This is to certify that the

dissertation entitled

A MASSIVELY PARALLEL ARCHITECTURE DESIGN
FOR PATH PLANNING APPLICATIONS

presented by

Lyle Amos Reibling

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in Computer Science

G C Stockman
Major professor

Date 15 May 1992

A MASSIVELY PARALLEL ARCHITECTURE DESIGN FOR PATH PLANNING APPLICATIONS

By

Lyle Amos Reibling

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science

1992

ABSTRACT

A MASSIVELY PARALLEL ARCHITECTURE DESIGN
FOR PATH PLANNING APPLICATIONS

By

Lyle Amos Reibling

This research has investigated computer system design issues for massively parallel architectures applied to path planning in two-dimensional risk fields.

Physical phenomena were found to be useful in modeling shortest path planning problems. These phenomena provide analogies in nature which form the basis for the models in path planning. The electrostatic model in field theory describes the distribution of current flow through a nonuniform conducting media, such as a plate of nonhomogeneous resistive material. The paths of current flow are used as the analogy in nature to path planning by associating the resistivity of the media with the cost function of the path planning problem.

The distribution of current flow for electrostatic fields is described by the Laplacian partial differential equation. An artificial neural network was designed to solve the Laplacian equation for nonhomogeneous media and was used in a massively parallel architecture to compute minimal cost and alternative paths. The scalar field for the nonhomogeneous media is found by computing a finite difference approximation to the partial differential equation. The artificial neural network computes this field. Path solutions are computed from a vector field which is

orthogonal to equipotential contours of the scalar field.

Connectivity of the massively parallel architecture was examined for its influence on the convergence rate of the neural network approximation. Experimental investigation of this technique has shown promising results. The solutions generated by the architectures have been checked against known admissible algorithm results. A simulation of an analog implementation of the artificial neural network with 3,600 neurons resulted in projected real-time convergence of 18 milliseconds.

A significant result of this research was the discovery of an architecture design paradigm for massively parallel architectures. This paradigm is called natural parallelism.

DEDICATION

To my wife, Anne, and our children: David, Rebekah, Rachel, and Daniel.

ACKNOWLEDGEMENTS

I am deeply indebted to my advisor, Dr. George Stockman, whose patience with me throughout this endeavor has been beyond measure. His guidance has been most crucial to the successful understanding and completion of this research. I also have deep appreciation for Dr. Michael Olinger, who has been both a friend and mentor with his intellectual stimulation and insight into this research.

I wish to also extend my thanks to my employer, Smiths Industries Aerospace & Defense Systems, Inc. Through its Doctoral Assistance Program, company resources have been provided to support this research. I am indebted to my managers for their support and encouragement; in particular, the late Don Gray who encouraged me at the very onset. Others who have been instrumental through the years were Richard Corey and Michael Olinger. Without a doubt, the most critical help I received was from the Chief Librarian at Smiths Industries, Scott Brackett. I have spent many hours with Scott talking about the research, conducting literature searches, reviewing documents, and explaining my investigation. I seriously doubt this research could have been undertaken without the excellent skill and interest of Scott and his library staff.

I have been blessed to have close friends who have stood by me and exhorted me to continue under the most trying circumstances. To Bob, Pamela, and Dave, I say thanks.

Table of Contents

List of Tables

List of Figures

List of Symbols

ϕ - Potential field (scalar)

ρ - Resistivity

σ - Conductivity

$\boldsymbol{j}$ - Current density field (vector)

t - Time

$\boldsymbol{E}$ - Electric field (vector)

R - Resistance

V - Voltage

W - Synaptic weight

u - Neuron input voltage

v - Neuron output voltage

Chapter 1

Introduction

1.1 Nature of the Problem

Many applications can be described as problems in combinatorial search. These problems require searching[3, 50, 72] through the states of a problem by tracing paths from an initial state to a final state along the allowable transitions between states. The purpose of the search is to find the best path according to some objective function which is defined over the states of the problem[57]. The objective function defines the criterion for the search. The best path is the one which minimizes the objective function when evaluated over the problem states on the path. The problem states are typically defined through the use of a cost function. The cost function is a hypersurface in the multidimensional state space of the problem. Figure 1.1 is an example of a problem space showing the risk field of a cost function. The risk is shown by the use of contour lines which describe the elevation of the function's values. The best path is the path $\mathcal{P}$ which minimizes the integrated risk along the path between the start and goal states of

the problem space. This is defined by

$$\min \int_{\mathcal{P}} C(s)\, ds \tag{1.1}$$

where $C(s)$ is the cost function which defines the risk along some path $\mathcal{P}$ which begins at the start state and ends at the goal state of the problem. The best path is also shown in the Figure 1.1 as a dashed line which is the optimal path between its end points for the cost function.

1.1.1 Search Requirements Make Demands on Architectures

The states of the problem and their transitions can be represented as a directed graph (digraph). As problems grow in size and computational complexity, their associated digraph becomes more complex. This complexity is due to the number of nodes and interconnections between the nodes of the digraph. Very complex problems make severe computing demands for search algorithms on serial processing architectures[53, 54].

A general purpose sequential processor, or even a group of multiprocessors, may not be effective in solving traditional algorithms for these demanding problems. This is because of the enormous number of possible solutions to these problems. The search problem is a very complex issue for algorithm design. In general, the complexity of the search problem is exponential. As will be shown in Chapter 2, the search problem can be represented by a tree. A solution path in the search problem is then found by tracing the paths in the search tree from its root to a terminal leaf node. If there are B successors to a state in the search

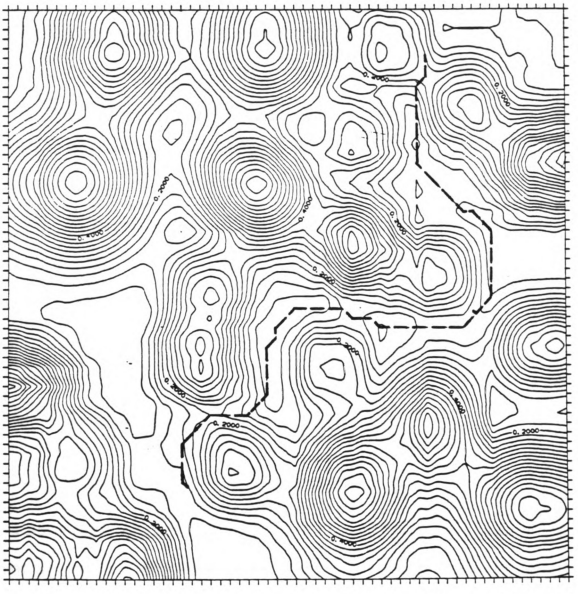

Figure 1.1: *Example Cost Function Showing Risk Values Using Contour Lines*

problem, then the tree has a branching factor of B. Therefore, each new level in the search tree will grow by B new nodes for each node in the previous level of the tree. This growth in the search tree is often referred to as combinatorial explosion. The general search problem with unrestricted state operators will have a worst case path which includes all states of the search space. Thus, the total number of solutions is $O(B^n)$, where n is the number of states in the search problem. This exponential number of solutions to the fundamental search process is why it is an NP-complete problem, even though there are only n state in the search space.

The Von Neumann architecture, a serial processing architecture, requires algorithms which are defined as serial instruction streams. Even multiprocessors, while simultaneously processing in parallel as a group, are processing serially as individual processors[19]. To meet the requirements of larger, more complex problems, the processing architecture and algorithms need to be more closely matched to the nature of the application problem[62].

1.1.2 Massively Parallel Architecture Needs

Advances in integrated circuit and optical computing technology have made implementation of massively parallel computer architectures feasible. These architectures are more complex than multiprocessor networks, often inspired by the structure of the brain as in artificial neural networks[31, 41]. Discovering properties of computation in these new architectures is a contemporary issue in computer science. Computation is not specified, or programmed, in the usual manner of

a stored program computer; rather, the collection of interconnection strengths between nonlinear parallel processing elements of these networks determine the computation of the processing system[29].

As artificial neural network technology is developed to implement solutions to search problems, obstacles need to be overcome in designing realizable hardware. One such obstacle is the need for a method to design the architecture of the artificial neural network. Several questions regarding a neural network need to be answered before an architecture can be defined to solve the problem: How many interconnections are required between the neurons? What pattern of interconnection is needed? How does the pattern of interconnections realize the numerical solution of the problem? What should be the definition of the interconnection strengths? Does the number of interconnections influence the feasibility of the hardware realization, the convergence and quality of the solution, or the speed of the network? What function does the neuron need to compute? How does the function realize the numerical solution to the problem?

While the previous questions describe the architecture needs, they do not address any process to use in designing architectures. Other questions illustrate needs in the design process. How does the development of a particular architecture relate to the algorithm solution? Is the algorithm needed first, and then the architecture definition follows, or does it make more sense the other way around? Is there an interaction between these needs? Is there some underlying mathematical model of the problem which will solve this algorithm/architecture interrelationship? If so, is the mathematical model a special one for each problem,

or is there some general principle or model which will solve the problem? How can the mathematical model be mapped into the artificial neural network? What class of problems can the resulting design compute?

Questions such as these have motivated this research. The research was directed to answer these questions, although not necessarily in the order given above. Taken as a whole, these questions aid in revealing the issues that need to be understood in developing a design methodology for search architectures using massive parallelism. These design issues are taken up in the subsequent chapters of this dissertation.

1.1.3 The Architecture Design Challenge

The traditional approach to algorithm conceptualization is influenced by the designer's own sequential thinking processes and understanding of the implementation of algorithms on serial processors. Parallel architectures developed from multiprocessor networks have incorporated serial processors[19, 33]. The design challenge of these networks address the control and data partitioning needed to map serial algorithms onto multiple communicating serial processors. This is not an effective method to develop massively parallel computing architectures. The enormous number of processor nodes and interconnections is beyond the designer's ability to partition algorithms due to the explosive number of partition combinations. Massive parallelism is characterized by collective computation and as such a new approach is needed for thinking about how the roles of application, algorithm, and architecture are interrelated in a design. This relationship

creates a design challenge for determining interdependence models between the *application* problem to be solved, the *algorithm* to solve the problem, and the massively parallel *architecture* which computes the solution. A simpler paradigm for architecture design must address these three related issues.

1.2 Objective of the Research

1.2.1 Research Motivation

The motivation of this research is the investigation of computer architectures and algorithms characterized by massive parallelism which have computational properties that yield the solution to combinatorial search problems[30] with application to path planning[6, 13, 20, 55]. The central thesis of this research is that a massively parallel computing architecture can be defined in which the computational units correspond identically to the explicit problem states and exhibits convergence to values which represent a solution to the particular problem. Questions fundamental to investigation of this central thesis are:

1. What is the theoretical basis for a model of path planning in the central thesis?

2. What is the computational definition of the massively parallel architecture in the central thesis?

3. How does the amount of interconnection among computational units in the massively parallel architecture influence its accuracy and convergence?

4. What is the performance of the massively parallel architecture?

5. What, if any, is the fundamental principle which unifies the distinct, yet interrelated roles of architecture, algorithm, and application in the definition of the massively parallel architecture of the central thesis?

1.2.2 Research Method

In this research, analogies in nature are used to design massively parallel architectures for path planning. These architectures are analyzed for their effectiveness in computing solutions to the search problem. The tool for this research is the simulation of the architecture models. This provided a method to study the topological properties of the architectures for their influence on solution convergence and performance characteristics through experimental verification. The solutions generated by the architecture simulations were checked against known admissible algorithm results.

1.2.3 Massively Parallel Architecture Approach

There needs to be an understanding of how to effectively determine the architectural definitions of interconnection strengths and processing element functions for performing desired computations. This dissertation describes the problem, and a solution methodology to design massively parallel architectures which use new parallel models for path planning applications.

This new approach is called the natural parallelism paradigm. This paradigm uses analogies in nature to develop massively parallel architecture designs. This research has shown that the paradigm is an effective methodology for designing

massively parallel architectures to solve combinatorial search problems such as path planning.

1.3 Remaining Organization of the Dissertation

Chapter 2 is a review of the literature which is background to this research. Traditional serial search algorithms for this problem are discussed. Multiprocessor architectures are discussed and developments in massively parallel architectures, particularly artificial neural networks, are discussed. Finally, the combinatorial search problem used in path planning is described.

Chapter 3 and Chapter 4 investigate the development of the parallel architecture for path planning. In Chapter 3, the mathematical model of a new parallel approach is developed. The parallel architecture design for the model is also described. In Chapter 4, the results of the architecture design for the path planning application are presented.

In Chapter 5, the key discovery of the new design paradigm of natural parallelism for massively parallel architectures is described. The natural parallelism concept, motivated from nature, is at the heart of the paradigm to enable mapping algorithms onto these architectures. The significance of the paradigm is discussed. The path planning results are discussed and its performance is analyzed. Chapter 6 is a summary of the research. The dissertation concludes with recommendations for further research.

Chapter 2

Searching, Parallelism, and the Path Planning Problem

This research concerns a new search technique and its application to path planning using massively parallel architectures. In this research, architectures are investigated for solving the parallel implementation of path planning. To develop an understanding of the background to this research, the interrelated roles of algorithm, architecture, and application in designing solutions are reviewed as distinct topics. The topics reviewed begin with a general discussion of search algorithms. The search algorithm discussion is be followed by some background discussion on traditional parallelism of multiprocessor computer architectures. Finally, the problem of path planning for aircraft trajectory generation is described.

2.1 Search Algorithms

Conventional serial-processing search algorithms have addressed adversarial game tree search, path search, and shortest path search problems. Much research regarding combinatorial search algorithms have focused on traditional algorithm development and analysis for sequential processors, or distributed

multiprocessor architectures. Many problems are solved using search[3, 50]. Winston[72] describes search as exploring alternatives. Such an exploration deals with situations in which choices are made, leading to new situations and more choices. This is a very common scenario in artificial intelligence endeavors.

An optimal solution to a problem is obtained when the search results in the best possible answer. But in many cases, the best solution may be too costly or prohibitive to find, so reasonable solutions are suitable alternatives. Good approximations to optimal solutions often serve well as satisfactory results. People are an example of problem solvers which are satisficers, not optimizers. A satisficing solution is often more appropriate because worst case situations requiring optimal solutions rarely arise in the real world.

Among the many uses of search algorithms, two major classes are adversarial board game problems and path search problems. The adversarial scenario looks for solutions leading to a winning outcome for the game. The path search problem finds routes through a problem space between initial conditions and problem solutions. This latter search problem can also require finding the best path which meets some defined criterion.

To generalize search algorithms, a method is needed to provide a mapping of the particular problem into a uniform description. This description is used by the search algorithm to solve the problem, and is called the problem space representation.

2.1.1 Problem Space Representation

The problem space representation is often a directed graph. A graph is a set of nodes with a set of edges which connect pairs of nodes. If an edge in the graph is not bidirectional, then it is called an arc. A graph which contains arcs is called a directed graph (digraph). When a graph contains an arc from a node n_i to node n_j, the node n_j is called a successor of node n_i, and conversely the parent of n_j is n_i. Arcs can be weighted with a value which represents the incremental cost from a node to its successor.

In many situations a particular directed graph called a search tree[72] is used. The search process is a traversal of this directed graph where each node of the directed graph is used to represent a state of the problem[63]. The arcs of the digraph define the state transition relationship between the states connected by the arc.

The digraph can be represented either explicitly or implicitly. In the former representation, the digraph is defined by enumerating the entire set of states. The arcs of the digraph are shown pointing from one state to another state. In the latter representation, the digraph is defined by establishing an initial set of states and operations which transforms one state into another state.

A search tree can be constructed from a digraph by expanding the possible successors to each node of the graph, beginning with some particular node as the root of the tree. Trees are expressed in geneological terms such as parent and child. A node with no parent is the root node. A node with no children is a

terminal node. Expanding a node means defining its children.

A search algorithm is a procedure for finding a path from the root node of a tree or graph (the start state) to a goal node (leaf). The first consideration of any search algorithm begins with the representation of the search space. This choice is either an explicit representation, or an implicit one. For example, the explicit representation in Figure 2.1 can be represented implicitly as

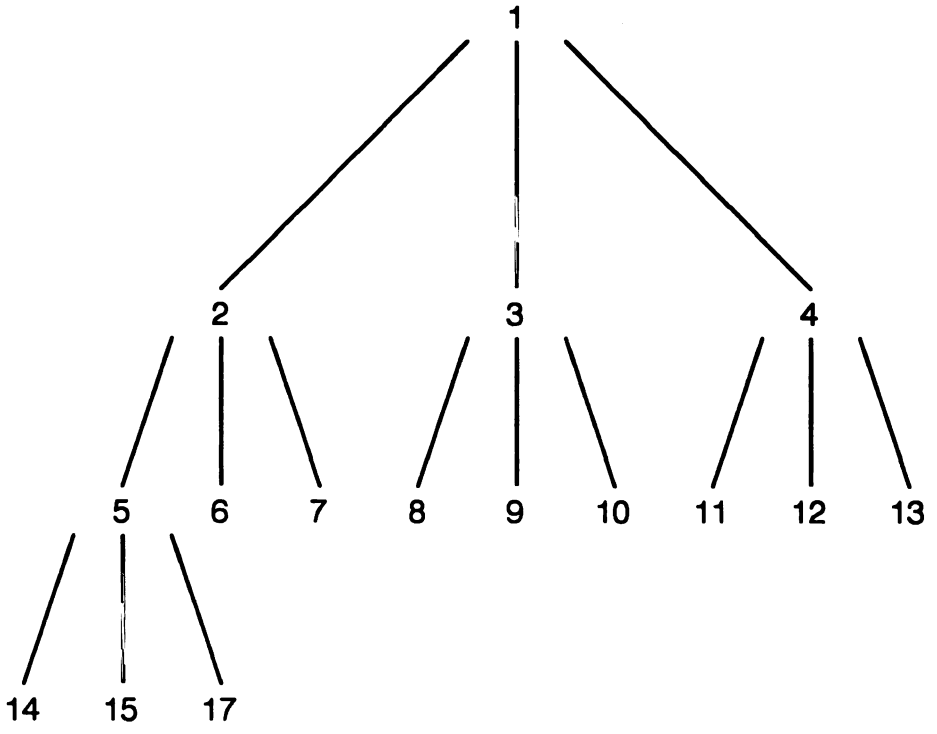

Figure 2.1: *Explicit Search Tree Representation Example for a Search Space*

$$\text{ROOT} = 1$$

$$\text{SUCC}(n) = \{m | m = 3n - 1, 3n, 3n + 1\}, 1 \leq n \leq 5$$

Each node has only a finite number of successors (which excludes an infinite-breadth tree). The depth of the search tree, however, may be infinite. Arcs of the search tree often have costs associated with them.

