A Study of Traveling Salesman Problem Using Fuzzy Self Organizing Map

Arindam Chaudhuri[1] and Kajal De[2]
[1]NIIT University, Neemrana,
[2]Netaji Subhas University, Kolkata,
India

1. Introduction

Traveling Salesman Problem (TSP) is classical and most widely studied problem in Combinatorial Optimization **(Applegate D. L. et al., 2006)**. It has been studied intensively in both Operations Research and Computer Science since 1950s as a result of which a large number of techniques were developed to solve this problem. Much of the work on TSP is not motivated by direct applications, but rather by the fact that it provides an ideal platform for study of general methods that can be applied to a wide range of Discrete Optimization Problems. Indeed, numerous direct applications of TSP bring life to research area and help to direct future work. The idea of problem is to find shortest route of salesman starting from a given city, visiting n cities only once and finally arriving at origin city. The investigation question which arises is:

In what order should the cities be visited such that the distance traveled is minimized?

TSP is represented by complete edge-weighted graph $G=(V,E)$ with V being set of $n=|V|$ nodes or vertices representing cities and $E \subseteq V \times V$ being set of directed edges or arcs. Each arc $(i,j) \in E$ is assigned value of length d_{ij} which is distance between cities i and j with $i,j \in V$. TSP can be either asymmetric or symmetric in nature. In case of asymmetric TSP, distance between pair of nodes i,j is dependent on direction of traversing edge or arc i.e. there is at least one arc (i,j) for which $d_{ij} \neq d_{ji}$. In symmetric TSP, $d_{ij} = d_{ji}$ holds for all arcs in E. The goal in TSP is thus to find minimum length Hamiltonian Circuit **(Cormen T. H. et al., 2001)** of graph, where Hamiltonian Circuit is a closed path visiting each of n nodes of G exactly once. Thus, an optimal solution to TSP is permutation π of node indices $\{1,.......,n\}$ such that length $f(\pi)$ is minimal, where $f(\pi)$ is given by,

$$f(\pi) = \sum_{i=1}^{n-1} d_{\pi(i)\pi(i+1)} + d_{\pi(n)\pi(1)}$$

TSP is NP-hard problem as the search space is huge viz. n! Thus, it is not possible to check all solutions for city sets with many thousands of cities **(Korte B. H. & Vygen J., 2008)**. Hence, a fast and effective heuristic method is needed. Based on a deterministic approach, the world record setting TSP solution is by **(Applegate D. L. et al.,1995)** which has solved instances as large as 24,978 cities to optimality. Trying to solve the course of exponentials parallel implementations of TSP were realized **(Christof T. & Reinelt G., 1995)**. However, for practicability reasons specifically for large numbers of cities, heuristic approaches for solving TSP are very popular, which try to produce an optimal or close to optimal solution. It arises as sub-problem in many transportation and logistic applications **(Chaudhuri A., 2007)**, for example the problem of arranging school bus routes to pick up children in a district. This application is of important significance to TSP since it provides motivation for Merrill Flood one of the pioneers of TSP research in 1940s. A second application from 1940s involved transportation of farming equipment from one location to another leading to mathematical studies by P. C. Mahalanobis and R. J. Jessen. More recent applications involve scheduling of service call at cable firms, delivery of meals to homebound persons, scheduling of stacker cranes in warehouses, routing of trucks for parcel post pickup etc. Although transportation applications are most natural setting for TSP, simplicity of the model has led to many interesting applications in other areas. A classic example is scheduling of machine to drill holes in circuit boards where holes to be drilled are cities and cost of travel is the time it takes to move the drill head from one hole to next.

TSP has some direct importance, since quite a lot of practical applications can be put in this form. It also has theoretical significance in Complexity Theory **(Garey M. & Johnson D., 1990)** since TSP is one of the classes of NP-Complete Combinatorial Optimization Problems **(Korte B. H. & Vygen J., 2008)** which are difficult optimization problems where the set of feasible solutions or trial solutions which satisfy constraints of problem but are not necessarily optimal is finite, though usually very large set. The numbers of feasible solutions grow as some combinatorics factor such as $n!$ where, n characterizes size of the problem. It has often been the case that progress on TSP **(Laporte G., 2010)** has led to the progress on many Combinatorial Optimization Problems. In this way, TSP is an ideal stepping stone for study of Combinatorial Optimization Problems.

Although many optimal algorithms exist for solving TSP it has been realized that it is computationally infeasible to obtain optimal solution to the problem. For large-size problem **(Cormen T. H. et al., 2001)** it has been proved that it is almost impossible to generate an optimal solution within reasonable amount of time. Heuristics instead of optimal algorithms are thus extensively used to solve such problems **(Hansen M. P., 2000)**. Many heuristic algorithms give near optimal solutions to the problem which are used for practicability reasons specifically for large numbers of cities. Heuristic approaches **(Lin S. & Kernighan B. W., 1973)** for solving TSP are thus very popular which try to produce an optimal or close to optimal solution. The commonly used heuristic approaches are: (a) Greedy Algorithms; (b) 2-opt Algorithm; (c) 3-opt Algorithm; (d) Simulated Annealing; (e) Genetic Algorithms and (e) Artificial Neural Network (ANN). However, efficiencies vary from case to case and from size to size.

Generally the most common heuristic is ANN which are well suited for solving problems that are hard to catch in mathematical models. However, the usage and employment of ANN in such application domains is often dependent on tractability of processing costs. The problem domains for employment of ANN are increasing **(Haykin S., 2008)** and also problem themselves are getting larger and more complex **(Arbib M., 2003)**. This leads to larger networks consisting of huge numbers of nodes and interconnection links which results in exceeding costs for network specific operations such as evaluation and training. Especially the cost intensive training phase of ANN inherits a major drawback due to the situation that large numbers of patterns viz. input and target values are fed into the network iteratively. The effectiveness of ANN can be improved by deployment of Fuzzy Logic **(Jang J. S. R. et al., 1997)** which is a computational paradigm that generalizes classical two valued logic for reasoning under uncertainty. This is achieved by the notation of membership. Two things are accomplished by this viz. (i) ease of describing human knowledge involving vague concepts and (ii) enhanced ability to develop a cost-effective solution to real-world problem. Fuzzy Logic is thus is a multi-valued logic **(Zadeh L. A., 1994) w**hich is model less approach and clever disguise of Probability Theory. ANN and Fuzzy Logic are two complementary technologies. ANN can learn from data and feedback. However, understanding knowledge or pattern learned by ANN has been difficult. More specifically it is difficult to develop an insight about the meaning associated with each neuron and its weight. Hence, ANN are often viewed as black box approach. In contrast, Fuzzy Rule Based Models are easy to comprehend because it uses linguistic terms and structure of if then rules. Unlike ANN, Fuzzy Logic does not come with learning algorithm. Since ANN can learn, it is natural to merge two technologies. This merger creates a new term i.e. Neuro Fuzzy networks. A Neuro Fuzzy network thus describes a Fuzzy Rule Based Model using an ANN like structure.

