Neural, Parallel, and Scientific Computations 17 (2009) 409-432

HOPFIELD NEURAL NETWORKS: ADAPTIVE COEFFICIENTS AND OTHER ENHANCEMENTS

LAURENE V. FAUSETT

Texas A&M University-Commerce, Department of Mathematics
Commerce, TX 75429-3011

ABSTRACT. The Hopfield neural network has been an important tool for solving the travelling salesman problem for over 25 years. However, computational issues have continued to make it difficult to use in practical settings. This paper presents two approaches to solving one of the continuing dilemmas, namely how to find appropriate coefficients for the terms in the energy function. Several other enhancements to the original model are also presented, which improve the reliability of modified networks to find good valid tours.

Keywords. Hopfield neural networks, travelling salesman problem, adaptive coefficients

1. INTRODUCTION

Neural networks provide an alternative to traditional computing techniques for many types of problems. Hopfield neural networks may be used for a variety of optimization and constrained optimization problems, including the classic traveling salesman problem.

In general, a neural network consists of a (large) number of simple processing elements (neurons), connected by weighted pathways over which signals are sent. The pattern of connection (network architecture) and method of determining the weights are two of the distinguishing characteristics of different types of neural networks.

A Hopfield neural network (in particular a continuous Hopfield network) is different from many other types of networks in that the weights on the connections are specified in advance, rather than trained. No examples of "correct" or "desired" results are provided to the network. Instead, an energy function is constructed so that a minimum of the energy function corresponds to a solution of the optimization problem. The question of setting the weights (which corresponds to the question of setting the coefficients on different terms in the energy function) has been a central question in the use of Hopfield networks.

In this paper, we consider several variations of the original Hopfield neural network for solving the classic travelling salesman problem. These include two approaches that allow the network to determine the appropriate coefficients for the

Received June 11, 2009

terms in the energy functions, as well as some alternatives to the standard random initialization of activations.

The paper is organized as follows. In the next section we present an overview of the travelling salesman problem (TSP) and the basic Hopfield neural network approach to the problem. In section 3, we present the balanced coefficient Hopfield method for the TSP. In section 4 we consider an approach to finding the coefficients based on Lagrange multipliers. Section 5 discusses various other computational considerations, including methods of initializing the activations to improve the quality of the valid tours. Finally, section 6 summarizes the results and draws a few conclusions.

2. TRAVELLING SALESMAN PROBLEM AND THE BASIC HOPFIELD NETWORK APPROACH

The travelling salesman problem is a classic constrained optimization problem. The goal is to find a path or tour of minimum distance which visits each of a set of n cities once and only once.

2.1. Hopfield Neural Networks.

A Hopfield neural network operates by letting the activations of the neurons evolve to minimize an energy function. The construction of an appropriate energy function is thus the key aspect of designing a Hopfield neural network for a particular problem. The energy function typically consists of several terms, each of which is minimized when a particular constraint is satisfied. In addition there is a term in the energy function corresponding to the objective function of the problem (in this case, finding the shortest possible tour). The weights on the connections between the neurons are directly related to the coefficients on the different terms in the energy function. However, in using a Hopfield network it is not necessary to explicitly display the weights for each neuron.

2.2. Architecture.

The basic approach to using a Hopfield network to solve the $n-$city travelling salesman problem was established early on (Hopfield and Tank, 1983, Szu, 1988). The network consists of n^2 neurons, arranged in an $n \times n$ array. The rows of the array represent cities to be visited, the columns represent the days (or stages) of the tour. The activation of each neuron ranges between 0 and 1, with 1 (or "on") denoting the fact that the city represented by the neuron is visited on the indicated day. Thus, a valid tour is given whenever the activations of the array are a permutation matrix (exactly one element on in each row and each column). The return of the salesman from the final city to the starting city is built into the network also.

2.3. **Energy Function.** The energy function used in the original presentation of the problem (Hopfield and Tank, 1983) had terms representing the row constraint (only one neuron on in each row), the column constraint (only one neuron on in each column), the total distance travelled (objective), and one "encouragement" term to ensure that exactly n cities were visited. This last term was intended to prevent a minimum tour of length zero achieved by the salesman staying home. Other researchers have found that a more specific form of encouragement was more effective. This approach uses two encouragement terms: one to ensure that exactly one city is visited on each day, and one to ensure that each city is visited exactly once. This is the form of the energy function that is most widely used at present, and the form that will be used in this paper. Thus the energy function has the form

$$E = 0.5[C_1E_1 + C_2E_2 + C_3E_3 + C_4E_4 + C_5E_5] \tag{1}$$

where

- E_1 represents the constraint that no city is visited more than once,
- E_2 represents the constraint that no more than one city is visited on any day,
- E_3 requires that the total number of cities visited on any day is one,
- E_4 requires that the total number of times a city is visited is one,
- E_5 represents the objective that total distance be as small as possible.

The internal activation of neuron $n_{x,i}$ is denoted $u_{x,i}$; its output signal is $v_{x,i}$ where $v = \tanh(u)$. The equations for these energy terms are

$$E_1 = \sum_x \sum_i \sum_{j \neq i} v_{x,i} v_{x,j}; \qquad E_2 = \sum_i \sum_x \sum_{y \neq x} v_{x,i} v_{y,i}; \tag{2}$$

$$E_3 = \sum_x (1 - \sum_j v_{x,j})^2; \qquad E_4 = \sum_i (1 - \sum_x v_{x,i})^2; \tag{3}$$

$$E_5 = \sum_x \sum_{y \neq x} \sum_i d_{x,y} v_{x,i} (v_{y,i+1} + v_{y,i-1}). \tag{4}$$

However, the issue has remained (until recently) as to how to set the coefficients on these terms.

