This article describes a Hopfield-Tank model of the Euclidean traveling salesman problem (using Aiyer’s subspace approach) that incorporates a “fuzzy” interpretation of the rows of the activation array. This fuzzy approach is used to explain the network’s behavior. The fuzzy interpretation consists of computing the centroid of the positive activations in each row. This produces real numbers along the time line, one for each city, and defines a tour in a natural way (referred to as a “fuzzy tour”). A fuzzy tour is computed at each iteration, and examination of such tours exposes fine features of the network dynamics. For example, these tours demonstrate that the network goes through (at least) three phases: “centroid,” “monotonic,” and “nearest-city.” The centroid (initial) phase of the network produces an approximation to the centroid tour (i.e., the tour that orders the cities by central angle with respect to the centroid of the cities). The second phase produces an approximation to the monotonic tour (i.e., the tour that orders the cities by their projection onto the principal axis of the cities). The third phase produces an approximation to a nearest-city tour, that is, a tour whose length is between the tour lengths of the nearest-city-best (best starting city) and nearest-city-worst (worst starting city) tours. Simulations are presented for up to 125 uniformly randomly generated cities.

The Hopfield-Tank model [12, 13, 16, 17] of the Euclidean traveling salesman problem (TSP) [7, 8, 14] has attracted many researchers because it demonstrates how a highly interconnected network of simple processing units can solve a complex problem. The method is also interesting because it exemplifies a key property of parallel-distributed systems: the emergence of global characteristics from local interactions. [9, 10] It is often difficult to understand how local interactions can lead to a particular global effect, yet the analysis of such effects is crucial to improved understanding of recently developed models of biological, ecological, and economical systems. We take the TSP as a representative problem and show how a fuzzy approach exposes the network’s organizational properties.

We use Aiyer’s subspace method [8] to construct a particular Hopfield-Tank TSP model. Briefly put, the subspace method is based on Hopfield’s [2] original approach but with a more carefully chosen Energy Function. This is a direct consequence of analyzing the orthogonal projection onto the feasible (or valid) space. The result is a network that keeps the activation vector close to the feasible space, which in turn facilitates the interpretation of a final convergent state (i.e., the final state of the network corresponds closely to a permutation matrix). We show that for up to 100 uniformly randomly generated cities, the network produces tours of nearest-city quality (by “nearest-city quality” we mean that the tour length is between the tour lengths of the nearest-city-best (best starting city) and the nearest-city-worst (worst starting city) tours). The focus of this article, however, is to provide an explanation for the network’s behavior, that is, to better understand the underlying mechanisms by which the network chooses a tour.

To provide a finer analysis of the network’s behavior, we introduce a fuzzy interpretation of each row of the activation array. This consists of calculating the center of mass (cm) of the positive activations in each row. These cms are real numbers, and therefore not restricted to the integers that represent the columns of the array. Thus, each row, and therefore each city, is associated with a real number on the time line, and the corresponding sequence is interpreted as the current tour (i.e., the “fuzzy tour”). Such a tour can be computed at any iteration. At the beginning of a run, as might be expected, the fuzzy tours are random, but as the network progresses, various patterns emerge—patterns that correlate with city statistics, such as the centroid and principal components. It is shown that the network goes through three phases:

1. Centroid Phase. The fuzzy tours are approximations to the centroid tour, that is, the tour that orders the cities by central angle with respect to the center of mass of the cities.
2. Monotonic Phase. The fuzzy tours are approximations to the monotonic tour, that is, the tour that orders the cities by their projections onto the principal axis of the cities.
3. Nearest-City Phase. The fuzzy tours are approximations to a nearest-city tour, that is, a tour that starts from an arbitrary city and then chooses nearest neighbors until a tour is constructed.

In simulations the network is initialized with very small random positive activations (less than $10^{-30}$). Initially, the fuzzy tours are themselves random but they quickly become approximations to the centroid tour. The emergence of the centroid tour is surprising, but it was first suggested by Aiyer. [1] We do not provide a proof that centroid tours must emerge, but it is evident in almost all simulations, with rare exceptions. However, a proof is provided that for small activations the network produces sinusoids of frequency $1$ in each row of the activation array, and that the peaks of the
sinusoids cluster around 2 columns of the network (half way around the tour from each other). Empirical evidence is then presented that the peaks of the sinusoids organize themselves so that the corresponding fuzzy tour is a close approximation to the centroid tour.

The centroid configuration is just the initial phase, a manifestation of the linear behavior of the network (dominant when the activations are small). The corner-seeking (i.e., nonlinear) tendency of the network ultimately forces the activations to move toward a nearest-city tour. In-between, there is a phase where the fuzzy tours correlate strongly with the monotonic tour. The relationship between the monotonic and nearest-city phases is critical to understanding the network’s overall performance. Unfortunately, it is very difficult to see the correlation between these phases, but we provide strong empirical evidence that the principal axis of the cities plays an important role in tying these two phases together.

1. Network Architecture

Following Hopfield and Tank,[2] an n-city TSP problem is represented by an n × n grid of neurons (cities at possible time stops). A row of the grid represents a city at all time stops; a column of the grid represents a time stop at all cities. The neighboring columns of the grid are connected (symmetrically) with a connection strength of −d(x, y), where d(x, y) is the distance between cities x and y. That is, neuron (i, x) is connected to neuron (i ± 1, y), with wrap around, with connection strength = −d(x, y).

Each neuron has an activation (aij), and we refer to the n × n array of activations as the activation array and to the n²-long vector obtained from the array (by concatenating the rows) as the activation vector (A). The index “i” indicates the column of the array (i.e., the “stop”), and the index “x” indicates the row of the array (i.e., the “city”). The net input to a neuron is the sum of activations times the connection strengths (there are no external inputs):

\[ \text{net}_i = \sum_j \sum_k w_{ik} a_{kj} \]  

1.1 The Subspace Approach

So far, the network architecture is too simple to be effective. Following Aiyer,[1] we analyze the orthogonal projection into the feasible space. The Hopfield-Tank model defines the feasible space as the affine space containing the n × n matrices whose rows and columns sum to 1 (i.e., doubly stochastic matrices). In our model the feasible space (Fr) is the space spanned by n × n matrices whose rows and columns sum to zero (Aiyer refers to this as the “zero-sum” space).