State Spaces

Problems may be defined as state spaces. Searching then occurs in an environment called the problem space, which has a set of states and operators[57]. These states describe the problem. An instance of the problem is defined by including a particular initial and final state with the problem states. Then problem solving becomes a state space search problem[63]. State space search is a problem solving method where the problem is encoded into a state representation, and the problem is to move through various configuration of the state space. This is done by combining the allowable move operations of the problem with search techniques. The searching task solves the problem by finding the sequence of operators which converts the initial state into the final state. This sequence is the problem solution, and is described as a path through the problem space from the initial state to the final state. In problem spaces which correspond to physical space, the resulting path represents a physical route.

2.1.2 Game Tree Search

An example of searching as exploration is common in board games. Adversarial choices are made to lead to winning (and losing) situations by the players. Some of the search algorithms used for this kind of problem are minimax and alpha-beta pruning.

The search tree has been used in computer programs as the data structure for modeling chess and other games. In adversarial board games the search tree

nodes correspond to the board configurations[72]. These board configurations define the game state. This tree is used to represent the possible continuation of the game from the current situation of the game via the moves of the game. The branches of the search tree transform the situation which corresponds to moves of the games. Since the tree represents the possible game continuation from the current situation, it is called a lookahead tree. The root of the tree is the current game configuration of the game pieces on the board. The successor nodes represent the new game configurations resulting from the choice of a single legal move of the game. The leaves of the tree represent the possible game configurations after a number of moves which correspond to the height of the tree.

Alpha nodes of the tree correspond to board configurations from which the program (the maximizer) selects the next move, and beta nodes correspond to the board configurations from which the opponent (the minimizer) selects a move. Each tree level thus alternatively consists of alpha and beta moves, with the root being an alpha node.

The leaves of the search tree are examined to determine a value of the resulting board configuration, since the leaves of the tree may not represent game termination. This value of the leaf is used as a numerical estimate of the relative strengths for the players in the leaf's board configuration. An analyzer converts the board configuration judgments into a single number for the quality of the situation. This is called the static evaluation score. A positive score favors one player called the maximizer, and a negative score favors the minimizing player. Evenly

matched board configurations have a value of zero. This numerical value of the static evaluation score is calculated using a static evaluation function (SEF).

The minimax backup procedure is used to propagate these values up to the root of the tree. For an alpha node, the greatest value in its successor nodes is selected, which represents the program making that move because it will be the best new board position. The least value is selected for the beta nodes, because this represents the move of the opponent, who is assumed to be hostile and perfect and, therefore, will make moves which are the worst new board position for the program. Thus, alpha node values are the maximum of its successors' values and beta node values are the minimum of its successors' values. The value at the root node indicates the maximum value available for the search tree.

Alpha-beta search is a minimax procedure for pruning the growth of the game tree[72]. It generates the successor nodes for the limited depth search tree in a depth-first order, and combines the development of the lookahead tree and the leaf node evaluations with the minimax backup procedure. This allows the omission of the development of subtrees by comparing the backed-up values of nodes and cutting off their generation when they will not contribute to improvement in the minimax values.

2.1.3 Path Search

Another use of search for exploring alternatives is finding the path from a start node to a goal node in a problem space. A path from node n_{i_1} to node n_{i_k} is a sequence of nodes $n_{i_1}, n_{i_2}, \ldots, n_{i_k}$, where every node n_{i_j} is the successor of the

node $n_{i_{j-1}}$, for $j \in [2, k]$. A node n_j is a descendant of the node n_i if there exists a path from n_i to n_j.

When the length (or cost) of the path is not important, simple procedures such as depth-first search, breadth-first search, hill climbing, beam search, and best-first search will suffice to find a solution path[72]. These procedures are easy to implement, but usually find only sub-optimal paths.

In depth-first search, one alternative is selected at every node. The search then works forward from the selected node. The remaining alternatives are ignored at first. As needed, the alternatives are selected in the order they were generated from this parent node expansion. The algorithm begins by selecting the root node. A successor of the root node is then selected. If the successor meets the goal condition, the algorithm stops. Otherwise a successor of the successor is selected, and the evaluation is continued. If there are no further successors, the previous available successor is selected and the search continues. The algorithm is a very intuitive approach to searching. It is good for small trees with many goal nodes. However, it is not guaranteed to find an optimal solution path, even if one exists.

Hill climbing uses a heuristic measure of the distance to a goal. The most promising alternatives are used first in a depth-first search fashion. The child nodes must be sorted to determine the best alternative. Three problems occur with the hill climbing approach. The foothill problem occurs when there is a secondary peak in the problem giving locally optimum results. The plateau problem occurs in large flat regions because no local improvement occurs. The ridge

problem occurs where none of the directions show improvement and requires more alternatives to detect the local improvement.

Breadth-first search uniformly pushes into the search tree by levels. In breadth-first search, each successor of the selected node is examined against the goal condition. If a successor meets the condition, then the algorithm stops. Otherwise, all successors of the previous successor are examined against the goal condition, and the algorithm continues. This algorithm is very easy to implement. If a solution exists, it is guaranteed to find it. Also, the solution will have a minimum path length (tree height, but not necessarily minimum cost).

Beam search is like breadth-first search by level. It is restricted to move down by only the w best nodes, where w is the width of the beam.

Best-first search extends the best partial path[72] by expanding the best open node. However, this requires sorting all of the open nodes.

Breadth-first search, depth-first search, and best-first search can be generalized to state space search. The states which are contained in the frontier of the search tree (the leaf nodes) are elements of the set called OPEN, and the interior nodes of the search tree are elements of the set called CLOSED. Consider an OPEN set which contains the search frontier as a specific data structure in the context of the following state-space search algorithm:

1. Put ROOT in OPEN

2. If OPEN = Φ, then STOP-FAILURE

3. Remove the "best" state, n, from OPEN, and place into CLOSED. If $n \in$

GOAL, STOP-SUCCESS

4. Generate successors of n and place them on OPEN

5. Go to STEP 2

If the OPEN set is implemented as a stack data structure, the above state-space search algorithm is depth-first search. Implementing OPEN as a queue results in the breadth-first search algorithm. In step 3 the removal of the "best" state n is accomplished by the remove function for the stack or queue. In order to actually remove the "best" state as in best-first search, the contents of the queue need to be ordered by some evaluation function. Thus, if the data structure is implemented as a prioritized queue ordered by the evaluation function, the resulting algorithm is best-first search.

Each algorithm has its advantages in certain situations[72]. Depth-first search is good when blind alleys are not too deep. Breadth-first search is good with few alternatives (small branching factor). Hill climbing is good when there is a natural measure for the goal distance, and the likely choice is a good one. Beam search is good when a natural measure exists and the likely partial paths are shallow. Best-first is good when there is a natural measure and good paths look bad at shallow levels.

2.1.4 Shortest Path Search

A third class of search algorithms are used when the best path is needed. These algorithms are more complicated, because the cost of traversing the path

is important for determining the best path[72]. Examples of algorithms used for shortest path search are British Museum search, branch and bound search, discrete dynamic programming, A^* search, and simulated annealing.

The British Museum procedure is to find all possible paths, and then select the best one. This is like breadth or depth-first search, but without stopping on a goal. Conceptually, it is a very simple approach. But it obviously suffers from combinatorial explosion, and is not very useful for other than simple problems.

Branch and bound search maintains a set of partially developed paths. It proceeds by extending the shortest incomplete path. It terminates when the shortest incomplete path is longer than the shortest complete path. Branch and bound is good for big search trees, and when bad paths turn very bad fast (at shallow levels)[72].

An improvement in these algorithms is to discard redundant paths. This is done by deleting paths to common nodes which have a larger cost than the minimum path to the same node. This technique underlies the dynamic programming principle[72]. In looking for the best path from start to goal, all paths from the start to any intermediate node except the minimum can be ignored. Bellman[4] introduced the dynamic programming concept for decision making processes. Dreyfus and Law[14] have extended this dynamic programming *principle of optimality* to unconstrained shortest path search:

> The minimum cost path from A to B has the property that, whatever the initial cost at A, the remaining path to B, starting from the next node after A, must be the minimum cost path from that node to B.

Dynamic programming is good when there are many paths to common nodes[72], such as with problem graphs defined on grids, where each grid point is a state node in the problem space.

Underestimates can be used to improve the efficiency of these search procedures by guessing the distance remaining to the goal. Several search ideas come together in the A^* search algorithm. It is branch and bound search together with a distance estimate and the dynamic programming principle[72].

Additional notation is required for the following discussion of the A^* search algorithm. Functions are defined that describe costs associated with the paths. When it is desirable to assign a cost to an arc to represent the cost of applying the transition represented by the arc, the notation $c(n_i, n_j)$ is used to represent the cost of the arc from n_i to n_j[1]. The least path cost from the start node, s, to any arbitrary node, n, is denoted as $g^*(n)$. A path from s to n which had this cost is an optimal path from s to n. The least cost from any arbitrary node n to some goal node g is denoted as $h^*(n)$. The function

$$f^*(n) = g^*(n) + h^*(n) \tag{2.1}$$

then represents the least cost from the start node s to the goal node g, with the path constrained to include the node n. This path through n however, may not necessarily be the minimum cost path from s to g. In the following algorithms leading to A^*, choices are made about which nodes to expand based upon some estimate of the "worth" of the node. $g(n)$ is used to estimate $g^*(n)$ and $h(n)$ is

[1]Costs may be associated with the nodes. In such a case, the cost $c(n_i)$ at node n_i can be assigned to the arc of each successor node n_{i_j} of n_i, where the resulting cost for the arc is $c(n_i, n_{i_j}) = c(n_i) \ \forall \ n_{i_j} \in \text{SUCCESSOR}(n_i)$

used as an estimate of $h^*(n)$. Thus, the evaluation function used in the choice of the next node to expand is

$$f(n) = g(n) + h(n) \tag{2.2}$$

Every node n which has been reached during the search belongs to one of two sets. $n \in$ CLOSED if n has been examined and is no longer in the search frontier. $n \in$ OPEN if n is a candidate for immediate exploration due to being in the search frontier.

A^* is a graph search with knowledge (heuristic estimate). In A^*, $h(n)$ is an underestimate of the cost from n to a goal node. Rather than optimizing $g(n)$, A^* will optimize[2] the function $f(n) = g(n) + h(n)$. A good estimate, h, is a prerequisite for a good A^*. Goodness is dependent upon the admissibility of the estimate and efficiency in evaluating the estimate.

The A^* search algorithm is as follows:

1. Initialize by placing ROOT in OPEN, g(ROOT) = 0, calculate h(ROOT), and f(ROOT) = h(ROOT).

2. Select N such that $f(N) = \min\{f(n) | n \in \text{OPEN}\}$

   Test if N is a goal. If yes, then STOP-SUCCESS with PATH(ROOT,N) the minimum cost solution, cost = $g(N)$.

   If OPEN = Φ, STOP-FAILURE.

   Test if there are any more nodes, N, such that $f(N) = \min\{f(n) | n \in \text{OPEN}\}$

[2]For any node n which is on the optimal path, A^* will optimize $g(n)$.

If there are, check to determine if any of the nodes satisfy the goal conditions and STOP-SUCCESS, otherwise choose one randomly to be the next N.

3. Put N in CLOSED.

   For all n_i such that n_i = successor(N), calculate $g'(n_i) = g(N) + c(N, n_i)$

   Test if n_i was previously opened and has had previous $g(n_i) \leq g'(n_i)$. If so, then ignore n_i. Otherwise, put n_i in OPEN and set $g(n_i) = g'(n_i)$ and calculate $f(n_i) = g(n_i) + h(n_i)$. Set path pointer from N to n_i.

   Go to STEP 2

These are some of the algorithms used for computing shortest path solutions on sequential processors. They provide a background of the search problem and historical approaches to finding shortest paths. These algorithms were all deterministic. In the next section, a non-deterministic procedure is described which has been used for shortest path search.

2.1.5 Simulated Annealing

Shortest path searching can be considered a combinatorial optimization procedure. In combinatorial optimization, the function to minimize is called the objective function[56]. The constraints on the optimization are the set of feasible options in the solution to the problem. A problem is combinatorial when the configuration elements are finite, or a countably infinite set.

The annealing concept has been posed as an applicable general routine to use on combinatorial optimization problems[56]. Annealing is a physical process

in condensed matter physics[70]. A solid is placed in a heat bath and heated to increase its temperature to a maximum value where its particles are then randomly arranged into a liquid. The liquid is then cooled by slowly lowering the temperature of the heat bath. As it cools, the liquid's particles are arranged into low energy ground states of a corresponding lattice. This process requires a sufficiently high temperature and slow cooling. Thermal equilibrium is reached at each temperature with the energy states arranged according to a Boltzmann distribution. As the temperature decreases, this distribution concentrates on the low energy states.

In simulated annealing[35], minimizing the cost function of an optimization problem is analogous to the slow cooling of a solid reaching a low energy state[70]. The cost function is substituted for energy and executing the Metropolis algorithm is analogous to the slowly decreasing temperature. Simulated annealing is also known as probabilistic hill climbing[70].

The Metropolis algorithm is a simulated evolution to thermal equilibrium of a solid for a given fixed temperature, T. It is a Monte Carlo method which generates a state sequence of a solid. A small random perturbation of a particle is applied. This new state results in a difference in the system's energy, ΔE. If $\Delta E < 0$, the process is continued for lower energy states. If $\Delta E \geq 0$, the new state is accepted with the probability $e^{-\Delta E/k_B T}$. This acceptance rule is called the Metropolis criterion[70].

To use this on a combinatorial optimization problem, a sequence of configurations of the problem space are generated. These configurations are like the

states of the solid. The cost function is like energy, and the control parameter is like temperature. The simulated annealing algorithm is a sequence of runs of the Metropolis algorithm which is evaluated at sequences of decreasing values of the control parameter.

To use this approach, a unique representation of the state configurations needs to be defined. This representation must proscribe a procedure for measuring the quality of a configuration, and there must be a neighbor relation for the configurations. A difficulty with simulated annealing is that there is no definitive answer for how to formulate and control the annealing process.

This review has provided a look at several sequential path searching algorithms and procedures which have been developed for execution on sequential processors. Because of their intended use, they have an attribute which is not only sequential but of singular activity. This attribute can be seen in cognitive psychology terms as the focus of attention and the serialization of the human thought process in problem solving. The tie between the serial thought process and resulting serial processing designs must be acknowledged. New massively parallel architecture design approaches will be more viable if they recognize this traditional relationship in existing architectures.

2.2 Computer Architectures and Parallelism

Parallel processing is a system consisting of at least two processing elements which are capable of simultaneous operation[19]. Multiprocessing has focused on integrating multiple functional units into a parallel processing system. Central

processing unit concurrency is obtained by either replicating functional units for concurrent operations on different problems as in a multiprocessor, or by pipelining, which performs operations incrementally on a stream of data at stations within the pipeline.

Using a definition of processing as the application of functions to data, or operations performed upon data (memory), the distribution of processing means that this processing is applied to multiple data simultaneously (either synchronously or asynchronously) in a non-serial fashion. This is to be distinguished from centralized processing where all data operations occur in a single processor, which results in serialized computation.

In their definitions for parallel computer structures, Hwang and Briggs[33] consider different configurations of parallel computers. They relate these configurations to a type of parallelism which results from the architecture. Temporal parallelism results from pipelined computers by the use of overlapped computations. Spatial parallelism is achieved from multiple synchronized arithmetic logic units in array processors. And asynchronous parallelism is achieved in multiprocessor systems by a set of interactive processors with shared resources such as memories, databases, etc.

2.2.1 Von Neumann Multiprocessing

The Von Neumann machine is a uniprocessor which had the same basic structure for all early stored program machines[19]. The prevalent structure through 1958 consisted of four main functional units. These units were input/output,

memory, control, and the arithmetic/logic unit (ALU). The weakness of this architecture is routing all of the I/O through the ALU. This made for high hardware costs because of sharing costly buffer registers. In addition, performance was degraded because no processing occurs during I/O. In the mid 1950's direct memory access (DMA) became available. The data flow paths were altered to allow direct memory access for I/O. The control unit still was responsible for I/O operations, however, and some degraded performance continued during I/O. I/O channels then became available. These independent channels separated the access paths to and from memory, and separated control for the I/O operations. Essentially, the I/O channel became a small computer which operated simultaneously (in parallel) with the main processor. The ALU operates free from I/O until interrupted.

A multiprocessor system requires at least two CPU's. The main processor memory is shared and accessible by all of the processors. I/O is sharable, and there is a single integrated operating system which controls the hardware and software of the multiprocessor. It has benefits of reliability because of the multiple units which may be reconfigurable. System performance is improved through the parallel execution of independent tasks. Distribution of memory does not constrain distributed processing which occurs locally. However, as local processing yields results which are shared with other remote processing units, the transmittal time effect of this sharing of data may become evident as a constraint on the global processing behavior.

The system hardware can be organized in several ways. Single access port

units can be time-shared or placed on a common bus. There is no direct connection between function units, but rather a parallel connection to the bus. With a crossbar switch matrix any memory module can be connected to a processor or I/O unit. This is space-division multiplexing. When memory has more than a single access port, a multibus or multiport connection architecture is possible. However, this requires additional conflict resolution logic to arbitrate simultaneous requests for memory access. Pipeline systems perform the same operation repeatedly on several items in a single data stream. Array or vector processors perform the same operation on large collections of data which are related concurrently.

Coupled systems are connected electrically and share common hardware resources. Indirectly coupled systems have no interaction between their programs except at the data set level such as is implemented by a mailbox. A directly coupled system shares addressable storage with high speed channel connections.

2.2.2 VLSI Technology and Massive Parallelism

Very large scale integration (VLSI) architectures are a dominant new technology for computer hardware construction[52]. VLSI offers more promising performance than that of conventional computing hardware[47]. Computer structures can be built by exploiting the properties of VLSI. In 1978, Kung and Leiserson[36] introduced the computational architecture known as the systolic array[24]. The systolic "array" is derived from its similarity to a grid, with each point of the grid as a processor, and the links of the grid consisting of the links between the

processors. This array characteristic of the architecture capitalizes on regular and modular structures which match the computational requirements of the problems they are intended to solve.

