A Systematic Circular Weight Initialisation of Kohonen Neural Network for Travelling Salesman Problem

Jawad Muhammad[1], Halis Altun[2]

[1]Electrical and Computer Engineering, Graduate School of Mevlana University, Konya, Turkey
jmuhammad@mevlana.edu.tr
[2]Electrical and Electronics Engineering, Karatay University, Konya, Turkey
halis.altun@karatay.edu.tr

Abstract

Self-organising neural networks or simply Kohonen networks have since been employed by researchers in solving the travelling salesman problem. However, with these networks, the final tour length as well as the convergence time, largely depends on the initial weights of the networks. In this paper, systematic initialisation of the network weight in a circle is presented, that involves randomly initialising the weights to be along a circular path, with centroid equals the centroid of all the cities. Our major contribution is on having the circle exhibit different radius on each test run, in order to effectively encompass all the cities. This will increase the chance of attaining the global solution by preventing effect of local minima due to some cities that may be poorly covered by the circle having specific radius. Furthermore, a system of determining the most efficient number of neurons needed in the circle is devised. It was experimentally found that, having a circle of size 1.5 times the number of cities gives the best performance. Results generated indicate an average deviation error of 2.74% from the optimal solutions of 13 TSPLIB benchmark TSP instances.

1. Introduction

Travelling salesman problem (TSP) is one of the most popular combinatorial optimization problems. It simply describes a salesman who must travel between N cities. Typical study in TSP involves optimizing the tour length whereby the tour with the minimal length is regarded as the optimal tour. Various approaches had been proposed to solve TSP. Some of the methods include the use of evolutionary algorithms such as genetic algorithm (GA) [1-2]. With GA, the problem is solved by formulating tours as the population members of the GA and iteratively evaluating these tours to arrive at the final solution. Ant –colony algorithm had also been used to solve TSP. Ghafurian, S. et al. (2011) propose a solution to a fixed destination multiple travelling salesmen using ant-colony [3]. Dorigo, M. et al. propose a system of ant that generate tours and the ant co-operate between each order to evaluate these tours using indirect form of communication mediated by a pheromone they deposit on the edges of the TSP graph while building the tours[4]. Heuristics and exact methods have also been used effectively to solve TSP [5-7]. Among most methods for solving TSP, Self-organizing map (SOM) neural network methods had proven to be among the efficient methods [8-12]. Hueter, G. J. (1988, July) first propose the use of a ring of nodes that adapts via Kohonen learning to form a complete tour through N cities in the plane [9]. Bai, Y., Zhang et al. (2006) propose an initialization technique of SOM using a rhombic frame [10]. Although, a lot of approaches had been proposed that solve TSP, as with any combinatorial optimization methods, a more reduced optimal tour length, faster and more efficient computation methods are still needed.

In this paper, we present an approach of solving TSP by formulating the tour as inter-connection of a circle of neurons in an SOM neural network. Through repeated experimental test simulations, we establish a fact about the most suitable number of neurons needed in the SOM network. We also compare our implemented approach with some published results.

In section II, the proposed algorithm is presented and in section III results and experiments performed will be explained.

2. The Algorithm

2.1. Solving TSP using Kohonen SOM

Kohonen type neural network belongs to a special class of competitive neural networks, where each neuron competes with others to get activated. The competition leads to the activation of only one output neuron (termed as the winning neuron) at a given time. The main objective of SOM is to capture a given pattern or topology in an input data. SOM neural networks are normally trained by a competitive training algorithm, were learning is restricted to only the winning neuron and its neighbours. The neighbours are determined by a given neighbourhood function. SOM network structure comprises of one input layer and one output layer. Each neuron in the input layer is connected to all the neurons in the output layer as shown

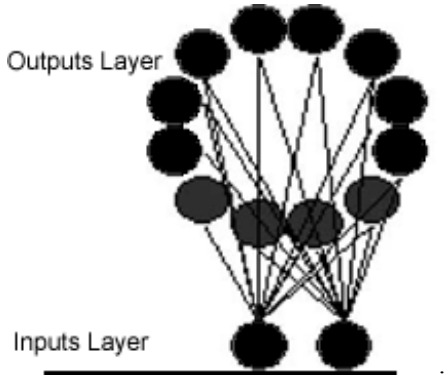

Fig. 1. Simple SOM structure for TSP

in Fig 1. Output neurons are arranged, such that they maintain the topology of the input data.

