AFIT/GSO/ENG/92D-01

COMPARISON OF ARTIFICIAL NEURAL NETWORKS TO A CONVENTIONAL HEURISTIC TECHNIQUE FOR OPTIMIZATION PROBLEMS

THESIS

Jeffrey Scott Gruner
Captain, USAF

AFIT/GSO/ENG/92D-01

AFIT/GSO/ENG/92D-01

COMPARISON OF ARTIFICIAL NEURAL NETWORKS TO A CONVENTIONAL HEURISTIC TECHNIQUE FOR OPTIMIZATION PROBLEMS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Space Operations

Jeffrey Scott Gruner, A.S., B.S., M.S.
Captain, USAF

December, 1992

...

Acknowledgements

I thank my family for their support during my eighteen months in AFIT: My beautiful wife Kelly, for her love, and for either doing or doing without the things I normally take care of; my daughter Erika for the moments of joy she brought me as a new father; and my parents for their patience and backing through the twenty-one previous years of schooling that brought me to this opportunity. Lastly, I thank the two members of my thesis committee for their support: my reader Major Ken Bauer and especially my advisor, Major Steve Rogers (a.k.a. Captain Amerika), whose perpetual confidence and technical insights proved invaluable in assisting my completion of this thesis.

Jeffrey Scott Gruner

Table of Contents

List of Figures

List of Tables

AFIT/GSO/ENG/92D-01

Abstract

This thesis investigates the utility of the Hopfield and Kohonen artificial neural networks to the traveling salesman optimization problem. A third, non-neural-network technique (the Christofides Algorithm - a competitive, bounded-solution, operations research technique) is also investigated for comparison to the artificial neural network solutions. An eight-city distribution is chosen for comparison of the solutions.

COMPARISON OF ARTIFICIAL NEURAL NETWORKS TO A CONVENTIONAL HEURISTIC TECHNIQUE FOR OPTIMIZATION PROBLEMS

I. Introduction

Many problems in government and business today can be formulated as optimization problems.[1] Optimization has become so important to government and business leaders that millions of research dollars are being spent each year looking for *the better solution*, when the one best is unattainable. This background will briefly review efforts being made today in the field of optimization, focusing on a single problem, the classic, traveling salesman problem (TSP).

Although the TSP is easy enough to describe, it is effectively impossible to solve for the one-best solution for large numbers of cities. For this reason, the TSP is representative of an important class of optimization problems, those for which perfect solutions are unattainable. Because perfect solutions are unattainable, there is a large interest in methods that can find near-optimal solutions in reasonable amounts of time.

Problem Statement

The traveling salesman problem is a classic in the field of combinatorial optimization, concerned with efficient methods for maximizing or minimizing a function

[1]Optimization meaning finding the best possible solution you can, when finding the one best solution is impracticable.

of many independent variables (5:629). In layman's terms, the problem is to find the shortest travel path to a number of cities, say n, visiting each city only once, and returning to the city where the tour started out (henceforth, referred to as the shortest closed tour).[2] Like many similar mathematical problems that turn out to be difficult to solve, the TSP is not difficult to solve because a method for it is not known, but because "brute-force" or non-elegant procedures take too long (11:262). The only known method certain to find the shortest possible path for the TSP is to investigate every distinct closed path, of which there are $n!/2n$ possible paths.[3] For a set of just *ten* cities, this equates to 181,440 possible paths to investigate! Increasing the number of cities, n, increases the number of possible tours exponentially. For a 30-city tour, there are a staggering $4.42x10^{30}$ distinct paths to investigate. So many in fact, that it would take even the fastest computers years to handle (11:262).

Artificial intelligence and artificial neural networks will be defined next, to provide a better understanding of what kind of technology is being applied to the problem.

Definitions of Key Terms

Artificial Intelligence. Artificial intelligence is the unnaturally-occurring, derivative form of the human faculty to think and reason that is programmed into electro-mechanical machinery.

Whereas artificial is defined as an imitation of a natural object or process, (20:124) and intelligence as the faculty of reasoning or understanding, (20:1174) artificial intelligence is the capability of an electro-mechanical machine to imitate the human mental function on a limited scale. It is limited in the sense that it

[2]For the story of how the traveling salesman problem came under academic scrutiny, refer to Appendix A.

[3]The mathematical notation "$n!$" stands for n-factorial, meaning to multiply the number n by n - 1, by n - 2, by . . . , by 1. For example, 4! = 4 x 3 x 2 x 1 = 24.

cannot exist autonomous of its human programming, and exists for the sole purpose of its design application.

The concept of artificial intelligence grew out of the desire to provide electro-mechanical machinery with a primitive decision-making ability, in order to improve their autonomous operating capabilities. Since the late 1970s, machines and weaponry possessing some measure of autonomous decision making ability (artificial intelligence) have earned the labels of "smart machines" and "smart weapons." The academic study of artificial intelligence encompasses computer science, electrical engineering, mathematics, physics, and numerous supporting disciplines.

Artificial Neural Networks. Artificial neural networks are collections of model nerve cells, inspired by those in the human brain, and the weighted interconnections among them. The nerve cells in the brain are called neurons. The model neurons of an artificial neural network represent an approximation to the biological neurons "in which a simplified set of important computational properties is retained" (7:625).

When a neuron fires, or releases, an electrical impulse while processing information, it broadcasts a signal to thousands of other neurons which, in turn, fire to millions more. In a split second, entire sections of the brain become involved and information processing seems to happen everywhere at once.[4] The brain has an estimated ten billion neurons and more than 1,000 times that many interconnections among them (12:92). Artificial neural networks are man's attempts to model this vast web of interconnections and neuron firings in hopes of producing computers and computer programs that are capable of intelligent human reasoning.

What di nguishes neural network computers from regular computers is a radical departure in how the computer's electronic components are organized. The design for virtually all of today's computers was established in the 1940's by a mathematician named John von Neumann. The design physically separates the computer's

[4]Man's attempt at representing of this phenomena is known as parallel processing.

memory and its processor, with a communications link in between.[5] Neural network computers, by comparison, attempt to map their memory directly onto the information processing network, thus eliminating the communications link and giving them the unique ability to *learn*, something that cannot be done by a von Neumann design computer.

Scope of Thesis

This thesis investigates three approaches to solving the TSP for an optimal or near-optimal solution. Two are based on a branch of artificial intelligence called artificial neural networks, the Hopfield and Kohonen networks. The third is a heuristic approach, the Christofides Algorithm. The methods of operation and differences among each of these approaches will be explained in detail in the succeeding chapters.

Thesis Organization

After this introduction to the thesis problem, Chapter 2 is a review of current approaches to solving the TSP. The material has been carefully sculpted to include the two, best-known artificial neural network approaches to solving the TSP, a closely-related analog approach, and two heuristic techniques. Following presentation of this background material, Chapter 3 is devoted soley to the research methodology of this thesis, Chapter 4 to interpreting the results of the research, and lastly, Chapter 5 to summarizing the results of this thesis.

The first approach reviewed in the background material is the artificial neural network designed by biology and chemistry professor John J. Hopfield of California Institute of Technology, and David W. Tank of the Molecular Biophysics Research Department of AT&T Bell Laboratories (12:93–94).

[5]The communications link is a bottle neck that slows the machine down, forcing the processor to wait while information needed for processing is plucked out of memory or sent back to it. As a result, computers based on the von Neumann design will never attain the hyperfast speeds needed to attack the tough problems facing science and engineering today.

II. Background Material

The Hopfield Network

The central idea of most optimization algorithms is to move about in a space of possible solutions, progressing in the direction that tends to decrease the cost function, hoping all the while that the "space and method of moving are smooth enough that a good solution will ultimately be found" (7:630).

The Hopfield network, first published in 1986, maps the optimization problem onto a neural network consisting of "nonlinear, graded-response model neurons organized into networks with effectively symmetric synaptic connections" (7:630). The mapping is done in such a way that the network configurations correspond to the possible solutions of the problem. Next, a derivative of a Lyapunov function[1] referred to as the network's computational energy, or just energy function, E, is chosen proportional to the possible solution configurations of the cost function of the problem.[2] As the network computes, the energy function is minimized and a path is constructed through the space, tending in the direction of minimum energy and, therefore, the minimum cost function. When a steady-state configuration is reached, it will correspond to a local minimum of the energy function and, one hopes, the optimum solution within the solution space.

For a better understanding of how the Hopfield network operates, picture a three-dimensional box with one of its corners perfectly fit into the corner formed by the intersection of the X - Y - Z coordinate axes. If the X and Y axes define

[1] A function that always decreases each time the network changes state, eventually reaching a minimum and stopping, thereby ensuring the network is stable/steady-state.

[2] "The term 'energy function' stems from an analogy between the network's behavior and that of certain physical systems. Just as physical systems may evolve toward an equilibrium state, a network of neurons will always evolve toward a minimum of the energy function. The stable states of a network of neurons therefore correspond to the local minima of the energy function. Hopfield and Tank had a key insight when they recognized that it was possible to use the energy function to perform computations" (16:1349).

the horizontal plane, visualize the box being open-ended in the positive Z (upward) direction. Imagine draping a sheet over the top of the open box containing a collection of randomly sized and shaped objects, and the sheet settling down around the objects until its shape no longer changes. The resulting surface of the sheet can be thought of as a representation of an energy function, with the four sides of the box and the X - Y plane defining its boundaries. The network described by Hopfield and Tank searches the energy function for the lowest possible point anywhere on the surface. The lowest point on the surface corresponds to the optimal solution within the solution space.

The 30-city TSP can be solved by a Hopfield network having just 900 model neurons. A neural network of this size will converge to an answer in about $1x10^{-6}$sec. With conventional algorithms, a comparably good solution to the TSP can be found "in about 0.1 second on a typical microcomputer having 10^4 times as many devices" (7:630).

In spite of the promise it holds, the Hopfield network is not without flaws. A major problem that has been the subject of much research is that, "even though such networks are capable of producing valid solutions to the TSP, it has been found on most occasions that the final state of the network is invalid" (8:942). Despite its flaws, further research has been carried out by others interested in the promise the network holds. These other efforts have addressed improving the validity of the network output, improving the frequency of shortest valid tours, and improving the speed of the network's convergence (8:942).

Wilson and Pawley later pointed out that it was difficult for the Hopfield network to converge to optimum solutions because of the stable energy states of the network corresponding to the local minima of the energy function (21). In order to escape the local minima, a stochastic process can be introduced into the network (1) (15), and an annealing technique can be introduced to improve the network's frequency of convergence to optimum solutions (22:407).

Kashmiri pointed out that with careful selection of four of the five parameters in the network energy equation, "the solutions obtained for random, normalized inter-city distances were always valid. In addition, 90% of the results were optimal solutions. The rest of the results were usually the second best path for the tour" (8:943).

A class of networks known as Boltzmann Machines have largely solved the tendency of Hopfield networks to stabilize to local rather than global minima. In Boltzmann Machines, neurons change state in a statistical rather than deterministic fashion. This method is often called simulated annealing, because there is a close analogy between this method and the way in which a metal is annealed, or tempered. A metal is annealed by heating it past its melting point and letting it gradually cool. At high temperatures, the atoms of the metal have very high energies. As the temperature of the metal cools, the atomic energies decrease and the system, or metal, settles into a minimum energy configuration. When cooling is complete, the system energy is at a global minimum (19:100-101).

At a given temperature, the probability distibution of system energies is determined by the Boltzmann probability factor, Equation 1.

$$P = exp(-E/kT) \tag{1}$$

where:

E = the system energy

k = Boltzmann's constant

T = temperature

The statistical distribution of energies allows the system to escape a local energy minimum, while the probability of high system energy decreases rapidly as temperature drops, thus creating a strong bias toward low-energy states at low temperature. If the Hopfield network neuron state-change rules are determined statistically rather than deterministically, the result is a simulated annealing system (19:100-102).

The Boltzmann-Machine technique can be applied to networks of virtually any configuration, although stability cannot be guaranteed, and the technique is known as computationally intense.

So from the results of Wilson, Pawley, Kashmiri, and Boltzmann, it appears that Hopfield networks can be successfully applied to the TSP for large numbers of cities, but must first be altered from their initial design.

The Elastic Net Method

The elastic net method was published in 1986, subsequent to Hopfield and Tank's article, by molecular biologist Richard Durbin of the King's College Research Centre in Cambridge, England and zoologist David Willshaw of the University of Edinburgh in Scotland.

Durbin and Willshaw describe how "a parallel analogue algorithm, derived from a formal model for the establishment of topographically ordered projections in the brain, can be applied to the travelling salesman problem. Using an iterative procedure, a circular closed path is gradually elongated non-uniformly until it eventually passes sufficiently near to all the cities to define a tour" (5:689).

For the layman, picture 30 nails driven at random, part way into a flat wooden board. Next, visualize centering a large circular rubber band at the center of the distribution of randomly driven nails. Lastly, visualize gradually stretching the rubber band non-uniformly outward to pass around each and every nail. With the nails representing the cities to be visited, the rubber band defines the shortest travel path to

each city, returning to where it started from, the shortest closed tour. This analogy demonstrates the essence of how the elastic net method operates.

The algorithm at the heart of this method "is a procedure for the successive recalculation of the positions of a number of points in the plane in which the cities lie. ... The elastic net method operates by integrating a set of simultaneous first-order difference equations" (5:690).

The elastic net method was applied to the TSP using a standard set of 30 cities and generated the shortest known tour in only 1,000 iterations. In a larger test using a standard set of 100 cities, the elastic net found a solution within one percent of the best tour known by any other method given the same distribution of cities.

Durbin and Willshaw pointed out the main computational cost of their method is having to recalculate the distances at each iteration, and that the method is only suitable to *geometrical* optimization problems, and cannot be extended to the case of the TSP where it is fed an arbitrary matrix of distances among cities.

The Kohonen Network

Teuvo Kohonen of the Helsinki University of Technology published an article in 1984 regarding something he called a self-organizing feature map.[3] Previously, a vast amount of effort had been spent studying supervised learning algorithms.[4] Less attention had been devoted to the class of algorithms which do not require explicit tutoring and which spontaneously self-organize when given a set of input patterns.[5] Kohonen's self-organizing feature maps "show how input signals of arbitrary dimensionality can be mapped, or adaptively projected, onto a structured set of processing

[3]Really just an algorithm which orders responses to inputs spatially.

[4]Learning can be either supervised or insupervised. Supervised learning requires an external controller to evaluate the performance of the system or network, and direct subsequent modifications as necessary. Unsupervised learning does not require a controller: the system or network self organizes to produce the desired changes and outputs (19:38).

[5]This area of research directly relates to the problem of understanding the internal representation of information in the brain (2:289).

units, in such a way that topological relations of the input patterns and of the representation patterns are kept similar" (2:289). Kohonen demonstrated applications of his self-organizing feature maps to various cognitive tasks, but it was not until 1988 when Angeniol, Texier, and Vaubois (henceforth, simply referred to as Angeniol) showed the potential of this approach to solving optimization problems such as the traveling salesman problem.