Many classical sequential devices fail to yield results in reasonable time. The technology for faster sequential devices now appears to be approaching limits on speed by the laws of physics[18]. The great concurrency of processing elements and memory can be easily implemented in VLSI at many processing sites on a chip. An example is custom VLSI machines for massively parallel AI architectures. Parallel computers with 100,000 to 1,000,000 gates per chip make it feasible to fabricate massively parallel computers.

Fahlman & Hinton[21] explain that exploring the use of massively parallel architectures is an attempt to get around the limitations of conventional symbolic processing. In the massively parallel architecture, the system's collection of permanent knowledge is stored as patterns of interconnection strengths among processing elements. The knowledge directly determines how the processing elements interact rather than residing passively in memory, awaiting examination by a central processing unit.

Comparatively slow (on the order of milliseconds) neuronal computing elements with complex parallel connections give rise to complex, parallel processing with fast processing effects. Time is often the critical resource in many computational problems. Neurons with a computational speed of a few milliseconds can account for complex behavior in a few hundred milliseconds. Entire complex behaviors are accomplished in a few hundred time steps of this computational

system.

The fundamental premise of massive parallelism is that individual neurons do not transmit large amounts of symbolic information, but rather compute by appropriate connections to large numbers of similar computational units. For a given massively parallel network defined by its interconnection topology and individual procedures, there is a need to characterize the class of functions which are computable by the device, and how the functions are computed[18]. This analysis is needed so that these devices can be designed and constructed to provide these functions. A theoretical basis is needed for models of computation in massive parallelism.

Neural networks[16, 17, 41] are a new computer architecture based on the massive parallelism of simple computational units (such as the neurons of the brain). These networks are based upon an abstraction of the understanding of the information processing properties of neurons. The development of neural network technology began in 1943 with the work of McCulloch and Pitts[46]. In their work, they showed how to use the biological neuron as a computational model to compute boolean operations. This type of computing is an alternative to the conventional Von Neumann architecture. A distinctive feature of neural networks is their massive parallelism. These networks have high speed and inherent fault tolerance. Artificial neural networks are a computational paradigm which is motivated by a model of the biological neuron. Hopfield [29] states that

> Parallel analog computation in a network of neurons is thus a *natural* way to organize a nervous system to solve optimization problems.

In developing a massively parallel architecture, the individual computational unit must be defined. This standard model of a *processing unit* loosely corresponds to the information processing model of a neuron. For an artificial neural network to be useful, the network of units requires a means to make decisions or to perform some coherent action. This requirement implies that these networks need to converge to stable states to effect decision making and coherent actions.

The architecture of neural networks is simple processors with a dense arrangement of interconnections. The processing power of a neural network is measured in terms of the number of updates per second of the interconnections, rather than instructions per second for Von Neumann machines. This measure is defined as the connections per second (CPS) of the neural network. Neural networks are characterized by having more interconnections than processing elements[12]. This is in contrast to more traditional parallel processing architectures which have a lower ratio and incorporate Von Neumann machines as the processing elements.

One problem is how to orchestrate the mapping of the designed architecture[2]. This includes the problem of the global control of local elements. How are the local computational rules changed to implement a general algorithm? This processing is coherent and parallel, and must be robust by not exhibiting sensitivity to the failure of a few individual nodes. Interaction of the devices is also an important consideration. The topology of concurrent systems require locality properties to reduce the interference from too much communication[47]. The foundations of the topology are found in the interconnections for processing. Wholly

interconnected arrays are not a viable general VLSI circuit design[2]. Techniques are needed which seek primarily local interconnections.

The Hopfield neural network can be used as an associative memory and also to compute solutions to optimization problems[30]. It fostered the rebirth of interest in neural networks[12]. Each neuron in the Hopfield neural network is connected to every other neuron except for itself. The connection strengths are symmetric and provide for convergence to stable states of the network. The objective function for the optimization problem is the energy function of the network. As the network is updated, its energy is reduced. This neural network has limitations, however. It gives limited good answers. For example, the network doesn't work well above 10 cities in the traveling salesman problem (TSP)[12]. It has the advantage, however, that it is an inherently parallel architecture. A solution to an optimization problem can be mapped onto a Hopfield network by generating an appropriate quadratic energy function which describes the constraints of the problem.

The network connections are derived from an energy function that produces the optimal solution at the lowest energy state of the network. The Hopfield network often fails to find valid solutions and is sensitive to parameterization and initial conditions.

A procedure to map the problem onto the Hopfield network requires the following steps[42]:

1. Select an encoding of the problem so the outputs of the neurons correspond

to the solution to the problem.

2. Define an energy function whose minimum corresponds to the optimal solution of the problem.

3. Derive connection weights and bias currents which represent the objective function and constraints of the problem.

4. Choose initial values for the neuron outputs so the network converges to a stable state which is a feasible solution to the problem.

An advantage of the Hopfield network is that it is relatively simple to implement as an analog circuit. One difficulty though, is the physical implementation of the high gain limit. Also, a globally optimal solution can only be obtained from the network when the conductances are infinite[43]. It is difficult to define an appropriate energy function which corresponds to the desired optimization problem. Some convergence states are local minima which represent infeasible solutions to the problem. The network is also very sensitive to parameterization and initial conditions.

Constrained optimization problems can be mapped onto neural networks by using parameterized penalty terms for the violation of constraints[59]. The quality of the solution depends on the selection of the network parameters in the energy function of Hopfield. Assigning values to the cost function parameters is crucial to the kinds of solutions the network will be able to find. Network parameters are needed to get sensible solutions, and determining their value depends on how to select them, and how to tune them. A linear relationship has been found between

some of the parameters of the Hopfield TSP cost function[27]. Inhibition between the neurons increased for feasible solutions. The global bias to insure complete tours must also increase to stay in the feasible solution region. As the size of the problem (number of cities) increases, this feasible region narrows. This suggests that the network does not scale up, but rather can find fewer feasible solutions as the network grows[27].

The three most important aspects of artificial neural network research are algorithms, application, and architecture[32]. Algorithm mapping technology needs to be used along with CAD technology to derive VLSI arrays from neural network algorithms.

2.3 The Path Planning Application

One application which can benefit from the development of search techniques using massively parallel architectures is path planning for aircraft or mobile robots. A very hard problem in this path planning application is trajectory generation, which requires the computation of the desired trajectory in an embedded real-time system. The tremendous computational demands for trajectory generation[54] make it a good candidate application for this research. The optimization technique which has been extensively used in aircraft trajectory generation systems is dynamic programming.

2.3.1 Overview of Path Planning

Three functions of a flight management system are trajectory generation from the mission plan[55], trajectory optimization for control objectives[20], and trajectory control[53]. Mission plans are defined by waypoint position definitions. Trajectory generation[6] attempts to maximize survivability by optimizing threat penetration paths for terrain masking[37].

A terrain and obstacle avoidance path planner called AUTOPATH was described in [13]. This algorithm is an optimization based technique to generate global flight trajectories. The global flight path incorporates terrain and obstacle avoidance instead of using local avoidance procedures to a preplanned global path. AUTOPATH is an example of a path planner which has several uses. One use is in mission planning. Mission planning uses apriori threat information and computes a baseline mission profile for the aircraft to use during flight. This profile is computed on the ground before the flight using general purpose machines with very good performance. A second use of path planning is to analyze the effectiveness of this baseline mission profile prior to flight. Effectiveness means the objectives of the mission can be met and the aircraft can be returned to base intact and undamaged. Mission effectiveness is a qualitative judgement which is often based upon a value which rates a flight profile. This value is usually a numerical measure defined as a probability of survival along the mission profile and considers the total threat exposure of the aircraft during the mission. The third use of path planning is the real-time onboard updating to the mission plan. This

occurs in-flight while the mission is underway and uses new threat information which is discovered by the sensor suite of the aircraft.

Manual flight trajectory generation is accomplished by inserting waypoints between which a smoothed flight path is created. To generate avoidance paths, additional waypoints are inserted in such a manner that the resulting smoothed flight path will deviate around the area to avoid. This technique works for threat areas which are well defined and have good separation. In addition, robots, drones, remotely-piloted vehicles, and particularly autonomous vehicles cannot use manual waypoints. Most threat scenarios involve regions where the threats are arranged in a very dense coverage with much overlap. Since there is no isolation between the threats within these regions, this manual type of avoidance trajectory generation is eliminated. What is required is a mathematical optimization process which generates trajectories with minimal threat exposure for the non-uniform continuum of threats which are associated with the dense threat coverage.

The path planning problem consists of a large problem space, typically with several state variables. Path planning requires finding a path out of all the possible paths in a multidimensional search space which is a path with the optimal performance measure. Due to its large search space, the path planning problem places severe demands on computational resources[54]. These problems often require real-time solutions unavailable from current technology.

System response time depends on the throughput, memory, and interfacing capacities of the computer or computer network used for airborne mechanization

of trajectory generation. This problem is difficult to solve in a time expedient fashion using the currently available airborne processing technology, due to the massive computational throughput requirement associated with an extensive search space.

2.3.2 Path Generation by Searching

Aircraft trajectory optimization is an application of path planning by searching through a multidimensional search space for some path which has the optimal performance measure through the search space. Optimization techniques to compute minimum danger paths for mission segments involving enemy threats have been developed[54]. Survivability is an example of a performance measure of mission effectiveness. It is quantitatively defined in a measure called danger by accounting for the threat exposure of the aircraft during an entire mission. This includes threat lethality and terrain masking advantages of the mission profile. Lethality is taken to be the time derivative of the probability of kill $\dot{P}_K$. Aircraft speed effects are implicitly included in this measure since it is a danger rate. The quantitative measure of performance is usually expressed as a probability of survival P_s at all points along the aircraft trajectory. This performance measure attempts to capture a numerical representation of the probability of successfully completing the mission without incurring damage or destruction from the threats. Survival is improved by finding a trajectory through the threat region for which the performance measure is maximized. Rather than maximizing the probability of survival performance measure, the problem is converted to the minimization of a

performance measure which is a monotone decreasing function of the negative logarithm of the probability of survival:

$$D = \mathcal{F}(-\log P_s) \tag{2.3}$$

This function computes a danger measure based upon the survivability in a dense threat environment taking into account the threat locations and lethality, aircraft exposure due to altitude, and masking effects from the terrain. This metric is used as the cost function for the search space by assigning a danger value to every state in the search space. Danger for any path is by definition:

$$D_{\text{path}} = \int_{\text{path}} dD \tag{2.4}$$

$$= \int_{\text{path}} \frac{dD}{ds} ds = \int_{\text{path}} \frac{dD}{dt}\frac{dt}{ds} ds \tag{2.5}$$

$$= \int_{\text{path}} \frac{\dot{D}}{V} ds = \int_{\text{path}} \dot{D}\, dt \tag{2.6}$$

The threat penetration problem was formulated as an optimization problem where a performance measure which is a function of the aircraft trajectory was defined. The threat penetration trajectory generation algorithm computes the trajectory which minimizes danger as the following performance measure (J):

$$J(\text{path}) = \int_{\text{path}} \dot{D}\, dt \tag{2.7}$$

$\dot{D}$ is the danger index function representing the risk per unit time imposed on the aircraft by the threat environment as a function of horizontal position, altitude, and heading. Dynamic programming[5] is the mathematical technique used for determining the trajectory which minimizes the performance measure by performing

a search of potential trajectories using a recursive formulation. By accumulating transition costs over the sequence of node transitions making up a particular trajectory, the performance measure (J) for the trajectory is determined, i.e.,

$$J = \sum_i \dot{D}_i \Delta t_i \tag{2.8}$$

AUTOPATH is a dynamic programming approach in which the performance measure for the mathematical optimization is defined as a military gain[13]. This gain is a usually a weighted sum of the probability of successfully completing a mission, and the probability of surviving the mission. It is implemented by partitioning the threat coverage region of space through which the aircraft flies into an array of small cells. These cells usually cover an area of one to four nautical miles (see Figure 2.2). The center of these cells are the coordinates of a

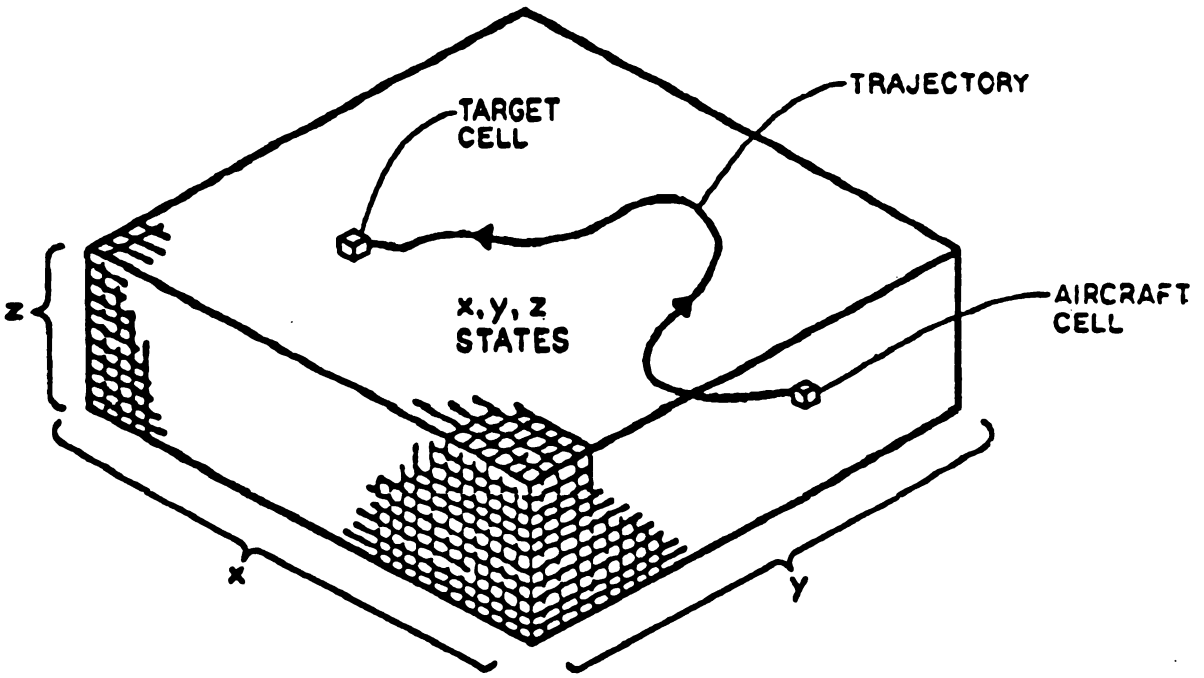

Figure 2.2: *Trajectory Generation Through a Region Partitioned into Grid Cells*

multidimensional grid. Each cell has an associated vector of values which reflect the probability of survival based upon the threat location and models of lethality.

The overall survivability of a path through the cells is the product along the path of the probability of survival through each cell[13]. A path is defined with a start cell, an end cell, and a contiguous set of intermediate cells from the start to end cell. The performance measure is implemented by setting the cell value to the negative logarithm of the probability of survival. There is a benefit in transforming the measure with this implementation. It is more desirable to work with summation than multiplication since it is a simpler computation. The logarithm accomplishes this, although it must be recognized that this transforms the problem from maximizing products to minimizing sums. The negative logarithm of the probability of survival is proportional to the probability of kill. This is, therefore, a simpler computation which is equivalent to minimizing the probability of kill (not surviving).

The generated mission flight trajectories are to minimize pilot risk (the integrated danger) due to the threats over the set of possible trajectories defined on a geographical grid of the mission[54]. Discrete dynamic programming is the optimization technique which has been used to compute trajectories on this grid. The state space of the trajectory generation problem is a discretized lattice of nodes. The trajectory is the successive transitions between adjacent nodes of the grid. The transition cost is the value of the danger at the node. The computing grid for this research is defined as an orthogonally connected mesh-connected processor array[47].

2.4 Summary

The algorithms described in this chapter are several of the traditional approaches to solving problems using search. All of these approaches have centered on algorithms for implementation on serial processors. Some research has addressed converting these algorithms into parallelized versions for execution on multiprocessors[1, 26, 34, 40]. Those approaches also begin with the serial perspective on algorithm design. This research will show a unique approach to searching that does not use a serial viewpoint, but rather a truly parallel model which encompasses the problem space representation as well as the algorithmic model.

VLSI offers opportunities to implement feasible, large scale neural networks to achieve massive parallelism. Optimization problems, such as shortest path search, can be mapped onto neural networks. However, the Hopfield network approach suffers from several difficulties. It is difficult to encode problems using the energy function minimization approach. Several parameters are required and little is known on how to specify their value. Evidence indicates that the network does not scale well for larger problems. Later chapters will describe a neural network which avoids these problems, since it is based on existing numerical analysis and has a good mathematical basis.

Trajectory generation for aircraft is a well studied application of path planning search. It has been mapped into a multidimensional grid representation with each cell of the grid containing the cost of flying through that cell. The trajectory

optimization problem then is to find the minimum cost path through the grid by searching. This application has been studied using discrete dynamic programming. In later chapters, this application will be used to illustrate results obtained from the massively parallel neural network architecture design for trajectory generation.

Chapter 3

Parallel Architecture Model For Path Planning

This chapter introduces a new parallel model for the shortest path problem[62]. The natural phenomenon of electrostatic fields are shown to have a similarity to path planning with multiple, alternative solution paths. Analysis of the electrostatic model results in a partial differential equation and its boundary conditions which correspond to the path planning problem requirements. A finite difference approximation for computing the numerical solution to the partial differential equation is also derived. This finite difference approximation is the basis for a new neural network designed to compute the numerical solution. Finally, a technique is developed for increasing the amount of connectivity in the neural network in order to decrease the solution time.

3.1 Overview Of An Analogy In Nature

The basis for this new model of path planning lies in the understanding that problems can be described by physical analogies in nature. This new path planning model uses an analogy of electromagnetic field theory for the mathematical

model. The solution to the application problem is provided by a partial differential equation in field theory.

The model taken from electromagnetic phenomena is based on electrostatics. The electrostatic model is used to compute multiple, parallel paths which avoid regions of high cost. The mathematical model describes paths through a region containing a variable cost function as a problem in mathematical physics. One such problem is finding the distribution of current flow through a nonuniform conducting media such as a plate of nonhomogeneous resistive material.

