Kohonen Self-Organizing Map for the Traveling Salesperson Problem

Lucas Brocki
Polish–Japanese Institute of Information Technology,
ul. Koszykowa 86, 02-008
Warsaw, Poland
lukasz.brocki@pjwstk.edu.pl

Abstract - This work shows how a modified Kohonen Self-Organizing Map with one dimensional neighborhood is used to approach the symmetrical Traveling Salesperson Problem. Solution generated by the Kohonen network is improved by the 2opt algorithm. The paper describes briefly self-organization in neural networks, 2opt algorithm and modifications applied to Self-Organizing Map. Finally, the algorithm is compared with Lin-Kerninghan algorithm and Evolutionary Algorithm with Enhanced Edge Recombination operator and self-adapting mutation rate.

1 Introduction

The idea of the Traveling Salesperson Problem (TSP) is to visit a set of n cities once and once only and return to the starting city. It might look simple, but it was proved that the TSP is NP-hard problem [10]. The search space is huge: $n!$. Thus, it is not possible to check all solutions for city sets with many thousands of cities and some engineering problems like VLSI designing need 1.2 million cities [11]. A fast and effective heuristic method is needed. The author intended to build a neural based algorithm and compare it with two well know and very effective heuristic methods: Lin-Kerninghan and Evolutionary Algorithm with Enhanced Edge Recombination operator.

2 Kohonen Self-Organizing Map basics

In 1975 Teuvo Kohonen introduced new type of neural network that uses competitive, unsupervised learning[1]. This approach is based on WTA (Winner Takes All) and WTM (Winner Takes Most) algorithms. Therefore, these algorithms will be explained here briefly. The most basic competitive learning algorithm is WTA. When input vector (a pattern) is presented, a distance to each neuron's synaptic weights is calculated. The neuron whose weights are most correlated to current input vector is the winner. Correlation is equal to scalar product of input vector and considered synaptic weights. Only the winning neuron modifies it's synaptic weights to the point presented by input pattern. Synaptic weights of other neurons do not change. The learning process can be described by the following equation:

$$W_i \leftarrow W_i + \eta (x - W_i)$$

where $i \in [0..numer\ of\ neurons]$, W_i represents all synaptic weights of the winning neuron, η is learning rate and x stands for current input vector. This simple algorithm can be extended. The most common extension is giving more chance of winning to neurons that are rarely activated. However, WTM strategy has better convergence than WTA. The difference between those two algorithms is that many neurons in WTM strategy adapt their synaptic weights in one learning iteration. In this case not only the winner, but also it's neighborhood adapts. The further the neighboring neuron is from the winner, the smaller the modification which is applied to its weights. This adaptation process can be described as:

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

for all neurons i that belong 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 a function that defines neighborhood. Classical Self Organizing Map (SOM) can be created when function N(i, x) is defined as:

$$N(i, x) = \begin{cases} 1 & for \quad d(i, w) \leq \lambda \\ 0 & for \quad others \end{cases}$$

where *d(i, w)* is Euclidean distance between winning and i-th neuron. λ is neighborhood radius. To train Kohonen SOM Euclidean distance between input vector and all neural weights has to be calculated. Neuron that has the shortest distance to input vector (the winner) is chosen and it's weights are slightly modified to direction represented by input vector. Then neighboring neurons are taken and their weights are modified in the 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 the beginning SOM tries to organize itself globally and with following iterations it performs more and more local organization, because learning rate and neighborhood get smaller.

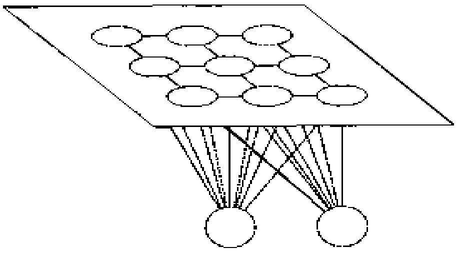

Figure 1: Kohonen SOM with two dimensional neighborhood and input vector.

