International Journal on Advanced Science Engineering Information Technology

Vol.7 (2017) No. 2
ISSN: 2088-5334

Water Flow-Like Algorithm with Simulated Annealing for Travelling Salesman Problems

Zulaiha Ali Othman, Nasser Hamed Al-Dhwai, Ayman Srour, Wu Diyi

Centre of Artificial Intelligence Technology (CAIT) Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, Selangor, 43600 Malaysia.
E-mail: zao@ukm.edu.my, laylawi@gmail.com, aymansrour@gmail.com, zzwudiyi@126.com

***Abstract*— Water Flow-like Algorithm (WFA) has been proved its ability obtaining a fast and quality solution for solving Travelling Salesman Problem (TSP). The WFA uses the insertion move with 2-neighbourhood search to get better flow splitting and moving decision. However, the algorithms can be improved by making a good balance between its solution search exploitation and exploration. Such improvement can be achieved by hybridizing good search algorithm with WFA. This paper presents a hybrid of WFA with various three neighbourhood search in Simulated Annealing (SA) for TSP problem. The performance of the proposed method is evaluated using 18 large TSP benchmark datasets. The experimental result shows that the hybrid method has improved the solution quality compare with the basic WFA and state of art algorithm for TSP.**

***Keywords*— Travelling Salesman problem; Water flow-like algorithm; Simulated Annealing; Meta-heuristics; combinatorial optimization.**

I. INTRODUCTION

The Travelling Salesman Problem (TSP) is a classic routing problem that attracts many researchers from different fields, including operational research, mathematics and several scientific and engineering fields. The TSP solution has contributed many real world applications such as decision making, navigation, stock market, transportation and problems. The TSP searches for the shortest path among a set of cities with known distances between a pairs of cities, and it can be formulated as a complete graph with a set of vertices, which is a set of edges weighted by the distance between two vertices (cities). The problem is finding the most inexpensive Hamiltonian cycle associated with visiting each city exactly once and returning to the original city. The TSP is well known as a Non-deterministic Polynomial time (NP-hard) problem, and the determination of the exact solution is difficult [1]. Previous studies have considered many approaches for solving the TSP problem, which are varied in their complexity and efficiency. There have been important advances in the development of exact and approximate algorithms. The exact algorithms are the oldest approaches used to solve the problem. Dantzig, et al. [2] [3-6] [7] [8-10]. The exact methods can only be used for small problem instances. Therefore, for large problem instances, the recent researches have been focused on applying approximate methods, i.e., greedy constructive heuristics, iterative improvement or local search heuristics and meta-heuristics, in solving the problem.

Earliest greedy constructive heuristics for the TSP build solutions from scratch by adding an unvisited city in each step based on cost-saving path. These methods are relatively fast to generate feasible solution and the solution can be iteratively improving by using local search heuristics such as 2-opt or 3-opt. Nevertheless, the main drawback of applying the local search heuristics only is the fact that the algorithm is easy to trap in local optima. Past researches has shown meta-heuristics can improved the problem, where the two algorithms such as tabu search (TS) [11], simulated annealing (SA) [12-16], etc, are popular used as local search in meta-heuristic. They have achieved some success with solving TSP in an acceptable amount of computation. Recently, Water Flow-like Algorithm (WFA) [32 - 34] has successfully applied to solve TSP [17]. The major key success of WFA for TSP (WFA-TSP) is the fact that WFA-TSP search problem solution space with dynamic population (solution search agents) size. Unlike traditional metaheuristic such as genetic algorithm or ant colony system which search problem solution space with fixed population size. The main advantages WFA is the fact that it presented a dynamic behaviour in which the WFA has the ability to adapt its population number during the optimization process according to problem instance size and complexity.

Now a day, the current research trend is move toward developing an efficient and effective algorithm for solving TSP by improving the existing meta-heuristics. Hybridization meta-heuristics with other meta-heuristics is a famous method used to improve a metaheuristic by employing the strength of other metaheuristic feature and by exploiting the complementary character of different optimization strategies. thus, having an adequate combination of complementary algorithmic concepts can be the key for achieving top performance in solving many hard optimization problems [18, 19].

In general, population based meta-heuristics are characterized by a good exploration of the problem search space, whereas single solution-based meta-heuristics are good for exploitation [20]. Therefore, the combination between both populations based meta-heuristics and single solution-based meta-heuristics have been widely recorded in the literature [18, 19, 21-25]. One of the major key usage of such combination is the fact that it can provides a good balance between exploration and exploitation, which then can improve the overall performance of the algorithm [26]. The balancing between exploration and exploitation in population based meta-heuristics is performed by increasing the algorithm exploitation capability. The algorithm might combine with strong local search or single solution meta-heuristic algorithm in order to achieve a such balance, as the single solution-based meta-heuristics are strong in solution search exploitation Zhao, et al. [27].