2.4. **Evolution of Activations.** The evolution of the network consists of each neuron changing its internal activation in a manner that reduces the overall energy of the network. This is a numerical (discrete time-step) approach to solving a differential equation describing the connection between changes in activation and changes in energy.

$$\frac{du}{dt} = -\frac{\partial E}{\partial v}. \tag{5}$$

Thus, the question of appropriate values for the coefficients is inter-related to other computational issues such as the size of the time step and the steepness of the output function. The update equation for a neuron is given by

$$\frac{du}{dt} = -[C_1\frac{du_1}{dt} + C_2\frac{du_2}{dt} + C_3\frac{du_3}{dt} + C_4\frac{du_4}{dt} + C_5\frac{du_5}{dt}]. \tag{6}$$

The equations for these update terms are

$$\frac{du_1}{dt} = \sum_{j\neq i} v_{x,j}; \qquad \frac{du_2}{dt} = \sum_{y\neq x} v_{y,i}; \tag{7}$$

$$\frac{du_3}{dt} = \sum_{j} v_{x,j} - 1; \qquad \frac{du_4}{dt} = \sum_{x} v_{x,i} - 1; \tag{8}$$

$$\frac{du_5}{dt} = \sum_{y\neq x} d_{x,y}(v_{y,i+1} + v_{y,i-1}). \tag{9}$$

3. THE BALANCED HOPFIELD NETWORK

The relative importance of the different terms in the energy function has been a major subject of investigation in the the use of the Hopfield network.

3.1. **Motivation.** The idea behind the balanced coefficients approach is that the proportion of the total energy that comes from each energy term should be reflected in the relative importance of each of the corresponding terms in the update equations (Park, 1996). The coefficients are found to be

$$C_j = \frac{E_j}{\sum_j E_j} \tag{10}$$

which is simply a normalization of the partial derivative of the total energy with respect to the corresponding coefficient. These coefficients change as the energy changes, but eventually stabilize at values that allow the neural network to find good, valid solutions.

In the balanced coefficients approach, a first phase of evolution is used to find these proportions. The change in the energy is easily computed using some of the same calculations that are needed for the updates of the activations also, so the additional computational effort is relatively small. During the second phase the coefficients are held fixed and the network evolves to find a valid (and usually good) tour.

The coefficients found for a tour of a particular number of cities (with coordinates and distances normalized to the same range) is relatively independent of the actual cities used, so the coefficients can be found for each problem size without recomputing on each run. It is found that the average of several such balancing phases can be used effectively for a wide variety of city sets (with the same number of cities).

Since determination of the appropriate coefficients can be carried out independent of a specific city-set, separate Matlab functions are given in the Appendix for the determining of the balanced coefficients (`AHNN_B`) and the solution of the travelling salesman problem using fixed coefficients (`AHNN_F`).

3.2. **Sample results.** Typical results using `AHNN_F` with coefficients found from `AHNN_B` are shown in Figure 1.

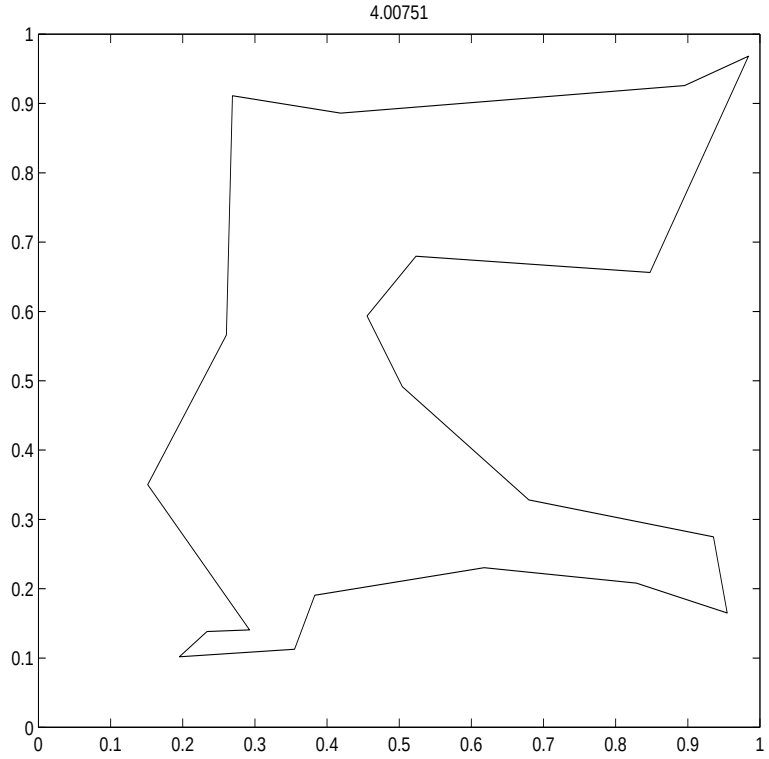

Figure 1: Typical 20-City tour for `AHNN_F`

The results using 10 different n-city sets, with 10 random initializations for each set:

n	C1	C2	C3	C4	C5
10	0.0526	0.0741	0.2301	0.2516	0.3916
15	0.0543	0.0708	0.2316	0.2481	0.3952
20	0.0546	0.0687	0.2292	0.2433	0.4041
30	0.0550	0.0684	0.2234	0.2368	0.4164

Rounding each of these to the nearest 0.05 (and making any necessary minor adjustments to ensure that the sum of the coefficients is 1), we have

```
C =  [ 0.05     0.05     0.25     0.25     0.40  ]
```

In some applications of this process, the coefficient of the objective term is held fixed, since the energy associated with that term cannot be driven to zero. For example, the results for $C_5 = 0.5$ and 10-cities may be approximated in a balanced form as

```
C = [  0.08    0.08    0.42    0.42    0.50  ]
```

Park investigated the use of different fixed values for C_5 and found that smaller values for larger city sets gave approximately half of the solutions as valid tours.

