

# IMPLEMENTATION OF TRAVELING SALESMAN'S PROBLEM USING HOPFIELD NEURAL NETWORKS

FINAL PROJECT REPORT  
(Fall 2011)

EECS 484 Computational Intelligence  
December 9, 2011

Arunprasath Shankar

# Solving Traveling Salesman's Problem Using Continuous Hopfield Network

Arunprasath Shankar

Department of Electrical Engineering and Computer Science

axs918@case.edu

## **ABSTRACT**

I have proposed an implementation of an algorithm in neural network for an approximate solution for Traveling Salesman's Problem. TSP is a classical example of optimization and constrain satisfaction problem, which falls under the family of NP-complete of problems. I have used Continuous Hopfield network to find the solution for the given problem.

## **PROBLEM**

There is a list of cities that are to be visited by a salesman. A salesman starts from a city and come back to the same city after visiting all the cities. Here the objective is to find the path, which follows following constrains

- 1) Salesman has to visit each city. He should not leave any city unvisited.
- 2) Each city should be visited only one time.
- 3) The distance that he travels till he returns back to the city he has started should be minimum.

## **INTRODUCTION**

The traveling salesman problem (TSP) is well known in optimization. The TSP problem is NP-complete problem. There is no algorithm for this problem, which gives a perfect solution. Thus any algorithm for this problem is going to be impractical with certain examples.

Here we assume that we are given  $n$  cities, and a non-negative integer distance  $D_{ij}$  between any two cities  $i$  and  $j$ . We try to find the tour for the salesman that best fits the above-mentioned criterion.

There are various neural network algorithm that can be used to try to solve such constraint satisfaction problems. Most solution have used one of the following methods

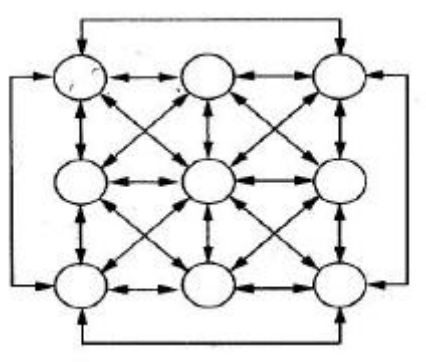
- Hopfield Network
- Kohonen Self-organizing map
- Genetic Algorithms

Here an approximate solution is found for TSP using Hopfield network.

## HOPFIELD NETWORK

Hopfield network is a dynamic network, which iterates to converge from an arbitrary input state. The Hopfield Network works as minimizing an energy function.

The Hopfield net is fully connected network. It is a weighted network where the output of the network is fed back and there are weights to each of this link. The fully connected Hopfield network is shown in following figure.



*Figure : Fully Connected Hopfield Network for TSP for 3 cities.*

Here we use  $n^2$  neurons in the network, where  $n$  is the total number of cities. The neurons here have a threshold and step- function. The inputs are given to the weighted input node. The network then calculates the output and then based on Energy function and weight update function, converges to the stable solution after little iteration. The most important task on hand is to find an appropriate connection weight. It should be such that invalid tours should be prevented and valid tours should be preferred.

## GENERAL PROCEDURE:

The use of Continuous Hopfield Feedback Network to solve 10-city traveling salesman problem (TSP) in which the coordinates are given as follows:

$$\begin{aligned}city1 &= (0.4000, 0.4439), city2 = (0.2439, 0.1463), \\city3 &= (0.1707, 0.2293), city4 = (0.2293, 0.7610), \\city5 &= (0.5171, 0.9414), city6 = (0.8732, 0.6536), \\city7 &= (0.6878, 0.5219), city8 = (0.8488, 0.3609), \\city9 &= (0.6683, 0.2536), city10 = (0.6195, 0.2634)\end{aligned}$$

Basic network parameters are:

$$A = B = D = 500, C = 200, \mu_0 = 0.02$$

In general, to solve any optimization problem using Hopfield Neural Networks, we must follow the procedure listed below with certain assumed prerequisites:

- (1) An appropriate representation corresponding to the problem to be solved.
- (2) A network energy function needs to be constructed to a minimum value so as to solve the specific problem (in our case TSP)
- (3) This energy function is then compared to the standard form. Neural network weights and bias expressions are introduced to meet the optimal solution for the problem.
- (4) By introducing network state update formula and by using iterative update formula – an optimal solution is obtained.