The approach offered by Angeniol is truly unique from the traditional algorithms, where the intermediate tours being examined represent all of the path permutations through the set of cities. For the Angeniol approach, picture a set of nodes joined together in a ring in a plane. The ring of nodes are allowed to move freely in the plane during an iterative process in which every city effectively captures a node of the ring until a complete tour is obtained. Each iteration consists of presenting just one city to the ring of nodes. The node closest to the city moves towards it, simultaneously inducing its neighbors to do likewise, but with a decreasing intensity further away on the ring. This induced movement on neighboring nodes is what minimizes the distance between neighbors, ultimately producing a short tour. As the nodes are captured by cities, they become more and more independent of each other, and eventually, a node is attached to every city and a shortest path tour is found.

When Angeniol's approach was applied to small sets of cities taken from Hopfield and Tank, and from Durbin and Willshaw, the network delivered satisfactory results in all cases. Run against the 30-city TSP, the network delivered, "a good average solution (less than 3% greater than optimum)" (2:292) in two seconds on standard computer hardware. In a comparison versus the elastic net method, the self organization method, on average, is equivalent, but its capability to start with a random order of the cities gives it a chance of getting better results.

The self-organizing feature map offers promise for three reasons. The first is that the total number of model neurons and inter-connections is proportional,

"only to the number of cities in the problem, thus scaling very well with problem size" (2:292). Second, only a single gain parameter controlling the total number of iterations has to be tuned.[6] And third, use of standard values of the gain parameter ensure good, near-optimum solutions in reasonable amounts of time.

Further theoretical work has been conducted that supports Angeniol's results. One of these works is that of Yoshihara and Wada in the Department of Research at the Olympus Optical Company, Tokyo, Japan. In their paper given at the 1991 Institute of Electrical and Electronic Engineers (IEEE) Conference, they describe a derivative of the Kohonen network, called an extended learning vector quantization (ELVQ), in which, "the best matching neuron of the self-organizing feature map is calculated with an energy function" (22:407). When an input vector is given to every model neuron in a two-dimensional self-organizing array and a best-matching neuron found, "then, not only is its own synaptic weight vector but the synaptic weight vectors of its topological neighbors are modified to increase the strength of the match" (22:407).

The results of the ELVQ are impressive. Defining the unit of a time step as the period of time required for self-organizing once for each of the n cities, the ELVQ found the optimum solution to the ten-city TSP within 50 steps with 100 percent probability. Versus the 30-city TSP, the optimum solution was obtained within 100 steps with a probability of 52 percent, while the other 48 percent were distributed within 1.2% of the optimum length (22:411).[7]

To summarize the ELVQ, both its rate of convergence to the optimum solution and number of steps to convergence are better than the other neural networks presented here. Since the self-organizing process can be controlled by the energy

[6]For a gain value of 0.01, the self-organizing feature map took twelve hours to solve a 1,000-city TSP. For a gain value of of just 0.2 against the same city set, the self-organizing feature map took just 20 minutes.

[7]Against the standard 30-city TSP, the optimum solution is obtained in just 13.0 seconds on a SPACRstation1. The Sun Work Stations at AFIT are SPACRstation2s.

function, the ELVQ can be applied not only to the general class of optimization problems, but to *pattern classification* problems as well (22:413).

Heuristic Methods

To measure the performance of the two artificial neural networks against a known and accepted standard, a heuristic algorithm was sought. Initial efforts focused on Lin and Kernighan's heuristic algorithm until it was learned that the algorithm delivers "unbounded" solutions, and is not good for starting out and finding a shortest tour.[8] Its strength lies in improving on a pre-existing path found by some other means. The search to find a "proven-bound" heuristic solution led to the Minimal Spanning Tree (MST) Method and Christofides Algorithm (3).

First, there is a big difference between a minimal spanning *tree* and the minimal spanning tree *method solution.* The minimal spanning *tree* is not a closed tour, but a connection of every node with at least one other by the shortest possible distance, hence it looks like a tree and not a loop or a closed tour. The minimal spanning tree *method* builds on the minimal spanning tree to find a closed tour. The solution it finds is guaranteed to be less than or equal to twice the optimal solution for the TSP, (MST Method Solution $\leq 2 \times$ TSP*, where * denotes the optimal solution).

Another improvement on the MST, believed by some to be *the best heuristic algorithm*, is known as the Christofides Algorithm. Like the MST Method, it builds on the minimal spanning *tree* to find a closed tour. The solution it finds is a little better than that of the MST Method, guaranteed to be less than or equal to one-and-one-half times the optimal solution for the TSP, (Christofides Solution $\leq 1.5 \times$ TSP*).[9]

[8] "Unbounded" meaning there is no guarantee on the quality of the solution. A "bounded" solution, for example, may be guaranteed to be less than or equal to twice the optimal solution.

[9] It is possible that applying the two algorithms in series, the Christofides followed by the Lin and Kernighan, would deliver a better quality solution than either one individually. Let the Christofides Method find a real-good closed tour, then let the Lin and Kernighan algorithm improve on it. If the artificial neural networks can beat those solutions, then they certainly out perform the very best

Minimal Spanning Tree Method The step-by-step Minimal Spanning Tree Method follows, illustrated pictorially in an example.

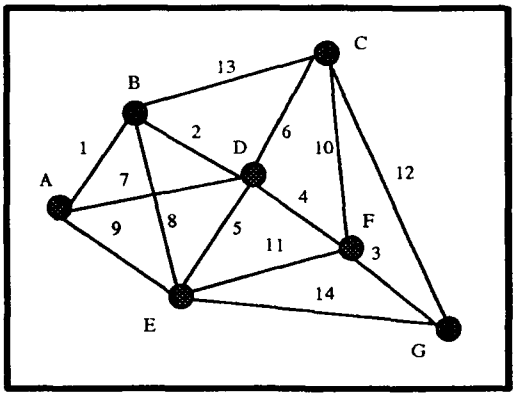

Figure 1. Distribution of Cities and Distances

Step 1. Find the minimal spanning tree.

a) Go through the list of cities $i = 1$ to n, determining the distances along the straight-line arcs between each city, and sort all of the arcs by distance from low to high, Figure 1.[10] The length of the arcs is independent of direction (that is, the distance ij = the distance ji, also known as the *symmetric* TSP), hence they are not shown as arrows in any of the diagrams until the short-cut routine.

b) Go through the ordered list from low to high, choosing arcs to connect all of the cities if and only if the additions do not create a closed loop or cycle, stopping when there are $n - 1$ arcs.

In Figure 1, the shortest arc length (1) emanates from City A and it connects to City B. The next shortest arc length in the distribution (2) connects City B with City C, and so forth until there are $n - 1$ arcs, each of the cities is

heuristic methods and hold great promise. However, investigating the quality of heuristic algorithm solutions was not the aim of this thesis.

[10]Not all of the arcs are shown to prevent cluttering the diagram.

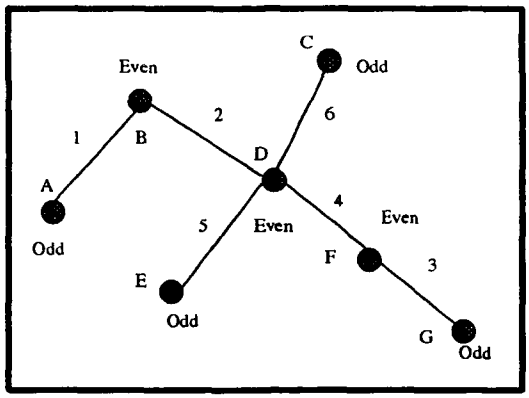

Figure 2. The Minimal Spanning Tree and Labeled Cities

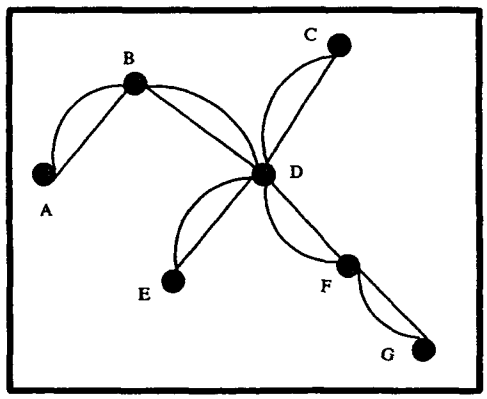

Figure 3. Doubling of All Arcs

connected to every other in a single tree structure, and there are no closed loops. This, by definition, completes the minimal spanning tree.

Step 2. Label each of the cities as either odd or even according to the number of arcs emanating from them. The method guarantees an even number of odd-labeled cities.

In Figure 2, cities B, D, F, and G are labeled "Even" while A, C, E, and G are labeled "Odd".

Step 3. Double all of the arcs on the tree *to make all of the cities even*, Figure 3.

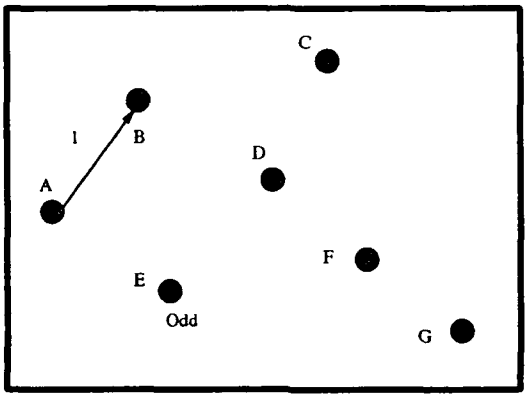

Figure 4. MST Short-Cut Routine, Step 1

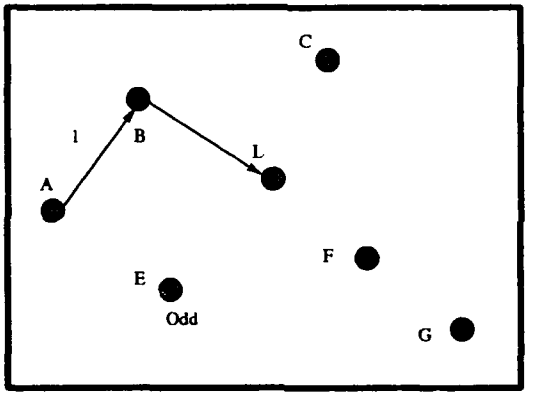

Figure 5. MST Short-Cut Routine, Step 2

Step 4. Conduct a short-cut routine to find the closed-cycle solution.

a) Starting at the city with the shortest arc length emanating from it i (henceforth referred to as the origin), take that shortest arc to the next city, $i + 1$.

City A is chosen as the origin since the shortest arc length (1) emanates from it and connects to City B, Figure 4.

b) Take the shortest arc emanating from the next city, $i + 1$, that connects to an *unvisited* city.[11] If there are no unvisited cities immediately adjacent to the

[11] An *unvisited* city is one that has not been added to the tour during the short-cut routine.

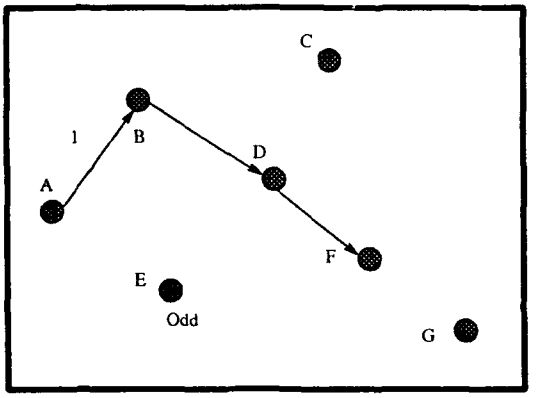

Figure 6. MST Short-Cut Routine, Step 3

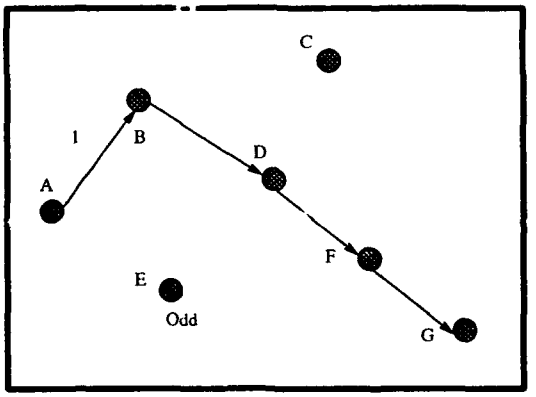

Figure 7. MST Short-Cut Routine, Step 4

current city, back-up one city in the path and look for the nearest unvisited city immediately adjacent to it.[12] If there are none, backup along the path one city at a time looking for an unvisited city immediately adjacent until one can be found. When one is found, go back to the current city and connect it with the unvisited city, whatever its distance.[13]

[12] ***Immediately adjacent*** means reachable by an existing arc on the minimal spanning tree from the next city, $i + 1$.

[13] If more than one unvisited city is found adjacent a previously visited city, the choice of which one to connect the current city with is totally arbitrary.

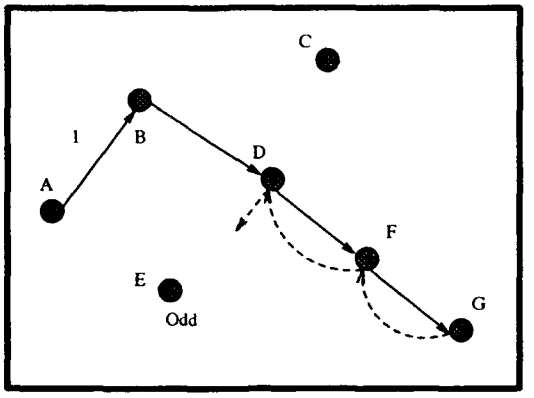

Figure 8. MST Short-Cut Routine, Step 5

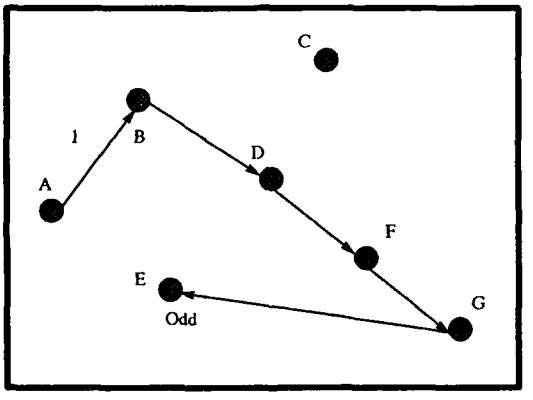

Figure 9. MST Short-Cut Routine, Step 6

In sequence: City B connects to City D by length 2, Figure 5; City D connects to City F by length 4, Figure 6; City F connects to City G by length 3, Figure 7, and then there are no unvisited cities adjacent City G. At this point, the technique back tracks along the tour one city and checks city (F) for an unvisited city adjacent to it. There are none, and again, the technique back tracks along the tour one city and checks city (D) for an unvisited city adjacent to it. There are two, C and E, and E is chosen to be connected with City G since it is the nearer of the two cities to City D, Figures 8 and 9. Once E becomes the next city, $i + 1$, it lacks an unvisited city adjacent to it, and

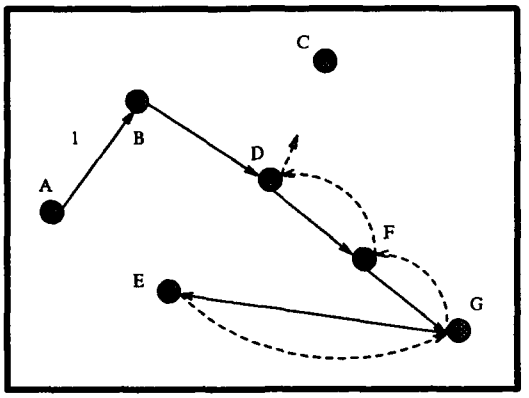

Figure 10. MST Short-Cut Routine, Step 7

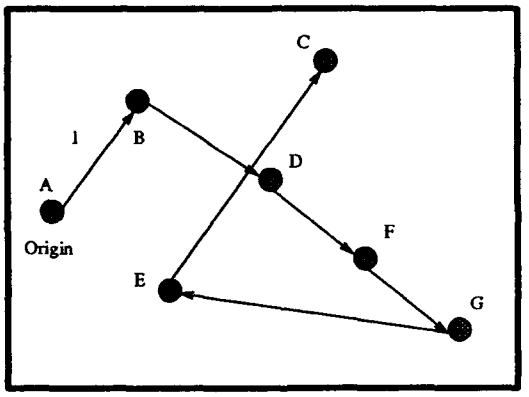

Figure 11. MST Short-Cut Routine, Step 8

the technique back tracks along the tour, stopping at City D with its unvisited neighbor C, Figure 10. City E is connected to City C, Figure 11.

c) Check to see if all of the cities have been visited. If yes, take the shortest existing arc, or create one if none exists, back to the origin city, whatever its distance, and exit the short-cut routine and minimal spanning tree method. If all of the cities have not been visited, return to Step b).

