

INTEGRATING NEURAL NETWORKS AND GIS FOR SOLVING THE TRAVELING SALESMAN PROBLEM

By

RITA JACK IBRAHIM

A Thesis

**Submitted in Partial Fulfillment of the
Requirements for the degree of Master of
Science in Computer Science**

**Department of Computer Science
Faculty of Natural and Applied Sciences
Notre Dame University- Louaize
Zouk Mousbeh, Lebanon**

June 2000

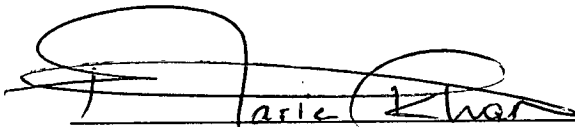
Integrating Neural Networks and GIS for Solving the Traveling Salesman Problem

By
Rita J. Ibrahim

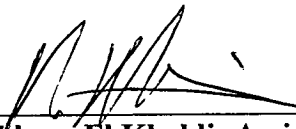
Approved:



Fouad Chedid: Associate Professor of Computer Science and Chairperson.
Advisor.



Marie Khair: Assistant Professor of Computer Science.
Member of Committee.



Khaldoun El Khaldi: Assistant Professor of Computer Science.
Member of Committee.



Jean Fares: Associate Professor of Mathematics and Chairperson
Member of Committee.

Date of thesis defense: July 5, 2000

ACKNOWLEDGEMENT

I feel indebted to many persons who were helpful to me in performing this work. So, rather than trying to thank everyone, I mention only four groups of people.

The first group is my family and specially my mother, Amal, for her support and encouragement and for my uncle, Tony Safi, for pushing me when it was important.

The second group consists of Khatib & Alami Company, headed by Mr. Jacques Ekmekji who gave me full supports, time wise and technology wise to accomplish this work successfully. Also, I thank my friend Michel Bridi for his help.

The third group consists of Dr. George Eid, Dean of the Faculty of Natural and Applied Sciences at NDU, for directing my academic studies and to everyone involved in the department of computer science.

Finally, I am deeply grateful to my advisor, Dr. Foad Chedid, for supervising and following up this work and for spending time with me to put this manuscript in its final form. A word of thanks also goes to the committee members: Dr. Marie Khair, Dr. Khaldoun El Khaldi, and Dr. Jean Fares.

Thanks again because without these peoples this work would not be accomplished.

Finally, I like to dedicate this work to my passing away father Dr. Jack Ibrahim who were an exemplary human being for me.

ABSTRACT

In this thesis, we investigate the use of neural networks for solving the Traveling Salesman Problem (TSP). First, we review the main elements of the theory of NP-completeness. Then, we explain what makes some problems computationally intractable. We review some heuristic approaches used to provide near-optimal solutions to NP-complete problems. Then, we introduce the topic of neural networks and describe some of the most popular neural network models. We pay a special attention to a recent model, named the Hybrid Neural Network model (HNN), used for solving optimization problems and the Hybrid Network Updating Algorithm (HNUA). We propose a modification version of the HNUA that modify the HNUA to produce an optimal to near-optimal solutions and demonstrate its efficiency using a specific NP-complete problem known as the Traveling Salesman Problem (TSP) as an example.

Our simulation shows how the modification version derives an efficient result. Also, a comparison is made between the results derived from the proposed modification model and the Depth First Search (DFS) model both for TSP to analyze and deduce the degree of optimization and validation for the proposed modification.

Besides, we build a TSP interface application using Geographic Information System (GIS) MapObject (MO) on Microsoft Visual Basic environment to create mapping application and adding mapping functionality for a better visualization of the problem where the user can easily view the cities and observe the resultant tour geographically.

TABLE OF CONTENTS

LIST OF TABLES	vi
LIST OF FIGURES	vii
LIST OF FIGURES IN APPENDIX A.....	ix
LIST OF FIGURES IN APPENDIX B.....	x

CHAPTERS

I. DEFINING THE PROBLEM.....	1
II. COMPUTATIONAL COMPLEXITY.....	3
2.1. COMBINATORIAL EXPLOSION	3
2.2. DETERMINISTIC VS. NON-DETERMINISTIC COMPUTATIONS	4
2.3. THE THEORY OF NP-COMPLETENESS	4
III. ARTIFICIAL NEURAL NETWORKS.....	7
3.1. HISTORICAL BACKGROUND	7
3.2. INTRODUCING NEURAL NETWORKS	8
3.3. ARCHITECTURE OF ANN	12
3.4. SOME EXAMPLES OF THE MOST POPULAR NEURAL NETWORKS	15
3.5. PROPERTIES AND CAPABILITIES	23
IV. HOPFIELD NETWORKS FOR SOLVING OPTIMIZATION	
PROBLEMS.....	25
4.1. HOPFIELD NETWORK	25
4.2. ENERGY FUNCTION	28
4.3. REVIEW OF THE MAIN ALGORITHM.....	29
4.4. RESULTS	30
V. A HYBRID NEURAL NETWORK MODEL FOR SOLVING	
OPTIMIZATION PROBLEMS.....	32

TABLE OF CONTENTS (Continued)

5.1. OPTIMIZATION PROBLEM REPRESENTATION AND TRANSFORMATION METHOD	32
5.2. THE HYBRID MODEL	36
5.3. THE HYBRID NETWORK UPDATING ALGORITHM	38
5.4. THE MODIFIED HYBRID NEURAL NETWORK UPDATING ALGORITHM.....	39
VI. THE HYBRID NEURAL NETWORK MODEL APPLIED TO	
THE TRAVELING SALESMAN PROBLEM.....	42
6.1. THE TRAVELING SALESMAN PROBLEM.....	42
6.2. NEURAL NETWORK REPRESENTATION FOR TSP	43
6.3. SIMULATION RESULTS AND ANALYSIS	44
VII. TSP INTERFACE APPLICATION USING GISMAPOBJECT	51
7.1. GIS AND MAPOBJECT DEFINITION AND INTRODUCTION	51
7.2. OBJECTIVES OF USING MAPOBJECT.....	52
7.3. TSP INTERFACE APPLICATION	53
VIII. CONCLUSION AND FUTURE RESEARCH.....	56
APPENDIX A	
A.1. TRAVELING SALESMAN PROBLEM WITH $N = 4$	58
A.2. TRAVELING SALESMAN PROBLEM WITH $N = 10$	63
APPENDIX B	
B.1. TSP INTERFACE APPLICATION USING GIS-MAPOBJECT	77
REFERENCES	

LIST OF TABLES

Table	Page
1. Characteristics of Artificial Intelligent and Artificial Neural Network.....	11
2. Differences between the Digital Computers and Neural Networks.....	12
3. Comparing both HNUA and MHNUA Models.....	45
4. Average Percentage of Optimization for MHNUA Model.....	47
5. The Different Characteristics of the DFSA Model versus MHNUA Model.....	50

LIST OF FIGURES

Figure	Page
1. Components of a Natural Neuron.....	8
2. The Synapse between Neurons.....	9
3. The Artificial Neuron Model.....	10
4. A Single-Layer ADALINE Network.....	15
5. A Multi-Layer Adaline (Madaline) Network.....	16
6. A Backpropagation Network.....	17
7. A Sigmoid Function.....	18
8. Different Kinds of Activation Functions.....	18
9. A Recurrent Neural Network.....	19
10. A Hopfield Network.....	20
11. Architecture of a Hopfield Network.....	20
12. A Kohonen Network.....	22
13. A 7x5 Kohonen Array Showing the Size of the Adaptation Zone.....	22
14. Another drawing for the Hopfield Network.....	26
15. Matrix Representation of the TSP Problem.....	27
16. The Structure of Hybrid Neural Network.....	36
17. The Architecture of the Hybrid Neural Network Model.....	37
18. Possible Combination for Undirected and Directed graph for N=4.....	43
19. A Matrix Array of 4 x 4 Neurons to Represent a Tour.....	43
20. Show the Two Sets T and T* derived from Δc_{xi} and $\partial \text{Cost} / \partial c_{xi}$	44
21. The State Space Tree for DFS.....	46
22. DFSA Output Result for N = 10.....	47
23. Average Percentage of Optimization for MHNUA Model.....	47
24. Showing the Distance Matrix for N = 5 and its DFS and MHNUA Tour Results....	48
25. The Different Characteristics of the DFSA Model versus MHNUA Model.....	50

LIST OF FIGURES (Continued)

Figure	Page
26. Showing the Distance Matrix for $N = 10$ and its DFS and MHNUA Tour Results..	49
27. GIS Five Components.....	51
28. GIS Geographic Layer Representation.....	51
29. TSP User Interface Application Form.....	51

LIST OF FIGURES IN APPENDIX A

Figure	Page
1. DFSA Output Result for $N = 10$	76

LIST OF FIGURES IN APPENDIX B

Figure	Page
1. Login Form.....	77
2. Map Contents Form.....	78
3. TSP Toolbar.....	78
4. Spatial Selection Form.....	79
5. The Selection Cities Highlighted in Yellow Color.....	80
6. Path Menu Form.....	80
7. Multi Cities Form.....	81
8. Running Mathematica TSP Application.....	81
9. The Traveling Salesman Resultant Tour.....	82

CHAPTER I

Defining the Problem

Most optimization problems have been shown to be NP-complete [3], [8], [10], [12]. This means that unless $P = NP$, most of these problems would continue to have a running time that is exponential in the size of the input. Such optimization problems are difficult to solve because of the large number of possibilities that any algorithm has to go through in order to find an optimal solution. One of these problems is the traveling salesman problem (TSP). In TSP, a salesman must visit n cities. He wishes to make a tour starting at a particular city, visiting every other city exactly once and finishing at the city of origin. The main issue here is that the salesman must do so following the shortest possible tour. Clearly, for an undirected graph with n cities, there are $(n-1)!/2$ possible tours and for directed graphs there are $(n-1)!$ possible tours. This $n!$ function grows very rapidly as the value of n increases. For example, for $n = 60$, trying all $59!$ possible tours would require more than 300 centuries even with the assistance of a super computer. Faced with this kind of difficulty, most researchers have proposed heuristic algorithms to deal with NP-complete problems. These heuristic algorithms do not necessarily lead to optimal solutions, but usually they produce near-optimal solutions in a reasonable amount of time. Among these are special purpose heuristics applicable only to the TSP and general methods useful in a variety of optimization problems, such as Branchand-Bound (BB) [19], Local Search Algorithms (LSA) [17], Tabu Search (TS) [6], Simulated Annealing (SA) [20], Elastic Nets (EN) [5], Genetic Algorithms (GA) [17] and Neural Networks (NN) [1], [2], [4], [12], [15]. For a better visualization of the TSP problem we built a GIS-MapObject application that uses mapping and GIS components to view and observe the resultant salesman tour. Such application is used to model our proposed MHNUA model on a graphical user interface (GIS) so that this model can be used on a real world streets for traveling salesman problem.

The rest of this thesis is organized as follows: In Chapter II, we introduce the theory of NP-completeness. In chapter III, we describe the architecture of the neural network, advantages and disadvantages, properties and capabilities and give some examples of the most popular neural network models. In chapter IV, we describe the Hopfield neural network model, energy function and algorithm. In chapter V, we emphasize a particular recent model called the hybrid neural network model, used to solve optimization problems and proposed a modifying version of this model. In chapter VI, we demonstrate our work using the traveling salesman problem and include the simulation results of our tests. In chapter VII, we integrate the proposed model MHNUA to Geographical user interface application using (GIS-MO) to view and observe the resultant salesman tour. In chapter VIII, we give our conclusion about the proposed model and the possible future researches.

CHAPTER II

Computational Complexity

This chapter introduces the theory of computational complexity emphasizing the importance of the traveling salesman problem. Also, this chapter shows how combinatorial optimization problems suffer from exponential time complexity in the worst case. Besides, it defines the computational process in both deterministic and non-deterministic systems. Finally, it introduces the theory of NP-completeness and focuses on the classes: P, NP, NP-complete and NP-hard.

2.1. Combinatorial Explosion

The traveling salesman problem has some theoretical importance in complexity theory since it is one of the problems in the NP-Complete class. NP-complete problems are intractable in the sense that they seem to defy fast solutions. They are also known to be equivalent to each other, if you knew how to solve one NP-complete problem, you could solve every other problem in the class.

Recall that the number of possible solutions for the TSP with n cities is greater than 2^n [3]. Such a problem cannot be solved by searching all possible solutions for the best one, so various approaches like heuristics, rules and pruning have to be used to try to reduce the number of options to be tried. In TSP, the complexity seems to explode as the problem gets bigger that is as the number of possible tours increases [7]. Typically, in TSP if the number of cities to be visited is of size n , then the possible solutions are of the order $n!$ or n^n , that is if you pick a particular city as the first city to be visited, then there are $(n-1)$ choices for the second city, $(n-2)$ choices for the third, and so on. Because of the combinatorial nature of these problems the time needed to solve them grows exponentially and therefore for large size.

2.2. Deterministic vs. Non-deterministic Computations

In a deterministic system, the state the system is in exactly determines what state it is going to be in next. In a deterministic system, if we are told what state the system is in now, we can simply calculate what state the system will be in in a moment's time, and at any time in the future. The reader should understand that, in a nondeterministic system, the state the system will be in in a moment time is still predictable, but where it will be in the more distant future is almost impossible to predict.

In a non-deterministic computation, the complexity is defined in terms of the most efficient (with respect to time) accepting computation. Since the program is allowed to guess an accepting computation it might as well be allowed to guess the most efficient accepting computation (informal guessing).

2.3. The Theory of NP-Completeness

2.3.1. The Class P

The class P refers to the set of all decision problems whose solutions can be solved deterministically in polynomial time. Class P admits $O(n^k)$ polynomial solutions on a deterministic computer [7]. For example, finding the largest value in a list of items takes $O(n)$ time, a polynomial of order 1. Another example is selection sort which sorts a list into descending order which runs in $O(n^2)$ time, a polynomial of order 2. These problems belong to the class P.

What about problems which are not in P? The most dramatic type is the kind of problems which solution requires exponential amount of time: this is a problem for which the solution requires time determined by an exponential function of the size of the problem. One class of exponential problems comprises those search problems that suffer from the

combinatorial explosion.

More formally, P is the class of decision problems (Languages) L such that there is a polynomial time function $F(x)$ where x is a string and $F(x) = \text{True}$ if and only if x is in L.

2.3.2. The Class NP

The class NP consists of all problems which seem to defy polynomial time solutions on a deterministic computer. However, once you have a guessed solution, you can check that it is correct in polynomial time. The class NP admits $O(n^k)$ polynomial time solutions on non-deterministic computers and $O(k^n)$ exponential time solutions on deterministic computers [4]. An example is the TSP problem. The time needed to find the shortest tour increases exponentially as the number of the city increases. However, you can test whether a tour is shortest by looking at that tour once, and confirming that it is the shortest. In conclusion, whether a problem is P or NP depends on the algorithm chosen to solve it. Thus some NP problems can actually be P problems if we can solve them in polynomial time.

More formally, NP is the class of decision problems (languages) L such that there is a polynomial time function $F(x, c)$ where x is a string, c is another string whose size is polynomial in the size of x , and $F(x, c) = \text{true}$ if and only if x is in L.

2.3.3. NP-complete and NP-hard Classes

A problem is NP-hard if it is harder than any other problem in NP. Now, if a NP-hard problem happens to be in NP then it is called NP-complete. The traveling salesman problem is also NP-hard [8]. NP problems are not necessarily unsolvable. They only take extreme amount of time for very large problems on a deterministic computer: for a small version of the problem, the solution can be very fast. Note that, if you can solve a NP

complete problem in polynomial time, then $P = NP$. Also, if you prove that any NP-complete problem is not solvable in polynomial time then, $P \neq NP$.

More formally, A decision problem L is said to be NP-hard if, for every problem L' in NP, L' is polynomially reducible to L .

A problem A is said to be reducible to another problem B , if and only if, for every instance x of A , we can construct in polynomial time, another instance $F(x)$ of B such that a solution to $F(x)$ gives a solution to x and visa versa.

More formally, A decision problem L is said to be NP-complete if it is both NP-hard and in NP.

CHAPTER III

Artificial Neural Networks

This chapter provides a general introduction to neural networks. Concepts and architectures of various neural network models are introduced. Some of the most popular neural networks are described. These networks are: ADALINE and MADALINE networks, Backpropagation networks, Recurrent networks, Hopfield networks and Kohonen networks. Finally, the properties and capabilities of neural networks are listed and briefly described.

3.1. Historical Background

A biological neural network is a collection of neurons which are living nerve cells. For example, the cortex of the brain is a neural network. Somehow, such a network of neurons can think, feel, learn and remember. In the past, many investigators attempted to build models to study neural networks. These models fall into two categories: biological modeling and technological modeling. In biological modeling the goal is to study the structure and function of the brain in order to explain biological data the brain on aspects such as behavior and learning. In technological modeling the goal is to study in order to extract concepts to be used in new computational methodologies [7].

Many tasks which seem simple for us, such as reading a handwritten note or recognizing a face, are difficult for even the most advanced computer. In an effort to increase the computer's ability to perform such tasks, programmers began designing software to act more like the human brain, with its neurons and synaptic connections. Thus, the field of artificial neural network was born.

The objectives of research in Artificial Neural Networks may be paraphrased as follows. The first objective is to understand how the brain imparts abilities such as perceptual interpretation, associative recall, and common sense reasoning and learning. Towards this goal it is necessary to understand how computations are organized and carried out in the brain. These computations are of different kinds than the formal manipulation of symbolic expressions (traditional AI). The second objective is to understand the subclass of neural network models that emphasize computational power rather than their biological fidelity. To achieve this objective, it is admissible to incorporate features in a model even if those features are not neurobiologically possible.

3.2. Introducing Neural Networks

3.2.1. Biological or Natural Neural Networks

Much is still unknown about how the brain trains itself to process information. The human brain is the most complex organ in the human body. It consists of many (about 10^{13}) single cells, each of which is connected to a number, between 10 and 10^3 , of other cells in the brain. In the human brain, a typical neuron collects signals from others through dendrites. The neuron sends out spikes of electrical activity through a long tube-like fiber known as an axon, which splits into thousands of branches. So a biological neuron (brain cell) may look something like the figure below.

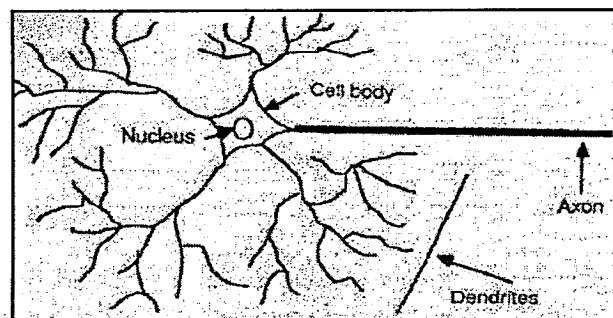


Figure 3.1: Components of a Natural Neuron

At the end of each branch, a structure called a synapse converts the activity from the axon into electrical effects that inhibit activity in the connected neurons. When a neuron receives excitatory input that is sufficiently large compared with its inhibitory input, it sends a spike of electrical activity down its axon [4]. Learning occurs by changing the effectiveness of the synapses so that the influence of one neuron on another changes. So synapse connection between biological neuron may look something like the figure below.

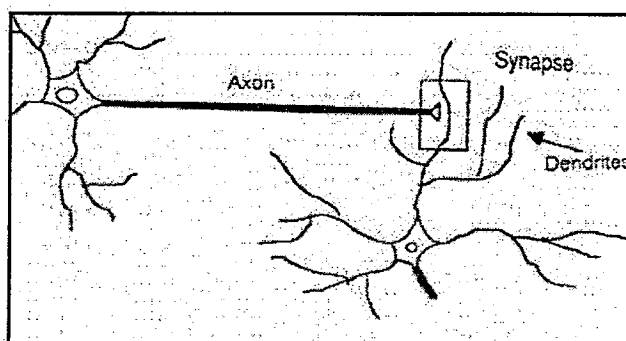


Figure 3.2: The Synapse between Neurons

3.2.2. Artificial Neural Networks

Artificial Neural Networks (ANNs) are presented as the computation equivalent of how we believe the neurons of the human brain work. Similar to a biological neural network, ANNs consist of many single units, each of which is connected to many other units using weighted links. The output of these units is passed onto units further on in the network. Input is similar to the dendrites and output is similar to axon. In ANNs, a unit takes its input, then performs some mathematical equations on the resultant figure, and passes the result along the output. Other (one or more) units as part of their input pattern may then pick up this output. So a typical artificial neuron model may be depicted as shown in figure below.

