A neural network algorithm in matrix form and a heuristic greedy method for Traveling Salesperson Problem

NICOLAE POPOVICIU
Hyperion University of Bucharest, Faculty of Mathematics-Informatics
Street Călăraşilor 169, Bucharest, ROMANIA
MIOARA BONCUȚ
Lucian Blaga University of Sibiu, Department of Mathematics
Avenue Victoriei 110, Sibiu, ROMANIA

Abstract. The work describes all the necessary steps to solve the traveling salesperson problem. This optimization problem is very easy to formulate -and a lot of works do it-, but it is rather difficult to solve it. The section 2 gives a heuristic greedy method and a numerical example, with the mention that this method doesn't assure the optimal route. By using [4] as a main reference, we formulate an algorithm in a matrix form to solve the problem for optimal route. The mathematical approach is based on Hopfield neural networks and uses the energy function with the descent gradient method. The algorithm in matrix form is easier to use or to write a computational program. The work has seven sections. The section 6 describes the algorithm to solve the traveling salesperson problem and the section 7 contains another numerical example.

Key-Words. Traveling salesperson problem, traveling salesperson algorithm, energy function, descent gradient, greedy method.

1 Introduction

The traveling salesperson problem (TSP) is an optimization problem. A salesperson must make a closed circuit through a certain n number of cities, visiting each of them only once , minimizing the total distance traveled and the salesperson returns to the starting point at the end of the trip.

We denote by

$$K = K_{n\times n}, K = (d_{XY}) , d_{XX} = 0 .$$

the distances matrix, where d_{XY} is the distance between the cities X and Y .

Related with TSP problem we have three types of solutions : a) the *possible solution* (the salesperson passes many times through certain cities); b) the *admissible solution* (the salesperson passes only once through each city, but the distance traveled is not minim); c) the *optimal solution* (the solution is admissible and the distance traveled is minim) . We are interested in finding the optimal solution.

Our task is to find the *unknown weights* v_{Xj} , the elements of *weights matrix* V

$$V = V_{N\times N}, V = (v_{Xj}),\ X = 1,n;\ j = 1,n$$

which describes the optimal solution, where the subscript X refers to the city and the subscript j refers to the position of the city X on the tour (route) R. In any admissible solution is satisfied the condition $v_{XJ} \in \{0;1\}$, and the weight changes with the route R, i.e. $V = V(R)$.

We denote by $R(n)$ all possible tours in a n-city problem. Then $R(n) = \frac{n!}{2n} = \frac{(n-1)!}{2}$. The function $R(n)$ is a rapidly increasing function [4]. So we obtain

For TSP problem there exists two cases.

Case 1. $n \le 6$. The optimal solution can be obtained by an exhaustive search through all admissible routs.

Case 2. $n \ge 7$. In this case the TSP problem belongs to the class known as NPC (non possible complete) problem. For several values of n we obtain

$$R(7) = 360, R(8) = 2520, R(9) = 20160$$

$$R(10) = 181440, R(11) = 1814400.$$

Nevertheless there are several methods which have as a goal to obtain a good admissible solution or even the optimal one.

2 Several methods for TSP. The greedy method

The work [1], chapter 4, gives a history of TSP computation and mentions several methods to solve TSP:

a). Branch-and-bound method.
b). Dynamic programming.
c). Gomory cuts.
d). The Lin-Kernighan heuristic.
f). TSP cuts.
g). Branch-and-cut method.

Here we present a heuristic approach named **greedy method.** For simplicity we denote the towns by 1, 2, … , n-1, n and. The distances are $d_{ij}, d_{ii} = 0$.

By greedy methods one obtains a good possible solution. We don't know if this solution is an optimal one.

Step 1. Write the symmetric matrix K of distances. The number of towns are arranged on a column (at the left of K) and a line (above the matrix K).

Step 2. Choose an arbitrary town i to be the beginning of the route. Let $i = 1$ be the first town. Ones marks the town 1 and hence the line 1 by an arrow $\rightarrow$.

Step 3. Compute the minimal value on the marked line. Let us say we have

$\min_{j} d_{1j} = d_{1k}$. This means the route passes from town 1 to the town k. Mark the value d_{1k} by a circle or by a star *.

Step 4. Mark the value d_{k1} by a double star **.

Step 5. Compute the minimal value on the line double star, except value d_{k1}, and denote it also by star.

