11th Symposium on Neural Network Applications in Electrical Engineering, *NEUREL-2012*
Faculty of Electrical Engineering, University of Belgrade, Serbia, September 20-22, 2012

Comparison of Neural Networks for Solving the Travelling Salesman Problem

Bert F. J. La Maire, Valeri M. Mladenov

***Abstract*—The TSP deals with finding a shortest path through a number of cities. This seemingly simple problem is hard to solve because of the amount of possible solutions. Which is why methods that give a good suboptimal solution in a reasonable time are generally used.**

In this paper three methods were compared with respect to quality of solution and ease of finding correct parameters: the Integer Linear Programming method , the Hopfield Neural Network, and the Kohonen Self Organizing Feature Map Neural Network.

***Index Terms*—Integer Programming, Hopfield Neural Network, Kohonen Self Organizing Feature Map Neural Network, Traveling Salesman Problem**

I. Introduction

In many fields of operation, it is important to find a shortest path through a collection of points. The most common example is the delivery of packages. The shortest route is chosen to save fuel and labour cost. Other examples are the drilling of printed circuits and order picking in a warehouse [1].

In Operations Research this type of problem is called the Traveling Salesman Problem (TSP). The problem states that a salesman has to visit a set of cities all only once, using the route within the shortest travelling distance. Solving the TSP seams fairly simple, but because this problem is in the class of NP-complete problems, it is rather hard to solve it.

NP complete problems can not be solved in polynomial time, but can be verified in polynomial time. Because of this property the problem is usually solved by methods which give a good suboptimal solution in a reasonable time. [2]

Many methods have been proposed to solve the TSP. An overview of some exact and approximate algorithms to solve the TSP can be found in the work of Laporte [3]. Amongst them are textbook methods such as the integer linear programming formulation and the branch-and-bound method.

Another class of solution methods is that of artificial neural networks. A review of neural networks is presented in the work of Altinel *et al.* [4]. One of the first artificial neural networks designed to solve the TSP is the model as proposed by Hopfield and Tank [5].

Recent work on this network can be found in Sarwar and Bhatti [6]. They compare a Hopfield Neural Network to a heuristic algorithm and propose new parameters for the Hopfield Neural Network. The results are still dissatisfying in comparison with the heuristic algorithm.

B. F. J. La Maire is with Department of Chemical Engineering and Chemistry, Eindhoven University of Technology, PO box 513, 5600 MB Eindhoven, The Netherlands

V. M. Mladenov is with the Department of Theoretical Electrical Engineering, Technical University of Sofia 8, Kliment Ohridski St, Sofia-1000, Bulgaria, (e-mail: valerim@tu-sofia.bg)

Surveys on hardware implementation of the Hopfield Neural Network can be found in the work of Lau and Widrow [7], Woodburn and Murray [8], Luo and Unbehauen [9], and Zhang [10].

Another well known neural network suitable for solving the TSP is the Kohonen (Self Organizing Feature Map) Neural Network. A general overview of applications of this type of neural network is given in the article of Kohonen [11]. A more specific treatment on the use of this network for the TSP can be found in the work of Agéniol *et al.* [12].

In most other work on these methods, results from one or at most two methods are analysed, e.g. the comparison between human planners and heuristic algorithms by Hill [13] and more recently the work of Sarwar and Bhatti [6], comparing the Hopfield Neural Network to the heuristic algorithm by Keld and Helsgaun.

In this article a comparison on solution quality and ease of finding the right parameters is made of the Integer Programming method, the Hopfield Neural Network and the Kohonen Neural Network approach, using particular instances of the TSP.

The remainder of the paper is organized in four sections. Starting with section II, in which the test cases and the use of the three solution methods is presented. Next, in section III the results of the comparison are discussed. Finally, in section IV conclusions are drawn and recommendations are made.

II. Utilized Methods

In this section, first, the problem instances that were used to compare the Integer Programming, the Hopfield Neural Network and the Kohonen Neural Network methods are described. And second the use of these three methods is explained.

A. Traveling Salesman Problem Instances

Four problem instances with respectively 20, 40, 60 and 80 cities were generated using MATLAB. The script generated random coordinate pairs in the unit square and calculated the resulting mutual distances between the cities.