SOM can be used to solve TSP by creating SOM network with two neurons at the input layer and *n* neurons at the output layer, where *n* is equal to the total number of cities. To generate a TSP solution, the network is trained by repeatedly passing of the coordinates of a randomly selected city as inputs to the network until the network converges. When a city is passed, a competition will be held and a neuron which is closest to the city will be chosen as the *winning neuron*. The *winning neuron* serve as the mirror image of the city and its position in the network is associated with the position of the city in the final tour. The weight of the *winning neuron* and its neighbours will then be updated. After passing all the cities, a tour is created by arranging the cities according to the position of their associated *winning neurons*.

2.2. Proposed Algorithm

We proposed SOM that can solve the TSP problem as depicted by the flowchart in Fig. 2. A single test run of the algorithm begins by loading of the cities co-ordinates. Then, the distance of each city to all other cities is calculated and stored in a distance matrix. This is necessary in order to avoid repetition in calculating the distance between cities. Angles between 0 and 360 are randomly generated. Number of angles generated equals the number of cities. The centroid and radius of the cities were calculated using (1), (2) and (3) respectively.

$$x_{centroid} = x_{min} + (x_{max} - x_{min})/2 \quad (1)$$

$$y_{centroid} = y_{min} + (y_{max} - y_{min})/2 \quad (2)$$

$$Initial\ Radius = \sqrt{(x_{centroid} - x_{min})^2 + (y_{centroid} - y_{min})^2} \quad (3)$$

Where x_{min} , y_{min} is the minimum cities co-ordinates, x_{max} , y_{max} is the maximum city co-ordinates and $x_{centroid}$, $y_{centroid}$ is the centroid of the cities. Using the angle, radius and centroid, circle co-ordinate can be generated using (4) and (5).

$$x_i = x_{centroid} + (Radius \times \cos \emptyset_i) \quad (4)$$

$$y_i = y_{centroid} + (Radius \times \sin \emptyset_i) \quad (5)$$

Where x_i and y_i are the x and y co-ordinates of the circle at point i. and $\emptyset_i$ is the angle at point i. The circle co-ordinates are assigned as the initial values of the weights in the network.

The network training begins by generating a list containing all the cities in a random order. Each city from the list will be passed to the network. When a city co-ordinate is supplied to the network, the weight that is closest to the city is selected as the *winning neuron*. If the *winning neuron* had already been selected before by another city during this iteration, a new neuron will be created and added to the network using (6).

$$w_{new} = (w_{win-1} + w_{win} + w_{win+1})/3 \quad (6)$$

The newly created neuron will be marked as already been selected neuron and the network remain unchanged with no weight updated. If the *winning neuron* has not been selected before by any neuron during the present iteration, the weight of the winning neuron is updated according to (7), (8) and (9) adopted from [10].