Ones takes care to not construct a closed sub-cycle This means that the column of this minim shouldn't contain any marked element.

Step 6. Mark the symmetric of this number by double star.

Step 7. Continue on the same way.

Finally, in the symmetric matrix K each line and each column has only one element marked by star and only one element marked by double star.

Step 8. Write the route by the above algorithm.

Step 9. Compute the length L_i of the route.

The algorithm can continue in the same way by repeating it for each town i as a beginning of the route. In this case the following step is necessary.

Step 10. Compute the minim length

$\min\{L_1, L_2, \cdots, L_n\} = L_p$.

The route which begins in the town $p \in \{1,2,\cdots,n\}$ is the best route for greedy method.

Application 1. We apply the greedy algorithm for $n = 6$ towns and the K matrix from table 1. Also, this table contains the computations for a route beginning from town 1.

Table 1.

town	1	2	3	4	5	6
→1	0	*3	10	11	7	**25
2	**3	0	*6	12	8	26
3	10	**6	0	9	*4	20
4	11	12	9	0	*5	**15
5	7	8	**4	*5	0	18
6	*25	26	20	**15	18	0

The route and the length route are

$i = 1$: Route 1={1, 2, 3, 5, 4, 6, 1} , $L_1 = 58$.

The others results are

$i = 2$: Route 2={2, 1, 5, 3, 4, 6, 2}, $L_2 = 64$.

$i = 3$: Route 3={3, 5, 4, 1, 2, 6, 3}, $L_3 = 69$.

$i = 4$: Route 4={4, 5, 3, 2, 1, 6, 4}, $L_4 = 58$.

$i = 5$: Route 5={5, 3, 2, 1, 4, 6, 5}, $L_5 = 57$ (*)

$i = 6$: Route 6={6, 4, 5, 3, 2, 1, 6}, $L_6 = 58$.

Route 1 equivalent Route 4 equivalent Route 6.
The best greedy route is Route 5.

Table 2.

town	1	2	3	4	5	6
1	0	**3	10	*11	7	25
2	*3	0	**6	12	8	26
3	10	*6	0	9	**4	20
4	**11	12	9	0	5	*15
→5	7	8	*4	5	0	**18
6	25	26	20	**15	*18	0

There exist a better route than the Route 5
{1, 2, 3, 6, 4, 5, 1} , $L = 56$ which isn't offered by greedy method.

3 The neural networks and TSP.

The weights matrix and energy function

In this work the solving of TSP problem is based on **neural network method**, which generates a TSP algorithm. We describe the TSP algorithm in a **matrix form**, rather then on components form. The neural network method has its origins in continuous Hopfield networks [4], page 144.

In a Hopfield network the input layer Sx is identical with the output layer Sy.

The neural network for TSP has n^2 neurons (processing elements) in layer Sx. Each neuron has an output function of sigmoid form

$$f : R \to (0;1), f(s) = \frac{1}{1+e^{-2\lambda s}},\ \lambda > 0 .$$

The output function is the same for all n^2 neurons.
The parameter λ is the curve slope. If $\lambda \ge 50$ then the function f is almost the Heaviside function , with the values 0 and 1.

During the algorithm we shall describe we use the columns and the lines of weights matrix V . That is why we use some special notations, as follows

$$V = \left(v_1\ v_2 \cdots v_j \cdots v_n\right) = \left(Vcol_1 \cdots Vcol_j \cdots Vcol_n\right)$$

$$V = \left(Vlin_a\ Vlin_b \cdots Vlin_X \cdots Vlin_r\right)^T ,\ Vcol_j \in R^n$$

$$V = \left(Vlin_1\ Vlin_2 \cdots Vlin_i \cdots Vlin_n\right)^T ,\ Vlin_i \in R^n$$

$$Vcol_j = \left(v_{aj}\ v_{bj} \cdots v_{Xj} \cdots v_{rj}\right)^T$$

$$Vlin_X = \left(v_{X1}\ v_{X2} \cdots v_{Xj} \cdots v_{Xn}\right) .$$

Also we use the sum of elements on line X and on column j and denote

$$VSline_X = \sum_{j=1}^{n} v_{Xj}\ ,\ VScol_j = \sum_{X=1}^{n} v_{Xj}\ .$$

Using the above notations we construct the extended matrix Vex having the form

$$Vex = \begin{pmatrix} v_{a1} \cdots & v_{aj} \cdots & v_{an} & VSlin_a \\ \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots \\ v_{X1} \cdots & v_{Xj} \cdots & v_{Xn} & VSlin_X \\ \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots \\ v_{r1} \cdots & v_{rj} \cdots & v_{rn} & VSlin_r \\ VScol_1 \cdots & VScol_j \cdots & VScol_n & 0 \end{pmatrix} .$$

