Report 2

Paweł Matykiewicz

University of Adam Mickiewicz in Poznan, Faculty of Social Sciences, Department of Philosophy

Abstract

In this report we try to achieve similar results as in [2]. We use discrete output function and compare it with results in [2]. TSP with transient chaotic dynamic is implemented in **Delphi 5.0**.

Key words: , neural networks, chaotic dynamics, transient chaos, TSP

1 Introduction

In [3] was constructed chaotic associative memory using Nagumo-Sato neuron model with continuous output function. They found that the algorithm has very good searching properties. On the base of this model one can build controlled transient chaotic neural network [2] and use it for a searching process in the Travelling Salesperson Problem. In *Report I* I have found that the same neural network but with the discrete output function has also good searching properties. In this report I try to compare searching with all-or-nothing (heaviside) function to one in [2].

2 Model

A classical combinatorial optimization problem which is *NP*-hard is the travelling salesperson problem (TSP). We denote d_{ij} to be distance from city i to city j. We assume that $d_{ij} = d_{ji}$, $d_{ii} = 0$ which is called symmetric TSP. We have $(n-1)!/2$ possible tours. We will see how two algorithms cope with that problem. We take d_{ij} similar to Hopfield-Tank original data:

Email address: pm@felix.fizyka.amu.edu.pl (Paweł Matykiewicz).

```
d[0,1]:=2.0/6;
d[1,2]:=0.7/6;
d[2,3]:=3.2/6;
d[3,0]:=2.2/6;
d[3,1]:=3.7/6;
d[0,2]:=1.9/6;
d[1,0]:=2.0/6;
d[2,1]:=0.7/6;
d[3,2]:=3.2/6;
d[0,3]:=2.2/6;
d[1,3]:=3.7/6;
d[2,0]:=1.9/6;
```

We take $n = 4$ (number of cities) in every simulation. We also use function E which measures energy (cost) needed for travel route x_{ij}, where i is city and j is visiting order.

$$E(X) = \frac{W_1}{2}\Big\{\sum_{i=1}^{n}(\sum_{j=1}^{n} x_{ij} - 1)^2 + \sum_{j=1}^{n}(\sum_{i=1}^{n} x_{ij} - 1)^2\Big\} + \frac{W_2}{2}\Big\{\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} x_{ij} d_{ik}(x_{kj+1} + x_{kj-1})\Big\} \tag{1}$$

Minimize function E means that we visit only once every city, not more than one city every day and the tour length (cost) should be small as possible.

We use two kinds of output functions continuous 2 and discrete 3. So we have two kinds of algorithms to solve TSP one we call Chaotic Simulated Annealing (CSA) and the other Devil‘s Staircase Simulated Annealing (DSSA). Both of them can be classified as Transient-Chaotic Neural Networks (TCNN).

$$f^C(u) = \frac{1}{1 + \exp(\frac{-u}{\varepsilon})} \tag{2}$$

$$f^D(u) = \begin{cases} 1 & \text{if} \quad u > 0 \\ 0 & \text{if} \quad u \leq 0 \end{cases} \tag{3}$$

We have respectively $E((f^C(y_{ij}(t))) \equiv E^C$ and $E((f^D(y_{ij}(t))) \equiv E^D$ functions where $(f^{(\star)}(y_{ij}))$ is $(i \times j)$ matrix of outputs.

2.1 Network Model

We use Nagumo-Sato neuron model in fully connected recurrent neural network. Instead of $\sum_{j=1}^{n} w_{iklj} x_{kj}(t) + \theta_{ij}$ we put $-\frac{\partial E}{\partial x_{ij}(t)}$. We assume that $\theta_{ij} = W_1$. Now we can rewrite N-S neuron model as:

$$\begin{aligned} y_{ij}(t+1) &= k y_{ij}(t) - z(t)(x_{ij}(t) - I_0) + \\ &+ \alpha \Big\{ - W_1 \Big(\sum_{l \neq j}^{n} x_{il}(t) + \sum_{k \neq i}^{n} x_{kj}(t) \Big) - \\ &- W_2 \Big(\sum_{k \neq i}^{n} d_{ik} xkj + 1(t) + \sum_{k \neq i}^{n} d_{ik} xkj - 1(t) \Big) + W_1 \Big\} \end{aligned} \tag{4}$$

k is dumping factor of membrane, α is a positive scaling parameter for inputs, z(t) is refractory parameter and is responsible for transient-chaotic dynamic:

$$z(t+1) = (1 - \beta) z(t) \tag{5}$$

Varying $1 - \beta$ and α means varying influence of chaos (searching properties) and finding right solution. Balance between β and α is when searching process will be long enough and influence of feedback connections will be strong enough to drive system into minimal energy (even global minimum).