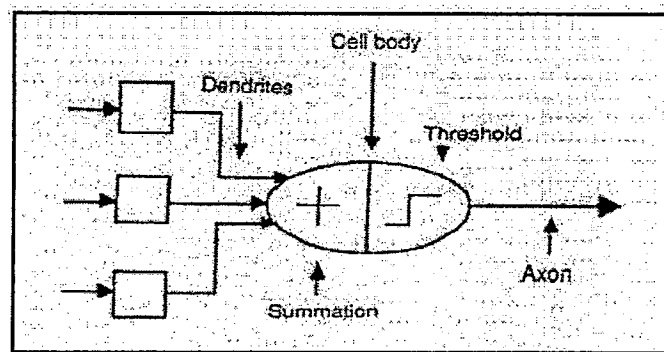


Figure 3.3: The Artificial Neuron Model

Neural networks are used as a theoretical model for parallel processing in particular an array of neuron where every element is a single processing unit. Neural networks are:

1. Biological models: Models of what goes on in nerve cells, both as individual cells and as small groups.
2. Cognitive models: cognitive-science models of how we think.
3. Connectionist models: models systems for computation, based on nerves and their behavior.

Neural networks have had most success in solving two types of problems: highly connected problems and classification problems. Connected problems are ones in which the solution to one part of the problem is intimately connected to the solution to the other parts. Traveling salesman problem is of this sort, that is the city you visited next depends on which cities you have visited so far.

Neural networks are of interest to artificial intelligence (AI) for two reasons. First, they are massively parallel computers, and hence offer advantages of speed in computation intensive tasks; second, they offer a different approach to computing than the serial-processing, rule-based algorithmic approach of traditional AI [13]. This has substantial potential for solving some intractable problems.

Artificial neural systems are unlike artificial Intelligence. Artificial intelligence programs use deductive reasoning to apply known rules to produce outputs. Each new situation may require another rule to be implemented. The program becomes large and complicated to address all possible situations. Artificial neural systems however automatically construct associations based upon the result of known situations. The characteristics of both the artificial intelligent and artificial neural network are summarized in the table below.

Artificial Intelligent systems	Artificial neural network systems
1. Imitation of the human reasoning process	1. Imitation of the structure and function of the brain
2. Sequential information processing	2. Parallel information processing
3. Explicit knowledge representation	3. Implicit knowledge representation
4. Use of deductive reasoning	4. Application of inductive reasoning to the processing of knowledge
5. Learning outside of the system	5. Learning occurs in the system

Table 3.1: Characteristics of Artificial Intelligent and Artificial Neural Network.

Artificial neural networks are unlike digital computers. Neural network offers a different way to analyze data, and to recognize patterns within that data, then digital computing methods. Digital computing method work well for problem that can be characterized. Digital computers are ideal for many applications. They can process data and track inventories. These applications do not need the special characteristic of neural network. While artificial neural networks offer a completely different approach to problem solving. They try to provide a tool that both programs itself and learns on its own [7]. The difference between digital computers and neural networks are shown in the figure below.

Digital Computers	Neural Networks
1. Deductive Reasoning. We apply known rules to input data to produce output.	1. Inductive Reasoning. Given input and output data, we construct the rules.
2. Computation is centralized, synchronous, and serial.	2. Computation is collective, asynchronous, and parallel.
3. Memory is packed and location addressable.	3. Memory is distributed, internalized, and content addressable.
4. Not fault tolerance One transistor goes and it no longer works.	4. Fault tolerance, redundancy, and sharing of responsibilities.
5. Fast. Measured in millions of a second.	5. Slow. Measured in thousands of a second
6. Exact.	6. Inexact.
7. Static connectivity.	7. Dynamic connectivity.

Table 3.2: Differences between the Digital Computers and Neural Networks

3.3 . Architecture of ANN

3.3.1. Advantages

3.3.1.1 . Parallel Processing

The topology of ANN offers a good model for parallel processing system in which the nodes or neurons in a neural network operate in parallel. Parallel processing is of interest to neural network specially for solving computational complexity problems, which requires great amounts of computer time. If that time can be telescoped into a parallel-processing solution, a lot of run time can be saved. Also, some problems are

better solved by parallel methods. For example, the traveling salesman problem is of particular interest because of its parallel implementation.

3.3.1.2 . Distributed Processing

Neural networks are also distributed processors in their own. The key distinction, which makes some parallel processing applications into distributed processing, is that the computation is distributed among a number of processors. Therefore, neural networks are called parallel-distributed processing systems. Clearly, it is not possible to say which neuron in a network is doing any part of a computation, since an algorithm does not describe the computation process. Rather, the whole network has an activity, which represent the progress of the computation, without being able to point to a particular neuron as performing any basic operation. Thus as well as distributing our program across a lot of processors, we distribute the basis of the computation itself.

3.3.1.3. Connectionism

The connections of neurons in a neural network are based on a biological idea. That is, neurons are linked by any number of connections to other elements that behave in a manner similar to nerve cells. In these kinds of networks, the connection carried no code information. Instead, information is coded according to which connections exist between elements of the network, not by the messages passing along them or by the memory states of the elements themselves. This means that it can be hard to decode the network's output, so networks can be arranged so that a particular connection signals a specific result or concept. Many neural network simulations could be viewed as programs for manipulating matrices, the network is being represented as an array of boxes representing all possible connections between the neurons that are filled with numbers representing the connections actual strength. This emphasizes the importance of the connection network, rather than the detail of how you draw a picture of it. Connectionist networks can come up with solutions to some problems very quickly where conventional

computers are very slow. Such problems have huge numbers of solutions, and are highly interconnected, so a decision about the solution to one part of the problem can affect many other parts. One good example is the traveling salesman problem.

3.3.2. Disadvantages

3.3.2.1. Heuristic Algorithms

An algorithm is considered efficient and good if it takes a reasonable amount of real-time to execute. The running time function is expressed as a function in the size of the input. In conventional programming languages like C, the programmer first designs an algorithm to solve the problem. This is then translated into a computer program, thus enable the computer to solve the problem too. On the other hand, problems developed for Artificial Intelligence (AI) do not put emphasis on algorithms. One class of problems solved by neural networks is combinatorial optimization problems, where most algorithms require exponential amount of time for large instances of the problem. This means that, for small problems the algorithm will function very fast. But for larger problems the algorithm would function very slowly as to be wholly impractical. Heuristic algorithms are therefore the only practical substitutes even if they occasionally give the wrong result.

Although this technique does not rely on algorithmic methods in quite the same as conventional programming, this is not to say that algorithms are not used at all. In search-based problems, the program has to look at a large number of different ways of solving the problem, algorithms are used to make sure that all possible ways of solving the problem are analyzed systematically and efficiently until the answer is found. One type of ANNs, which have been used to solve optimization problems, is called Hopfield Networks. As a result, neural networks showed that it can be used in a range of optimization problems, and able to find a good near-optimal solution in a few steps [2], [15].

3.4. Some Examples of the Most Popular Neural Networks

3.4.1. Supervised Learning

Supervised learning is a process of training a neural network by giving it examples of the task we want it to learn. Such as, learning with a teacher. In supervised training the network is given both input data and desired output data (correct answers as examples). After each trial, the network compares its own output with the right answers (desired one), correct any differences that is the errors are then propagated back through the system, causing the system to adjust the weight, and tries again, until the output error reaches an acceptable level.

3.4.1.1. ADALINE Network

The Adaptive Linear Neuron (ADALINE) is considered one of the earliest and simplest neural networks. It was the first neural network to be applied to a real world problem. Basically, it is a single-layer backpropagation network (Backpropagation is an abbreviation for the backward propagation of error). It can be modeled as shown in the figure below:

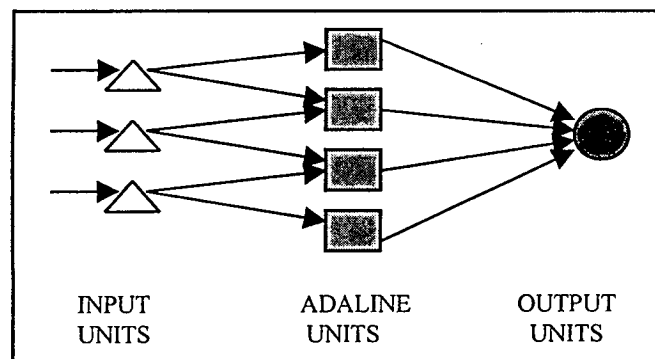


Figure 3.4: A Single-Layer ADALINE Network

ADALINE network consists of three layers, the first layer is the input layer which received inputs from outside. The second layer is a single hidden layer called the Adaline

layer which received multiple input from the input layer and transmit them to the output layer. The output layer a single processing element that receives multiple input, that is a single artificial neuron that takes its inputs from many other similar units. ADALINE assumes a linear relationship between input and output. The inputs to the unit take bipolar values (0 & 1) and include a bias. A bias is simply a dummy input unit whose output value is always 1. The output of an Adaline unit is usually a bipolar value too.

As shown in the figure above that, several Adaline units arrange in a single layer, each connected to a layer of input units. This is called a single-layer Adaline. But when Adaline units takes their input from other Adaline units, they are called multi-layer Adaline networks, Madaline as shown in figure below.

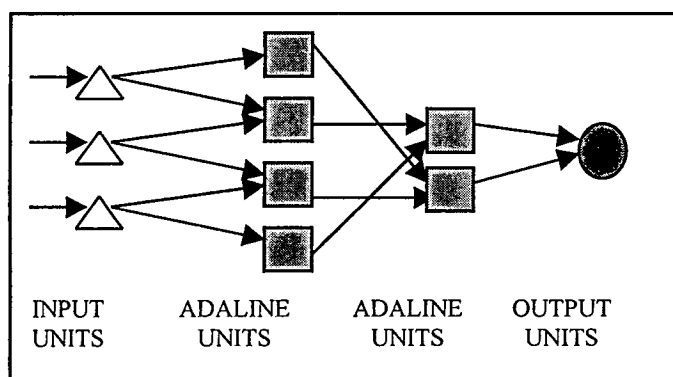


Figure 3.5: A Multi-Layer Adaline (Madaline) Network

3.4.1.2. Backpropagation Network

As mentioned in the previous section, an ADALINE is a single-layer backpropagation network. However, backpropagation networks are multi-layer networks; i.e. they are typically organized in layers. Layers are made up of a number of interconnected nodes that contain an activation function. The network consists of an input layer, a number of hidden layers and an output layer. The outputs of each node in a layer are connected to the inputs of all of the nodes in the subsequent layer. A Backpropagation Network is shown in figure below

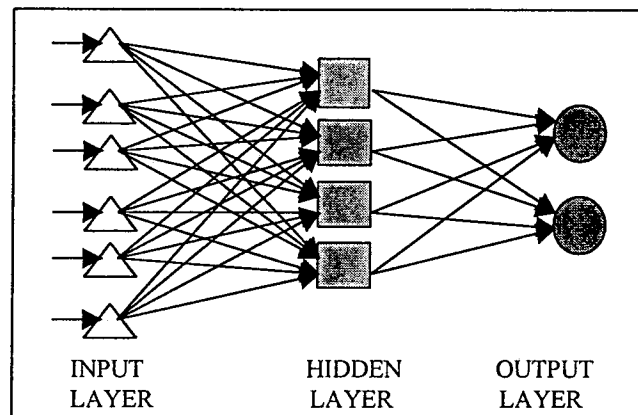


Figure 3.6: A Backpropagation Network

Most artificial neural networks contain some form of learning rule that modifies the weights of the connections according to the input patterns that it is presented with. In a sense, artificial neural networks learn by example, as do their biological counterparts; a child learns to recognize dogs from examples of dogs. Although there are many different kinds of learning rules used by neural networks, this section is concerned only with one named the delta rule [14]. With the delta rule, as with other types of backpropagation rules, learning is a supervised process that occurs with each cycle (i.e. each time the network is presented with a new input pattern) through a forward activation flow of outputs, and the backwards error propagation of weight adjustments. Clearly, with backpropagation algorithm a set of input and corresponding output data is collected that the network is needed to learn. An input pattern is applied to the network and an output is generated. This output is compared to the corresponding target output and an error is produced. The error is then propagated back through the network, from output to input, and the network weights are adjusted in such a way as to minimize a cost function, typically the sum of the errors squared. The procedure is repeated through all the data in the training set and numerous passes of the complete training data set are usually necessary before the cost function is reduced to a sufficient value.

Therefore, the process of training a neural network involves adjusting the input weights on each neuron such that the output of the network is consistent with the desired output. Within each hidden layer node, there is an activation function that polarizes network activity and helps stabilize it. In backpropagation network, the activation function chosen is the sigmoid function [14], which compresses the output value into the range between 0 and 1. A typical example is shown in the figure below.

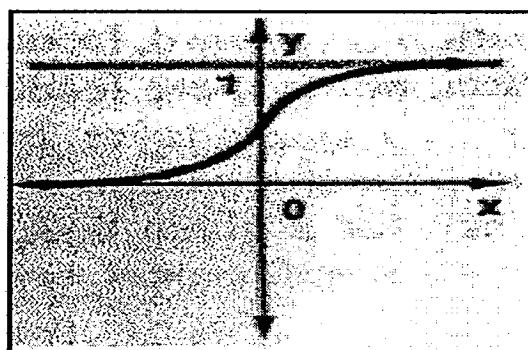


Figure 3.7: A Sigmoid Function

We note that various types of activation functions are found, which are shown in the figure below.

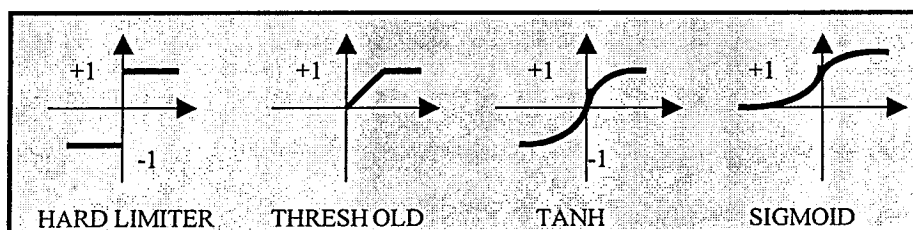


Figure 3.8: Different Kinds of Activation Functions

Once a neural network is trained to a satisfactory level it may be used as an analytical tool on other data. To do this, the user no longer specifies any training runs and instead allows the network to work in forward propagation mode only. New inputs are presented to the input pattern where they filter into and are processed by the middle layers as though training were taking place, however, at this point the output is retained and no backpropagation occurs. The output of a forward propagation run is the predicted

model for the data that can then be used for further analysis and interpretation. Therefore, backpropagation networks learn to predict future activities from the historical data collection.

3.4.1.3. Recurrent Network

Recurrent neural networks are called such because every neuron in the hidden layer has feedback paths from its outputs to every other unit in the hidden layer except itself, therefore, they have feedback paths from their outputs back sent to every unit in the network except the original. This means that after applying input, the output is calculated and then turned back to modify the input. The output is then recalculated. This process repeats itself successively until the network reaches a stable state. such networks actually receive two types of input: One is from the current incoming data and the other from the state information at the preceding time that is fed back to the network. This gives recurrent network the capability of integrating temporal information dating back to the starting point using a limited number of neurons. A recurrent network is shown in the figure below.

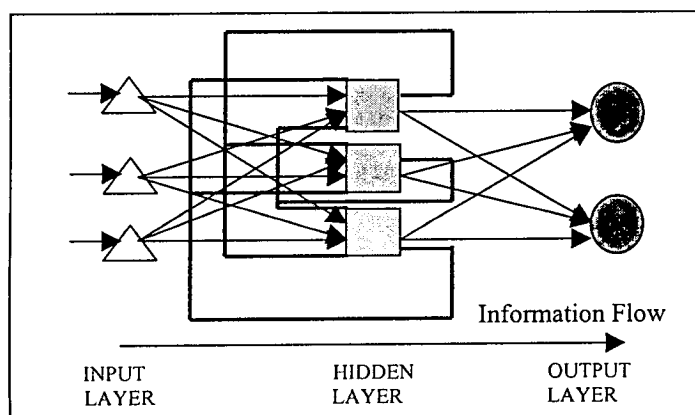


Figure 3.9: A Recurrent Neural Network

3.4.1.4. Hopfield Networks

Hopfield networks are commonly used for solving optimization problems. They

consist of two layers: an input layer and a Hopfield layer. Each node in the input layer is directly connected to only one node in the Hopfield layer. The nodes in the latter layer are neuron models with either hard limiting [4] or sigmoid activation functions [14]. The output of these nodes are weighted and fed back to the inputs of all of the other nodes. A Hopfield network is shown in the figure below.

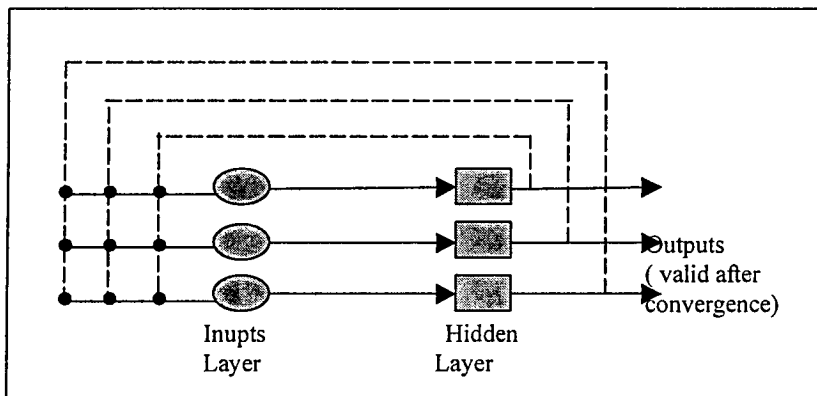


Figure 3.10: A Hopfield Network

The topology of the Hopfield network differs from other networks mentioned previously, whereby every unit is connected to every other unit except itself. In addition, the connections are bi-directional (information flows in both directions along them), and symmetric, there is a weight assigned to each connection which is applied to data moving in either direction. Some other drawings show a Hopfield network as a complete connected graph, as shown in the figure below.

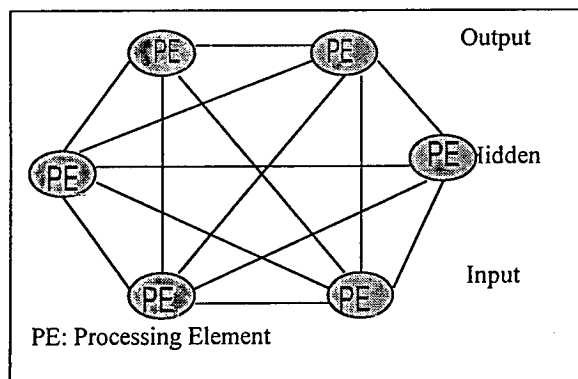


Figure 3.11: Architecture of a Hopfield Network

Hopfield Networks are useful both for auto-association and for optimization tasks (such as the combinatorial best route for a traveling salesman problem [9]). By encoding each hypothesis as a unit and encoding constraint between hypotheses by weights. Hopfield network is a recurrent network where stability is the major issue.

3.4.2. Unsupervised Learning

In Unsupervised learning, the network is given input data but no desired output data; instead, after each trial or series of trials it is given a grade or performance score that tells it how well it is doing. Kohonen neural networks use this type of learning. Supervised learning process is done to the network by changing the state of connectivity. Some involve adding and retrieving connection as well as changing their weight values.

3.4.2.1. Kohonen Network

The Kohonen network has self-organizing properties and is capable of recognition. Application examples of the Kohonen network include recognition of images and speech signals. The main distinguishing feature of this network, from previous discussed networks is that no output data is required for training. The Kohonen Network consists of a single Kohonen layer of nodes plus an input layer. Each input is connected to each and every neuron on the Kohonen layer which are organized in a two dimensional grid. A typical example of a Kohonen network with 2 input and 25 neurons on the Kohonen layer is shown in the figure below.