This physics problem is described using partial differential equations. The cost function for the path planning problem is analogous to the nonuniform resistivity of the media. The solution to the Laplacian partial differential equation is a potential field in the media. Paths are computed orthogonal to the equipotential contours of this field.

3.2 The Electrostatic Model for Parallel Path Planning

The electrostatic model is a physical model in nature which is used for the natural analogy of parallel path planning. The natural analogy contains corresponding entities between the physical model and the abstract path planning model. The paths of the path planning model correspond to the current flux lines of the electrostatic model. These flux lines describe the distribution of current flow. The variable cost region of the path planning model corresponds to the nonuniform conducting medium of the electrostatic model. A plate of nonhomogeneous resistive material is an example of such a medium. With this analogy,

finding good paths is like describing the manner in which current flows through the medium. The solution of the distribution of current flow can be used as a corresponding solution for path planning.

When a potential difference is applied at two different points in a resistive media, current flows along paths between the points where the different potentials are applied. The point with the higher potential is the source point, and the point with the lower potential is the sink point. The paths of current flow are determined by the resistivity of the media and the locations of the source and sink points. The distribution of current flow along these paths is optimized by nature in some fashion. Current path distribution is less dense through regions where the media has a higher concentration of resistivity. Conversely, regions with more conductivity (lower resistivity) have a greater density of current distribution.

Current flow is along flux lines which correspond to the current density field. The current density field is continuous, so there are an infinite number of flux lines from the source point to the sink point. From the source to the sink, the current flux lines can be traced along the direction specified by the current density vector field. (Just as magnetic flux lines can be seen between the poles of a bar magnet by sprinkling metal filings upon a sheet of paper placed on top of the magnet.)

In the electrostatic model for path planning, several analogies to nature are defined. The variable cost function for the search problem is represented by the nonuniform resistivity of the media. This media has the same dimensionality as the problem space. The range of each dimension of the problem space defines the extent of a grid upon which a spatial sampling is performed for the architecture.

Time may be explicitly represented as one of the dimensions, if the cost function is a function of time. The source and sink locations represent the start and goal nodes, respectively of the search space.

3.2.1 The Mathematical Model

The mathematical model is based upon electric field theory[51]. From the definition for current density and its divergence in steady state conditions the N- dimensional electric field potential $\phi \equiv \phi(x_1, x_2, \ldots, x_N)$ with x_i as orthogonal coordinates of a nonuniform conductive media can be derived[60] as the solution to the second order partial differential equation

$$\nabla^2\phi + \rho\nabla\sigma \cdot \nabla\phi = 0 \tag{3.1}$$

where $\rho \equiv \rho(x_1, x_2, \ldots, x_N)$ is the nonuniform resistivity of the media, and $\sigma = \rho^{-1}$. The cost function is modeled by the resistivity of the conducting media. Expanding the gradient and divergence operators in the two dimensions x, y results in the following second order partial differential equation

$$\phi_{xx} + \phi_{yy} + \rho\sigma_x\phi_x + \rho\sigma_y\phi_y = 0 \tag{3.2}$$

Since the electric field and current density results from the gradient of a scalar field, they are conservative fields, and the field lines emanate from source charges and terminate on sink charges[71]. These field lines represent possible solutions to the search problem as multiple paths. Figure 3.1 is an example of this concept. This example is the field which results from dipole charges on a uniformly conductive surface. This corresponds to the path planning problem in a region which

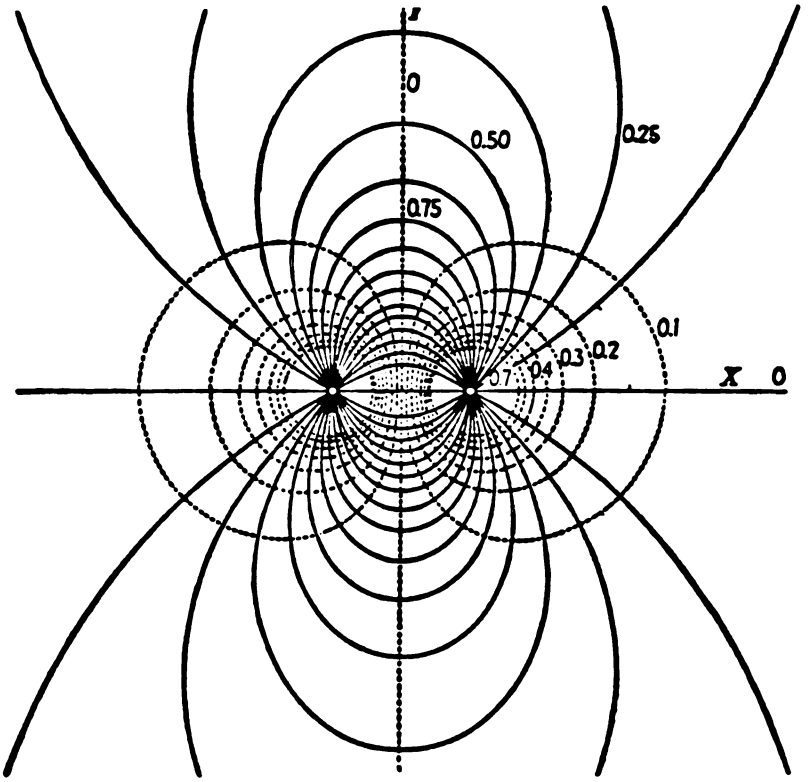

Figure 3.1: *Example of Field Lines Used as Multiple Paths*

has a constant cost. The least cost path between the end points is a straight line. This is shown by a straight field line in the figure. Additional curved field lines are shown which "bulge" out from the straight field lines between the dipole charges. These curved lines then correspond to multiple paths of the corresponding path planning problem. These multiple path solutions do not include all of the possible solution paths in the exponential search space. In using the field lines as paths, the solution space is restricted to include only those paths which do not share any path segments. The multiple paths will not split from any common path or join together into a single path. The multiple paths all originate at the start state as unique alternatives.

Background Theory

This section reviews those aspects of electric field theory which are pertinent to the model. For a more extensive explanation of electromagnetic phenomenon, see [45, 51, 71]. The following analysis will be used to derive Equation 3.1.

The divergence of the current density $\boldsymbol{j}$ is defined as the charge (ρ) rate of change

$$\nabla \cdot \boldsymbol{j} = -\frac{\partial \rho}{\partial t} \tag{3.3}$$

In the steady state condition, no charge distribution changes occur

$$\frac{\partial \rho}{\partial t} = 0 \tag{3.4}$$

Thus, in the steady state condition,

$$\nabla \cdot \boldsymbol{j} = 0 \tag{3.5}$$

The current density is defined in terms of the electric field $\boldsymbol{E}$ and the conductivity σ as

$$\boldsymbol{j} = \sigma \boldsymbol{E} \tag{3.6}$$

Substitution of Equation 3.6 into Equation 3.5 gives

$$\nabla \cdot (\sigma \boldsymbol{E}) = 0 \tag{3.7}$$

or, equivalently through a vector identity that

$$\nabla \sigma \cdot \boldsymbol{E} + \sigma \nabla \cdot \boldsymbol{E} = 0 \tag{3.8}$$

The electric field is also defined as the negative gradient of the potential field ϕ

$$\boldsymbol{E} = -\nabla \phi \tag{3.9}$$

Substituting Equation 3.9 into Equation 3.8 yields

$$\nabla \sigma \cdot (-\nabla \phi) + \sigma \nabla \cdot (-\nabla \phi) = 0 \tag{3.10}$$

After dividing the equation by $(-\sigma)$, the equation for the electric field potential of a nonuniform conductive media is

$$\nabla^2 \phi + \frac{1}{\sigma} \nabla \sigma \cdot \nabla \phi = 0 \tag{3.11}$$

Expanding the gradient and divergence operators in two dimensions results in the following second order partial differential equation

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{1}{\sigma} \frac{\partial \sigma}{\partial x} \frac{\partial \phi}{\partial x} + \frac{1}{\sigma} \frac{\partial \sigma}{\partial y} \frac{\partial \phi}{\partial y} = 0 \tag{3.12}$$

This is expressed more conveniently using subscript notation for partial derivatives as

$$\phi_{xx} + \phi_{yy} + \frac{1}{\sigma} \sigma_x \phi_x + \frac{1}{\sigma} \sigma_y \phi_y = 0 \tag{3.13}$$

By solving Equation 3.13 for ϕ and substituting into Equation 3.9, the current flow vector field can be found by substituting Equation 3.9 into Equation 3.6.

Field Lines Form Paths

The field lines for the electrostatic model are used as the path definitions. The field lines are computed once the electrostatic potential field ϕ is derived for the problem from the solution to Equation 3.13. The current density vector field $\boldsymbol{j}$ of Equation 3.6 is used to trace the field lines. This vector field is the tangential field for the solution. There are an infinite number of field lines which leave the source point and enter the sink point. At every point in the potential field with the exception of the source and sink, the vector field (the electric field vector) is defined. From the source point, an angle (initial heading at the start node) is selected from which to trace out the field line. Using a small incremental path step, the path progresses along this direction. From this point on, the gradient vector can be computed which will define the direction of the path descent over the potential field surface, obtaining the path as the field line. Gradient descent over the potential field will not be troubled with local minima since the potential field cannot have maximum or minimum values except at the source and sink locations[68].

In addition to the source and sink values included as boundary conditions described previously, it is necessary to establish boundary conditions along the edge of the grid for the problem. These boundary conditions will determine what happens to the paths where they approach the edge of the grid. The grid itself is

assumed to be large enough to include a sufficient portion of the path planning area in which multiple solutions are needed. Since all of the paths should be continuous from start to goal within this planning area, it is desired that none of the solution paths run off the grid. Thus, asymptotically near the edge of the grid, each path should approach a parallel course to the edge.

For the field lines to be asymptotically parallel at the edge of the problem grid, it is necessary to establish the equipotential contours as normal to the grid edges as they approach the edge. This is accomplished by setting up the following boundary conditions:

$$\left.\frac{\partial \phi}{\partial y}\right|_{(y = y_{\min}, y = y\max)} = 0 \tag{3.14}$$

and

$$\left.\frac{\partial \phi}{\partial x}\right|_{(x = x_{\min}, x = x\max)} = 0 \tag{3.15}$$

The first boundary condition (Equation 3.14), causes the field lines, $\boldsymbol{E}$, to be parallel to the unit vector, $\boldsymbol{x}$, since along the top and bottom edges of the grid the y-component of the potential gradient is zero. Similarly, the second boundary condition (Equation 3.15) causes parallel solutions of $\boldsymbol{E}$ along the left and right edges of the grid because the x-component of the potential gradient is zero also.

3.2.2 Analysis

In this section, the analysis of the minimization performed by the electrostatic model is reviewed. The analysis uses the calculus of variations [7, 8, 9, 11, 38, 44, 48]. The partial differential equation for the electrostatic solution to the

nonuniformly conductive media (Equation 3.13) is derivable from the calculus of variations. The derivation shows the corresponding minimization function for the electrostatic model.

The objective of the calculus of variations is to find the extremal surface $u(x, y)$ which minimizes the variational integral

$$\iint I(u, x, y, u_x, u_y)\, dx\, dy \tag{3.16}$$

This is satisfied when the Euler Equation is found:

$$\frac{\partial I}{\partial u} - \frac{\partial}{\partial x}\frac{\partial I}{\partial u_x} - \frac{\partial}{\partial y}\frac{\partial I}{\partial u_y} = 0 \tag{3.17}$$

This gives a second order partial differential equation in the cases of interest.

The following will derive the Euler Equation for two independent variables, x and y. It is desired to find a function $u(x, y)$ which will make stationary the integral

$$\int_{x_1}^{x_2}\int_{y_1}^{y_2} I(u, x, y, u_x, u_y)\, dx\, dy \tag{3.18}$$

The stationarity condition is satisfied when

$$\int_{x_1}^{x_2}\int_{y_1}^{y_2} \delta I\, dx\, dy = 0 \tag{3.19}$$

The variation, δu, is the increment in passing from the extremal, u, to the neighboring function, U, while holding the independent variables x, y fixed. From this $\delta x = \delta y = 0$, which leaves

$$\delta I = \frac{\partial I}{\partial u}\delta u + \frac{\partial I}{\partial u_x}\delta u_x + \frac{\partial I}{\partial u_y}\delta u_y \tag{3.20}$$

In order to evaluate Equation 3.19 and Equation 3.20, it should be noticed that

$$\iint \frac{\partial I}{\partial u_x}\delta u_x\, dx\, dy = \iint \frac{\partial I}{\partial u_x}\delta\left(\frac{\partial u}{\partial x}\right) dx\, dy = \iint \frac{\partial I}{\partial u_x}\frac{\partial}{\partial x}(\delta u)\, dx\, dy \tag{3.21}$$

Now integrate Equation 3.21 by parts by letting

$$v = \frac{\partial I}{\partial u_x} \quad \text{and} \quad dw = \frac{\partial}{\partial x}(\delta u)\,dx = \delta u'\,dx \tag{3.22}$$

while holding y fixed. From this,

$$dv = \frac{\partial}{\partial x}\left(\frac{\partial I}{\partial u_x}\right)dx \quad \text{and} \quad w = \delta u \tag{3.23}$$

So upon substitution into

$$\int v\,dw = vw - \int w\,dv \tag{3.24}$$

there is

$$\int_{x_1}^{x_2} \frac{\partial I}{\partial u_x}\frac{\partial}{\partial x}(\delta u)\,dx = \frac{\partial I}{\partial u_x}\delta u\Bigg|_{x_1}^{x_2} - \int \delta u \frac{\partial}{\partial x}\left(\frac{\partial I}{\partial u_x}\right)dx \tag{3.25}$$

This yields

$$\frac{\partial I}{\partial u_x}\delta u_x\Bigg|_{x_1}^{x_2} = \frac{\partial I}{\partial u_x}\left[U(x_2,y) - u(x_2,y) - U(x_1,y) + u(x_1,y)\right] \tag{3.26}$$

with y held fixed. But, $U(x_2,y) = u(x_2,y)$ and $U(x_1,y) = u(x_1,y)$ so what is left is

$$\int_{x_1}^{x_2} \frac{\partial I}{\partial u_x}\frac{\partial}{\partial x}(\delta u)\,dx = -\int_{x_1}^{x_2} \frac{\partial}{\partial x}\left(\frac{\partial I}{\partial u_x}\right)\delta u\,dx \tag{3.27}$$

Similar evaluations for y gives

$$\int_{y_1}^{y_2} \frac{\partial I}{\partial u_y}\frac{\partial}{\partial y}(\delta u)\,dy = -\int_{y_1}^{y_2} \frac{\partial}{\partial y}\left(\frac{\partial I}{\partial u_y}\right)\delta u\,dy \tag{3.28}$$

Combining Equation 3.19 and Equation 3.20 with substitution by Equation 3.27 and Equation 3.28 gives

$$\int_{x_1}^{x_2}\int_{y_1}^{y_2}\left(\frac{\partial I}{\partial u} - \frac{\partial}{\partial x}\frac{\partial I}{\partial u_x} - \frac{\partial}{\partial y}\frac{\partial I}{\partial u_y}\right)\delta u\,dx\,dy = 0 \tag{3.29}$$

Thus, the Euler Equation now is

$$\frac{\partial I}{\partial u} - \frac{\partial}{\partial x}\frac{\partial I}{\partial u_x} - \frac{\partial}{\partial y}\frac{\partial I}{\partial u_y} = 0 \tag{3.30}$$

To apply the calculus of variations to the electrostatic model, set the function to minimize, I, to

$$I(\phi, x, y, \phi_x, \phi_y) = \boldsymbol{j} \cdot \boldsymbol{E} \tag{3.31}$$

This right side is by definition

$$\boldsymbol{j} \cdot \boldsymbol{E} = \sigma(x, y)(\phi_x^2 + \phi_y^2) \tag{3.32}$$

so the function is

$$I(\phi, x, y, \phi_x, \phi_y) = \sigma(x, y)(\phi_x^2 + \phi_y^2) \tag{3.33}$$

From this definition

$$\frac{\partial I}{\partial \phi} = 0 \tag{3.34}$$

$$\frac{\partial I}{\partial \phi_x} = 2\sigma\phi_x \tag{3.35}$$

$$\frac{\partial I}{\partial \phi_y} = 2\sigma\phi_y \tag{3.36}$$

$$\frac{\partial}{\partial x}\left(\frac{\partial I}{\partial \phi_x}\right) = 2\sigma_x\phi_x + 2\sigma\phi_{xx} \tag{3.37}$$

$$\frac{\partial}{\partial y}\left(\frac{\partial I}{\partial \phi_y}\right) = 2\sigma_y\phi_y + 2\sigma\phi_{yy} \tag{3.38}$$

The Euler Equation (Equation 3.30) for two dimensions is

$$\frac{\partial I}{\partial \phi} - \frac{\partial}{\partial x}\left(\frac{\partial I}{\partial \phi_x}\right) - \frac{\partial}{\partial y}\left(\frac{\partial I}{\partial \phi_y}\right) = 0 \tag{3.39}$$

Substitution of Equation 3.34, Equation 3.37, and Equation 3.38 into Equation 3.39 yields

$$-2\sigma_x\phi_x - 2\sigma\phi_{xx} - 2\sigma_y\phi_y - 2\sigma\phi_{yy} = 0$$

Dividing through by -2σ gives the final result:

$$\phi_{xx} + \phi_{yy} + \frac{1}{\sigma}\sigma_x\phi_x + \frac{1}{\sigma}\sigma_y\phi_y = 0 \tag{3.40}$$

This shows that the partial differential equation (Equation 3.40) minimizes the energy loss due to heat loss ($\boldsymbol{j} \cdot \boldsymbol{E}$) in a nonuniform conducting medium[22, 58].

3.3 Parallel Architecture Design For Path Planning

In the previous section, path planning using the Electrostatic Model was developed. A nonhomogeneous media was used to model the cost function by representing regions of high cost through high resistivity values in the media. The Laplacian partial differential equation was derived using the nonhomogeneous media. The solution to the Laplacian is the potential surface over which the electrostatic field lines represent multiple paths which avoid regions of high cost.

This section develops the architectural design of a neural network[61] that computes the numerical solution to the Laplacian partial differential equation. The architecture implements a finite difference approximation[10] to compute a numerical solution to the partial differential equation. This section begins with a review of the numerical approximation which maps into the mathematical physics model of the problem. The approximation defines an artificial neural network processing unit and its weighted interconnections to a selected set of neighboring processing units.