Kohonen SOM is shown in Figure 1. It maps input vectors of any dimension onto map with one, two or more dimensions. Input patterns, which are similar to one another in the input space are put close to one another in the map. The input vector is passed to every neuron. A Kohonen SOM is made of a vector or matrix of output neurons. If vector representation is chosen each neuron has two neighbors (on the left and on the right). It is called one-dimensional neighborhood:

Figure 2: One dimensional neighborhood of Kohonen SOM

If two-dimensional matrix representation is used, neurons have 4 neighbors (left, right, top and bottom). This is classical two dimensional neighborhood (figure 3). Neighborhood can be expanded. Instead of taking four nearest neurons, 8 or more can be taken (figure 4). As many dimensions can be used as needed: 1D, 2D, 3D or more. However, 2D neighborhood is most common.

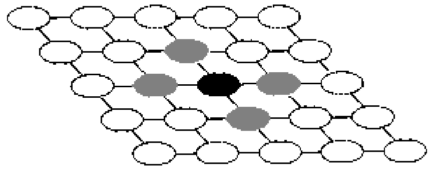

Figure 3: Classical two dimensional neighborhood

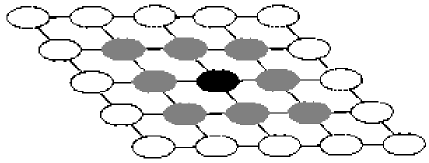

Figure 4: Extended two dimensional neighborhood of Kohonen SOM

Basic SOM algorithm can be described as follows:

procedure train_SOM
begin
randomize weights for all neurons
for *(i = 1 to iteration_number)* ***do***
begin
take one random input pattern
find the winning neuron
find neighbors of the winner
modify synaptic weights of these neurons
reduce the η *and* λ
end
end

3 Experiments on self-organization

Most interesting results of self-organization can be achieved in networks that have two dimensional input vector and two-dimensional neighborhood. In this case input to network consists of two values: 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 object that should be learned. Second square illustrates network just after randomization

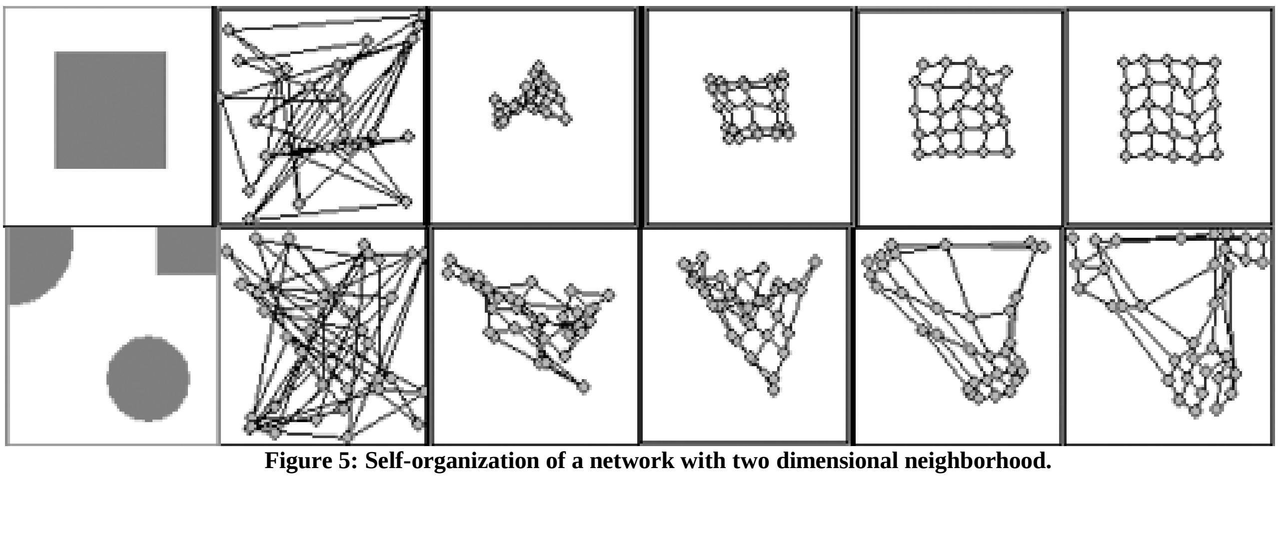

Figure 5: Self-organization of a network with two dimensional neighborhood.