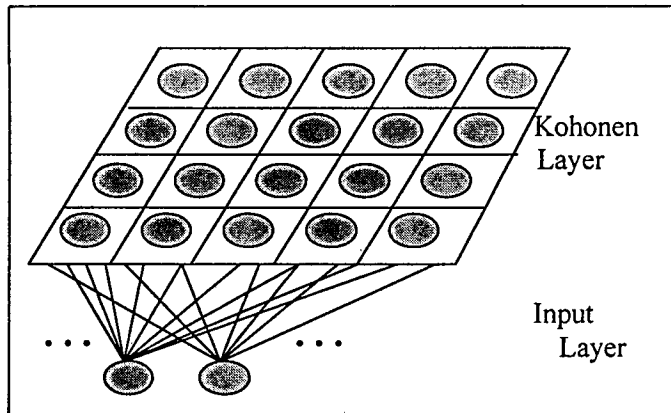


Figure 3.12: A Kohonen Network

Kohonen network is constructed of a fully interconnected array of neurons i.e. the output of each neuron is an input to all neurons, including itself and each neuron receives the input pattern.

Applying an input pattern to the network, consisting of a set of continuous-data and the output of each neuron is computed. The neurons are then allowed to interact with each other and the neuron that has the largest output is found. This neuron and its neighbor neurons (adaptive zone see figure below), within a certain distance, are allowed to adjust their weights to become more responsive to the particular input. A Kohonen network is shown in the figure below.

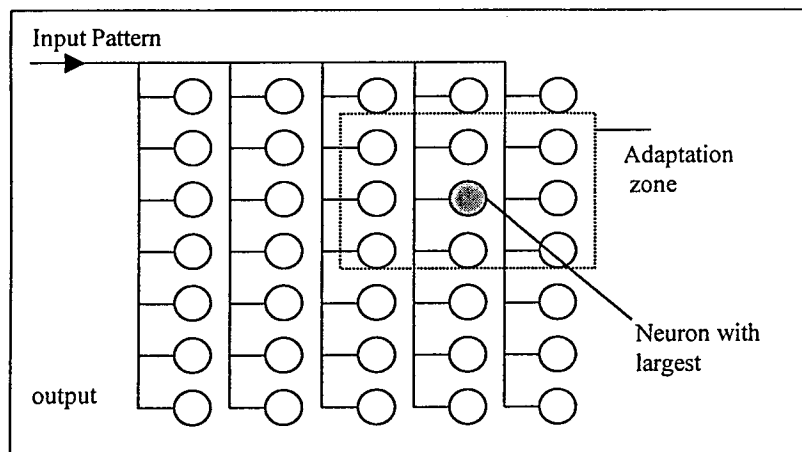


Figure 3.13: A 7x5 Kohonen Array Showing the Size of the Adaptation Zone

The size of the Adaptable set starts with a big enough size and decreases as the number of times the network has gone through all the data patterns increases. Within the Adaptable set, the amount by which the weights of each node changed depends on how far it is from the winning node. The closer the neurons are to the winning node, the bigger the amount by which their weights change.

3.5. Properties and Capabilities

We conclude this chapter by listing the properties and capabilities of a neural network in general.

3.5.1. Nonlinearity

A neural network is made up of an interconnection of neurons, that is it is itself nonlinear. Moreover, this nonlinearity property is special kind in the sense that it is distributed through the network. Nonlinearity is a highly important property.

3.5.2. Supervised Learning (Input-Output Mapping)

A popular kind of learning is called supervised learning which involves the modification of the weights of a neural network by applying a set of training samples or task examples. Each example consists of a unique input and its desired output. The network is given the examples picked at random from the set, and the weights of the network are modified so as to minimize the difference between the desired output and the actual output of the network produced by the input. The training of the network is repeated for many examples in the set until the network reaches a steady state. Thus, the network learns from the examples by constructing an input-output mapping for the problem in hand.

3.5.3. Adaptivity

Neural networks have a built-in capability to adapt their weights to changes in the environment. In particular, a neural network trained to operate in a specific environment can be easily retrained to deal with minor changes in the operating environmental conditions. As a general rule, it may be said that the more adaptive we make a system in a properly designed fashion, assuming the adaptive system is stable, the more robust its performance will be. But adaptivity does not always lead to robustness; indeed, it may do the very opposite. For example, an adaptive system with short time constants may change rapidly and therefore tend to respond to spurious disturbances, causing degradation in system performance. The benefit of adaptive system is that the time constants of the system should be long enough for the system to ignore disturbances and yet short enough to respond to meaningful changes in the environment.

3.5.4. Fault Tolerance

A neural network has the potential to be inherently fault tolerant in the sense that its performance is degraded gracefully under adverse operation conditions. For example, if a neuron or its connecting links are damaged, the damage has to be extensive before the overall response of the network is degraded seriously. Thus a neural network exhibits a graceful degradation in performance rather than failure.

3.5.5. VLSI Implementability

The parallel nature of a neural network makes it potentially fast for the computation of certain tasks. This feature also makes a neural network suited for implementation using very-large-scale-integrated technology. In other words, it captures truly complex behavior that makes it possible to use a neural network as a tool for real-time applications involving pattern recognition, signal processing and control.

CHAPTER IV

Hopfield Networks for Solving Optimization Problems

The research in neural networks as a naturally parallel approach to artificial intelligence, showed that they could be used to find near-optimal solutions to optimization problems. An optimization problem can be defined as a computational problem in which the object is to find the best of all possible solutions. More formally the problem asks to find a solution in the feasible region which has the minimum or maximum value of the objective function.

Combinatorial optimization problems are divided into classes according to the computational time needed to solve them. The most important and difficult class of combinatorial problems is NP-complete problems. For NP-complete problems, no algorithm is known which provides an exact solution to the problem in a computational time which is a polynomial in the size of the problem. Recently, a new approach has arisen to solve such problems efficiently and almost in real-time by applying neural networks. Although the application of neural networks to some NP-complete problems is still controversial, we think that an algorithm solves a specified combinatorial optimization problem successfully if it is able to find a good near-optimal solution - not necessarily optimal – in a reasonable amount of time.

4.1. Hopfield Network

Hopfield network is a simple artificial network and considered one of the first types of neural networks. The Hopfield neural network has no special input or output neurons, but all are both input and output, and all are connected to all others in both

directions (with equal weights in the two directions). Input is applied simultaneously to all neurons which then output to each other and the process continues until a stable state is reached, which represents the network output. Hopfield neural networks are recurrent because of the dynamical feedback mechanism of changing the input as a function of the output of the network. A network is stable if after successive iterations the output change and become smaller and smaller until it reaches equilibrium state where all the outputs will become constant in the successive iterations.

The Hopfield network works with input patterns which are vectors of real numbers or binary vectors. In all cases, the units of the Hopfield network have states, which described by number belonging to the set pattern values. In binary case the state are either 1 or -1 . In the continuous case, the states are numbers of $[a,b]$. Patterns are entered to the network by setting the states of each unit with the its appropriate values.

The Hopfield network is not trained in the same way as backpropagation. Instead, a set of exemplar patterns is given to the network to initialize the weights of the network. Once this is done, any pattern can be presented to the network, which will respond by displaying the exemplar pattern that is similar to the input pattern. The output pattern can be read from the network by reading the states of the units in the order determined by the mapping of the components of the input vector to the units. A Hopfield network is shown in the figure below.

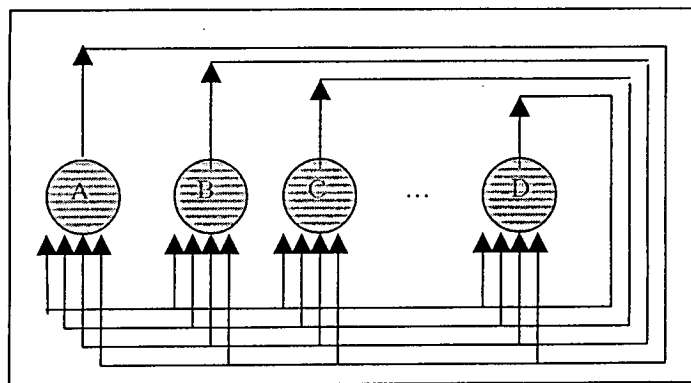


Figure 4.1: Another drawing for the Hopfield Network

Hopfield neural networks have been applied to various optimization problems, many of which are NP-complete. The realization of the traveling salesman problem in Hopfield neural networks consists of associating a neuron with the distance between each pair of cities. Consider a matrix array of $N \times N$ neurons to represent a tour taken to visit N cities. The N^2 neurons are grouped into N groups of N neurons and there are $N!$ possible solutions. Each group of N neurons is used to represent the position in the tour of a particular city. Each row corresponds to a city and each column corresponds to a position in the tour. The state of neuron B_{xy} , means the x^{th} city visited in the y^{th} order. If it is set to one, the city is really visited in that order, if it is set to zero, then it is not visited in that order. A network can be proven to be stable if the synaptic matrix is symmetric with zeros on its main diagonal, that means, $d_{ij} = d_{ji}$, and $d_{ii} = 0$.

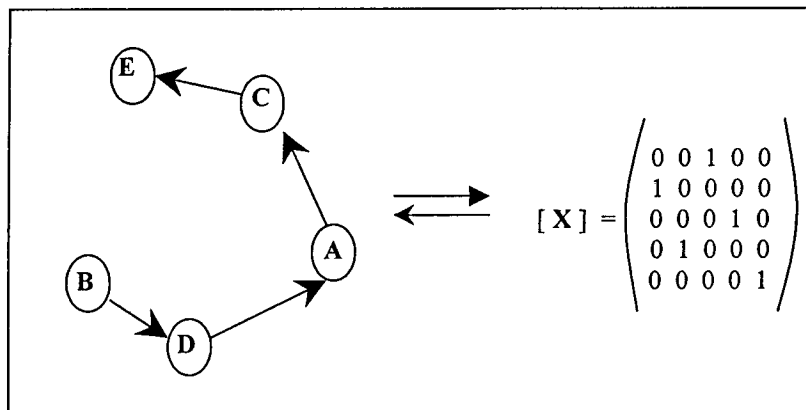


Figure 4.2: Matrix Representation of the TSP Problem

The figure above is an example where the city B is visited first, followed by D, A, C and E. The first group of the five neurons corresponds to the city B $[1 \ 0 \ 0 \ 0 \ 0]$ means B is the first city to be visited. The second group corresponds to the city D $[0 \ 1 \ 0 \ 0 \ 0]$ means D is the second city to be visited and so on. The matrix shown in figure 14 should represent the neuron outputs at the end of the optimization process. If distance between cities B and D is $d_{B,D}$, then the total path length of the tour would be $d_{B,D} + d_{D,A} + d_{A,C} + d_{C,E} + d_{E,B}$. If this tour is optimal, this means that the total path length of the tour is minimal.

4.2. Energy Function

To enable the Hopfield network to compute a solution to a problem, an energy function must be defined. The idea is to encode each hypothesis as a neuron and to encode constraints between hypothesis as weights. Positive weights represent supporting relationships, whereas negative weights represent incompatible relationships. As the Hopfield network reach a stable state, this state reflects which hypothesis is true or false under the constraints. Many combinatorial optimization problems can be mapped onto neural networks by constructing an energy function and then transforming the problem of their minimization into associated systems of differential equations. In general, such energy function can be written in the form of two terms:

Energy Function = Global Constraints + Cost.

Our goal is to minimize the Cost function and simultaneously maximize the number of Global Constraints that are satisfied. The energy function constructed from the traveling salesman problem constraints is:

$$E_s (\text{TSP}) = E_1 + E_2 + E_3 + \text{Cost}$$

The first three terms of the energy function represent the constraints. The first term E_1 means that each row contains no more than a single 1. The second term E_2 means that each column contains no more than a single 1. The third term E_3 means that there should be exactly n 1s in the array. The last term is the cost function which measures the total distance of the tour which is to be minimized. Mathematically the computation energy function can often be expressed as [9]:

$$E_s = \frac{A}{2} \sum_x \sum_i \sum_{j \neq i} v_{xi} v_{xj} + \frac{B}{2} \sum_i \sum_x \sum_{y \neq x} v_{xi} v_{yi} + \frac{C}{2} \left(\sum_x \sum_i v_{xi} - n \right)^2$$

$$\text{Cost} = \sum_x \sum_{y \neq x} \sum_i d_{xy} v_{xi} (v_{y,i+1} + v_{y,i-1})$$

Where d_{xy} is the distance between cities x and y , and $A, B, C,$ and D are constant values. It is clear that E_s is zero if there is no more than one entry per row or column of v_{xi} . This is to say that no city can be visited more than once, no two cities can be visited at the same time, and the tour is a valid tour. v_{xi} is the activation level of the unit denoting that city x is the i^{th} city in the tour. In solving an optimization problem, this energy function is compared with another function built from problem constraints in order to determine the network weights.

The difficulty in using a neural network to optimize an energy function is that the iteration procedure may often be trapped into local minimum, which usually corresponds to an invalid solution. Moreover, the values assigned to the parameters of an energy function affect the convergence rate of iterations.

4.3. Review of the Main Algorithm

Step1: Write an energy function based on the problem constraints

Step2: Compare the above energy function with the following energy Function of the Hopfield net to determine the weights

Step3: Calculate the Activation

a. At time = 0

$$O_j(0) = \text{a randomized small value}$$

Where $O_j(t)$ is the activation level of unit j at time t .

b. At time = t ($t > 0$),

$$O_j(t+1) = f\left(\sum_i w_{ji} o_i(t) + I_j\right)$$

Where the function F is a hard-limiting function:

$$f(x) = 1 \quad (x > j)$$

$$f(x) = -1 \quad (x < j)$$

$$f(x) = 0 \quad (x = j)$$

- c. Repeat step 3.2 until equilibrium. The activation levels of nodes remain unchanged with further iteration. Then, the pattern of activation upon equilibrium represents the optimized solution.

4.4. Results

There are three major difficulties of doing optimization problems by Hopfield neural network. These difficulties are 1) The tendency of the energy function to stuck into local minima which is an advantage for association but disadvantageous for optimization.

Deeper minima correspond to shorter tours and global minimum to the shortest tour 2) No systematic approach to formulate energy function to solve optimization problems. 3) Not certain about whether a global minimum (optimal solution) can really be achieved because it has a limited network capacity. For a network of N binary nodes, the capacity limit is of the order N rather than 2^N . 4) Difficulties in finding suitable values of the parameters A , B , C and D . Setting A , B , C and D to too small values has its effects on the cost function and usually gives invalid tours. On the other hand, making A , B , C and D to too large values causes the parameters to become so great that the network will converge to any feasible solution regardless of its total length.

As a consequence, it will not be easy for the network to find a valid solution since it frequently faces such problems. The convergence of the network to an acceptable tour is difficult to reach and that this possibility decreases rapidly with the increase of number of cities (for a 10 cities problem, after 1000 iterations of the network an acceptable tour was obtained). As has been demonstrated by several researches the performance of the Hopfield neural network for TSP is rather slow and very controversial [10], [15]. They tried to improve the algorithm in three ways: by changing the values of the energy parameters, and changing the initial conditions. Even with all this changes the final result

was not better than the original one. A new hybrid neural network model discussed in the next chapter shed a new light on the TSP problem.

CHAPTER V

A Hybrid Neural Network Model for Solving Optimization Problems

Neural networks have been used to solve a wide variety of optimization problems. Hopfield and Tank's work on solving optimization problems by neural network appeared in [9]. Moreover, their algorithm produced a long list of NP-complete problem that takes deterministic time to be solved. Therefore, our aim is to find out ways to solve these complex problems to produce a feasible solution in a lesser time. Sun and Fu introduced a recent model of neural networks, named the Hybrid Neural Network Model (HNN) for solving optimization problem [15]. The HNN model contains two sub-networks: the constraints-network to satisfy the constraints, and the goal-network to optimize the goal function. These two sub-networks are put to work together to guarantee fast convergence of the constraints sub-network; where the main algorithm called the Hybrid Network Updating Algorithm (HNUA) is used to drive the HNN model. The best thing about the HNUA is that it reaches a feasible solution very rapidly, but it guarantees very little in terms of the quality of the generated solution. So we propose a modifying version of the HNUA (MHNUA) that is able to produce a more feasible solution (optimal or near optimal) in a reasonable amount of time.

5.1. Optimization Problem Representation and Transformation Method

In [15], Sun and Fu produce a systematic transformation method to construct an energy function to represent the traveling salesman problem. The systematic transformation method contains three steps: define the problem as logical expressions; mapping these logical expressions into a set of algebraic equations; and formulating an energy function from these algebraic equations.

5.1.1. From Problem Definition to Logical Expressions

Since most optimization problems contain two parts: constraint and cost criteria. To achieve an optimal solution, constraints must be satisfied and the cost criteria must be minimized. For example, in the traveling salesman problem, the logical symbol C_{xj} - or its complement - is used to represent whether or not a city x is being visited at time j during a tour. The algebraic symbol d_{xy} is used to represent the distance between cities x and y . The constraints and the cost criteria of a TSP are represented as follows:

Constraint 1: Each city must be visited only once.

This constraint can be expressed by the following set of logical equations:

A. At least one C_{ik} is true,

$$C_{i1} \vee C_{i2} \vee C_{i3} \vee \dots \vee C_{iN} = \text{True},$$

B. No more than one C_{ik} is true,

$$\begin{aligned} C_{i1} \wedge C_{i2} &= \text{False}, C_{i1} \wedge C_{i3} = \text{False}, \dots, C_{i1} \wedge C_{iN} = \text{False}, \\ C_{i2} \wedge C_{i3} &= \text{False}, \dots, C_{i2} \wedge C_{iN} = \text{False}, \dots, C_{iN-1} \wedge C_{iN} = \text{False}. \end{aligned}$$

Constraint 2: A salesman arrives in at least one and no more than one city at any time j during the tour. Similarly this constraint can be expressed by the following set of logical equations:

A. At least one C_{kj} is true,

$$C_{1j} \vee C_{2j} \vee C_{3j} \vee \dots \vee C_{Nj} = \text{True},$$

B. No more than one C_{kj} is true,

$$\begin{aligned} C_{1j} \wedge C_{2j} &= \text{False}, C_{1j} \wedge C_{3j} = \text{False}, \dots, C_{1j} \wedge C_{Nj} = \text{False}, \\ C_{2j} \wedge C_{3j} &= \text{False}, \dots, C_{2j} \wedge C_{Nj} = \text{False}, C_{N-1j} \wedge C_{Nj} = \text{False}. \end{aligned}$$

Cost criteria: To find the shortest tour length. The minimization of cost criteria can be represented by the following logical expression:

$$\text{Min} \sum_{x,y} \sum_i [d_{xy} (C_{xi} \wedge (C_{y,i+1} \vee C_{y,i-1}))]$$

5.1.2. From Logical Expressions to Algebraic Equations

Mapping logical expressions into algebraic equations without changing their semantic meaning is the second step of the transformation method. The mapping method works as follows: Replace each instance of

- a. True by 1
- b. False by 0
- c. Logical variable C_{ij} by c_{ij}
- d. A NOT operator by subtraction
- e. An AND operator by multiplication.

When OR operator is needed, it can be derived by combining the NOT and AND operators:

$$[(x \vee y) = 1 - ((1 - x) \wedge (1 - y))]$$

Based on this mapping method, the logical expressions of a TSP can be transformed into the following algebraic equation:

Constraint 1: Each city must be visited only once.

$$(1 - c_{i1})(1 - c_{i2})(1 - c_{i3}) \dots (1 - c_{iN}) = 0,$$

$$c_{i1}c_{i2} = 0, c_{i1}c_{i3} = 0, \dots, c_{i1}c_{iN} = 0,$$

$$c_{i2}c_{i3} = 0, \dots, c_{i2}c_{iN} = 0, \dots, c_{iN-1}c_{iN} = 0.$$

Constraint 2: A salesman arrives in at least one and no more than one city at any time j during the tour.

$$\begin{aligned}(1-c_{1j})(1-c_{2j})(1-c_{3j})\dots(1-c_{Nj}) &= 0, \\ c_{1j}c_{2j} &= 0, c_{1j}c_{3j} = 0, \dots, c_{1j}c_{Nj} = 0, \\ c_{2j}c_{3j} &= 0, \dots, c_{2j}c_{Nj} = 0, \dots, c_{N-1j}c_{Nj} = 0.\end{aligned}$$

Cost criteria: To find the shortest tour length.

$$Cost = \sum_{x,y} \sum_i^N d_{xy} c_{xi} (c_{yi+1} + c_{yi-1} - c_{yi+1}c_{yi-1})$$

Under constraints 1 and 2, the space of feasible solutions can be represented by a binary value matrix $[C]_{N \times N}$. In matrix $[C]$, there can be only one 1 (true) in each row and each column; all other variables must be 0 (false), so that constraints 1 and 2 are satisfied and $\partial E_s / \partial c_{xi} = 0$; otherwise it is different than zero [15].

