Traveling Salesperson Problem and Neural Networks. A Complete Algorithm in Matrix Form

NICOLAE POPOVICIU
Faculty of Mathematics-Informatics
Department of Mathematics
Hyperion University of Bucharest
Str. Călăraşilor 169, Bucharest
ROMANIA

MIOARA BONCUȚ
Faculty of Sciences
Department of Mathematics
Lucian Blaga University of Sibiu
Victoriei Avenue 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. By using [1] as a main reference, we formulate an algorithm in a matrix form to solve the problem. 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 computation program. The work has six sections. The section 5 describes the algorithm to solve the traveling salesperson problem and the section 6 contain an numerical example.

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

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 [1].

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. The solving of TSP problem is based on neural network method, which generates a TSP algorithm. In this work 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 [1], 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 $\lambda \geq 50$ then the function f is almost the Heaviside function , with the values 0 and 1.

2 The weights matrix and energy function

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

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

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

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

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

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

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 [1], page 151 ; [3]

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

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

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

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

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

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

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

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

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

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

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

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

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

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).

3 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 [1], 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 [1] become

u_{Xj}, $X = 1,n; j = 1,n$ in TSP problem.

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

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

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

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

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

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

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.

4 The explicit correction form and new matrix notations

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

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

$$- B \sum_{Y=1;Y\neq X}^{n} v_{Yj}(t) - C\left(\sum_{Y=1}^{n}\sum_{k=1}^{n} v_{Yk}(t) - n'\right) -$$

$$- D \sum_{Y=1}^{n} d_{XY}\left(v_{Y,j+1} + v_{Y,j-1}\right)]\Delta t \qquad (9)$$

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

$$S\left(Vlin_X; k \neq j\right) = VSlin_X - v_{Xj}(t)$$

$$S\left(Vcol_j; Y \neq X\right) = VScol_j - v_{Xj}(t)$$

$$S\left(V; n'\right) = \sum_{Y=1}^{n}\sum_{k=1}^{n} v_{Yk}(t) - n'$$

$$S\left(Klin_X; j\right) = \sum_{Y=1}^{n} d_{XY}\left(v_{Y,j+1} + v_{Y,j-1}\right) \qquad (10)$$

(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

$$\Delta u_{Xj}(t) = \{-\frac{1}{\tau} u_{Xj}(t) - A\left[VSlin_X - v_{Xj}(t)\right] -$$

$$- B\left[VScol_j - v_{Xj}(t)\right] - C\left[S\left(V; n'\right)\right] -$$

$$- D\left[S\left(Klin_X; j\right)\right]\}\Delta t \qquad (11)$$

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**

$U(t) = U(t)_{n\times n}$, $\Delta U(t) = \Delta U(t)_{n\times n}$

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

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 $u^0_{X\,j} = u_{X\,j}(1)$ or the initial matrix $U^0 = U(1)$ for first route.

5 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 $U^0 = \left(u^0_{X\,j}\right)$, $X = 1,n; j = 1,n$.

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 $V = \left(v_{X\,j}(t)\right)$

DO L2 X=1,n

DO L1 j=1,n

$v_{X\,j}(t) = f\left[u_{X\,j}(t)\right]$

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 Σ_4 from proposition 1.

* compute the energy function

$E(t) = \dfrac{A}{2}\Sigma_1 + \dfrac{B}{2}\Sigma_2 + \dfrac{C}{2}\Sigma_3 + \dfrac{D}{2}\Sigma_4$

* optional: print the values $t, V(t), E(t)$.

* compute the following matrices at time t

$VSL = \left(VSlin_X(t)\right), \quad X = 1,n$

$VSC = \left(VScol_j(t)\right), \quad j = 1,n$

$\widetilde{V} = \left(v_{X,j+1}(t) + v_{X,j-1}(t)\right)$, 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

$V(N) = \left(v_{X\,j}(N)\right)$.

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

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

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 $v_{X\,j^*}(N) = 0$. If $v_{X\,j^*}(N) \in (\varepsilon;1), \varepsilon > 0.8$ (for example) then we set $v_{X\,j^*} = 1$ and all the other elements $v_{X\,j}(N) = 0,\ j \neq j^*$ 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

6 Application

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

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

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

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.

References

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

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

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

[4]. 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.

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