## MODIFIED GENERAL PROCEDURE FOR SOLVING TRAVELING SALESMAN PROBLEM

To solve Traveling Salesman Problem by using Continuous Hopfield Networks, the general procedure can be modified as follows:

- (1) Of N cities in TSP problem, in the travel route transposition matrix, each row and each column has only one element = 1, the rest is 0. One element of its abscissa X represents the city name, the vertical axis I represents the city's position in the access route.
- (2) The network energy function consists of four parts as shown below, so as to ensure the legitimacy of the length of the shortest route.
- (3) The equation for the energy function is obtained by comparing it with the standard form and is given as below. This equation is used to get the network weights and the bias.

First a representation scheme for the tour needs to be found, or, more specifically, the tour has to be encoded by the states of a set of neurons. Hopfield and Tank used a scheme in which  $N^2$  neurons are needed for a N-city problem. The tour can be conveniently represented by an  $N \times N$  matrix. If the  $ij$ -element is equal to one, then city  $i$  has the  $j^{\text{th}}$  position in the tour.

An example for the matrix representation of the TSP for a 4-city tour can be seen below:

|   | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| A | 0 | 1 | 0 | 0 |
| B | 0 | 0 | 0 | 1 |
| C | 0 | 0 | 1 | 0 |
| D | 1 | 0 | 0 | 0 |

The matrix indicates that city D is the first visited city. The total tour is D-A-C-B.

Secondly an appropriate Hopfield energy function and the corresponding weights have to be determined. Hopfield and Tank proposed the following energy function:

$$E = \frac{A}{2} \sum_x \sum_i \sum_{j \neq i} v_{xi} v_{xj} + \frac{B}{2} \sum_i \sum_x \sum_{y \neq x} v_{xi} v_{yi} + \frac{C}{2} \left( \sum_x \sum_i v_{xi} - n \right)^2 + \frac{D}{2} \sum_x \sum_i \sum_{y \neq x} d_{xy} v_{xi} (v_{y,i+1} + v_{y,i-1})$$

(4) Thus, the network update formula is:

$$\begin{cases} W_{xi,yj} = -A\delta_{xy}(1 - \delta_{ij}) - B\delta_{ij}(1 - \delta_{xy}) - C - Dd_{xy}(\delta_{j,i+1} + \delta_{j,i-1}) \\ I_{xi} = C \cdot n \end{cases}$$

## PROGRAMMING

Based on the above derivation, we can design the Hopfield network in MATLAB to solve the TSP.

The program obeys the following conditions:

**\* Program output rule:** As the end of each iteration cannot guarantee that the solution is legitimate, and when there is a large number of cities, it is inconvenient for the people to check the legality of the optimum route. Therefore the program should be able to state the legitimacy of the solution after the end of each iteration and if a legal solution is achieved, the program should terminate otherwise again start solving.

**\* Parameter adjustment:** In the experiment, it is found that when the network parameters take the values given for the first time, they have little access to legal solutions, observed after the end of each iteration of the solution, found that most of the next only eight have each row and column 1 and only one case, in addition to two all 0. This shows that the energy function is vital in ensuring the legitimacy of N in a relatively small proportion manner. Also the parameter choice is an important criterion to be noted.

**\* Iteration condition:** Provided the weights are symmetric, the hop field networks can be used as an approximate method for solving 0-1 optimization problems because the network converges to a minimum of the energy function. The proof of stability of such Hopfield network relies on the fact that the energy function is a Lyapunov function. Hopfield and Tank showed that if a combinatorial optimization problem can be expressed in terms of a quadratic energy function of the general form, a Hopfield network could be used to find locally optimally solutions of the energy function, which may translate to local minimum solutions of the optimization problem.

## THE PROGRAM FLOW

- I. Initialization: the number of cities, the city coordinates, and the network parameters
- II. Transposition matrix with random numbers and status array initialization
- III. Array and transposed array of state, a 1000-step synchronous update, the solution is finally transposed matrix V
- IV. To determine the legitimacy derived from V, if a legal solution, the order is given access, travel route maps and route length, the program ends; otherwise, go to step II.