In this chapter, Fuzzy Self Organizing Map (FSOM) **(Bezdek J. C., 1981; Kohonen T., 2001; Arbib M., 2003; Haykin S., 2008)** with one dimensional neighborhood is used to find optimal solution for symmetrical TSP. The solution generated by FSOM algorithm is improved by 2opt algorithm **(Aarts E. H. & Lenstra J. K., 2003)**. FSOM algorithm is compared with Lin-Kerninghan **(Lin S. & Kernighan B. W., 1973)** and Evolutionary algorithm **(Goldberg D. E., 1989; Deb K., 2001)** with enhanced edge recombination operator and self-adapting mutation rate. Experimental results indicate that FSOM 2opt hybrid algorithm generates appreciably better results compared to both Evolutionary and Lin-Kerninghan algorithms for TSP as number of cities increases. Some other optimization algorithms other than 2opt algorithm give better results. One of the best operators for TSP is enhanced edge recombination operator in comparison to permutation operators which are for other permutation problems. The chapter is structured as follows. In section 2 a brief survey of SOM is given. The next section illustrates FSOM. Section 4 describes the heuristic solution of TSP using FSOM and the corresponding mathematical characterization is given. In section 5 numerical results are presented along with an indepth run time analysis. Finally, in section 6 conclusions are given.

2. Self organizing map

SOM introduced by Teuvo Kohonen **(Kohonen T., 2001)** is an ANN that is trained using competitive, unsupervised learning **(Haykin S., 2008)** to produce low-dimensional discretized representation of input space of training samples called a map which preserves *topological* properties of input space. The development of SOM as neural model is motivated by distinct feature of human brain which is organized in many places in such a way that different sensory inputs are represented by *topologically ordered computational maps*. The output neurons of network compete among themselves to be activated or fired, with the result that only one output neuron or one neuron per group is on at one time. An output neuron that wins competition is called *winner takes all* or *winning* neuron **(Arbib M., 2003)**. SOM is thus useful for visualizing low-dimensional views of high-dimensional data which

is identical to multi-dimensional scaling. They generally operate in two modes viz. training and mapping. Training builds map using input examples, which is a competitive process also called vector quantization. Mapping automatically classifies a new input vector.
In SOM neurons are placed at nodes of *lattice* which is usually one or two dimensional. The neurons become selectively tuned to various input patterns or classes of input patterns in course of competitive learning process. The locations of neurons so tuned become ordered with respect to each other in such way that a meaningful coordinate system for different input features is created over lattice **(Kohonen T., 2001)**. As a neural model, SOM provides a bridge between two levels of adaptation viz. (a) adaptation rules formulated at microscopic level of single neuron and (b) formation of experimentally better and physically accessible patterns of feature selectivity at microscopic level of neural layers.
The competitive learning algorithm of SOM is either based on *winner takes all* or *winner takes mode* approach. However, *winner takes most* strategy is most common. When input vector is presented, distance to each neuron's synaptic weights are calculated. The neuron whose weights are most correlated to current input vector is winner. Correlation is equal to scalar product of input vector and considered synaptic weights. Only winning neuron modifies its synaptic weights to the point presented by input pattern. Synaptic weights of other neurons do not change. The learning process is described by **(Arbib M., 2003)**:

$$W_i \leftarrow W_i + \eta(x - W_i) \text{ where, } i \in \{0\ldots\ldots\ldots \textit{number of neurons}\},$$

W_i represents all synaptic weights of winning neuron, η is learning rate and x is current input vector. This simple algorithm can be extended giving more chance of winning to neurons that are rarely activated. The *winner takes most* has better convergence than *winner takes all* strategy. The difference is that many neurons in *winner takes most* strategy adapt their synaptic weights in single learning iteration only. In this case not only the winner but also its neighborhood adapts. The further neighboring neuron is from winner, smaller the modification which is applied to its weights. This adaptation process is described as **(Bishop C. M., 1995)**:

$$W_i \leftarrow W_i + \eta N(i,x)(x - W_i)$$

for all neurons i that belongs to winner's neighborhood. W_i stands for synaptic weights of neuron i and x is current input vector, η stands for learning rate and N (i, x) is function that defines neighborhood. Classical SOM is created when function N (i, x) is defined as **(Hertz J., Krogh A. & Palmer R. G., 1991)**:

$$N(i,x) = \{^{1\, for d(i,w) \le \lambda}_{0\, for others}$$

where, $d(i,w)$ is euclidean distance between winning and ith neuron and λ is neighborhood radius. To train SOM euclidean distance between input vector and all neural weights are calculated. Neuron that has shortest distance to input vector i.e. winner is chosen and its weights are slightly modified to direction represented by input vector. Then neighboring neurons are taken and their weights are modified in same direction. η and λ are multiplied with $\Delta\eta$ and $\Delta\lambda$ respectively during each learning iteration. These two last parameters are always less than one. Therefore, η and λ become smaller during learning process. At beginning SOM tries to organize itself globally and with following iterations it performs more and more local organization because learning rate and neighborhood gets smaller. Kohonen SOM is shown in Figure 1 **(Kohonen T., 2001)**. It maps input vectors of any dimension onto map with one, two or more dimensions. Input patterns which are similar to one another in input space are put close to one another in the map. The input vector is passed to every neuron. Kohonen SOM is made of vector or matrix of output neurons. If vector representation is chosen each neuron have two neighbors, one on left and other on right then it is called one-dimensional neighborhood as shown in Figure 2. If two-dimensional matrix representation is used neurons have 4 neighbors (viz. left, right, top and bottom). This is classical two dimensional neighborhood as shown in Figure 3. Instead of taking 4 nearest neurons 8 or more can be taken as shown in Figure 4. As many dimensions can be used as required viz. one, two, three or more dimensions. However, two dimensional neighborhood is most common.