5.1.3. Formulating an Energy Function from Algebraic Equation

A squared error function E_s can be formulated from the constraints of the TSP:

$$E_s = \sum_{i=1}^{N'} (t_i - a_i)^2$$

where t_i represents the target value, a_i represents the iteration value of each algebraic expression of the constraints, and N' is the total number of algebraic equations of constraints. An energy function E can be obtained by combining the cost criteria and the error function as follows:

$$E = E_s + Cost$$

$$E_s = \frac{A}{2} \sum_x \sum_i \sum_{j \neq i} c_{xi} c_{xj} + \frac{B}{2} \sum_i \sum_x \sum_{y \neq x} c_{xi} c_{yi} + \frac{C}{2} \sum_x \sum_i (c_{xi} - n)^2$$

$$Cost = \sum_x \sum_{y \neq x} \sum_i d_{xy} c_{xi} (c_{y,i+1} + c_{y,i-1})$$

where A, B and C are the parameters. When the constraints are satisfied (i.e., $E_s = 0$) and cost criteria is minimized, the shortest tour length of the TSP is obtained. Choosing improper values for the parameters in the energy function slows down the convergence speed and results in invalid solutions.

5.2. The Hybrid Model

Many algorithms have been proposed for solving optimization problems. Most algorithms often give invalid solutions or need a large amount of computation time to reach a feasible solution. Sun and Fu designed a hybrid neural network (HNN) model that finds an optimal (or near optimal) solution within a short computation time [15]. The architecture of the HNN model has two sub-networks: the constraint-network and the goal-network as shown in the figure below.

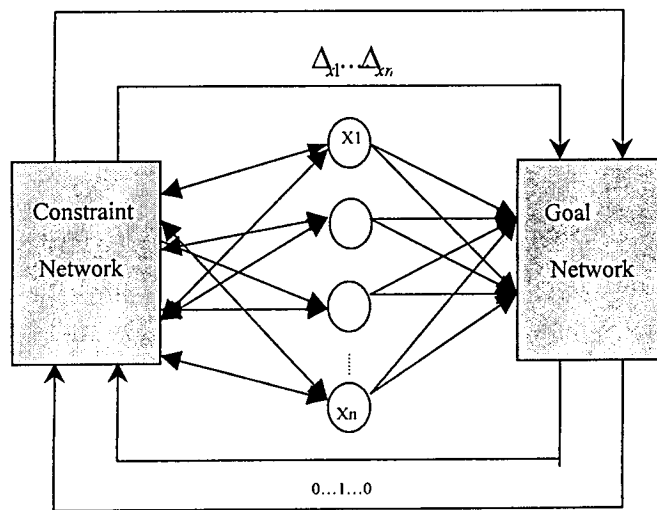


Figure 5.1: The Structure of Hybrid Neural Network

The constraint network models the constraints of the problem. For each neuron it computes the gradient value (i.e., $\Delta x_1, \Delta x_2, \dots, \Delta x_n$). Therefore, we can say that the constraint network contains the updating values. By using these gradient value Δx_i 's, the goal network computes the direction of convergence during each iteration for minimizing

minimizing the cost criteria. The x_i 's are the problem variables and assume values 0 or 1 only; that is to say false or true, respectively. Each neuron x_i in the constraint network computes the updating value Δx_i . To minimize the squared error function E_s , Newton method is used to update the state of the neural at each iteration. The updating value Δx_i is defined as follows:

$$\Delta x_i = \frac{\frac{\partial E_s}{\partial x_i}}{\frac{\partial^2 E_s}{\partial x_i^2}}$$

All updating values (i.e., $\Delta x_1, \Delta x_2, \dots, \Delta x_n$) are passed to the goal sub-network where the MAX-MIN portion determines which neuron (e.g., x_i) is to be updated – corresponds to maximum Δx_i . The Goal sub-network computes $\partial \text{cost} / \partial x_i$. Both the Δx_i and $\partial \text{cost} / \partial x_i$ are sent to the MAX-MIN network to determine which one among the x_i (with maximum Δx_i) is to be updated – corresponds to optimum $\partial \text{cost} / \partial x_i$. The output of the MAX-MIN network enables the selector module to output the corresponding updating value Δx_i which will be passed back to the input neurons to update their states. A more detailed figure is shown below.

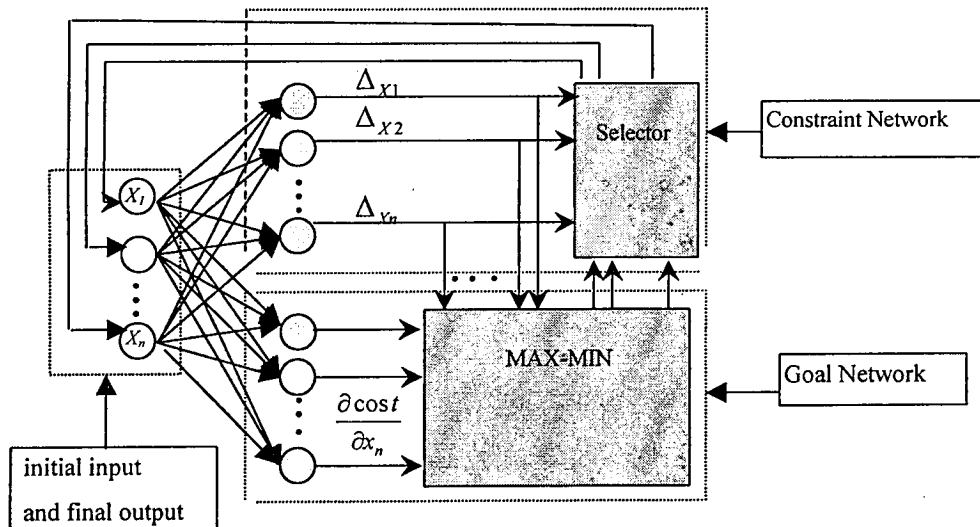


Figure 5.2: The Architecture of the Hybrid Neural Network Model

That is, one of the x_i , say x_1 , is chosen at random and passed to the selector. The selector passes the corresponding Δx_1 to the network input to update the value of x_1 . During iteration, x_1 assumes the value $x_1 + \Delta x_1$. This process is repeatedly executed until the gradient of E_s is equal to zero; that is the constraints are satisfied (i.e., $\Delta x = 0$).

5.3. The Hybrid Network Updating Algorithm

Sun and Fu formulate an energy function similar to Hopfield and Tank for the TSP except that they forced the entries on each row and each column of matrix c_{xi} to be at most equals to 1. Sun and Fu defined the energy function as follows:

$$\begin{aligned}
 E &= E_s + Cost \\
 E_s &= \sum_x \sum_i \sum_{j \neq i} (c_{xi} c_{xj})^2 + \sum_i \sum_x \sum_{y \neq x} (c_{xi} c_{yi})^2 + \\
 &\quad \sum_x \sum_i \sum_{j \neq i} ((1 - c_{xi})(1 - c_{xj}))^2 + \sum_i \sum_x \sum_{y \neq x} ((1 - c_{xi})(1 - c_{yi}))^2. \\
 Cost &= \sum_x \sum_{y \neq x} \sum_i d_{xy} c_{xi} (c_{yi+1} + c_{yi-1} - c_{yi+1} c_{yi-1}).
 \end{aligned}$$

Notice the absence of any constraints (i.e., A, B, C, ...) in the expression E_s . It is shown in [15] that the function E_s monotonically converges to a stable state; because E_s has non-zero second partial derivative over a set of order two variables, that is $\partial E_s / \partial c_{xi} \neq 0$. When $\partial E_s / \partial c_{xi} = 0$ then function E_s reaches a stable state and all the constraints are satisfied.

Procedure for Hybrid Network Updating Algorithm (HNUA)

[1] Use the Newton method to minimize the square error function E_s by computing the updating value Δx_i as follows:

$$\Delta x_i = \frac{\frac{\partial E_s}{\partial x_i}}{\frac{\partial^2 E_s}{\partial x_i^2}}$$

[2] For each variable x_i , compute the partial derivation of the cost function as follows:

$$PartialDerivative(x_i) = \frac{\partial Cost}{\partial x_i}$$

[3] Determine the maximum values Δc^* of $|\Delta c_{xi}|$'s:

$$\Delta c^* = \max(\{|\Delta c_{xi}|, 1 \leq x, i \leq n\}),$$

where n is the number of cities, then form a set Γ of variable c_{yj} that corresponds to the maximum updating value Δc^* , i.e.,

$$\Gamma = \{c_{yj} \text{ such that } |\Delta c_{yj}| = \Delta c^*, \forall y, j\}.$$

[4] Among the c_{yj} 's in the set Γ , select a variable c_{wk} which corresponds to the minimum value of the partial derivation of the Cost function .

$$c_{wk} = \min \left[\frac{\partial Cost}{\partial c_{yj}}, \forall c_{yj} \in \Gamma \right]$$

Update c_{wk} by adding the updating value obtained in step [1].

$$c_{wk} = c_{wk} + \Delta c_{wk}$$

[5] Test the gradient of function E_s . If the gradient of E_s is equal to zero, then the E_s function converges to a stable state and the algorithm terminate.

$$\left[\frac{\partial E_s}{\partial c_{xi}} = 0 (1 \leq x, i \leq n) \right]$$

Otherwise, return to step [1] for next iteration.

5.4. Modification of Hybrid Neural Network Updating Algorithm

We observe from the results of HNUA algorithm that the constraints are satisfied very rapidly (the number of iterations is equal to $n-1$) while the generated solutions guarantee very little in terms of their quality. In step [15], of the HNUA algorithm, we

notice that it generates a set of choices for C_{wk} and that the selection is done randomly on that set. So to accept one choice randomly is a costly operation where its results will have a random effect on the length of the generated tour. Our proposed argument for modifying HNUA (MHNUA) is the following: it is worst it to trade some additional running time in return for a better solution We rewrite step [4] in the HNUA algorithm so that the goal function (tour length) is better served. The idea in the MHNUA is to try to select the best from the set of c_{wk} at each iteration rather than a mere random selection.

The cost function of the c_{wk} set is given by:

$$Costc_{wk} = \sum_x \max \left(\frac{\partial Cost}{\partial c_{xi}}, (1 \leq i \leq n) \right)$$

where the sum runs over all the cities x irrespective of their order. The term $\partial cost/\partial xi$ measures the length of that part of the tour associated with visiting city x at time i . For each iteration, the MHNUA computes the length of the entire tour - $Costc_{wk}$ for all the elements in set c_{wk} - and selects the one that gives the smallest additional cost. Obviously this MHNUA algorithm takes longer time to run than HNUA but it provides better solutions. The judgment as to which algorithm to use lies in what is more important for a particular application.

In the following, we rewrite step [4] as it should appear in the MHNUA:

Step 4a: Determine the minimum values $\Delta Cost^*$ of $|\partial Cost/\partial c_{yj}|$'s

$$\Delta Cost^* = \min \left(\frac{\partial Cost}{\partial c_{yj}}, \forall c_{yj} \in \Gamma \right)$$

then form a set Γ^* of variables $\partial Cost/\partial c_{wk}$

$$T^* = \left\{ c_{wk} \in \Gamma \mid \frac{\partial Cost}{\partial c_{wk}} = \Delta Cost^* \right\}$$

Step 4b: For each c_{wk} in T^* do the followings:

Updated c_{wk} by adding the updated value obtained in step [1]:

$$c_{wk} = c_{wk} + \Delta c_{wk}$$

Compute $Costc_{wk}$

$$Costc_{wk} = \sum_x \max \left(\frac{\partial Cost}{\partial c_{xi}}, (1 \leq x, i \leq n) \right)$$

Step 4c: Choose and update the c_{wk} which corresponds to minimum $Costc_{wk}$.

CHAPTER VI

The Hybrid Neural Network Model Applied to the Traveling Salesman Problem

This chapter defines the traveling salesman problem (TSP). TSP is used as a good example to test and compare the solutions generated by HNUA, MHNUA and DFSA applications. A MHNUA application is developed to implement the MHNUA algorithm and another application called DSFA application is designed and developed to implement the depth first search algorithm. After we developed these applications, we observed that MHNUA produced efficient (optimal or near-optimal) solutions in a reasonable amount of time. All developed applications are integrated with geographic information system (GIS) to provide a single interface for calculating the length between cities and visualizing the resultant tour graphically.

6.1. The Traveling Salesman Problem

The traveling salesman problem (TSP) occupies a central place among the NP-complete combinatorial optimization problems and of its parallel implementation. Therefore, TSP is constitutes a good example to simulate the MHNUA model. TSP describes the case of a salesman who wishes to visit N cities, visiting each city exactly once and finishing at the city he starts from. There is a cost to travel from city I to city J . Besides, the salesman wishes to make the tour whose total cost is minimum, where the total cost is the sum of the individual costs along the edges of the tour. For an undirected graph with N cities, there are $(N-1)!/2$ possible tours and for directed graphs there are $(N-1)!$ possible tours. This $N!$ function grows very rapidly as the value of N increases. An example which shows all possible combination tours for $N = 4$ are shown in the figure below.

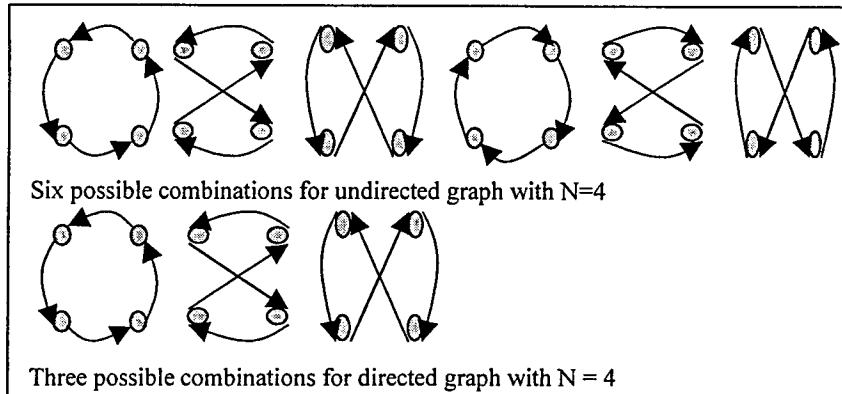


Figure 6.1: Possible Combination for Undirected and Directed graph for N=4

6.2. Neural Network Representation for TSP

Consider a matrix array of $N \times N$ neurons to represent a tour taken to visit N cities. The N^2 neurons are grouped into N groups of N neurons. Each group of N neurons is used to represent the position in the tour of a particular city. Each row corresponds to a city and each column corresponds to a position in the tour. For example, in a four city problem if the cities are labels A, B, C, and D, then a possible tour starting at C would be $C \rightarrow D \rightarrow B \rightarrow A \rightarrow C$. If the distance between cities x any y is d_{xy} then the total path length of the tour would be $d_t = d_{CD} + d_{DB} + d_{BA} + d_{AC}$. For this tour to be optimal, the total path length d_t of the tour should be minimal. The neuron output at the end of the optimization process would give a matrix as shown in figure below.

$$d = \begin{pmatrix} & 1 & 2 & 3 & 4 \\ A & 0 & 0 & 0 & 1 \\ B & 0 & 0 & 1 & 0 \\ C & 1 & 0 & 0 & 0 \\ D & 0 & 1 & 0 & 0 \end{pmatrix}$$

Tour derived from matrix d is: $C \rightarrow D \rightarrow B \rightarrow A \rightarrow C$

Figure 6.2: A Matrix Array of 4 x 4 Neurons to Represent a Tour

The first group of the four neurons corresponds to A [0 0 0 1]. This means A is the fourth city to be visited. The second group corresponds to city B [0 0 1 0] which means B is the third city to be visited and so on.

6.3. Simulation Results and Analysis

6.3.1. HNUA Model Compared to MHNUA Model

A HNUA application is created to implement the hybrid network updating algorithm. In addition, a MHNUA application is also developed to implement the proposed modification on HNUA discussed previously in chapter V. After we developed both applications and run them for different number of cities ranging between 4 and 25, we observed that MHNUA produced better solutions (optimal or near-optimal) than HNUA. While HNUA results were not as feasible in terms of quality, they always required lesser time. As described in the previous chapter, HNUA uses a basic greedy-like technique in its drive to maximize the goal function. MHNUA takes that greedy technique a step further so that the goal function is better served.

We recall that MHNUA computes the cost of all candidates Γ^* found at each iteration and selects the one with smallest cost instead of a mere random selection used by HNUA. The steps used in computing both Δc_{xi} and $\partial \text{Cost} / \partial c_{xi}$ and the set of maximum Δc_{xi} and minimum $\partial \text{Cost} / \partial c_{xi}$ are shown in the figure below.

$$d = \begin{pmatrix} 0 & 3 & 4 & 2 & 7 \\ 3 & 0 & 4 & 6 & 3 \\ 4 & 4 & 0 & 5 & 8 \\ 2 & 6 & 5 & 0 & 6 \\ 7 & 3 & 8 & 6 & 0 \end{pmatrix} \quad \Delta c_{xi} = \begin{pmatrix} 0 & .5 & .5 & .5 & .5 \\ .5 & 1 & 1 & 1 & 1 \\ .5 & 1 & 1 & 1 & 1 \\ .5 & 1 & 1 & 1 & 1 \\ .5 & 1 & 1 & 1 & 1 \end{pmatrix} \quad \frac{\partial Cost}{\partial c_{xi}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 3 \\ 0 & 4 & 0 & 0 & 4 \\ 0 & 2 & 0 & 0 & 2 \\ 0 & 7 & 0 & 0 & 7 \end{pmatrix}$$

$$\Gamma = \max \{ \Delta c_{xi} \} = \{ c_{22}, c_{23}, c_{24}, c_{25}, c_{32}, c_{33}, c_{34}, c_{35}, c_{42}, c_{43}, c_{44}, c_{45}, c_{52}, c_{53}, c_{54}, c_{55} \}$$

$$\Gamma^* = \min \left\{ \frac{\partial Cost}{\partial c_{xi}} \right\} = \{ c_{23}, c_{24}, c_{33}, c_{34}, c_{43}, c_{44}, c_{53}, c_{54} \}$$

Figure 6.3: Show the Two Sets T and T* derived from Δc_{xi} and $\partial Cost / \partial c_{xi}$

A full example for $N = 4$ is included in appendix A.1 and another for $N = 10$ is included in appendix A.2. Obviously, MHNUA which takes longer time to run produces a better or near optimal solution compared to HNUA. The length of the tours produced by both MHNUA and HNUA models for different number of cities are shown in the table below.

Number of Cities	HNUA Tour Length	MHNUA Tour Length
5	17	19
10	39	32
15	55	46
20	81	72
25	96	85

Table 6.1: Comparing both HNUA and MHNUA Models.

6.3.2. DFS Algorithm Compared to MHNUA Model

In addition to the applications developed for HNUA and MHNUA, a DFSA application coded in C++ to solve the TSP problem using the depth first search algorithm (DFS) is designed. DFS is a search technique where by the search begins at a particular

city and then proceeds as deep as possible in the state space tree for the problem until it reaches a dead end. At that time the algorithm backs up and explore another path. The DFS traversal of the state space tree for a TSP with four cities (A, B, C, and D) is shown in the figure below.

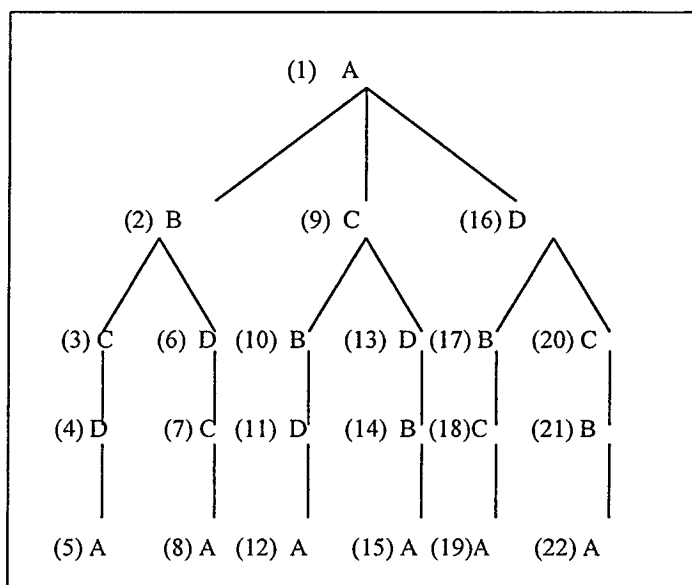


Figure 6.4: The State Space Tree for DFS

DFS is an important technique to solve TSP because it continues searching in the forward (deeper) direction as long as possible. Edges are explored out of the most recently discovered vertex v that still has unexplored edges leaving it. When all of v 's edges have been explored, the search backtracks to explore edges leaving the vertex from which v was discovered. This process continues until we have discovered all the vertices that are reachable from the original source vertex. If any vertices remain, one of them is selected as a new source and the search is repeated from that source. This entire process is repeated until all vertices are discovered. Applying such technique on TSP by generating a tree-like structure of all the different permutations we can cover. We will go deep in the tree calculating the distance we have covered at each step. If the distance is still less than the minimum distance already found, we will go deeper into the tree, else we will drop that tour.