The Hopfield-Tank model uses an external input to shift (i.e., parallel translation) Fr to the affine space containing the doubly stochastic matrices—otherwise the neural architectures are identical.

The orthogonal projection consists of the following calculation:

\[ a_{ix}^{proj} = a_{ix} + \text{act}_{avg} \cdot \text{row}_{avg} - \text{col}_{avg} \]  

The “act_{avg}” is the average of all the activations at that iteration; row_{avg} is the average of the activations in the xth row; col_{avg} is the average of the activations in the yth column.

After projection, the array (a_{ix}^{proj}) has the property that the sum of any row or column is zero and is therefore in the feasible space.[6]

The effectiveness of the projection is a direct consequence of the analysis provided by Aiyer[1] (see also Gee et al.[3] and Aiyer et al.[4]). It helps to realize that R^n decomposes into the following mutually orthogonal subspaces:

\[ R^n = \mathcal{F}_0 \oplus \mathcal{E}_r \oplus \mathcal{E}_c \oplus \mathcal{D} \]  

where \( \mathcal{D} \) is the one-dimensional space spanned by a constant vector (e.g., a_{ij} = 1); \( \mathcal{E}_r \) is spanned by harmonics in the x-direction (e.g., sinusoids parallel to the rows of the neural grid); \( \mathcal{E}_c \) is spanned by harmonics in the y-direction (e.g., sinusoids parallel to the columns of the neural grid); and \( \mathcal{F}_0 \) (the feasible space) is spanned by the two-dimensional harmonics that are orthogonal to \( \mathcal{D}, \mathcal{E}_r \), and \( \mathcal{E}_c \).

With these definitions the feasible space becomes a linear subspace of R^n (further details are provided in Aiyer,[1] Wolfe et al.[5], and Wolfe and Ulmer[6]).

The projection is introduced here, separate from the rest of the network architecture, for reasons of clarity, but the projection calculation can be easily embedded in the neural architecture introduced above. To do this we add to the network connections: add 1/n² to all connections; add −1/n to all connections between neurons in a common row; add −1/n to all connections between neurons in a common column. Notice that this results in self-connections of −(2n − 1)/n². These additional connections, in effect, allow the activations to be incrementally projected as part of the neural update sequence.

As pointed out by Aiyer,[1] in this model we do not have variables equivalent to the “A, B, C, D” as in the original Hopfield-Tank model.[2] That is, the subspace approach specifies these parameters unambiguously.

Putting these definitions together we get the following network architecture:

\[ w_{ij}^{proj} = \begin{cases} 1/n^2 - 1/n & y = x, j \neq i; \\
1/n^2 - 1/n & y \neq x, j = i; \\
1/n^2 - 2/n & y = x, j = i; \\
1/n^2 - d(x, y) & y \neq x, j = i + 1, or j = i - 1; \\
1/n^2 & j \neq i - 1, i, i + 1, and y \neq x; \end{cases} \]

The energy function is given by:

\[ E = -1/2 \left\{ \sum_i \sum_j \sum_k a_{ij} a_{kj} w_{ij}^{proj} \right\} \]

\[ = -1/2 \left\{ \sum_i \sum_j \sum_k (-d(x, y) a_{ix} a_{jy} + a_{ix} a_{jy}) \right\} \\
+ \sum_i \sum_j \sum_k (-1/n) a_{ij} a_{iy} + \sum_i \sum_j \sum_k (-1/n) a_{ij} a_{iy} \]

\[ + \sum_i \sum_j \sum_k (1/n^2) a_{ij} a_{iy} \]
We have found that substituting \( d(x, y) / n \) for \( d(x, y) \) in the weights puts more emphasis on seeking the feasible space in the early phase of the network dynamics and leaves the “distance effect” for later (similar to setting \( D = 1/n \), and \( A = B = C = 1 \), in the original Hopfield-Tank model).

Let:
\[
M = \text{maximum activation} = +1; \quad m = \text{minimum activation} = -1/(n - 1).
\]

Call \((a_i)\) a feasible state if there is exactly one maximum activation in each row and column, and the rest of the entries are equal to the minimum activation. Feasible states correspond to tours in the usual way. Also notice that with \( M = +1 \) and \( m = -1/(n - 1) \) the sum of any row or column of a feasible state is 0 (i.e., \((a_i) \in \mathcal{F}_0\)). (See the Appendix for a proof that the feasible states have lower energy when they correspond to shorter tours.)

### 1.2 Network Dynamics

Assume that we have \( n \) neurons, indexed by \( i = 1, \ldots, n \), with activations \( a_i \), such that \( m \leq a_i \leq M \), connection strengths \( w_{ij} \) and no external inputs. We define the Hopfield-Tank dynamics by the following equations:

\[
du_i/dt = -\mu u_i + \text{net}_i,
\]

where \( \text{net}_i = \sum_j w_{ij} a_j; \quad a_j = g(u_j); \quad g(x) = (M - m)/(1 + \exp(-x)) + m. \)

It is easy to show that when \( \mu = 0 \) these equations can be approximated by the following (see the Appendix for the proof):

\[
da_i/dt = (a_i - m)(M - a_i)\text{net}_i.
\]

Setting \( \mu = 0 \) is usually a good idea because it represents a decay factor that impedes the convergence of the network to the boundary of the hypercube (see Takefuji[15] for further justification).

### 2. City Statistics

Treating the coordinates of the cities as data points in a two-dimensional feature space, the following statistics are defined: centroid, correlation matrix, principal components, and effective partition. The centroid, correlation matrix, and principal components are standard[12] but the concept of an effective partition is introduced here because it helps explain the network’s behavior.

