118	124,544.80	4,522.32	9.21
ESTPSO	109,974	110,529.74	2,404.78	1.67
IETPSO	109,565	110,540.74	2,242.94	1.29
FOGS-ACO	108,864	110,886.73	1,976.53	0.65
FOGS-PSO	**108,159**	**110,556.14**	**1,875.73**	**0.00**

Table 7. For Problem Eil76 with 76 Cities. The Optimal Result Is 538

Algorithms	Best Result	Average	S.D.	Error
PSO	571.36	572.77	32.47	6.20
ESTPSO	564.07	582.44	9.92	4.84
IETPSO	560.44	572.19	7.53	4.17
FOGS-ACO	546.83	548.63	5.79	1.64
FOGS-PSO	**545.66**	**565.92**	**7.29**	**3.11**

Table 8. For Problem KroA100 with 100 Cities. The Optimal Result Is 21,282

Algorithms	Best Result	Average	S.D.	Error
PSO	23,221	23,447.83	859.42	9.11
ESTPSO	21,644	23,476.51	927.06	1.70
IETPSO	21,282	22,484.26	647.23	0.73
FOGS-ACO	21,414	21,427.20	634.28	0.62
FOGS-PSO	**21,529**	**21,676.20**	**624.26**	**1.16**

Table 9. For Problem Rd100 with 100 Cities. The Optimal Result Is 7,910

Algorithms	Best Result	Average	S.D.	Error
PSO	8,295	8,604.86	234.83	4.87
ESTPSO	8,167	8,909.82	219.71	3.25
IETPSO	7,944	8,455.85	214.42	0.43
FOGS-ACO	7,919	8,087.81	93.94	0.11
FOGS-PSO	**7,919**	**8,082.99**	**94.56**	**0.11**

Table 10. For Problem Eil101 with 101 Cities. The Optimal Result Is 629

Algorithms	Best Result	Average	S.D.	Error
PSO	688	714.63	38.16	9.52
ESTPSO	675	698.27	19.02	7.39
IETPSO	663	683.99	11.48	5.45
FOGS-ACO	633	653.65	21.46	0.70
FOGS-PSO	**633**	**694.01**	**18.39**	**0.70**

Table 11. For Problem KroA200 with 200 Cities. The Optimal Result Is 29,368

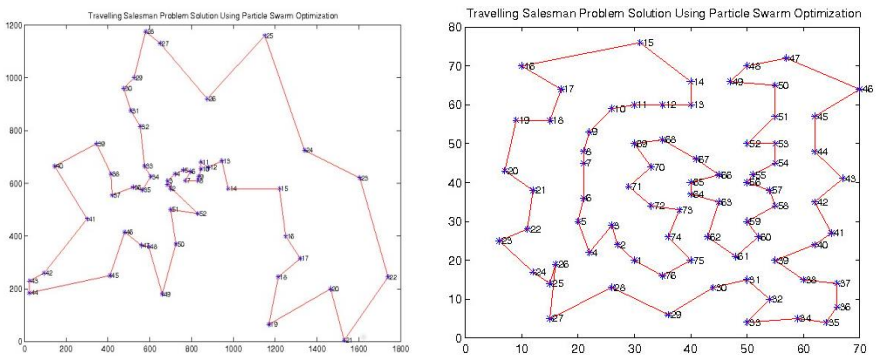
Algorithms	Best Result	Average	S.D.	Error
PSO	32,401	33,225.63	828.18	10.32
ESTPSO	31,836	32,640.29	881.34	8.40
IETPSO	31,221	31,420.59	650.62	6.31
FOGS-ACO	29,717	31,515.93	674.08	1.18
FOGS-PSO	**29,731**	**31,519.94**	**644.58**	**1.23**

Table 12 shows the average execution time used to find the possibly best result of each problem when compared with IESTPSO and FOGS-ACO. The number of iterations was 500 to assure the optimal solution can be found. Table 12 shows the summary of execution time for all algorithms; our FOGSPSO can do better than the others.