Simulated annealing algorithm have been widely used as local search algorithm to improve the population based meta-heuristics [28-31]. The main feature of using SA is it can improve the solution search exploitation without being trap in local optima. In this study, a combination between WFA algorithm and SA algorithm is proposed in order to improve the overall performance of WFA algorithm by balancing between exploration and exploitation. Making such balance between exploration and exploitation can be achieved by improving WFA algorithm exploitation using SA algorithm.

This paper is organised as follows. Section 2 discusses the related works including a brief background of WFA and SA algorithms performance. Section 3 details the propose hybrid WFA with SA for solving TSP problem. The experimental results and conclusion is presented in sections 4 and 5 respectively.

II. Material and Method

A. Water flow-like algorithm

Water flow-like algorithm (WFA) [32] considered as a new meta-heuristic algorithm, inspired by the natural behaviour of water flowing from higher to lower levels. The water flows can split or merge according to the surface scenery. The advantages of the WFA are that it is self-adaptive and dynamic in its population sizes and parameter settings. The solution agent size is not fixed, unlike in the traditional population based meta-heuristics. The flow number is subject to increase or decrease during the optimisation process. The population size changes are based on the problem diminution and solution quality found by the agents. However, Yang and Wang [32] describe and map the dynamic size of the solution agents based on the natural behaviour of water flows as they split, move and merge.

The first version of the WFA was developed by Yang and Wang as the bin-packing [32]. WFA also has presented is good performance is various domain such [32], manufacturing cell fraction [33] and nurse scheduling problems [34] and recent work [17] has been applied in Travel sales problem. The most advantages of WFA is the fact that its ability to reach the quality solution very fast, especially in large data set. WFA has performed more 90% faster than ACS. This is because the nature of WFA itself behave dynamic population size. The number changes over the time and the needs of agent solution optimized during the finding the solutions.

The basic operations of the WFA for solving the Travelling salesman Problem (TSP) include initialization, flow splitting and moving, flow merging, water evaporation, and water precipitation [17]. Initialization includes parameter setting and initial solution generation. The original WFA is adopted for the parameter setting [2]. The initial solution is generated by the Nearest Neighbour (NN). Normally, the process of enhancement of the WFA only starts after the initialization process. The flow splitting operation is conducted and depends on the flow momentum value. After that, the moving operation is conducted where the design is based on the type of target problem being solved. The algorithm for the flow moving process combines two types of neighbourhood structures, namely the insertion move and k-opt.

The flow merging operation is the operation that combines more than two flow moves to the same location. This operation merges more than one flow into a single flow. Masses and momentums accumulate to compose an integrated flow to reinforce the solution search. This accumulation helps the stagnant flow to escape from the trapped location. This operation checks a flow with others whether they share the same location, and if it does, the latter flow will merge into the former one. The merging operation is executed to eliminate redundant flows.

The water evaporation operation is performed after the flow merging operation. This operation aims to simulate the natural evaporation of water into the air. Water evaporation is executed when evaporation conditions are met. The WFA uses the concept of water evaporation for preparing regeneration flows to increase the wideness of a solution search. When the evaporated water accumulates to a certain amount, it will return to the ground. This process is known as the precipitation operation, and it is the natural rainfall behaviour. This operation achieves redistribution of flows that escape from the local optima and spread the solution search range. Two types of precipitation are used in the WFA known as the enforced precipitation and regular precipitation.

There are many methods that can improve the meta-heuristic solution quality such as a hybrid of different meta-heuristics, adding heuristics and using some memory to avoid redundant searches. However, this research aims to improve the local search using simulated annealing.

B. Simulated Annealing

Simulated Annealing (SA) is one of the oldest among the metaheuristic methods, which is developed for solving COP's by locating a good approximation to the global optimum of the objective function in the search space. SA was first proposed by [35, 36]. The basic idea of SA is stimulating the physical process, where the annealing process requires heating, then slowly cooling of the material in order to increase the size of its crystals and reduce their defects and then obtaining a strong crystalline structure [36, 37].

SA has obtained some degrees of successes [12-16]. Although it differs from other meta-heuristics, [12-16] SA has shown a comparative performance in improving local optimization in terms of computation time. This is due to its requirement of less memory space, in comparison to the population-based meta-heuristics. Therefore, SA has a potential to improve several meta-heuristics. As recorded in literature, SA have been widely used as local search algorithm to improve the population based meta-heuristics [28-31]. The main feature of using SA is it can improve the solution search exploitation without being trap in local optima.

C. Proposed Method

The proposed solution approach has adopted WFA-TSP basic operations of initialization, flow splitting and moving, flow merging, water evaporation and water precipitation [3]. The SA is embedded into WFA-TSP as local search procedure in order to improve solutions by making an intensive search for the best solution that produced by WFA-TSP after flow moving and splitting operation. Ayman in [17] has used 2-opt local search. The search process of the embedded SA depends on the neighbourhood structure of the TSP. There are 4 additional steps added to the original algorithm between steps 6 and 7 in [17], which earlier uses different random move of neighbourhood. However this research has used various structure such as random swap move, 2-opt move, 3-opt move and 4-opt move in SA and the step 7 in [17] move to steps 11 onwards.