Let \( S = \{(a_i, b_i)|0 \leq a_i, b_i \leq 1, i = 1, \ldots, n\} \) be the set of city coordinates. Let \( (a_{\text{avg}}, b_{\text{avg}}) \) be the centroid of the cities:

\[
a_{\text{avg}} = \frac{\sum_i a_i}{n} \quad \text{and} \quad b_{\text{avg}} = \frac{\sum_i b_i}{n}.
\]

The correlation matrix is given by:
\[
C = (c_{ij}), \quad i, j = 1, 2;
\]

where \( c_{11} = 1/n \sum_i (a_i - a_{\text{avg}})^2; c_{12} = c_{21} = 1/n \sum_i (a_i - a_{\text{avg}})(b_i - b_{\text{avg}}); \quad \text{and} \quad c_{22} = 1/n \sum_i (b_i - b_{\text{avg}})^2. \)

Let \( \mathbf{v}_1, \mathbf{v}_2 \) and \( \lambda_1, \lambda_2 \) be the solution to the eigenvalue problem: \( \mathbf{C} \mathbf{v} = \lambda \mathbf{v} \). Since \( \mathbf{C} \) is a correlation matrix the eigenvalues are real and nonnegative, and the eigenvectors are orthogonal. Assume \( \lambda_1 \geq \lambda_2 \). The eigenvalues, \( \lambda_1 \) and \( \lambda_2 \), measure the variance of the points when projected onto \( \mathbf{v}_1 \) and \( \mathbf{v}_2 \), respectively. The direction of \( \mathbf{v}_1 \) corresponds to the direction with maximum variance. The “principal axis” of the cities is defined by placing \( \mathbf{v}_1 \) at the centroid.

Let \( [G_1, G_2] \) be a partition of \( S: G_1 \cup G_2 = S, G_1 \cap G_2 = \emptyset \). Call it an “effective” partition if it has the property that the sum of the distances from a city in \( G_1 \) to all other cities in \( G_1 \) is less than the sum of its distances to the cities in \( G_2 \), and similarly for a city in \( G_2 \). That is, \( [G_1, G_2] \) is an effective partition if:

\[
\sum_{j \in G_1} d(j, k) < \sum_{j \in G_2} d(j, k) \quad \text{if} \quad k \in G_1,
\]

\[
\sum_{j \in G_1} d(j, k) \geq \sum_{j \in G_2} d(j, k) \quad \text{if} \quad k \in G_2.
\]

Such partitions are relatively easy to find. For example, if the cities are uniformly spaced on a circle, any diameter that does not intersect a city defines an effective partition. In general, an effective partition can be created by seeking a local minimum to a specific objective function (see the Appendix for a proof).

A simple 10-city example is provided in Figure 1. For a large number of cities uniformly generated within a unit square, the principal axis will (approximately) lie along one of the diagonals of the square, and the bisector of the principal axis (i.e., \( \mathbf{v}_2 \)) will (approximately) separate the cities into an effective partition.

### 3. TSP Heuristics: Centroid, Monotonic, Nearest-City, 2-Opt

Using the notation from the previous section, we can now define, precisely, the TSP heuristics that we will use in
evaluating the neural network. The centroid and monotonic tours are nonstandard, so we define them here. The nearest-city tour is a standard approach to the TSP,[7, 14] but our use of the terms “nearest-city-best” and “nearest-city-worst” is not common, so we also define these. The 2-Opt is another standard approach to the TSP,[7, 14] and we include its definition for the sake of completion.

3.1 Centroid Tour
For each city \( i \) (location \((a_i, b_i)\)), let \( \theta_i = \arctan((b_i - b_{avg})/(a_i - a_{avg})) \). Sort the cities into the sequence \((i_1, i_2, \ldots, i_n)\) by \( \theta_i \leq \theta_{i+1} \).

3.2 Monotonic Tour
Let \( v_1 = (a_i, b_i) \) be the principal axis of the cities. For each city \( i \), let \( \pi_i = (a_i - a_{avg})v_1 + (b_i - b_{avg})v_2 \). Sort the cities into the sequence \((i_1, i_2, \ldots, i_n)\) by \( \pi_i \leq \pi_{i+1} \).

3.3 Nearest-City Tour
For each possible starting city, construct a tour by choosing the nearest unvisited city for the next city until all the cities are exhausted and return to the starting city. The length of the shortest of these tours is defined to be the “nearest-city-best” result, and the longest of these tours is defined to be the “nearest-city-worst” result.

3.4 2-Opt
Begin with a random tour. Consider all possible pairs of edges in the tour. For each pair delete the edges from the tour and add 2 new edges to reconnect the cities into a tour. After checking each edge pair, pick the one that improves the tour the most, and iterate. If none of them improves the tour, stop.

4. Fuzzy Read Out
Assume that \((a_i)\) represents the activations, with “\( i \)” representing the column and “\( x \)” the row of the array. We introduce a “fuzzy” way to read each row of the activation array. We compute the center of mass of the positive activations and treat that as an estimate of where the city represented by that row should be placed along the time line.

Let:

\[ a_{ix} = a_{ix} \quad a_{ix} > 0; \]
\[ a_{ix} = 0 \quad a_{ix} \leq 0. \]

For each row \((x)\) of the activation array let:

\[ cm_x = \sum_i (a_{ix}) / \sum_i a_{ix}. \]

\( cm_x \) is undefined when there is no positive activation in the row. Also, to account for “wrap around” on the time line, the \( cm \) is computed relative to the position of the peak activation in the row. If \( i_{peak} \) is the column with the largest activation, then the sums are performed from \( i = i_{peak} - (n/2 - 1) \) to \( i = i_{peak} + (n/2 - 1) \) (with \( a_{ix} = a_{(ix \mod n)} \) and \( cm_x = [cm_x \mod n] \)). The results are not significantly affected by summing from \( i_{peak} - b \) to \( i_{peak} + b \), where \( b \) is a small fraction of \( n \). We used this method of computing the cms in all the simulations presented in this article, but it is worth pointing out that a more natural computation (i.e., one that exploits the cyclic, translation invariance, of the system) may be used. That is, similar results can be obtained by defining the cm to be the phase of the first Fourier coefficient:

\[ cm_x = \arctan\left( \frac{\sum_i a_{ix}\sin(2\pi i/n)}{\sum_i a_{ix}\cos(2\pi i/n)} \right). \]

An important feature of either of these methods is that the cms are real numbers, and therefore not restricted to the integers that represent the columns of the neural array (producing what might be called “sub-pixel” accuracy). The cms are computed for each row and sorted along the time line. Since each cm corresponds to a row, and each row to a city, a tour is easily deduced from the sequence of cms.