We conclude that the DFSA application is an optimal method that works well with instances of the problem of small size. However, for large number of cities, it becomes non-practical (for e.g., it requires about 300 centuries with the assistance of a super computer to solve an instance of $N = 60$). An example for the output results produced by a DFSA application for $N = 10$ is shown in the figure below.



Figure 6.5: DFSA Output Result for $N = 10$

For the TSP, we tested 5, 10, 15, 20, 25 city problems with randomly chosen city coordinates. For each case (case1: $N=5$, case2: $N=10$ and so on) about 10 different city coordinate are tested on both DFSA and MHNUA applications. Comparing both output results, we were able to calculate the average percentage optimization and validation for the proposed model. The table below shows the average quality of the MHNUA generated solutions in comparison with the optimal solutions generated by the DFSA.

N (Number of cities)	Average of Percentage Optimization
5	100%
10	89 – 80%
15	88 – 79%
20	79 – 70%
25	70 – 67%

Table 6.2: Average Percentage of Optimization for MHNUA Model

Also, we observed that the DFSA running time for small number of cities between 5 and 15 is much faster than MHNUA. While for N between 15 and 20 the running times get closer to each other and finally for N between 20 and 25 MHNUA, it was able to produce a faster solution than DFSA. We conclude that the break even number of cities is around 15. Two examples for the TSP problem showing the path and length using both technique DFS and MHNUA are shown in the successive figures below.

$$d = \begin{pmatrix} & A & B & C & D & E \\ A & 0 & 3 & 4 & 2 & 7 \\ B & 3 & 0 & 4 & 6 & 3 \\ C & 4 & 4 & 0 & 5 & 8 \\ D & 2 & 6 & 5 & 0 & 6 \\ E & 7 & 3 & 8 & 6 & 0 \end{pmatrix}$$

DFS → A→C→B→E→D→A → 4+4+3+6+2 = 19

→ Optimal Path

MHNUA → A→C→B→E→D→A → 4+4+3+6+2 = 19

→ Optimal Path

Figure 6.6: Showing the Distance Matrix for N = 5 and its DFS and MHNUA Tour Results

$$d = \begin{pmatrix} & A & B & C & D & E & F & G & H & I & J \\ A & 0 & 4 & 9 & 2 & 6 & 5 & 9 & 9 & 5 & 2 \\ B & 4 & 0 & 6 & 6 & 4 & 10 & 8 & 2 & 7 & 1 \\ C & 9 & 6 & 0 & 5 & 7 & 7 & 10 & 10 & 10 & 1 \\ D & 2 & 6 & 5 & 0 & 8 & 10 & 4 & 4 & 10 & 2 \\ E & 6 & 4 & 7 & 8 & 0 & 2 & 6 & 1 & 8 & 8 \\ F & 5 & 10 & 7 & 10 & 2 & 0 & 4 & 2 & 9 & 3 \\ G & 9 & 8 & 10 & 4 & 6 & 4 & 0 & 2 & 1 & 3 \\ H & 9 & 2 & 10 & 4 & 1 & 2 & 2 & 0 & 8 & 8 \\ I & 5 & 7 & 10 & 10 & 8 & 9 & 1 & 8 & 0 & 1 \\ J & 2 & 1 & 1 & 2 & 8 & 3 & 3 & 8 & 1 & 0 \end{pmatrix}$$

DFS → A→B→H→E→F→G→I→J→C→D→A

→ 4+2+1+2+4+1+1+1+5+2 = 23

→ Optimal path

MHNUA → A→B→H→E→C→J→I→F→G→D→A

→ 4+2+1+7+1+1+3+4+4+2 = 29

→ Near-optimal path

Figure 6.7: Showing the Distance Matrix for N = 10 and its DFS and MHNUA Tour Results

The different characteristics of the DFSA model versus MHNUA model are summarized in the table shown below.

DFSA Model	MHNUA Model
1. Works for small instances i.e. for large N it no longer works	1. Works for large instances
2. Sequential computation (synchronous)	2. Parallel or collective computation (asynchronous)
3. Produce optimal solutions	3. Produce optimal or near-optimal solutions
4. Static connectivity	4. Dynamic connectivity
5. Fast. Measured in millions of seconds, when applicable	5. Slow. Measured in thousands of seconds

Table 6.3: The Different Characteristics of the DFSA Model versus MHNUA Model

CHAPTER VII

TSP INTERFACE Application Using GIS MapObject

7.1. GIS and MapObject Definition and Introduction

A geographic information system (GIS) is a computer-based tool for mapping and analyzing things that exist and events that happen on earth. GIS technology integrates common database operations such as query and statistical analysis with the unique visualization and geographic analysis benefits offered by maps. These abilities distinguish GIS from other information systems and make it valuable to a wide range of public and private enterprises for explaining events, predicting outcomes, and planning strategies

A GIS stores information as a collection of thematic layers that can be linked together by geography. This simple but extremely powerful and versatile concept has proven invaluable for solving many real-world problems from tracking delivery vehicles, to recording details of planning applications, to modeling global atmospheric circulation. A working GIS integrates five key components: hardware, software, data, people, and methods.

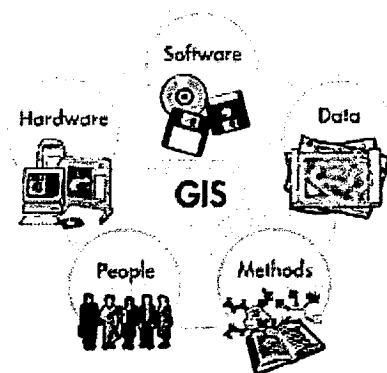


Figure 7.1: GIS Five Components

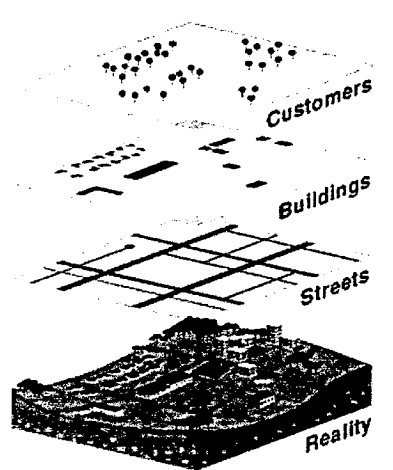


Figure 7.2: GIS Geographic Layer Representation

MapObjects is a set of mapping and GIS components for application developers. MapObjects consists of an ActiveX Control and a collection of programmable ActiveX Automation objects that let application developers add mapping and GIS capabilities to applications.

MapObjects is built upon Microsoft's ActiveX 2.0 standard. ActiveX is the most widely supported object-based software integration architecture available today. ActiveX components are used like building blocks to create and integrate Windows applications. An ActiveX Control is a software component that encapsulates a specific set of functionality. ActiveX automation objects have properties and methods that are used programmatically to control their appearance, behavior, and interactions. MapObjects is an ActiveX Control for mapping and GIS. The MapObjects map control plugs directly into the tool sets of many standard development environments. You can manipulate the map through property sheets found in development environments like Visual Basic environment.

7.2. Objectives of Using MapObject

The purpose of GIS-MapObject in this thesis is to perform visualization processes or tasks. For many types of geographic operation the end result is best visualized as a

map or graph. Maps are very efficient at storing and communicating geographic information. GIS gives the power to create maps, integrate information, visualize scenarios, solve complicated problems, present powerful ideas, and develop effective solutions.

Also, MapObjects allows you to create customized applications that use mapping and GIS components such as:

- Display of data as multiple layers in a map with the ability to pan and zoom throughout.
- Spatial selection through a wide variety of search and spatial operators.
- Feature attribute selection and query.

In this thesis we built a TSP interface application using MapObject in order to create mapping applications and adding mapping functionality to spatially select the cities, to calculate the length between cities and to visually observe the resultant tour path geographically.

7.3. TSP Interface Application

The purpose of building the TSP Interface application is to model our proposed MHNUA model on a graphical user interface, so that this model can be used on a real world streets for traveling salesman problem. TSP Interface application is built using Geographic Information System MapObject (GIS-MO). MapObject is a GIS map object used with Visual Basic object oriented environment. MapObject has a set of mapping components and functions that allows the user to add maps to his application and to simulate any graphical applications.

7.3.1. TSP User Interface Application

To get a better understanding of how the TSP Interface application looks like the discussion their follows refers to the figure below.

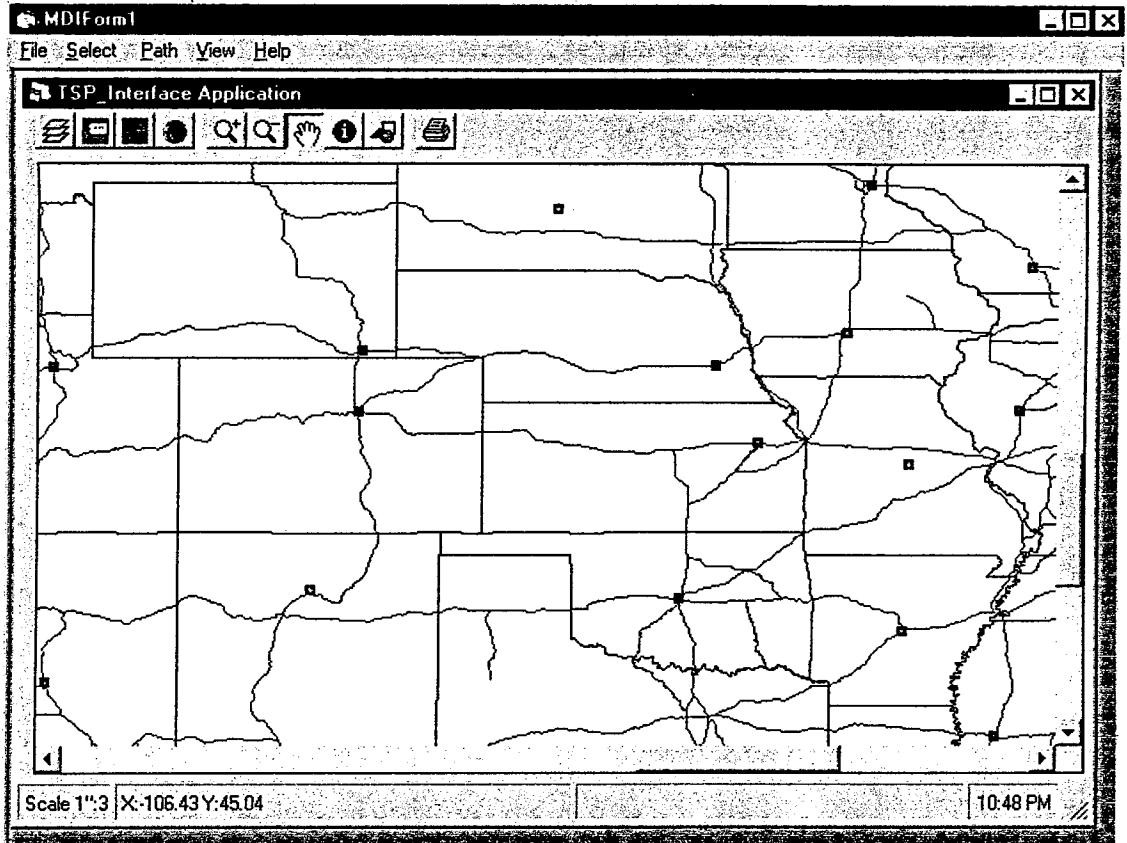


Figure 7.1: TSP User Interface Application Form

7.3.1.1. Spatial Selection Function

This function is used to select the cities we want to visit graphically using the mouse. After displaying the city layer inside the map zone the selection process begins. Spatial selection function is done by either selecting each city at a time by pointing to it using the mouse pointer or selecting group of cities at once by using the rectangle or polygon objects that selects all cities that intersect or totally within the specified object. The selected cities are highlighted with yellow color to be able to view them clearly on the map.

7.3.1.2 Two Cities Function

This function allows the user to find the streets length between two selected cities and to view it with a highlighted color on the map. The Two Cities Form is shown in appendix B.

7.3.1.3. Multi Cities Function

This function allows the user to find the street length between multi selected cities and display their length in a matrix-like object (Shown in appendix B). This matrix object is copied to an output file to be used as input parameter file for Mathematica application. After running the TSP Mathematica application that code the MHNUA algorithm, the resultant path is displayed in a different output file. Finally, the Visual Basic TSP interface application reads the results from the file and the MapObject displayed the result tour graphically on the map, that is it allows visualizing of the resultant tour between selected cities. The Multi Cities Form is shown in appendix B.

7.3.2. File structure of the TSP Application

The Visual Basic output results which represent the length between the cities are displayed in an output file called TSPFILE, the Mathematica engine application uses the TSPFILE as an input file to be used in order to find the tour path. The result of the Mathematica engine is displayed in an output file called TSPENGINE, the visual basic reads the result from the TSPENGINE file and display the output results with corresponds to the tour length is displayed in the map canvas to view it geographically.

7.3.3. Operating the Application

Launching and running the TSP Interface application and the steps needed to follow sequentially to execute the application and to show the final resultant tour in a clear-displayed figure is shown in the appendix B.

CHAPTER VIII

Conclusion and Future Research

In this thesis, the theory of NP-completeness and what makes some problems computationally intractable has been presented. Also, how combinatorial optimization problems suffer from exponential time complexity in the worst case has been showed. Combinatorial optimization problems are divided into classes, the most important class is the NP-complete problems. For NP-complete problems, no algorithm is known provides an exact solution to the problem in a computation time which is a polynomial in the size of the problem. Recently, a new approach has been arisen to solve such problems efficiently and almost in a real-time by applying neural networks. Our aim in this thesis is to solve these complex problems to produce a feasible solution in a lesser time. A hybrid neural network (HNN) and modification of the HNUA have been proposed. The HNN model contains two sub-networks: the constraints-network to satisfy the constraints, and the goal-network to optimize the goal function. These two sub-networks are put to work together to guarantee fast convergence of the constraints sub-network; where the main algorithm is called Hybrid Neural Updating Algorithm (HNUA) is used to drive the HNN model. The best thing about the HNUA is that it reaches a feasible solution very rapidly, but it guarantees very little in terms of the quality of the generated solution. So we work with the modification of the HNUA called Modified HNUA (MHNUA) that is able to produce a more feasible solution (optimal or near-optimal) in a reasonable amount of time. In summary, the proposed neural network technique for solving the TSP has the following advantages: it provides a mapping technique of the TSP problem into an energy function, it does not require selection of proper values for parameters in the energy function and it prevent the energy function from being trapped into an invalid solution. A MHNUA, HNUA and DFSA applications are developed to calculate the degree of optimization and validation of the proposed model. We observed that

MHNUA produced efficient solutions in a reasonable amount of time. Besides a TSP interface using Geographic Information System Map Object (GIS-MO) is developed to calculate the length between cities and to view the resultant tour.

Our future research will be to use the concept of coordinate descent method totally [17] to update the neural state. This will minimize the squared error function E and faster the convergence speed. The coordinate descent method works through minimizing the function with respect to one of the coordinate variables until the function reaches zero. Also, be updating with the MapObject Software functionality's for better use.

We believe that neural networks will become an important tool in the future for solving optimization problem. So it is important to follow up the new techniques and models for solving optimization problems and specially the traveling salesman problem using neural networks.

APPENDIX A

A.1. The Traveling Salesman Problem with $N = 4$

Consider the cities A, B, C, D and the following distance matrix:

$$d = \begin{pmatrix} & A & B & C & D \\ A & 0 & 2 & 10 & 9 \\ B & 2 & 0 & 5 & 8 \\ C & 10 & 5 & 0 & 1 \\ D & 9 & 8 & 1 & 0 \end{pmatrix}$$

The variables for this instance are:

$$v_{xi} = \begin{pmatrix} v_{11} & v_{12} & v_{13} & v_{14} \\ v_{21} & v_{22} & v_{23} & v_{24} \\ v_{31} & v_{32} & v_{33} & v_{34} \\ v_{41} & v_{42} & v_{43} & v_{44} \end{pmatrix}$$

Iteration #1

$$v_{xi} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \Delta v_{xi} = \begin{pmatrix} 0 & .5 & .5 & .5 \\ .5 & 1 & 1 & 1 \\ .5 & 1 & 1 & 1 \\ .5 & 1 & 1 & 1 \end{pmatrix} \quad \frac{\partial Cost}{\partial v_{xi}} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 2 \\ 0 & 10 & 0 & 10 \\ 0 & 9 & 0 & 9 \end{pmatrix}$$

$$\Gamma^* = \{v_{22}, v_{23}, v_{24}, v_{32}, v_{33}, v_{34}, v_{42}, v_{43}, v_{44}\}$$

$$\Gamma = \{v_{23}, v_{33}, v_{43}\}$$

Step1: Let $v_{wk} = v_{23}$ (try city B). We update v_{23} ; $v_{23} = v_{23} + \Delta v_{23} = 0 + 1 = 1$.

Next we see how does this choice influence the other cities (only those order has not been decided). We compute $(\partial Cost / \partial v_{xi})$ for $x = 3, 4$.

$$\frac{\partial Cost}{\partial v_{xi}} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 15 & 0 & 15 \\ 0 & 17 & 0 & 17 \end{pmatrix}$$

This says that now city C has an associate cost of length 15 if visited at step 2 or 4. City D has an associate cost of length 17 if visited at step 2 or 4. To verify the above data, examine the corresponding tours:

A->C->B->D->A (total length = 32)

A->D->B->C->A (total length = 32)

Finally, we compute $Costv_{xi}$, the project cost that this choice may contribute to the length of the entire tour:

$$Costv_{23} = \sum_{x=3,4} \max\left(\frac{\partial Cost}{\partial v_{xi}}, 1 \leq i \leq 4\right) = 32.$$

Step2: Let $v_{wk} = v_{33}$ (try city C). We update v_{33} ; $v_{33} = v_{33} + \Delta v_{33} = 0 + 1 = 1$, and compute $(\partial Cost / \partial v_{xi})$ for $x = 2, 4$.

$$\frac{\partial Cost}{\partial v_{xi}} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ 0 & 7 & 0 & 7 \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 10 & 0 & 10 \end{pmatrix}$$

To verify the above data, examine the corresponding tours:

A->B->C->D->A (total length = 17)

A->D->C->B->A (total length = 17)

$$Costv_{33} = 17.$$

Step3: Let $v_{wk} = v_{43}$ (try city D). We update v_{43} ; $v_{43} = v_{43} + \Delta v_{43} = 0 + 1 = 1$, and compute $(\partial \text{Cost} / \partial v_{xi})$ for $x = 2, 3$.

$$\frac{\partial \text{Cost}}{\partial v_{xi}} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ 0 & 10 & 0 & 10 \\ .0 & .11 & .0 & .11 \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

To verify the above data, examine the corresponding tours:

A -> B -> D -> C -> A (total length = 21)

A -> C -> D -> B -> A (total length = 21)

$\text{Cost}_{v_{43}} = 21$.

Since it is V_{33} which provide the minimum cost, we choose to visit city C at step 3 during the tour. The gradient of E_s

$$\nabla E_s = \begin{pmatrix} 0 & -2 & 0 & -2 \\ -2 & -4 & -2 & -4 \\ 0 & -2 & 0 & -2 \\ -2 & -4 & -2 & -4 \end{pmatrix}$$

Iteration #2

$$v_{xi} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \Delta v_{xi} = \begin{pmatrix} 0 & .5 & 0 & .5 \\ .5 & 1 & .5 & 1 \\ 0 & .5 & 0 & .5 \\ .5 & 1 & .5 & 1 \end{pmatrix} \quad \frac{\partial \text{Cost}}{\partial v_{xi}} = \begin{pmatrix} 0 & 10 & 0 & 10 \\ 0 & 7 & 0 & 7 \\ 0 & 10 & 0 & 10 \\ 0 & 10 & 0 & 10 \end{pmatrix}$$

$$\Gamma = \{v_{22}, v_{24}\}$$

$$\Gamma^* = \{v_{22}, v_{24}, v_{32}, v_{34}\}$$

This decides whether to visit city B during the second step or the fourth step.