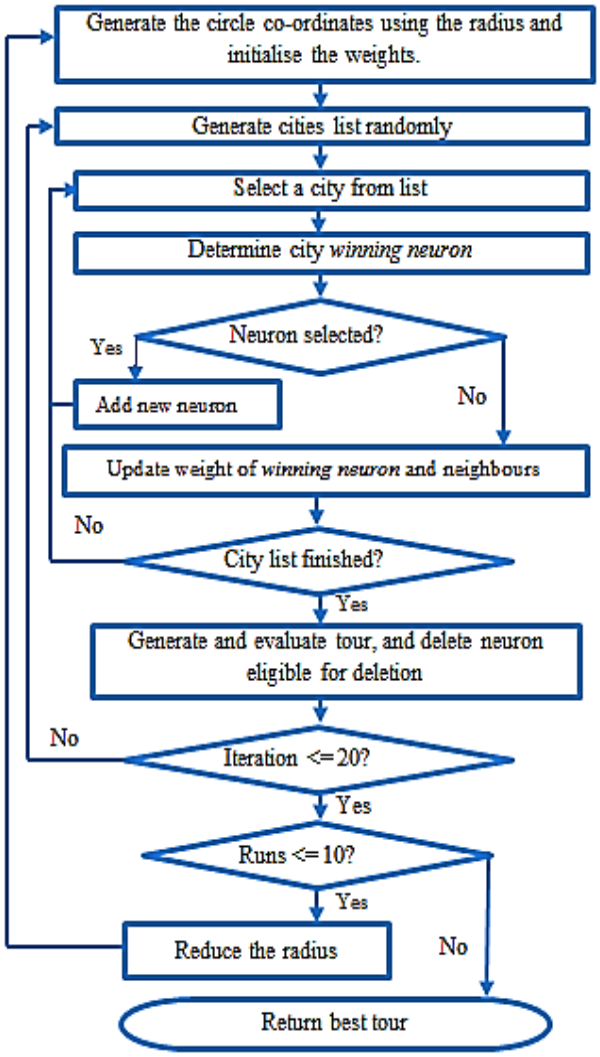

Fig. 2. Flowchart for the proposed algorithm

$$d = \min \{||j - J||,\ m - ||j - J||\} \quad (7)$$

$$f(\sigma, d) = e^{-d^2/\sigma^2} \quad (8)$$

$$w_j^{new} = w_j^{old} + \alpha f(\sigma, d)\ (c_i - w_j^{old}) \quad (9)$$

Where d is the cardinal distance measured along the circle ring from the *winning neuron* at J to a neuron at j, m is the total number of neurons, σ is the user defined standard deviation, w_j^{new}, w_j^{old} is the new and old weight value of the neuron j respectively, α is the learning rate, and c_i is the selected city i.

The process is repeated for all the cities in the list. To complete iteration, all the cities in the list have to be processed by the network. A new iteration begins by generating new city list randomly. A neuron is deleted if it has not been selected as a winning neuron for two iterations. At the end of iteration, a tour is generated by arranging the cities according to the position of their associated *winning neuron* in the circle. A tour is evaluated by calculating the distance covered by the tour. After twenty iterations, one test run is completed and the tour with the minimal distance is selected as the best tour for the run. A new

test run begins by calculating new radius using (10) and the previous procedures are repeated. The algorithm terminate when it reaches the 10^{th} run. Fig. 3 shows cities from one of the TSP instances and how the weights are randomly distributed around the cities for the 10 runs with different radius.

$$Radius_{new} = Radius_{old} - (Radius_{old}/20) \quad (10)$$

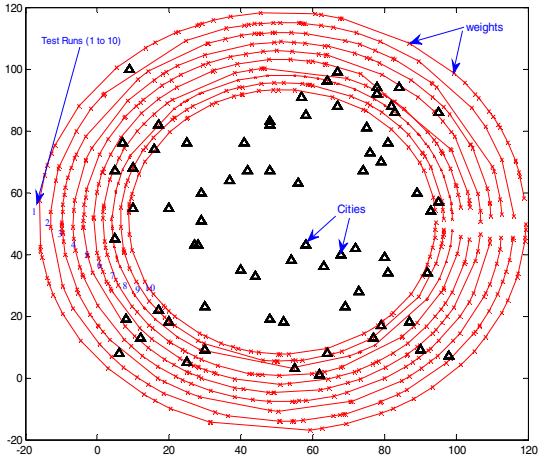

Fig. 3. Cities weights distribution in circles with different radii

3. Results, Simulations and Discussion

3.1. Simulations

Simulations were performed on a windows 8.1 Intel® Core™ i7-4510U CPU @ 2.00GHz 2.60 GHz computer. The initial values of the parameters used were $\sigma = 10$ *and* $\alpha = 1$. The parameters were made to vary during the training iterations according to (11) and (12) adopted from [11].

$$\alpha(t+1) = \alpha(t)(1 - \frac{t}{T}) \quad (11)$$

$$\sigma(t+1) = 1 + (\sigma(t) - 1)(1 - \frac{t}{T}) \quad (12)$$

Where t is the iteration step and T the maximum iteration that can be reached during training is $T = 20$.

For each result obtained, 20 test runs were performed. The solution with the minimum tour length is chosen as the final result.