If: \( cm_{x1} \leq cm_{x2} \leq \ldots \leq cm_{xn} \)

Fuzzy Tour: \( x_1, x_2, \ldots, x_n \).

This definition is ambiguous when two or more cms are equal. Equal cms represent the boundary between regions of hyperspace representing different fuzzy tours. In simulations, these boundaries are often crossed but convergence to states on such boundaries is extremely rare, and usually corresponds to having collocated cities.

The fuzzy approach gives meaning to “interior” states of the hypercube. A tour is now represented by a relatively large region of hyperspace containing, for example, the line from the origin (but not including the origin) to the tour’s hypercube corner (i.e., permutation matrix). For a given tour \( \tau \), let \( R_{\tau} \) be the region of hyperspace that \( \tau \) represents. \( R_{\tau} \) contains hypercube corners corresponding to permutation matrix representations of \( \tau \), including the redundant corners associated with rotated tours (same tour, but a different starting city) and reverse tours (same tour, but traversed in the opposite direction). \( R_{\tau} \) splits into disjoint subsets:

\[ R_{\tau} = R_{\tau}^1 \cup R_{\tau}^2 \]

where \( R_{\tau}^1 \) represents \( \tau \) with clockwise orientation, and \( R_{\tau}^2 \) represents \( \tau \) with counter-clockwise orientation.

It is easy to see that \( R_{\tau}^1 \) and \( R_{\tau}^2 \) are disjoint \((R_{\tau}^1 \cap R_{\tau}^2 = \emptyset)\) and each is a connected set. That is, there is no hyperspace point that is in both \( R_{\tau}^1 \) and \( R_{\tau}^2 \). This is clear since the order of the cms would have to be completely reversed to move from \( R_{\tau}^1 \) to \( R_{\tau}^2 \). The only points in hyperspace that might represent a tour in both \( R_{\tau}^1 \) to \( R_{\tau}^2 \) are the points corresponding to all equal cms. These points are ambiguous and, by definition, in neither \( R_{\tau}^1 \) nor \( R_{\tau}^2 \). (Points where all the cms are equal are called maximally ambiguous since a small perturbation can cause the point to represent any possible tour.) By a continuous deformation of the activations, however, any point in \( R_{\tau}^1 \) can be moved into any other point in \( R_{\tau}^1 \), without leaving \( R_{\tau}^1 \), and the same is true of \( R_{\tau}^2 \).

Network dynamics can be viewed as moving the activation vector through these regions toward the hypercube boundary. Along the way it “sees” many different tours, and the best of these can be saved as the network output.
effectively resolved by the fuzzy read out.

Figure 2. Fuzzy read out for a 20-city example. The squares represent the level of neural activation. The sequence of cms produces the tour at the bottom. Notice that column competition between cities such as B and R is effectively resolved by the fuzzy read out.

Figure 2 shows a 20-city example. The fuzzy tour is computed (at the 500th iteration in this case) and drawn at the bottom of the figure. Notice that when activations for two rows compete for the same column (e.g., cities B and R), the cities are located close to each other, yet the cms unambiguously define a tour (see also cities D and S).

The main purpose of the cm calculation is to expose the stages of network behavior. We do not see how to add this computation to the neural architecture without corrupting the symmetric and homogeneous nature of the neurons. However, more neurons could be added to each row to (asymmetrically) accumulate the sums necessary for calculating the cms. This would retain the massively parallel nature of the network if implementation were the goal, but, again, the cms are used here primarily for diagnosis and not necessarily included in the neural architecture.

5. Phase I Dynamics: Centroid Tours

Analysis of a linear approximation to the network dynamics provides vivid insight into the network behavior when the activations are small. Since the activation vector is initialized in the interior of the hypercube (near the origin), and the incremental activation updates are small, the following linear equation approximates the network dynamics in the first stage of updates:

$$\frac{da}{dt} \approx Wa$$

where W is the $n^2 \times n^2$ connection strength matrix.

Based on the properties of W, it will be shown that sinusoids of frequency $= 1$, but arbitrary phase, must emerge in each row of the $n \times n$ “activation array” during the first phase of network dynamics. First, it is shown that the connection strength matrix W ($n^2 \times n^2$) is composed of symmetric circulant blocks $A_{xy}$ ($n \times n$), one for each pair of cities x, y. This implies that the eigenvalues of $A_{xy}$ are given by the discrete Fourier transform (DFT) of the first row of $A_{xy}$ and that the corresponding eigenvectors ($n \times 1$) are sinusoids of all frequencies (see Davis[11] for the properties of circulant matrices).

Let $\mathbf{a}$ be the “activation vector.” That is, $\mathbf{a}$ is an $n^2$-long column vector whose first $n$ components are the activations of the first row of the array of neurons, and so on. Therefore $\mathbf{a}$ can be written:

$$\mathbf{a} = \mathbf{v}_1\mathbf{1} + \mathbf{v}_2\mathbf{2} + \cdots + \mathbf{v}_n\mathbf{n},$$

where $\mathbf{v}_i$ is an $n \times 1$ column vector containing the activations of the $i^{th}$ row of neurons. To save space we also use the notation: $\mathbf{a} = (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n)$.

Let W be the connection strength matrix. W can be written as:

$$W = A_{11} A_{12} \cdots A_{1n}$$
$$A_{12} A_{22} \cdots A_{2n}$$
$$A_{1n} A_{2n} \cdots A_{nn}$$

where $A_{xy}$ is the $n \times n$ matrix that contains the connection strengths between the $x^{th}$ and $y^{th}$ rows of neurons. It is easy to see that $A_{xy} = A_{yx}$, and each $A_{xy}$ is a symmetric circulant matrix.

Now we invoke well-known theorems concerning symmetric circulant matrices (see Davis[11] and Aiyer[5]). In particular, we know that sinusoids of any frequency are eigenvectors of $A_{xy}$, and that the (real) eigenvalues are obtained by taking the DFT of the first row of $A_{xy}$.