Step1: Let $v_{wk} = v_{22}$ (try step 2). We update v_{22} ; $v_{22} = v_{22} + \Delta v_{22} = 0 + 1 = 1$, and compute $(\partial \text{Cost} / \partial v_{xi})$ for $x = 3$.

$$\frac{\partial Cost}{\partial v_{xi}} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ 5 & 10 & 5 & 10 \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

$$Cost_{v_{22}} = 5$$

Step2: Let $v_{wk} = v_{24}$ (try step 4). We update v_{24} ; $v_{24} = v_{24} + \Delta v_{24} = 0 + 1 = 1$, and compute $(\partial Cost / \partial v_{xi})$ for $x = 3$.

$$\frac{\partial Cost}{\partial v_{xi}} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ 5 & 10 & 5 & 10 \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

$$Cost_{v_{24}} = 5$$

Since both choices project similar cost, we choose either one (at random); say V_{22} .

The gradient of E_s

$$\nabla E_s = \begin{pmatrix} 0 & 0 & 0 & -2 \\ 0 & 0 & 0 & -2 \\ 0 & 0 & 0 & -2 \\ -2 & -2 & -2 & -4 \end{pmatrix}$$

Iteration #3

$$v_{xi} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \Delta v_{xi} = \begin{pmatrix} 0 & 0 & 0 & .5 \\ 0 & 0 & 0 & .5 \\ 0 & 0 & 0 & .5 \\ .5 & .5 & .5 & 1 \end{pmatrix} \quad \frac{\partial Cost}{\partial v_{xi}} = \begin{pmatrix} 2 & 10 & 2 & 10 \\ 0 & 7 & 0 & 7 \\ 5 & 10 & 5 & 10 \\ 8 & 10 & 8 & 10 \end{pmatrix}$$

$$\Gamma = \{v_{44}\}$$

$$\Gamma^* = \{v_{44}\}$$

This decides that city D is to be visited during the 2nd step of the tour. We set v_{44} ,
 $v_{44} = v_{44} + \Delta v_{44} = 0 + 1 = 1$, and compute the gradient of E_s .

$$\nabla E_s = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Since the gradient of $E_s = 0$ then the constraints are now satisfied. The algorithm ends, and the solution is stored in

$$v_{xi} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Which correspond to the tour A->B->C->D->A of length 17 which is optimal in this case.

Comparing the difference between both HNUA and MHNUA: The HNUA algorithm would have made 2 iterations and generate any of the following tours (following random values of v_{wk} at step [4]):

A->C->B->D->A of length 22

A->D->B->C->A of length 22

A->B->D->C->A of length 21

A->C->D->B->A of length 21

A->B->C->D->A of length 17

A->D->C->B->A of length 17

On the other hand, MHNUA makes 6 iterations exploring all possible values of v_{wk} and generating an optimal solution: A->B->C->D->A.

A.2. The Traveling Salesman Problem with N = 10

Consider the cities A, B, C, D, E, F, G, H, I, J and the following distance matrix:

$$d = \begin{Bmatrix} 0 & 4 & 10 & 13 & 7 & 9 & 3 & 17 & 8 & 13 \\ 4 & 0 & 9 & 1 & 2 & 6 & 15 & 8 & 13 & 5 \\ 10 & 9 & 0 & 8 & 9 & 9 & 6 & 4 & 9 & 3 \\ 13 & 1 & 8 & 0 & 6 & 7 & 8 & 12 & 16 & 12 \\ 7 & 2 & 9 & 6 & 0 & 13 & 15 & 16 & 12 & 3 \\ 9 & 6 & 9 & 7 & 13 & 0 & 12 & 8 & 16 & 6 \\ 3 & 15 & 6 & 8 & 15 & 12 & 0 & 16 & 11 & 11 \\ 17 & 8 & 4 & 12 & 16 & 8 & 16 & 0 & 2 & 4 \\ 8 & 13 & 9 & 16 & 12 & 16 & 11 & 2 & 0 & 9 \\ 13 & 5 & 3 & 12 & 3 & 6 & 11 & 4 & 9 & 0 \end{Bmatrix}$$

The Variables for this instance are:

$$V_{xi} = \begin{Bmatrix} v_{11} & v_{12} & v_{13} & v_{14} & v_{15} & v_{16} & v_{17} & v_{18} & v_{19} & v_{110} \\ v_{21} & v_{22} & v_{23} & v_{24} & v_{25} & v_{26} & v_{27} & v_{28} & v_{29} & v_{210} \\ v_{31} & v_{32} & v_{33} & v_{34} & v_{35} & v_{36} & v_{37} & v_{38} & v_{39} & v_{310} \\ v_{41} & v_{42} & v_{43} & v_{44} & v_{45} & v_{46} & v_{47} & v_{48} & v_{49} & v_{410} \\ v_{51} & v_{52} & v_{53} & v_{54} & v_{55} & v_{56} & v_{57} & v_{58} & v_{59} & v_{510} \\ v_{61} & v_{62} & v_{63} & v_{64} & v_{65} & v_{66} & v_{67} & v_{68} & v_{69} & v_{610} \\ v_{71} & v_{72} & v_{73} & v_{74} & v_{75} & v_{76} & v_{77} & v_{78} & v_{79} & v_{710} \\ v_{81} & v_{82} & v_{83} & v_{84} & v_{85} & v_{86} & v_{87} & v_{88} & v_{89} & v_{810} \\ v_{91} & v_{92} & v_{93} & v_{94} & v_{95} & v_{96} & v_{97} & v_{98} & v_{99} & v_{910} \\ v_{101} & v_{102} & v_{103} & v_{104} & v_{105} & v_{106} & v_{107} & v_{108} & v_{109} & v_{1010} \end{Bmatrix}$$

Iteration #1:

$$v_{xi} = \begin{Bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix} \Delta_{xi} = \begin{Bmatrix} 0 & 05 & 05 & 05 & 05 & 05 & 05 & 05 & 05 & 05 \\ 05 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 05 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 05 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 05 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 05 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 05 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 05 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 05 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 05 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{Bmatrix} \frac{\partial \text{Cost}}{\partial v_{xi}} = \begin{Bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 \\ 0 & 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 \\ 0 & 13 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 13 \\ 0 & 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7 \\ 0 & 9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \\ 0 & 17 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 17 \\ 0 & 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 8 \\ 0 & 13 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 13 \end{Bmatrix}$$

$$\Gamma = \{v_{22}, v_{23}, v_{24}, v_{25}, v_{26}, v_{27}, v_{28}, v_{29}, v_{210}, v_{32}, \dots, v_{102}, v_{103}, v_{104}, v_{105}, v_{106}, v_{107}, v_{108}, v_{109}, v_{1010}\}$$

$$\Gamma^* = \{v_{23}, v_{24}, v_{25}, v_{26}, v_{27}, v_{28}, v_{29}, v_{33}, \dots, v_{39}, v_{43}, \dots, v_{49}, \dots, v_{103}, v_{104}, v_{105}, v_{106}, v_{107}, v_{108}, v_{109}\}$$

We are about to select the next city to visit during the tour. All the possible cases shown in Γ^* are tested. Only the city with minimum length is listed below not all:

Let $v_{wk} = v_{107}$ (try city J). We update v_{107} ; $v_{107} = v_{107} + \Delta v_{107} = 0 + 1 = 1$.

Next we see how does this choice influence the other cities (only those order has not been decided). We compute $(\partial \text{Cost} / \partial v_{xi})$ for $x = 2, 3, 4, 5, 6, 7, 8, 9$.

$$\frac{\partial \text{Cost}}{\partial v_{xi}} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 4 & 0 & 0 & 0 & 5 & 0 & 5 & 0 & 4 \\ 0 & 10 & 0 & 0 & 0 & 3 & 0 & 3 & 0 & 10 \\ 0 & 13 & 0 & 0 & 0 & 12 & 0 & 12 & 0 & 13 \\ 0 & 7 & 0 & 0 & 0 & 3 & 0 & 3 & 0 & 7 \\ 0 & 9 & 0 & 0 & 0 & 6 & 0 & 6 & 0 & 9 \\ 0 & 3 & 0 & 0 & 0 & 11 & 0 & 11 & 0 & 3 \\ 0 & 17 & 0 & 0 & 0 & 4 & 0 & 4 & 0 & 17 \\ 0 & 8 & 0 & 0 & 0 & 9 & 0 & 9 & 0 & 8 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

This says that city B has an associate cost of length 4 if visited at step 2 or 10 and of length 5 if visited at step 6 or 8. City C has an associate cost of length 10 if visited at step 2 or 10 and of length 3 if visited at length 6 or 8 and so on. The above data corresponds to tour lengths which are: 71 and 53.

$$\text{Cost}_{v_{107}} = \sum \max\left(\frac{\partial \text{Cost}}{\partial v_{xi}}, 1 \leq i \leq 10\right) = 71$$

Since it is V_{107} which provides the minimum cost from the entire element in Γ^* , we choose to visit city J at step 7 during the tour. The gradient E_s :

$$\nabla E_s = \begin{pmatrix} 0 & -2 & -2 & -2 & -2 & -2 & 0 & -2 & -2 & -2 \\ -2 & -4 & -4 & -4 & -4 & -4 & 0 & -4 & -4 & -4 \\ -2 & -4 & -4 & -4 & -4 & -4 & 0 & -4 & -4 & -4 \\ -2 & -4 & -4 & -4 & -4 & -4 & 0 & -4 & -4 & -4 \\ -2 & -4 & -4 & -4 & -4 & -4 & 0 & -4 & -4 & -4 \\ -2 & -4 & -4 & -4 & -4 & -4 & 0 & -4 & -4 & -4 \\ -2 & -4 & -4 & -4 & -4 & -4 & 0 & -4 & -4 & -4 \\ -2 & -4 & -4 & -4 & -4 & -4 & 0 & -4 & -4 & -4 \\ 0 & -2 & -2 & -2 & -2 & -2 & 0 & -2 & -2 & -2 \end{pmatrix}$$

Iteration #2:

$$v_{xi} = \begin{Bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{Bmatrix} \Delta v_{xi} = \begin{Bmatrix} 0 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0 & 0.5 & 0.5 & 0.5 \\ 0.5 & 1 & 1 & 1 & 1 & 1 & 0.5 & 1 & 1 & 1 \\ 0.5 & 1 & 1 & 1 & 1 & 1 & 0.5 & 1 & 1 & 1 \\ 0.5 & 1 & 1 & 1 & 1 & 1 & 0.5 & 1 & 1 & 1 \\ 0.5 & 1 & 1 & 1 & 1 & 1 & 0.5 & 1 & 1 & 1 \\ 0.5 & 1 & 1 & 1 & 1 & 1 & 0.5 & 1 & 1 & 1 \\ 0.5 & 1 & 1 & 1 & 1 & 1 & 0.5 & 1 & 1 & 1 \\ 0.5 & 1 & 1 & 1 & 1 & 1 & 0.5 & 1 & 1 & 1 \\ 0.5 & 1 & 1 & 1 & 1 & 1 & 0.5 & 1 & 1 & 1 \\ 0.5 & 1 & 1 & 1 & 1 & 1 & 0.5 & 1 & 1 & 1 \\ 0 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0 & 0.5 & 0.5 & 0.5 \end{Bmatrix} \frac{\partial \text{Cost}}{\partial v_{xi}} = \begin{Bmatrix} 0 & 0 & 0 & 0 & 0 & 13 & 0 & 0 & 13 & 0 \\ 0 & 4 & 0 & 0 & 0 & 5 & 0 & 5 & 0 & 4 \\ 0 & 10 & 0 & 0 & 0 & 3 & 0 & 3 & 0 & 10 \\ 0 & 13 & 0 & 0 & 0 & 12 & 0 & 12 & 0 & 13 \\ 0 & 7 & 0 & 0 & 0 & 3 & 0 & 3 & 0 & 7 \\ 0 & 9 & 0 & 0 & 0 & 6 & 0 & 6 & 0 & 9 \\ 0 & 3 & 0 & 0 & 0 & 11 & 0 & 11 & 0 & 3 \\ 0 & 17 & 0 & 0 & 0 & 4 & 0 & 4 & 0 & 17 \\ 0 & 8 & 0 & 0 & 0 & 9 & 0 & 9 & 0 & 8 \\ 0 & 13 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 13 \end{Bmatrix}$$

$$\Gamma = \{v_{22}, v_{23}, v_{24}, v_{25}, v_{26}, v_{28}, v_{29}, v_{210}, v_{32}, \dots, v_{36}, v_{38}, \dots, v_{310}, \dots, v_{92}, \dots, v_{96}, v_{98}, \dots, v_{910}\}$$

$$\Gamma^* = \{v_{22}, v_{23}, v_{24}, v_{25}, v_{29}, v_{32}, v_{33}, v_{34}, v_{35}, v_{39}, v_{42}, v_{43}, v_{44}, v_{45}, v_{49}, \dots, v_{92}, v_{93}, v_{94}, v_{95}, v_{99}\}$$

We are about to select the next city to visit during the tour. All the possible cases shown in Γ^* are tested. Only the city with minimum length is listed below:

Let $v_{wk} = v_{44}$ (try city D). We update v_{44} ; $v_{44} = v_{44} + \Delta v_{44} = 0 + 1 = 1$.

Next we see how does this choice influence the other cities (only those order has not been decided). We compute $(\partial \text{Cost} / \partial v_{xi})$ for $x = 2, 3, 5, 6, 7, 8, 9$.

$$\frac{\partial \text{Cost}}{\partial v_{xi}} = \begin{Bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 4 & 1 & 0 & 1 & 5 & 0 & 5 & 0 & 4 \\ 0 & 10 & 8 & 0 & 8 & 3 & 0 & 3 & 0 & 10 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 7 & 6 & 0 & 6 & 3 & 0 & 3 & 0 & 7 \\ 0 & 9 & 7 & 0 & 7 & 6 & 0 & 6 & 0 & 9 \\ 0 & 3 & 8 & 0 & 8 & 11 & 0 & 11 & 0 & 3 \\ 0 & 17 & 12 & 0 & 12 & 4 & 0 & 4 & 0 & 17 \\ 0 & 8 & 16 & 0 & 16 & 9 & 0 & 9 & 0 & 8 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{Bmatrix}$$

This says that city B has an associate cost of length 4 if visited at step 2 or 10, of length 5 if visited at step 6 or 8 and 1 if visited at step 3 or 5.

City C has an associate cost of length 10 if visited at step 2 or 10, of length 3 if visited at length 6 or 8 and of length 3 if visited at length 6 or 8 and so on. The above data corresponds to the tour lengths which are: 58 and 41.

$$Costv_{44} = \sum \max\left(\frac{\partial Cost}{\partial v_{xi}}, 1 \leq i \leq 10\right) = 58$$

Since it is V_{44} which provides the minimum cost from the entire element in Γ^* , we choose to visit city D at step 4 during the tour. The gradient E_s :

$$\nabla E_s = \begin{pmatrix} 0 & -2 & -2 & 0 & -2 & -2 & 0 & -2 & -2 & -2 \\ -2 & -4 & -4 & -2 & -4 & -4 & 0 & -4 & -4 & -4 \\ -2 & -4 & -4 & -2 & -4 & -4 & 0 & -4 & -4 & -4 \\ 0 & -2 & -2 & 0 & -2 & -2 & 0 & -2 & -2 & -2 \\ -2 & -4 & -4 & -2 & -4 & -4 & 0 & -4 & -4 & -4 \\ -2 & -4 & -4 & -2 & -4 & -4 & 0 & -4 & -4 & -4 \\ -2 & -4 & -4 & -2 & -4 & -4 & 0 & -4 & -4 & -4 \\ -2 & -4 & -4 & -2 & -4 & -4 & 0 & -4 & -4 & -4 \\ -2 & -4 & -4 & -2 & -4 & -4 & 0 & -4 & -4 & -4 \\ 0 & -2 & -2 & 0 & -2 & -2 & 0 & -2 & -2 & -2 \end{pmatrix}$$

Iteration #3:

$$v_{xi} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \Delta v_{xi} = \begin{pmatrix} 0 & 0.5 & 0.5 & 0 & 0.5 & 0.5 & 0 & 0.5 & 0.5 & 0.5 \\ 0.5 & 1 & 1 & 0.5 & 1 & 1 & 0.5 & 1 & 1 & 1 \\ 0.5 & 1 & 1 & 0.5 & 1 & 1 & 0.5 & 1 & 1 & 1 \\ 0 & 0.5 & 0.5 & 0 & 0.5 & 0.5 & 0 & 0.5 & 0.5 & 0.5 \\ 0.5 & 1 & 1 & 0.5 & 1 & 1 & 0.5 & 1 & 1 & 1 \\ 0.5 & 1 & 1 & 0.5 & 1 & 1 & 0.5 & 1 & 1 & 1 \\ 0.5 & 1 & 1 & 0.5 & 1 & 1 & 0.5 & 1 & 1 & 1 \\ 0.5 & 1 & 1 & 0.5 & 1 & 1 & 0.5 & 1 & 1 & 1 \\ 0.5 & 1 & 1 & 0.5 & 1 & 1 & 0.5 & 1 & 1 & 1 \\ 0 & 0.5 & 0.5 & 0 & 0.5 & 0.5 & 0 & 0.5 & 0.5 & 0.5 \end{pmatrix} \frac{\partial Cost}{\partial v_{xi}} = \begin{pmatrix} 0 & 0 & 13 & 0 & 13 & 13 & 0 & 13 & 0 & 0 \\ 0 & 4 & 1 & 0 & 1 & 5 & 0 & 5 & 0 & 4 \\ 0 & 10 & 8 & 0 & 8 & 3 & 0 & 3 & 0 & 10 \\ 0 & 13 & 0 & 0 & 0 & 12 & 0 & 12 & 0 & 13 \\ 0 & 7 & 6 & 0 & 6 & 3 & 0 & 3 & 0 & 7 \\ 0 & 9 & 7 & 0 & 7 & 6 & 0 & 6 & 0 & 9 \\ 0 & 3 & 8 & 0 & 8 & 11 & 0 & 11 & 0 & 3 \\ 0 & 17 & 12 & 0 & 12 & 4 & 0 & 4 & 0 & 17 \\ 0 & 8 & 16 & 0 & 16 & 9 & 0 & 9 & 0 & 8 \\ 0 & 13 & 12 & 0 & 12 & 0 & 0 & 0 & 0 & 13 \end{pmatrix}$$

$$\Gamma = \{v_{22}, v_{23}, v_{24}, v_{25}, v_{26}, v_{28}, v_{29}, v_{210}, v_{32}, \dots, v_{36}, v_{38}, \dots, v_{310}, \dots, v_{92}, \dots, v_{96}, v_{98}, \dots, v_{910}\}$$

$$\Gamma^* = \{v_{22}, v_{23}, v_{24}, v_{25}, v_{29}, v_{32}, v_{33}, v_{34}, v_{35}, v_{39}, v_{42}, v_{43}, v_{44}, v_{45}, v_{49}, \dots, v_{92}, v_{93}, v_{94}, v_{95}, v_{99}\}$$

We are about to select the next city to visit during the tour. All the possible cases shown in Γ^* are tested. Only the city with minimum length is listed below:

Let $v_{wk} = v_{39}$ (try city C). We update v_{39} ; $v_{39} = v_{39} + \Delta v_{39} = 0 + 1 = 1$.

Next we see how does this choice influence the other cities (only those order has not been decided). We compute $(\partial Cost / \partial v_{xi})$ for $x = 2, 5, 6, 7, 8, 9$.

$$\frac{\partial Cost}{\partial v_{xi}} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 4 & 1 & 0 & 1 & 5 & 0 & 14 & 0 & 13 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 7 & 6 & 0 & 6 & 3 & 0 & 12 & 0 & 16 \\ 0 & 9 & 7 & 0 & 7 & 6 & 0 & 15 & 0 & 18 \\ 0 & 3 & 8 & 0 & 8 & 11 & 0 & 17 & 0 & 9 \\ 0 & 17 & 12 & 0 & 12 & 4 & 0 & 8 & 0 & 21 \\ 0 & 8 & 16 & 0 & 16 & 9 & 0 & 18 & 0 & 17 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

$$Costv_{39} = \sum \max\left(\frac{\partial Cost}{\partial v_{xi}}, 1 \leq i \leq 10\right) = 94$$

This says that city B has an associate cost of length 4 if visited at step 2, of length 1 if visited at step 3 or 5, of length 5 if visited at step 6, of length 14 if visited at step 8 and of length 13 if visited at step 10 and so on.