Step 7: Start and initialized the SA procedure.
Step 8: Search for new neighbourhood solution of SolutionBest using random neighbourhood move. If the new solution found is better than SolutionBest, Then update the best solution and skip to step 9, else accept the new solution with probability.
Step 9: Update the temperature.
Step 10: Repeat steps 8 and 9 until the termination condition is met.

Figure 1 shows the algorithm WFA with SA for TSP adopt WFA for TSP in [17], where line 6 is call the three variants Simulated Annealing using three neighbourhood structures show in Figure 2 (a), (b) and (c).

```
1  WFA_SA_Algorithm ()
2   Generate an initial solution using NN
(nearest neighbor) //refer step1 in [17]
3   WHILE (stop criterion is false) {
4   Cal. no. of sub-flows; //refer equation 1 in
[17]
5  Assign sub-flows new locations using insertion
move //refer step 3 in [17]
6   Apply the Simulated Annealing ()
7  Distribute mass of flow to its sub-flows;
//refer equation 2 and 3 in [17]
8  Calculate the improvement in objective function;
9  Flow merging. //refer step 6 in [17]
10  Water evaporation. //refer step 7 in [17]
11  IF (rainfall required) {
12    Precipitation. //refer step 8 and step 9 in
[17]
13    Flow Merging. //refer step 6 in [17]
14    }
15  IF (new best solution found)
16      Update best solution record.
17  }
```

Fig. 1 WFA with Simulated Annealing

The SA implementation starts from a single solution, which is obtained by the best subflows after flow splitting and moving operation. It search systematically a random move generation of solution neighbourhood and allow worst move to be accepted or to be rejected according to the SA criterion. In the pseudo-code, the algorithm starts with initializing the SA parameters such as initial solution Sinitial, initial temperature T0, current temperature Tc, max iteration Maxitr and cooling rate C. Next, each iteration in the SA procedure, a stochastic approach is used to guide the search. It guides the search in the following way. If the generated neighbourhood solution S' using one move of solution S decreases the objective value or leaves it unchanged, then the move is always accepted. Formally, the solution S' is always accepted as current solution in case of $\Delta \leqslant 0$, where Δ =f(s)-f(s') and the f(s) is the value of the objective function. On the other hand, the solution s' is accepted with probability e^(Δ/T_c) in order to allow the search to explore more region in the search space and to prevent it to be trapped in local optimum.

In our implementation and after experimentation, the initial parameter for cooling rate is set to at T0=100, Tc=0.001, and max iteration is set Maxitr=50. The temperature is decreased according to the cooling rate C which was set to 0.5.

In the implemented SA, the random move generation method of solution neighbourhood can be performed using any applicable neighbourhood structure of TSP. However, in this paper we test and implement four common types of neighbourhood structure, namely, swap move (swap two randomly selected cities), 2-opt move, 3-opt move and 4 opt move. Each single neighbourhood structure are implemented and tested separately with SA. The pseudo-code for the four neighbourhood structures used in this paper are illustrated in Figures 2 (a), (b) and (c) , respectively.

```
1  SA-swap-move ()
2  Begin
3  While (Max_itr bound    reached)
4       s = flow_best.getsoliution();
5       s' = s.clone();
6       Begin
7               City_A = s'.gettour.getRandomCity();
8               City_B = s'.gettour.getRandomCity();
9               Aux = city_A;
10              City_A = City_B;
11              City_B = Aux;
12      End
13      If (s'.SolutionCost < s.SolutionCost)
14              s = s';
15      Else
16              Accept s' with probability
17      End if
18      Update temp.
19      End while
20 End
```

Fig. 2(a) Algorithm Simulated Annealing with Swap

```
1  SA-2-opt-move()
2  Begin
3  While (Max_itr bound reached)
4       s = flow_best.getsoliution();
5       s' = s.clone();
6       Begin
7       Do
8       If (Edge1 > Edge2)
9               Edge1 = Edge 2; Else Edge2 = Aux;
10      End If
11      While (Edge1 = Edge2)
12       Newtour = currenttour
13       int i =0;
14       While (i < Edge1);
15
Newtour.CityIndex[i]=currenttour.CityIndex[i];
16       i++;
17       j++;
18       End While;
19     int j =Edge2;
20     While (i < Edge2+1);
21       Newtour.CityIndex[i] =
currenttour.CityIndex[j];
22       i++;
23       j++;
24     End While;
25     While (i < tour.CityNumber);
26       Newtour.CityIndex[i] =
currenttour.CityIndex[i];
27       j++;
28     End While;
29     Return    Newtour;
30     End Do
31    If (s'.SolutionCost < s.SolutionCost)
32     s = s';
33    Else
34     Accept s' with probability
35    End If
36     Update Temp.
37   End while
38   End
```