Theorem 1. The eigenvalues of $A_{xy}$ are given by:

when $x \neq y$: 

$$\lambda_0 = -(2)d(x, y),$$
$$\lambda_p = -1/n - 2d(x, y)\cos(2\pi p/n),$$

$p = 1, \ldots, n - 1.$
when \( x = y \):

\[
\begin{align*}
\lambda_0 &= -1, \\
\lambda_p &= -1/n, \\
p &= 1, \ldots, n - 1.
\end{align*}
\]

**Proof.** When \( x \neq y \), the first row of \( A_{xy} \) is:

\[
1/n^2 - 1/n, \quad 1/n^2 - d(x, y), \quad 1/n^2, \ldots, \quad 1/n^2,
\]

\[
1/n^2 - d(x, y).
\]

When \( x = y \), the first row of \( A_{xy} \) is:

\[
1/n^2 - 1/n, \quad 1/n^2 - 1/n, \ldots, \quad 1/n^2 - 1/n.
\]

The corresponding DFTs are given by the coefficients in the Theorem. \( \square \)

Thus, except for the “DC” term (\( \lambda_0 < 0 \)), the eigenvalue with the largest magnitude corresponds to the first harmonic (\( p = 1 \)). The eigenvalue for the first harmonic is negative, and its magnitude is larger when \( d(x, y) \) is larger. In particular, notice that for \( x \neq y \), \( \lambda_1 \approx -2d(x, y) \), for large \( n \). So, without loss of generality we will make the identification:

\[
\lambda_1 = -d(x, y)
\]

for the rest of the analysis.

**Theorem 2.** Sinusoids of frequency \( = 1 \) will emerge in each row of the activation array as the network updates the activations.

**Proof.** Consider the coupling that occurs between the various \( A_{xy} \) as \( W \) acts on \( a \):

\[
(Wa)_k = \sum_j A_{kj} v_j
\]

Now let \( u_j \) be the first harmonic in the Fourier decomposition of \( v_j \):

\[
u_j = \langle \sin(2\pi r/n + \phi_j) | r = 0, \ldots, n - 1 \rangle
\]

By Theorem 1, the largest magnitude effect that \( A_{kj} \) has on \( v_j \) is on the first harmonic:

\[
A_{kj} v_j = -d(j, k) u_j
\]

Therefore \( (Wa)_k \) will be a superposition of sinusoids of frequency \( = 1 \) but with differing amplitudes and phases, which will produce another sinusoid of frequency \( = 1 \). Thus, at each neural iteration the network dynamics will add a relatively large sinusoid of frequency \( = 1 \) to each row of the activation array. \( \square \)

Notice that the proof of Theorem 2 implies that analyzing \( W \) reduces to analyzing \( D \) where \( D \) is the negative distance matrix \((-d(i, j))\) and \( z \) is an \( n \times 1 \) column vector with arbitrary complex exponentials in each component:

\[
z = \langle z_1, z_2, \ldots, z_n \rangle,
\]

\[
z_j = r \exp(\sqrt{-1} \phi).
\]

**Theorem 3.** Let the cities \( j \) and \( k \) be the two cities that are the furthest away among all the cities (i.e., cities \( j, k \), such that \( d(j, k) = \max_{x,y} d(x, y) \)). The network dynamics will force

the \( j \)th and \( k \)th rows of the activation array to be \( 180^\circ \) out of phase.

**Proof.** Consider the \( j \)th and \( k \)th components of \( Dz \):

\[
(Dz)_j = -\sum_p d(p, j) z_p = \ldots -d(k, j) z_k - \ldots
\]

\[
(Dz)_k = -\sum_p d(p, k) z_p = \ldots -d(j, k) z_j - \ldots
\]

The term \(-d(k, j) z_k\) will contribute a large multiple of \(-z_k\) to \( z_j\), and the term \(-d(j, k) z_j\) will contribute a large multiple of \(-z_j\) to \( z_k\). This will cause the phases of \( z_j \) and \( z_k \) to move in opposite directions (see Figure 3). \( \square \)

From Theorem 3 it is easy to see that, in general, the network dynamics will adjust the phase of each row of the activation array so as to “oppose” the distance-weighted effect of the other rows (see Figure 4).

**Theorem 4.** Let \( \{G_1, G_2\} \) be an effective partition of the set of cities, and let \( z \) have the property that \( z_k = z \) if \( k \in G_1 \), and \( z_k = -z \) if \( k \in G_2 \). The network dynamics will then multiply each component of \( z \) by a positive scalar, and therefore maintain the phase relationships between the rows of the activation array.
Thus, each component of \( z \) is scaled in proportion to the corresponding difference in sum of distances to \( G_1 \) and \( G_2 \), which, by the definition of an effective partition, must be positive.

These Theorems predict that sinusoids of frequency \( = 1 \) will emerge in the rows of the activation array in the network’s early stages, and Theorem 4 indicates that the sinusoids will group themselves according to an effective partition. That is, rows that correspond to the cities in \( G_1 \) will be approximately 180° out of phase with rows that correspond to cities in \( G_2 \). Furthermore, simulations will show that for uniformly randomly generated cities the cities are roughly divided into \( G_1 \) and \( G_2 \) by the line that is perpendicular to the principal axes of the cities at the centroid.

What is left to verify is that the network dynamics will organize the peaks of these sinusoids in such a way that their order corresponds to the centroid tour. This is difficult to prove, so we present only empirical evidence in the next section.

6. Network Performance: 50-City Example

The following simulations validate the theoretical results of the previous sections. In particular, the simulations show that in the network’s initial phase:

- sinusoids of frequency \( = 1 \) emerge in each row of the array;
- the peaks of the sinusoids are arranged in centroid order;
- the peaks form 2 clusters, centered about 2 columns of the array;
- the clusters are approximately half way around the tour from each other;
- the clusters correspond to an effective partition of the cities.