Since it is V_{39} which provides the minimum cost from the entire element in Γ^* , we choose to visit city C at step 9 during the tour. The gradient E_s :

$$\nabla E_s = \begin{pmatrix} 0 & -2 & -2 & 0 & -2 & -2 & 0 & -2 & 0 & -2 \\ -2 & -4 & -4 & -2 & -4 & -4 & -2 & -4 & -2 & -4 \\ 0 & -2 & -2 & 0 & -2 & -2 & 0 & -2 & 0 & -2 \\ 0 & -2 & -2 & 0 & -2 & -2 & 0 & -2 & 0 & -2 \\ -2 & -4 & -4 & -2 & -4 & -4 & -2 & -4 & -2 & -4 \\ -2 & -4 & -4 & -2 & -4 & -4 & -2 & -4 & -2 & -4 \\ -2 & -4 & -4 & -2 & -4 & -4 & -2 & -4 & -2 & -4 \\ -2 & -4 & -4 & -2 & -4 & -4 & -2 & -4 & -2 & -4 \\ -2 & -4 & -4 & -2 & -4 & -4 & -2 & -4 & -2 & -4 \\ 0 & -2 & -2 & 0 & -2 & -2 & 0 & -2 & 0 & -2 \end{pmatrix}$$

Iteration #4:

$$v_{xi} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \Delta v_{xi} = \begin{pmatrix} 0 & 0.5 & 0.5 & 0 & 0.5 & 0.5 & 0 & 0.5 & 0 & 0.5 \\ 0.5 & 1 & 1 & 0.5 & 1 & 1 & 0.5 & 1 & 0.5 & 1 \\ 0 & 0.5 & 0.5 & 0 & 0.5 & 0.5 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0.5 & 0.5 & 0 & 0.5 & 0.5 & 0 & 0.5 & 0 & 0.5 \\ 0.5 & 1 & 1 & 0.5 & 1 & 1 & 0.5 & 1 & 0.5 & 1 \\ 0.5 & 1 & 1 & 0.5 & 1 & 1 & 0.5 & 1 & 0.5 & 1 \\ 0.5 & 1 & 1 & 0.5 & 1 & 1 & 0.5 & 1 & 0.5 & 1 \\ 0.5 & 1 & 1 & 0.5 & 1 & 1 & 0.5 & 1 & 0.5 & 1 \\ 0.5 & 1 & 1 & 0.5 & 1 & 1 & 0.5 & 1 & 0.5 & 1 \\ 0 & 0.5 & 0.5 & 0 & 0.5 & 0.5 & 0 & 0.5 & 0 & 0.5 \end{pmatrix} \frac{\partial Cost}{\partial v_{xi}} = \begin{pmatrix} 0 & 0 & 13 & 0 & 13 & 13 & 0 & 23 & 0 & 10 \\ 0 & 4 & 1 & 0 & 1 & 5 & 0 & 14 & 0 & 13 \\ 0 & 10 & 8 & 0 & 8 & 3 & 0 & 3 & 0 & 10 \\ 0 & 13 & 0 & 0 & 0 & 12 & 0 & 20 & 0 & 21 \\ 0 & 7 & 6 & 0 & 6 & 3 & 0 & 12 & 0 & 16 \\ 0 & 9 & 7 & 0 & 7 & 6 & 0 & 15 & 0 & 18 \\ 0 & 3 & 8 & 0 & 8 & 11 & 0 & 17 & 0 & 9 \\ 0 & 17 & 12 & 0 & 12 & 4 & 0 & 8 & 0 & 21 \\ 0 & 8 & 16 & 0 & 16 & 9 & 0 & 18 & 0 & 17 \\ 0 & 13 & 12 & 0 & 12 & 0 & 0 & 3 & 0 & 16 \end{pmatrix}$$

$$\Gamma = \{v_{22}, v_{23}, v_{25}, v_{26}, v_{28}, v_{210}, v_{52}, v_{53}, v_{55}, v_{56}, v_{58}, v_{510}, \dots, v_{92}, v_{93}, v_{95}, v_{96}, v_{98}, v_{910}\}$$

$$\Gamma^* = \{v_{23}, v_{25}\}$$

We are about to select the next city to visit during the tour. All the possible cases shown in Γ^* are tested. Only the city with minimum length is listed below:

Let $v_{wk} = v_{25}$ (try city B). We update v_{25} ; $v_{25} = v_{25} + \Delta v_{25} = 0 + 1 = 1$.

Next we see how does this choice influence the other cities (only those order has not been decided). We compute $(\partial Cost / \partial v_{xi})$ for $x = 5, 6, 7, 8, 9$.

$$\frac{\partial Cost}{\partial v_{xi}} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 7 & 6 & 2 & 6 & 5 & 0 & 12 & 0 & 16 \\ 0 & 9 & 7 & 6 & 7 & 12 & 0 & 15 & 0 & 18 \\ 0 & 3 & 8 & 15 & 8 & 26 & 0 & 17 & 0 & 9 \\ 0 & 17 & 12 & 8 & 12 & 12 & 0 & 8 & 0 & 21 \\ 0 & 8 & 16 & 13 & 16 & 22 & 0 & 18 & 0 & 17 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

$$Cost_{v_{25}} = \sum \max\left(\frac{\partial Cost}{\partial v_{xi}}, 1 \leq i \leq 10\right) = 81$$

Since it is V_{25} which provides the minimum cost from the entire element in Γ^* , we choose to visit city B at step 5 during the tour. The gradient E_5 :

$$\nabla E_5 = \begin{pmatrix} 0 & -2 & -2 & 0 & 0 & -2 & 0 & -2 & 0 & -2 \\ 0 & -2 & -2 & 0 & 0 & -2 & 0 & -2 & 0 & -2 \\ 0 & -2 & -2 & 0 & 0 & -2 & 0 & -2 & 0 & -2 \\ 0 & -2 & -2 & 0 & 0 & -2 & 0 & -2 & 0 & -2 \\ -2 & -4 & -4 & -2 & -2 & -4 & -2 & -4 & -2 & -4 \\ -2 & -4 & -4 & -2 & -2 & -4 & -2 & -4 & -2 & -4 \\ -2 & -4 & -4 & -2 & -2 & -4 & -2 & -4 & -2 & -4 \\ -2 & -4 & -4 & -2 & -2 & -4 & -2 & -4 & -2 & -4 \\ -2 & -4 & -4 & -2 & -2 & -4 & -2 & -4 & -2 & -4 \\ 0 & -2 & -2 & 0 & 0 & -2 & 0 & -2 & 0 & -2 \end{pmatrix}$$

Iteration #5:

$$v_{x_i} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \Delta v_{xi} = \begin{pmatrix} 0 & 0.5 & 0.5 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0.5 & 0.5 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0.5 & 0.5 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0.5 & 0.5 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0.5 \\ 0.5 & 1 & 1 & 0.5 & 0.5 & 1 & 0.5 & 1 & 0.5 & 1 \\ 0.5 & 1 & 1 & 0.5 & 0.5 & 1 & 0.5 & 1 & 0.5 & 1 \\ 0.5 & 1 & 1 & 0.5 & 0.5 & 1 & 0.5 & 1 & 0.5 & 1 \\ 0.5 & 1 & 1 & 0.5 & 0.5 & 1 & 0.5 & 1 & 0.5 & 1 \\ 0.5 & 1 & 1 & 0.5 & 0.5 & 1 & 0.5 & 1 & 0.5 & 1 \\ 0 & 0.5 & 0.5 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0.5 \end{pmatrix} \frac{\partial Cost}{\partial v_{xi}} = \begin{pmatrix} 0 & 0 & 13 & 4 & 13 & 17 & 0 & 23 & 0 & 10 \\ 0 & 4 & 1 & 0 & 1 & 5 & 0 & 14 & 0 & 13 \\ 0 & 10 & 8 & 9 & 8 & 12 & 0 & 3 & 0 & 10 \\ 0 & 13 & 0 & 1 & 0 & 13 & 0 & 20 & 0 & 21 \\ 0 & 7 & 6 & 2 & 6 & 5 & 0 & 12 & 0 & 16 \\ 0 & 9 & 7 & 6 & 7 & 12 & 0 & 15 & 0 & 18 \\ 0 & 3 & 8 & 15 & 8 & 26 & 0 & 17 & 0 & 9 \\ 0 & 17 & 12 & 8 & 12 & 12 & 0 & 8 & 0 & 21 \\ 0 & 8 & 16 & 13 & 16 & 22 & 0 & 18 & 0 & 17 \\ 0 & 13 & 12 & 5 & 12 & 5 & 0 & 3 & 0 & 16 \end{pmatrix}$$

$$\Gamma = \{v_{52}, v_{53}, v_{56}, v_{58}, v_{510}, v_{62}, v_{63}, v_{66}, v_{68}, v_{610}, \dots, v_{92}, v_{93}, v_{96}, v_{98}, v_{910}\}$$

$$\Gamma^* = \{v_{72}\}$$

We are about to select the next city to visit during the tour. All the possible cases shown in Γ^* are tested. Only the city with minimum length is listed below:

Let $v_{wk} = v_{72}$ (try city G). We update v_{72} ; $v_{72} = v_{72} + \Delta v_{72} = 0 + 1 = 1$.

Next we see how does this choice influence the other cities (only those order has not been decided). We compute $(\partial Cost / \partial v_{xi})$ for $x = 5, 6, 8, 9$.

$$\frac{\partial Cost}{\partial v_{xi}} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 15 & 7 & 21 & 2 & 6 & 5 & 0 & 12 & 0 & 16 & \\ 12 & 9 & 19 & 6 & 7 & 12 & 0 & 15 & 0 & 18 & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 16 & 17 & 28 & 8 & 12 & 12 & 0 & 8 & 0 & 21 & \\ 11 & 8 & 27 & 13 & 16 & 22 & 0 & 18 & 0 & 17 & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

$$Cost_{v_{72}} = \sum \max\left(\frac{\partial Cost}{\partial v_{xi}}, 1 \leq i \leq 10\right) = 95$$

Since it is V_{72} which provides the minimum cost from the entire element in Γ^* , we choose to visit city G at step 2 during the tour. The gradient E_s :

$$\nabla E_s = \begin{pmatrix} 0 & 0 & -2 & 0 & 0 & -2 & 0 & -2 & 0 & -2 \\ 0 & 0 & -2 & 0 & 0 & -2 & 0 & -2 & 0 & -2 \\ 0 & 0 & -2 & 0 & 0 & -2 & 0 & -2 & 0 & -2 \\ 0 & 0 & -2 & 0 & 0 & -2 & 0 & -2 & 0 & -2 \\ -2 & -2 & -4 & -2 & -2 & -4 & -2 & -4 & -2 & -4 \\ -2 & -2 & -4 & -2 & -2 & -4 & -2 & -4 & -2 & -4 \\ 0 & 0 & -2 & 0 & 0 & -2 & 0 & -2 & 0 & -2 \\ -2 & -2 & -4 & -2 & -2 & -4 & -2 & -4 & -2 & -4 \\ -2 & -2 & -4 & -2 & -2 & -4 & -2 & -4 & -2 & -4 \\ 0 & 0 & -2 & 0 & 0 & -2 & 0 & -2 & 0 & -2 \end{pmatrix}$$

Iteration #6:

$$v_{xi} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \Delta x_i = \begin{pmatrix} 0 & 0 & 0.5 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0.5 \\ 0.5 & 0.5 & 1 & 0.5 & 0.5 & 1 & 0.5 & 1 & 0.5 & 1 \\ 0.5 & 0.5 & 1 & 0.5 & 0.5 & 1 & 0.5 & 1 & 0.5 & 1 \\ 0.5 & 0.5 & 1 & 0.5 & 0.5 & 1 & 0.5 & 1 & 0.5 & 1 \\ 0.5 & 0.5 & 1 & 0.5 & 0.5 & 1 & 0.5 & 1 & 0.5 & 1 \\ 0.5 & 0.5 & 1 & 0.5 & 0.5 & 1 & 0.5 & 1 & 0.5 & 1 \\ 0 & 0 & 0.5 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0.5 \end{pmatrix} \frac{\partial Cost}{\partial v_{xi}} = \begin{pmatrix} 3 & 0 & 16 & 4 & 20 & 17 & 7 & 23 & 0 & 10 \\ 15 & 4 & 16 & 0 & 3 & 5 & 2 & 14 & 0 & 13 \\ 6 & 10 & 14 & 9 & 17 & 12 & 9 & 3 & 0 & 10 \\ 8 & 13 & 8 & 1 & 6 & 13 & 6 & 20 & 0 & 21 \\ 15 & 7 & 21 & 2 & 6 & 5 & 0 & 12 & 0 & 16 \\ 12 & 9 & 19 & 6 & 20 & 12 & 13 & 15 & 0 & 18 \\ 0 & 3 & 8 & 15 & 23 & 26 & 15 & 17 & 0 & 9 \\ 16 & 17 & 28 & 8 & 28 & 12 & 16 & 8 & 0 & 21 \\ 11 & 8 & 27 & 13 & 28 & 22 & 12 & 18 & 0 & 17 \\ 11 & 13 & 23 & 5 & 15 & 5 & 3 & 3 & 0 & 16 \end{pmatrix}$$

$$\Gamma = \{53, v_{56}, v_{58}, v_{510}, v_{63}, v_{66}, v_{68}, v_{610}, \dots, v_{93}, v_{96}, v_{98}, v_{910}\}$$

$$\Gamma^* = \{v_{56}\}$$

We are about to select the next city to visit during the tour. All the possible cases shown in Γ^* are tested. Only the city with minimum length is listed below:

Let $v_{wk} = v_{56}$ (try city E). We update v_{56} : $v_{56} = v_{56} + \Delta v_{72} = 0 + 1 = 1$.

Next we see how does this choice influence the other cities (only those order has not been decided). We compute $(\partial \text{Cost} / \partial v_{xi})$ for $x = 6, 8, 9$.

$$\frac{\partial \text{Cost}}{\partial v_{xi}} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 12 & 9 & 19 & 6 & 20 & 12 & 13 & 15 & 0 & 18 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 16 & 17 & 28 & 8 & 28 & 12 & 16 & 8 & 0 & 21 \\ 11 & 8 & 27 & 13 & 28 & 22 & 12 & 18 & 0 & 17 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

$$\text{Cost}_{v_{56}} = \sum \max\left(\frac{\partial \text{Cost}}{\partial v_{xi}}, 1 \leq i \leq 10\right) = 76$$

Since it is V_{56} which provides the minimum cost from the entire element in Γ^* , we choose to visit city E at step 6 during the tour. The gradient E_s :

$$\nabla E_s = \begin{pmatrix} 0 & 0 & -2 & 0 & 0 & 0 & 0 & -2 & 0 & -2 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & -2 & 0 & -2 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & -2 & 0 & -2 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & -2 & 0 & -2 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & -2 & 0 & -2 \\ -2 & -2 & -4 & -2 & -2 & -2 & -2 & -4 & -2 & -4 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & -2 & 0 & -2 \\ -2 & -2 & -4 & -2 & -2 & -2 & -2 & -4 & -2 & -4 \\ -2 & -2 & -4 & -2 & -2 & -2 & -2 & -4 & -2 & -4 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & -2 & 0 & -2 \end{pmatrix}$$

Iteration #7:

$$v_{x_i} = \begin{Bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{Bmatrix} \Delta v_{xi} = \begin{Bmatrix} 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 \\ 0.5 & 0.5 & 1 & 0.5 & 0.5 & 0.5 & 0.5 & 1 & 0.5 & 1 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 \\ 0.5 & 0.5 & 1 & 0.5 & 0.5 & 0.5 & 0.5 & 1 & 0.5 & 1 \\ 0.5 & 0.5 & 1 & 0.5 & 0.5 & 0.5 & 0.5 & 1 & 0.5 & 1 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 \end{Bmatrix} \frac{\partial Cost}{\partial v_{xi}} = \begin{Bmatrix} 3 & 0 & 16 & 4 & 20 & 17 & 7 & 23 & 0 & 10 \\ 15 & 4 & 16 & 0 & 3 & 5 & 2 & 14 & 0 & 13 \\ 6 & 10 & 14 & 9 & 17 & 12 & 9 & 3 & 0 & 10 \\ 8 & 13 & 8 & 1 & 6 & 13 & 6 & 20 & 0 & 21 \\ 15 & 7 & 21 & 2 & 6 & 5 & 0 & 12 & 0 & 16 \\ 12 & 9 & 19 & 6 & 20 & 12 & 13 & 15 & 0 & 18 \\ 0 & 3 & 8 & 15 & 23 & 26 & 15 & 17 & 0 & 9 \\ 16 & 17 & 28 & 8 & 28 & 12 & 16 & 8 & 0 & 21 \\ 11 & 8 & 27 & 13 & 28 & 22 & 12 & 18 & 0 & 17 \\ 11 & 13 & 23 & 5 & 15 & 5 & 3 & 3 & 0 & 16 \end{Bmatrix}$$

$$\Gamma = \{v_{63}, v_{68}, v_{610}, v_{83}, v_{88}, v_{810}, v_{93}, v_{98}, v_{910}\}$$

$$\Gamma^* = \{v_{88}\}$$

We are about to select the next city to visit during the tour. All the possible cases shown in Γ^* are tested. Only the city with minimum length is listed below:

Let $v_{wk} = v_{88}$ (try city H). We update v_{88} ; $v_{88} = v_{88} + \Delta v_{72} = 0 + 1 = 1$.

Next we see how does this choice influence the other cities (only those order has not been decided). We compute $(\partial Cost / \partial v_{xi})$ for $x = 6, 9$.

$$\frac{\partial Cost}{\partial v_{xi}} = \begin{Bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 12 & 9 & 19 & 6 & 20 & 12 & 13 & 15 & 8 & 18 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 11 & 8 & 27 & 13 & 28 & 22 & 12 & 18 & 2 & 17 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{Bmatrix}$$

$$Cost_{v_{88}} = \sum \max\left(\frac{\partial Cost}{\partial v_{xi}}, 1 \leq i \leq 10\right) = 48$$

Since it is V_{88} which provides the minimum cost from the entire element in Γ^* , we choose to visit city E at step 6 during the tour. The gradient E_s :

$$\nabla E_s = \begin{pmatrix} 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & -2 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & -2 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & -2 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & -2 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & -2 \\ -2 & -2 & -4 & -2 & -2 & -2 & -2 & -2 & -2 & -4 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & -2 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & -2 \\ -2 & -2 & -4 & -2 & -2 & -2 & -2 & -2 & -2 & -4 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & -2 \end{pmatrix}$$

Iteration #8:

$$v_{xi} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \Delta v_{xi} = \begin{pmatrix} 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 \\ 0.5 & 0.5 & 1 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 1 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 \\ 0.5 & 0.5 & 1 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 1 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 \end{pmatrix} \frac{\partial Cost}{\partial v_{xi}} = \begin{pmatrix} 3 & 0 & 16 & 4 & 20 & 17 & 7 & 23 & 0 & 10 \\ 15 & 4 & 16 & 0 & 3 & 5 & 2 & 14 & 0 & 13 \\ 6 & 10 & 14 & 9 & 17 & 12 & 9 & 3 & 0 & 10 \\ 8 & 13 & 8 & 1 & 6 & 13 & 6 & 20 & 0 & 21 \\ 15 & 7 & 21 & 2 & 6 & 5 & 0 & 12 & 0 & 16 \\ 12 & 9 & 19 & 6 & 20 & 12 & 13 & 15 & 0 & 18 \\ 0 & 3 & 8 & 15 & 23 & 26 & 15 & 17 & 0 & 9 \\ 16 & 17 & 28 & 8 & 28 & 12 & 16 & 8 & 0 & 21 \\ 11 & 8 & 27 & 13 & 28 & 22 & 12 & 18 & 0 & 17 \\ 11 & 13 & 23 & 5 & 15 & 5 & 3 & 3 & 0 & 16 \end{pmatrix}$$

$$\Gamma = \{v_{63}, v_{610}, v_{93}, v_{910}\}$$

$$\Gamma^* = \{v_{910}\}$$

We are about to select the next city to visit during the tour. All the possible cases shown in Γ^* are tested. Only the city with minimum length is listed below:

Let $v_{wk} = v_{910}$ (try city H). We update v_{910} ; $v_{910} = v_{910} + \Delta v_{72} = 0 + 1 = 1$.