Fig. 2(b) Algorithm Simulated Annealing with Swap

```
1  // k denote to 3opt or 4-opt
2  SA-k-opt-move()
3  Begin
4  While (Max_itr bound    reached)
5       s = flow_best.getsoliution();
6       s' = s.clone();
7       Begin
8               Int Opt = 2;
9               Int i = CurrentTour.GetRandomEdge;
10              Int j = CurrentTour.GetRandomEdge;
11              NewTour = TwoOpt(Currenttour,i,j);
12              Do
13              TempTour1=twoOpt(NewTour , i,
14              NewTour. GetRandomEdge);
15              TempTour2=twoOpt(NewTour , NewTour.
16              GetRandomEdge, j);
17              If (TempTour1.GetTourLength <
18              TempTour2.GetTourLength)
19                      NewTour= TempTour1Else
20                              Opt++;
21              While (Opt<K);
22              Return NewTour;
23                      NewTour= TempTour2;
24              End
25  End
```

Fig. 2(c): Algorithm Simulated Annealing with k-opt

III. Results and Discussion

In this section, the experimental results of WFA-SA-TSP are presented. However, to measure the performance of WFA-SA-TSP, a number of experiments were carried out using TSP benchmark datasets. Eighteen TSP benchmark datasets, available at TSPLIB [38], were used in the conducted experiments as similar in [17]. The number of cities of the datasets are ranged from 51 to 3705 cities. The WFA-TSP and WFA-SA-TSP were implemented using Java platform JDK 1.6, a Windows environment and a personal computer with an Intel core i5 (3.00 GHz CPU speed and 4 GB RAM). The WFA-TSP and WFA-SA-TSP were implemented to compare them and measure their performance. The experiments used the parameter setting as in [17].

Table 1 and Table 2 present a comparison results in terms of the best solution quality, average iterations number and computation time of the algorithms tested. The tables also show the comparison between WFA-TSP and WFA-SA-TSP in terms of the solution accuracy (in percentage) and the solution deviation of the mean values regarding the best-known solution. The p-values are provided for solution quality to see if there is any significant difference between the WFA-TSP and WFA-SA-TSP algorithms.

The experiments measured the solution quality of the WFA-SA-TSP obtained from 10 runs for each neighbourhood structure, such as swap, 2- opt, 3-opt and 4-opt. That means for each data sets, the WFA-SA-TSP was executed 40 times. The termination condition of the WFA-SA-TSP is activated once it reach 10,000 iterations for each independent run. The number of iterations required to reach the best solution was also considered. The results of top 10 out of 40 independent run that includes the best solution, the time elapsed to find the best solution, the average of the solution cost and the best neighbourhood structure were also recorded.

The results of the experiments are presented in Table 2 in which the performance of the WFA-SA-TSP is compared with the basic WFA-TSP, by considering the best solution found and the average solutions from 10 independent runs. In the table, BKS represents the best-known solution, Mean represents the average, best represents the best solution found and Best NS represents the best neighbourhood structure of 4o independent ruins. It can be seen that the results of the WFA-SA-TSP when the embedded SA use the four-neighbourhood structure are generally better than WFA-TSP in large size datasets. Where the WFA-SA-TSP outperformed the WFA-TSP 9 datasets in terms of the best solution found. These datasets are lin318, rat576, rat787, u1060, fl1400, d1655, u1817, d2103 and fl3795. In small size datasets, both the WFA-TSP and the WFA-SA-TSP can obtain the optimal solution in 9 out of 18 datasets in terms of the best solution found. The WFA-SA-TSP also outperformed the WFA-TSP in nearly all datasets in terms of the average solutions, except for datasets eil76, eil101 and ch150.

Table 1 shows the average computational time of both algorithms changes when the number of cities is increased. It can be seen from the table that the average computation time of both algorithms is clearly increasing with the larger dataset size in both algorithms. From the table, it can be deduced that with small, medium and large dataset size, the computation time of WFA-SA-TSP is only slightly higher that WFA-TSP.