Once City E is connected to City C, all of the cities have been visited. The only thing left is to close the tour back at the origin. Since an arc does not exist

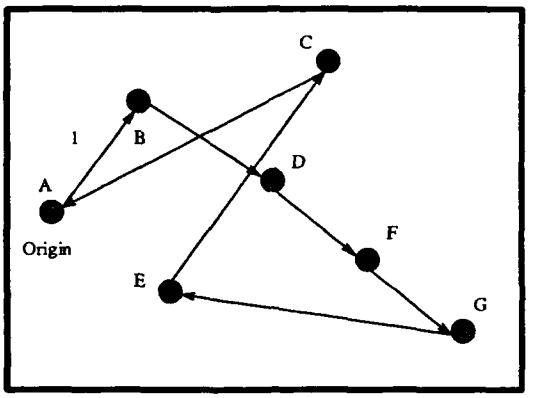

Figure 12. MST Short-Cut Routine, Step 9

between C and A, one is created, and the technique is finished, the solution found, Figure 12.

Christofides Algorithm The step-by-step Christofides Algorithm follows, also illustrated pictorially in an example.

Step 1. Find the minimal spanning tree.

Same as Step 1 of the MST Method, Figure 1.

Step 2. Label each of the cities as either odd or even according to the number of arcs emanating from them.

Same as Step 2 of the MST Method, Figure 2.

Step 3. Add arcs between pairs of nearest, unpaired, odd-labeled cities to make all of them even.

Cities A and E are connected to each other, and Cities C and G are connected to each other, Figure 13.

Step 4. Conduct a short-cut routine to find the closed-cycle solution.

As in the MST Method, City A is chosen as the origin and connects to City B, Figure 14, which connects to City D, Figure 15, which connects to City F,

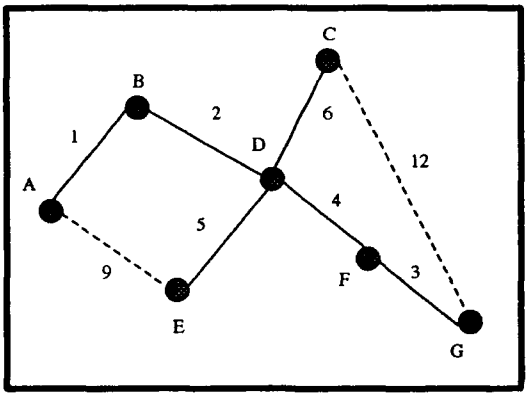

Figure 13. Connection of Nearest Odd-Labeled Cities

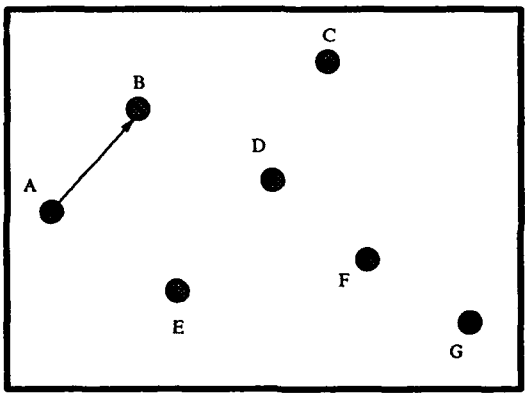

Figure 14. Christofides Short-Cut Routine, Step 1

Figure 16, which connects to City G, Figure 17, which connects to City C, Figure 18. Since City C does not have an unvisited neighbor, a back track is conducted turning up unvisited City E adjacent City D, Figure 19. City C is connected to City E, Figure 20, which connects with the origin, City A, and the technique is complete, a solution found, Figure 21.

Heuristics Summary The only difference between the Minimal Spanning Tree Method and Christofides Algorithm is seen in Step 3 of both of them. While it is only a subtle difference (doubling each of the arcs to make all of the cities even versus

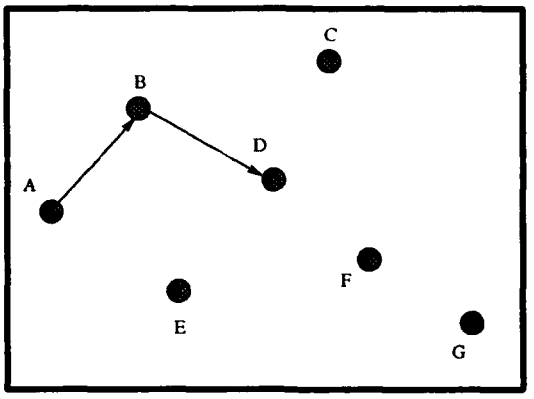

Figure 15. Christofides Short-Cut Routine, Step 2

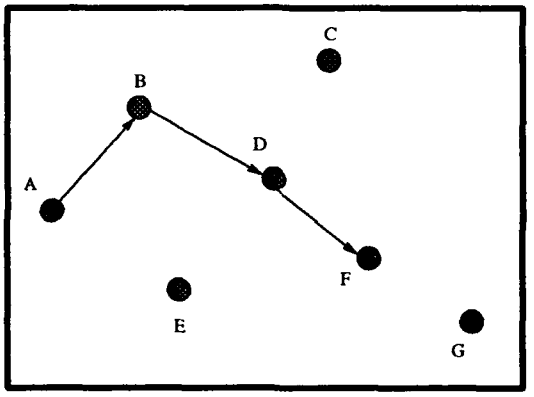

Figure 16. Christofides Short-Cut Routine, Step 3

adding arcs between pairs of nearest, unpaired, odd-labeled cities to make all of the cities even), the solutions each converge to are noticeably different from each other.

Discussion of Background Material

This literature research has reviewed five current approaches to solving the traveling salesman problem. Two of the approaches use artificial neural networks, the Hopfield and Kohonen networks, the third was an analogue approach using an elastic net method, and the fourth and fifth methods were heuristics.

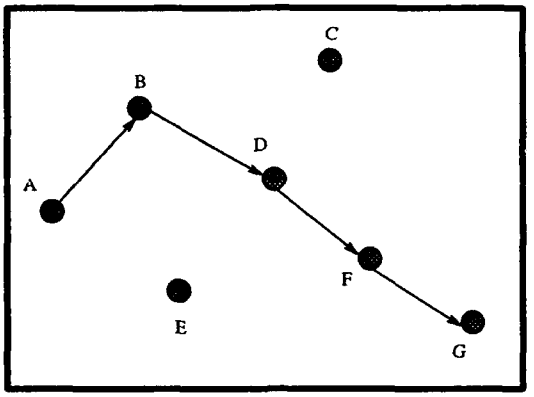

Figure 17. Christofides Short-Cut Routine, Step 4

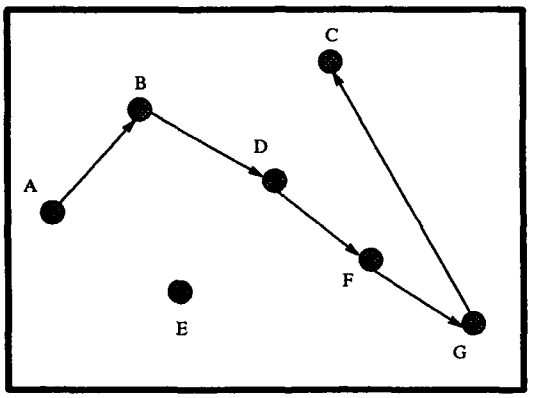

Figure 18. Christofides Short-Cut Routine, Step 5

The Hopfield network was shown to hold promise for solving the TSP in spite of its flaws. In fact, supporting research has turned up several means of improving its performance.

- Stochastic processes can be introduced into the network that will enable the algorithm to escape local minimas of the network energy function, and provide better convergence to optimal solutions (1) (15).
- Annealing techniques can be introduced into the network to improve its frequency of convergence to optimum solutions (22:407).

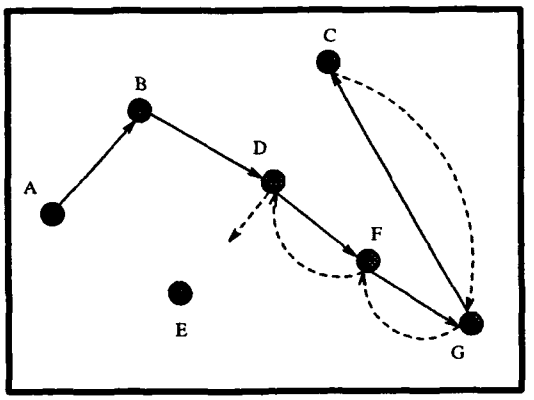

Figure 19. Christofides Short-Cut Routine, Step 6

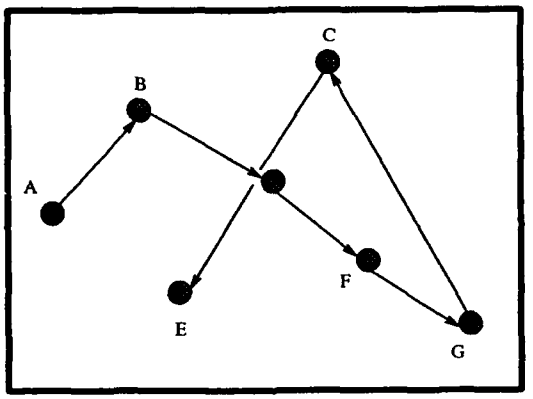

Figure 20. Christofides Short-Cut Routine, Step 7

- Careful selection of four of the five parameters in the network energy equation will not only ensure valid solutions, but will also deliver optimum solutions a higher percentage of the time, and near-optimum solutions for the rest.

Durbin and Willshaw's elastic net method uses an iterative procedure to gradually elongate a closed circular path non-uniformly until it eventually passes sufficiently near to all of the cities to define a tour. Although the elastic net method is capable of regularly generating optimum or near-optimum solutions to the TSP, it has two drawbacks which invalidate it for this thesis. First, this thesis is concerned

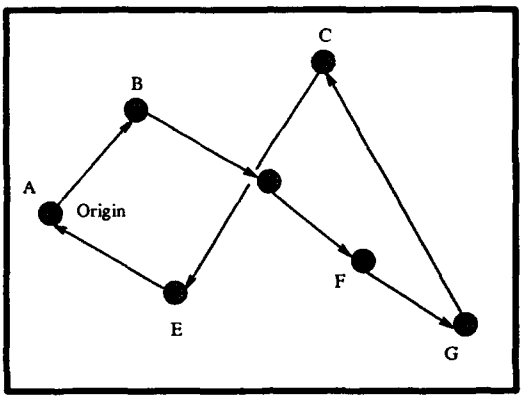

Figure 21. Christofides Short-Cut Routine, Step 8

with applying an artificial neural network to the TSP, and not an analogue approach like the elastic net method. And second, the elastic net is limited to geometrical applications of the problem, and cannot be fed a matrix of numbers. To use the elastic net method would introduce an unacceptable degree of abstraction into the problem.

The third approach covered by this review was the Kohonen network. Originally published for application to cognitive tasks, Kohonen networks were not immediately applied to optimization. Kohonen networks use an algorithm that does not require explicit tutoring and which spontaneously self-organize when given a set of input vectors. Angeniol, Texier and Vaubois were the first to apply Kohonen networks to optimization and the TSP (2). When their approach was first published, it was truly unique from traditional algorithms being applied to optimization. Their experimental results offered promise while at the same time, leaving room for improvement. Angeniol, Texier and Vaubois did identify a single parameter controlling the total number of iterations that has to be tuned when applying the Kohonen network to the TSP.

Research conducted by Yoshihara and Wada at the Olympus Optical Company supported Angeniol's results, and refined the network performance even further. Their derivative of the Kohonen network was called the extended learning

vector quantization, and delivered the best results of the neural network approaches reviewed here (22).

The Minimal Spanning Tree Method and Christofides Algorithm both appear to be simple to implement, and deliver "proven bound" solutions, something that will be important when quantifying the results of the neural network techniques. Of the two, the Christofides Algorithm finds a slightly better solution, and therefore may be more desirable to implement for the purpose of this thesis.

This chapter has reviewed: the artificial neural network of Hopfield and Tank; the Elastic Net Method of Durbin and Willshaw, a method suitable to geometric optimization problems; the Kohonen artificial neural network and some of its variants; and two heuristic approaches for solving the TSP, the Minimal Spanning Tree Method and Christofides Algorithm. The next chapter will present the research methodology used in comparing the two artificial neural networks to Christofides' heuristic technique for solving the traveling salesman optimization problem.

III. Methodology

The Hopfield Network

The Solution Form and its Constraints To understand the methodology for applying Hopfield and Tank's artificial neural network to the TSP, it is first necessary to understand the solution form the network converges to, and a little bit about the constraints on that form. The network requires the use of n^2 neurons to solve an n-city TSP, the final output of the network an $n \times n$ matrix, where each row of the matrix corresponds to a city, and each column corresponds to the position of a city on the tour. The output matrix consists of n ones and $n^2 - n$ zeros. For example, in a four-city tour of cities labeled A, B, C, and D, there will be four ones and twelve zeros in the output matrix. Say the optimal solution, or shortest travel tour, is ordered C-A-D-B-C.[1] The matrix below and in Figure 22 would be the output of the network. The one in the second column of the first row means that city A would be visited second on the tour. The one in the fourth column of the second row indicates that city B would be the fourth visited, et cetera.

$$OUT = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

In one-dimensional form, the output would be the vector OUT below.

$$OUT = [(0100)(0001)(1000)(0010)].$$

[1]Recall, the solution must return to the city from which it began.