## RESULTS:

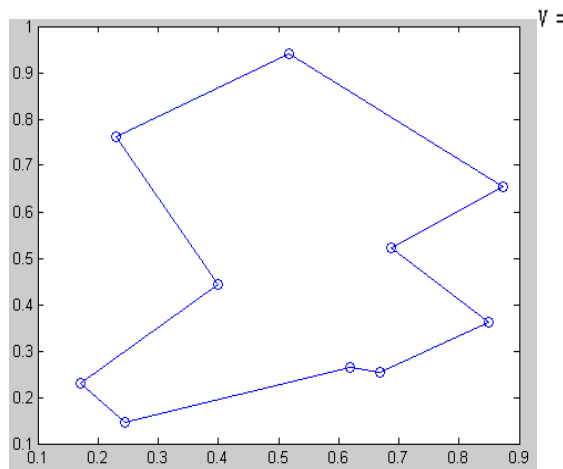
### 1. BASIC RESULTS

Take the number 10 in the city, take the basic parameters of the network is too run the program and statistical results, were:

$$A = B = D = 500, C = 1000, \mu_0 = 0.02, \text{lamda} = 0.0001$$

Run the program and statistical results, were:

|  |               |
|--|---------------|
| <i>Iteration Count</i>                           | <i>200</i>    |
| <i>Number Of Times The Legal Solution</i>        | <i>29</i>     |
| <i>Optimal Number</i>                            | <i>1</i>      |
| <i>Optimal Solution (Route Length)</i>           | <i>2.6907</i> |
| <i>Number Of Times The Sub-Optimal Solution</i>  | <i>1</i>      |
| <i>Sub-Optimal Solution (Route Length)</i>       | <i>2.7693</i> |
| <i>Distance Solution1 (Total Line Length)</i>    | <i>2.7782</i> |
| <i>Distance Solution2 (Total Line Length)</i>    | <i>2.8352</i> |
| <i>The Average Time Required For One Run (s)</i> | <i>0.8813</i> |



$V =$

|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

Figure1 The optimal route (2.6907)

Figure2 The optimal solution matrix transposition

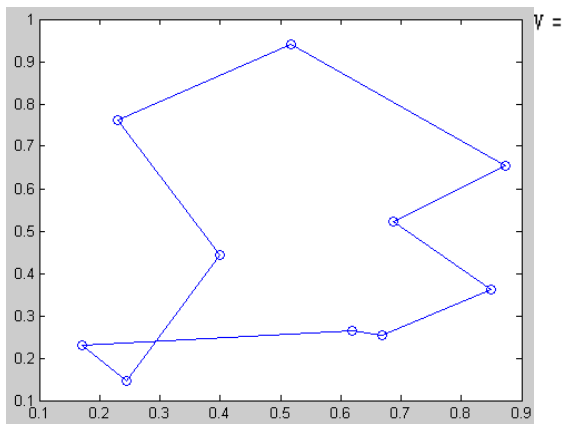


Figure 3 Sub-optimal route (2.7693)

$$V = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 4 Sub-optimal matrix transposition

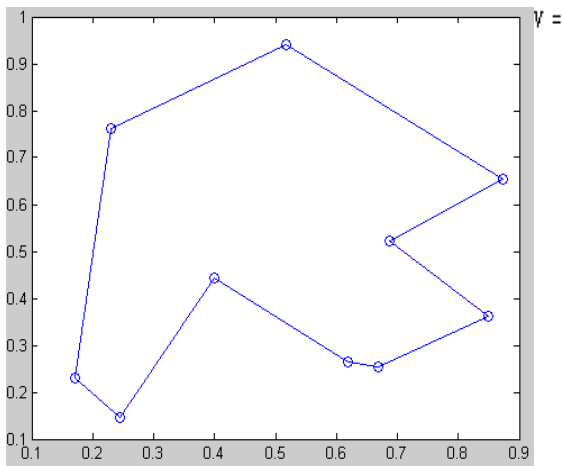


Figure Sub-Optimal Solution (Route Length)

$$V = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 6 Matrix transposition

## 2. INFLUENCE OF PARAMETERS

### (i) Run-time estimate

For number  $N$  in the city, step size  $\lambda$  is fixed and the case is updated each time solving the time spent  $V$  which is also fixed, therefore, more legitimate solution is obtained every time spent by comparing the number of cycles can be carried out. In the following discussion of the impact parameter, the number of cycles was compared by the relative length of time.