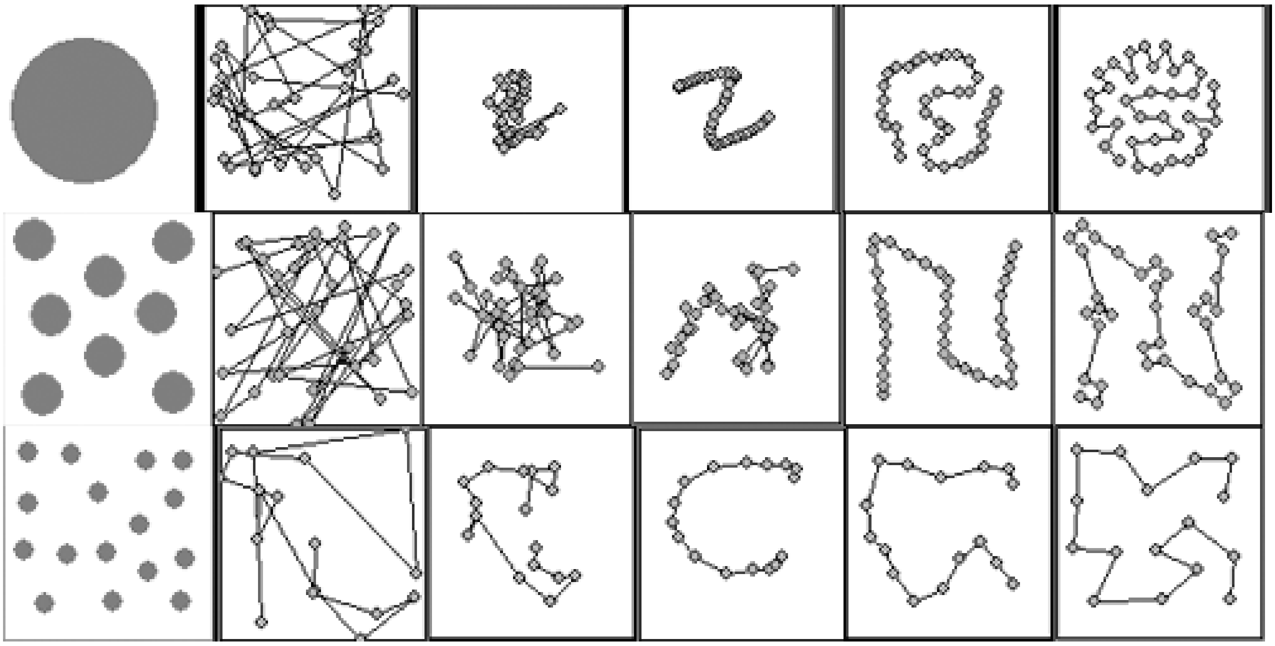

Figure 6: Self-organization of a network with one dimensional neighborhood.

of all neural weights. Following squares describe learning process. Please note that each neuron (a circle) represents a point whose coordinates are equal to neuron's weights. These figures illustrate that Kohonen neural network is a powerful self-organizing and clustering tool. However, it is also possible to create a network with one dimensional neighborhood and two dimensional input. Learning process of this is shown in Figure 6.

It can be observed that this network tries to organize it's neurons in such a way, that a relatively short route between all neurons emerges. These experiments were a stimulus to build a system based on Kohonen one-dimensional SOM that would solve TSP problems.

4 SOM based TSP solver

To solve TSP problem a one dimensional network must be created. Number of neurons must be equal to the number of cities. If the weights of a neuron are equal to some city's coordinates this neuron represents that city. In other words a neuron and a city are assigned to each other and there is a 1-to-1 mapping between the set of cities and the 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 the SOM is able to fully solve this problem. This is because the real-valued neural weights may never equal exactly the coordinates of the cities. To solve the problem an algorithm that would modify Kohonen solution, to one that is valid, has been created. Positions of cities and positions of neurons may not equal. However, adequate neural weights and cities' coordinates are very close to each other. An algorithm that modifies neural weights so they equal to cities' coordinates has been applied. These weights need to be modified in such a way to restore the 1-to-1 mapping assumed on the beginning. If neuron A is assigned to a city B it means that weights of neuron A are equal to coordinates of city B.