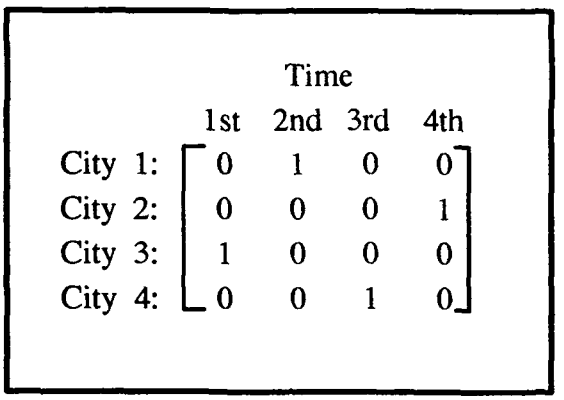

Figure 22. Hopfield Solution Form

For the n^2 neurons to compute a solution, the problem must first be mapped onto the network in such a way that the network's configurations correspond to the solutions of the problem. In order to do this, the cost function[2] must be constructed in the form of a positive, definite Lyapunov function, commonly known in engineering circles as an energy function. The energy function, denoted by E, is defined below.

$$E = \frac{-1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} OUT_i OUT_j \tag{2}$$

where:

n = the number of cities

W_{ij} = the inter-connection weight between cities i and j

OUT_i = the network outputs of the neurons representing city i

OUT_j = the network outputs of the neurons representing city j

[2]The one you are trying to optimize.

For Hopfield's application, it must encompass the problem constraints and its lowest energy state (value) must correspond to the optimal solution/shortest travel path. This levies two requirements on the energy function.

- First, the energy function must favor stable states in either the one or two-dimensional output forms shown above.
- Second, of all the valid possible solutions, the energy function must favor those representing the shortest travel paths.

These two requirements on the energy function translate into three conditions on the output matrix.

- First, there can only be a single one in each column of the matrix.
- Second, there can only be a single one in each row of the matrix, thus only n ones in the entire matrix and $n^2 - n$ zeros.
- Third, the output matrix must not only be a valid solution, but must also be the optimal solution, or shortest possible tour length.

The Original Hopfield Hopfield and Tank's energy equation satisfying the three conditions is:

$$\begin{aligned} E &= \frac{A}{2}[\sum_{OUT}\sum_{i}\sum_{j\neq i} OUT_{Xi}OUT_{Xj}] \\ &+\frac{B}{2}[\sum_{i}\sum_{X}\sum_{Y\neq X} OUT_{Xi}OUT_{Yi}] \\ &+\frac{C}{2}[(\sum_{X}\sum_{i} OUT_{Xi}) - n]^2 \\ &+\frac{D}{2}[\sum_{X}\sum_{Y\neq X}\sum_{i} d_{XY}OUT_{Xi}(OUT_{Y,i+1} + OUT_{Y,i-1})] \end{aligned} \tag{3}$$

where:

$$
\begin{aligned}
X &= \text{city subscript} \\
Y &= \text{city subscript} \\
i &= \text{neuron subscript} \\
j &= \text{neuron subscript} \\
d_{XY} &= \text{distance between cities X and Y} \\
i-1 &= \text{neuron subscript in modulo } n \\
i+1 &= \text{neuron subscript in modulo } n
\end{aligned}
$$

The first triple summation is zero if the first condition is satisfied (tour visits each city only once), the second triple summation is zero if the second condition is satisfied (tour visits only one city at a time), and the third is zero *if and only if* there are exactly n ones in the entire matrix (tour visits all of the cities in the problem). The fourth triple summation measures the total tour length for valid solutions and satisfies the third condition.

Each one of the n^2 neurons connects to itself and to every other neuron in the network, thus there are $n^2 \times n^2$, or n^4, neuron inter-connection weights. The inter-connection weights are generally held in an $n^2 \times n^2$ matrix. The formula for determining the weights is:

$$
\begin{aligned}
W_{Xi,Yj} &= -A\delta_{XY}(1-\delta_{ij}) - B\delta_{ij}(1-\delta_{XY}) \\
&\quad -C - Dd_{XY}(\delta_{j,i+1} + \delta_{j,i-1}),
\end{aligned} \tag{4}
$$

where:

$$
\begin{aligned}
\delta_{ij} &= 1 \quad \text{if} \quad i = j \\
&= 0 \quad \text{otherwise} \\
\delta_{XY} &= 1 \quad \text{if} \quad X = Y \\
&= 0 \quad \text{otherwise}
\end{aligned}
$$

The terms with the constants A, B, and C as coefficients provide the general constraints required for any TSP.[3] The term with the D constant as a coefficient is the data term describing the locations of cities and distances between them.

Each neuron also has a bias weight, I_i, with a value of $C \times n$ that is connected to an input value of one.

Each neuron sums all of the products of the inputs and interconnection weights routed to them. The summations are called the neuron states, denoted u_i, where the subscript i is the individual neuron identifier. The states are all passed to a characterization function which transforms their summation values into an output value and assembles all of them in an $n^2 \times 1$ matrix (or column vector) of ones and zeros depending on whether the summation values exceeded the characterization threshold. The only restriction on the choice of the output characterization function is that it be differentiable everywhere. Typically a hyperbolic tangent, piecewise-linear function, or sigmoid is used.

After each neuron's state has been characterized and assembled with all of the other neuron states into an $n^2 \times 1$ output vector, the vector must be checked to determine if it has changed since the last iteration through the network. This check is really to see whether or not the energy function has settled into an energy well, or minima. If the output vector has not changed, this indicates the network has indeed settled into a minima (that is, converged to a solution) which is then output,

[3] A and B were both chosen as eight (8). As explained later, the remainder of the coefficients are determined from formulas depending on A and n.

terminating the computer program.[4] If the output vector has changed, it gets routed back to the input layer for another iteration through the network. For a look at the network configuration, refer to Figure 23 below.

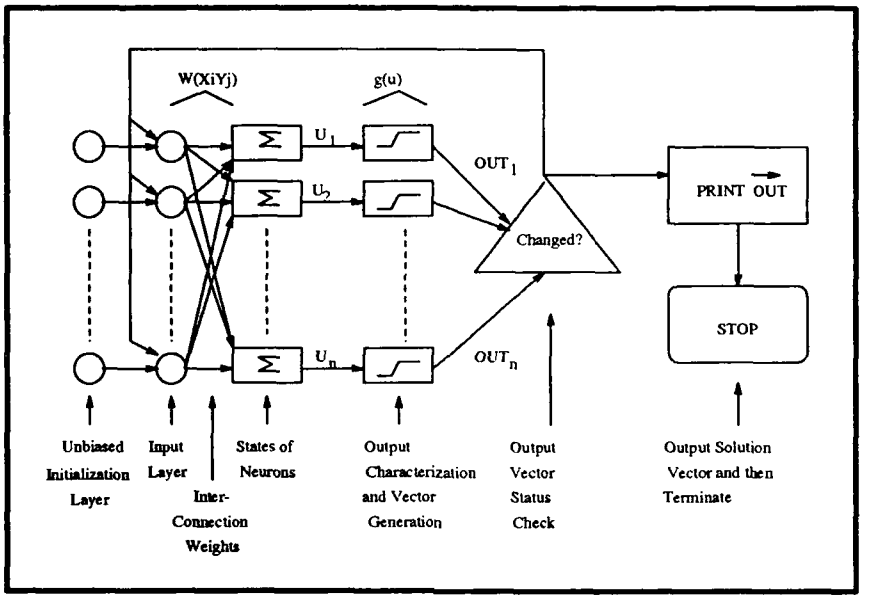

Figure 23. Hopfield Network

To summarize the mathematical formulation of the Hopfield network for this problem, there are three key equations:

- The energy equation, E, for mapping the problem constraints and solutions onto the network.
- The formula for computing the neuron-interconnection weights, based upon the energy function.
- An output function to characterize the states of the neurons into outputs of either ones or zeros.

[4]For a look at how the Hopfield network mathematically searches along the energy function for a minima, refer to Appendix C.

Kashmiri Variant The literature research for this thesis turned up a paper containing two equations describing *the dynamics of the states of the neurons* in the Hopfield network, reference equations 5 and 6 for continuous and discrete applications respectively (8:940–943). These equations were thought to mean that if a *state* changes between iterations, it must be incrementally updated before being passed through the piece-wise linear function. Since this variation seems to conflict with all of the other published information on the Hopfield network, it was investigated further.

$$\frac{du_i}{dt} = -\frac{u_i}{\tau} + \Sigma_{j=1}^{n} W_{ij} OUT_j + I_i \tag{5}$$

$$u(i+1) = u(i) + \delta((W \times OUT) + I_i) \tag{6}$$

where:

δ = the time step of integration

The Hopfield Network Algorithm The first step of the Hopfield network algorithm was to assemble all of the inter-city distances in the form of a matrix.

The second step was to choose values for the constants A and B in the energy equation. To ensure the first two conditions on the output matrix were satisfied, the values chosen for A and B were small positive numbers, and always equal so that each condition received equal weighting.

The third step was to compute values for the constants C, D, and F. From Kashmiri, $C = \frac{A}{n}, D = \frac{An}{80}$, and a new constant $F = 0.97A$ (E was already being used to denote the energy equation).

The fourth step was to determine the n^2 inter-connection weights and the single bias weight. Borrowing from Kashmiri again, two new terms using the constant F were added to the inter-connection weight formula. The new formula for determining inter-connection weights became what is shown in Equation 7.[5]

$$\begin{aligned} W_{Xi,Yj} &= -A\delta_{XY}(1-\delta_{ij}) - B\delta_{ij}(1-\delta_{XY}) \\ &\quad -2F\delta_{XY}\delta_{ij} - C + 2\frac{(An - A + F)}{n^2} \\ &\quad -Dd_{XY}(\delta_{j,i+1} + \delta_{j,i-1}), \end{aligned} \tag{7}$$

where again:

$$\begin{aligned} X &= \text{city subscript} \\ Y &= \text{city subscript} \\ i &= \text{neuron subscript} \\ j &= \text{neuron subscript} \\ i-1 &= \text{neuron subscript in modulo } n \\ i+1 &= \text{neuron subscript in modulo } n \\ d_{XY} &= \text{the inter-city distance between } X \text{ and } Y \\ \delta_{ij} &= 1 \text{ if } i = j,\ 0 \text{ otherwise} \\ \delta_{XY} &= 1 \text{ if } X = Y,\ 0 \text{ otherwise} \\ I_i &= (C \times n) \end{aligned}$$

[5]In this notation, $W_{21,34}$ represents the inter-connection weight between the first neuron of city number two and the fourth neuron of city number three, and so forth.

The fifth step was choosing an output characterization function. Still following the lead of Kashmiri, a piece-wise linear function, $OUT = g(u_i)$, was used:

$$\begin{aligned} g(u_i) &= 0 && \text{if} \quad u_i \leq -0.5 \\ &= u_i + 0.5 && \text{if} \quad -0.5 < u_i < +0.5 \\ &= 1 && \text{if} \quad u_i \geq +0.5 \end{aligned}$$

The sixth step was to initialize the input and output vectors for the first iteration.

The seventh and final step was to start the network running, sit back, and wait a few milliseconds for it to converge to an answer.

The Kohonen Network

The Kohonen algorithm used is a variation on the one developed by the team of French researchers, Bernard Angeniol, Gael De La Croix Vaubois, and Jean-Yves Le Texier (2). Although their work was interesting, the specifics of their algorithm were not clear from their paper, so it was rewritten for this thesis. First are some starting conditions:

- The cities presented to the network are numbered from $i = 1$ to M. Each city's location is denoted by a two-dimensional vector, $\vec{X}_i = (x_{i1}, x_{i2})$, where x_{i1} is the x-axis coordinate and x_{i2} is the y-axis coordinate for the ith city.
- The nodes the network creates are numbered from $j = 1$ to N. Each node location is also denoted by a two-dimensional vector, $\vec{C}_j = (c_{j1}, c_{j2})$, where c_{j1} is its x-axis coordinate and c_{j2} is its y-axis coordinate for the jth node.
- Each node C_j is related to its two nearest neighbors in the ring of nodes, C_{j-1} and C_{j+1}.[6]

[6]There are instances where nodes are deleted and a surviving node's neighbors simply become the next highest or lowest surviving nodes in the ring.

- At the start of the algorithm, only one node exists, located at (0,0). Additional nodes are added according to a creation process explained two subsections ahead.
- In each epoch, every city i is found a closest matching node, or winner, by means of a Euclidean distance computation.[7] This Euclidean distance is in the input space where the vectors $\vec{X}_i$ and $\vec{C}_j$ are defined. The distance computation process is called a survey because for a given city all nodes are surveyed to find the winner.

The Survey For each city i, find the node j_c which is closest to city i. Compute the Euclidean distance d_{ji} of each node j to city i, and select the minimum distance node, or winner, j_c, by competition.

$$d_{ji} = \sqrt{(x_{i1} - c_{j1})^2 + (x_{i2} - c_{j2})^2} \tag{8}$$

Move the winning node j_c and its neighboring nodes, where neighboring nodes are adjacent nodes in the output space (on the ring), towards city i. The distance each node moves is determined by the function $f(G, n)$, where G is a gain parameter, and n is the shortest distance measured along the ring between each node j, and the winner, j_c.[8] As G decreases to zero, only the winning node, j_c, moves toward city i. The larger G is, the greater the number of nodes that will move toward city i.

$$f(G, n) = (\frac{1}{\sqrt{2}}) \times \left(exp^{-(\frac{n^2}{G^2})}\right) \tag{9}$$

The distance measured along the ring, n, from the jth node to the winning node j_c, is calculated by Equation 10.

[7] An epoch is defined as one complete iteration through the list of cities, finding each of them a winner and taking M steps, starting from $i = 1$ and going to $i = M$.

[8] The choice of the function $f(G, n)$ is a candidate means of improving the performance of the network.

$$n = \text{infemum}[j - j_c(\text{modulo}N), j_c - j(\text{modulo}N)] \tag{10}$$

Infemum is defined as the greatest lower bound of a given set of numbers, in some instances such as this, it is the minimum value. Modulo is cyclic addition and subtraction. In this application, it is used to find the shortest path around a closed circular ring between two numbers, either clockwise or counterclockwise. For an example, refer to Figure 24. In this figure, the number of nodes, n, equals 6. If the winning node were $j_c = 3$, the modulo distances between nodes 2 and 3 would be computed as in Equations 11 and 12.

$$2 - 3 = -1[mod(6)] = 5 \tag{11}$$

$$3 - 2 = +1[mod(6)] = 1 \tag{12}$$

The infemum of the set of two distances is $inf(5, 1) = 1$. If the winning node were $j_c = 6$, the modulo distances between nodes 2 and 6 would be computed as in Equations 13 and 14.

$$6 - 2 = +4[mod(6)] = 4 \tag{13}$$

$$2 - 6 = -4[mod(6)] = 2 \tag{14}$$

The infemum of this set of distances is $inf(4, 2) = 2$.