Neurons are updated cyclically. When all neurons are updated we count one iteration and and so on.

To improve solution we can change values from the open set $(0, 1)$ to 0, 1 values:

$$x_{ij}^{CD} = \begin{cases} 1 & \text{if} \quad x_{ij}^{C} > \sum_{k=1}^{n} \sum l = 1^{n} x_{kl} / n^2 \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

If we put $E(X^{CD}(t))$ then we will call this function E^{CD} where $X^{CD}(t) = (x_{ij}^{CD}(t))$ is a matrix. In CSA algorithm for TSP it is enough to code $X^{CD}(t_{end})$ at the end of calculation.

We can calculate number of iterations needed to put system in desired value $z(t)$:

$$p(t_{end}) = \log_{1-\beta}(\frac{z(t_{end})}{z_0}) \tag{7}$$

If we round the p value we will have an integer number of needed iterations. We assume that $p = 1800$ or $z(t_{end}) = 0.02$ which is enough and after that we do not observe chaos.

3 Implementation

Here is listing for `ctsp.exe` main procedures and functions in **Delphi 5.0**:

```
function fc(u,aepsilon:real):real;

begin
result:=1/(1+exp(-u/aepsilon));
end;


procedure stepc
(ak,aalpha,ai0,aw1,aw2,aepsilon,abeta:real;an:integer;ad:matrix;
var az:real; var ay:matrix;var ax:matrix);

var i,j:integer;

begin
for
i:=0 to an-1 do
 for j:=0 to an-1 do
  begin
   ay[i,j]:=ak*ay[i,j]-az*(ax[i,j]-ai0)
   +aalpha*(dedx(aw1,aw2,an,i,j,ax,ad));
   ax[i,j]:=fc(ay[i,j],aepsilon);
  end;
az:=(1-abeta)*az;
end;

function fd(u:real):real;

begin
if u>0 then result:=1 else result:=0
end;


procedure stepd
(ak,aalpha,ai0,aw1,aw2,aepsilon,abeta:real;an:integer;ad:matrix;
var az:real;var ay:matrix;var ax:matrix);
```

```
var i,j:integer;

begin
for i:=0 to an-1 do
 for j:=0 to an-1 do
  begin
   ay[i,j]:=ak*ay[i,j]-az*(ax[i,j]-ai0)
   +aalpha*(dedx(aw1,aw2,an,i,j,ax,ad));
   ax[i,j]:=fd(ay[i,j]);
  end;
az:=(1-abeta)*az;
end;
```

Here are $-\frac{\partial E}{\partial x_{ij}(t)}$ and E functions:

```
function dedx(aw1,aw2:real;an,ai,aj:integer;ax,ad:matrix):real;

var k,l:integer;
    sum1,sum2,sum3,sum4:real;
    fx:matrix;

begin
Setlength(fx,an);
for k:=0 to an-1 do
 Setlength(fx[k],an+2);
for k:=0 to an-1 do
 begin
 fx[k,0]:=ax[k,an-1];
 fx[k,an+1]:=ax[k,0];
 end;
for k:=0 to an-1 do
 for l:=0 to an-1 do
  fx[k,l+1]:=ax[k,l];
sum1:=0;
for l:=0 to an-1 do
if l=aj then sum1:=sum1 else
 sum1:=sum1+ax[ai,l];
sum2:=0;
for k:=0 to an-1 do
 if k=ai then sum2:=sum2 else
 sum2:=sum2+ax[k,aj];
sum3:=0;
for k:=0 to an-1 do
 if k=ai then sum3:=sum3 else
```

```
  sum3:=sum3+ad[ai,k]*fx[k,aj+2];
sum4:=0;
for k:=0 to an-1 do
 if k=ai then sum4:=sum4 else
  sum4:=sum4+ad[ai,k]*fx[k,aj];
result:=-aw1*(sum1+sum2)-aw2*(sum3+sum4)+W_1;
end;


function energy(aw1,aw2:real;an:integer;ax,ad:matrix):real;

var i,j,k:integer;
    sum1,sum2,sum3,sum4:real;
    fx:matrix;

begin
Setlength(fx,an);
for i:=0 to an-1 do
 Setlength(fx[i],an+2);
for i:=0 to an-1 do
 begin
 fx[i,0]:=ax[i,an-1];
 fx[i,an+1]:=ax[i,0];
 end;
for i:=0 to an-1 do
 for j:=0 to an-1 do
  fx[i,j+1]:=ax[i,j];
sum1:=0;
for i:=0 to an-1 do
 begin
  sum2:=0;
  for j:=0 to an-1 do
   sum2:=sum2+ax[i,j];
  sum1:=sum1+sqr(sum2-1);
 end;
sum3:=0;
for j:=0 to an-1 do
 begin
  sum2:=0;
  for i:=0 to an-1 do
   sum2:=sum2+ax[i,j];
  sum3:=sum3+sqr(sum2-1);
 end;
sum4:=0;
for i:=0 to an-1 do
```

```
 for j:=0 to an-1 do
  for k:=0 to an-1 do
     sum4:=sum4+(fx[k,j+2]+fx[k,j])*ad[i,k]*ax[i,j];
result:=(aw1/2)*(sum1+sum3)+(aw2/2)*sum4;
end;
```