Scheme of the repair algorithm:

procedure repair
begin
    Iterate through all neurons
        begin
            nearest_city = find the nearest city to current neuron
            if *(nearest_city is not assigned to any neuron)*
                assign nearest_city and current neuron
            else
                delete this neuron
        end
    Iterate through all cities
    begin
        if *(current city is not assigned to any neuron)*
            begin
                create a new_neuron and assign it to current city
                nearest_neuron = find the nearest neuron to current city
                insert new_neuron before or after nearest_neuron, depending on which tour is locally shorter
            end
    end
end

After applying this algorithm a good and fast solution is obtained, however it is not locally optimal. Therefore, it needs to be optimized using well know 2opt algorithm. 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 (Figure 8). The 2opt algorithm is based on one simple rule. It selects a part of the tour, reverses it, and inserts back in the cycle. If the new tour is shorter than the 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 the new cycle is: (A, D, C, B, E, F). After 2opt optimization the solution is locally optimal.

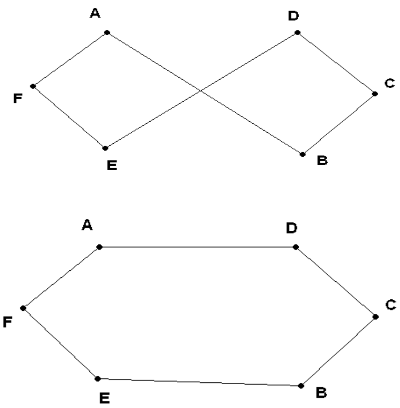

Figure 7: 2opt optimization. If there is a cycle (A, B, C, D, E, F) and a path (B, C, D) is reversed, then the new cycle is: (A, D, C, B, E, F)

Optimal Kohonen SOM training parameters should be chosen adequately to number of cities to achieve best results. It was found empirically that good training parameters are:

- for 100 cities:
  $\eta = 0.6$
  $\Delta\eta = 0.9997$
  $\Delta\lambda = 0.999$

- for 500 cities:
  $\eta = 0.7$
  $\Delta\eta = 0.999985$
  $\Delta\lambda = 0.9994$

- for 1000 cities:
  $\eta = 0.9$
  $\Delta\eta = 0.99992$
  $\Delta\lambda = 0.9996$

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

5 The Experiment

Two types of tests were administered:

- Using city sets taken from TSPLIB. Some optimal solutions are already there.
- Using randomly chosen cities.

TSPLIB city sets are rather hard to solve. The reason for this is that in many cases cities are not chosen randomly (Figures 9, 10). Often larger city sets consist of smaller patterns. City set shown in Figure 10 consist of two different patterns, but each of them is used eight times. Therefore, the optimal tour is identical in each one of these smaller patterns (Figure 10, left). SOM tries to figure out a unique tour in each smaller pattern (Figure 10, right).

Testing using randomly chosen cities is more objective. It is based on the Held-Karp Traveling Salesman bound [8]. An empirical relation for expected tour length is used:

$$L = k\sqrt{n \cdot R}$$

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 $n \geqslant 100$ it is:

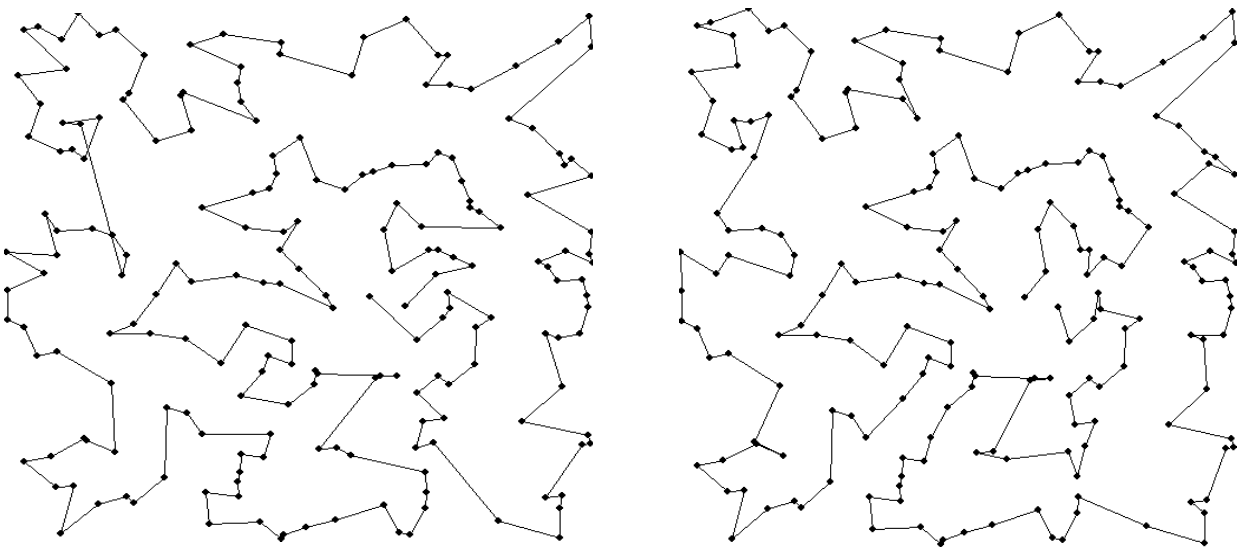

Figure 8: SOM solution without 2opt optimization (left). There are two local loops on the left. First and last neuron can be seen in the middle. They are not connected it the picture, but distance between them is also computed. The same solution improved by 2opt (right). Loops on the left have been erased. Additional changes can be observed.

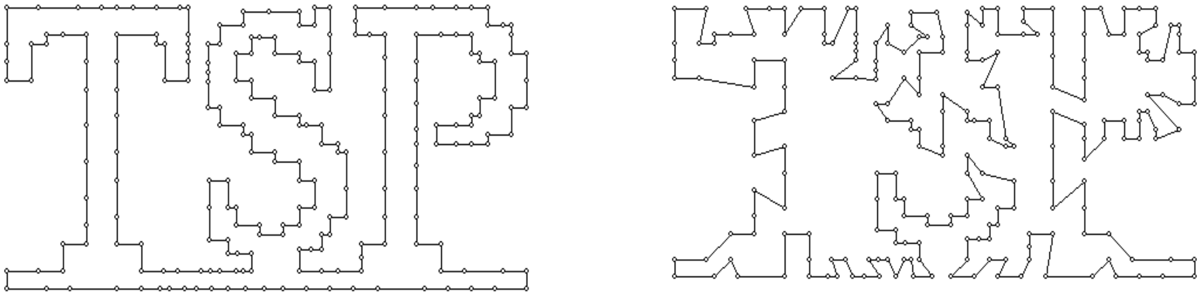

Figure 9: Optimal tour for 225 city set taken from TSPLIB (left). It's length is 3916. Tour generated by SOM 2opt hybrid (right). It's length is 4130, which is 5.19% worse than optimum.