The computational definition of the massively parallel architecture is based on numerical solutions to the partial differential equation. A neural network architecture

is defined which computes the scalar potential field solution to the partial differential equation by using the method of finite difference approximation. The approximation templates of the difference formula define the connection strengths.

The experimental results of this method are presented in Section 4.2.

3.3.1 Constructing the Difference Formula

It is desired that the general second-order partial differential equation

$$L(u) = A\frac{\partial^2 u}{\partial x^2} + B\frac{\partial^2 u}{\partial x \partial y} + C\frac{\partial^2 u}{\partial y^2} + D\frac{\partial u}{\partial x} + E\frac{\partial u}{\partial y} + Fu = 0 \tag{3.41}$$

be approximated[23, 25, 39, 49, 64, 65] on a unit grid by

$$L_1(u) \approx \alpha_o u_P - \sum_{i=1}^{n} \alpha_i u_{Q_i} \tag{3.42}$$

with the n neighbors of P being $Q_1, \ldots, Q_n$, where $Q_i = (x + \xi_i, y + \eta_i)$. The Taylor series about $P = (x, y)$ in two variables is

$$\begin{aligned} f(x + ih, y + jk) &= f(x,y) + ih\frac{\partial}{\partial x}f(x,y) + jk\frac{\partial}{\partial y}f(x,y) \\ &+ \frac{(ih)^2}{2}\frac{\partial^2}{\partial x^2}f(x,y) + ijhk\frac{\partial^2}{\partial x \partial y}f(x,y) \\ &+ \frac{(jk)^2}{2}\frac{\partial^2}{\partial y^2}f(x,y) + \ldots \end{aligned} \tag{3.43}$$

Let the neighborhood of P be defined as the n points (an $n+1$ point approximation template) $Q_i = \{(x + \xi_i, y + \eta_i)\}$, where ξ_i, η_i are integers ($h = k = i$ for the unit grid). The expansion of $u(x + \xi_i, y + \eta_i)$ about P is

$$\begin{aligned} u(x + \xi_i, y + \eta_i) = U_{Q_i} &= u(x,y)|_P + \xi_i \left.\frac{\partial u}{\partial x}\right|_P + \eta_i \left.\frac{\partial u}{\partial y}\right|_P \\ &+ \frac{\xi_i^2}{2}\left.\frac{\partial^2 u}{\partial x^2}\right|_P + \xi_i\eta_i \left.\frac{\partial^2 u}{\partial x \partial y}\right|_P \\ &+ \frac{\eta_i^2}{2}\left.\frac{\partial^2 u}{\partial y^2}\right|_P + \ldots \end{aligned} \tag{3.44}$$

So Equation 3.42 upon substitution by Equation 3.44 and collection of common terms becomes

$$L_1(u) = 0 = \left.\frac{\partial^2 u}{\partial x^2}\right|_P \frac{1}{2}\sum_{i=1}^{n}\alpha_i\xi_i^2 + \left.\frac{\partial^2 u}{\partial x \partial y}\right|_P \sum_{i=1}^{n}\alpha_i\xi_i\eta_i + \left.\frac{\partial^2 u}{\partial y^2}\right|_P \frac{1}{2}\sum_{i=1}^{n}\alpha_i\eta_i^2 + \left.\frac{\partial u}{\partial x}\right|_P \sum_{i=1}^{n}\alpha_i\xi_i + \left.\frac{\partial u}{\partial y}\right|_P \sum_{i=1}^{n}\alpha_i\eta_i + u_P\left(\sum_{i=1}^{n}\alpha_i - \alpha_0\right) \quad (3.45)$$

Substituting the coefficients of Equation 3.45 for those of Equation 3.41 gives the system of equations

$$\sum_{i=1}^{n}\alpha_i - \alpha_0 = F \quad (3.46)$$

$$\sum_{i=1}^{n}\alpha_i\eta_i = E \quad (3.47)$$

$$\sum_{i=1}^{n}\alpha_i\xi_i = D \quad (3.48)$$

$$\sum_{i=1}^{n}\alpha_i\eta_i^2 = 2C \quad (3.49)$$

$$\sum_{i=1}^{n}\alpha_i\xi_i\eta_i = B \quad (3.50)$$

$$\sum_{i=1}^{n}\alpha_i\xi_i^2 = 2A \quad (3.51)$$

The solution of the coefficients α_i of the above system of equations allows approximating the value of a grid point by solving Equation 3.52 for u_P

$$u_P = \frac{1}{\alpha_0}\sum_{i=1}^{n}\alpha_i u_{Q_i} \quad (3.52)$$

For the nonhomogeneous equation, set the quantities $C_x = \sigma_x/\sigma$ and $C_y = \sigma_y/\sigma$, resulting in

$$L_1(\phi) = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + C_x\frac{\partial\phi}{\partial x} + C_y\frac{\partial\phi}{\partial y} = 0 \quad (3.53)$$

and the coefficients of Equation 3.41 are $A = 1, B = 0, C = 1, D = C_x, E = C_y$, and, $F = 0$. It is desired to derive a five-point finite difference formula. The

coefficients are as shown in Table 3.1. Substituting the values of Table 3.1 into

Table 3.1: *Coordinates of the 5-pt. Finite Difference Formula*

i	ξ_i	η_i
1	1	0
2	0	1
3	-1	0
4	0	-1

the system defined by Equation 3.46 through Equation 3.51, yields the following system of equations

$$\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = \alpha_0 \tag{3.54}$$

$$\alpha_2 - \alpha_4 = C_y \tag{3.55}$$

$$\alpha_1 - \alpha_3 = C_x \tag{3.56}$$

$$\alpha_2 + \alpha_4 = 2 \tag{3.57}$$

$$\alpha_1 + \alpha_3 = 2 \tag{3.58}$$

The solution to this set of equations is

$$\alpha_1 = 1 + \frac{C_x}{2} \tag{3.59}$$

$$\alpha_2 = 1 + \frac{C_y}{2} \tag{3.60}$$

$$\alpha_3 = 1 - \frac{C_x}{2} \tag{3.61}$$

$$\alpha_4 = 1 - \frac{C_y}{2} \tag{3.62}$$

$$\alpha_0 = 4 \tag{3.63}$$

Substituting Equation 3.59 through Equation 3.63 into Equation 3.42 and solving

for U_P gives the finite difference formula

$$U_P = \frac{1}{4}\left[\left(1 + \frac{C_x}{2}\right)U_{Q_1} + \left(1 + \frac{C_y}{2}\right)U_{Q_2} + \left(1 - \frac{C_x}{2}\right)U_{Q_3} + \left(1 - \frac{C_y}{2}\right)U_{Q_4}\right] \tag{3.64}$$

When the template overlaps a boundary of the grid, a method is needed to define the appropriate connections for the off-grid node. The solution is to use the method of images[58, 68, 71]. The method of images is an analysis technique for dealing with boundary conditions in electrostatic fields. The technique requires a mirror image of the conductivity function to be reflected about the boundary of the medium. In addition to reflecting the conductivity function, the point charges are likewise reflected. This creates a dipole charge relationship for which a field solution is easily generated using this expanded region of the medium. The problem is reflected about the boundary whose condition is

$$\frac{\partial f}{\partial x_i} = 0$$

As the problem has an identical image, this results in maintaining the boundary condition. For example, if solving the half-plane A shown in Figure 3.2 by reflecting the half-plane A into B, and reflecting the point charge P, solving the entire plane as a dipole charges results in maintaining the boundary condition

$$\frac{\partial f}{\partial y} = 0$$

along the reflection line x. Now the boundary region approximations will result in reflecting back into the original region. So if the approximation for the point P as shown in Figure 3.3 uses Q' in the image problem, Q' can be reflected back into

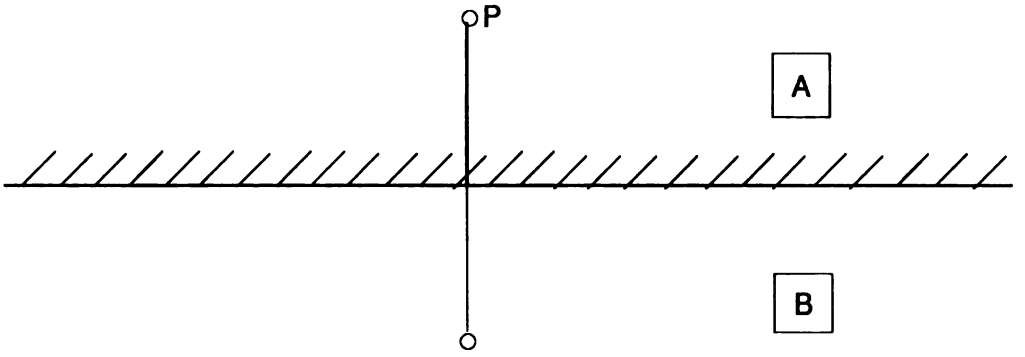

Figure 3.2: *Image Dipoles Resulting from Reflecting Half-Plane A into B*

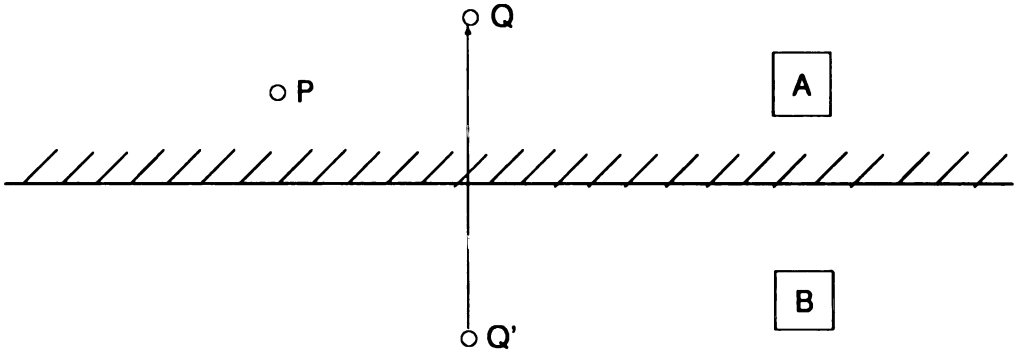

Figure 3.3: *Mapping the Approximation point Q' for P into Q*

Q which can be used instead of Q in the approximation formula.

For the path planning problem space the reflections shown in Figure 3.4 need to be established. The required problem space with the needed reflection images are shown in Figure 3.5. Although there are eight reflections for the problem space to be completely surrounded, only three unique reflection images are required. These unique reflection images are shown in Figure 3.5. For each of the reflection images (I, II, and III), the coordinates need to be reflected back into the original problem space. In Figure 3.5, notice the correspondence between the problem and image maps and the quadrant assignments. This quadrant

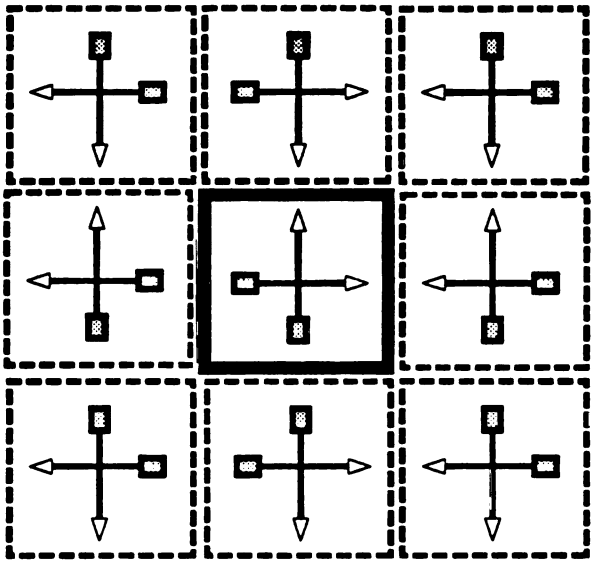

Figure 3.4: *Image Reflections Surrounding the Path Planning Problem Space*

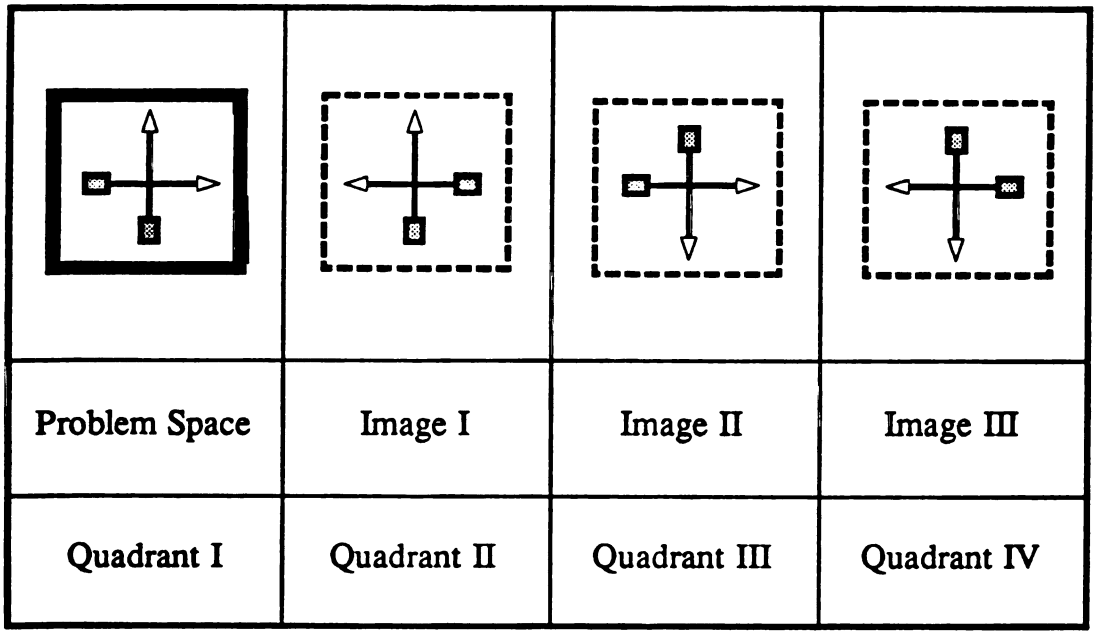

Figure 3.5: *The Unique Reflection Images of the Path Planning Problem Space*

assignment is shown in Figure 3.6. If the row or column index for a template co-

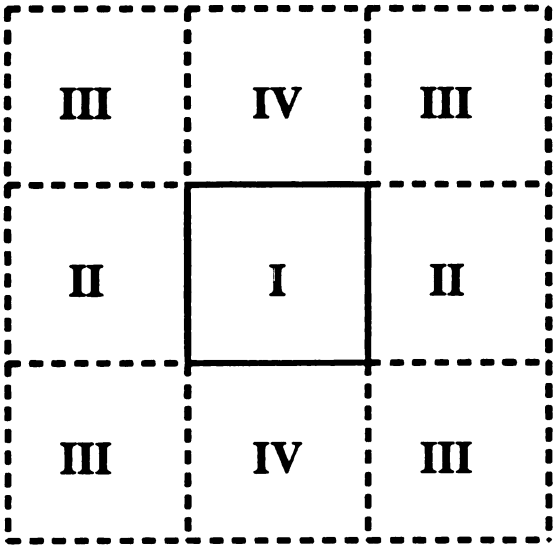

Figure 3.6: *Reflection Images Assigned to Quadrants of the Problem Space*

ordinate occurs in one of these image maps, then the index needs to be mapped back into the problem space. If the index $k > n$, then to reflect the index back into the problem it is necessary to compute $n - k \bmod n$. If the index $k < 1$, then to reflect the index back into the problem it is necessary to compute $2 - k$. So the coordinates to map a quadrant Q point $(i,j)_Q$ into a point in the first quadrant, I, $(i,j)_I$ is

$$(i,j)_I = \begin{cases} ((n - i \bmod n), j)_{II} & \text{if } i > n \\ ((2-i), j)_{II} & \text{if } i < 1 \end{cases}$$

$$(i,j)_I = \begin{cases} (i, (n - j \bmod n))_{IV} & \text{if } j > n \\ (i, (2-j))_{IV} & \text{if } j < 1 \end{cases}$$

For quadrant III, $(i,j)_I = (k,l)$, where

$$k = \begin{cases} n - (i_{III} \bmod n) & \text{if } i_{III} > n \\ 2 - i_{III} & \text{if } i_{III} < 1 \end{cases}$$

and

$$l = \begin{cases} n - (j_{III} \bmod n) & \text{if } j_{III} > n \\ 2 - j_{III} & \text{if } j_{III} < 1 \end{cases}$$

The algorithm for mapping the image index is shown in Figure 3.7. An example of

```
 1.    begin IMAGEINDEX(p, q, n, p', q')
 2.        p' ⟵ p;
 3.        q' ⟵ q;
 4.        if p' > n then
 5.            p' ⟵ n − p' mod n;
 6.        elseif p' < 1 then
 7.            p' ⟵ 2 − p';
 8.        endif
 9.        if q' > n then
10.            q' ⟵ n − q' mod n;
11.        elseif q' < 1 then
12.            q' ⟵ 2 − q';
13.        endif
14.    end IMAGEINDEX;
```

Figure 3.7: *Algorithm to Map Image Indicies into Problem Space Indicies*

this mapping using the five point approximation for a nonuniform medium follows.

The approximation at the $y = n$ border is defined as follows

$$\phi_{i,n} = \frac{1}{4}\left[(1 + \frac{C_x}{2})\phi_{i+1,n} + (1 - \frac{C_x}{2})\phi_{i-1,n} + (1 + \frac{C_y}{2})\phi_{i,n+1} + (1 - \frac{C_y}{2})\phi_{i,n-1}\right] \tag{3.65}$$

But, by definition

$$\left.\frac{\partial \phi}{\partial y}\right|_{i,n} \equiv 0 = \frac{\phi_{i,n+1} - \phi_{i,n-1}}{2} \tag{3.66}$$

Therefore,

$$\phi_{i,n+1} = \phi_{i,n-1}$$

which, of course is the expected image mapping. By substituting this back into

Equation 3.65, the final result is

$$\phi_{i,n} = \frac{1}{4}\left[(1 + \frac{C_x}{2})\phi_{i+1,n} + (1 - \frac{C_x}{2})\phi_{i-1,n} + 2\phi_{i,n-1}\right] \tag{3.67}$$

Notice, however, an intermediate step in the simplification of the above equation which resulted in Equation 3.67, included the step

$$\phi_{i,n} = \frac{1}{4}\left[(1 + \frac{C_x}{2})\phi_{i+1,n} + (1 - \frac{C_x}{2})\phi_{i-1,n} + (1 + \frac{C_y}{2})\phi_{i,n-1} + (1 - \frac{C_y}{2})\phi_{i,n-1}\right] \tag{3.68}$$

But Equation 3.68 is just the computation template architecture including the appropriate image mapping. So the desired approximation at the boundary (Equation 3.67) is accomplished using the `IMAGEINDEX` mapping for the approximation (Equation 3.68).