B. Integer Programming

Laporte gives the following Integer Linear Programming (IP) formulation of the TSP [3]:

$$\text{Minimalize} \quad \sum_{i,j} x_{i,j}\, d_{i,j} \tag{1}$$

$$\text{Subject to} \quad \sum_{i} x_{i,j} = 1,\ j \in \{1,\ldots,n\} \tag{2}$$

$$\sum_{j} x_{i,j} = 1,\ i \in \{1,\ldots,n\} \tag{3}$$

$$\sum_{i,j \in S} x_{i,j} \le |S| - 1, \quad S \subset C,\ 2 \le |S| \le n-2 \tag{4}$$

$$x_{i,j} \in \{0,1\}, i,j = 1,\ldots,n,\ i \ne j. \tag{5}$$

Equation (1) is the linear objective function. In this equation $x_{i,j}$ is the decision variable that has value 0 if the path from city i to city j is not used and 1 if the path is used. The following contraints are added to ensure that solutions actually form a valid tour: constraints (2) and (3) ensure that every city is respectively left and entered once, constraints (4) ensure that no subtours are formed, i.e. solutions in which two or more separated tours exist. Finally constraint (5) imposes binary conditions on decision variable x.

The IP method as described by Jünger *et al.* [1] has been used. In this method, first constraint (4) and (5) are relaxed. This results in an easier to solve linear programming problem. This solution is subsequently tested for sub tours. If such sub tours still exist, a cut is introduced to the original problem, i.e. an extra constraint that avoids choosing the same sub tour again.

The implementation of this method was taken from the GAMS model library (http://www.gams.com/). The outcome of this method was proved to be very close to the optimal solution by Jünger *et al.* [1] and the outcomes of this method will therefore be used as a reference solution for the Hopfield and Kohonen Neural Network outcomes.

C. Hopfield Neural Network

For the Hopfield Neural Network, the work of Bose and Liang [2] was followed. A grid of n by n neurons (n is the number of cities) like in figure 1 is used. Each neuron has two indices, one indicating the city (denoted by k or r) and one indicating the step (denoted by i or j). Each neuron is connected to all the other neurons. The base for the connection weights is the following error function:

$$\begin{aligned} E = \; & \tfrac{A}{2}\sum_{k}\sum_{i}\sum_{j\ne i}\nu_{ki}\nu_{kj} + \frac{B}{2}\sum_{j}\sum_{k}\sum_{r\ne k}\nu_{kj}\nu_{rj} \\ & + \tfrac{C}{2}\left(n - \sum_{k}\sum_{i}\nu_{ki}\right)^2 \\ & + \tfrac{D}{2}\sum_{k}\sum_{r\ne k}\sum_{j} d_{kr}\nu_{kj}\left(\nu_{r(j+1)} + \nu_{r(j-1)}\right) \end{aligned} \tag{6}$$

The first term (A) of equation (6) sums up the penalty for not visiting city k only once. The second term (B) sums up the penalty for visiting more than one city in step j. The third term (C) adds a penalty for not visiting n cities in total and finally the last term (D) sums up the total distance of the tour.

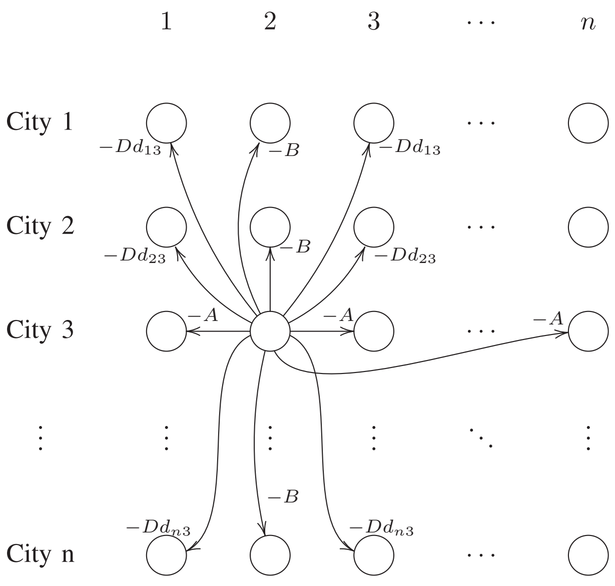

Fig. 1. A neural network for solving the TSP. Connections for only one neuron are shown. The connection weights are explicitly shown, and the minus sign indicates that these weights are inhibitory. The distance between the *i*th and *j*th cities is d_{ij}, and the upper-case letters are selected constants. This figure was adapted from the work of Bose and Liang [2].