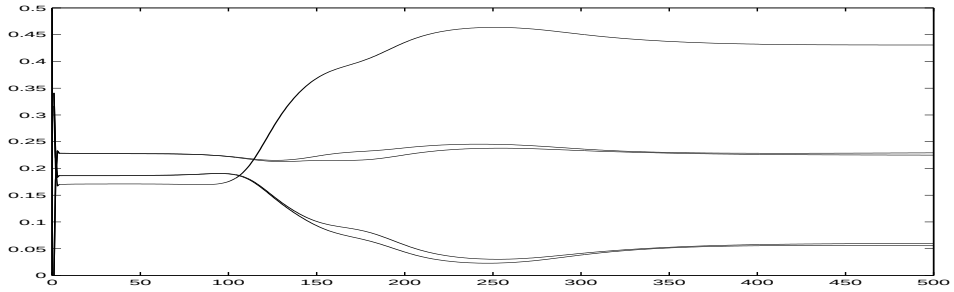

Figure 2: Example of balancing the coefficients

The Balanced Hopfield network gives much better results than those obtained using many other possible values for the coefficients. Another approach to the problem of determining the coefficients is considered in the next section.

4. THE LAGRANGE HOPFIELD NETWORK

The relative importance of the different terms in the energy function has been a major subject of investigation in the the use of the Hopfield network.

4.1. **Motivation.** This approach views the coefficients as Lagrange multipliers, and evolves both coefficients and activations simultaneously (van den Berg, 1996). The activations, $u(x, y)$, update as in the Balanced Hopfield Network.

The equations for the updating of the coefficients are

$$\frac{dC_1}{dt} = v_{y,x} \sum_{j \neq x} v_{y,j}; \qquad \frac{dC_2}{dt} = v_{y,x} \sum_{i \neq x} v_{i,x}; \tag{11}$$

$$\frac{dC_3}{dt} = (\sum_{j} v_{y,j} - 1)^2; \qquad \frac{dC_4}{dt} = (\sum_{i} v_{i,x} - 1)^2. \tag{12}$$

A Matlab function `AHNN_L` to implement this model is given in the Appendix.

4.2. **Sample Results.** The coefficients for the constraints in the Lagrange-Hopfield network evolve away from zero as the network seeks a valid tour. The size of these constraint coefficients continues to grow as the network evolves, although the changes become smaller as the network nears a valid tour. The Matlab function `AHNN_L` uses the same stopping condition (test for valid tour) and basic time-step (time-step for updating the activations) as the function `AHNN_F`. However, because the constraint coefficients grow more rapidly for larger sized problems, the time-step used for the coefficients is the basic time-step divided by the number of cities. It is worth noting that this network consistently finds valid tours, and in a relatively few epochs.

A solution of a 20-city problem using `AHNN_L` would specify the epoch at which a valid tour was first found, the path `pp`, and the length of the tour `len`. If the tour is shorter than any found previously, its length is designated as `len_s`. The path is a vector of the cities in the order in which they are visited. The coefficients may also be displayed if desired, in the vector `C`. Note that the coefficients given are the final coefficients when a valid tour was first detected, and would not work effectively as fixed coefficients. Typically several different initializations of the neurons would be used for each problem. An example of typical results is given below.

```
valid path at epoch 17
C =  [  0.8553,    0.8573,    0.8988,    0.8969,    1.0000  ]
pp = [ 19,13, 7,10,12, 6, 5,17, 3,15, 4,16,11,20, 1,18, 9, 2,14, 8]
len_s =    4.0634
```

The Matlab scripts given in the Appendix also graph the tour each time a shorted tour is found.

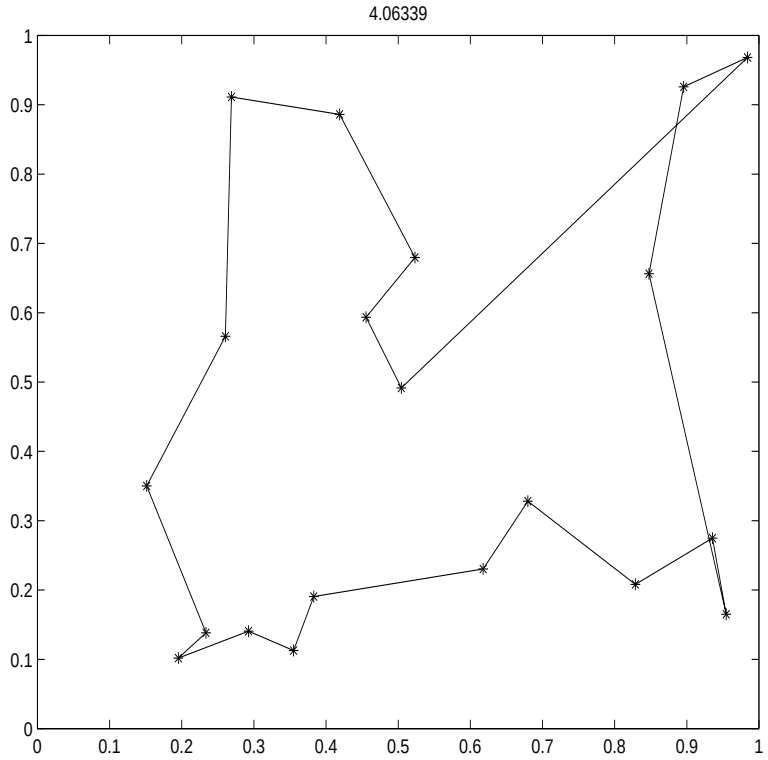

Figure 3: Typical 20-City tour for `AHNN_L`

The quality of the solutions found by the Lagrange-Hopfield model is influenced by the initial activations of the neurons, as is the case for the Balanced-Hopfield model discussed in the previous section. The next section considers this aspect, and others that impact the computational characteristics of Hopfield neural network solutions of the travelling salesman problem.