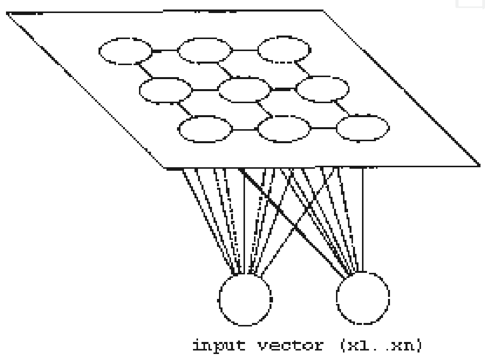

Fig. 1. Kohonen SOM with two dimensional neighborhood and input vector

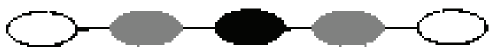

Fig. 2. One dimensional neighborhood of Kohonen SOM

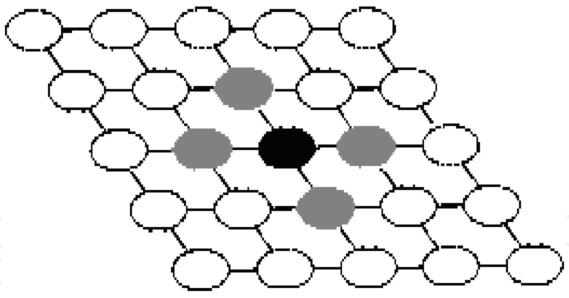

Fig. 3. Classical two dimensional neighborhoods

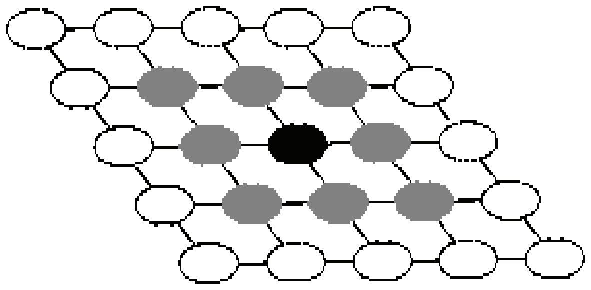

Fig. 4. Extended two dimensional neighborhood of Kohonen SOM

3. Fuzzy self organizing map

FSOM introduces the concept of membership function **(Bezdek J. C., 1981; Kohonen T., 2001; Arbib M., 2003; Haykin S., 2008)** in the theory of Fuzzy Sets to learning process. The membership R_{lj} of each pattern l to each neuron j is calculated and weight vector of each neuron is adjusted according to memberships of all patterns to the neuron. The learning algorithm is illustrated below. In FSOM some network parameters related to neighborhood in SOM are replaced with the membership function **(Bezdek J. C., 1981; Fritzke B., 1994; Kohonen T., 2001)**. Also the learning rate parameter is omitted. FSOM considers all input data at each iteration step. It is thus more effective at decreasing oscillations and avoiding dead units. FSOM used here is a combination of SOM and Fuzzy C Means (FCM) **(Bezdek J. C., 1981)** Clustering Algorithm.

3.1 Fuzzy C means clustering algorithm

FCM technique is a method of clustering which allows one piece of data to belong to two or more clusters. The method is developed by Dunn **(Dunn J. C.,1973)** and improved by Bezdek **(Bezdek J. C., 1981)** is frequently used in Pattern Recognition. It is based on minimization of the following objective function:

$$J_m = \sum_{i=1}^{N}\sum_{j=1}^{C} u_{ij}^m \,||x_i - c_j||^2 \,, 1 \le m < \infty$$

where, m is any real number greater than 1, u_{ij} is degree of membership of x_i in cluster j, x_i is i^{th} *i*th of d dimensional measured data, c_j is d dimension center of cluster and $||*||$ is any norm expressing similarity between any measured data and the center. FCM thus processes N vectors in d space as data input and uses them in conjunction with first order necessary conditions for minimizing FCM objective functional to obtain estimates for two sets of unknowns. Fuzzy partitioning is carried out through an iterative optimization of objective function with update of membership u_{ij} and cluster centers c_j by:

$$u_{ij} = \frac{1}{\sum_{k=1}^{C}\left(\frac{||x_i - c_j||}{||x_i - c_k||}\right)^{\frac{2}{m-1}}} \,, c_j = \frac{\sum_{i=1}^{N} u_{ij}^m \cdot x_i}{\sum_{i=1}^{N} u_{ij}^m}$$

This iteration will stop when $\max_{ij}\{|u_{ij}^{(k+1)} - u_{ij}^{(k)}|\} < \varepsilon$ where ε is termination criterion betweem 0 and 1 in k iteration steps. The procedure converges to a local minimum or a saddle point of J_m. The algorithm is composed of following steps:

a. Initialize the matrix $U = [u_{ij}]$ to $U^{(0)}$.
b. At k step calculate centre vectors $C^{(k)} = [c_j]$ with $U^{(k)}$,

$$c_j = \frac{\sum_{i=1}^{N} u_{ij}^m \cdot x_i}{\sum_{i=1}^{N} u_{ij}^m}$$

c. Update the matrices $U^{(k)}$ and $U^{(k+1)}$,

$$u_{ij} = \frac{1}{\sum_{k=1}^{C}\left(\frac{||x_i - c_j||}{||x_i - c_k||}\right)^{\frac{2}{m-1}}}$$

d. If $||U^{(k+1)} - U^{(k)}|| < \varepsilon$ then stop, otherwise goto step (b).

The data are bound to each cluster by means of membership function which represents the fuzzy behaviour of this algorithm. This is achieved by the matrix U whose factors are numbers between 0 and 1, and represent the degree of membership between data and centers of clusters. In FCM approach as the same given datum does not belong exclusively to a well defined cluster, it is placed somewhere in the middle such that the membership function follows a smoother line to indicate that every datum may belong to several clusters with different values of membership coefficient.

FCM is generalized in many ways **(Bezdek J. C., 1981)** such as, the memberships includes possibilities; prototypes have evolved from points to linear varieties to hyper-quadrics to shells to regression functions; the distance includes Minkowski (non-inner product induced) and hybrid distances. There are many relatives of FCM for dual problem called relational FCM which is useful when data are not object vectors but relational values viz. similarities between pairs of objects. There are also many acceleration techniques for FCM as well as very large versions of FCM that utilize both progressive sampling and distributed clustering. Many techniques use FCM clustering to build Fuzzy rule bases for Fuzzy Systems design. Numerous applications of FCM exist **(Arbib M., 2003; Haykin S., 2008)** virtually in every major application area of clustering.