Once the value for the position update function is known (specific to each node), the nodes are moved one at a time from their current positions at c^-, to new ones at c^+ according to Equation 15. For a graphical aid, refer to Figure 25.

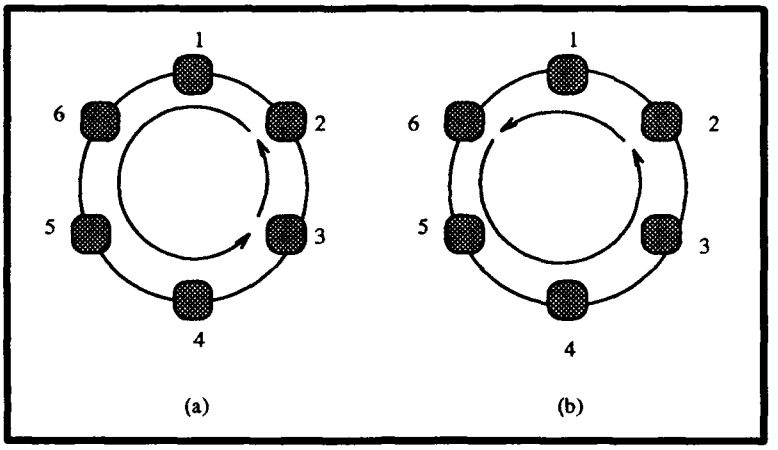

Figure 24. Modulo Example.

$$c^{+} = c^{-} + f(G,n) \times (\vec{X} - \vec{c^{-}}) \tag{15}$$

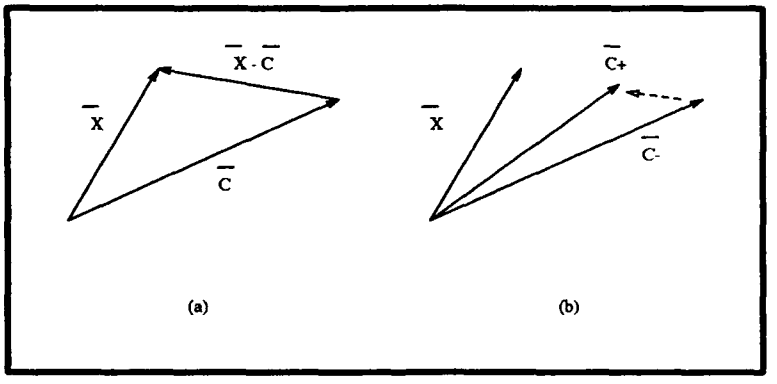

Figure 25. Node Update Example.

At the end of each epoch, the gain, G, is decreased according to Equation 16, where α is a constant fixed prior to each trial.[9]

$$G^{+} = (1 - \alpha) \times G^{-} \tag{16}$$

[9]Adjusting alpha is another candidate means of improving the network's performance.

Node Creation Process A node is duplicated if it is chosen as the winner, or closest node, for two different cities during the same epoch. A new node is created, or cloned, with the same coordinate location as the winner, j_c, but with the next available node number on the ring, j_{n+1}.[10] For a visual aid to understand this process, refer to Figures 26 and 27 to see the first epoch's node creation process.

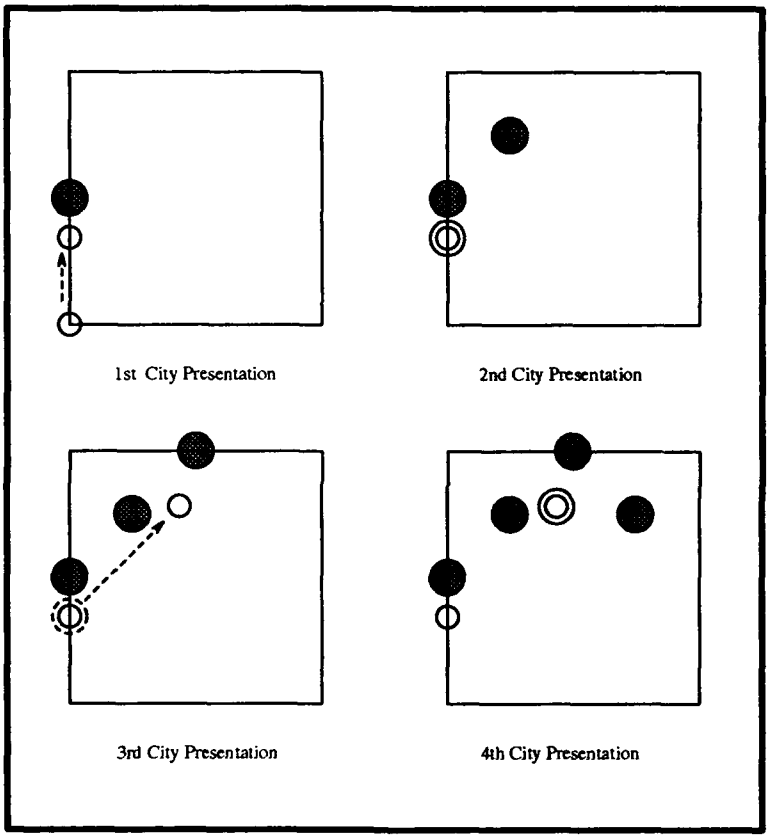

Figure 26. First Half of First Epoch's Node Creation Process.

To understand the node creation process it is necessary to understand inhibition, movement, and numbering. When an uninhibited node, j, wins for the first time in an epoch, becoming node j_c for that city, it is moved towards the city (along with its neighbors on the ring) and inhibited from moving again until the next epoch.

[10]This is where the algorithm used in this thesis differs from that of Angenoil. Their algorithm adds the node numbered adjacent the winning node vice the next higher number on the ring.

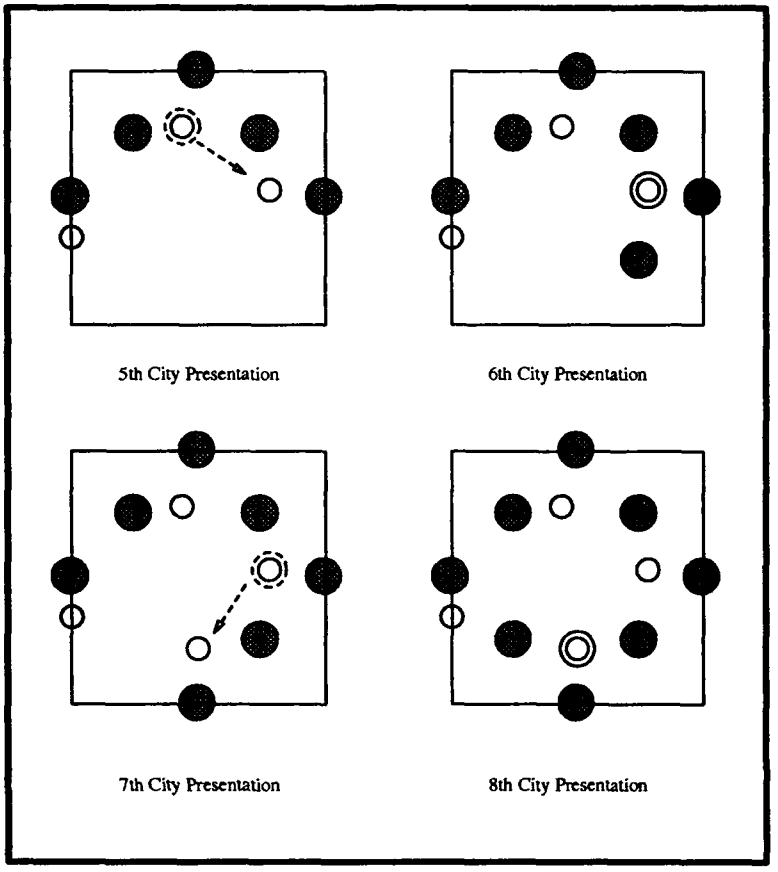

Figure 27. Second Half of First Epoch's Node Creation Process.

It is not inhibited from winning for a second city in the same epoch though. If it wins a second time in an epoch, a new node is created at the same location with the next available node number, j_{n+1}, but neither the winner or created node moves toward the city presented. When the next city is presented, the newly created node can compete to win for it, and will either be pulled away from its creator as a winning node or as a near-enough neighbor of a winning node. Having won twice in an epoch, the creator node is inhibited from both winning and moving as a neighbor until the start of the next epoch. As prevoiusly mentioned, the way the gain function works in the update process, not all of the nodes will have their positions significantly updated as near-enough neighbors of the winning nodes. As the gain

is successively decreased with each epoch, the size of the update neighborhood decreases and fewer nodes are updated along with the winner, until just the winners are updated, reference Equation 16.

Node Deletion Process A node is deleted if it is not chosen as a winner during three complete epochs.[11] When a node has just been created, its epoch-win counter does not start until the beginning of the next epoch.[12]

General Discussion The Kohonen algorithm is related to the Elastic Net Method (5), basically expanding and deforming a path to fit the shortest possible tour it can find. Instead of beginning with a ring at the center of the distribution of cities like the Elastic Net Method, this algorithm starts with a single node at a corner of the city space, moving it towards the distribution while adding and deleting nodes from the ring as necessary.[13] The Kohonen algorithm is also capable of handling an arbitrary matrix of city locations, something the Elastic Net Method is uncapable of, therefore it can be applied beyond just geometrical optimization problems.

Research Methodology This thesis investigated the use of these ideas for solving the traveling salesman problem, beginning here with three of the four candidate means of improving the Kohonen network performance. The first two means are altering the gain parameter and its step-size adjuster, alpha, and the third is adjusting the node deletion criteria down from three winless epochs to two.

[11] When this happened, the nodes were not realigned (shifted down one number in the ring), but the computer code was written to check the status of each column vector representing a node when computing modulo distance. If the status check of a column vector revealed an invalid node, it would not count the node, but would move on to the next node and check its status, adding one to the distance summation if it was indeed valid.

[12] This is another candidate means of improving network performance, investigating whether deleting nodes not chosen a winner in two or four epochs will lead to better or faster solutions.

[13] Of course, there is not a true "ring" of nodes until there are three or more of them in the city space.

The algorithm lacks a stopping criteria. Two means of stopping it were considered. The first was to set a sum total movement threshold (stopping the algorithm when the sum total movement of all nodes drops below a threshold, reference Formula 17), and the second was to set a minimum distortion tolerance (stopping the algorithm when there is a single node within a certain tolerance of every city, reference Formula 18). The sum total movement threshold was chosen as the stopping criteria for this network. When the sum total movement (calculated by the Euclidean distance formula in Equation 8) of all existing nodes reached less than or equal to 1×10^{-10}, the program was stopped. Since the network was only expected to converge to an eight node solution, 1×10^{-10} seemed to be a reasonable criteria.

$$\Sigma_{j=1}^{N}(\vec{C_j^+} - \vec{C_j^-})^2 \leq 1 \times 10^{-10} \tag{17}$$

$$\Sigma_{i=1}^{M}(\vec{C_j} - \vec{X_i})^2 \leq \text{some threshold} \tag{18}$$

Once the generic algorithm was coded in FORTRAN, compiled, and running properly, it was modified to meet the stopping criteria. The number of nodes created and deleted during each trial, the number of epochs taken until convergence was reached, and the final positions of the surviving nodes were saved. A standard set of city locations and presentation orders were used. Figure 28 shows the standard city layout, and Table 2 shows the 30 series of city presentation orders that were run against each network configuration. The cities were presented to the network one-by-one in the order shown in the table reading from left to right across each row.

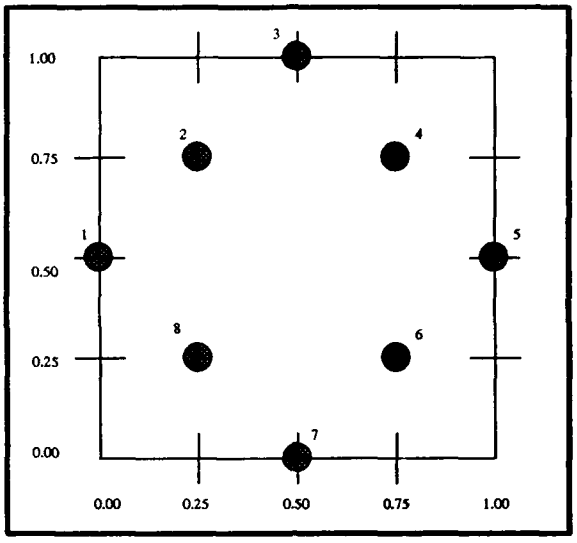

Figure 28. Eight-City TSP Locations

The algorithm was run against eight different configurations in order to learn which of the three performance improvement factors was most significant.[14] The eight different configurations are shown in Table 1.

The Christofides Algorithm

The Christofides algorithm was chosen for implementation rather than the Minimal Spanning Tree Method, because of its narrower bounded solution (less than 1.5 times the optimal solution versus less than 2.0 times the optimal solution). Recall, the intent of this portion of the thesis was to quantify the performance of the Hopfield and Kohonen neural networks against an accepted and competitive heuristic method.

Understanding the Algorithm No variant of the algorithm presented earlier in the literature review was necessary to solve the TSP. The four steps of the algorithm

[14]This treads on the edge of feature selectivity or saliency, which is another thesis topic by itself. We acknowledge awareness of it and awareness that features are generally not "most significant" across the entire feature space, but are dominant in given regions. For an in depth discussion of feature saliency, refer to the works of Dennis Ruck (14:40-48) and Casimir Klimasauskas (9:16-24, 78-84).

were easy to follow, and its solution form clear and unambiguous: an ordered list of arcs beginning at an origin city, connecting to every city in the problem only once, and finishing back at the origin. Computation of the tour length was simply a matter of adding up the n arc lengths chosen by the algorithm. Furthermore, there was no need to run the algorithm multiple times to generate a statistical database on the solution found: the algorithm will always converge to the same solution unless the problem data itself is changed.

Coding the Algorithm The only difficulty encountered in implementing the algorithm was figuring out how to code the concepts of "adjacency" and "closed path" into FORTRAN. Both of these hurdles were overcome through the liberal use of some strings, matrices, and a piece of knowledgeable advice from a resident faculty member (3). Adjacency was solved by simply building a matrix where each "tail" city had a line of entries indicating which cities could be reached by an existing arc originating in the desired "tail" city. Closed path was solved by labeling cities alike that could be reached along the same path. If an arc joined two cities labeled alike, it would create a closed path, and was therefore discarded. If the cities being joined were unlabeled, a new label was created. And finally, if the cities being joined were labeled differently, every city wearing the same labels as the two being joined were given a new label indicating their being joined into a single path.

Conclusion

This chapter has presented and discussed the three methodologies used in this thesis to collect data on the performance of the two artificial neural networks and the heuristic algorithm versus the traveling salesman problem.

The methodology of the Hopfield network was covered first: its $n \times n$ output matrix solution form, and how to interpret the solution from the matrix; the three constraints on the matrix form (only a single one in each column of the matrix, only

a single one in each row of the matrix, and the requirement that the output matrix not only be valid, but that it also be the optimal solution); differences between the the original Hopfield network and the Kashmiri variant; and lastly, how the original Hopfield network and Kashmiri variant were implemented versus the TSP.