5. OTHER CONSIDERATIONS

Other computational considerations include the initialization of activations, and the choice of update order for the neurons (for each cycle, or "epoch"). In theory neurons are updated in a random order, but this raises the question as to whether all neurons are in fact being updated in a uniform manner. Other possible update strategies are sequential, or cycle with random start, or batch updates. The effect of these choices is discussed below.

5.1. **Initializations.** There is a lot of variation in the quality of the results from both the Lagrange-Hopfield and Balanced-Hopfield approaches (even for the same city set). This suggests that the random initialization of the activations (which has been standard for Hopfield neural networks in general) of the neurons may be a significant difficulty. To mitigate this problem, a biasing of the initialization towards shorter tours has been developed.

5.1.1. *Biased Initializations.* The biased initialization begins by specifying a starting city (say city j) for the tour (by increasing the initial activation of that neuron on day 1). It then proceeds to encourage the city that is closest to city j to be visited on day 2 (again by increasing the initial activation of that neuron). The initialization continues in this way until the initializations have biased the tour by increasing the initial activation of one neuron in each column. To prevent selecting the same city to be encouraged on different days, once a nearest city is selected, it is prevented from being selected again. This means that the tour is most strongly biased toward short paths for the first few days, i.e. for cities that are close to the starting city. For this reason, the biased initialization is used with a sequence of n starting initializations, with each of the n cities in the problem chosen as the starting city in one of the initializations. Scripts for running `AHNN_F` and `AHNN_L` with biased initializations are given in the Appendix (codes `AHNN_F_run_bias` and `AHNN_L_run_bias` respectively).

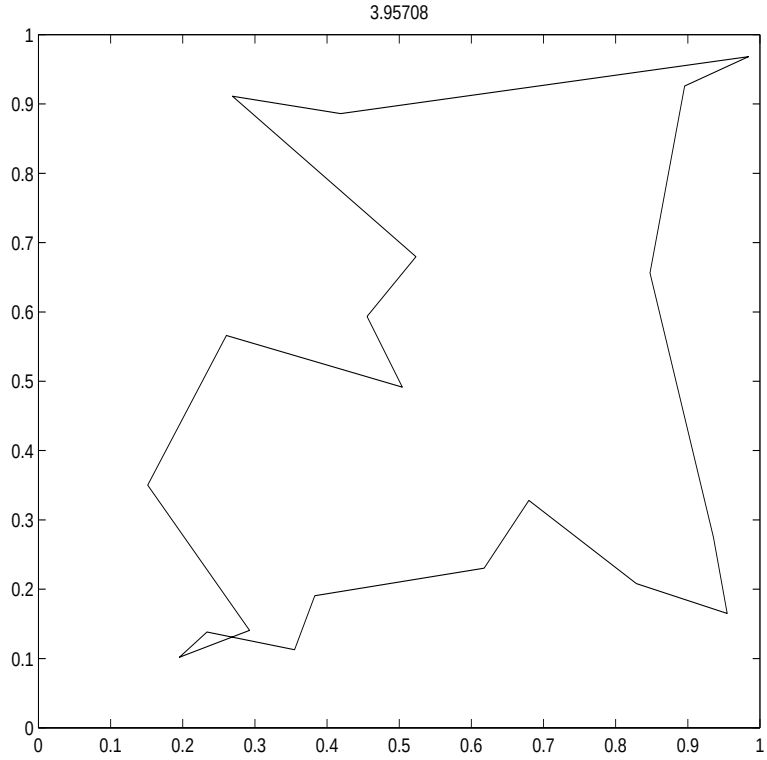

Figure 4: 20-City tour for `AHNN_F` with biased initialization

5.1.2. *Encouraged Initializations.* Another approach to improving the quality of the tours found by Balanced Hopfield and Lagrange Hopfield neural networks is to encourage all cities within a specified distance (for example 0.5 units) of the city being visited on day 1, (and also, if desired, discourage all that are further than a certain distance away). This approach allows for some experimentation in terms the radius for encouragement (the specified distance) or discouragement, as well as the strength of the encouragement or discouragement. This only starts the solution in the direction of a valid tour, and initializations with different starting cities are still important.

The following segment of Matlab code illustrates the changes that would need to be made to the codes `AHNN_F_run_bias` and `AHNN_L_run_bias` to use the encouraged initialization. In addition to the code shown, one would define the matrix `d1 = d + eye(N_CT)` just after computing matrix `d` (where `N_CT` is the number of cities in the problem). Then insert the following code after the computation that finds the city (which has not been encouraged previously) that is nearest the current city, and encourages it to be visited on the next day. The radius of encouragement (0.5) and the strength of encouragement (0.25) may be modified as desired.

```
% encourage visiting nearby cities on day kk = k + 1
      for i = 1:N_CT-1
          if d1(j, i)< 0.5
              u(i, k+1) =  u(i,k+1)+ 0.25;
```

```
        end
    end
```

This provides a small additional encouragement for visiting any of the cities which are within 0.5 units of the city currently being visited.

Typical results for a 10-city problem illustrate the use of both biased initialization and biased and encouraged initialization for the `AHNN_F` and `AHNN_L` networks. The matrix `p` gives the coordinates of the cities in Matlab format. The elapsed time (for 10 initializations) is computed using the Matlab functions `tic` and `toc`. The function `tic` starts a stopwatch timer and `toc` displays the elapsed time (in seconds) at the end of the run.

The coordinates of the cities:

```
p =  [  0.9226    0.8743
        0.2258    0.4316
        0.5838    0.8260
        0.0928    0.7593
        0.6253    0.8739
        0.6371    0.4706
        0.7204    0.8275
        0.8487    0.7155
        0.9101    0.4381
        0.0808    0.7972  ]
```

The Balanced Hopfield network with biased initialization finds valid tours for 4 of the 10 initializations. The best tour is

```
pp =   [  2     4    10     3     5     7     1     8     9     6 ]
len_s =    2.4212
elapsed_time =    9.9540
```

As in other examples, the `AHNN_F` network uses the coefficients found earlier, namely `C = [ 0.05,  0.05, 0.25, 0.25, 0.4 ]`.