3.2. Results and Discussions

One of the important parameters that affect the final results is the maximum number of neurons to be allowed in the circles. To determine this value, we conducted some test on four TSP benchmark instances of TSPLIB TSP from the work of Reinelt, G. (1991) [13]. We make the number of neurons fixed without addition or deletion. The aim was to find the number of neurons that produce the least deviation error. The variation was in form of multiples of the number of cities and results obtained are shown in Fig. 4 and Fig.5.

Results obtained indicates that, the higher the number of neurons the higher the error. However, if observed from Fig.4, the error drops after × 1 (when number of neurons equal to number of cities) and increases after × 2 (2 times number of cities). This signifies that, choosing a value of number of neurons between a value equals number of cities and value twice number of cities gives the least deviation error. This is evident from Fig.4 and Fig. 5 where × 1.5 (1.5 times number of cities) gives the best results.

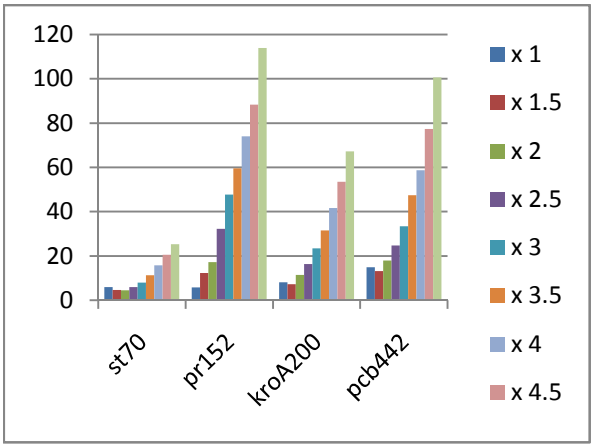

Fig. 4. Average distance results of running proposed algorithm on four TSPLIB TSP benchmark instances with some defined fixed number of neurons in multiples of number of cities

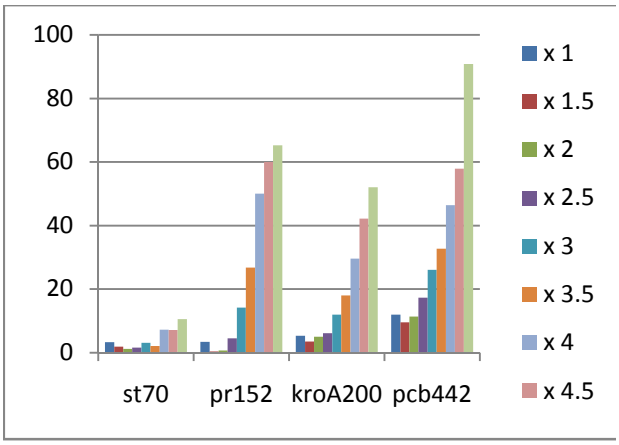

Fig. 5. Best distance results of running proposed algorithm on four TSPLIB TSP benchmark instances with some defined fixed number of neurons in multiples of number of cities

Table 1 shows the results of running the proposed algorithm with the maximum number of neurons equals 1.5 times the number of cities. Table 2 shows the comparison of the deviation error of the proposed algorithm of 13 TSPLIB TSP instances and some published results. SETSP - SOM (efficiently applied to the TSP) algorithm from [12], KL – KNIES TSP Local (Kohonen Network Incorporating Explicit Statistics Local) from [15] and KG- KNIES TSP Global (Kohonen Network Incorporating Explicit Statistics Global) from [15] were compared with our results. It could be observed that our

proposed algorithm has shown better results in most of the TSP instances with a reduced average deviation error.

Table 1. Results of the proposed algorithm on 13 TSP instances

	TSP instance	Optimal distance	Proposed algorithm best distance
1	st70	675	684.32
2	pr152	73,682	74154.14
3	kroA200	29,368	30149.62
4	pcb442	50,778	54368.62
5	pr107	44,303	44484.77
6	pr136	96,772	99906.46
7	rat195	2,323	2480.01
8	eil76	538	564.33
9	eil51	426	437.61
10	lin105	14,379	14475.71
11	rd100	7,910	8031.74
12	bier127	118,282	121565.80
13	pr124	59,030	59504.74

Table 2. Comparison of the deviation error of the proposed algorithm of 13 TSPLIB TSP instances and some published results