TABLE I
RESULTS OF THE COMPARISON BETWEEN WFA-TSP AND WFA-SA-TSP ALGORITHMS

Dataset	BKS	WFA-TSP			WFA-SA-TSP				Improvement %
		Avg. Time	Mean	Best	Avg. Time	Mean	Best	Best NS	
eil51	426	0.22	426.4	*426*	8.58	**426.2**	*426*	Swap	0.05
eil76	538	0.55	**538**	*538*	12.11	538.8	*538*	Swap	-0.15
kroA100	21282	0.56	*21282*	*21282*	4.16	*21282*	*21282*	Swap	0.00
eil101	629	1.23	**630.7**	*629*	13.73	631	*629*	Swap	-0.05
bier127	118282	3.56	118451.4	*118282*	5.95	**118326.6**	*118282*	Swap	0.11
ch130	6110	3.45	6133.9	*6110*	19.94	**6133.8**	*6110*	4 opt	0.00
ch150	6528	3.53	**6542.2**	*6528*	15.62	6544.3	*6528*	2 opt	-0.03
kroA150	26524	4.41	26568.4	*26524*	12.96	**26550.7**	*26524*	4 opt	0.07
kroA200	29368	9.81	29438.2	*29368*	17.39	**29395.9**	*29368*	3 opt	0.14
lin318	42029	23.18	42490.1	42278	38.90	**42382.1**	**42029**	2 opt	0.25
rat575	6773	79.43	7000.8	6971	270.92	**6890.8**	**6843**	4 opt	1.57
rat783	8806	147.65	9180.2	9126	618.44	**8979.9**	**8942**	4 opt	**2.18**
u1060	224094	317.62	231401.5	230481	657.27	**229729**	**228321**	4 opt	0.72
fl1400	20127	571.01	20449	20365	1255.60	**20275.7**	**20229**	4 opt	0.85
d1655	62128	746.34	64878.6	64313	1877.80	**63740.6**	**63371**	4 opt	1.75
u1817	57201	980.08	60155	59713	2533.36	**59036.7**	**59394**	4 opt	1.86
d2103	80450	1240.28	82769.7	81769	3036.30	**82235**	**81463**	4 opt	0.65
fl3795	28772	4971.05	29578.8	29400	16415.24	**29285.6**	**29151**	4 opt	0.99

Table 2 presents a comparison of the experimental results by using the percentage deviations of the average and best solutions from the best-known solution of the WFA-TSP and WFA-SA-TSP. The percentage deviation of the average solution from the best-known solution as PDavg while the calculated deviation of the best solution from the best-known solution is denoted as PDbest. The table shows that the deviation of the average solution from the PDavg of the WFA-SA-TSP for the major datasets was significantly better than that of the other algorithms. The difference in the performance of WFA-SA-TSP in each dataset compared with that of the WFA-TSP in terms of the PDavg value. Table 2 also shows that the deviation of the best solution from the best-known solution (PDbest) of the WFA-SA-TSP is generally better than that of WFA-TSP for datasets lin318, rat576, rat787, u1060, fl1400, d1655, u1817, d2103 and fl3795. The table also demonstrates the significant of the statistical results of all datasets. It can be seen that WFA-SA-TSP with 4-opt has a significant improvement in terms of the solution quality where the p-values for the large size datasets are less than 0.05.

TABLE III
COMPARISONS OF THE SOLUTION DEVIATION OF WFA-SA-TSP RESULTS WITH WFA-TSP RESULTS

Datasets	Optimum	WFA-TSP		WFA-SA-TSP		Statistical test
		PD_{best}	PD_{avg}	PD_{best}	PD_{avg}	*p*-value
eil51	**426**	0.00	0.09	0.00	0.05	0.443
eil76	**538**	0.00	0.00	0.00	0.15	0.343
kroA100	**21282**	0.00	0.00	0.00	0.00	~
eil101	**629**	0.00	0.27	0.00	0.32	0.831
bier127	**118282**	0.00	0.14	0.00	0.04	0.065
ch130	**6110**	0.00	0.39	0.00	0.39	0.989
ch150	**6528**	0.00	0.22	0.00	0.25	0.789
kroA150	**26524**	0.00	0.17	0.00	0.10	0.156
kroA200	**29368**	0.00	0.24	0.00	0.10	0.321
lin318	**42029**	0.59	1.10	0.00	0.84	0.360
rat575	**6773**	2.92	3.36	1.03	1.74	0.000
rat783	**8806**	3.63	4.25	1.54	1.97	0.000
u1060	**224094**	2.85	3.26	1.89	2.51	0.000
fl1400	**20127**	1.18	1.60	0.51	0.74	0.000
d1655	**62128**	3.52	4.43	2.00	2.60	0.000
u1817	**57201**	4.39	5.16	3.83	3.21	0.000
d2103	**80450**	1.64	2.88	1.26	2.22	0.000
fl3795	**28772**	2.18	2.80	1.32	1.79	0.000

Figure 3 shows a general WFA-SA-TSP searching behaviour and the performance of the algorithm in terms of exploration and exploitation when using SA with swap, 2-opt, 3-opt and 4-opt move to solve ch130 dataset problem. The use of SA as local search algorithm has improved the algorithm performance in finding good solution in less amount of time, which prevents the algorithm from early convergence and avoids it to trap in local optima.

The dashed lime arrow points to the best objective values that obtained by WFA-SA-TSP in respect to WFA-TSP. The pointed area indicates that the exploitation process which using SA is performed better in this region comparing to the WFA-TSP. On the other hand, the solid line arrow point to a sample of solution exploration regions with worst solution area more that WFA-TSP which are clearly shows that the WFA-SA-TSP can explore move more solution area in the problem search space.