The weights associated with each connection, as displayed in figure 1, can be derived from the energy function as follows. The energy function, equation (6), ignoring the constant term, can be rewritten as:

$$E = -\frac{1}{2}\sum_{ki}\sum_{rj} w_{(ki)(rj)}\nu_{ki}\nu_{rj} - \sum_{ki}\nu_{ki}I_{ki} \tag{7}$$

with

$$\begin{aligned} w_{(ki)(rj)} = & -A\delta_{kr}\left(1-\delta_{ij}\right) - B\delta_{ij}\left(1-\delta_{kr}\right) - C \\ & -Dd_{kr}\left(\delta_{j(i+1)} + \delta_{j(i-1)}\right) \end{aligned} \tag{8}$$

where δ is the Kronecker Delta.

The value of each neuron is determined by the activation function:

$$\nu_{ki} = \frac{1}{2}\left(1 + \tanh\left(\frac{a_{ki}}{a_0}\right)\right) \tag{9}$$

This function assigns a value to each neuron ν_{ki} between 0 and 1, according to the weighed sum of the values of the other nererons a_{ki}. Parameter a_0 determines the steepness of the activation function.

The network was initialized by setting each neuron to random values between 0 and 1 to begin with. The output of the Hopfield Neural Network is a table of values ν_{ki} between 0 and 1. This is equivalent to the Integer Linear Program that was presented in equations (1)-(5) where the binary condition (5) is relaxed.

To find actual routes, an heuristic is used to enforce the binary condition. This heuristic finds the largest value in each column of ν_{ki}, and sets this value to 1. All other values are set to 0. Finally it is checked whether this gives a feasible route.

To smooth out the effect of the random initial state of the network, each problem instance was solved three times (with different random initial states). The average outcome is reported.

Using the separate terms of the error function, equation (7), the effect of changing error function parameters $A - D$ can be evaluated. The error term values presented in this paper are not pre multiplied by the error function parameters $A - D$.

The implementation of this method was performed in MATLAB and was not optimized for speed, therefore no comparison on calculation time will be presented.

D. Kohonen Self Organizing Feature Map Neural Network

The Kohonen Neural Network was implemented after the work of Ghaziri and Osman [14]. They start with a set of cities in the unit square. Then add a circle on which m equidistant neurons are placed. To calculate the distance between a city and a neuron, the Euclidean distance is used and to calculate the distance between two neurons, the lateral distance is used, i.e. the distance between the two neurons on the ring.

In each step of the algorithm, the neuron that is closest to a randomly chosen city is determined. This neuron is then moved towards the city. All other neurons on the ring also move towards that city, but with a decreasing intensity for neurons with larger lateral distances.

To avoid oscillation of a neuron between different neighbouring cities, the number of neurons m should be at least three times the number of cities n ($m \geq 3n$).

In our implementation it was chosen to use a linear function to describe the attraction of the neuron, characterized by an initial weight (the fraction of distance that is left after the neuron is moved) and the number of attracted neurons (after which the neurons no longer move).

Again, the implementation of this method was performed in MATLAB and was not optimized for speed, therefore no comparison on calculation time will be presented.

The output of this method is an approximate route. An heuristic first finds the nearest neuron to each city. Next the cities are sorted by the place on the ring of the neurons that are closest to them. In this way, an actual route is found.

III. Experimental Results & Discussion

After having discussed the utilized methods, this section will discus the experimental results.

A. Integer Programming

The results of the Integer Programming method are presented in table I. The results are the length of the optimal route (objective value), the number of cuts used to eliminate sub-tours and the number of possible routes. As mentioned before the Integer Programming results in a provably good route through all cities. The number of cuts shows the number of sub tours that had to be eliminated. The total number of possible solutions shows that the solution space of the problem grows rapidly when the problem size increases.

B. Hopfield Neural Network

For the Hopfield Neural Network we will only present the 20 city case, because the method used produces a large amount of data. The best found results for respectively the 40, 60 and 80 city problem instances can be found in table IV.