3.2 FSOM learning algorithm

In ANN structure, each output neuron directly corresponds to a city in network of cities **(Haykin S., 2008)**. The number of output neurons used to describe the cities is generally arbitrary. However, if number of neurons is equal to number of cities the problem gets simplified. The more the number of neurons, the greater is accuracy of model. The number of output neurons needed for good accuracy depends on complexity of the problem. The more complex the problem, more output neurons are required. The number of output neurons is manually selected. The weight W connects input vector components and output neurons. The weight vectors are of same dimensions as sample vectors. The weight components are initialized randomly and adjusted gradually using self organizing learning algorithm and ultimately a mapping is done from input to output. Let M denote number of input patterns, N number of input vector components and K number of output neurons. The learning algorithm consists of the following steps **(Bezdek J. C., 1981)**:

a. Randomize weights for all neurons.

b. Input all patterns $X_l = \{X_{l1}, \dots\dots\dots, X_{lN}\}, l = 1, \dots\dots\dots, M$ .Take one random input pattern and calculate euclidean distances from each pattern X_l to all output neurons.

$$d_{lj}(t) = \sqrt{\sum_{i=1}^{N}(X_{li} - W_{ij}(t))^2}\,; l = 1, \dots\dots\dots, M, j = 1, \dots\dots\dots, K.$$

c. Compute memberships of each pattern to all neurons **(Tao T., Gan J. R. & Yao L. S., 1992)**.

$$R_{lj}(t) = \frac{\{d_{lj}(t)\}^{-2}}{\sum_{m=1}^{K}\{d_{lm}(t)\}^{-2}}; l = 1,\dots\dots, M, j = 1,\dots\dots, K$$

d. Find winning neuron and neighbors of winner.
e. Adjust synaptic weights of each neuron according to computed memberships.

$$W_{ij}(t+1) = W_{ij}(t) + \frac{\sum_{l=1}^{M} R_{lj}(t)(X_{li} - W_{ij}(t))}{\sum_{l=1}^{M} R_{lj}(t)}$$

f. Reduce values of parameters η and λ.
g. Determine stability condition of network.

$$\begin{aligned}&\max\{|W_{ij}(t+1) - W_{ij}(t)|\} < \varepsilon\\&1 \le i \le N\\&1 \le j \le K\end{aligned}$$

If the stability condition is satisfied or predefined number of iterations is achieved, then learning process terminates, otherwise go to Step (b) for another loop of learning. From above learning procedure, it is observed that FSOM eases the difficulty of selecting network parameters. In above learning procedure, weights are adjusted only once in each learning loop and features of all input samples are taken into consideration once weights are adjusted **(Arbib M., 2003)**. Thus, learning speed and estimation accuracy are greatly improved.

4. Heuristic solution for traveling salesman problem by fuzzy self organizing map

Most interesting results of self-organization (Dittenbach M. et al., 2000; Kohonen T., 2001; Junfei Q. et al., 2007) are achieved in networks that have two dimensional input vectors and two dimensional neighborhoods. In this case input to network consists of two values viz. x and y which represent a point in two dimensional space. This kind of network can map two dimensional objects in such a way that a mesh which covers this object is created. This process is illustrated in Figure 5. Each example consists of six squares. First one shows the object that should be learned. The second square illustrates network just after randomization of all neural weights. Following squares describe the learning process. It is to be noted that each neuron or a circle represents a point whose coordinates are equal to neuron's weights. These figures illustrate that Kohonen ANN is powerful self-organizing and clustering tool. However, it is also possible to create network with one dimensional neighborhood and two dimensional inputs (Arbib M., 2003). Learning process of this is shown in Figure 6. It is observed that this network tries to organize its neurons in such a

way that a relatively short route between all neurons emerges. These experiments are stimulus to build system based on one-dimensional FSOM that would solve TSP problems **(Xu W. & Tsai W. T., 1991; Burke L. I., 1994; Sentiono R., 2001)**.
To solve TSP problem a one dimensional network is created. If the weights of neuron is equal to some city's coordinates the neuron represents that city. In other words a neuron and a city are assigned to each other and there is a one-to-one mapping **(Haykin S., 2008)** between set of cities and set of neurons. All neurons are organized in a vector. This vector represents sequence of cities that must be visited. However, some modifications need to be done before FSOM is able to fully solve this problem. This is because the real valued neural weights are never exactly equal to coordinates of cities. To solve the problem an algorithm that modifies FSOM solution to a valid one is created. Positions of cities and neurons may not equal. However, adequate neural weights and cities coordinates are very close to each other. An algorithm that modifies neural weights so that they equal to cities coordinates is applied. These weights are modified in such a way to restore one-to-one mapping assumed at beginning. If neuron A is assigned to a city B it means that weights of neuron A are equal to coordinates of city B. After applying this algorithm a good and fast solution is obtained. However, it is not locally optimal **(Applegate D. L. et al.,2006; Laporte G., 2010)**. Thus it needs to be optimized using well known 2opt algorithm **(Aarts E. H. & Lenstra J. K., 2003)**. In this case 2opt works fast even for large amount of cities because current solution is already good. Usually 2opt does not change the solution a lot as shown in the Figure 7. The 2opt algorithm is based on one simple rule which selects a part of the tour, reverses it and inserts back in the cycle. If new tour is shorter than original cycle, then it is replaced. The algorithm stops when no improvement can be done. For example if there is a cycle (A, B, C, D, E, F) and a path (B, C, D) is reversed, then new cycle is: (A, D, C, B, E, F). After 2opt optimization the solution is locally optimal as shown in Figure 8. FSOM optimal training parameters are chosen adequately to number of cities to achieve best results **(Arbib M., 2003; Haykin S., 2008)**. It is found empirically that good training parameters are as follows:

a. For 200 cities: $\eta = 0.5$, $\Delta\eta = 0.9667$, $\Delta\lambda = 0.966$
b. For 700 cities: $\eta = 0.6$, $\Delta\eta = 0.9665$, $\Delta\lambda = 0.9664$
c. For 1200 cities: $\eta = 0.8$, $\Delta\eta = 0.9662$, $\Delta\lambda = 0.9666$

In every case the number of iterations is set to 25000.