TSP instance	Optimal Distance	KL	KG	SETSP	Proposed algorithm
bier127	118,282	2.76	3.08	1.85	**2.78**
eil51	426	2.86	2.86	2.22	**2.73**
eil76	538	4.98	5.48	4.23	**4.89**
kroA200	29,368	2.84	3.67	3.12	**2.66**
lin105	14,379	1.98	1.29	1.3	**0.67**
pcb442	50,778	11.07	10.45	10.16	**7.07**
pr107	44,303	0.73	0.42	0.41	**0.41**
pr136	96,772	4.53	5.15	4.4	**3.24**
pr152	73,682	0.97	1.29	1.17	**0.64**
rat195	2,323	12.24	11.92	11.19	**6.76**
rd100	7,910	2.09	2.62	2.6	**1.54**
st70	675	1.51	2.33	1.6	**1.38**
pr124	59,030	-	-	-	**0.8**
Average :		4.05	4.21	3.69	2.74

6. Conclusions

In this work, we present a systematic method of varying the radius of the circular ring used in initializing of the neurons weight during kohonen neural network training. We also present a neuron addition method of assigning the average value of the winning neuron with its left and right neighbour as the value of the newly added neuron. We have established the fact that, using 1.5 times the number of cities as the maximum number of neurons gives the best result.

7. References

[1] G., P. Dinh., H. T. T. Binh, & B. T. Lam (2015). New Mechanism of Combination Crossover Operators in Genetic Algorithm for Solving the Traveling Salesman Problem. In *Knowledge and Systems Engineering* (pp. 367-379). Springer International Publishing.

[2] A., Murat, & N. Allahverdi, (2011). Development a new mutation operator to solve the traveling salesman problem by aid of genetic algorithms. *Expert Systems with Applications*, *38*(3), 1313-1320.

[3] G. Soheil, & N. Javadian (2011). An ant colony algorithm for solving fixed destination multi-depot multiple traveling salesmen problems. *Applied Soft Computing*, *11*(1), 1256-1262.

[4] D. Marco., & L. M. Gambardella (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *Evolutionary Computation, IEEE Transactions on*, *1*(1), 53-66.

[5] R. Durbin & D. Willshaw (1987). An analogue approach to the travelling salesman problem using an elastic net method. *Nature*, (326), 689-91.

[6] C. H. Papadimitriou (1977). The Euclidean travelling salesman problem is NP-complete. *Theoretical Computer Science*, *4*(3), 237-244.

[7] M. Nenad., D. Urošević, , & A. Ilić (2012). A general variable neighborhood search for the one-commodity pickup-and-delivery travelling salesman problem.*European Journal of Operational Research*, *220*(1), 270-285

[8] F. Favio, & R. Walker (1991). A study of the application of Kohonen-type neural networks to the travelling salesman problem. *Biological Cybernetics*,*64*(6), 463-468.

[9] G. J. Hueter (1988, July). Solution of the traveling salesman problem with an adaptive ring. In *Neural Networks, 1988., IEEE International Conference on* (pp. 85-92). IEEE.

[10] B. Yanping., Z. Wendong, & Z. Jin (2006). An new self-organizing maps strategy for solving the traveling salesman problem. *Chaos, Solitons & Fractals*, *28*(4), 1082-1089.

[11] X. Xinshun, , Z. Jia, J. Ma, & J. Wang (2008, October). A Self-Organizing Map Algorithm for the Traveling Salesman Problem. In *Natural Computation, 2008. ICNC'08. Fourth International Conference on* (Vol. 3, pp. 431-435). IEEE.

[12] V. F. Carvalho, A. D. D. Neto & J. A. F.Costa (2003). An efficient approach to the travelling salesman problem using self-organizing maps. *International journal of neural systems*, *13*(02), 59-66.

[13] G. Reinelt (1991). TSPLIB—A traveling salesman problem library. *ORSA journal on computing*, *3*(4), 376-384.

[14] F. Jan, M. Kulich, V. Vonásek & L. Přeučil (2011). An application of the self-organizing map in the non-Euclidean Traveling Salesman Problem.*Neurocomputing*, *74*(5), 671-679.

[15] A. Necati, B. J. Oommen, & I. K. Altınel (1999). The Kohonen network incorporating explicit statistics and its application to the travelling salesman problem. *Neural Networks*, *12*(9), 1273-1284.