The Kohonen network was presented second, beginning with five starting conditions and moving into the elements of the algorithm (the survey process, the node creation and deletion processes), its similarity to the Elastic Net Method, the choice of a stopping criterion, and lastly, how the network was implemented versus the TSP.

The Christofides Algorithm was the last of the methodologies discussed in this chapter. First was a short explanation of why it was chosen as the heuristic metric instead of the Minimal Spanning Tree Method (a narrower bounded solution than the MST Method), followed by a short discussion of its solution form (an ordered list of cities beginning and ending at an origin city), how its tour length was computed (adding up the n arc lengths chosen by the algorithm), and lastly, its deterministic solution (always converging to the same solution unless the problem data itself is changed).

Next chapter will cover the results of these three methods when applied to the traveling salesman problem. As in this chapter, the order of presentation will begin with the Hopfield network, followed by the Kohonen network, and Christofides Algorithm.

Table 1. Network Configurations

Configuration	Parameter Change	Alpha	Gain	Deletion Criteria
Control	None	0.2	1.0	3
1.	Alpha	0.02	1.0	3
2.	Alpha-Del	0.02	1.0	2
3.	Del	0.2	1.0	2
4.	Gain	0.2	1.5	3
5.	Gain-Alpha	0.02	1.5	3
6.	Gain-Alpha-Del	0.02	1.5	2
7.	Gain-Del	0.2	1.5	2

Table 2. City Orders Presented to Network.

Presentation Number:	1	2	3	4	5	6	7	8
Series Number								
1.	8	3	2	4	5	6	7	1
2.	8	7	6	5	4	3	2	1
3.	8	7	6	4	5	3	2	1
4.	8	2	3	4	5	6	7	1
5.	8	7	3	4	5	6	2	1
6.	1	3	5	7	2	4	6	8
7.	1	2	3	4	5	6	7	8
8.	1	2	8	3	7	4	6	5
9.	1	8	5	7	3	4	6	2
10.	8	5	2	4	3	6	7	1
11.	8	4	3	2	1	7	6	5
12.	8	4	3	2	1	5	6	7
13.	2	4	6	8	1	3	5	7
14.	2	3	4	1	5	6	7	8
15.	2	3	4	5	1	8	7	6
16.	2	3	8	7	1	6	5	4
17.	2	3	8	7	1	6	4	5
18.	2	3	8	7	6	1	4	5
19.	2	1	8	5	4	6	3	7
20.	5	3	1	7	8	2	4	6
21.	5	4	1	2	3	8	7	6
22.	5	4	6	3	2	8	7	1
23.	4	6	5	1	2	8	7	3
24.	4	5	1	6	2	8	7	3
25.	4	1	6	2	5	8	7	3
26.	4	5	6	3	2	1	8	7
27.	3	7	4	8	2	1	5	6
28.	3	1	2	4	5	6	8	7
29.	7	2	4	8	6	5	1	3
30.	7	8	6	2	1	3	5	4

IV. Results

Hopfield Results.

The Two Versions. Two versions of the Hopfield artificial neural network were run against a standard city distribution.[1] Hopfield and Tank's original configuration was the first, the other the variant developed by Sarwat Kashmiri at the Tennessee Technological Institute. As mentioned in Chapter Three, there are three major differences between the two techniques.

The first difference has to do with the computation of the neuron interconnection weights and their collective matrix. Hopfield's interconnection weight formula, Equation 4, uses fewer terms than Kashmiri's, Equation 7, and very distinctly calls out the fact that zeros are needed on the interconnection weight matrix diagonal. Kashmiri uses more terms to compute the interconnection weights claiming they ensure valid solutions, and does not specifically call-out the need for having zeros on the weight matrix diagonal.

The second difference is the use of an update routine within the network. Hopfield and Tank's original network does not conduct any updates of either the states of the neurons or the network output prior to being fed back into the input layer of the network. Kashmiri's configuration, however, calls for an update of the states of the network neurons prior to characterization.[2]

The third difference has to do with the choice of the output characterization function. The original Hopfield uses a hyperbolic tangent to generate a *sigmoid* output characterization function. The Kashmiri variant uses a *piece-wise linear* output characterization function, Chapter Three.

[1] Standard within this thesis for sake of comparing the different techniques.

[2] "Characterization" meaning the interpretation or categorization of the states of the neurons, or activations, by passing them through an output characterization function.

Hopfield Results In five separate instances, the original Hopfield network with a piece-wise linear output characterization function was run against a half-scale version of the problem and converged to the optimal solution.[3] The successes were limited though, since the five starting configurations were either at the global minima, or fairly near to it. These five runs confirmed the network's ability to remain at a global minima or converge to it given a close enough starting point. This network configuration was not very robust, since it converged to either invalid or wrong solutions given random initial inputs/starting conditions.

No update routine was used in any of the five successes, and in each of them, zeros were coded to appear along the interconnection weight matrix diagonal. The only differences between the starting conditions of the five runs were as follows:

- Instance 1: The baseline for the remaining four runs, this one was given the optimal solution as its initial starting point and as its previous network output, { (1 0 0 0), (0 1 0 0), (0 0 1 0), (0 0 0 1) }.
- Instance 2: This instance was given an initial input and previous network output *very* near the optimal solution, { (.99 0 0 0), (0 .98 0 0), (0 0 .99 0), (0 0 0 .98) }.
- Instance 3: This instance was given an initial input and previous network output where the order of two cities in the tour had been reversed { (0 1 0 0), (1 0 0 0), (0 0 1 0), (0 0 0 1) }.
- Instance 4: This instance was given an initial input and previous network output *fairly* near the optimal solution, { (.80 0 0 0), (0 .90 0 0), (0 0 .95 0), (0 0 0 .86) }.

[3] "Half-scale version" means the removal of the even-numbered cities in the problem: 2, 4, 6, and 8.

- Instance 5: This instance was given an initial input and previous network output *fairly* near the optimal solution *and with one value way off*, { (.80 0 0 0), (0 .90 0 0), (0 0 .95 0), (0 0 0 .01) }.

In the remaining runs of the network against random initial starting points, the network converged to either invalid or wrong solutions, or did not converge at all, getting caught in an infinite loop between two local minima.

Kashmiri Results. The Kashmiri variant of the Hopfield network was also run against a half-scale version of the problem and also met with limited success. The variant was used with an update routine for the states of the network neurons before being put through a piece-wise linear output characterization function. The variant network converged to the optimal solution only once, with the following starting conditions:

- Zeros were used along the weight matrix diagonal, despite not being called out in his research paper, and the network was given the optimal solution as its initial starting point and as its previous network output, { (1 0 0 0), (0 1 0 0), (0 0 1 0), (0 0 0 1) }.

The remainder of the Kashmiri variant runs either converged to invalid solutions or did not converge at all, oscillating between two local minima.

Discussion of Hopfield Network Results Based on these results, the Hopfield network is very capable of staying at either a local or global minima, demonstrating stability, but is extremely sensitive to where it starts out. This thesis did not explore that sensitivity or solutions to dealing with it, rather it concerned itself with the ease of use of artificial neural networks and quality of solutions found. Judging the Hopfield network by those two criteria, prior knowledge of the network, and the limited successes encountered, more work needs to be done clarifying how to apply this network to this class of problem.

Kohonen Results.

Configurations. The algorithm was run in eight different configurations against 30 different city presentation orders for a total of 240 independent trials. The data collected during these trials focused on three quantities: valid tour lengths; the number of epochs required for the configurations to reach convergence; and the number of nodes created during each trial. The data was divided into eight sets according to the configurations of the algorithm. The eight configurations were:

- CONTROL: the control set where alpha was set equal to 0.2, the deletion threshold was set at 3 winless epochs, and the gain parameter was set at 1.0.
- ALPHA: alpha was lowered from its control setting of 0.2 by a factor of ten to 0.02, and the other two parameters remained unchanged from their control settings.
- ALPHA-DEL: alpha was lowered from its control setting of 0.2 by a factor of ten to 0.02, the deletion threshold was lowered to 2 winless epochs, and the gain parameter remained unchanged from its control setting.
- DEL: the deletion threshold was lowered from its control setting of 3 to 2, and the other two parameters remained unchanged from their control settings.
- GAIN: the gain parameter was increased from its control setting of 1.0 by 50 percent to 1.5, while the other two parameters remained unchanged from their control settings.
- GAIN-ALPHA: the gain parameter was increased from its control setting of 1.0 by 50 percent to 1.5, alpha was lowered from its control setting of 0.2 by a factor of ten to 0.02, and the deletion threshold remained unchanged from its control setting.
- GAIN-DEL: the gain parameter was increased from its control setting of 1.0 by 50 percent to 1.5, the deletion threshold was lowered from 3 to 2, and alpha remained unchanged from its control setting.

- GAIN-ALPHA-DEL: the gain parameter was increased from its control setting of 1.0 by 50 percent to 1.5, alpha was lowered from its control setting of 0.2 by a factor of ten to 0.02, and the deletion threshold was lowered from 3 to 2.

Tours. Valid tours were defined as those finishing with only eight nodes, one at each city. Some solutions either missed cities or had more than one node located at a single city location. Out of the thirty different runs of each of the eight configurations, the number of valid tours found ranged from a low of 23 to a high of 29.

Epochs. Each of the 240 trials was allowed a maximum of 100 epochs to reach convergence. Convergence was defined to be the point at the end of an epoch where the sum total movement of all existing nodes during the epoch, by Euclidean distance, was less than or equal to 1×10^{-10}. The maximum number of epochs any of the 240 trials took to reach convergence was 91, while the minimum was just 10. The maximum and minimum average number of epochs required to reach convergence ranged from 14.8 to 89.9.

Tour Lengths. The optimum tour length for the set of eight city locations is 2.8284.[4] Given the constraint of the convergence threshold, the best individual tour length reached was 2.82843 (within 0.001 percent of optimum). The best average tour length for a single configuration against all thirty orders of city presentations was 2.94975 with a standard deviation of 0.35739, obtained by simultaneously increasing the gain from 1.0 to 1.5 and decreasing alpha from 0.2 to 0.02 (Configuration 5). The worst single valid solution reached was 5.70246. The worst average tour length was 4.51914 with a standard deviation of 0.50805, obtained by lowering the deletion criteria from three winless epochs to 2, while holding both alpha and gain at their control values (Configuration 3). These statistics were compiled on a soft-

[4]The pattern of cities laid out inside of a unit square is another square with sides of length 0.7071 (4 sides $\times$ 0.7071/side = 2.8284), and rotated 45 degrees from parallel to the standard x-y coordinate axes. For a look at the pattern, refer back to Figure 28.

ware spreadsheet using Formulas 19 and 20 for the mean and standard deviation, respectively.

$$\vec{X} = \frac{\Sigma_{i=1}^{30} X_i}{30} \tag{19}$$

$$\sigma = \frac{\Sigma_{i=1}^{30} X_i^2}{30} - \vec{X}^2 \tag{20}$$

where:

$\vec{X}$ = the mean

X_i = the individual tour lengths

σ = the standard deviation

Nodes Created. The mean number of total nodes created (but not necessarily all existing at the same time) until convergence ranged from a low of 10.2 to a high of 25.0.

These statistics are all shown in Table 3.

Discussion of Kohonen Network Results Despite a non-rigorous statistical treatment, it is still apparent from these results that alpha was the most significant of the three features looked at during the 240 trials. Every configuration that included a change of alpha from its control value yielded a significantly higher number of valid tours and shorter mean tour length than the control set. The gain parameter was also significant, but not to the same degree as alpha. The deletion criteria does not appear to be significant at all, since its lone results are not much different from the control set and its mean tour length even a bit longer at the second decimal place. When the deletion criteria was coupled with changed values of

Table 3. Statistical Results of 240 Trials

Configuration	Number of Valid Tours	Mean Tour Length (TL)	TL Standard Deviation	Mean # Nodes Created	Mean # Epochs Run
Control	23	4.41946	0.49206	10.6	14.8
Alpha	29	3.89735	0.55143	24.5	70.6
Alpha-Del	28	3.79422	0.61332	25.0	70.6
Del	25	4.51914	0.50805	10.2	15.0
Gain	28	4.17824	0.59317	11.0	16.3
Gain-Alpha	29	2.94975	0.35739	16.1	89.9
Gain-Alpha-Del	28	2.97192	0.36855	17.1	90.0
Gain-Del	28	4.15523	0.57733	11.3	16.4

alpha and or gain, it did not produce any noticeable positive change in results there either. In fact, its influence could be taken as slightly negative rather than neutral or positive. The conclusion drawn from these results is that it is desirable to take *longer* to converge, using either a very small value for alpha, a moderately larger starting value for the gain, or a combination of the two.[5]

Christofides Result

Given the particular distribution of cities in this traveling salesman problem (a square within a square of length one) the Christofides algorithm had no difficulty finding the optimal tour path, 1-2-3-4-5-6-7-8-1, with its length of 2.82843.[6] In fact, applied manually to the problem (Figures 29 through 32), the algorithm reached a solution visually recognizable as the optimal solution in only three steps, not the usually required four.[7]

[5]For further investigation into the saliency of these two parameters, refer to Appendix D.

[6]Given this particular distribution of cities, the distance between each city in the optimal solution is the same: the square root of $(0.25^2+0.25^2) = 0.35355339$. Eight times this length is 2.828427125, which gets rounded off to 2.82843.

[7]For a look at how well the Christofides Algorithm performed against a larger scale problem, refer to Appendix D.

Conclusions of Research Results

This chapter has presented the successes of all three methods in solving the traveling salesman optimization problem.

The chapter began by describing the capability of the original Hopfield network and the Kashmiri variant to stay at either a local or global minima, or converge to one if started near enough to it. The research also highlighted the networks' sensitivity to where they begin searching along the energy surface for the correct answer, sometimes converging to invalid solutions, other times not converging at all. No in-depth exploration was conducted to determine how better to apply the Hopfield network or its Kashmiri variant to the traveling salesman optimization problem.

The results of eight different Kohonen network configurations were presented second. These results discussed the feature saliency of the constant alpha in the gain parameter update equation, the starting value of the gain parameter itself, and the node deletion criterion. Trial results indicated a significant sensitivity to changes in alpha, an equal or somewhat lesser sensitivity to changes in the starting values of the gain parameter, and no appreciable sensitivity to changes in the node deletion criterion. The conclusion drawn from these results was that it is desirable to take longer to converge, using either a very small value for alpha, a moderately larger starting value for the gain, or even better, a balanced combination of the two.

The Christofides Algorithm converged to a single, deterministic result that matched the best single performance of the Kohonen network, finding the optimal solution to the eight-city traveling salesman problem. A non-automated application of the algorithm to the problem even delivered a visually recognizable solution before completion of all of its steps.

The next chapter will summarize the overall research conducted by this thesis and present the conclusions drawn from it.