4 Simulation results

In all simulations we have:

$k = 0.9, \quad \varepsilon = 0.004, \quad I_0 = 0.65, \quad z(0) = 0.08.$

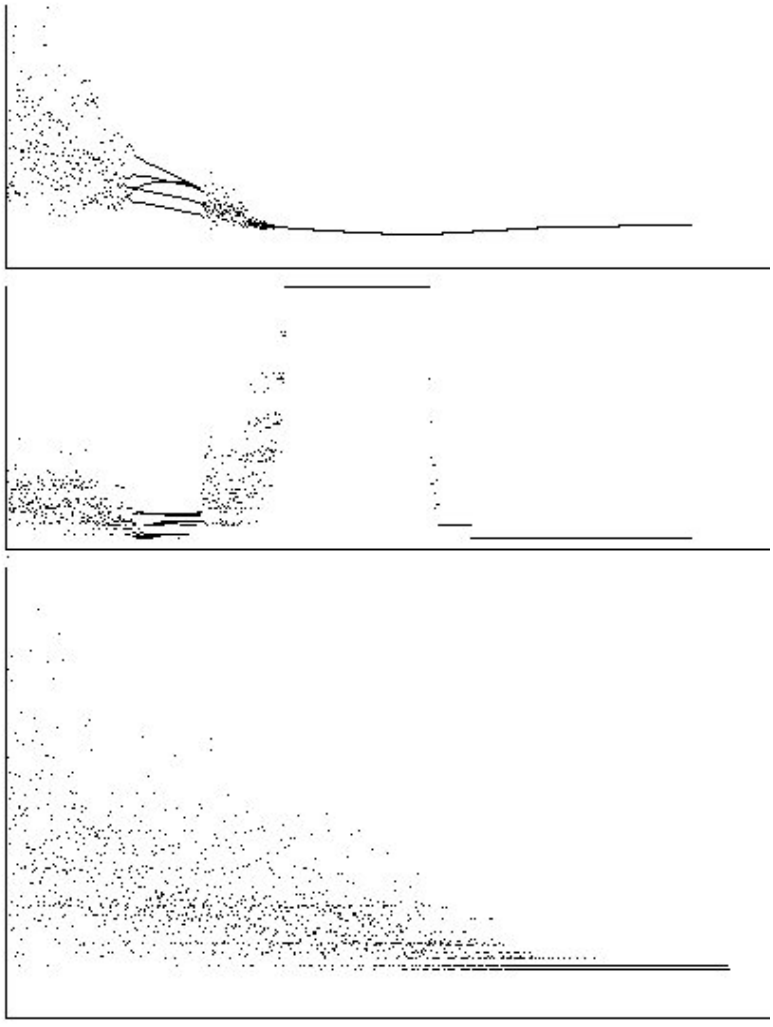

Fig. 1. Time evolutions of E^C, E^{CD}, E^D functions in simulation of TCNN for TSP with 4 cities: $\beta = 0.001$, $\alpha = 0.015$, $W_1 = 1$, $W_2 = 1$

If W_2 is to close to W_1 then in DSSA we can get two (or more) cyclical solutions. We can try to take $W_2 = 0.5$ or use DSSA+CSA algorithm.