Next we see how does this choice influence the other cities (only those order has not been decided). We compute $(\partial Cost / \partial v_{xi})$ for $x = 6$.

$$\frac{\partial Cost}{\partial v_{xi}} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 28 & 9 & 19 & 6 & 20 & 12 & 21 & 15 & 24 & 18 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

$$Costv_{910} = \sum \max\left(\frac{\partial Cost}{\partial v_{xi}}, 1 \leq i \leq 10\right) = 28$$

Since it is V_{910} which provides the minimum cost from the entire element in Γ^* , we choose to visit city I at step 10 during the tour. The gradient E_s :

$$\nabla E_s = \begin{pmatrix} 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -2 & -2 & -4 & -2 & -2 & -2 & -2 & -2 & -2 & -2 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Iteration #9:

$$v_{xi} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \Delta v_{xi} = \begin{pmatrix} 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0.5 & 1 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \frac{\partial Cost}{\partial v_{xi}} = \begin{pmatrix} 11 & 0 & 16 & 4 & 20 & 17 & 24 & 23 & 25 & 10 \\ 28 & 4 & 16 & 0 & 3 & 5 & 10 & 14 & 21 & 13 \\ 15 & 10 & 14 & 9 & 17 & 12 & 13 & 3 & 13 & 10 \\ 24 & 13 & 8 & 1 & 6 & 13 & 18 & 20 & 28 & 21 \\ 27 & 7 & 21 & 2 & 6 & 5 & 16 & 12 & 28 & 16 \\ 28 & 9 & 19 & 6 & 20 & 12 & 21 & 15 & 24 & 18 \\ 11 & 3 & 8 & 15 & 23 & 26 & 31 & 17 & 27 & 9 \\ 18 & 17 & 28 & 8 & 28 & 12 & 16 & 8 & 2 & 21 \\ 11 & 8 & 27 & 13 & 28 & 22 & 14 & 18 & 2 & 17 \\ 20 & 13 & 23 & 5 & 15 & 5 & 7 & 3 & 13 & 16 \end{pmatrix}$$

$$\Gamma = \{v_{63}\}$$

$$\Gamma^* = \{v_{63}\}$$

We are about to select the next city to visit during the tour. All the possible cases shown in Γ^* are tested. Only the city with minimum length is listed below:

Let $v_{wk} = v_{63}$ (try city F). We update v_{63} : $v_{63} = v_{63} + \Delta v_{63} = 0 + 1 = 1$, and compute

Since it is V_{63} which provides the minimum cost, we choose to visit city F at step 3 during the tour. The gradient E_s :

$$\nabla E_s = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Since gradient of $E_s = 0$ then the constraints are now satisfied. The algorithm ends, and the solution is stored in

$$v_{x_i} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Which corresponds to $A \rightarrow G \rightarrow F \rightarrow D \rightarrow B \rightarrow E \rightarrow J \rightarrow H \rightarrow C \rightarrow I \rightarrow A$ of length 53 which is near optimal solution.

If we run DFSA application on the same input file holding the same number of cities we observe that the hybrid neural network produce 85% optimal result. The DFSA application output result is shown in the figure below:



```
DepthFirstSearch_TSP
Auto
The number of cities are: 10
The Matrix showing distance between 10 cities is:
0 4 10 13 7 9 3 17 8 13
4 0 9 1 2 6 15 8 13 5
10 9 0 8 9 9 6 4 9 3
13 1 8 0 6 7 8 12 16 12
7 2 9 6 0 13 15 16 12 3
9 6 9 7 13 0 12 8 16 6
3 15 6 8 15 12 0 16 11 11
17 8 4 12 16 8 16 0 2 4
8 13 9 16 12 16 11 2 0 9
13 5 3 12 3 6 11 4 9 0

TSP Tour is: 1 7 3 10 5 2 4 6 8 9
of distance 46
```

Figure A.1: DSFA Output Result for N = 10

APPENDIX B

B.1. TSP Interface Application Using GIS-MO

B.1.1. Launching the Application

The user has only to click the icon dedicated for the application. Next he will click on the OK button in the form listed below.

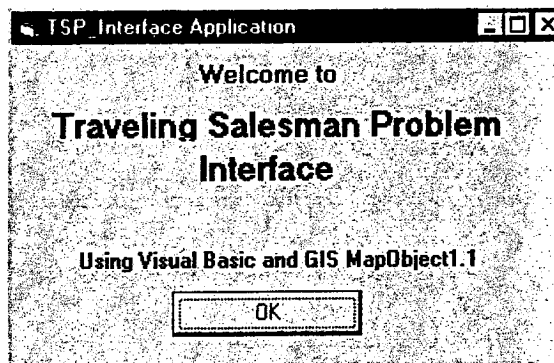


Figure B.1: Login Form

B.1.2. Running the Application

Step1: The shapefile layers used in this application are: city layer (capital), street layer (ushigh) and states layer. After launching the application, the three layers are automatically displayed in the map canvas. If we want to add, remove or change the color of any layer, click on the Map Content button in the toolbar, the Map Content form will appear as below.

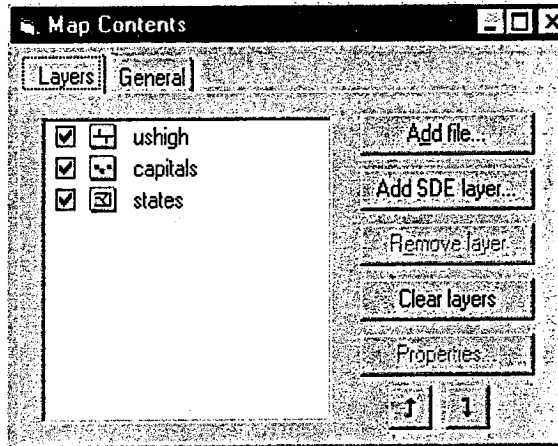


Figure B.2: Map Contents Form

What follows is a description of some buttons in the toolbar:

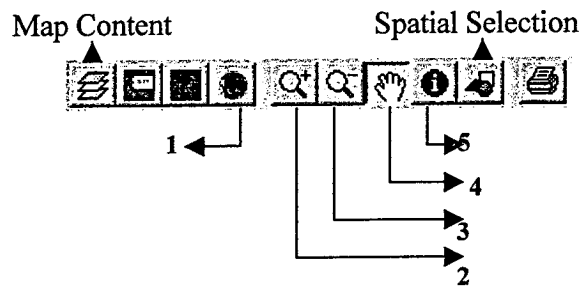


Figure B.3: TSP Toolbar

1. Full Extent: change the map extent to the full view spanned by all the extents in the layer.
2. Zoom Out: used to zoom in by clicking on a user-specified area.
3. Zoom In: used to zoom in by box to a user-specified area.
4. Pan: used to drag left and right.
5. Identify used to get information about a selected feature.

Step2: The next step is to spatially select the cities we want to visit. When we click on the Spatial Selection button in the toolbar, the Spatial Selection form gets displayed:

The image shows a dialog box titled "Spatial selection". It has three main sections for configuration:

- 1. Choose a layer for selecting features:** A dropdown menu is set to "capitals". Below it is a small square symbol with the instruction "Click on the symbol to change how selected features are".
- 2. Choose a shape type for cursor selection or another layer to select against:** A dropdown menu is set to "rectangle : shape drawn with the cursor".
- 3. Choose a method for spatial selection:** A dropdown menu is set to "Shape & feature boundaries overlap".

At the bottom of the dialog are two buttons: "Close" and "Clear selection".

Figure B.4: Spatial Selection Form

The Spatial Selection form allows us to specify a layer for selecting features (e.g. city layer), what type of search feature (e.g. a rectangular shape), and one of the spatial search methods (e.g. shapes and feature boundaries overlap). Move around the map with the tools available to locate the city area you want then select the cities with the mouse. The selected cities will be highlighted yellow as shown in the figure below:

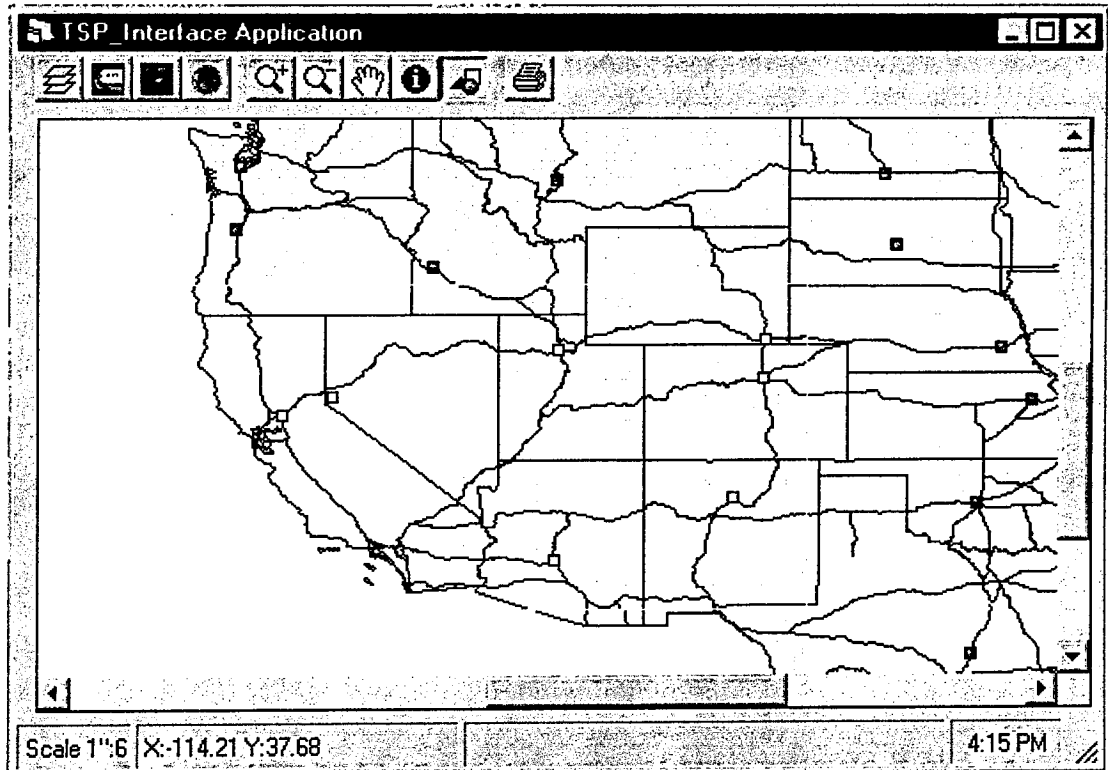


Figure B.5: The Selection Cities Highlighted in Yellow Color

Step3:

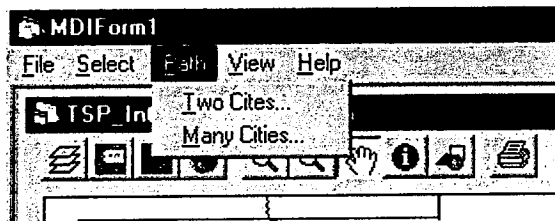


Figure B.6: Path Menu Form

Click on the Many Cities sub-menu as shown in figure above and the form will appear. To get a better understanding of how this form looks like, refer to the figure below

City Names

List of State Names: 7

Number of selected cites are: Cheyenne, Salt Lake City, Denver, Carson City, Sacramento, Santa Fe, Phoenix

Copy Matrix to File

Copy to File "TSPFILE"

Path Graphically

Show Path Graphically

City Distance

Matrix representing the distances between cities:

		Cheyenne	Salt Lake Ci	Denver	Carson City	Sacramento	Santa Fe	Phoenix
	Cheyenne	0	8	1	19	19	8	
	Salt Lake Ci	8	0	9	11	11	16	
	Denver	1	9	0	23	23	7	
	Carson City	19	11	23	0	99	29	
	Sacramento	19	11	23	99	0	29	
	Santa Fe	8	16	7	29	29	0	
	Phoenix	16	99	14	12	12	8	

Close OK

Figure B.7: Multi Cities Form

Step 4: Click on the button Copy to TSPFILE shown in figure B.6 to copy the TSP matrix with corresponds to the distance between the cites to an output file called TSPFILE. Move to the Mathematica software and run the TSP Engine application by clicking on the Evaluate Notebook (refer to figure below). After running the Mathematica TSP Engine application the output results that hold the salesman tour is displayed in an output file called TSPENGINE to be displayed in the TSP Interface application.

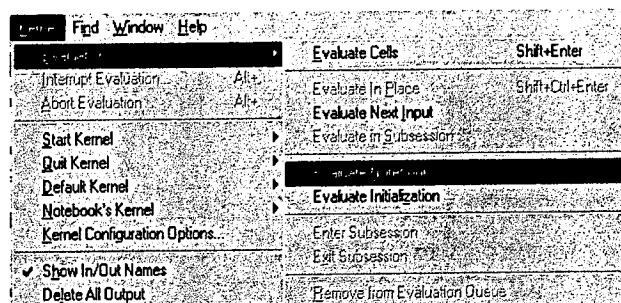


Figure B.8: Running Mathematica TSP Application

Step5: Return to the TSP Interface application and click on ‘Show Path Geographically’ as shown in figure B.6. The result is depicted from the TSPENGINE and visually displayed on the map canvas highlighted in a red color. As a result, an attractive tour is shown and allowing the salesman to visual the efficient path and be able to benefit from it in a real world bases. An example showing the salesman path is shown in the figure below.

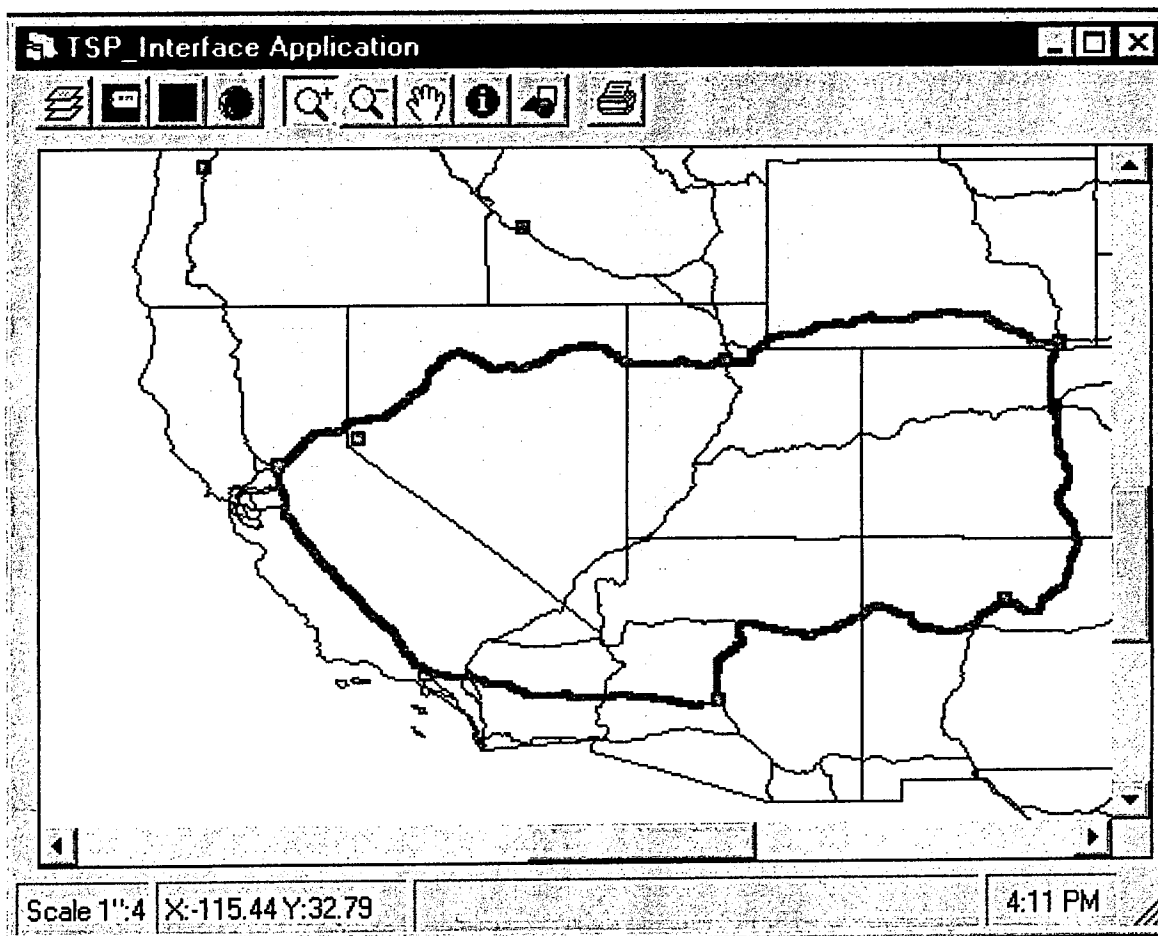


Figure B.9: The Traveling Salesman Resultant Tour

REFERENCES

- [1] E. Aarts and H. Stehouwer, "Neural Networks and the Travelling Salesman Problem," in Proc. Int. Conf. On Artificial Neural Networks, 1993.
- [2] F. Chedid, "On the Hybrid Neural Network Model for Solving Optimization Problems," Lecture Notes in Computer Science, M.Deza, R. Euler and I. Manoussakis (EDS.), vol. 1120, Springer_Verlay (1996) 182-193.
- [3] N. Christofides, "The Travelling Salesman Problem," Report 88-11, Department of Management Science, Imperial College, London.
- [4] A. Cichocki and R. Unbehauen, "Neural Networks for optimization and Signal Processing," John Wiley & Sons, Ch. 9: Neural Network for Combinatorial Optimization Problem, pp. 480-505.
- [5] R. Durbin, R. Szeliski and A. Yuille, "An Analysis of the Elastic Net Approach to the Travelling Salesman Problem," Neural Computation, Vol. 1,1994.
- [6] L. Fiecher, "A Parallel Tabu Search Algorithm for Large Travelling Salesman Problem," Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science, 1994.
- [7] L. Fu, "Neural Networks in Computer Intelligence," McGraw-Hill, Ch: 2, pp. 33-55 and Ch: 4, 80-100.
- [8] M. Garey and D.S. Johnson, *Computers and Intractability: A Guide to Theory of NP-completeness*. San Francisco, CA: Freeman, 1979.
- [9] J. Hopfield and D.W. Tank, "Neural Computation of Decisions in Optimization Problems," Biological Cybernetics, vol. 52, pp. 1-25, 1985.
- [10] E. Lawler "Combinatorial Optimization: Networks and Matroids". Holt, Rinehart and Winston, New York, 1976.
- [11] A. Le Gall and V. Zissimopoulos, "Extended Hopfield Models for Combinatorial Optimization ", IEEE Transactions on Neural Network Vol. 10, No.1, January 1999.
- [12] C. Peterson and B. Soderberg, "A New Method for Mapping Optimization Problems onto Neural Network," Int. J. Neural Syst., vol. 1, pp.10-25, 1989.
- [13] J. Raggett and W. Bains, "Artificial Intelligence from A to Z," Chapman & Hall.
- [14] W. Stornetta, B. Huberman, "An Improved Three-Layer Back-Propagation Algorithm," IEEE International Conference on Neural Networks, San Diego, 1987.
- [15] K. Sun and H. Fu, "A Hybrid Neural Network Model for Solving optimization Problems," IEEE Trans. On Computers, Vol.42, No.2, pp.218-227, Feb. 1993.

- [16] "K. Sun and H. Fu, "Solving Satisfiability Problems with Neural Networks," in Proc. IEEE Region 10 Conf. Comput. and Commun. Syst., vol. 1, Hong Kong, Sept. 24 -27, 1990, pp. 17-22.
- [17] N. Ulder, E. Aarts, H. Bandelt, P. van Laarhoven and E. Pesch, "*Genetic Local Search Algorithm for Travelling Salesman Problem,*" in Parallel Problem Solving from Nature I, pp. 109-116, Springer-Verlag, 1990.
- [18] A. Van Ooyen and B. Nienhuis, "*Increasing the learning rate of the Backpropagation Method,*" Netherlands Institute for Brain Research.
- [19] T. Volgnant, R. Jonker, "*A Branch and Bound Algorithm for the Symmetric Travelling Salesman Problem Based on the 1-tree Relaxation,*" European Journal of Operational Research (1982).
- [20] P. Yip and Y. Pao, "*Combinatorial Optimization with Use of Guided Evolutionary Simulated Annealing*", IEEE Transactions on Neural Network, Vol. 6, No.2, March 1995.

The END

...

•
•
•
•
•
•
•
•
•