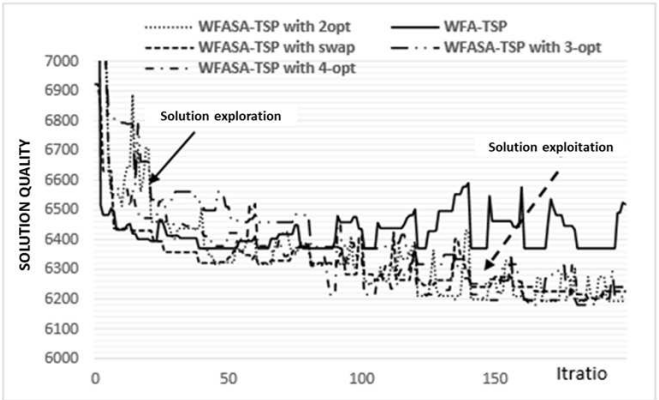

Fig. 3 A comparison of searching behaviour between the WFA-TSP and ASA-TSP with swap, 2-opt, 3-opt and 4-opt move when solve ch130 dataset

Figure 4 also shows the log file of WFA-SA-TSP for four individual datasets which include the global and the local best of solutions obtained during the first 200 iterations. As shown in the figure sketch a, b, c and d, a good balancing between exploration and exploitation process is clearly seen in data sets ch130, lin318, rat783 and d1655, respectively, where the shaded area of the local best values are fluctuating somehow in more systematic way.

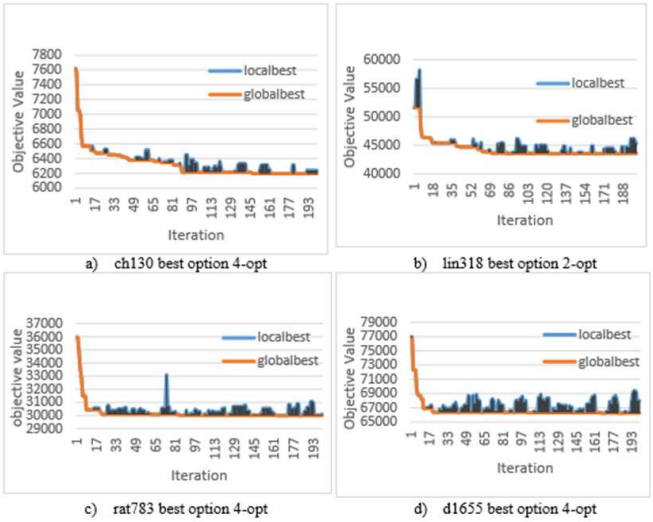

Fig. 4 Comparison of the global best and local best of the objective value during the optimization process of WFA-SA-TSP

A. *Performance of WFA-SA-TSP versus other meta-heuristics*

This section provides the comparisons among WFA-SA-TSP and Discrete Cuckoo Search Algorithm (DCS) [39] and Genetic Simulated Annealing Ant Colony System with Particle Swarm Optimization Techniques (GSAACS-PSOT) by Chen & Chien (2011), Pastiand Castro algorithm by Pasti & De Castro (2006) and Masutti,and Castro algorithm by Masutti & de Castro (2009) in [17] in terms of solution quality using in 18 TSP datasets (see Table 3).

TABLE IIIII
RESULTS OF THE COMPARISONS BETWEEN WFA-SA-TSP AND OTHER META-HEURISTICS

TSP instance	BKS	Pastiand Castro algorithm (Pasti & De Castro 2006)		Masutti,and Castro algorithm (Masutti & de Castro 2009)		GSAACS-PSO (Chen & Chien 2011)		DCS (Ouaarab et al. 2014)		WFA-SA-TSP	
		Mean	Best	Mean	Best	Mean	Best	Mean	Best	Mean	Best
eil51	426	438.70	429	437.47	427	427.27	427	426	426	**426.2**	**426**
eil76	538	556.10	542	556.33	541	540.2	538	**538.03**	538	538.8	538
kroA100	21282	21868.00	21369	21522.73	21333	21370.47	21,282	21,282	21,282	**21282**	21282
eil101	629	654.83	641	648.63	638	635.23	630	**630.43**	629	631	**629**
bier127	118282	121780.00	118760	120886.30	118970	119421.8	118282	118359.6	118,282	**118326.6**	118282
ch130	6110	6291.80	6142	6282.40	6145	6205.63	6141	6,135.96	6110	**6133.8**	6110
ch150	6528	6753.20	6629	6738.57	6602	6563.7	6528	6,549.90	6,528	**6544.3**	6528
kroA150	26524	27346.00	26932	27355.97	26678	26899.2	26_524	26,569.26	26,524	**26550.7**	26524
kroA200	29368	30258.00	29594	30190.27	29600	29738.73	29383	29,446.66	29,382	**29395.9**	**29368**
lin318	42029	43705.00	42975	43696.87	42834	43002.9	42487	42,434.73	42,125	**42382.1**	**42029**
rat575	6773	7125.10	7039	7115.67	7047	6933.87	6891	6,956.73	6,896	**6890.8**	**6843**
rat783	8806	9326.30	9185	9343.77	9246	**9079.23**	8988	9,043	9,109.26	8979.9	**8942**
fl1400	20127	21071.00	20745	21110.00	20851	21349.63	20_593	-	-	20275.7	**20229**
d1655	62128	71432.00	70823	72113.17	70918	65621.13	64_151	-	-	**63740.6**	**63371**