The mathematical model of TSP problem needs two supplementary weights having the meaning [1]

$$v_{X(n+1)} = v_{X1}, v_{X0} = v_{Xn} \qquad (1)$$

Any admissible route R has an associated matrix V and an **energy function** denoted $E = E(R)$.

Definition. The energy function is defined by four sums, as it follows [4], page 151 ; [6]

$$E(R) = \frac{A}{2}\Sigma_1 + \frac{B}{2}\Sigma_2 + \frac{C}{2}\Sigma_3 + \frac{D}{2}\Sigma_4 \qquad (2)$$

$$\Sigma_1 = \sum_{X=1}^{n}\sum_{i=1}^{n}\sum_{j=1, j\ne i}^{n} v_{Xi} v_{Xj}$$

$$\Sigma_2 = \sum_{j=1}^{n}\sum_{X=1}^{n}\sum_{Y=1, Y\ne X}^{n} v_{Xj} v_{Yj}$$

$$\Sigma_3 = \left(\sum_{X=1}^{n}\sum_{j=1}^{n} v_{Xj} - n\right)^2$$

$$\Sigma_4 = \sum_{X=1}^{n}\sum_{\substack{Y=1 \\ Y\ne X}}^{n}\sum_{j=1}^{n} d_{XY} v_{Xj}\left(v_{Y,j+1} + v_{Y,j-1}\right)$$

A lot of papers and books limit the discussions at this formula and do not show how to use it in a solving algorithm.

Let us investigate the contribution of the four sums in the function energy.

The contribution of the sum Σ_1 will be zero if and only if a single town appears in each row of the V matrix i.e. each town appears only once in the route. So a single $v_{Xj} = 1$ and all other $v_{Xj} = 0$.

The contribution of the sum Σ_2 is zero if and only if each column of the V matrix contains a single value of 1. This means that each position on a route has an unique town associated with it.

The third sum Σ_3 contains a simple summation of all n^2 elements. There should be only n of these terms that have a value of 1; all other elements should be zero. Then

$$\sum_{X=1}^{n}\sum_{j=1}^{n} v_{Xj} = n .$$

If more or less than n terms are equal to 1, then the contribution of this sum will be greater than zero.

We remark that the first three sums do not depend of the distances between towns.

The sum Σ_4 contains the distances d_{XY} . This sum computes a numerical value proportional to the distance traveled on the route. Thus, a minimum distance route results in a minimum contribution of this term to the energy function (2).

Work [4] mentions that the weights matrix is defined in terms of inhibitions between the n^2 processing elements of the attached neural network. The desire to minimize the sum Σ_4 can be exposed into connections between units that inhibit the selection of adjacent towns in proportion to the distance between those towns.

Proposition 1 . The four sums from the energy function are represented in the following vector form

$$\Sigma_1 = 2 \sum_{1 \le k < j \le n} < v_k ; v_j >$$

$$\Sigma_2 = 2 \sum_{1 \le i < k \le n} < Vlin_i ; Vlin_k >$$

$$\Sigma_3 = \left(\sum_{X=1}^{n} VSlin_X - n \right)^2$$

$$\Sigma_4 = 2d_{ab}[v_{a1}(VSlin_b - v_{b1}) + v_{a2}(VSlin_b - v_{b2}) + \cdots + v_{an}(VSlin_b - v_{bn})] + $$
$$+ 2d_{ac}[v_{a1}(VSlin_c - v_{c1}) + v_{a2}(VSlin_c - v_{c2}) + \cdots + v_{an}(VSlin_c - v_{cn})] + \cdots ,$$

where the last sum is extended for all distances in the upper superior triangular positions , i.e.

$d_{XY} = d_{ik}, 1 \le i < k \le n$.

The notation $< u ; v >$ means the scalar product

$< u; v >= u^T v, u \in R^n, v \in R^n$.

Proof . One uses the definitions of sums $\Sigma_1, \Sigma_2, \Sigma_3, \Sigma_4$ with a convenient association of the weights v_{Xj} (End).