3.3.2 Defining the Neural Network Architecture

This artificial neural network architecture is similar to the computational architecture known as the systolic array. From its similarity to a grid, each point of the grid is a processor, and the links of the grid consist of the links between the processors. This array characteristic of the architecture capitalizes on the regular and modular structures which match the computational requirements of the model.

The mapping of the finite difference approximation into this grid is illustrated in Figure 3.8. The block diagram which shows the neuron model definition which implements the five point approximation of Equation 3.68 is also shown in Figure 3.8.

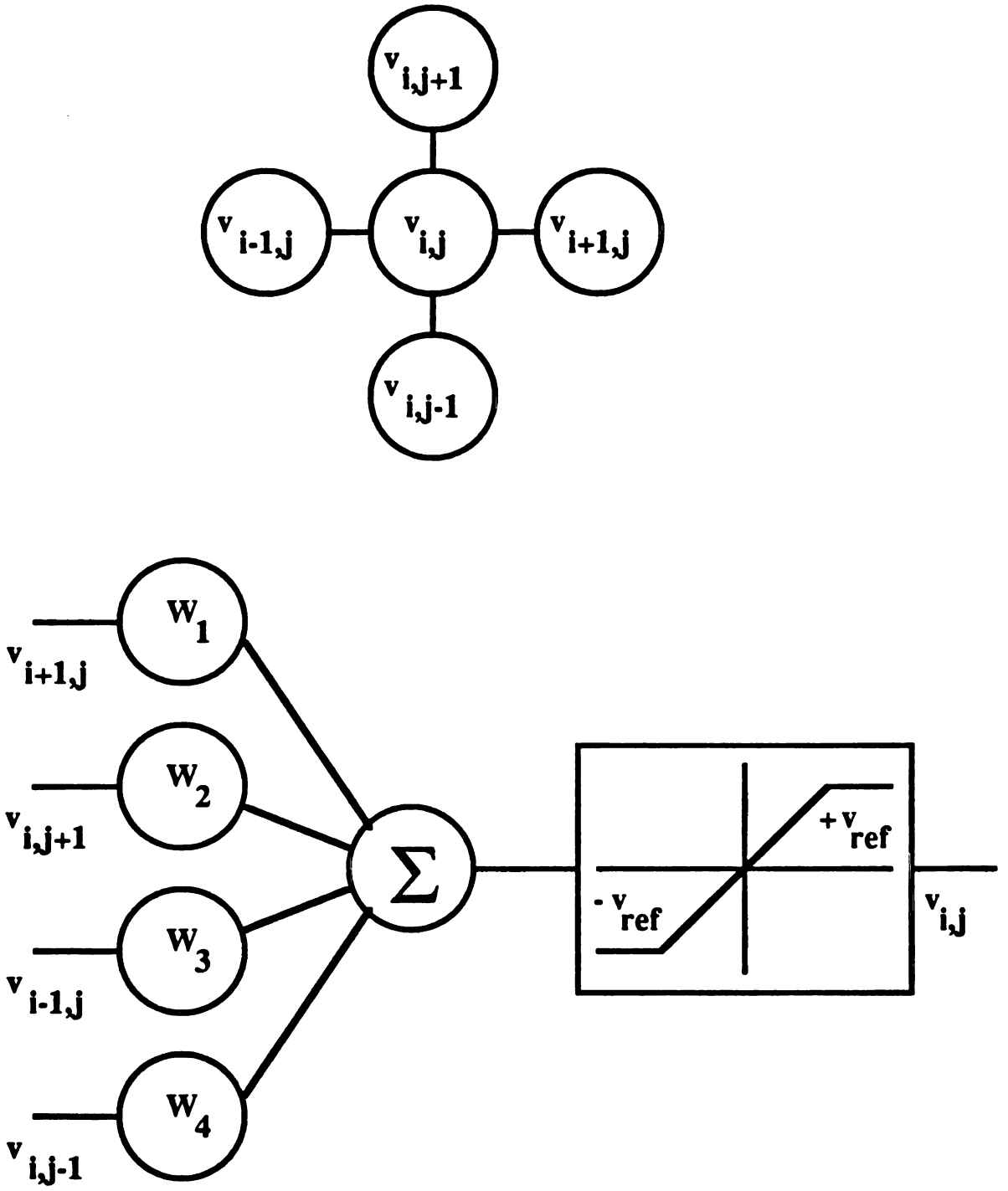

Figure 3.8: *Neuron Model for the 5-pt. Finite Difference Approximation*

An artificial neural network, as defined by the model, uses an electronic circuit (see Figure 3.9) as its basic neuron model, with the interconnecting resistors representing the synapses between the neurons. The numerical approximation computes the solution to a continuous system of equations (the implicit difference equations) and solves Equation 3.64. The circuit equation for the inverting operational amplifier implementation of the neuron is

$$V_o = -R_f \sum_{i=1}^{n} V_i / R_i \tag{3.69}$$

where V_i is the output voltage of neuron i, limited to the range $[-Vref, +Vref]$, R_f is the feedback resistor for the neuron operational amplifier and R_i is the interconnecting resistor from neuron i. Thus the synaptic weight W_i is implemented as the ratio of resistances: $-R_f/R_i$.

The neuron model is defined as consisting of an input function and output function. The input function computes the weighted sum of the interconnections from the other neurons,

$$u_o = \sum_{i=1}^{n} W_i V_i \tag{3.70}$$

and the output function computes the linear ramp between the saturation voltages $[-Vref, +Vref]$ of the operational amplifier.

The neural network is used to perform a parallel computation of the scalar potential field ϕ at every spatial point. Thus, the network computes a numerical solution to Equation 3.13. The architecture of the neural network must solve an elliptic partial differential equation. There are many numerical methods for solving partial differential equations. In this research, finite differences were employed to

solve the potential field Equation 3.13. Such an approximation is the five-point formula defined in Equation 3.64. For the neural network implementation, the neuron input function $u_{i,j}$ is defined as

$$u_{i,j} = \frac{1}{4}\left[(1+\frac{\sigma_x}{2\sigma})v_{i+1,j} + (1+\frac{\sigma_y}{2\sigma})v_{i,j-1} + (1-\frac{\sigma_x}{2\sigma})v_{i-1,j} + (1-\frac{\sigma_y}{2\sigma})v_{i,j-1}\right] \tag{3.71}$$

The synaptic weights implement the non-uniformity of the cost functions (the coefficients of Equation 3.71 which contain σ_x and σ_y). The weights to neuron(i,j) are shown in Table 3.2. The non-linear neuron output function $v_{i,j}$ is defined as

$$v_{i,j} = \begin{cases} -V_{ref} & : \ u_{i,j} < -V_{ref} \\ u_{i,j} & : \ -V_{ref} \leq u_{i,j} \leq +V_{ref} \\ +V_{ref} & : \ u_{i,j} > +V_{ref} \end{cases} \tag{3.72}$$

The entire search space is constructed by replicating this portion of the neural network over the entire grid. The output of the source and sink neurons which correspond to the start and goal nodes are clamped to $+V_{ref}$ and $-V_{ref}$, respectively. Since the clamping of the source and sink to the maximum and minimum (respectively) potential value occurs on the boundary of the problem, it is appropriate to use these clamped values as the limits of the neuron output function (Equation 3.72), since all potential values inside the boundary of the problem are guaranteed to lie between these limits.

Table 3.2: *Neural Network Weight Definitions for the 5-pt. Approximation*

Weight	From Neuron	Value
W_1	$(i+1,j)$	$(1+\sigma_x/2\sigma)/4$
W_2	$(i,j+1)$	$(1+\sigma_y/2\sigma)/4$
W_3	$(i-1,j)$	$(1-\sigma_x/2\sigma)/4$
W_4	$(i,j-1)$	$(1-\sigma_y/2\sigma)/4$

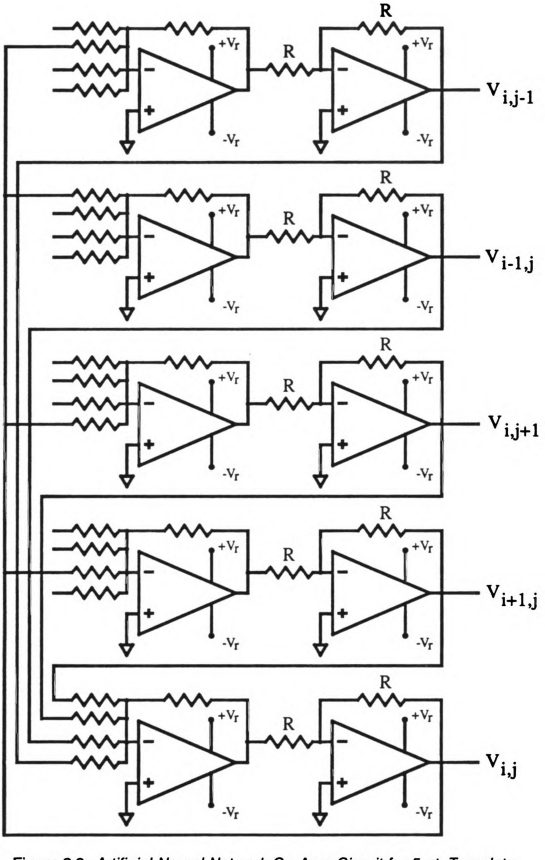

Figure 3.9: *Artificial Neural Network Op-Amp Circuit for 5-pt. Template*

3.3.3 Extracting Paths from the Neural Network Architecture

The paths are extracted from the neural network once it settles. The network computes the scalar potential field, as described previously. The final values of the nodes represent a surface with a global maximum at the start node and a global minimum at the goal node. A direction is selected for a path at the start node. Gradient descent over the potential surface is used to extract the path for that direction. Bivariate interpolation of the potential values is used between the grid points. Figure 3.10 illustrates the gradient descent algorithm. The algorithm computes the new location, new accumulated cost, and new gradient angle. The incremental path length is given as S. Each pass through the algorithm generates another point along the path. This algorithm is repetatively invoked until the entire path from the start node to the goal node is extracted from the gradient descent of the potential surface. This will have a complexity of $O(L)$, where L is the length of the path.

```
1.    begin GRADIENT-DESCENT
2.        real constant S : incremental path length;
3.        real C_x, C_y : current location;
4.        real C_A : accumulated cost;
5.        real G : gradient angle;
6.        I_c ⟵ ⌊C_x⌋;
7.        J_c ⟵ ⌊C_y⌋;
8.        if I_c = N then
9.            I_c ⟵ I_c − 1;
10.       endif
11.       if J_c = N then
12.           J_c ⟵ J_c − 1;
13.       endif
14.       X_N ⟵ C_x + S cos G;
15.       Y_N ⟵ C_y + S sin G;
16.       I_N ⟵ ⌊X_N⌋;
17.       J_N ⟵ ⌊Y_N⌋;
18.       C_c ⟵ COST(I_c, J_c);
19.       C_N ⟵ COST(I_N, J_N);
20.       Δ_C ⟵ S (C_c + C_N)/2;
21.       C_A ⟵ C_A + Δ_C;
22.       x_p ⟵ C_x − I_C;
23.       y_p ⟵ C_y − J_C;
24.       f_y^p ⟵ BIVARIATE(x_p, y_p, f_y, I_c, J_c);
25.       f_x^p ⟵ BIVARIATE(x_p, y_p, f_x, I_c, J_c);
26.       G ⟵ arctan (f_y^p / f_x^p);
27.       C_x ⟵ X_n;
28.       C_y ⟵ Y_n;
29.   end GRADIENT-DESCENT;
```

Figure 3.10: *Algorithm to Extract Path Points from the Potential Surface*

3.3.4 Variable Connectivity for the Architecture

In this section, the development of a method to implement a variable connectivity of the architecture is developed. The concept of variable connectivity means that rather than solving the partial differential equation with each node of the grid using inputs from only four immediate neighboring nodes, additional nodes are connected. The assumption is that increased connectivity will lower the convergence time of the architecture. This increase in architecture complexity should not outweigh the benefits obtained in convergence time, or accuracy of the results.

The Taylor-series based approach of Section 3.3.1 was used to develop template definitions for increasing sizes of templates. This was an intuitive approach for increased connectivity. The approach had a sound mathematical basis for defining approximation templates. Given a template pattern, it seemed straightforward to enlarge the pattern from five points to templates with nine, 13, or 25 points. However, this approach failed to provide the accuracy and convergence results which were anticipated. Increasing connectivity by using larger templates actually decreased the convergence rate. Section 4.3 gives the detailed results of this approach and discusses why it failed to produce the anticipated improvements in convergence rate.

A second method to accomplish the desired variable connectivity in the artificial neural network is now developed. A parallel summation technique using the 5-point finite difference approximation is developed which is based on the Euler method of integration.

Unidirectional Parallel Summation

Considering one dimension, a first-order differential equation can be solved numerically using the Euler method. The Euler method can be viewed as a recurrence relation. In expanding the recurrence relation for each successive solution point, all of the summations arrived at by the recurrence relation can be seen. Considering the terms of the summation for each solution point, and comparing to the total summation of previous solution points, previous solution points for these terms can be substituted in a systolic fashion. There are many ways to partition these summation terms, which in effect varies the parallelism of the architecture and thus the connectivity of the neural network. This section defines the equations and illustrates architectures of this approach. A good approximation is retained over the entire problem space, because the fundamental approximation to the derivative is unchanged. The variable connectivity is accomplished by collecting computations of the numerical approximation into different groupings which may be implemented in parallel without altering the basic approximation to the derivative. The connectivity is variable in a straightforward manner, with computational speed results apparent from the inherent serial or parallel combinations of the architecture. These connection equations (summation formula) are developed from first-order one-dimensional to two-dimensional for their use in the simulation.

Consider the differential equation

$$\frac{d}{dx}f(x) = g(x), \quad \text{with initial condition } f(x_0) = f_0 \tag{3.73}$$

Substitute the numerical approximation for the derivative and obtain

$$\frac{f(x_{i+1}) - f(x_i)}{\Delta x_i} = g(x_i) \tag{3.74}$$

Solving for $f(x_{i+1})$ results in the Euler forward integration equation[64]

$$f(x_{i+1}) = f(x_i) + \Delta x_i g(x_i) \tag{3.75}$$

Let $h = \Delta x_i$ be used as the grid size in a parallel architecture.

Recursive application of Equation 3.75 can be used to solve for any value $f(x_n)$:

$$\begin{aligned} f(x_0) &= f_0 \\ f(x_1) &= f(x_0) + hg(x_0) \\ f(x_2) &= f(x_1) + hg(x_1) \\ &\vdots \\ f(x_n) &= f(x_{n-1}) + hg(x_{n-1}) \end{aligned}$$

This concept is illustrated in Figure 3.11. If the definition of the $f(x_i)$ terms on the right hand side of the above set of equations are expanded, this gives

$$\begin{aligned} f(x_0) &= f(0) \\ f(x_1) &= f(0) + hg(x_0) \\ f(x_2) &= f(0) + h\,[g(x_0) + g(x_1)] \\ f(x_3) &= f(0) + h\,[g(x_0) + g(x_1) + g(x_2)] \end{aligned}$$

For the nth term this is

$$f(x_n) = f(0) + \sum_{i=0}^{n-1} hg(x_i) \tag{3.76}$$

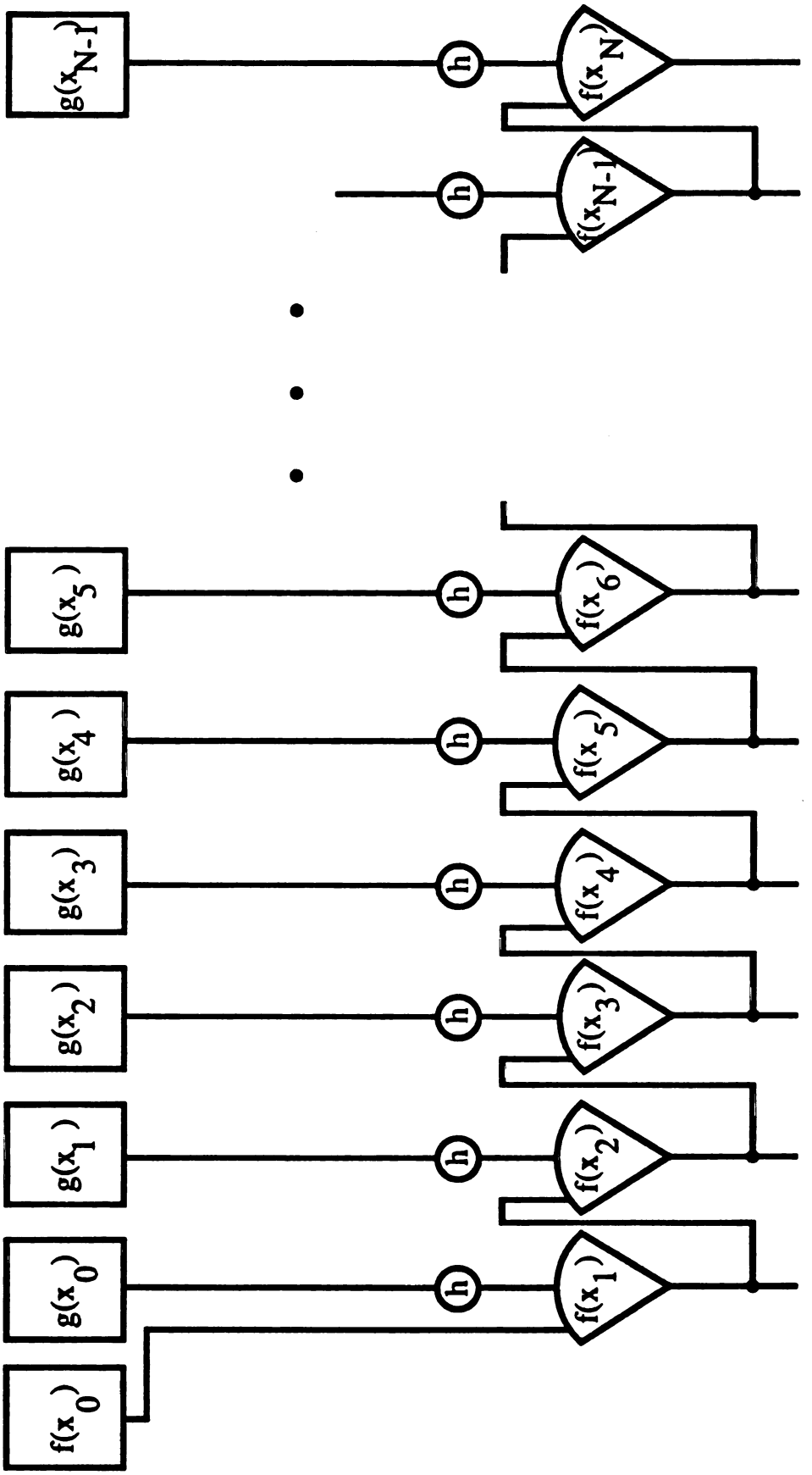

Figure 3.11: *Spatial Illustration of the Euler Integration Concept*

The summation in Equation 3.76 can be split into two summations:

$$f(x_n) = f(0) + \sum_{i=0}^{n-k-1} hg(x_i) + \sum_{j=n-k}^{n-1} hg(x_j) \tag{3.77}$$

The first two terms of Equation 3.77 is $f(x_{n-k})$, so this becomes

$$f(x_n) = f(x_{n-k}) + \sum_{j=n-k}^{n-1} hg(x_j) \tag{3.78}$$

The connectivity will be shown later to be the fan-in $K = k + 1$ to the neuron. Equation 3.78 gives a definition for the *variable connectivity* in the parallel summation for the Euler integration. This variable connectivity encompasses varying degrees of parallelism via the fan-in, and has the effect of varying the amount of serial delay in computing its results. The first term of Equation 3.78 is the serially cascaded computation. The second term is the parallel (in addition to the serial input) computation. Of course, it will have a valid output only after the cascaded serial summation delay. The connectivity is derived as follows. Equation 3.78 is spatially constructed for parallel summations. So how many summations are there? The last term has $(n - 1) - (n - k) + 1$ addends, for a total of k inputs to the parallel summer. Including the input for the first term, this is $k + 1 \equiv K$ total inputs to the summer.