The Balanced Hopfield network using biased initialization with additional encouragement finds valid tours for 3 of the initializations. The best tour is the same as that found with biasing alone; the elapsed time is similar.

The Lagrange Hopfield network with biased initialization finds valid tours for all 10 initializations. The best tour is

```
valid path at epoch 10
C =  [  0.2879    0.2855    1.3809    1.3514    1.0000 ]
ppp =   [  2     4    10     3     5     7     8     1     9     6 ]
len_s =    2.5363
```

```
elapsed_time =    2.7240
```

The Lagrange Hopfield network using biased initialization with additional encouragement finds valid tours for all 10 initializations. The best tour found is the same as (a permutation of) that found with biasing alone, but the time required is approximately 1/2 of the time for biasing alone.

5.1.3. *Path Following.* A third possible way of improving the quality of the solutions is based on the idea of path-following used in some numerical solutions of partial differential equations. This approach may be combined with either a biased initialization or an encouraged initialization (or with random initialization for that matter). In order to use path-following, the function implementing the neural network must return the matrix of internal activations $u1$ for the valid tour. A new initialization is then formed using an average (or other weighted combination) of the activations for the valid tour and random activations U_r (denoted as `noise` in the following segment of Matlab code). Depending on the proportion of $u1$ and U_r, the network may find a tour which is similar to the previous solution, but possibly shorter. If too much emphasis is placed on the activations from the valid tour, the network will simply find the same solution several times.

The approach is illustrated in the following segment of code, which can be used to replace the corresponding part of the scripts given in the Appendix. Portions of the computations which are the same as in the other scripts, such as the plotting of better solutions, are omitted.

```
for k = 1:k_max
    [len, path, u1] =  AHNN_F(d, u, MAX_EP_B, tm_st, C, u0);
    if len ~= 0
       u = 0.5*noise + 0.5*u1;
    end
end
```

This tends to give the same tour for each retry. Adjusting the proportion of the previous activations and the new noise improves the results in some cases, but as with the encouraged initialization, there is room for experimentation in adjusting the parameters.

5.2. **Update Options.** The basic Hopfield network was originally presented in terms of random updates of the neurons. This has biological motivation, but may cause difficulties in implementation. Sequential updates of the neurons ensures that all neurons update on each epoch, and simplifies coding, but raises the question as to whether the order of updates plays any role in the quality of the solution.

Within the structure of the MATLAB codes given in the Appendix for the Balanced Hopfield Network, several variations of the basic sequential update scheme have been investigated.

The basic updating strategy is given in the following code:

```
for x = 1:N_CT
    for y = 1:N_CT
        [UinTot, S] = Sum_in(y, x, v, d, C, N_CT);
        u(y,x) = u(y,x) + tm_st*UinTot;
        v(y,x) = 0.5*(1.0+tanh(u(y,x)/u0));
    end
end
```

It may be summarized as

```
for x = 1:N_CT
    for y = 1:N_CT
        update u;     update v
    end
end
```

Other variations include:
Interchanging the order of the loops:

```
for y = 1:N_CT
    for x = 1:N_CT
        update u;     update v;
    end
end
```

Updating by rows or by columns (vectorizing the update of v).

```
for x = 1:N_CT
    for y = 1:N_CT
        update u
    end
    update v
end
```

Updating the entire matrix of v.

```
for x = 1:N_CT
    for y = 1:N_CT
        update u
    end
```

```
end
update v
```

No significant difference in the quality of the solutions, or the speed of the computation, was detected. Further vectorization of the updates of u would require modification of the subroutine `Sum_in`, and was not investigated.

5.3. **Energy Function Options.** The energy function in the original Hopfield network included only one "encouragement term" which was intended to ensure that the number of neurons "on" was the same as the number of cities. One of the first modifications made by other researchers was the introduction of more specific encouragement, specifically the use of two terms, one to ensure that there was one neuron "on" in each row, and one in each column.

The original Hopfield network also included a decay term in the update equations for the activations, which corresponded to an integral term in the energy function. It was pointed out (Takefuji, 1992) that using the decay term did not guarantee convergence to an energy function constructed (without the integral term) to specify the desired solution to the travelling salesman problem. Many, but not all researchers since then have not included the decay term in the activation updates. None of the codes included in this study use the decay term, although it should be noted that the introduction of the use of Lagrange multipliers in a Hopfield network (van den Berg, 1996) on which the Lagrange Hopfield network presented here is based, did use the decay term.

Further modifications of the energy function may be investigated in future work. For example, if specific row and column encouragement terms are beneficial (and their wide acceptance indicates that is the case), then perhaps further modification of these terms so that shorter tours are encouraged could also be helpful.

5.4. **Stopping Conditions.** All of the results reported in this paper are based on declaring a neuron to be "on" if its output signal v is greater than 0.5. The iterations stop when a valid tour is detected (or after a maximum number of epochs. Some other investigators have taken a much higher threshold value for declaring neurons to be "on" or "off", and have therefore required many more epochs. The number of epochs required is also directly influenced by the time step used in the computations. Researchers who include the decay term as given in the original Hopfield network use a much smaller time step in order to keep the influence of the decay term quite small.

6. SUMMARY AND CONCLUSIONS

Investigations of the use encouraged initializations, or path-following, do not indicate that their basic forms provide any advantage over the use of the biased

initialization described in section 5.1.1. However, comparison of AHNN_F using the coefficients determined by AHNN_B, and AHNN_L, provide some interesting insights into the usefulness of Hopfield-type neural networks for solving the travelling salesman problem.