The simulations also show that the network moves through three phases: centroid, monotonic, and nearest-city. It is difficult to explain, to any degree of satisfaction, the relationship between the phases. It does appear, however, that the monotonic phase creates a biased introduction to the final, nearest-city, phase. Therefore, a better understanding of the monotonic phase might lead to network improvements.

Figures 5A–D show a 50-city example. The city locations are randomly placed in the unit square. The 2-opt, nearest-city-best, nearest-city-worst, centroid, and monotonic tours are computed before running the neural network (see Figure 5A). Notice that the tour lengths are given in the figure and that the 2-opt is the better of the tours, with the nearest-city-best the next best. In terms of tour length, the monotonic and centroid tours are relatively poor. (The monotonic and centroid tours get increasingly worse, in comparison to the 2-opt and nearest-city-best, as the number of cities increases.) To facilitate the interpretation of phase 1, the cities are ordered according to the centroid result before launching the neural network. This makes the effective partition “jump out” of the sketch of the activations.

Figure 5B shows the fuzzy tours, and activations, at iteration 10, 280, and 550. At iteration 280 the network is clearly in the centroid phase: the fuzzy tour is nearly identical to the centroid tour, the activations are sinusoidal across each row, they form 2 clusters, the clusters correspond to an effective partition of the cities, and the partition has one group of cities at each end of the principal axis. (In Figures 5B and 5C the activations are “drawn to scale.” That is, the largest box drawn represents the largest activation currently in the array, which might be a very small activation value, especially in the initial phase.) At iteration 550 the network reaches the monotonic phase: the fuzzy tour correlates strongly with the monotonic tour. These phases emerge in almost all simulations, and the monotonic phase is typically less accurate and briefer than the centroid phase.

Notice that the clustered sinusoids create closely spaced cms in each of the two groups, yet, as the fuzzy tour shows, there is enough resolution to specify the centroid tour unambiguously. As was noted earlier, equal cms are ambiguous, and the state consisting of all equal cms is maximally ambiguous. Considering how close the cms are in the centroid phase it is not that surprising that the network moves to a significantly different set of tours in the monotonic phase within just a few iterations.

The monotonic phase sets the stage for the nearest-city phase. It appears that it creates a bias that affects the nearest-city phase. Figure 5C shows the final phase of the network: nearest-city tours. In this case the network beat the nearest-city-best result, but notice that the tours in Figure 5C show a tendency to snake along the principal axis. This phenomenon is present in almost all simulations, and it seems to be the source of increasingly poor performance as the number of cities increases.

Figure 5D shows the fuzzy tour lengths for iteration 1 through 1000. The three phases of the network are evident. Notice that the fuzzy tour lengths do not reach the monotonic tour length (Figure 5B shows that the fuzzy tours are only an approximation to the monotonic tour). This appears to be the result of the fact that the centroid phase has partitioned the cities into 2 groups and the monotonic effect is acting on each group separately.

Figure 5D also shows that the fuzzy tour lengths are relatively stable in the middle of the centroid phase, but they are erratic in the monotonic phase, and they vary only slightly in the nearest-city phase. This is typical for the runs we did.

7. Network Performance: 10 to 100 Cities

Figure 6 shows the results of averaging 50 runs for each problem size from 10 to 70 cities. For any neural run we
define the “neural result” to be the best of all the fuzzy tours that are encountered in the run. Although we do not keep track of the phase that produced the best tour, in most cases the best tour is produced in the nearest-city phase, but for 10 to 30 cities it is often the case that the centroidal phase produces the best tour. For a small number of cities (10–20), the neural result competes with the 2-opt and nearest-city-best. This is correlated with the fact that, for 10 to 20 randomly generated cities, centroid tours also compete very closely with the 2-opt and nearest-city-best. That is, the network gets to pick the better of the centroid and nearest-city solutions. Although the neural results reliably fall between the nearest-city-best and nearest-city-worst results, there is some indication that the results are worsening as the number of cities grows, possibly because of the dramatically increasing length of the average centroid and monotonic tours as the number of cities grows.

Figure 7 provides additional data summarizing many simulations. It shows the number of times the neural result (i.e., the best of all the fuzzy tours for that run) was better than the 2-opt, nearest-city-best, or nearest-city-worst for 50 instances of city sizes 25, 50, 70, and 100. It is clear that the neural results are slowly degrading, although even at 100 cities the neural result is better than the nearest-city-worst approximately 80% of the time. The degradation in performance appears to be a consequence of the fact that centroid and monotonic tours degrade very rapidly as the number of cities increases.

Figure 8 shows two 125-city runs. Notice that the neural tours tend to snake around the diagonal of the city location box (the principal axis is approximately along one of the diagonals). The neural results are better than the nearest-city-worst results approximately 50% of the time for 125 randomly generated cities.

8. Special Cases: Circular and Linear City Configurations

8.1 Circular Cities

It is important to realize that the simulations presented above were based on (uniformly) randomly generated cities in a square. The network does much better on circular and linear configurations of cities. For example, for up to 40 cities that are uniformly placed on a circle the network almost
always gets the optimal solution. This is attributable to the centroid phase, since it seeks the centroid solution that is optimal in circular cases.

For up to 100 cities equally spaced on a circle the network often finds the optimal solution. Furthermore, a simple variation of the network architecture produces 100% optimal
solutions for such cases. That is, add connections to columns beyond 1 column away from any neuron:

connect neuron \((i, x)\) to neuron \((i \pm q, y)\), for \(q = 1, \ldots, q_{\text{max}}\)
with wrap around, with a connection strength of \(-d(x, y)\).

Thus, \(q_{\text{max}}\) is the number of columns away that we connect a given neuron, using the same distance-weighted connections we would use for a single column away. For the simulations presented in the earlier sections \(q_{\text{max}}\) was set to 1. In this section we examine the network performance for larger values of \(q_{\text{max}}\).