5. Numerical simulation

In the quest of finding solution to TSP problem **(Applegate D. L. et al., 2006)** using FSOM following two types of tests are done:

a. Using city sets taken from TSPLIB **(Reinelt G., 1991)** in which there are already some optimal solutions present
b. Using randomly chosen cities

TSPLIB city sets are hard to solve because in many cases the cities are not chosen randomly as shown in Figures 9 and 10. Generally larger city sets consist of small patterns. City set shown in Figure 10 consists of two different patterns and each of them is used nine times. Thus, optimal tour is identical in each one of these smaller patterns shown in Figure 10 top. FSOM tries to figure out a unique tour in each of the smaller pattern shown in Figure 10 bottom. The testing process using randomly chosen cities is more objective. It is based on Held-Karp Traveling Salesman bound **(Johnson D. S. et al., 2000)**. An empirical relation for expected tour length is:

$$L = k\sqrt{nR}$$

where L is expected tour length, n is a number of cities, R is an area of square box on which cities are placed and k is an empirical constant. For $n \geq 100$ value of k is:

$$k = 0.70805 + \frac{0.52229}{\sqrt{n}} + \frac{1.31572}{n} - \frac{3.07474}{n\sqrt{n}}$$

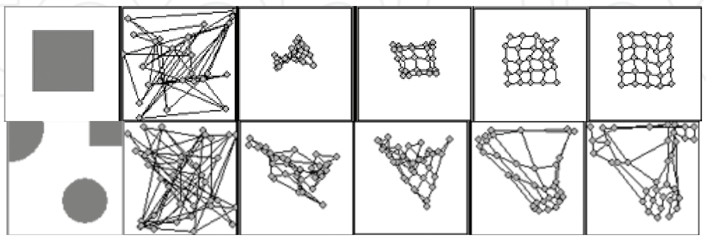
Fig. 5. Self-organization of network with two dimensional neighborhoods

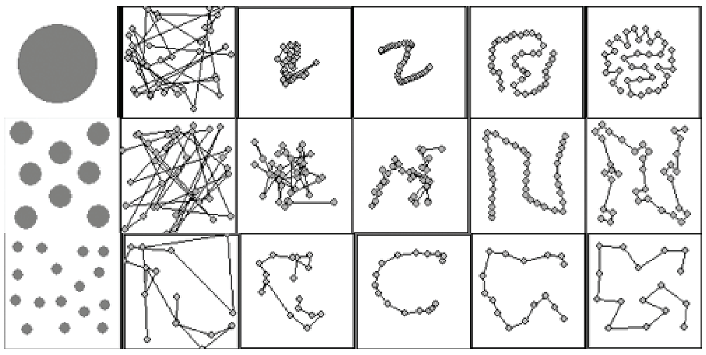
Fig. 6. Self-organization of network with one dimensional neighborhood

The three random city sets viz. 200, 700, 1200 cities are used in this experiment; square box edge length is restricted to 500. All statistics for FSOM are generated after 75 runs on each city set. When number of iterations is taken as 100, the average results did not show any considerable difference. Better results are obtained on increasing the number of iterations. FSOM generates a tour in relatively short time, such as 225 cities set is solved in 254 ms and 1000 cities set in less than 2 seconds. The average tour lengths for city sets up to 2000 cities are comparatively better than optimum. FSOM thus generates solutions that are noticeably good from optimal tour. FSOM is compared with the Evolutionary Algorithm **(Goldberg D. E., 1989; Deb K., 2001)**. Evolutionary Algorithm uses enhanced edge recombination operator **(Rahendi N. T A. & Atoum J., 2009)** and steady state survivor selection where always the worst solution is replaced with tournament parent selection where tournament size depends on number of cities and population size. Scramble mutation is used here. The optimal mutation rate depends on number of cities and state of evolution. Therefore, self-adapting mutation rate is used. Every genotype has its own mutation rate **(Michalewicz Z., 1996)** which is modified in a similar way as in evolution strategies. This strategy adapts mutation rate to number of cities and evolution state automatically, so it is not needed to check manually which parameters are optimal for each city set. Evolution stops when population converges **(Goldberg D. E., 1989)**. Population size is set to 1000 **(Michalewicz Z., 1996)**. With smaller populations, Evolutionary Algorithm did not work that well. When Evolutionary Algorithm stopped, its best solution is optimized by 2opt algorithm. The results for FSOM, Evolutionary Algorithm and 2opt Algorithm are shown in the Table 1. For Evolutionary Algorithm there are 20 runs of algorithm for sets EIL51, EIL101 and RAND100. For other sets Evolutionary Algorithm is run twice. The optimum solutions for instances taken from TSPLIB are already present there and optimum solutions for random instances are calculated from empirical relation described above.