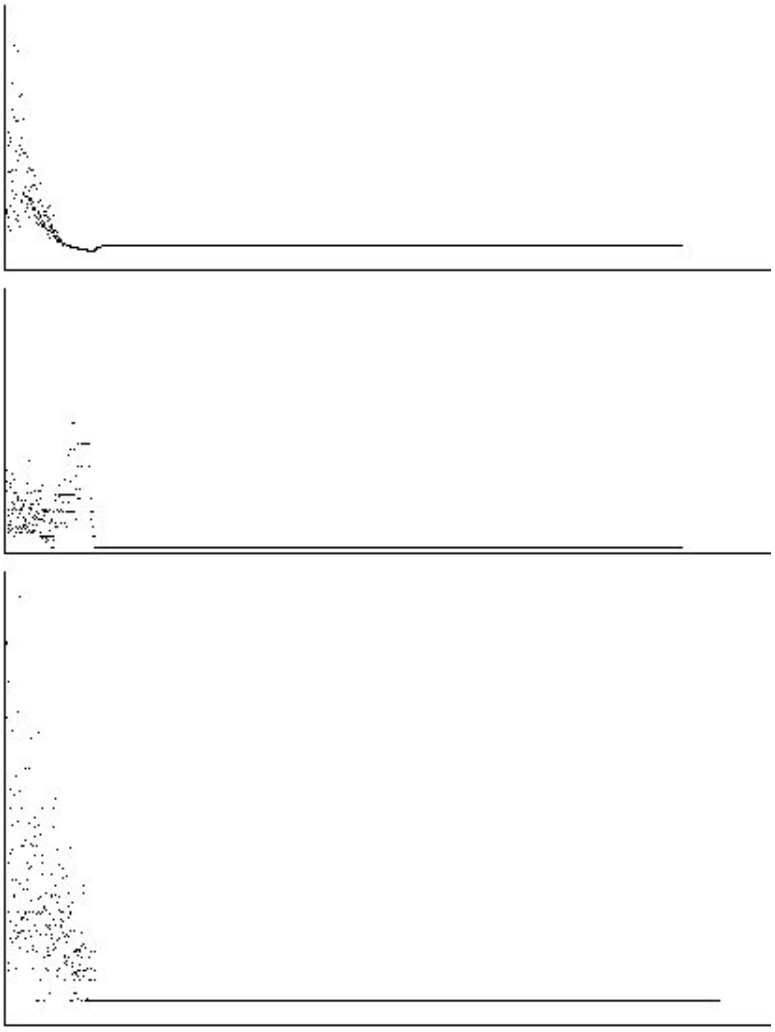

Fig. 2. Time evolutions of E^C, E^{CD}, E^D functions in simulation of TCNN for TSP with 4 cities: $\beta = 0.015$, $\alpha = 0.015$, $W_1 = 1$, $W_2 = 1$

Here we put $W_1 = 1$, $W_2 = 0.5$ and $z_{end} = 0.02$.