When we increase \(q_{\text{max}}\) we observe increasingly better performance on circular cities. Figure 9 shows the improvement in performance as \(q_{\text{max}}\) ranges from 1 to 3. For larger

Figure 5c. The nearest-city phase. The sinusoidal clusters break up and the activations form a pattern consistent with a nearest-city tour. From iteration 655 to 995 the network refines the nearest-city tour to the point where it is competitive with the nearest-city-best (7.66). Notice that these nearest-city tours tend to “snake” around the principal axis.
values of $q_{\text{max}}$ the network found the optimal solution on all simulations (up to 100 cities). Figure 10 shows a 100-city example with $q_{\text{max}} = 20$. In this case it took only 26 iterations to reach the optimal solution. In almost all cases, with $q_{\text{max}} = n/5$ the network finds the optimal solution within 50 iterations.

It is relatively easy to see that Theorem 1 can be generalized to the modified architecture with the result that the eigenvalue with the largest magnitude is now $\lambda_1 \approx -2 \cdot q_{\text{max}} d(x, y)$. This enhances the centroid affect in the first phase of the network and helps the network find optimal solutions for circular configurations.

The affect of this modified architecture on randomly generated cities is still under investigation, but initial simulations indicate that it does not improve overall network performance despite the fact that it improves the performance on circular cities.

9.2 Linear Cities
To test linear configurations of cities we placed cities equally spaced along a line. For up to 100 cities (as far as we tested) the network, without modification (i.e., $q_{\text{max}} = 1$) produced the optimal solution very quickly (less than 50 iterations) on all the runs that we did (dozens of runs for each problem size from 10 to 100 cities). In this case the monotonic phase of the network provides the good performance, as might be expected since the monotonic tour is optimal. We also modeled “near-linear” city configurations by restricting one of the city coordinates to the range $[0.45, 0.55]$ (see Figure 11). In these cases the neural results were much better than the results shown in Figure 6 for uniformly generated cities in the unit square. This is explained by the fact that the deleterious effects caused by the “snaking” along the principal axis are minimized for near-linear city configurations.

Finally, it is interesting to note that the centroid phase is absent, or very brief, for near-linear cities, and similarly, the monotonic phase is absent for circular cities. It appears that the centroid and monotonic phases express the degree of “circular-ness” and “linear-ness” of a randomly generated set of cities.

9. Conclusions
We have shown that a neural network, using an implementation of Aiyer’s subspace method and a fuzzy approach to reading the rows of the array, can reliably produce nearest-city quality tours for up to 100 randomly generated cities. It has been demonstrated that the fuzzy approach reveals some of the inner workings of the network, exposing cen-
troid, monotonic, and nearest-city phases. It is unclear, however, exactly how the phases affect each other. It appears that the network adopts a bias as it passes through a phase, but it is difficult to correlate a subsequent phase with the previous phase. The principal axis plays a major role in the evolution of the network, and it is clear that cities with a larger projection onto the major axis have larger activations during the centroid phase. It is also clear that the effective partition forms on opposing ends of the principal axis. Furthermore, the nearest-city phase shows a strong tendency to snake around the principal axis. A deeper analysis of these relationships might lead to further improvements in network architecture.

Although further analysis of the 3 network phases may suggest strategies for an improved network, it remains plausible that the network is limited by its nearest-city behavior. That is, in the nearest-city phase the network appears to grow separate small, locally short, paths of cities, but fails to organize them into a good overall tour. This is at least partly because each of these short paths grows with a specific direction, and when they meet up the directions may be incompatible with a good tour. For example, the 2-opt heuristic allows for reversing the direction of any sub-sequence.

Figure 6. The plot shows the average of 50 random city runs for city sizes of 10 through 70. The neural results reliably fall between the nearest-city-best and nearest-city-worst results. The neural results, however, show signs of worsening with problem size, possibly because of the significant worsening of the centroid tours as the number of cities increases.

Figure 7. These histograms show the number of times the neural result was better than the 2-opt, nearest-city-best, or nearest-city-worst for 50 runs of 25, 50, 70, and 100 random cities. There is a noticeable degradation of the neural results as the number of cities increases, but even at 100 cities the neural result beat the nearest-city-best twice (but it was also worse than the nearest-city-worst tour 11 times).
of cities within the current tour. This is the source of much of its power. The network, on the other hand, has no mechanism for such reversals of subpaths, and the number of subpaths that might grow independently appears to increase with the number of cities, a fact that is consistent with degrading performance as the problem size grows.

Finally, there may be some value in applying the fuzzy read out method to other optimization networks, especially in cases where the network often converges to ambiguous states.

Figure 8. Two 125-city cases, showing the nearest-city-worst and neural tours. The top pair of tours shows the nearest-city-worst (tour length = 11.63) and the neural tour (tour length = 11.88), while the bottom pair of tours shows another example with tour lengths 12.19 and 11.20. For 125 randomly generated cities the neural tour is shorter than the nearest-city-worst tour about 50% of the time.

Acknowledgments

Professor Ross M. McConnell, of the University of Colorado at Denver, made major contributions to the development of the ideas in this article. Thanks to the anonymous IEEE and INFORMS reviewers who made important observations about earlier versions of this article. Further thanks to Professors Jay Rothman, Gita Alaghband, Harvey Greenberg, and Steve Billups, and students Ed Chang, Frank Ducca, Mike Pitt, Ben Bahr, John Porter, Dan Szabo, Brad Swanson, Ross Thomason, Brian Powell, Donovan Nelson, Joe Gee, Jeff Donovan, Jean Watson, Jonathan McSwain, Jiang Xu,
are equal to the minimum activation. Feasible states correspond to tours in the usual way. Also notice that with \( M = +1 \) and \( m = -1/(n - 1) \) the sum of any row or column of a feasible state is 0 (i.e., \((a_{ij}) \in \mathcal{F}_0\)). Now we show that feasible states have lower energy when they correspond to shorter tours.

**Theorem A1.** Feasible states have energy given by:

\[
E = \beta((n - 2)/(n - 1)^2) + (n/(n - 1))^2\text{TL},
\]

where \( \beta = \sum_x \sum_y d(x, y) \), and TL is the length of the tour defined by the feasible state.