4 The relation between continuous Hopfield model and TSP problem

The Hopfield network with n^2 processing elements, attached to TSP problem proceeds from the continuous Hopfield model [4], page 144. The continuous model is described by two differential equations (with independent notations [1])

$$p \frac{du_i}{dt} = \sum_{j=1}^{n} w_{ij} v_j - \frac{u_i}{R_i} + I_i \qquad (3)$$

$$\frac{dE}{dt} = -\sum_{i=1}^{n} p \frac{df^{-1}(v_i)}{dv_i} \left(\frac{dv_i}{dt} \right)^2 \qquad (4)$$

where $v_i = f(u_i), u_i = f^{-1}(v_i), i = 1, n$.

Two things are very important in the future: the time delay $-u_i / R_i$ from equation (3) and $E = E(v)$ from (4).

The variables u_i from continuous model [4] become $u_{Xj}, X = 1, n; j = 1, n$ in TSP problem.

We denote $U = U_{n \times n}, U = (u_{Xj})$, where u_{Xj} are input variables. Then we compute the weights

$$v_{Xj} = f(u_{Xj}) , \; v_{Xj} = \left(1 / \left(1 + e^{-2\lambda u_{Xj}} \right) \right)$$

$$V = V_{n \times n}, V = (v_{Xj}) \qquad (5)$$

According to the general techniques of neural networks, the variables u_{Xj} are updated when the algorithm passes from time t (the route t) to time $t+1$ (the route $t+1$). The updating is done by a recurrent relation which has two equivalent forms: a component form or a matrix form, respectively

$$u_{Xj}(t+1) = u_{Xj}(t) + \Delta u_{Xj}(t) \qquad (6)$$

$$U(t+1) = U(t) + \Delta U(t) \; , \; \Delta U(t) = (\Delta u_{Xj}(t)) \qquad (7)$$

Now the main question is to find the appropriate form of corrections $\Delta u_{Xj}(t)$. Again we use the general neural networks theory: the corrections are defined by **descent gradient** of energy function. So we have the following dependences:

$E = E(R), E = E(v) \; , \; v = f(u), E = E(u)$.

The derivative are positive, namely $\frac{dE}{dv} > 0, \frac{dE}{du} > 0$.

Due to time delay from (3) and the descent gradient, we define the corrections by the relation

$$\Delta u_{Xj}(t) = -\frac{1}{\tau} u_{Xj}(t) - \frac{dE}{dv_{Xj}} < 0 \qquad (8)$$

The $\tau > 0$ is a parameter controlled by the user.

5 The explicit correction form and new matrix notations

The formula (8) and $E = E(v)$ give the following corrections

$$\Delta u_{Xj}(t) = [-\frac{1}{\tau} u_{Xj}(t) - A \sum_{k=1; k \ne j}^{n} v_{Xj}(t) -$$

$$-B\sum_{Y=1;Y\neq X}^{n} v_{Yj}(t)-C\left(\sum_{Y=1}^{n}\sum_{k=1}^{n} v_{Yk}(t)-n'\right)-$$
$$-D\sum_{Y=1}^{n} d_{XY}\left(v_{Y,j+1}+v_{Y,j-1}\right)]\Delta t \qquad (9)$$

where appear some parameters for user's disposal

$\tau\in(0;1)$,$\Delta t\in(0;1)$,$n'\geq 0$,$n<n'\leq 1.5n$

In order to compute the laborious formula (9) we use new notations, as it follows

$$S\left(Vlin_X;k\neq j\right)=VSlin_X-v_{Xj}(t)$$
$$S\left(Vcol_j;Y\neq X\right)=VScol_j-v_{Xj}(t)$$
$$S\left(V;n'\right)=\sum_{Y=1}^{n}\sum_{k=1}^{n}v_{Yk}(t)-n'$$
$$S\left(Klin_X;j\right)=\sum_{Y=1}^{n}d_{XY}\left(v_{Y,j+1}+v_{Y,j-1}\right) \quad (10)$$

(the meanings of the letters are: S is the sum in the matrix V or K etc.) .