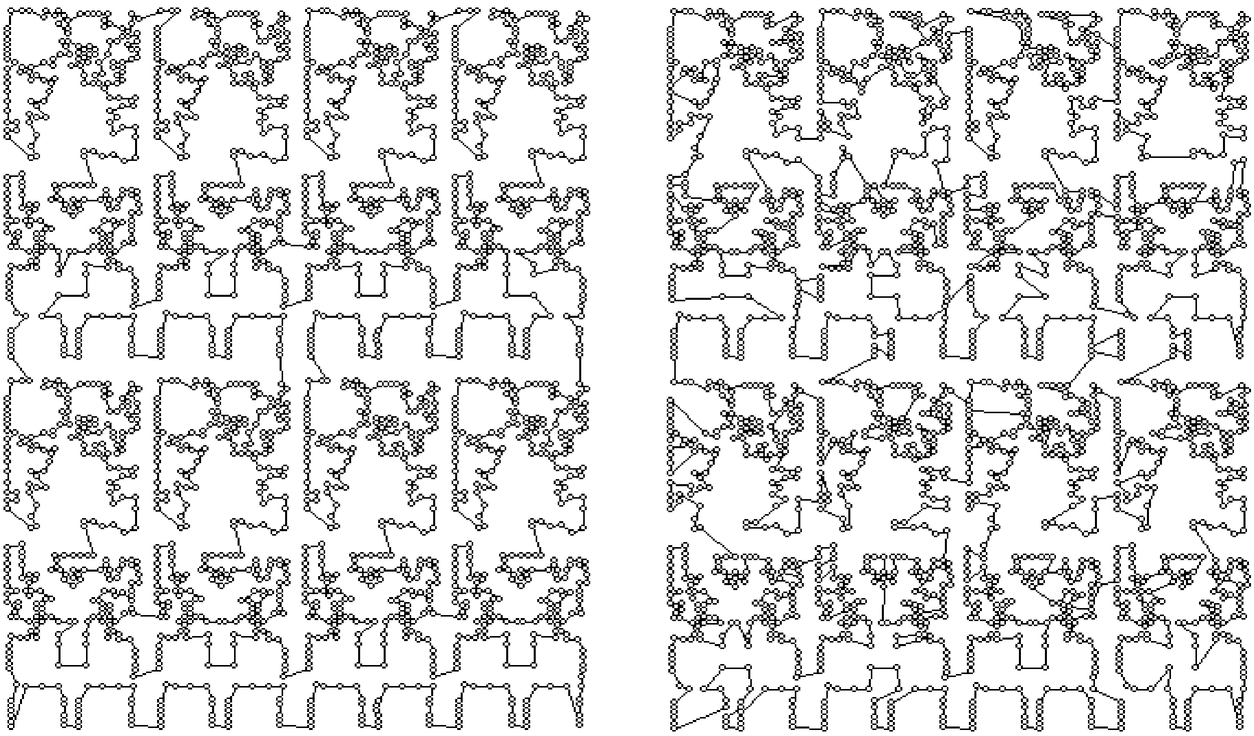

Figure 10: Optimal tour for 2392 city set taken from TSPLIB (left). It's length is 378037. Tour generated by SOM 2opt hybrid (right). It's length is 411442, which is 8.12% worse than optimum.

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

Three random city sets were used in this experiment (100, 500, 1000 cities). Square box edge length was 500.

All statistics for SOM were generated after 50 runs on each city set. When amount of iterations was adjusted to 100 average results did not change a lot. SOM can generate a tour in relatively short time. 225 city set is solved during just 300 ms, and 1000 city set in less then 2.5 second (using machine described later). Average tour lengths for city sets up to 2000 cities are around 5 to 6 percent worse than optimum. SOM approach can generate solutions that are almost always less that 10% worse from the optimal tour. However, in most cases the difference is just a few percent. SOM has been compared with evolutionary algorithm (EA). EA used Enhanced Edge Recombination (EER) operator[2, 5], Steady-State survivor selection (where always the worst solution is replaced), Tournament parent selection with tournament size depending on number of cities and population size. Scramble mutation was used. Optimal mutation rate depends on amount of cities and state of evolution. Therefore, self-adapting mutation rate has been used. Every genotype has it's own mutation rate, 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's not needed to check manually which parameters are optimal for each city set. Evolution stops when population converges. Population size was set to 1000 (as in [5]). With smaller populations EA did not work that well. When EA stopped it's best solution was optimized by 2opt algorithm. Results for both SOM and EA are shown in Table 2. All statistics for SOM were generated after 50 runs on each city set. For EA there were 10 runs of the algorithm for sets: EIL51, EIL101 and RAND100. For other sets EA was run only once. Optimum solutions for instances taken from TSPLIB were already there and optimum solutions for random instances are calculated from empirical relation described above. All computations were performed on AMD Athlon 64-bit 3500+ processor. However, the program was built using 32-bit compiler, so it did not use full computational power of this machine.

Experiments show that EA finds better solutions for instances with up to 101 cities. Both Average and Best Results are better than SOM's. For city sets with 50 or less cities EA finds optimum in almost every execution. Results for 225 cities are comparable for both algorithms, however for larger amount of cities (442 and more) SOM wins the competition. The more cities the instance has, the bigger the difference between both algorithms. With more cities, search space increases significantly and EA needs bigger population size. For TSP225 with population size 1000 EA's result was 4044, but when population size was expanded to 3000 a tour with length 3949 was found - much better than SOM's solution. This underlines the fact that when EA is used one can always expand population size so the algorithm has greater chance of achieving good result. Unfortunately, the algorithm is much slower then.