**Proof.** Assume that \((a_{ij})\) is a feasible state. By direct computation:

\[
\sum_i \sum_x (1/n) a_{ij} a_{jp} = \sum_i \sum_x \sum_y (-1/n) a_{ij} a_{yp} = 0.
\]

Thus, the energy of a feasible state is given by the first term in the energy equation:

\[
E = -1/2 \sum_i \sum_y (-d(x, y)) a_{ij}(a_{i+1y} + a_{i-1y}).
\]

Let \( q \) be the row that has activation = 1 in the \( i^{th} \) column of the array (i.e., \( a_{iq} = 1 \)), and let \( y_1 \) and \( y_2 \) be the rows on the left and right that also have activation = 1 (i.e., \( a_{i-1y_1} = 1 \), and \( a_{i+1y_2} = 1 \)). Then we can expand:

\[
\sum_i \sum_y (-d(x, y)) a_{ii}(a_{i+1y} + a_{i-1y})
\]

\[
= -\sum_{z \neq y_1} (m)(1)d(q, z) - \sum_{z \neq y_2} (m)(1)d(q, z) - (d(q, y_1) + d(q, y_2))
\]

\[
+ \sum_{z \neq y_1} d(r, z)m^2 - \sum_{z \neq y_2} d(r, z)m^2 - m(d(q, y_1) + d(q, y_2))
\]

\[
= -2ms_q + (m - 1)(d(q, y_1) + d(q, y_2)) - 2m^2(\beta - s_q) + m(m - 1)(s_{y_1} + s_{y_2}) - m(m - 1)
\]

\[
\cdot (d(q, y_1) + d(q, y_2))
\]

where \( s_q \) is the sum of the distances from \( q \) to all other cities.

Now summing over all the columns:

\[
E = -1/2 \sum_i \sum_y (-d(x, y)) a_{ii}(a_{i+1y} + a_{i-1y})
\]

\[
= -1/2(-2m^2(n - 1)\beta + m(m - 1)(2\beta) - 2m(m - 1)\text{TL} + 2m\beta + 2(m - 1)\text{TL}).
\]

Finally, substituting \( m = -1/(n - 1) \):

\[
E = \beta(n - 2)/(n - 1)^2 + (n/(n - 1))^2\text{TL}.
\]

---

**Figure 9.** In this experiment the cities were equally spaced on a circle, and ten runs were made for each problem size from 10 to 100 cities. The number of optimal solutions in the ten runs is plotted. For \( q_{\max} = 1 \) the network produced all optimal solutions for up to 30 cities; from there performance degrades to only 3 optimal solutions at 100 cities. For \( q_{\max} = 3 \), however, the (modified) network produced all optimal solutions except for 1 run at 90 cities.

**Figure 10.** In this example 100 cities were equally spaced on a circle. The network was initialized with random activations and \( q_{\max} = 20 \). After only 26 iterations the network found the optimal solution.

Kenny Simler, Lincoln Carlton, Mike Brockway, Qibin Tao, Thomas Sheperd, Janet Siebert, Robert Keeble, and Libby Pflum of the University of Colorado at Denver, and Steve Sorensen of Hughes Information Technology Systems, for their insightful comments concerning the technical details of this article. And finally, thanks to Cheryl Dwyer for technical support.

**Appendix**

Let \((q_{ij})\) be a feasible state if there is exactly one maximum activation in each row and column, and the rest of the entries
Therefore:

\[ G_1. \text{ Arbitrarily assign the cities to a partition} \{G_1, G_2\} \]

\[ G_2. \text{ Loop over the cities:} \]

\[ a_i = (M - m)/(1 + \exp(-u_j)) + m \]

for \( u_j \), gives:

\[ u_j = \ln((a_i - m)/(M - a_i)). \]

Therefore:

\[ du_j/dt = (a_i - m)(M - a_i)/\exp(-u_j)du_j/dt. \]

Dropping the scale factor \( 1/(M - m) \), and substituting \( du_j/dt = \text{net}_j \), gives:

\[ da_i/dt = (a_i - m)(M - a_i)\text{net}_i. \]

\[ \text{Theorem A2.} \text{ The Hopfield-Tank model can be approximated by the equation:} \]

\[ da_i/dt = (a_i - m)(M - a_i)\text{net}_i. \]

\[ \text{Proof.} \text{ Setting } \mu = 0, \text{ and solving the equation} \]

\[ a_i = (M - m)/(1 + \exp(-u_j)) + m \]

for \( u_j \), gives:

\[ u_j = \ln((a_i - m)/(M - a_i)). \]

Therefore:

\[ du_j/dt = (a_i - m)(M - a_i)/\exp(-u_j)du_j/dt. \]

Dropping the scale factor \( 1/(M - m) \), and substituting \( du_j/dt = \text{net}_j \), gives:

\[ da_i/dt = (a_i - m)(M - a_i)\text{net}_i. \]

\[ \text{Theorem A3.} \text{ Given an arbitrary set of cities, an effective partition can be constructed by the following procedure:} \]

1. Arbitrarily assign the cities to a partition \( \{G_1, G_2\} \).
2. Loop over the cities:
   a. If a city violates the condition for an effective partition, put it in the other group.
   b. Stop when no city violates the condition.

\[ \text{Proof.} \text{ Let Score}(G_1, G_2) \text{ be the } \text{“score”} \text{ of the partition } \{G_1, G_2\}, \text{ defined as:} \]

\[ \text{Score}(G_1, G_2) = \sum_{i \in G_1} \left( \sum_{j \in G_1} d(j, k) - \sum_{j \in G_2} d(j, k) \right) \]

\[ -\sum_{j \in G_2} \left( \sum_{i \in G_1} d(j, k) - \sum_{i \in G_1} d(j, k) \right). \]

The procedure will move cities between \( G_1 \) and \( G_2 \), and in each move the Score will decrease. For example, if city \( p \) is in \( G_1 \) but \( \sum_{j \in G_1} d(j, p) - \sum_{j \in G_2} d(j, p) > 0 \), then moving city \( p \) to \( G_2 \) causes the Score to decrease:

\[ \Delta \text{Score} = 2 \left( \sum_{j \in G_2} d(j, p) - \sum_{j \in G_1} d(j, p) \right) < 0. \]

Therefore, the procedure will continue to decrease the Score until an effective partition is reached.  

\[ \text{References} \]