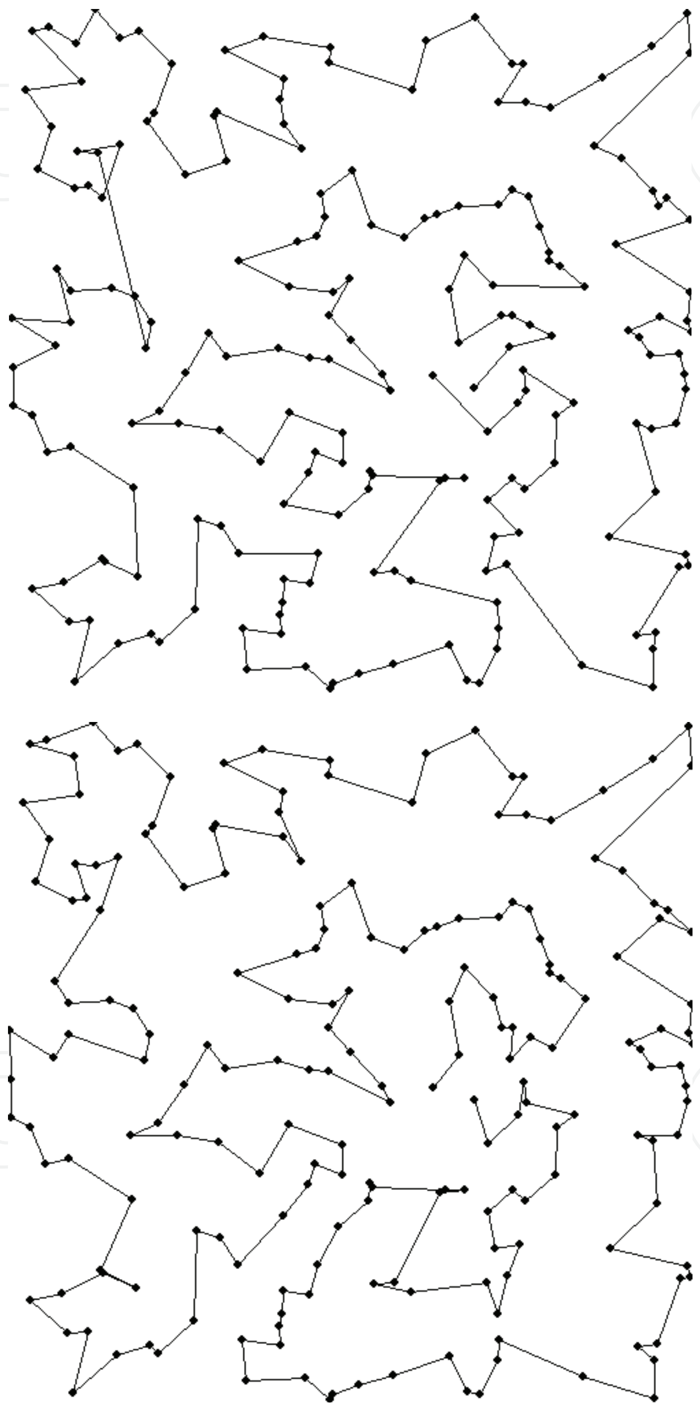

Fig. 7. Self Organizing Map solution without 2opt optimization (top). There are two local loops on left. First and last neuron can be seen in the middle. They are not connected in figure but distance between them is also computed. The same solution improved by 2opt (bottom). Loops on left have been erased. Additional changes can be observed

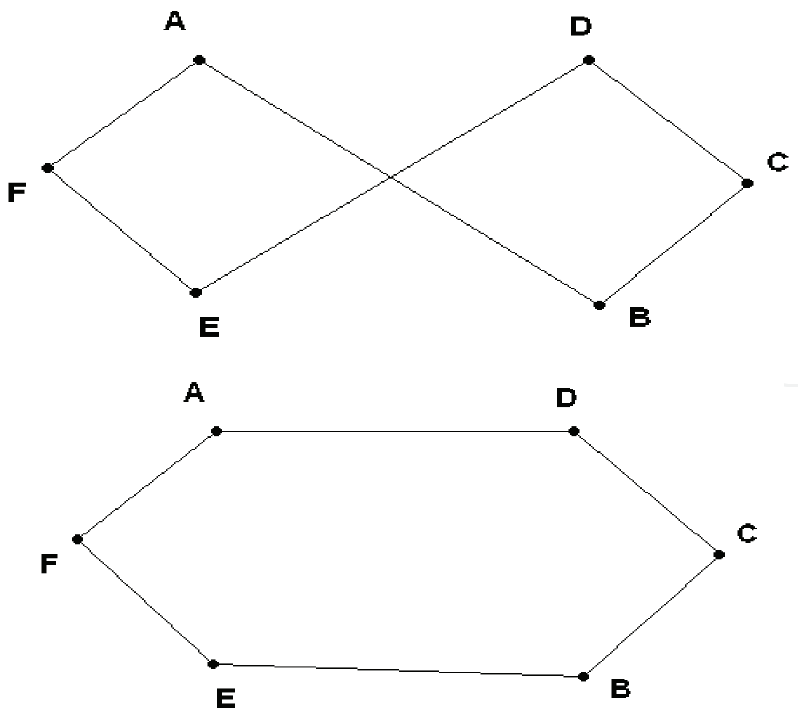

Fig. 8. 2opt optimization. If there is a cycle (A, B, C, D, E, F) and path (B, C, D) is reversed, then new cycle is (A, D, C, B, E, F).

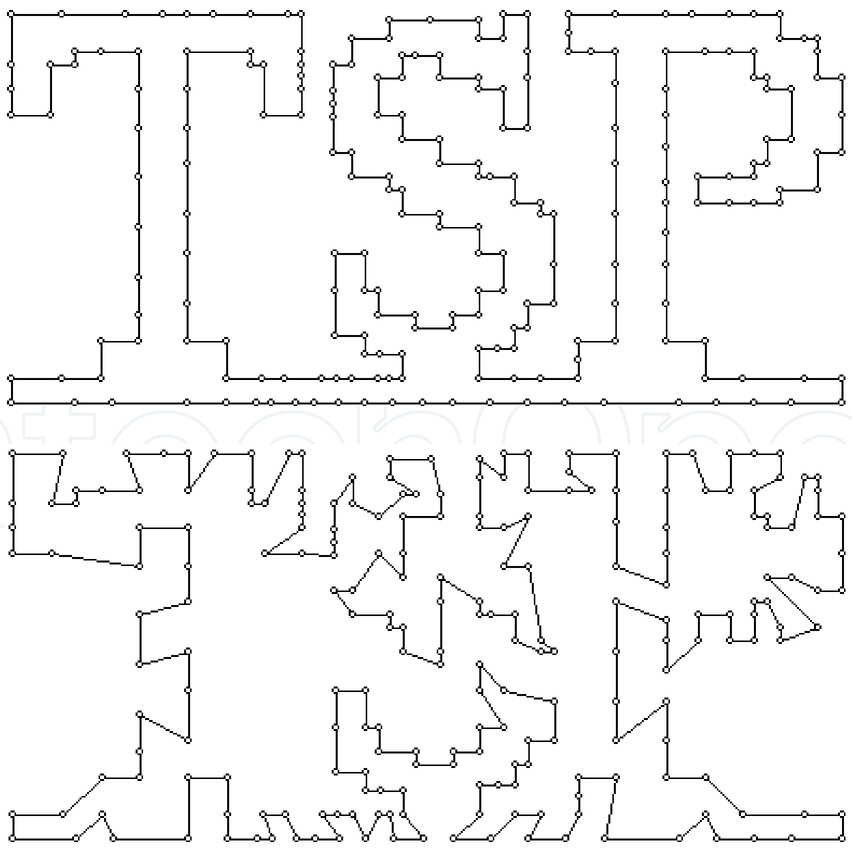

Fig. 9. Optimal tour length for 225 city set taken from TSPLIB (top) is 3916. Tour length generated by FSOM 2opt hybrid (bottom) is 3899.