It is interesting to compare SOM algorithm to other non-evolutionary approach. One of the best TSP algorithms, which is also extremely fast, is the Lin-Kerninghan algorithm. Results for this algorithm have been taken from [4]. The algorithm was run 10 times on each city set. Average results and average times were taken from are shown in Table 1. A Pentium Pro 180 Mhz was used, so it is rather hard to compare times from Table 1 and times for SOM from Table 2 (an AMD 3500+ processor was used). Anyway, Lin-Kerninghan is faster than SOM even on a many times slower machine. There is not a big difference in time for a small 51-city instance (0.012 seconds for Lin-Kerninghan and 0.068 seconds for SOM). On the other hand for 2392-city instance Lin-Kerninghan needed just 0.719 seconds and SOM almost 13 seconds. This is because SOM 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 wins in all cases. The more cities there is, the bigger the difference between both algorithms.

SOM was also used to generates initial population for EA. Such initialization takes only a fraction of time needed for EA to finish, because SOM is a fast algorithm. In this case EA tended to converge much faster and finally it did not improve much best solution generated by SOM alone. It seems that all initial solutions were very similar to each other, thus population diversity was low, so EA lost exploration abilities.

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

Table 1: Results for the Lin-Kerninghan algorithm

6 Conclusions

It seems that SOM-2opt hybrid is not a very powerful algorithm for the TSP. It has been outperformed by both: EA and Lin-Kerninghan algorithms. It's speed might be impressive, but it still is slower than Lin-Kerninghan.

There are a couple of things that can be optimized. Here are some of them:

- an optimal network parameter settings should be found (η , $\Delta\eta$, $\Delta\lambda$, number of iterations)
- experiments with other self-organizing networks should be performed, Gaussian neighborhood and “conscience mechanism” may be applied. Conscience mechanism can improve TSP solutions generated by neural networks, as reported in [6].
- 2opt algorithm is not very sophisticated. Some other optimization method may be better.

The are many algorithms that solve permutation problems. Evolutionary Algorithms have many different operators that work with permutations. EER is one of the best operators for the TSP [5]. However, it was proved that other permutation operators, which are worse for the TSP than EER, are actually better for other permutation problems (like warehouse/shipping scheduling) [5]. Therefore, it might be possible that SOM 2opt hybrid might work better for other permutation problems than for the TSP.

		Self-Organizing Map			Evolutionary Algorithm		
Instances	**Optimum**	**Ave. Result**	**Best Result**	**Ave. Time**	**Ave. Result**	**Best Result**	**Ave. Time**
EIL51	426	444	431	0.068	428.2	426	10
EIL101	629	662	646	0.127	653.3	639	75
TSP225	3916	4193	4106	0.302	----	4044	871
PCB442	50778	56634	55138	0.703	----	55657	10395
PR1002	259045	278481	274036	2.425	----	286908	25639
PR2392	378037	418739	411442	12.965	----	----	----
RAND100	3851,81	4051	3883	0.131	3931.4	3822	69.6
RAND500	8203,73	8888	8697	0.824	----	9261	11145
RAND1000	11475,66	12483	12343	2.311	----	12858	56456

Table 2: SOM and Evolutionary Algorithm comparison

Acknowledgements

I am grateful to Prof. Zbigniew Michalewicz for influencing and helping me to write this paper.

Bibliography

1. Kohonen T. (2001), Self-Organizing Maps, Springer, Berlin
2. Michalewicz Z. (1996), Genetic Algorithms + Data Structures = Evolution Programs, Springer – Verlag
3. Arbib M. (1998), The Handbook of Brain Theory and Neural Networks, The MIT Press
4. Tao G., Michalewicz Z., Inver-over Operator for the TSP
5. Starkweather T., McDaniel S., Whitley C., Mathias K., Whitley D., (1991), A Comparison of Genetic Sequencing Operators
6. Burke Laura I., (1993), Neural Methods for the Traveling Salesman Problem: Insights From Operations Research
7. Xu W., Tsai W. T. (1990), Effective Neural Algorithm for the Traveling Salesman Problem
8. Johnson, D.S., McGeoch, L.A., and Rothberg, E.E., Asymptotic experimental analysis for the Held-Karp traveling salesman bound
9. Lin S., Kernighan B. W., (1971), An effective Heuristic for the Traveling Salesman Problem
10. Garey M., Johnson D., (1971), Computers and Intractability, W.H. Freeman, San Francisco
11. Korte B., (1988), Applications of Combinatorial Optimization
12. Reinelt G., (1995), TSPLIB 95 documentation, University of Heidelberg