The following summary of results illustrates the typical behavior of the Balanced Hopfield and Lagrange Hopfield networks using the original 10-city problem from (Hopfield, 1984), 5 other random 10-city problems, 5 random 20-city problems, and 5 random 30-city problems, and 5 random 40-city problems.

6.1. **10-city problems.** In this section we illustrate typical results for the Balanced Hopfield and Lagrange Hopfield networks using several different initializations. In each case the original Hopfield city-set, and 5 rother random city-sets are used. There are 10 initializations for each city set. The results for the Balanced Hopfield network are found using AHNN_F with the coefficients found previously using AHNN_B.

6.1.1. *Balanced Hopfield Network; Random Initializations.* For the original Hopfield 10-city problem, 2 valid tours were found, one of which was optimal. The optimal tour was found after 150 epochs; its length is 2.6907.

```
pp =   [  7     8     9    10     2     3     1     4     5     6 ]
```

Five random 10-city sets were also used. No valid tours for any of these city-sets.

6.1.2. *Lagrange Hopfield Network; Random Initializations.* The Lagrange Hopfield network found 10 valid tours for each city-set; for the original data set the optimal tour was found in 19 epochs.

For the 5 random 10-city sets, the Lagrange Hopfield network required approximately 15 epochs to find a valid (and very good) tour. In each case the final coefficients were very similarly, approximately

```
C =  [  0.5    0.5    2.0    2.0    1.0 ]
```

6.1.3. *Balanced Hopfield Network; Biased Initializations.* For the original data, 6 of the 10 trials gave valid tours, including the optimal tour. No valid tours were found for four of the random 10-city sets; for the fifth city-set, 3 of the 10 trials gave valid tours.

6.1.4. *Lagrange Hopfield Network; Biased Initializations.* The Lagrange Hopdield network found 10 valid tours for each city-set. For the 5 random 10-city sets, the Lagrange Hopfield network with biased initialization required approximately 10 epochs to find a valid (and very good) tour. In each case the final coefficients were approximately

```
C =  [  0.25    0.25    1.5    1.5    1.0 ]
```

6.2. **20-city problems.** We now illustrate the behavior of the Balanced Hopfield and Lagrange Hopfield networks for 5 random 20-city sets;, using 20 initializations for each city set.

6.2.1. *Balanced Hopfield Network; Biased-Encouraged Initializations.* For 2 of the city sets there were no valid tours found; for the other city sets 2/20, 3/20 and 7/20 valid tours were found. The best tours found for each of the 5 city-sets appeared to be of very good quality.

6.2.2. *Balanced Hopfield Network; Biased-only Initializations.* For 1 of the city sets there were no valid tours found; for the other city sets 1/20, 3/20, 4/20 and 4/20 valid tours were found. The best tours found for each of the 5 city-sets appeared to be of good or very good quality. The elapsed time for each city set is a little over 90 seconds; city sets in which there are more valid tours execute more quickly, as fewer epochs are computed.

6.2.3. *Lagrange Hopfield Network; Biased-Encouraged Initializations.* Valid tours were found in all but one case. The best tour found for each of the 5 city-sets appeared to be of very good quality.

6.2.4. *Lagrange Hopfield Network; Biased-only Initializations.* Valid tours were found in all cases. The best tour found for each of the 5 city-sets appeared to be of very good quality. The elapsed time for each city set is between 12 and 15 seconds.

6.3. **Larger problems.** We now consider some 30-city and 40-city problems. In each case there are 5 randomly generated city-sets and the same number of initializations for each city-set as there are cities.

6.3.1. *Balanced Hopfield Network; Biased-only Initializations.* The results for the Balanced Hopfield network in solving 5 randomly generated 30-city problems, are summarized as follows. The city sets had 1/30, 1/30, 3/30, 11/30 and 12/30 valid tours; the elapsed time ranged from 284 to 337 seconds for the the 30 runs for each city-set.

6.3.2. *Lagrange Hopfield Network; Biased-only Initializations.* For the 5 randomly generated 30-city problems, valid tours were found in all cases. The elapsed time ranged from 40 to 61 seconds.

The best tour found for each of the 5 city-sets appeared to be of good to very good (but not optimal) quality.

6.3.3. *Lagrange Hopfield Network; Biased-only Initializations.* Due to the relatively low proportion of valid tours, and the rather long computational times required, results for larger city sets using the balanced Hopfield network are not included. 5 random city sets; 40 initializations for each city set. valid tours found in all cases.

Elapsed time ranged from 100 to 144 seconds. The best tour found for each of the 5 city-sets appeared to be of good to very good (but not optimal) quality.

6.4. **Conclusions.** The results of our investigation of two methods of determining the coefficients for a Hopfield network to solve the travelling salesman problem demonstrate that allowing the network to adapt these coefficients (either in advance, as is done for the Balanced Hopfield network, or during the solution process, as is the case for the Lagrange Hopfield network) improves the proportion of valid tours that are found by the network. The variation in the results for different random initializations indicates that initialization of the activations plays a significant role in the validity and quality of the solutions. Although the Lagrange Hopfield network almost always finds a valid tour, and finds it quickly, the quality of the tour may not be particularly good unless the initial activations are biased towards a shorter tour. The quality of the results produced by both the Balanced and the Lagrange Hopfield networks is greatly improved by using a biased initialization of the activations.

ACKNOWLEDGEMENTS

The author wishes to thank Jason Moore for his work on translating the C programs in Dr. Park's thesis into Matlab. That Matlab program formed the starting point for the various functions developed for this research and given in the Appendix.

REFERENCES

1. Fausett, L. V. (1994). *Fundamentals of Neural Networks.* Englewood Cliffs: Prentice Hall.
2. Hopfield, J. J. & Tank, D. W. (1986). Computing with neural circuits: a model, *Science*, (8), 625-633.
3. Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities, *Proceedings of the National Academy of Sciences*, (79), 2554-2558.
4. Hopfield, J. J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons, *Proceedings of the National Academy of Sciences* (81), 3088-3092.
5. Park, C.-Y. & Fausett, D. W. (1995). Energy Function Analysis for Improved Performance of Hopfield-Type Neural Networks, *Intelligent Engineering Systems Through Artificial Neural Networks*, Vol. 5, ASME Press, pp. 995-1000.