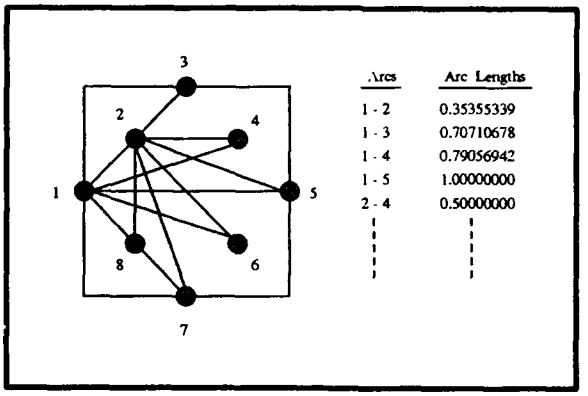

Figure 29. Distribution of Cities and Distances

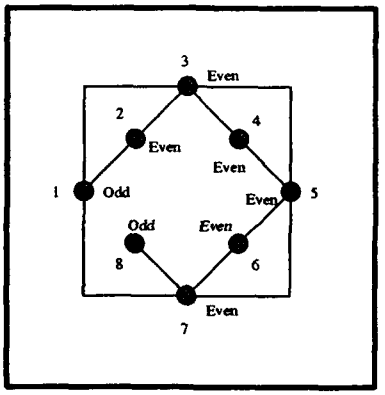

Figure 30. The Minimal Spanning Tree and Labeled Cities

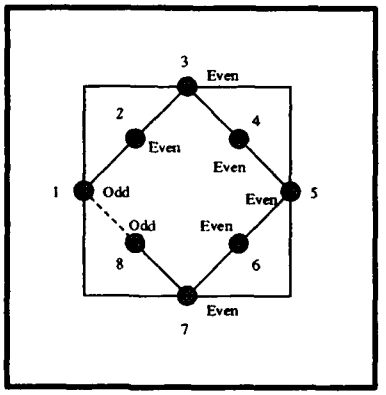

Figure 31. Connection of Nearest Odd-Labeled Cities

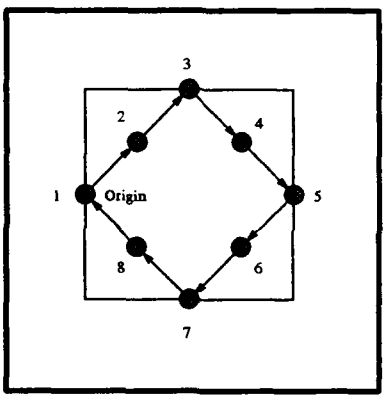

Figure 32. Short-Cut Routine and Christofides Method Solution

V. Conclusions

The primary goal of this thesis was to apply the Hopfield and Kohonen artificial neural networks against a classic optimization problem, and compare the solutions with those obtained by an accepted heuristic algorithm. This thesis was successful in applying both the Hopfield and Kohonen artificial neural networks, and the Christofides heuristic algorithm to solving the classic, traveling salesman optimization problem. Although there are many other classic optimization problems, most of them have been "well solved" where as the TSP has not been, and it remains a worthwhile challenge.

Results

Between the artificial neural networks, the Kohonen network proved most robust, by successfully solving an eight-city TSP for the optimal solution. The Hopfield network had only limited success in solving a half-scale version of the same problem (limited in the sense of only converging to the global minima if started at or near enough to it, and not being able to converge to it if given a random initial starting point). A variant of the Hopfield network was also successfully applied to solving the TSP, but suffered the same limitations as the original. The best *single* Kohonen network solution, found repeatedly throughout the 240 trials conducted, matched the Christofides solution which also proved to be the optimal solution to the problem. Across the eight different configurations investigated though, the average performance of the Kohonen network was consistently less than that of the Christofides Algorithm against the standard pattern chosen for this thesis. Given the simplicity of the particular pattern though, it would not be unreasonable to expect the Kohonen solutions to match or exceed the quality of the Christofides solutions against a more challenging pattern. Recall, the Christofides Algorithm applied manually to

the problem found a visually recognizable optimal solution in only three of its four steps.

Lessons Learned

Coding the Hopfield network in FORTRAN proved straight-forward and easier than coding the Kohonen network. The reason for this was the use of constant-size matrices needed to code the Hopfield network algorithm, versus the constantly changing size of matrices needed for the Kohonen network algorithm (due to the creation and deletion of nodes each epoch). This translates into easier hardware implementation for the Hopfield network than the Kohonen.

Coding the Christofides Algorithm was initially challenging due to the requirement of coding the concepts of "adjacency" and "closed path". After the initial uncertainty in understanding the algorithm, the coding task proved as easy as coding the Hopfield network.

All four methods were coded in FORTRAN and initially applied to only a half-scale version of the eight-city TSP (to facilitate debugging the programs and validating their results). Although none of the programs were written for dynamic data entry, it was fairly easy to edit the source code and scale it to the larger sized problem. The edits typically involved just changing the size of some matrices and counters, and not any additional code.

Looking just at the number of lines of FORTRAN code required for each method (wholly subjective, depending on the computer language and the programmer's ability in that language), the two Hopfield networks averaged 23 percent shorter than the Kohonen network and Christofides Algorithm.

With the exception of when the Hopfield networks were unable to converge to a solution and got caught in an infinite loop, all of the methods tested took just seconds or less to converge to a solution, and no method significantly outperformed all of the others. From knowledge of other research though, it is expected the Hopfield

network will outperform the Kohonen network in terms of speed to convergence, given that neither network has any starting advantage over the other.

Knowing what the optimal tour length was, it was possible to adjust three features of the Kohonen network algorithm to determine which the TSP was most sensitive to (feature saliency), and consistently improve the quality of the network's solutions.[1]

The Christofides Algorithm, being deterministic in nature, does not have any parameters that can be adjusted to improve, or narrow, the bounds on its solution. Although not the focus of this investigation, the Christofides Algorithm may benefit from the development of an improvement to the short-cut routine used in its fourth and final step: during the back-track search for an unvisited city to connect city $i+1$ with, instead of arbitrarily choosing a city from a plurality of choices adjacent to some previously visited city, chose the one closest city $i+1$.

In 1988, Angeniol demonstrated the ability of the Kohonen network to solve optimization problems, and in particular, the traveling salesman problem. The Hopfield network can also be applied to optimization problems other than the TSP. In order to do so, the network requires the design and use of a new energy function. An explanation of how to design a new energy function can be found in Wasserman (19).

The Christofides Algorithm was developed specifically for the traveling salesman problem but can also be extended to certain routing optimization problems while maintaining the bound on its solution (for non-Euclidean distance problems).

[1]The metric for measuring performance versus the TSP is simply distance: the shorter the better. Given some other type of optimization problem, it may not be so easy to identify which parameters can be changed to bring about the most improvement in network performance.

Contributions

This is the only known direct comparison of the Hopfield and Kohonen networks with a known and accepted heuristic method, the Christofides Algorithm.

This thesis investigated the saliency of three features in the Kohonen network algorithm: the gain parameter; the constant alpha in the gain parameter decrement formula; and the node deletion criteria. Alpha was identified as the most salient of the three parameters and the gain parameter was a close second. The node deletion criteria did not appear to be statistically significant. The conclusion drawn from the trial results is that it is desirable to take *longer* to converge to a solution, by using either a very small value for alpha, a moderately large initial gain parameter, or a balanced combination of the two.

This thesis also served as another data point for the known fact that the ability of the Hopfield network to converge to an optimal or near-optimal solution is highly dependent on where it starts its search and on the coefficients of the terms in the energy equation. Previous research has demonstrated that optimal or near-optimal solutions can be found by proper selection of the initial network conditions (18). Unfortunately for the network's usability, the problem of finding an optimal tour is often replaced by the problem of finding the best initial network conditions, and there is no systematic way to do that (18). Once found though, its results can be quite impressive.

If choosing an artificial neural network to apply to real-world, Air Force problems, based soley on this thesis, the Kohonen network is the preferred choice. Despite taking longer time to code and more lines of it than the Hopfield network, it was robust in finding optimal tours. If a clear, systematic means of determining the Hopfield energy function coefficients and starting conditions becomes known, the Hopfield network will then become the preferred choice due to its ease of use and brevity of coding.

Appendix A. *Evolution of the TSP*

The TSP is not a significant problem because hordes of salesmen are clamoring for an algorithm or because its mathematical model precisely fits numerous engineering or scientific applications. Its importance stems from the fact that it remains a "not-well-solved" example of the combinatorial optimization genre. In fact, it is "the most prominent of the unsolved combinatorial optimization problems ... and the most common conversational comparator ..." (10:2).

In the latter half of the nineteenth century, Irish mathematician Sir William Rowan Hamilton invented a system of noncommutative algebra he named *Icosian Calculus*. This, in turn, became the basis of a puzzle marketed as "The Icosian Game". Another version of the game, "in which the vertices of a solid dodecahedron represented important cities, was known as the 'Traveler's Dodecahedron'. A thread looped around pegs set at the vertices to form a cycle was 'a voyage around the world'." (10:3).

It is not known for certain who brought the TSP into mathematical circles, but Merrill Flood of Columbia University is credited for publicizing it within that and the operations research communities. In 1948, Flood was urged "to popularize the TSP at the RAND Corporation, at least partly motivated by the purpose of creating intellectual challenges for models outside the theory of games. In fact, a prize was offered for a significant theorem bearing on the TSP. There is no doubt that the reputation and authority of RAND, which quickly became the intellectual center of much of the operations research theory, amplified Flood's advertizing ... And, of course, the TSP became popular because it had a name that reminded people of other things. The traveling salesman was one of the classic personalities of American mythology, with a special chapter in the annals of ribald humor, and some of the disproportionate attention which the TSP has received in the world of combinarorial optimization must surely be credited to the resonances of its title" (10:5-6).

Appendix B. *NP-Completeness*

Definitions

To fully understand NP-Complete problems takes a rather lengthy explanation and begins with several definitions. First, a *problem* is a general question possessing several free variables, called *parameters*, whose values are undefined. An *instance* of a problem is obtained by specifying particular values for all of the problem parameters. An *algorithm* is a step-by-step procedure for solving a problem, and is only said to solve a problem if the algorithm can be applied to any instance of the problem and always produce a valid solution for that instance. The time required for an algorithm to converge to a solution is generally the most important metric for deciding whether or not to apply an algorithm in practice, and the most efficient algorithm is also usually the fastest. Time requirements of algorithms are expressed in terms of the *size* of a problem instance, which is meant to reflect the amount of input data needed to describe the instance. The *time complexity function* expresses the time requirements of an algorithm by giving the largest amount of time needed by the algorithm to solve a problem instance for each possible input length (6:4-6).

Efficiency

Different algorithms have different time complexity functions, and characterizing which are "too inefficient" or "efficient enough" depends on the application. Furthermore, efficiency depends on the method of computation used, for example, the efficiency of pencil and paper versus those on a hand-held calculator, personal computer, or Cray super-computer. Depending on the application, the important metric can be either *the number of individual calculations* an algorithm necessitates, or *the measured time* in which a given piece of computer hardware and software can return a valid solution. Some computer hardware is designed to handle a lot of computations, for example, those with a math co-processor chip, while others are not.

For those that are not, an algorithm requiring a lot of individual computations will be very time inefficient.

Polynomial and Exponential Time Distinction

There is a major distinction that can be made between algorithms on the basis of the measured-time metric. The distinction is that between polynomial-time and exponential-time algorithms, relating the increase in the number of calculation iterations of an algorithm to the linear increase in the number of problems parameters. If the defining element of what a function's value will be when its primary variable is increased to infinity is polynomial, the algorithm is said to be polynomially bounded, and is known as a polynomial-time algorithm. If an algorithm is not polynomially bounded as its primary variable is increased to infinity, it is known as an exponential-time algorithm.[1]

The distinction between these two types of algorithms is particularly significant when faced with a large problem instance. For a comparison of polynomial-time and exponential-time complexity functions, refer to Table 4. The first four functions beneath the Time Complexity Function heading are polynomial and the bottom two are exponential.

The time differences in Table 4 show answers for polynomial time algorithms can be calculated orders of magnitude quicker than exponential-time algorithms, clearly indicating why polynomial-time algorithms are more desirable than exponential-time algorithms. In the mid-1960's, polynomial-time algorithms became equated with "good" algorithms, and an argument was raised that certain problems might not be solvable by such "good" algorithms, and in fact is usually the case. For

[1] For a more esoteric definition, let a function $f(n)$ be denoted by $O(g(n))$ whenever there exists a constant c such that $|f(n)| \leq (c)|g(n)|$ for all $n \geq 0$. A *polynomial time function* is a function whose time complexity is $O(p(n))$ for some polynomial function p, where n is used to denote the input length. Algorithms whose time complexity functions do not meet this criteria are called *exponential-time algorithms.*

Table 4. Polynomial-time versus Exponential-time Complexity Functions. (6:7)

Size of n	10	20	30	40	50
Time Complexity Function					
n	.00001 second	.00002 second	.00003 second	.00004 second	.00005 second
n^2	.0001 second	.0004 second	.0009 second	.0016 second	.0025 second
n^3	.001 second	.008 second	.027 second	.064 second	.125 second
n^5	.1 second	3.2 seconds	24.3 seconds	1.7 minutes	5.2 minutes
2^n	.001 second	1.0 second	17.9 minutes	12.7 days	35.7 years
3^n	.059 second	58 minutes	6.5 years	3855 centuries	$2x10^8$ centuries

these certain problems, either the insight necessary for developing a polynomial-time algorithm is lacking, or else their nature is such that they simply cannot be solved by one.

> "Most exponential algorithms are merely variations on exhaustive search, whereas polynomial time algorithms generally are made possible only through the gain of some deeper insight into the structure of a problem. There is wide agreement that a problem has not been "well-solved" until a polynomial time algorithm is known for it. Hence, we shall refer to a problem as *intractable* if it is so hard that no polynomial time algorithm can possibly solve it" (6:8).

The Naming of NP-Complete Problems

In 1971, Stephen Cook wrote a paper focusing attention on the class of non-deterministic polynomial (NP) decision problems (whose solutions are either "yes" or

"no" answers) by proving that every other NP problem can be polynomially reduced to one particular NP problem called the "satisfiability" problem. If the satisfiability problem could be solved with a polynomial-time algorithm, then so could every other problem in NP. Since this whole class of intractable problems can be reduced to the satisfiability problem, then the satisfiability problem must also be intractable. Therefore, the satisfiability problem must be the single hardest problem in the NP class.

Subsequent to Cook's work, Richard Karp proved that the decision problem versions of many well known combinatorial problems, *including the traveling salesman problem*, are just as hard as the satisfiability problem. Since Karp's proof in 1972, "a wide variety of other problems have proved equivalent in difficulty to these problems, and this equivalence class, consisting of the 'hardest' problems in NP, has been given a name; the class of *NP-complete problems*" (6:14).

