Solving the TSP Using Chaotic Simulated Annealing

James Tunnell and Sami Abdul-Wahid

June 10, 2014

0.1 Abstract

We apply chaotic simulated annealing (CSA) using a transiently chaotic neural network (TCNN) to the traveling salesman problem (TSP). Our implementation follows the method described by Chen and Aihara in [CA95]. We apply the CSA process to several TSP instances. The resulting system is more effective at solving the TSP than a Hopfield Neural Network (HNN).

0.2 Background

0.2.1 Hopfield Neural Networks

Hopfiled neural networks (HNN) consist of a fully connected recurrent neural network structure. They were introduced by Jon Hopfield in 1982 as method of implementing content addressable memory. The weights between the matrices are determined according to the inputs to to be stored in the network, after which the system can be initialized by a partial state which will cause the network to converge to the reconstructed state. States are represented by the states of all the neurons after they have settled into a stable state. One can define a function, called an energy function, of the network's states that will be usefull in determining the weights. The energy function is used to encode a particular problem to be minimized. From the energy function, the set of weights interconnecting the neurons can be determined such that, given any starting condition of the network, the energy function will decrease nearly monotonically. As such, one can encode problems such as the traveling salesman problem using a hopfield neural network, and come out with an approximate solution.

0.2.2 Traveling Salesman Problem

In the traveling salesmen problem (TSP), a set of cities are to be visited in a tour, without revisiting cities. For a symmetric TSP distance between cities is the same in both directions. In this case, there are $\frac{n!}{2n}$ unique tours, so an exhaustive search quickly becomes infeasible.

0.2.3 Solving the TSP Using a HNN

Hopfield and Tank [HT85] first used neural networks to solve the TSP, using a HNN. To solve an n-city TSP, n^2 nodes are needed, and node outputs are arranged in a n-by-n matrix. The output matrix of a solution amounts to a permutation matrix on the tour order, so a '1' in a position (row i, column j) indicates that city i will be the jth city visited. To be valid, the outputs should have only one '1' in each row and in each column.

To get a HNN to solve the TSP, the energy function is defined as

$$E = A/2 \sum_X \sum_i \sum_{j!=i} V_{Xi} V_{Xj} + B/2 \sum_i \sum_X \sum_{X!=Y} V_{Xi} V_{Yi} + C/2 \left(\sum_X \sum_i V_{Xi} - n \right)^2 + D/2 \sum_X \sum_{Y!=X} \sum_i d_{XY} V_{Xi} (V_{Y,i+1} + V_{Y,i-1})$$

where V_{Xi} corresponds to the output of the neuron representing city X in tour position i. Here, A, B, C, and D are constants to weight the energy terms, either emphasizing validity or optimality.

0.2.4 Simulated Annealing

The main difficulty with the HNN is the purely monotically decreasing nature of the energy function. This means that the network will converge to a local minimum determined purely by the starting state. As such, it may be rare to pick an initial state that will lead to convergance at a global minimum.

Stochastic Simulated Annealing

This problem can be addressed by routinely perturbing the state of the network at each iteration of its evolution. One method of doing this is injecting random solutions into the system with decaying overall quantities. Thus the state of the system can be perturbed out of local minima and have a better chance at reaching the global optimum, presumably because of attraction tendencies to the global optimum would be statistically greater than those due to other, local, minima.

Chaotic Simulated Annealing

Aside from injection of stochastic numbers, a chaotic function can be used instead. Chaotic behavior of a system is, classically, a completely deterministic phenomenon characterized by highly consistent tendenices of the system in question to gravitate towards a certain sup-region of the state space called the attractor. It is presumed that the attractor encompases some or all of the local minima in a problem's solution space. Because a chaotic system would be systematically traversing regions in the attractor space, and thus encountering all the encompassed local minima, the approach to the local minima would thus be more efficient than the haphazard wanering of a stochastic based system. As such, convergence in chaotic systems is faster than when using purely stochastic annealing techniques.

0.3 Chaotic Neural Network

The chaotic neural network (CNN) model used in [CA95] is established by adding a self-connection weight, so each neuron feeds back to itself to some degree. This model leads

to rich dynamics with co-existing attractors, both fixed-point and strange. This behavior is in contrast to the standard HNN, that will converges monotonically to equilibrium (if weights are symmetric).

0.3.1 Transient Chaos

While the chaotic dynamics are desirable for searching, they do not lead to an equilibrium state at one of the minima. So, to harness the chaotic behavior and still retain a convergent behavior in the end, the self-connection weight is reduced over time by an exponentially decaying parameter. For this reason, the CNN model used is called a transiently chaotic neural network (TCNN).