6. Park, C.-Y. (1996). *Energy Landscape Analysis of the Performance of Hopfield Neural Networks as a Method of Solving Combinatorial Optimization Problems.* Ph. D. Dissertation, Florida Institute of Technology, Melbourne, FL.
7. Takefuji, Y. (1992). *Neural Network Parallel Computing.* Boston: Kluwer Academic Publishers.
8. Tank, D. W. & Hopfield, J. J. (1987). Collective computation in neuronlike circuits, *Scientific American*, (257), 104-114, .
9. van den Berg, J. (1996). *Neural Relaxation Dynamics.* Ph.D. Dissertation, Erasmus University, Rotterdam. Netherlands.
10. Wilson, G. V. & Pawley, G. S. (1988). On the stability of the travelling salesman problem algorithm of Hopfield and Tank, *Biological Cybernetics*, (58), 63-70.

APPENDIX

Function to solve TSP by evolving coefficients and activations.

```
function [len, path, C] =  AHNN_L(d, u, EP_a, tm_st, u0)
[N_CT, M] = size(d);   v = 0.5*(1.0+tanh(u));       u = u*u0;
n_EP_a=0;   C = [0, 0, 0, 0, 1];   tm_st_C = tm_st/N_CT;
while n_EP_a < EP_a
   for x = 1:N_CT
       for y = 1:N_CT
           [UinTot, S] = Sum_in(y, x, v, d, C, N_CT);
           u(y,x) = u(y,x) + tm_st*UinTot;
           v(y,x) = 0.5*(1.0+tanh(u(y,x)/u0));
           C(1) = C(1) + tm_st_C*v(y,x)*S(1);
           C(2) = C(2) + tm_st_C*v(y,x)*S(2);
           C(3) = C(3) + tm_st_C*S(3)*S(3);
           C(4) = C(4) + tm_st_C*S(4)*S(4);
       end
   end
   C;
   n_EP_a = n_EP_a+1;
% check for valid path
   [invalid, path] = check_valid_tour(v, N_CT);
   if invalid == 0
       out = [ 'valid path at ', num2str(n_EP_a-1) ];   disp(out)
       C,     break
   end
end
if invalid == 1
   len = 0;    return
end
%compute path length%
len=0;
for i=1:(N_CT-1)
    len=len+d(path(i),path(i+1));
end
len = len + d(path(N_CT),path(1));
```

Functions used by AHNN_L and AHNN_F; function Sum_in is also used by AHNN_B.

```
function [invalid, path] = check_valid_tour(v, N_CT)
   invalid=0;      pos=0;  tol=0.5;
   path = zeros(1, N_CT);
   for j=1 : N_CT
       on=0;
       for i=1:N_CT
           if v(i,j)>=tol
              on=on+1;
              k = i;
           end
       end
       if on==1
           path(j) = k;
           for jj = 1 : j - 1
               if path (jj)==k
                  invalid = 1;
                  break
               end
           end
       else
           invalid=1;
       end
   end
```

```
function [UinTot, S] = Sum_in(y,x,v,d,C,N_CT)
    S = [0 0 0 0 0];
    if x < 2,         x_m = N_CT; else x_m = x-1;  end
    if x > (N_CT-1),  x_p = 1;    else x_p = x+1;  end
    S_row = sum(v(y,:));          S_col = sum(v(:,x));
    S(1) = S_row - v(y,x);        S(2) = S_col - v(y,x);
    S(3) = S_row - 1;             S(4) = S_col - 1;
    S(5) = S(5)+d(y,:)*(v(:,x_p)+v(:,x_m));
    Uin  = C.*S;
    UinTot = -1.0*sum(Uin);
```

Function to solve TSP using fixed coefficients.

```
function [len, path, u] =  AHNN_F(d, u, MAX_EP_B, tm_st, C, u0)
[N_CT, M] = size(d);  v = 0.5*(1.0+tanh(u));  u = u*u0;  n_EP = 0;
while n_EP < MAX_EP_B
    for x = 1:N_CT
        for y = 1:N_CT
           [UinTot, S] = Sum_in(y, x, v, d, C, N_CT);
           u(y,x) = u(y,x) + tm_st*UinTot;
        end
    end
    v = 0.5*(1.0+tanh(u/u0));
   [invalid, path] = check_valid_tour(v, N_CT);
   n_EP=n_EP+1;
   if invalid == 0
       out = [ 'valid path at epoch ', num2str(n_EP-1) ];   disp(out)
       break
   end
end
if invalid == 1
   len = 0;    return
end
%compute path length%
len=0;
for i=1:(N_CT-1)
    len=len+d(path(i),path(i+1));
end
len = len + d(path(N_CT),path(1));
```