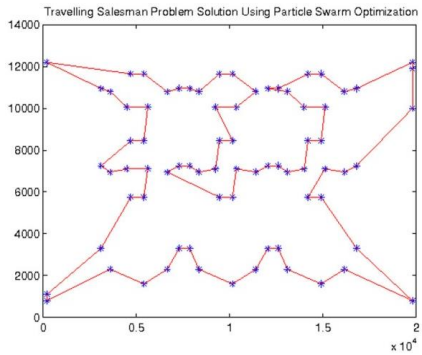
Table 12. The Comparisons of Average Execution Times (in Seconds) of All Algorithms Used to Fine the Possibly Optimal Solution and the Average Execution Time of IESTPSO, FOGS-ACO and FOGS-PSO

Problems	IESTPSO	FOGS-ACO	FOGS-PSO
Att48	263	260	169
Eil51	260	283	169
Berlin52	259	272	174
St70	394	279	274
Eil76	755	281	247
Pr76	533	279	280
KroA100	1,644	289	247
Rd100	423	223	242
Eil101	457	285	276
KroA200	528	293	307

FOGS-PSO can find the possibly best routes of some problems i.e. Berlin52, Eil76, Pr76 are shown in Fig. 3. Each circle represents a city located in the coordinates listed on the x-axis and y-axis. Each number represents the order of traversal of each problem. Fig. 4 shows the convergence speeds of FOGS-PSO. The x-axis denotes the number of iterations and y-axis denotes the total tour length as fitness value. The speed of our algorithm for some problem is shown in Fig. 4(a) Berlin52 (b) Eil76 and (c) Pr76. (a) Berlin52: number of generations = 52; distance= 7464. (b) Eil76: number of generations = 89; distance= 545.66. (c) Pr76: number of generations = 77; distance= 108,396.

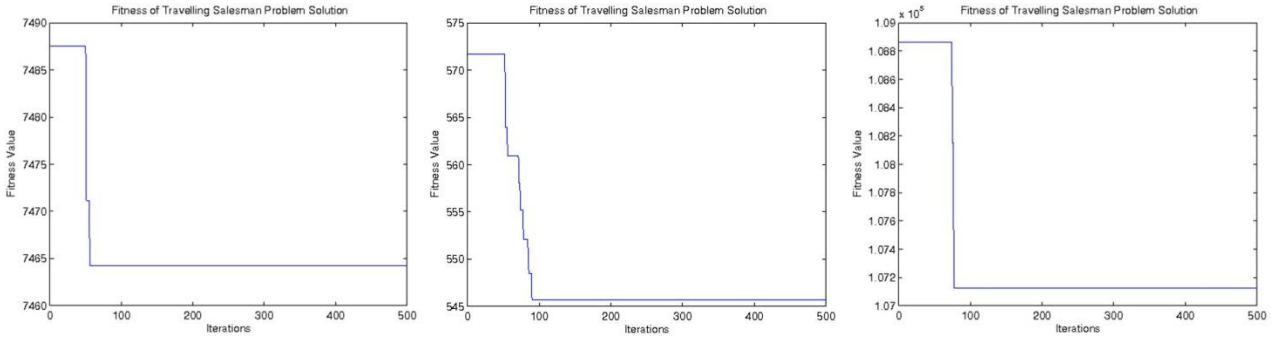


(a) Berlin52: distance=7542, the best result is 7,542. (b)Eil76: distance=545.66, the best result is 538.



(c) Pr76: distance=108,159, the best result is 108,159.

Fig. 3. Some of the problems with the best routes and their total travelling distances.



(a) Berlin52 (b) Eil76 (c) Pr76

Fig. 4. The performance evaluation in terms of number of generations found the best solution and distance.