0.3.2 Chaotic Neural Network Model

The overall model for the CNN is much like HNN, except for the extra self-connection term, weighted by $z_i(t)$, and also that the neuron inputs are weighted by parameter α. See Figure 0.1. Of course, if $z_i(t) = 0$ and $\alpha = 1$ then this model reduces to the standard HNN.

$$y_i(t+1) = k y_i(t) + \alpha \left(\sum_{j=1, j \neq i}^{n} w_{ij} x_j(t) + I_i \right) - z_i(t)(x_i(t) - I_0)$$

Figure 0.1: The neuron update equation for the CNN model presented in [CA95]

0.4 Implementation

The CNN model described in [CA95] was implemented in Python using numpy. The code is available in the Git repository at `https://github.com/jamestunnell/chaotic_tsp`. The primary file is `tcnn.py`, which contains the `TCNN` class for creating and running CSA to solve the TSP. There is also a utility function in `tsplib.py` for extracting a distance matrix from a TSPLIB XML file.

Finally, `csa_tsp.py` provides a command-line interface to run TSP-solving scenarios on different TSP files and parameters. The usage is as follows:

```
Solve the traveling salesman problem (TSP) by chaotic simulated annealing
(CSA), using a transiently chaotic neural network (TCNN).

Usage:
    csa_tsp.py TSP_FILE [options]

Arguments:
    TSP_FILE The TSPLIB XML file containing distances between cities
             (TSP problem instance).

Options:
```

```
--nruns=NRUNS          Solve TSP by CSA NRUNS times [default: 1]
--maxiter=MAXITER      Maximum iterations to run CSA for [default: 1000]
--k=K                  Nerve membrane damping factor, 0 to 1 [default: 0.9]
--alpha=ALPHA          Scaling parameter for neuron inputs [default: 0.015]
--beta=BETA            Self-connection damping, 0 to 1 [default: 0.01]
--z0=SELFCONN          Self-connection weight start value [default: 0.1]
--I0=INPUTBIAS         Input bias [default: 0.5]
--epsilon=EPSILON      Neuron output function steepness [default: 0.004]
--W1=VALIDITYWT        Weight of validity constraint [default: 1]
--W2=OPTIMALITYWT      Weight of tour optimality constraint [default: 1]
--energy               Graph energy data, as line plot
--percent              Graph percent valid data, as line plot
--length               Graph tour length data, as histogram
```

0.5 Results

The TCNN was used to perform CSA on several TSP instances. The resulting tour lengths obtained are shown in the tables below. The global optimum is given for reference. All TSP instances are obtained from TSPLIB.

In all cases, the CSA parameters used are as shown in Table 0.1

Table 0.1: CSA parameters used to find solutions for burma14

Parameter	Value
I_0	0.5
$W1$	1
$W2$	1
α	0.015
β	0.01
ϵ	0.004
k	0.9
$z(0)$	0.1

0.5.1 14-city TSP: burma14

The global optimum for *burma14* is 3323. The optima found are shown in Table 0.2. See that 25% of the solutions found were the global optimum, and 48% of solutions were within 1% of the global optimum

0.5.2 17-city TSP: gr17

The global optimum for *gr17* is 2085. The optima found are shown in Table 0.3. In this case, none of the the solutions found were the global optimum, and 4% of solutions were within 1% of the global optimum. However, 68% of solutions were withing 5% of the global optimum.

Table 0.2: Solutions produced by CSA on the *burma17* TSP instance

Tour length	Times found
3323	5
3336	3
3346	4
3359	5
3369	1
3381	2
3386	1
3413	1
3431	1
3432	1

0.6 Conclusions and Future Work

CSA was found to be an effective method to find TSP solutions, though it can be slow on larger problems, and will less consistently approach the global minimum.

In the future, this method can be improved by combining the chaotic dynamics with stochastic dynamics, called stochastic chaotic simulated annealing (SCSA), as presented in [WLTF04].

Table 0.3: Solutions produced by CSA on the *gr17* TSP instance

Tour length	Times found
2103	1
2133	1
2142	1
2144	1
2151	4
2155	1
2159	1
2164	3
2167	1
2170	1
2176	1
2183	1
2191	1
2194	1
2198	1
2215	1
2228	1
2251	1
2320	1

Bibliography

[CA95] Luonan Chen and Kazuyuki Aihara. Chaotic simulated annealing by a neural network model with transient chaos. *Neural networks*, 8(6):915–930, 1995.

[HT85] John J Hopfield and David W Tank. neural computation of decisions in optimization problems. *Biological cybernetics*, 52(3):141–152, 1985.

[WLTF04] Lipo Wang, Sa Li, Fuyu Tian, and Xiuju Fu. A noisy chaotic neural network for solving combinatorial optimization problems: Stochastic chaotic simulated annealing. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 34(5):2119–2125, 2004.