### *(ii) Weights A, B, C*

Weight matrix A, B, C, D reflects the relative size of the solution requirements, in which A, B, C is the solution to ensure the legal entry of weights. A is 1 per line to ensure that the weight coefficient; ensure B is at most one 1 per column weights; C is to ensure that there are N ones; D - The total length of the shortest route is to ensure the item weights.

When C is relatively small compared to A and B ( $A = B = 500$ ,  $C = 200$ ), the experiment is difficult to appear legitimate solution, the majority of the whole solution has two zeros, the program often falls into an infinite loop. This shows that the solution of the legitimacy of the third did not receive enough attention. Therefore, when there is a gradual increase in C, observe the result. When C is 500, we still do not notice any significant improvement in the situation. When C is taken as 1000, the legal solution significantly increases the frequency (200 times the experiment, the average once every 6.7 times the legal solution), which also appeared in the optimal solution (see 1 in the experimental results); when C is taken as 2000, it is the average of once every six full-solution method, which is also seen an optimal solution.

Summary, minor changes in C cannot guarantee the legitimacy of the solution, C larger when the frequency was significantly increased legal solution, but the shortest route C is large the right term coefficient D is relatively small; hence, the frequency of the optimal solution has declined.

### *(iii) Weights D*

Weight coefficient D reflects the length of the route the proportion of energy function. When D is taken as 200, an average of once every 1.5 times the legal solution, but the line length is very large, generally about 4.0, there can hardly be the optimal solution; when D is taken as 500, an average of once every 6.7 times the legal solution, optimal solution, which also appeared (see 1 in the experimental results); when D is taken as 600, appear legitimate solution decreased the frequency; when D is taken as 700, 151 running once distance solution; when D is taken as 1000, the program is almost caught in an infinite loop, there is very low risk of legal solutions.

Summary, D is small, relatively stronger legitimacy of mediation, resulting in a greater frequency of legal solutions, but the line length of the great; D is large, there has decreased the frequency of legal solutions, but the route length was significantly smaller, appear to increase the relative likelihood of the optimal solution; and when the D is too large, due to too much emphasis on line length, is difficult to appear legitimate solution, so the program is easy to freeze.

### *(iv) Step lambda*

Lambda of 0.0001, when the results as described in 1; when lambda taken to be 0.001, an average of once every 2.5 times the legal solution. Visible, lambda is large, larger changes in the state matrix, the solution will increase the frequency of appear legitimate; When lambda is too large, the state matrix becomes difficult due to dramatic changes appearing in the legitimate solution; When lambda is small, the update is too slow or even frozen.

(v) *Initial  $\mu_0$*

$\mu_0$  taken as 0.02 when the results obtained as described in 1; when  $\mu_0$  taken as 0.005, the average once every 3.6 times the optimal solution;  $\mu_0$  taken as 0.3, an average of once every 41 full-Solving.

$\mu_0$  *small*

Visible, small, discrete activation function value tends to shorten the optimization time there, but less prone to the optimal solution

$\mu_0$  *larger*

Greater activation function is too flat, not conducive to convergence.

### 3. CHANGE THE NUMBER OF CITIES

The number of cities is given below for 5 and 11 cities case, when the network parameters gives constant results. From the experimental results, decrease the number of cities, time optimization; we can see the solution to be legitimate and a significant increase in frequency of the optimal solution.

In addition to that, because of the impact of network parameters on the experiment with a similar front, not repeat them here.

(1) The number of 11 cities (11 cities in the coordinates (0.9125,0.9568))

Occur once every 7.5 times the legal solution, the solution of which a route length of 3.1382, graphics are as follows:

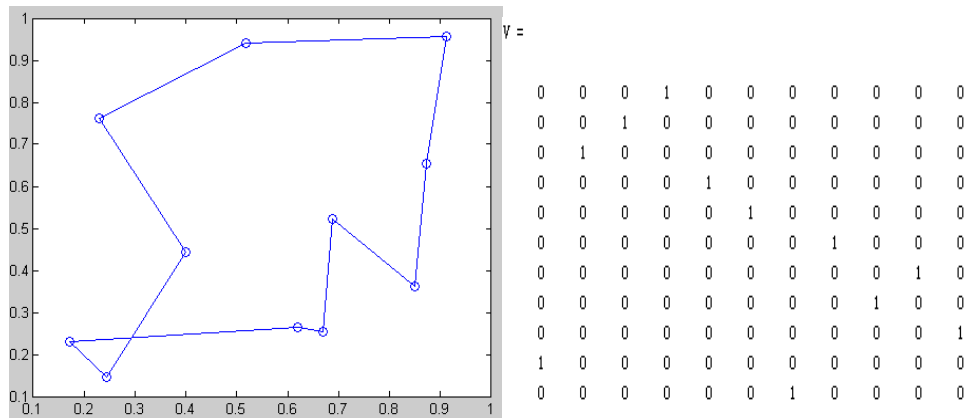


Figure 7 11 City TSP

(2) The number of cities 5 (taking the coordinates of the top 5 cities)

An average of once every five times the legal solution, 35 experiments appears three times in the optimal solution. Optimal solution is 1.8324, which also occur several times solution 1.8904, graphics are as follows:

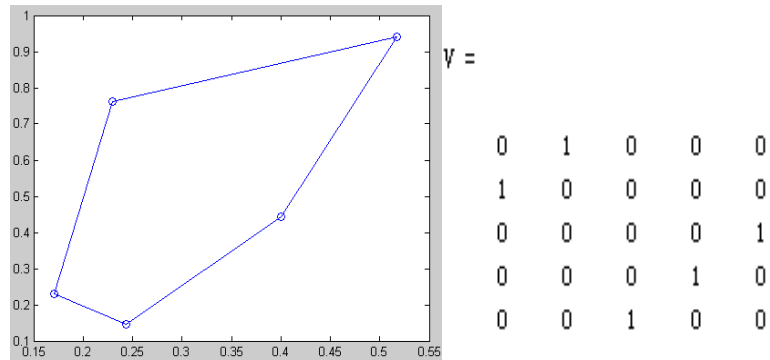


Figure 8 5 City TSP (Optimal Solution)

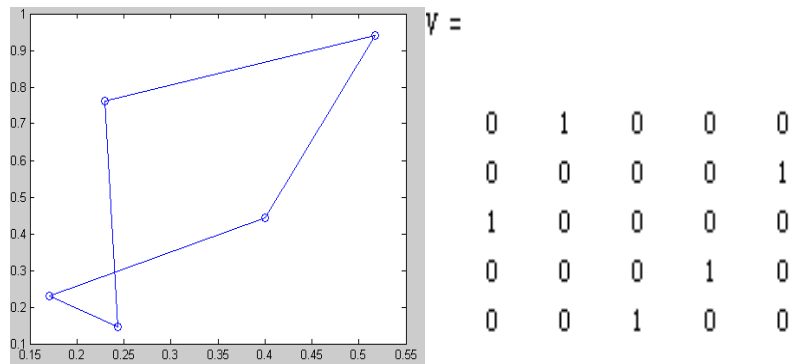


Figure 9 5 City TSP

## CONCLUSION

I think Hopfield's idea to solve the TSP with a dynamic system is very elegant. However, according to the experiences gained by this project, I conclude that the Hopfield net is not the best effective methodology that is applicable for solving combinatorial optimization problems. Hopfield net produces only a very low percentage of optimal solutions. The problem is that a constrained optimization problem is transformed into an unconstrained one. Furthermore, the optimal choice of the coefficients of the energy function is unknown.

A second disadvantage of the Hopfield net is that  $N^2$  neurons are needed to solve a N-city TSP. Hence computation time increases exponentially with the number of cities. The calculation time of my program for a 10-city TSP and 1000 time steps, for example, was about 8.5s on a Mac with a clock rate of 2 GHz. Therefore, even when using a faster programming language, it is not very efficient to use a Hopfield net to solve TSPs with hundreds or thousands of cities.

## REFERENCES

- [1] Hopfield, J.J and Tank, D.W., Neural Computation of Decision in Optimization Problems, Biol. Cybern. No. 52
- [2] Laurene V. Fausett, Fundamentals of Neural Networks: Architectures.
- [3] S. Haykin, 1999. Neural Networks-a Comprehensive Foundation, Prentice Hall