Proposition 2 . The corrections (9) take the form

$$\Delta u_{Xj}(t)=\{-\frac{1}{\tau}u_{Xj}(t)-A\left[VSlin_X-v_{Xj}(t)\right]-$$
$$-B\left[VScol_j-v_{Xj}(t)\right]-C\left[S\left(V;n'\right)\right]-$$
$$-D\left[S\left(Klin_X;j\right)\right]\}\,\Delta t \quad (11)$$

Proof . One uses the notations (10). (End).

The formula (10) determine us to introduce the following matrix

$$\widetilde{V}=\widetilde{V}_{n\times n},\widetilde{V}=\left(v_{Y,j+1}+v_{Y,j-1}\right) \qquad (12)$$

Proposition 3 . All the sums from (10) create a new matrix (as a product)

$$K\widetilde{V}=\left(S\left(Klin_X;j\right)\right) \qquad (13)$$

Proof . One uses (1) and the direct computation. (End).

We can write the elements $\Delta u_{Xj}(t)$ from (11) or equivalent the matrix $\Delta U(t)$ from (7) if we introduce the matrices (denoted by a succession of two or three letters)

$VS=VS_{n\times n}$,$VSL=VSL_{n\times n}$,$VSC=VSC_{n\times n}$.

Explicitly, for n=4, the above matrices have the forms

$$VS=\begin{pmatrix} S(V;n') & S(V;n') & S(V;n') & S(V;n') \\ S(V;n') & S(V;n') & S(V;n') & S(v;n') \\ S(V;n') & S(V;n') & S(V;n') & S(V;n') \\ S(V;n') & S(V;n') & S(V;n') & S(V;n') \end{pmatrix}$$

$$VSL=\begin{pmatrix} VSlin_a & VSlin_a & VSlin_a & VSlin_a \\ VSlin_b & VSlin_b & VSlin_b & VSlin_b \\ VSlin_c & VSlin_c & VSlin_c & Vslin_c \\ VSlin_d & VSlin_d & Vslin_d & VSlin_d \end{pmatrix}$$

$$VSC=\begin{pmatrix} VScol_1 & VScol_2 & VScol_3 & VScol_4 \\ VScol_1 & VScol_2 & VScol_3 & VScol_4 \\ VScol_1 & VScol_2 & VScol_3 & VScol_4 \\ VScol_1 & VScol_2 & VScol_3 & VScol_4 \end{pmatrix}$$

Proposition 4 . The corrections (11) from the proposition 2 have the **matrix form**

$$U(t)=U(t)_{n\times n}\ ,\ \Delta U(t)=\Delta U(t)_{n\times n}$$
$$\Delta U(t)=\{-\frac{1}{\tau}U(t)-A\left[VSL-V(t)\right]-B\left[VSC-V(t)\right]-$$
$$-C\left(VS\right)-D(K\widetilde{V})\}\,\Delta t \qquad (14)$$

Proof . We use (11) and the special matrices VS, VSL and VSC. (End).

The updating recurrent relations (6) or the equivalent matrix form (7) work if we know the initial values $u_{Xj}^{0}=u_{Xj}(1)$ or the initial matrix $U^{0}=U(1)$ for first route.

6 The TSP algorithm in matrix form

Having all the above notations, formulas and ideas we can describe the TSP algorithm. We choose to describe this algorithm in matrix form.

Step 1. We introduce the input data :

a). n - number of towns; $K=\left(d_{XY}\right)$; N - number of algorithm iterations.

b). general parameters $n',\lambda,\tau,\Delta t$;

c). inhibitions parameters A,B,C,D .

d). output function $f(s)=\dfrac{1}{1+e^{-2\lambda s}}$.

e). initial values $U^{0}=\left(u_{Xj}^{0}\right)$, $X=1,n;j=1,n$.

f). we declare the dimensions for all matrices :

$K,U,V,\widetilde{V},Vex$ and so on.