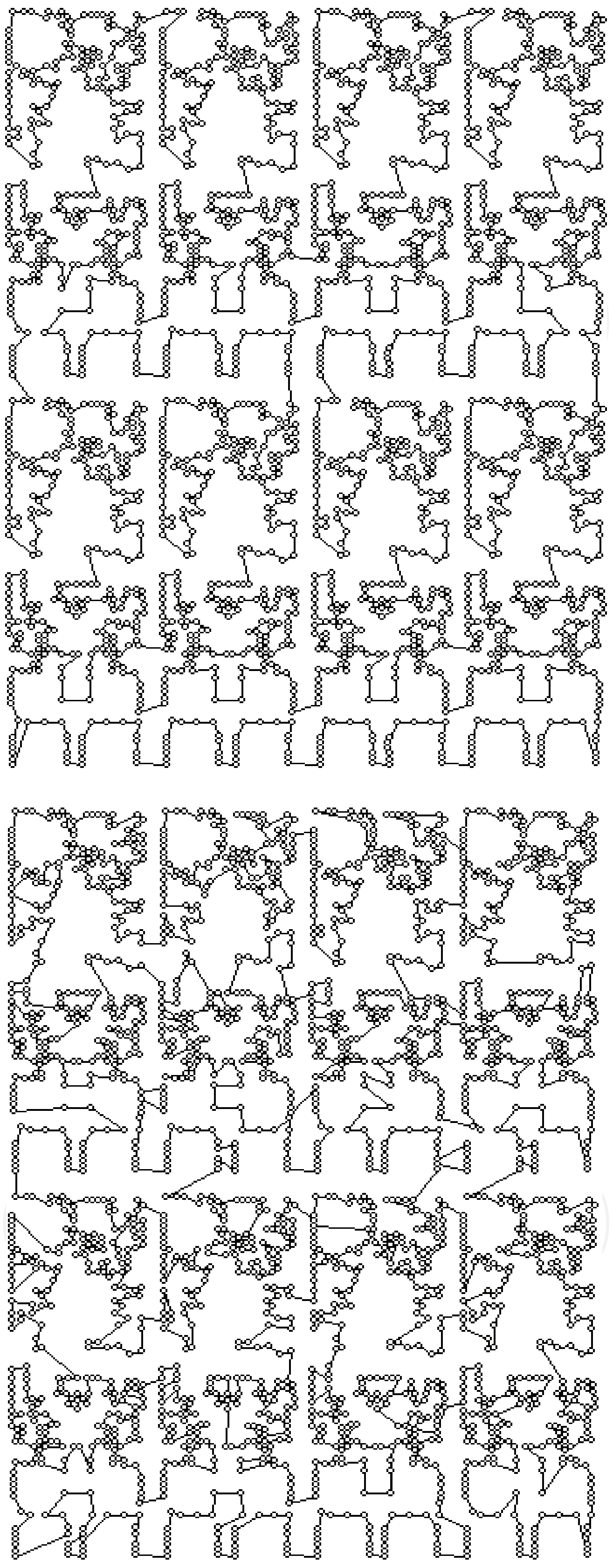

Fig. 10. Optimal tour length for 2392 city set taken from TSPLIB (top) is 378037. Tour length generated by FSOM 2opt hybrid (bottom) is 377946.

The experiments show that Evolutionary Algorithm **(Goldberg D. E., 1989; Deb K., 2001)** finds better solutions for instances with up to 100 cities. Both average and best results are better than FSOM. For city sets with 50 or less, Evolutionary Algorithm finds optimum in every execution. The results for 225 cities are nearly comparable for both algorithms. However, for larger amount of cities viz. 442 and more FSOM yields better solutions. With more number of cities search space increases significantly and Evolutionary Algorithm needs bigger population size. For TSP225 with population size of 1000 Evolutionary Algorithm result is 4044, but when population size is expanded to 3000 a tour with length 3949 is found which is comparable to FSOM solution. This underlines the fact that when Evolutionary Algorithm is used one can always expand population size **(Michalewicz Z., 1996)**, so the algorithm has greater chance of achieving good results. However, algorithm is much slower then.
It is interesting to compare FSOM algorithm to other Non-Evolutionary approaches **(Chaudhuri A., 2007)**. One of the best TSP algorithms which is appreciably fast is Lin-Kerninghan Algorithm **(Lin S. & Kernighan B. W., 1973)**. The algorithm is run 20 times on each city set. Average results and times are shown in Table 2 which indicate that Lin-Kerninghan is comparable to that of FSOM. There is no considerable difference in time for small 51-city instance which is 0.012 seconds for Lin-Kerninghan and 0.024 seconds for FSOM. On other hand, for 2392-city instance Lin-Kerninghan needed just 0.719 seconds and FSOM required almost 7 seconds. This is because FSOM is optimized by 2opt which is the slowest part of this algorithm. When average results are compared it can be easily seen that Lin-Kerninghan is superior in all cases. The higher is number of cities, bigger the difference between both algorithms. FSOM is also used to generate initial population for Evolutionary Algorithm. Such initialization takes only a fraction of time needed for Evolutionary Algorithm to finish because FSOM is fast algorithm. In this case, Evolutionary Algorithm tends to converge much faster and finally it did not improve the best solution generated by FSOM alone. It seems that all initial solutions are very similar to each other, thus population diversity is low and so the Evolutionary Algorithm lost all exploration abilities.

6. Conclusion

The experimental results indicate that FSOM 2opt hybrid algorithm generates appreciably better results compared to both Evolutionary and Lin Kerninghan Algorithm for TSP as number of cities increases. There are some parameters such as $\eta, \Delta\eta, \Delta\lambda$ that can be optimized. Experiments with other Self Organizing networks should be performed and gaussian neighborhood and conscience mechanism can be applied which can improve TSP solutions generated by ANN **(Christof T. & Reinelt G., 1995)**. Some other optimization algorithms may be used other than 2opt algorithm which gives better results. There are many algorithms that solve permutation problems. Evolutionary Algorithms have many different operators that work with permutations. Enhanced edge recombination is one of the best operators for TSP **(Goldberg D. E., 1989; Deb K., 2001)**. However, it is proved that other permutation operators which are worse for TSP than enhanced edge recombination are actually better for other permutation problems like warehouse or shipping scheduling applications **(Korte B. H. & Vygen J., 2008)**. Therefore, it might be possible that FSOM 2opt hybrid might work better for other permutation problems than for TSP.