In this table, the best results are marked in bold. It can be seen that the experimental results of WFA-SA-TSP in the 18 datasets is generally better than Pasti and Castro (2006), Masutti and Castro (2009), GSAACS-PSOT (2011) and DCS [39] algorithms. The WFA-SA-TSP also better compare with discrete invasive weed optimization algorithm, in all three data sets shown in [40].

IV. CONCLUSION

WFA for TSP has shown its abilities in solving TSP, as it characterized with dynamic population size which make it more suitable of solving variant instance size of TSP without reconfiguring the parameter setting. In this paper, we propose an improved version of WFA for solving TSP problem (WFA-SA-TSP). The performance of WFA- SA-TSP was tested using 18 TSP benchmark datasets considering the best solution, average solution, and standard deviation, the percentage deviation of the average solution to the best-known solution and the percentage deviation of the best solution to the best-known solution. The experimental results show that the WFA-SA-TSP is generally outperforms the WFA-TSP. This study demonstrates that the WFA-TSP. can be subject for further improvement in terms of solution search exploration and exploitation using various neighborhood structure.

Since WFA-TSP consists of several components that influencing in the algorithm therefore many potential improvements can be made using the WFA algorithm for the TSP especially improving the water flow splitting and movement using more better neighbor search strategies.

REFERENCES

[1] E. L. Lawler, *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley-Interscience Series in Discrete Mathematics, 1985.

[2] Nur Ariffin Mohd Zin, Siti Norul Huda Sheikh Abdullah, Noor Faridatul Ainun Zainal, Esmayuzi Ismail, *A Comparison of Exhaustive, Heuristic and Genetic Algorithm for Travelling Salesman Problem in PROLOG.* International Journal on Advanced Science, Engineering and Information Technology (IJASEIT), vol. 2, 2012.

[3] Mansour Alssager, Zulaiha Ali Othman, Masri Ayob, Cheapest Insertion Constructive Heuristic based on Two Combination Seed Customer Criterion for the Capacitated Vehicle Routing Problem. Mathematical programming, International Journal on Advanced Science, Engineering and Information Technology (IJASEIT), vol. 7, no. 1, 2017.

[4] T. H. C. Smith, V., Srinivasan, and G. Thompson, *Computational performance of three subtour elimination algorithms for solving asymmetric travelling salesman problems.* Annals of Discrete, 1977.

[5] G. Carpeneto and P. Toth, *Some new branching and bounding criteria for the asymmetric travelling salesman problem.* Management Science, pp. 736-743, 1980.

[6] E. Balas and N. Christofides, *A restricted Lagrangean approach to the travelling salesman problem.* Mathematical programming, vol. 21, pp. 19-46, 1981.

[7] J. D. Little, K. G. Murty, D. W. Sweeney, and C. Karel, *An algorithm for the travelling salesman problem.* Operations research, vol. 11, pp. 972-989, 1963.

[8] H. Crowder and M. W. Padberg, *Solving large-scale symmetric travelling salesman problems to optimality.* Management Science, pp. 495-509, 1980.

[9] G. Reinelt, *The travelling salesman: computational solutions for TSP applications.* Springer-Verlag, 1994.

[10] M. Grötschel and O. Holland, *Solution of large-scale symmetric travelling salesman problems.* Mathematical programming, vol. 51, pp. 141-202, 1991.