Step 2. We execute the computations in a DO loop as it follows

```
L0 CONTINUE
    DO L3 t=1, N
    * compute the sigmoid outputs and create
```

```
        the matrix $V = (v_{Xj}(t))$
        DO  L2  X=1,n
        DO  L1  j=1,n
        $v_{Xj}(t) = f[u_{Xj}(t)]$
    L1  CONTINUE
    L2  CONTINUE
        * compute the sums $\Sigma_1, \Sigma_2, \Sigma_3$ from
        proposition 1
        $\Sigma_1 = 2 \sum_{1 \le k < j \le n} < v_k(t); v_j(t) >$
        $\Sigma_2 = 2 \sum_{1 \le i < k \le n} < Vlin_i(t); Vlin_k(t) >$
        $\Sigma_3 = \left( \sum_{Y=1}^{n} \sum_{k=1}^{n} v_{Yk}(t) - n \right)^2$
        * compute the extended matrix $Vex(t)$
        * using $K, V(t), Vex(t)$ we compute the sum
        $\Sigma_4$ from proposition 1.
        * compute the energy function
        $E(t) = \frac{A}{2}\Sigma_1 + \frac{B}{2}\Sigma_2 + \frac{C}{2}\Sigma_3 + \frac{D}{2}\Sigma_4$
        * optional: print the values $t, V(t), E(t)$ .
        * compute the following matrices at time t
        $VSL = (VSlin_X(t)), \quad X = 1,n$
        $VSC = (VScol_j(t)), \quad j = 1,n$
        $\widetilde{V} = (v_{X,j+1}(t) + v_{X,j-1}(t))$, for
        all $X = 1,n; j = 1,n$
        $K\widetilde{V}$
        * compute the correction matrix $\Delta U(t)$ by
        using the formula (14) from proposition 4
        * update the input matrix $U$ by the recurrent
        equation
        $U(t+1) = U(t) + \Delta U(t)$
    L3  CONTINUE  (the DO  loop until t=N ) .
```

Step 3. Verify if the closed Do loop generates an admissible TSP solution, by the matrix

$$V(N) = (v_{Xj}(N)).$$

There are several possibilities (versions)

Version 1. The matrix $V(N)$ generates an admissible TSP solution. Then GO TO label L4.

Version 2. The matrix $V(N)$ do not generate an admissible solution because

$v_{Xj}(N) \notin \{0;1\}$ i.e. $v_{Xj} \in (0;1)$.

Then we replace N by $N', N' > N$ and GO TO label L0 and resume the cycle DO loop.

Version 3. One uses the matrix $V(N)$ and compute the maxim element on each line X, $X = 1,n$. We denote it by $v_{Xj^*}(N) = 0$. If $v_{Xj^*}(N) \in (\varepsilon;1), \varepsilon > 0.8$ (for example) then we set $v_{Xj^*} = 1$ and all the other elements $v_{Xj}(N) = 0, j \ne j^*$ on the line X. (winner-take-all). The resulting matrix is denoted $V_l^*(N)$, where l means the work on lines. Analogous we can compute the maxim element $v_{X^*j}(N)$ on each column $j = 1,n$. So we obtain the matrix $V_c^*(N)$, where the letter c means the work on columns.

Compute the routes described by matrices $V_l^*(N)$, $V_c^*(N)$ and take the best one. GO TO L4.

```
    L4  CONTINUE
    Step 4. Print the final results :
    $N$, $V_l^*(N)$ or $V_c^*(N)$ , $E(N)$ and the route $R^*$ .
        STOP
        END
```

7 Application 2

Let n be with the value $n = 4$ and the distances between the towns a, b, c, d given by the matrix

$$K = \begin{pmatrix} 0 & 7 & 3 & 2 \\ 7 & 0 & 2 & 5 \\ 3 & 2 & 0 & 1 \\ 2 & 5 & 1 & 0 \end{pmatrix}$$

. Apply the above algorithm to find the best route.

Solution. We use the parameters

$n = 4, n' = 5, \lambda = 10, \tau = 1, \Delta t - 0.01, N = 10$

$A = 10, B = 10, C = 4, D = 10$, $f(s) = \frac{1}{1 + e^{-20s}}$

The initial inputs are

$$U^0 = U(1) = \begin{pmatrix} 1 & 2 & 0 & 1 \\ 2 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \\ 1 & 1 & 2 & 0 \end{pmatrix}$$

. For $t = 1$ we obtain

$$V(1)=\begin{pmatrix} 0.999 & 1.000 & 0.500 & 0.999 \\ 1.000 & 0.500 & 0.999 & 0.999 \\ 0.500 & 0.999 & 0.999 & 1.000 \\ 0.999 & 0.999 & 1.000 & 0.500 \end{pmatrix}$$

$\Sigma_1 = 35.952$, $\Sigma_2 = 35.952$, $\Sigma_3 = 0.252$
$\Sigma_4 = 369.56$; $E(1) = 22707.84$ and so on.