Script to run AHNN_L, with biased initialization

```
% AHNN_L_run_bias, biased initialization
clear,   tic,   N_CT = 20,  p = rand(N_CT,2)
d2 = zeros(N_CT,N_CT);
for i=1:N_CT
    d2(:,i)=((p(i,1)-p(:,1)).^2+(p(i,2)-p(:,2)).^2);
end
d = d2.^(1/2);       vc = 0;          len_s = N_CT;
EP_a = 100;          tm_st = 0.1;     u0 = 0.1;
noise = rand(N_CT)-0.5;
for j = 1:N_CT
   u = 0.2*noise + atanh(2.0/N_CT-1);
   v = 0.5*(1.0+tanh(u));
   k = 1;     u(j,k) = 1;      %start at city j, on day k = 1
   clear enc, enc(1)=j;   dd =d+eye(N_CT);   dd(:,j) =dd(:,j)+2;
   for kkk = 1:N_CT-1
      [a, jj]  =  min(dd(j, :));   kk = k + 1;   enc = [enc, jj];
      % encourage city jj on day kk = k + 1
      u(jj, kk) =  u(jj,kk) + 0.5;
      % prevent city j from being chosen again
      j = jj;    dd(:, j) =  dd(:, j)+2;      k = kk;
   end
   [len, path, C] =  AHNN_L(d, u, EP_a, tm_st, u0);
   ppp = path(1:N_CT),    len, C;
   if len ~= 0
      vc = vc + 1;
      if len < len_s            % graph path if it is better
         len_s = len,            figure(vc),
         x = p(path(1:N_CT), 1); y = p(path(1:N_CT), 2);
         X = [x; x(1)];          Y = [y; y(1)];
         plot(X,Y, p(:, 1), p(:, 2), '*');
         axis([ 0, 1, 0, 1]);    title(len)
      end
   end
end
vc, len_s, toc
```

Script to run AHNN_F with biased initialization

```
clear,  tic,  N_CT = 10,    p = rand(N_CT, 2)
d2 = zeros(N_CT,N_CT);
for i=1:N_CT
    d2(:,i)=((p(i,1)-p(:,1)).^2+(p(i,2)-p(:,2)).^2);
end
d = d2.^(1/2);        vc = 0;             len_s = 10;
MAX_EP_B = 200;       tm_st = 0.1;        u0 = 0.1;
C =[ 0.05     0.05       0.25      0.25      0.4  ];
noise = rand(N_CT)-0.5;
for j = 1:N_CT
   u = 0.2*noise + atanh(2.0/N_CT-1);
   v = 0.5*(1.0+tanh(u));
   k = 1;     u(j,k) = 1;      %start at city j, on day k = 1
   clear enc,  enc(1)=j;  dd = d+eye(N_CT);   dd(:,j)= dd(:,j)+ 2;
   for kkk = 1:N_CT-1
      [a, jj]  =  min(dd(j, :));   kk = k+1;   enc = [enc, jj];
      % encourage city jj on day kk = k + 1
      u(jj, kk) =  u(jj,kk) + 0.5;
      % prevent city j from being chosen again
      j = jj;    dd(:, j) =  dd(:, j)+2;      k = kk;
   end
   [len, path, u] =  AHNN_F(d, u, MAX_EP_B, tm_st, C, u0);
   ppp = path(1:N_CT);     len;
   if len ~= 0
         vc = vc + 1;
         if len < len_s            % graph path if it is better
            len_s = len,            ppp,    figure(k),
            x = p(path(1:N_CT), 1); y = p(path(1:N_CT), 2);
            X = [x; x(1)];          Y = [y; y(1)];
            plot(X,Y, p(:,1), p(:,2), '*');  axis([0,1,0,1]);
            title(len)
         end
    end
end
toc; vc
```

Function to compute balanced coefficients (used by `AHNN_F` to solve the TSP.)

```
function C =  AHNN_B(d, u, EP_a, tm_st, u0)
[N_CT,M] = size(d); v = 0.5*(1.0+tanh(u));  u = u*u0;  n_EP_a=0;
% compute initial energy of network
E = zeros(1,5);
for i=1:N_CT
    for j=1:N_CT
        for k=1:N_CT
            if(j~=k), E(1) = E(1)+v(i,j)*v(i,k);        end
            if k~=j,  E(2) = E(2)+v(j,i)*v(k,i);        end
            k_m = k-1;    if(k_m < 1),    k_m = N_CT;   end
            k_p = k+1;    if(k_p > N_CT), k_p = 1;      end
            E(5)= E(5)+d(i,j)*v(i,k)*(v(j,k_p)+v(j,k_m));
        end
    end
    E(3) = E(3)+(sum( v(i,:) )-1)^2;
    E(4) = E(4)+(sum( v(:,i) )-1)^2;
end
E2 = 0.5*E;
%balance the coefficients%
while n_EP_a < EP_a
   ET = sum(E2(1:5));     C = E2/ET;
   for x = 1:N_CT
       for y = 1:N_CT
           [UinTot, S] = Sum_in(y, x, v, d, C, N_CT);
           u(y,x) = u(y,x) + tm_st*UinTot;
           v_new  = 0.5*(1.0+tanh(u(y,x)/u0));
           del_v  = v_new- v(y,x);     v(y,x) = v_new;
           del(1) = del_v*S(1);        del(2) = del_v*S(2);
           del(3) = del_v*(S(3)+0.5*del_v);
           del(4) = del_v*(S(4)+0.5*del_v);
           del(5) = del_v*S(5);        E2 = E2 + del;
       end
   end
   n_EP_a = n_EP_a+1;
end
```

Script to find the balanced coefficients.

```
% run AHNN_B, j_max random cities, k_max initializations for each
clear, N_CT = 10,  j_max = 5, k_max = 2
for j = 1:j_max
    p = rand(N_CT,2);    d2 = zeros(N_CT,N_CT);
    for i=1:N_CT
        d2(:,i)=((p(i,1)-p(:,1)).^2+(p(i,2)-p(:,2)).^2);
    end
    d = d2.^(1/2);    EP_a  = 200;     tm_st = 0.1;    u0 = 0.1;
    for k = 1:k_max
        noise = rand(N_CT)-0.5;
        u = 0.2*noise + atanh(2.0/N_CT-1);
        C = AHNN_B(d, u, EP_a, tm_st, u0);
        CC(k,:)= C(1, :);
    end
    CM(j, 1:5) = max(CC(:, 1:5));
    Cm(j, 1:5) = min(CC(:, 1:5));
    C_ave(j, 1:5) = sum(CC(:, 1:5))/k_max;
end
C_ave,       CCC = sum(C_ave(:, 1:5))/j_max
```