TABLE I
Results of solving four TSP model problems using integer programming

Cities	Tour length	Cuts	Possible routes
20	4.121	24	$6.08 \cdot 10^{16}$
40	5.900	110	$1.02 \cdot 10^{46}$
60	6.634	150	$6.93 \cdot 10^{79}$
80	7.238	1085	$4.47 \cdot 10^{116}$

TABLE II
Results for solving the 20 city problem instance using the Hopfield Neural Network

ID	**Parameters**				a_0	Δt	**Error in contraint**				**Dist.**
	A	**B**	**C**	**D**			**1**	**2**	**3**	**4**	
1	1	1	1	1	15	3	25.80	25.81	84.22	24.96	—
2	40	40	40	40	15	3	2.94	3.91	1.36	7.44	—
3	80	80	40	40	15	3	0.03	0.05	1.24	8.11	10.31
4	80	80	40	40	15	20	0.002	0.002	1.81	8.15	10.94
5	70	70	50	50	15	20	0.12	0.07	1.13	8.57	11.29
6	40	40	80	80	15	20	0.46	8.65	0.40	1.98	—
7	40	60	80	80	15	20	0.26	7.47	0.56	2.19	—
8	40	80	80	80	15	20	1.61	1.90	0.74	7.68	—
9	40	100	80	80	15	20	1.77	0.08	0.69	9.07	—
10	60	100	80	80	15	20	0.90	0.01	0.80	7.93	—

The results of the 20 city problem instance, solved using the Hopfield Neural Network method are shown in table II. The first column gives an identifier to the parameter set used to solve the problem. The next four columns give the values of the error function parameters $A - D$ (as in equation (6), a_0 is the steepness parameter in the activation function (9) and Δt is the time period for which the differential equations that describe this system were solved. Finally the individual terms of the error function are presented and the mean total distance of three repetitions.

Each new parameter set was chosen with to give a better outcome than the previous setting. For example, from the difference in the results of parameter settings 1, 2 and 3 it can be seen that enlarging the error function parameters has a positive effect on the overall value of the error function of the network.

From the difference between parameter set 6 and 7 it can be clearly seen that all terms of the error function are interdependent, i.e. changing only one error function parameter effects all of the error function terms.

A final finding was that only 3 out of 10 parameter settings resulted in feasible routes.

C. Kohonen Self Organizing Feature Map Neural Network

Also for the Kohonen Neural Network only the 20 city problem instance is presented in detail in table III. Here the first column gives an identifier to each parameter set. Next the number of epochs, neurons and the two parameters that define the linear function for the attraction of neurons. Finally the relaxed distance found before applying the heuristic, and finally the distance of the tour after applying the heuristic. The best found solutions for respectively the 40, 60 and 80 city problem instances can be found in table IV.

TABLE III
RESULTS FOR SOLVING THE 20 CITY PROBLEM INSTANCE USING THE KOHONEN SELF ORGANIZING FEATURE MAP NEURAL NETWORK

ID	Epochs	Neurons	Initial Weight	Attracted neurons	Distance	
					Relaxed	Tour
1	100	60	0.5	3	4.29	4.83
2	100	60	0.5	4	3.60	4.30
3	1000	60	0.5	4	3.57	4.26
4	1000	80	0.5	4	4.04	4.15
5	1000	80	0.4	4	4.23	4.33
6	1000	80	0.3	4	4.12	4.25

From the results in table III it can be seen that a close to optimal route was found in only four steps. The parameter settings were chosen by visually inspecting the behaviour of the network, i.e. by looking at the state of the network at several moments in time. For example, in the first run, it was seen that some of the neurons stayed in their initial position. In the second run therefore the number of attracted neurons was enlarged. Then in the second run it was seen that not each city had enough chance to draw neurons towards itself, so in run three the number of epochs was enlarged, resulting in again a better route.

D. Comparison of Solution Methods

In table IV a summary of the best found solutions for all problem instances is given. In general, the Integer Programming solutions were expected to be very close to optimal solutions. Only in the 40 city case, a better solution was found by the Kohonen Neural Network.

The main conclusion is that the Integer Programming and Kohonen Neural Network resulted in close to optimal solutions, whereas the Hopfield Neural Network stayed far from optimal.