Use of Equation 3.78 for specifying the connectivity of the network architecture will be illustrated. Figure 3.12 contains the recursive equations and Figure 3.13 shows the corresponding interconnections used with various configurations to show how adjusting the fan-in results in a unique architecture. This set of illustrations uses ten points of the function ($n = 10$).

$$f(x_1) = f(0) + h \cdot g(x_0)$$

$$f(x_2) = f(0) + h \cdot g(x_0) + h \cdot g(x_1)$$

$$f(x_3) = f(0) + h \cdot g(x_0) + h \cdot g(x_1) + h \cdot g(x_2)$$

$$f(x_4) = f(0) + h \cdot g(x_0) + h \cdot g(x_1) + h \cdot g(x_2) + h \cdot g(x_3)$$

$$f(x_5) = f(0) + h \cdot g(x_0) + h \cdot g(x_1) + h \cdot g(x_2) + h \cdot g(x_3) + h \cdot g(x_4)$$

$$f(x_6) = f(0) + h \cdot g(x_0) + h \cdot g(x_1) + h \cdot g(x_2) + h \cdot g(x_3) + h \cdot g(x_4) + h \cdot g(x_5)$$

$$f(x_7) = f(0) + h \cdot g(x_0) + h \cdot g(x_1) + h \cdot g(x_2) + h \cdot g(x_3) + h \cdot g(x_4) + h \cdot g(x_5) + h \cdot g(x_6)$$

$$f(x_8) = f(0) + h \cdot g(x_0) + h \cdot g(x_1) + h \cdot g(x_2) + h \cdot g(x_3) + h \cdot g(x_4) + h \cdot g(x_5) + h \cdot g(x_6) + h \cdot g(x_7)$$

$$f(x_9) = f(0) + h \cdot g(x_0) + h \cdot g(x_1) + h \cdot g(x_2) + h \cdot g(x_3) + h \cdot g(x_4) + h \cdot g(x_5) + h \cdot g(x_6) + h \cdot g(x_7) + h \cdot g(x_8)$$

$$f(x_{10}) = f(0) + h \cdot g(x_0) + h \cdot g(x_1) + h \cdot g(x_2) + h \cdot g(x_3) + h \cdot g(x_4) + h \cdot g(x_5) + h \cdot g(x_6) + h \cdot g(x_7) + h \cdot g(x_8) + h \cdot g(x_9)$$

Figure 3.12: *Parallel Summation Equations Used in Variable Connectivity Analysis*

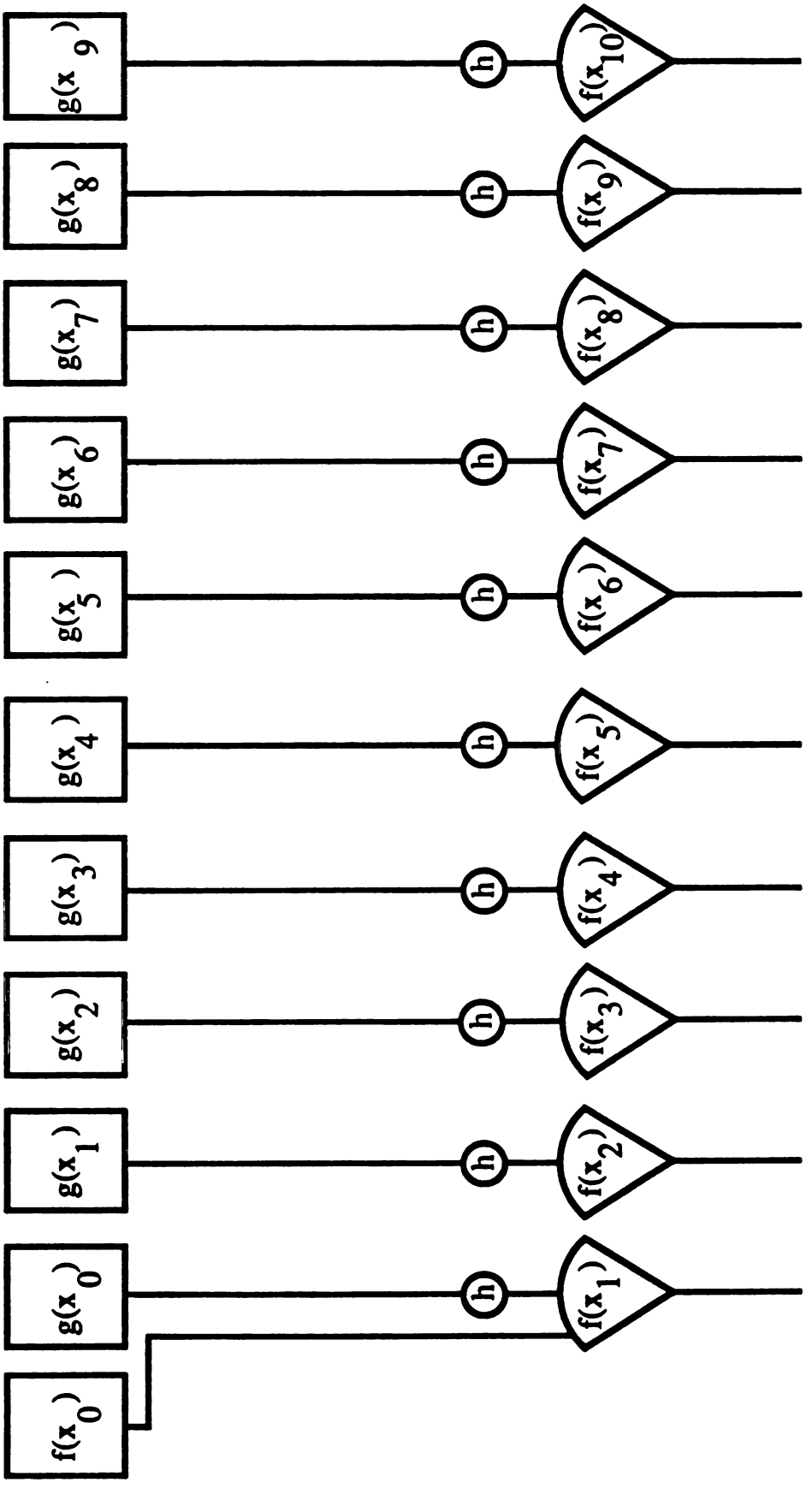

Figure 3.13: *Components of the Parallel Summation Architecture*

Figure 3.14 illustrates the summation equation with a fan-in of 2, which is

$$f(x_n) = f(x_{n-1}) + hg(x_{n-1}) \tag{3.79}$$

Each output on the left side of the equations in the figure picks up two terms, the previous output term, and the last input term on the right hand side of the equations. This architecture, as shown in Figure 3.15 is parallel in space, but serial in time. It has the minimum number of connections, and the maximum time to settle because of the cascaded ripple wave computations.

The next illustration is Figure 3.16 for the fan-in of 3. Its equation is

$$f(x_n) = f(x_{n-2}) + \sum_{i=n-2}^{n-1} hg(x_i) \tag{3.80}$$

Each output on the left side of the equations of the figure picks up three terms, the second output term preceeding this output, and the last two input terms on the right hand side. This architecture shown in Figure 3.17 is the next step in adding more connectivity, which has a corresponding decrease in the time required for the cascaded serial computational wave.

Figure 3.18 illustrates the next step with a fan-in of 4. The corresponding equation is

$$f(x_n) = f(x_{n-3}) + \sum_{i=n-3}^{n-1} hg(x_i) \tag{3.81}$$

The output term on the left side now picks up four terms. The serial term which was picked up preceeds the current term by three delay steps in the cascade addition. The last three terms of the right hand side are picked up in the summation. Figure 3.19 illustrates this architecture.

In Figure 3.20 the fan-in for the complete connectivity of $n + 1$ is shown. The equation describing this is

$$f(x_n) = f(0) + \sum_{i=0}^{n-1} hg(x_i) \tag{3.82}$$

Here no serial terms are picked up from the cascade ripple wave. All of the input terms are from the right hand side of the equation. This architecture is parallel in space, and also parallel in time. This has the maximum number of connections and minimum time because there is no ripple wave for the computation due to no cascading structure. The architecture is shown in Figure 3.21.