$S(V;n') = 13.992 - 5 = 8.992$.
Construct the matrix VS .
Use the formula
$\widetilde{V} = \left(v_{Y,j+1} + v_{Y,j-1}\right), v_{Y5} = v_{Y1}, v_{Y0} = v_{Y4}$
for $j = 1,4$ and one obtains the matrix

$$\widetilde{V}=\begin{pmatrix} 1.999 & 1.499 & 1.999 & 1.499 \\ 1.499 & 1.999 & 1.499 & 1.999 \\ 1.999 & 1.499 & 1.999 & 1.4999 \\ 1.499 & 1.999 & 1.499 & 1.999 \end{pmatrix}.$$

The matrix $K\widetilde{V}$ has the elements

$$K\widetilde{V}=\begin{pmatrix} 19.488 & 22.488 & 19.488 & 22.488 \\ 25.486 & 23.486 & 25.486 & 23.486 \\ 10.949 & 10.494 & 10.494 & 10.494 \\ 13.492 & 14.492 & 13.492 & 14.492 \end{pmatrix}$$

$$\Delta U(1)=\begin{pmatrix} -2.818 & -3.128 & -2.908 & -3.118 \\ -3.428 & -3.308 & -3.418 & -3.218 \\ -2.009 & -1.919 & -1.919 & -1.928 \\ -1.219 & -2.319 & -2.228 & -2.408 \end{pmatrix}$$

$U(2) = U(1) + \Delta U(1)$

$$U(2)=\begin{pmatrix} -1.818 & -1.128 & -2.908 & -2.118 \\ -1.428 & -3.308 & -2.418 & -2.218 \\ -2.009 & -0.918 & -0.919 & 0.071 \\ -1.219 & -1.319 & -0.288 & -2.408 \end{pmatrix}$$

$t = 2$. The matrix $V(2)$ has the elements

$$v_{Xj}(2) = \frac{1}{1+e^{-2\lambda u_{Xj}(2)}} \ , \ \lambda = 10$$

$X = 1,4;\ j = 1,4$.

$$V(2)=\begin{pmatrix} 0.0 & 1.59\cdot 10^{-10} & 0.0 & 0.0 \\ 3.95\cdot 10^{-3} & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.04\cdot 10^{-8} & 1.04\cdot 10^{-8} & 0.0 \\ 2.58\cdot 10^{-11} & 3.49\cdot 10^{-12} & 0.01 & 0.0 \end{pmatrix}$$

$v_{Xj}(2) \notin \{0;1\}$. The algorithm must continue.

References

[1]. APPLEGATE L. David, BIXBY E. Robert, CHVATAL Vasek, COOK J. William, *TSP cuts which do not conform to template paradigm*, Computational Combinatorial Optimization (M. Junger, D. Naddef, editors), Springer, 2001.

[2]. APPLEGATE L. David, BIXBY E. Robert, CHVATAL Vasek, COOK J. William, *The Traveling Salesman Problem: A Computational Study*, Princeton Universitary Press, 2006.

[3]. CHRISTOF T. , REINELT G., *Parallel Cutting Plane Generation for the TSP*. In Parallel Programming and Applications (R. Fritszon, L. Finino, editors), IOS Press, page 163-169, 1995.

[4]. FREEMAN A. James, SKAPURA M. David, *Neural Networks*: *Algorithms, Applications and Programming Techniques*, Addison-Wesley Publishing Company, 1991.

[5]. JAIN K. Anil, MAO Jianchang, MOHIUDDIN K. M. , *Artificial Neural Networks: A Tutorial*, IEEE, March, 1996.

[6]. KRÖSE Ben, Van der SMAGT Patrick, *An Introduction to Neural Networks* , Chapter 5, University of Amsterdam, Eighth Edition, November 1996.

[7]. PADBERG M. W., Rinaldi G. , *A Branch-and-Cut* Algorithm *for the Resolution of Large-Scale Symmetric TSP*, SIAM Review 33, page 60-100, 1991.

[8]. POPOVICIU Nicolae, BONCUT Mioara, *A Complete Sequential Learning Algorithm for RBF Neural Networks with Applications* , **WSEAS** Transactions on Systems, Issue 1, Volume 6, January 2007, pages 24-32.

[9]. SYED SAAD ALZHAR A. , *RBF Neural Networks Based Self-Tuning Adaptive Regulator*, **WSEAS** Transactions on Systems, Issue 9, Volume 3, Nov. 2004.