[11] F. Glover, *Tabu search-part I*. ORSA Journal on computing, vol. 1, pp. 190-206, 1989.
[12] T. Peng, W. Huanchen, and Z. Dongmo, *Solving the Travelling Salesman Problem by Simulated Annealing*. Journal of Shanghai Jiaotong University, 1995.
[13] E. Bonomi and J. L. Lutton, *The N-city travelling salesman problem: Statistical mechanics and the Metropolis algorithm*. SIAM review, pp. 551-568, 1984.
[14] B. L. Golden and C. C. Skiscim, *Using simulated annealing to solve routing and location problems.* Naval Research Logistics Quarterly, vol. 33, pp. 261-279, 1986.
[15] S. Nahar, S. Sahni, and E. Shragowitz, *Simulated annealing and combinatorial optimization.* pp. 293-299, 1986.
[16] C. C. Lo and C. C. Hsu, *An annealing framework with learning memory.* Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, vol. 28, pp. 648-661, 1998.
[17] Ayman Srour, Zulaiha Ali Othman, Abdul Razak Hamdan, *A Water Flow-Like Algorithm for the Travelling Salesman Problem.* Advances in Computer Engineering, 2014.
[18] C. Blum, J. Puchinger, G. R. Raidl, and A. Roli, *Hybrid metaheuristics in combinatorial optimization: A survey*. Applied Soft Computing, vol. 11, pp. 4135-4151, 2011.
[19] E. G. Talbi, *Meta-heuristics: From Design to Implementation: 2009*. Wiley Publishing, 2009.
[20] C. Blum, J. Puchinger, G. Raidl, and A. Roli, *A brief survey on hybrid meta-heuristics.* Proceedings of BIOMA, pp. 3-18, 2010.
[21] C. Blum, A. Roli, and M. Sampels, *Hybrid meta-heuristics: An emerging approach to optimization.* vol. 114: Springer, 2008.
[22] Moaath Shatnawi, Mohammad Faidzul Nasrudin, Shahnorbanun Sahran, A new initialization technique in polar coordinates for Particle Swarm Optimization and Polar PSO. International Journal on Advanced Science, Engineering and Information Technology (IJASEIT), vol. 7, no. 1, 2017.
[23] C. Grosan, A. Abraham, and H. Ishibuchi, *Hybrid evolutionary algorithms*. Springer Publishing Company, Incorporated, 2007.
[24] E.-G. Talbi, *A taxonomy of hybrid meta-heuristics.* Journal of heuristics, vol. 8, pp. 541-564, 2002.
[25] M. Lozano and C. *García-Martínez, Hybrid meta-heuristics with evolutionary algorithms specializing in intensification and diversification: Overview and progress report*. Computers & Operations Research, vol. 37, pp. 481-497, 2010.
[26] G. Zhao, W. Luo, H. Nie, and C. Li, *A Genetic Algorithm Balancing Exploration and Exploitation for the Travelling Salesman Problem.* pp. 505-509, 2008.
[27] K. Ganesh and M. Punniyamoorthy, *Optimization of continuous-time production planning using hybrid genetic algorithms-simulated annealing.* The International Journal of Advanced Manufacturing Technology, vol. 26, pp. 148-154, 2005.
[28] W.-M. Hung, W.-C. Hong, and T.-B. Chen, *Application of hybrid genetic algorithm and simulated annealing in a SVR traffic flow forecasting model.* Evolutionary Computation, 2009. CEC'09. IEEE Congress. pp. 728-735, 2009.
[29] X.-G. Li and X. Wei, *An improved genetic algorithm-simulated annealing hybrid algorithm for the optimization of multiple reservoirs.* Water Resources Management, vol. 22, pp. 1031-1049, 2008.
[30] G. Nallakumarasamy, P. Srinivasan, K. V. Raja, and R. Malayalamurthi, *Optimization of Operation Sequencing in CAPP Using Superhybrid Genetic Algorithms-Simulated Annealing Technique.* ISRN Mechanical Engineering, vol. 2011.
[31] F. C. Yang and Y. P. Wang, *Water flow-like algorithm for object grouping problems.* Journal of the Chinese Institute of Industrial Engineers, vol. 24, pp. 475-488, 2007.
[32] T. H. Wu, S. H. Chung, and C. C. Chang, *A water flow-like algorithm for manufacturing cell formation problems*. European Journal of Operational Research, vol. 205, pp. 346-360, 2010.
[33] P. S. Shahrezaei, R. T. Moghaddam, M. Azarkish, and A. Sadeghnejad- Barkousaraie, *Water Flow-Like and Differential Evolution Algorithms for a Nurse Scheduling Problem.* American Journal of Scientific Research, pp. 12- 32, 2011.
[34] S.-M. Chen and C.-Y. Chien, *Solving the travelling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques.* Expert Systems with Applications, vol. 38, pp. 14439-14450, 2011.
[35] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, *Optimization by simulated annealing.* Science, vol. 220, pp. 671, 1983.
[36] T. El-Ghazali, *Meta-heuristics: from design to implementation.* Jonh Wiley and Sons Inc., Chichester, 2009.
[37] S. Lin and B. W. Kernighan, *An effective heuristic algorithm for the travelling-salesman problem.* Operations research, pp. 498-516, 1973.
[38] G. Reinelt, *TSPLIB - A travelling salesman problem library*. ORSA Journal on computing, vol. 3, pp. 376-384, 1991.
[39] A. Ouaarab, B. Ahiod, X. S. Yang, *Discrete cuckoo search algorithm for the travelling salesman problem.* Neural Computing and Applications; 24(7-8): 1659-1669, 2014.
[40] Y. Q. Zhou, H. C. Luo, H. Chen, A. P. He, J. Z. Wu, *A discrete invasive weed optimization algorithm for solving traveling salesman problem.* Neurocomputing; 151: 1227-1236, 2015.