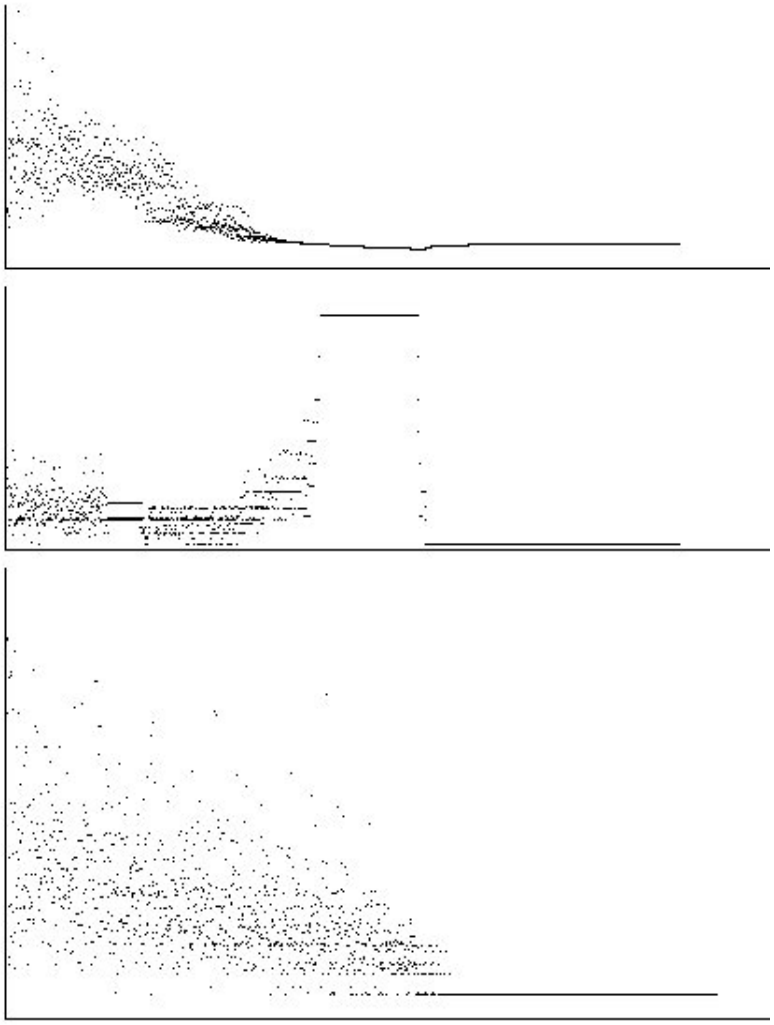

Fig. 3. Time evolutions of E^C, E^{CD}, E^D functions in simulation of TCNN for TSP with 4 cities: $\beta = 0.001$, $\alpha = 0.015$, $W_1 = 1$, $W_2 = 0.5$

740	732	752	742	694	698	763	863	934	989	1000	1000	1000	1000	1000	success
0,015	0,014	0,013	0,012	0,011	0,01	0,009	0,008	0,007	0,006	0,005	0,004	0,003	0,002	0,001	beta
0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	z0
93	99	107	116	126	139	154	174	198	231	278	347	462	693	1387	Iterat.

Fig. 4. Result of 1000 different initial conditions for each value of β on the 4 city TSP with f^C and f^{CD} functions: $t_{end} := 0.02$(CSA)

We also tried to see what give mixed algorithm DSSA+CSA. Varying ap iterations for DSSA and $(1-a)p$ iterations for CSA ($a \epsilon [0,1]$) we can get better and even worse results then in 100% DSSA algorithm.

737	740	772	799	786	828	849	843	860	902	902	953	977	979	1000	success
0,015	0,014	0,013	0,012	0,011	0,01	0,009	0,008	0,007	0,006	0,005	0,004	0,003	0,002	0,001	beta
0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	z0
93	99	107	116	126	139	154	174	198	231	278	347	462	693	1387	Iterat.

Fig. 5. Result of 1000 different initial conditions for each value of β on the 4 city TSP with f^D function: $t_{end} := 0.02$(DSSA)

616	647	634	636	641	646	635	650	672	677	704	638	698	743	928	success
0,015	0,014	0,013	0,012	0,011	0,01	0,009	0,008	0,007	0,006	0,005	0,004	0,003	0,002	0,001	beta
0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	z0
93	99	107	116	126	139	154	174	198	231	278	347	462	693	1387	Iterat.

Fig. 6. Result of 1000 different initial conditions for each value of β on the 4 city TSP with 75% of iterations with f^D function and 25% of iterations with f^C and f^{CD} functions: $t_{end} := 0.02$(DSSA+CSA)

5 Conclusion and discussion

We see that DSSA not much worse then CSA. The computational time (CPU clock units) for f^D is smaller then for f^C and there is much less rounding faults.

In my opinion DSSA should give better results in large scales TSP. Article [4] should be investigated for even shorter computational time to solve large scale TSP.

References

[1] Aihara K., Takabe T., Toyoda M., Chaotic neural networks, Phys. Lett. A, 144, 333-340, (1990)

[2] Chen L., Aihara K., Chaotic simulated Annealing by a Neural Network Model with Transient Chaos, Neural Networks, 8, 915-930, (1995)

[3] Adachi M., Aihara K., Associative dynamics in chaotic neural network, Neural Networks, 10, 83-98, (1997)

[4] Hasegawa M., Ikeguchi T., Aihara K., Solving large scale traveling salesman problems by chaotic neurodynamics, Neural Networks 15, 271-283, (2002)