Instances	Optimum	FSOM			Evolutionary Algorithm			2opt Algorithm		
		Average Result	Best Result	Best Time	Average Result	Best Result	Best Time	Average Result	Best Result	Best Time
EIL51	426	435	428	0.024	428.2	426	10	537	524	1.44
EIL101	629	654	640	0.069	653.3	639	75	869	789	2.96
TSP225	3916	3909	3899	0.254		4044	871		4679	6.7
PCB442	50778	50635	50537	0.407		55657	10395		56686	12.37
PR1002	259045	259024	259010	1.999		286908	25639		292069	29
PR2392	378037	377969	377946	7.967						
RAND200	3851.81	3844	3769	0.131	3931.4	3822	69.6	4344	4037	5.9
RAND700	8203.73	8199	8069	0.824		9261	11145		14116	17.8
RAND1200	11475.66	11469	11437	2.311		12858	56456		24199	37

Table 1. Comparison of FSOM, Evolutionary Algorithm and 2opt Algorithm

Instances	Optimum	Lin Kerninghan	
		Average Result	Average Time
EIL51	426	427.4	0.012
EIL101	629	640	0.039
PCB442	50778	51776.5	0.137
PR2392	378037	389413	0.719

Table 2. Results for Lin-Kerninghan Algorithm

7. References

Applegate, D. L., Bixby, R. E., Chvåtal, V. & Cook, W. J. (1995). *Finding cuts in TSP* (*A preliminary report*),Technical Report DIMACS 95-05, Rutgers University, Piscataway, New Jersey.

Applegate, D. L., Bixby, R. E., Chvåtal, V. & Cook, W. J. (2006). *The Traveling Salesman Problem: AComputational Study*, Princeton University Press, Princeton, New Jersey.

Arbib, M. (2003). *The Handbook of Brain Theory and Neural Networks*, Second Edition, The MIT Press, Cambridge, Massachusettes.

Aarts, E. H. & Lenstra, J. K. (2003). *Local Search in Combinatorial Optimization*, Princeton University Press, Princeton, New Jersey.

Bezdek, J. C. (1981). *Pattern Recognition with Fuzzy Objective Function Algorithms*, Plenum Press, New York.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*, Clarendon Press, Oxford.

Burke, L. I. (1994). Neural Methods for the Traveling Salesman Problem: Insights from Operations Research. *Neural Networks*, Vol. 7, No. 4, pp. 681–690.

Chaudhuri, A. (2007). A Dynamic Algorithm for looking Traveling Salesman Problem as a Fuzzy Integer Linear Programming Problem in an optimal way. *Proceedings of International Conference on Information Technology*, Haldia Institute Technology, India, Vol. 1, pp. 246–251.

Christof, T. & Reinelt, G. (1995). Parallel Programming and Applications, In: *Parallel Cutting Plane Generation for TSP*, P. Fritzson, L. Finmo, (Editors), pp. 163–169, IOS Press.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2001). *Introduction to Algorithms*, Second Edition, MIT Press Cambridge, Massachusetts.

Deb, K. (2001). *Multi-Objective Optimization using Evolutionary Algorithms*, John Wiley and Sons, New York.

Dittenbach, M., Merkel, D. & Rauber, A. (2000). The growing Hierarchical Self Organizing Map. *Proceedings of the International Joint Conference on Neural Networks*, vol. VI, pp.15–19.

Dunn, J. C. (1973). A Fuzzy relative of ISODATA Process and its use in detecting compact well separated Clusters. Journal of Cybernetics, Vol. 3, pp. 32-57.

Fritzke, B. (1994). Growing Cell Structure — A Self Organizing Neural Network for Unsupervised and Supervised Learning. *Neural Networks*, Vol. 7, No. 9, pp. 1441–1460.

Garey, M. & Johnson, D. (1990). *Computers and Intractability*: *A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, San Francisco.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*, Reading, Mass, Addison Wesley.

Hansen, M. P. (2000). Use of Substitute Scalarizing Functions to Guide a Local Search Based Heuristics: The Case of MOTSP. *Journal of Heuristics*, Vol. 6, No. 3, pp. 419–431.

Haykin, S. (2008). *Neural Networks and Learning Machines,* Third Edition, Prentice Hall.

Hertz, J., Krogh, A. & Palmer, R. G. (1991). *Introduction to the Theory of Neural Computation*, Addison Wesley, Massachuttes.

Jang, J. S. R., Sun, C. T., Mizutani, E. (1997**).** *Neuro-Fuzzy and Soft Computing*. *A Computational Approach to Learning and Machine Intelligence*, Prentice Hall.

Johnson, D. S., McGeoch, L. A. & Rothberg, E. E. (2000). Asymptotic Experimental Analysis for the Held-Karp Traveling Salesman Bound. *Proceedings of ACM-SIAM symposium on Discrete Algorithms*.

Junfei Q., Honggui, H. & Yanmei, J. (2007). An Adaptive Growing Self Organizing Fuzzy Neural Network. *Proceedings of 5th International Conference on Wavelet Analysis and Pattern Recognition 2007*, pp. 711–715.

Kohonen, T. (2001). *Self–Organizing Maps*, Third Edition, Springer Verlag, Berlin.

Korte, B. H. & Vygen, J. (2008). *Combinatorial Optimization*: *Theory and Algorithms*, Algorithms and Combinatorics, Fourth Edition, Springer Verlag.

Laporte, G. (2010). A Concise Guide to Traveling Salesman Problem. *Journal of the Operational Research Society*, Vol. 61, No. 1, pp. 35–40.

Lin S., Kernighan B. W. (1973). An effective Heuristic for Traveling Salesman Problem. *Operations Research*, Vol. 21, pp. 498–516.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*, Springer Verlag.

Rahendi, N. T A. & Atoum, J. (2009). Solving the Traveling Salesman Problem using New Operators in Genetic Algorithms. *American Journal of Applied Sciences*, Vol. 6, No. 8, pp. 1586–1590.

Reinelt G. (1991). TSPLIB – A Traveling Salesman Problem Library. *ORSA Journal on Computing*, Vol. 3, pp. 376–384.

Sentiono R. (2001). Feed-forward Neural Network Construction using Cross Validation. *Neural Computation*, Vol. 13, No. 12, pp. 2865–2877.

Tao T., Gan J. R. & Yao L. S. (1992). Application of Fuzzy Neural Computing in Circuit Partitioning. *Chinese Journal of Compuing*, Vol. 15, No. 9, pp. 640–647.

Xu, W. & Tsai, W. T. (1991). Effective Neural Algorithm for the Traveling Salesman Problem. *Neural Networks*, Vol. 4, No. 2, pp. 193–205.

Zadeh, L. A. (1994). Fuzzy Logic, Neural Networks and Soft Computing. *Communications of the ACM*, Vol. 37, No. 3, pp. 77–84.