4. Conclusion

This paper presents an approach for solving travelling salesman problem based on improved particle swarm optimization, called FOGS-PSO. This proposed algorithm contributed two considerable issues. The first issue is the FOGS to carry out searching on the manifold and generating new candidate solutions. The second issue is the application of Particle Swarm Optimization (PSO) to apply the output of FOGS phase as the initial swarm population for generating new solutions. The advantage of FOGS is to find new solutions in the same way as binary search using the backtracking on the manifold of the cost functions. In this study, FOGS can be combined with PSO to create a new solution that approaches the optimal solution. To study the performance of FOGS in the searching step, the study identified that a portion of FOGS may give a more serious effect than other based PSO. For the combining issue, this study also confirms that FOGS can combine with PSO and another technique to enhance a better solution with less time.

References

[1] Wu, H., & Yang, Y. (2004). Application of continuous Hopfield network to solve the TSP. *Proceedings of ICARCV 2004 the 8th Control, Automation, Robotics and Vision Conference* (pp. 2258-2263). Kunming, China.

[2] Hopfield, J. J., & Tank, D. W. (1985). Neural computation of decisions in optimization problems. *Biol. Cybern, 52*, 141-152.

[3] Saenphon, T., Phimoltares, S., & Lursinsap, C. (2014). Combining new fast opposite gradient search with ant colony optimization for solving travelling salesman problem. *Engineering Applications of Artificial Intelligence, 35*, 324-334.

[4] Zhu, Q. B., & Chen, S. Y. (2007). A new ant evolution algorithm to resolve TSP problem. *Proceedings of*

Sixth International Conference on Machine Learning and Applications (pp. 62-66). Ohio, Japan.

[5] Akhand, M. A. H., Akter, S., Rashid, M. A., & Yaakob, S. B. (2013). Velocity tentative particle swarm optimization to solve TSP. *Proceedings of 2013 International Conference on Electrical Information and Communication Technology (EICT)* (pp. 1-6). Khulna, Bangladesh.

[6] Deng, W., Chen, R., He, B., Liu, Y., Yin, L., & Guo, J. (2012). A novel two stage hybrid swarm intelligence optimization algorithm and application. *Soft Computing, 16(10)*, 1707-1722.

[7] Liao, Y. F., Yau, D. H., & Chen, C. L. (2012). Evolutionary algorithm to travelling salesman problems. *Computers and Mathematics with Applications, 64*, 788-797.

[8] Xiaohui, H., Xiaoyang, L., & Junlian, C. (2009). Hybrid genetic algorithm based on strategy of greedy for TSP. *Journal of Lanzhou Jiaotong University, 28(3)*, 58-61.

[9] Guvenc, U., Duman, S., Saracoglu, B., & Ozturk, A. (2011). A hybrid GAPSO approach based on similarity for various types of economic dispatch problems. *Electron Electr. Eng. Kaunas: Technologija, 2(108)*, 109-114.

[10] Kezong, T., Zuoyong, L., Limin, L., & Bingxiang, L. (2015). Multi-strategy adaptive particle swarm optimization for numerical optimization. *Engineering Applications of Artificial Intelligence, 37*, 9-19.

[11] Yan, X., Zhang, C., Luo, W., Li, W., Chen, W., & Liu, H. (2012). Solve traveling salesman problem using particle swarm optimization algorithm. *International Journal of Computer Science*, *9*, 264-271.

[12] Zhang, J., & Si, W. (2010). Improved enhanced self-tentative PSO algorithm for TSP. *Proceedings of Sixth IEEE International Conference on Natural Computation* 2010 (pp. 2638-2641). Shandong, China.

Thirachit Saenphon received the M.S. degree in information technology from King Mongkut University of Technology Thonburi, Thailand. She is currently a Ph.D. student in computer and information technology at Faculty of Science, Chulalongkorn University. She also works as a faculty staff in the Faculty of Information and Communication Technology. Her current research interest is optimization and evolutionary algorithm.