Appendix C. *Hopfield Energy-Surface Proof*

In two-dimensional space and on an $x - y$ coordinate system, picture a continuous random line segment with several peaks and valleys along its length. This two-dimensional "surface" can be thought of as the energy function or energy surface. In order for the network to search for and find the global minima of the function, it must somehow move along the energy surface in search of a point without either a positive or negative slope, a minima. To do this, take the derivative of the energy function, E, using the product rule of calculus.

$$E = \frac{-1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} OUT_i OUT_j \tag{21}$$

$$\frac{dE}{dOUT_k} = \frac{-1}{2} \left[\sum_{i=1, i \neq k}^{n} W_{ik} OUT_i + 2W_{kk} OUT_k + \sum_{j=1, j \neq k}^{n} W_{kj} OUT_j \right] \tag{22}$$

where: the second term inside the square brackets goes to zero because $W_{kk} = 0$. Breaking the summations out term by term yields:

$$\begin{aligned} \frac{dE}{dOUT_k} = \frac{-1}{2} [& W_{1k} OUT_1 + W_{2k} OUT_2 + \cdots \\ & + W_{k-1k} OUT_{k-1} + W_{k+1k} OUT_{k+1} + \cdots + W_{nk} OUT_n + \cdots \\ & + W_{k1} OUT_1 + W_{k2} OUT_2 + \cdots \\ & + W_{kk-1} OUT_{k-1} + W_{kk+1} OUT_{k+1} + \cdots \\ & + W_{kn} OUT_n] \end{aligned} \tag{23}$$

Since the W_{ij} matrix is symmetrical (that is, $W_{ij} = W_{ji}$), the above expression reduces to just:

$$\frac{dE}{dOUT_k} = -\sum_{i=1, i \neq k}^{n} W_{ik} OUT_i \tag{24}$$

This formula shows how the neuron-update equation searches along the energy surface looking for a minima, which, if the energy function has been properly chosen to correspond to the constraints and solutions of the problem, will be the solution to the TSP (13:1).

Appendix D. *42-City Problem Results*

Problem Identification

The Christofides Algorithm and Kohonen Artificial Neural Network, both successful in solving the 8-city TSP for its optimal solution, were also applied to a 42-city problem instance. The 42-city problem was taken from Dantzig, Fulkerson, and Johnson (4). The problem began as an arrangement of 49 cities, one in each of the 48 states and Washington D.C., Table 5.[1] A subset of 42 cities was focused on after they realized that seven of the 49 cities lay directly enroute between two cities in the subset, and did not add any complexity to the problem. Dantzig provided a table of inter-city distances for those in the subset. The true distances were transformed to integers less than 256 by Equation 25 and round-off, to "permit compact storage of the distance table in binary representation" (4:394).

$$d_{ij} = \frac{1}{17} \times (d'_{ij} - 11) \tag{25}$$

where:

d'_{ij} = road distance in miles from City I to J.

The optimal tour for this problem follows the numbers given beside each of the cities in Table 5, and has a length of 12,345 miles, or 725 adjusted distance units.

[1]Dantzig was published in 1954, and recall, Alaska and Hawaii did not gain statehood until 1959.

Table 5. Set of 42 Cities

Optimal Tour Position	City	Optimal Tour Position	City
1.	Manchester, NH	22.	Denver, CO
2.	Montpelier, VT	23.	Cheyenne, WY
3.	Detroit, MI	24.	Omaha, NE
4.	Cleveland, OH	25.	Des Moines, IO
5.	Charleston, WV	26.	Kansas City, MO
6.	Loiusville, KY	27.	Topeka, KS
7.	Indianapolis, IN	28.	Oklahoma City, OK
8.	Chicago, IL	29.	Dallas, TX
9.	Milwaukee, WI	30.	Little Rock, AR
10.	Minneapolis, MN	31.	Memphis, TN
11.	Pierre, SD	32.	Jackson, MS
12.	Bismarck, ND	33.	New Orleans, LA
13.	Helena, MT	34.	Birmingham, AL
14.	Seattle, WA	35.	Atlanta, GA
15.	Portland OR	36.	Jacksonville, FL
16.	Boise, ID	37.	Columbia, SC
17.	Salt Lake City, UT	38.	Raleigh, NC
18.	Carson City, NV	39.	Richmond, VA
19.	Los Angeles, CA	40.	Washington D.C.
20.	Phoenix, AZ	41.	Boston, MA
21.	Santa Fe, NM	42.	Portland, ME

Christofides Methodology and Results.

The methodology for solving the 42-city TSP with the Christofides Algorithm was no different than for solving the 8-city TSP. The FORTRAN code had to be modified slightly to accomodate the larger storage requirements for the increased number of cities, but the algorithm remained unchanged. The inter-city distance matrix provided by Dantzig proved convenient for the Christofides Algorithm, negating the need for doing those computations inside of the program, and eliminating a few lines of computer code.

The Christofides Algorithm solved the 42-city TSP for a tour length solution of 929 adjusted distance units, well beneath the 1,088 given by the solution bound of 1.5 $\times$ the optimal solution.

Kohonen Methodology and Results.

Methodology Like the Christofides Algorithm, the methodology for solving the 42-city TSP with the Kohonen network algorithm was no different than for solving the 8-city TSP. The FORTRAN code had to be modified slightly to accomodate the increased number of cities, but the algorithm remained unchanged. However, unlike the Christofides Algorithm, the Kohonen network algorithm does not readily accept its data in the form of inter-city distances. It needs two-dimensional coordinates for each of the cities so that its nodes can move about in two-dimensional space and capture each city. So instead of using the atlas to find inter-city distances like Dantzig did, the atlas was used to find latitude and longitude coordinates for each city. Degrees of west longitude were converted to degrees of longitude east of the international dateline so that the city space was in the upper right quadrant of the $X - Y$ coordinate axes with their origin fixed at the intersection of the international dateline and the equator. Lastly, the degrees of latitude and longitude were converted to minutes, and divided by 10,000 to bring all of the coordinates between zero and one.

Configurations. The Kohonen network algorithm was run in seven different configurations against 30 different city presentation orders for a total of 210 independent trials. The data collected during these trials focused on just two quantities: the number of valid tours found; and the mean tour lengths of the valid tours. The

data was divided into seven sets according to the configurations of the algorithm. The seven configurations, different than those used against the 8-city problem, were:[2]

- CONTROL: the control set where alpha was set equal to 0.2, the deletion threshold was set at 3 winless epochs, and the gain parameter was set at 1.0.
- ALPHA: alpha was lowered from its control setting of 0.2 by a factor of ten to 0.02, and the other two parameters remained unchanged from their control settings.
- ALPHA-GAIN (A): alpha was lowered from its control setting of 0.2 to 0.02, the gain parameter was increased from its control setting of 1.0 by 50 percent to 1.5, and the deletion threshold remained unchanged from its control setting.
- ALPHA-GAIN (B): alpha was lowered from its control setting of 0.2 by a factor of 100 to 0.002, the gain parameter was increased from its control setting of 1.0 to 1.5, and the deletion threshold remained unchanged from its control setting.
- ALPHA-GAIN (C): alpha was lowered from its control setting of 0.2 to 0.02, the gain parameter was increased from its control setting of 1.0 by a factor of 2 to 2.0, and the deletion threshold remained unchanged from its control setting.
- DEL: the deletion threshold was lowered from its control setting of 3 to 2, and the other two parameters remained unchanged from their control settings.
- GAIN: the gain parameter was increased from its control setting of 1.0 to 1.5, while the other two parameters remained unchanged from their control settings.

Results. Valid tours were defined as those finishing with exactly 42 nodes, one at each city. Some solutions either missed cities or had more than one node located

[2]The configuration GAIN-ALPHA-DEL was dropped from consideration against the 42-city TSP given its lack of significant difference from the GAIN-ALPHA configuration in the 8-city TSP results.

at a single city location. Out of the thirty different trials of each configuration, the number of valid tours found ranged from a low of 0 to a high of 11. As expected from the 8-city results, changes in alpha and combinations of changes in alpha and the gain proved most successful, Table 6.

The shortest individual tour length reached was 803 adjusted distance units (slightly less than $1.108 \times$ optimum), by the ALPHA-GAIN (C) configuration, which also delivered the highest number of valid tours, 11, and the shortest mean tour length, 849. Of the three configurations able to find valid tours, the ALPHA configuration delivered the longest individual tour length, 973, the longest mean tour length, 924, and the fewest number of valid tours, 7.

Table 6. Statistical Results of 210 Trials

Configuration	Number of Valid Tours	Mean Tour Length (TL)	Average TL Times Optimal	Minimum Tour Length	Maximum Tour Length
Control	0	-	-	-	-
Alpha	7	924	1.274	889	973
Alpha-Gain (A)	9	865	1.194	808	938
Alpha-Gain (B)	0	-	-	-	-
Alpha-Gain (C)	11	849	1.171	803	909
Del	0	-	-	-	-
Gain	0	-	-	-	-

Again, despite a non-rigorous statistical treatment, it is still apparent from these results that alpha was the most significant of the three *individual* parameters. Whereas the other ' parameters were individually uncapable of finding any valid tours, the ALPHA configuration found 7. Although the individual gain parameter configuration did not find any valid tours, raising doubts about its saliency, its synergistic effect when combined with changes in the parameter alpha proved once again to be most salient. A word of caution though, because as evidenced by the failure of the ALPHA-GAIN (B) configuration to find any valid tours, the choice

of values for alpha and gain must be a balanced choice backed up by thorough investigation of the limits of the saliency region.

Conclusions

The conclusions drawn from these results are that:

- while the Christofides Algorithm can consistently deliver competitive length and proven-bound solutions, on average, the Kohonen network is capable of delivering even better results against larger-scale traveling salesman problems.
- in order to get these better results, a balanced combination of changes in the alpha and gain parameters is required.

Bibliography

1. Akiyama, Y., A. Yamashita, M. Kajiura, and H. Aso, "Combinatorial Optimization With Gaussian Machines", *Proceedings of the IJCNN90-San Diego.* Vol I, pp. 533-540. 1990.
2. Angeniol, Bernard, Gael De La Croix Vaubois, and Jean-Yves Le Texier, "Self-Organizing Feature Maps and the Travelling Salesman Problem", Neural Networks, Vol I, 1988, pp.289-293.
3. Borsi, Major John J., Instructor of Operations Research. Personal interviews. Air Force Institute of Technology, School of Engineering, Department of Operations Research, Wright-Patterson AFB, OH, January through October 1992.
4. Dantzig, G., R. Fulkerson, and S. Johnson, "Solution of a Large-Scale Traveling Salesman Problem", Journal of the Operations Research Society of America, Vol 2, 1954, pp. 393-410.
5. Durbin, Richard, and David Willshaw, "An Analogue Approach to the Travelling Salesman Problem Using an Elastic Net Method", Nature, 16 April 1987, pp. 689-691.
6. Garey, Michael R. and David S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, San Francisco, California, 1979, pp. 4-14.
7. Hopfield, John J., and David W. Tank, "Computing With Neural Circuits: A Model", Science, 8 August 1986, pp.625-633.
8. Kashmiri, Sarwat, "The Travelling Salesman Problem and the Hopfield Neural Network", *Proceedings of the IEEE SOUTHEASTCON '91.* Vol. II, pp. 940-943. 1991.
9. Kilmasauskas, Casimir C., "Neural Nets Tell Why", Dr. Dobbs Journal, April 1991, pp. 16-24 and 78-84.
10. Hoffman, A.J., P. Wolfe, "History", Edited by E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys, *The Traveling Salesman Problem*, John Wiley and Sons Ltd, Chichester, Great Britain, 1985, pp. 1 - 6.
11. Peterson, Ivars, "Netting a Better Sales Route", Science News, 25 April 1987, pp.262.
12. Port, Otis, "Computers That Come Awfully Close to Thinking", Business Week, 2 June 1986, pp. 92-97.
13. Priddy, Captain Kevin, "Hopfield Nets", Unpublished paper, 8 March 1990.

14. Ruck, Dennis W., Steven K. Rogers, and Matthew Kabrisky, "Feature Selection Using a Multilayer Perceptron", Journal of Neural Network Computing, Fall 1990, pp. 40-48.

15. Rumelhart, D. E., J. L. Mc Clelland and the PDP Research Group, "Parallel Distributed Processing Vol. 1 Chap. 7", MIT Press, 1986.

16. Tagliarini, G. A., J. F. Christ, and E. W. Page, "Optimization Using Neural Networks", IEEE Transactions on Computers, Vol 40, No. 12, December 1991, pp. 1347-1358.

17. Tou, J.T. and R.C. Gonzalez, *Pattern Recognition Principles*, Addison-Wesley Publishing Company, Reading, MA, 1974, pp. 6.

18. Van den Bout, David E., and T.K. Miller, "A Traveling Salesman Objective Function That Works", IEEE International Conference on Neural Networks, Vol 2, 1988, pp. 299-303.

19. Wasserman, Philip D., *Neural Computing. Theory and Practice*, Van Nostrand Reinhold, New York, 1989.

20. *Webster's 3rd New Colegiate Dictionary*, G & C Merriam Company, Publishers, Springfield, Massachusetts, 1969, pp. 124 and 1174.

21. Wilson, G.V., and G. S. Pawley, "On the Stability of the Travelling Salesman Problem Algorithm of Hopfield and Tank", Biological Cybernetics 58, 1988, pp. 63-70.

22. Yoshihara, Takafumi, and Toshiaki Wada, "Optimization by Extended LVQ", *Conference Proceedings of the IEEE.* pp. 407-414.

Vita

Capt Gruner was born August 20, 1959 in Chicago, Illinois. He graduated from Maine Township High School West in 1977. Following high school he earned an Associate of Science Degree in Engineering from Oakton Community College, a Bachelor of Science Degree in Mathematics and Natural Sciences from the University of Wyoming, and a Master of Science Degree from the University of Southern California in Systems Management.

Capt Gruner entered the Air Force via the reserve officers training corps program at the University of Wyoming, and was commissioned a second lieutenant in December of 1983. He was assigned to the Air Force Satellite Control Facility, Sunnyvale AFS, California, from 1984 to 1988. There he served as the Satellite Operations Director for the Defense Satellite Communications System (DSCS) Phase II constellation, Data Systems Evaluator, Chief of Exercise Evaluation, and Deputy Mission Director for Contingency Operations.

Capt Gruner was reassigned to the 4th Satellite Communications Squadron (Mobile), Holloman AFB, New Mexico, from 1988 to 1991. There he served as a Space Systems Crew Commander, Test Director, and Chief of Deployment Site Control.

In 1991, Capt Gruner was reassigned to the Air Force Institute of Technology as a degree candidate in space operations.

Permanent address: 1028 Webster Lane
Des Plaines, Illinois 60016

December 1992 Master's Thesis

COMPARISON OF ARTIFICIAL NEURAL NETWORKS WITH A CONVENTIONAL HEURISTIC TECHNIQUE FOR OPTIMIZATION PROBLEMS

Jeffrey S. Gruner
Captain, USAF

Air Force Institute of Technology, WPAFB OH 45433-6583

AFIT/GSO/ENG/92D-01

This research investigates the utility of the Hopfield and Kohonen artificial neural networks to the traveling salesman optimization problem. A third, non-neural-network technique (the Christofides Algorithm - a competitive, bounded-solution operations research technique) is also investigated for comparison to the artificial neural network solutions. An eight and forty-two city distribution are chosen for comparison of the solutions.

Neural Networks, Christofides Algorithm, and Optimization 87

UNCLASSIFIED UNCLASSIFIED UNCLASSIFIED UL