TABLE IV
COMPARISON OF THE RESULTS OF THE DIFFERENT SOLVING METHODS

	Tour length			
	20 cities	40 cities	60 cities	80 cities
Integer Programming	4.121	5.900	6.634	7.238
Hopfield Neural Network^1	10.307	20.697	29.090	24.435
Kohonen	4.152	5.706	6.539	7.771

1) This value is an average of the feasible routes found in this run.

IV. CONCLUSION & RECOMMENDATIONS

The aim of this work was to compare the solution quality and the ease of finding the right parameters of the Integer Programming method, the Hopfield Neural Network and the Kohonen Self Organizing Feature Map Neural Network.

All three methods were successfully implemented and tested with four TSP instances, containing 20, 40 60 and 80 cities respectively.

From the results it was found that overall the Integer Programming method is the best method, because it is very efficient and finds the nearly optimal solution with little effort, without other input than the mutual distances of the cities.

The Kohonen Neural Network was found to come close to the performance of the Integer Programming method, in terms of solution quality, but still needs some parameter tuning by the user.

The Hopfield Neural Network performed insufficient in comparison with the Integer Programming method and the Kohonen Neural Network; it required much work to tune the parameters and resulted in poor solutions. The poor solutions can be understood from the fact that in this work the network was simulated in software, rather than implemented on hardware.

The Hopfield Neural Network implemented on hardware enables to solve the problem in much shorter time (real time, due to parallel execution) and carry out more executions of the problem with different initial conditions, resulting in a higher chance of finding the global optimum amongst many global minima.

REFERENCES

[1] M. Jünger, S. Thienel, and G. Reinelt, "Provably good solutions for the traveling salesman problem," *Mathematical Methods of Operations Research*, vol. 40, pp. 183–217, 1994.

[2] N. K. Bose and P. Liang, *Neural Network Fundamentals with Graphs, Algorithms, and applications*, ser. McGraw-Hill Electrical and Computer Engineering Series. McGraw-Hill, Inc., 1996.

[3] G. Laporte, "The traveling salesman problem: An overview of exact and approximate algorithms," *European Journal of Operational Research*, vol. 59, pp. 2331–247, 1992.

[4] I. K. Altinel, N. Aras, and B. J. Oommen, "Fast, effcient and accurate solutions to the hamiltonian path problem using neural approaches," *Computers & Operations Research*, vol. 27, pp. 461–494, 2000.

[5] J. J. Hopfield and D. W. Tank, ""neural" computation of decisions in optimization problems," *Biological Cybernetics*, vol. 52, pp. 141–152, 1985.

[6] F. Sarwar and A. A. Bhatti, "Critical analysis of hopfield's neural network model for tsp and its comparison with heuristic algorithm for shortest path computation," in *Proceedings of 2012 9th International Bhurban Conference on Applied Sciences & Technology (IBCAST)*, 2012.

[7] C. Lau and B. Widrow, Eds., *Special Issue on Neural Networks*, ser. Proceedings IEEE, vol. 78, September/October 1990.

[8] R. Woodburn and A. Murray, *Neural Network Analysis, Architecture and Applications*, 1997, ch. Pulse-Stream Techniques and Circuits for implementing Neural Netorks.

[9] F.-L. Luo and R.Unbehauen, *Applied Neural Network for Signal Processing*. Cambridge University Press, 1998.

[10] D. Zhang, *Parallel VLSI Neural System Design*. Singapore: Springer-Verlag, 1999.

[11] T. Kohonen, "The self-organizing map," in *Proceedings of the IEEE*, vol. 78, no. 9, 1990, pp. 1464 – 1480.

[12] B. Angéniol, G. de La Croix Vaubois, and J.-Y. L. Texier, "Self-organizing feature maps and the traveling salesman problem," *Neural Networks*, vol. 1, pp. 289–293, 1988.

[13] A. V. Hill, "An experimental comparison of human schedulers and heuristic algorithms for the traveling salesman problem," *Journal of Operations Management*, vol. 2, pp. 215–223, 1982.

[14] H. Ghaziri and I. H. Osman, "A neural network algorithm for the traveling salesman problem with backhauls," *Computers & Industrial Engineering*, vol. 44, pp. 267–281, 2003.