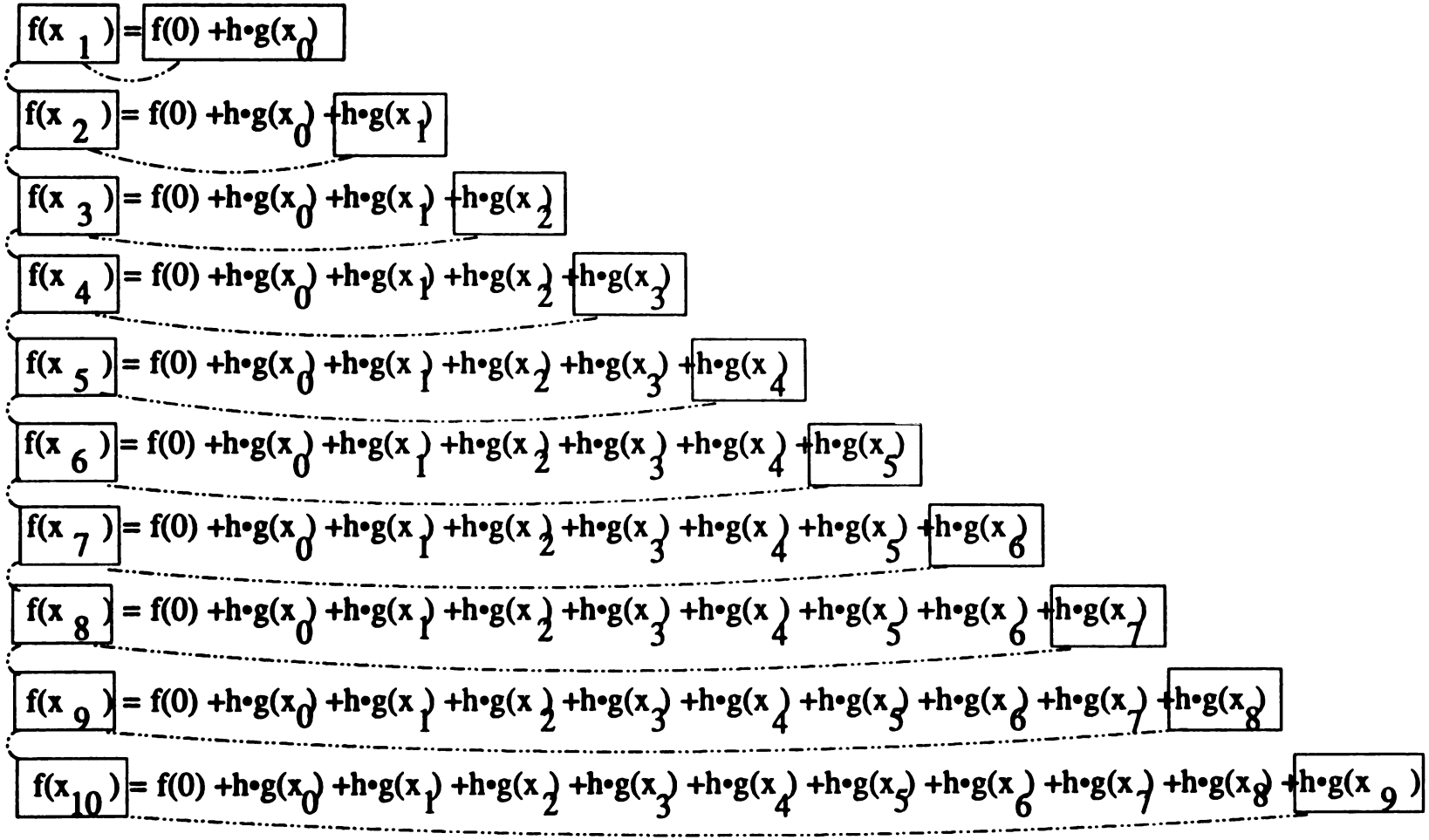

Figure 3.14: *K=2 Connectivity of the Parallel Summation Equations*

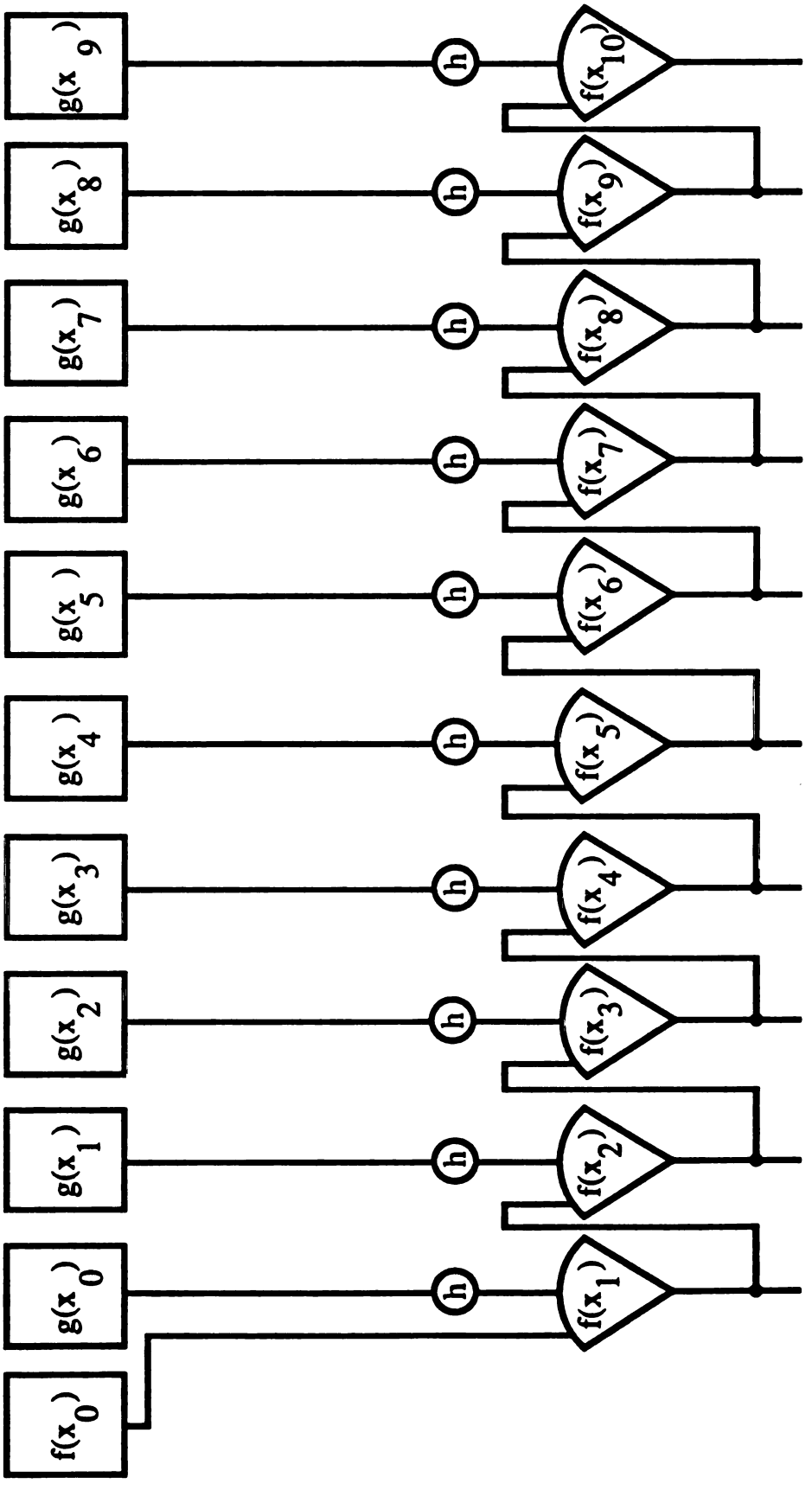

Figure 3.15: *K=2 Interconnections for the Parallel Summation Architecture*

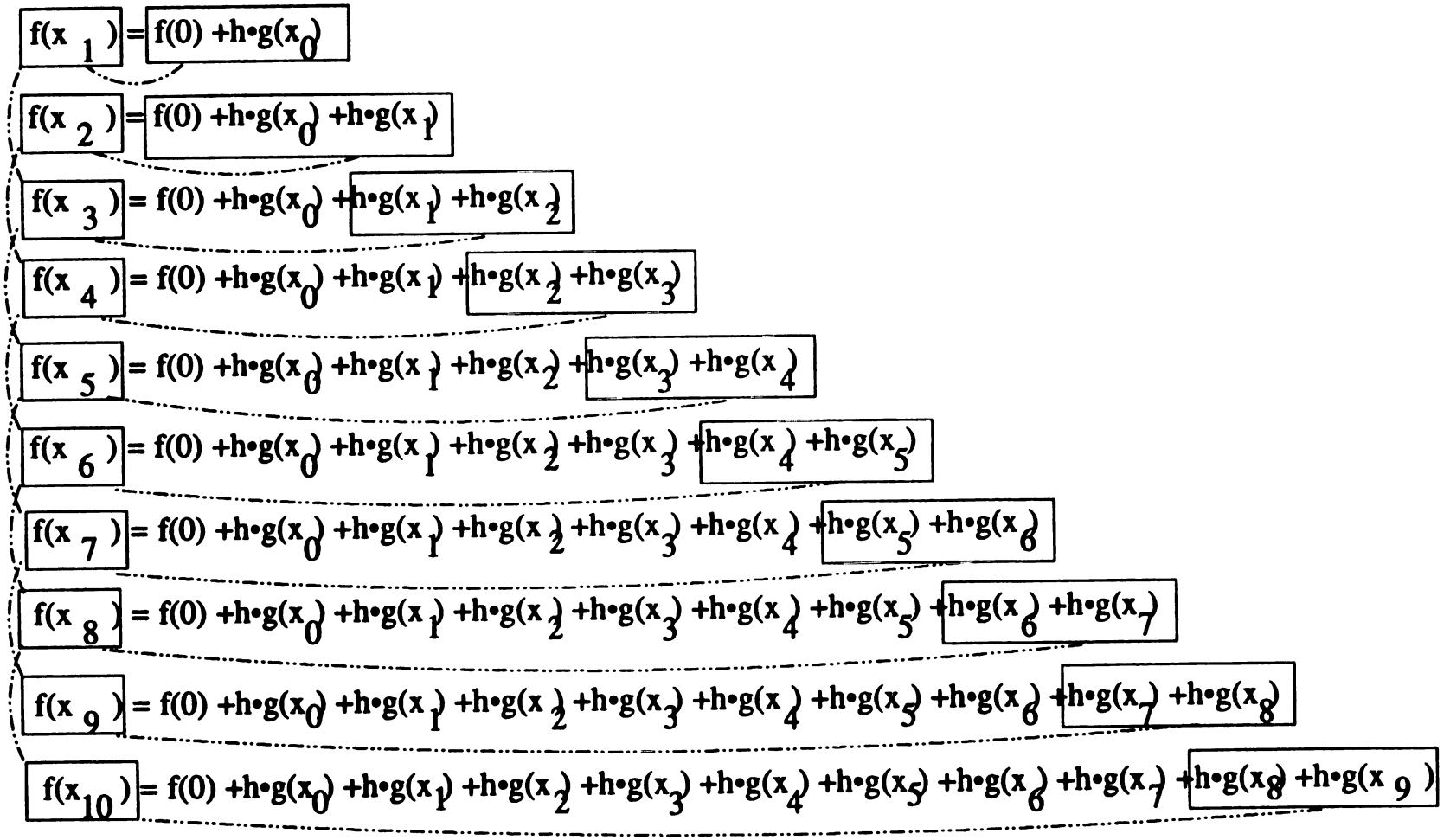

Figure 3.16: *K=3 Connectivity of the Parallel Summation Equations*

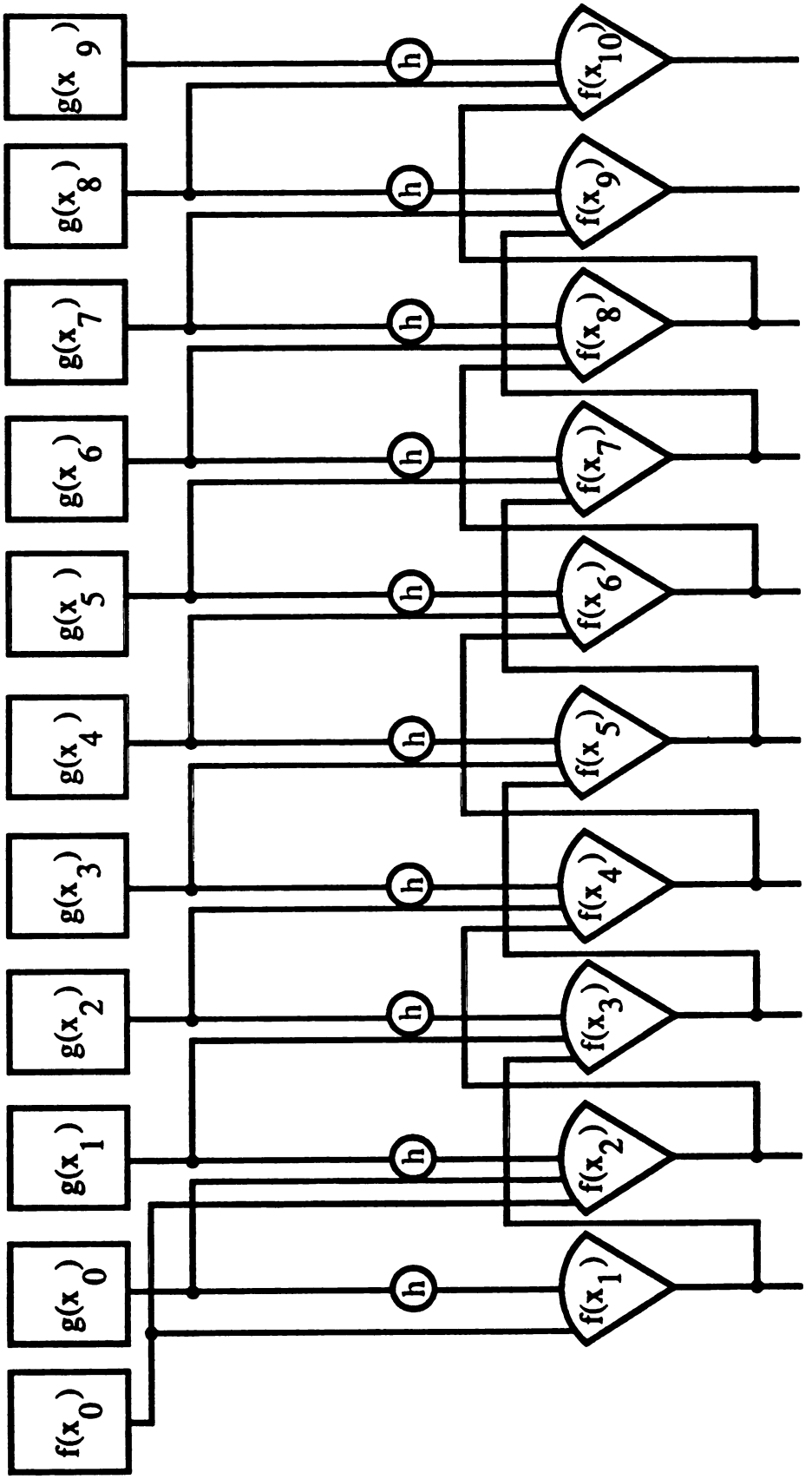

Figure 3.17: *K=3 Interconnections for the Parallel Summation Architecture*

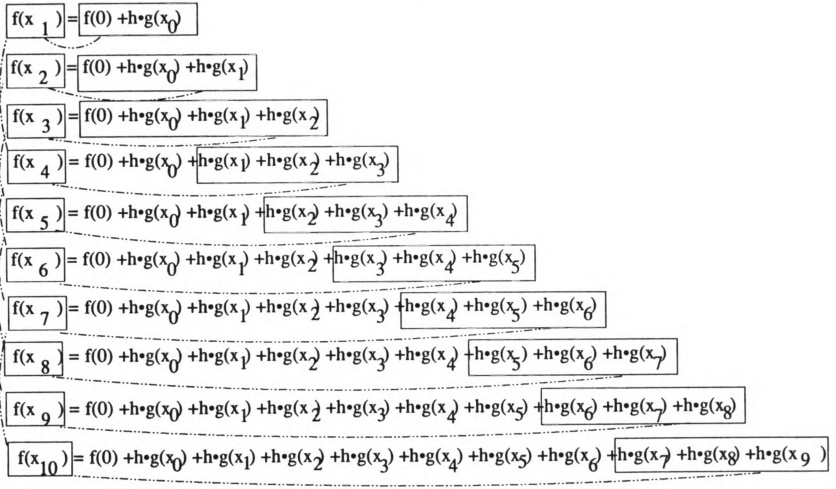

Figure 3.18: *K=4 Connectivity of the Parallel Summation Equations*

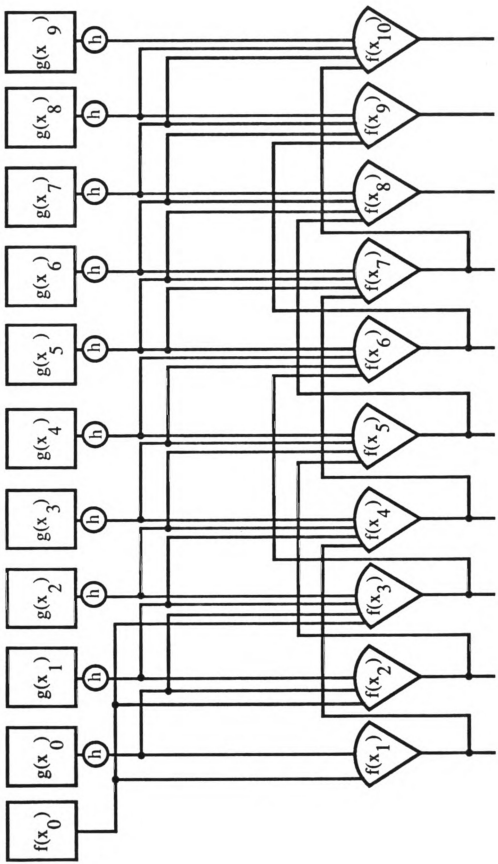

Figure 3.19: *K=4 Interconnections for the Parallel Summation Architecture*

Figure 3.20: *K = n + 1 Connected Grouping of the Parallel Summation Equations*

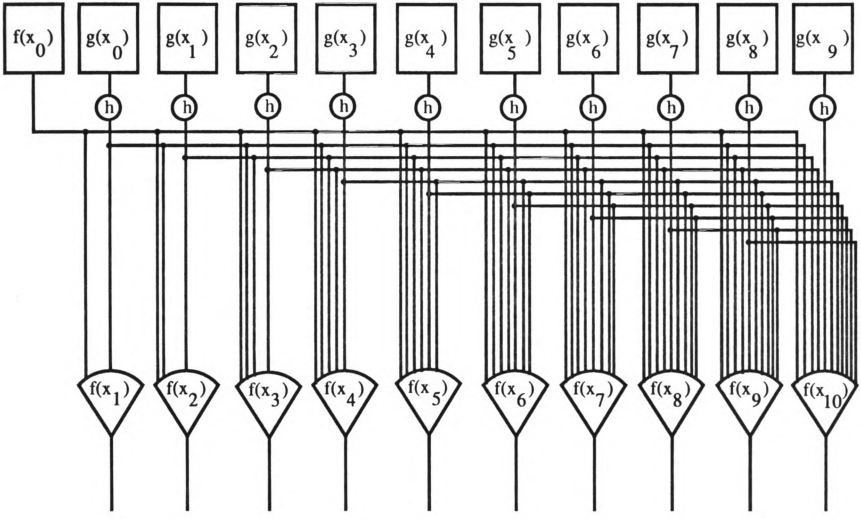

Figure 3.21: *K = n + 1 Interconnection for the Parallel Summation Architecture*

Bidirectional Parallel Summation

This section develops the next step in the construction of the parallel summation architecture. Since the objective of the parallel summation architecture is to have a variable amount of connections for a multidimensional problem space, the summation technique must not have a unidirectional flow to the computation. This requirement is necessary because the local connection architecture must be robust enough to support the processing of data arriving from any direction within the architecture. Therefore a bidirectional parallel summation is needed to eliminate the unidirectional computation flow of the previous architecture.

To develop a bi-directional formula, the backward difference formula first needs to be derived. Begin again with the differential equation

$$\frac{d}{dx}f(x) = g(x), \quad \text{with initial condition } f(x_{N+1}) = f_{N+1} \tag{3.83}$$

The backward difference approximation to Equation 3.83 is

$$\frac{f(x_{i+1}) - f(x_i)}{\Delta x_i} = g(x_{i+1}) \tag{3.84}$$

Letting $h = \Delta x$ be the grid size, and solving Equation 3.84 for $f(x_i)$ gives

$$f(x_i) = f(x_{i+1}) - hg(x_{i+1}) \tag{3.85}$$

The general expression then is

$$f(x_j) = f(x_{j+k}) - \sum_{i=j+1}^{j+k} hg(x_i) \tag{3.86}$$

The architecture for this equation is shown in Figure 3.22.

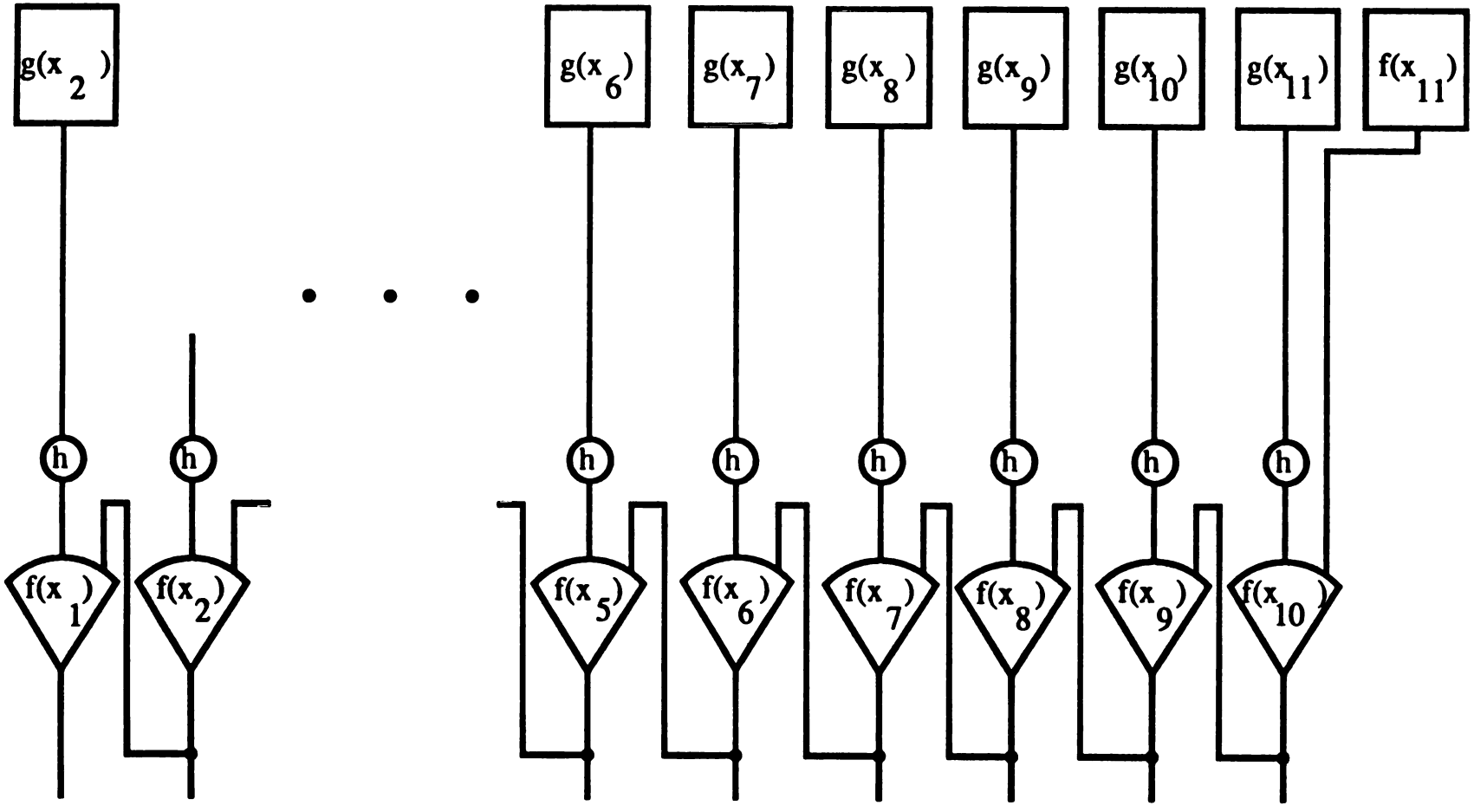

Figure 3.22: *Illustration of Backward Difference Parallel Summation Architecture*

As should be expected, this result compared with Figure 3.11 is seen to be inverted along the single dimension.

Now a central difference formula is derived by combining the forward and backward difference formulas.

Recall the forward difference expression as

$$f(x_j) = f(x_{j-k}) + \sum_{i=j-k}^{j-1} hg(x_i) \tag{3.87}$$

and the backward difference is Equation 3.86. Adding these together and solving for $f(x_j)$ yields

$$f(x_j) = \frac{f(x_{j+k})}{2} + \frac{f(x_{j-k})}{2} + \sum_{i=j-k}^{j-1} \frac{h}{2} g(x_i) - \sum_{i=j+1}^{j+k} \frac{h}{2} g(x_i) \tag{3.88}$$

Figure 3.23 shows the architecture for Equation 3.88 with $k = 1$.

There is a pattern to the increasing parallelism (k) in the summation equations and the architectural configuration. This is illustrated as shown in Figure 3.24 where the right side of the figure shows the pattern after increasing the parallel connections to the example on the left. When increasing the parallelism of the summation, the input to neuron C is increased. This is done by disconnecting the output O_B of neuron B, and adding connections from the input I_B of neuron B directly as additional inputs to neuron C. This will include connecting the output O_A of neuron A to the input of neuron C. Thus the pattern of increasing parallelism in the summation equations incrementally bypasses neurons by adding connections from their inputs directly to those neurons which they were connected to by their original output.

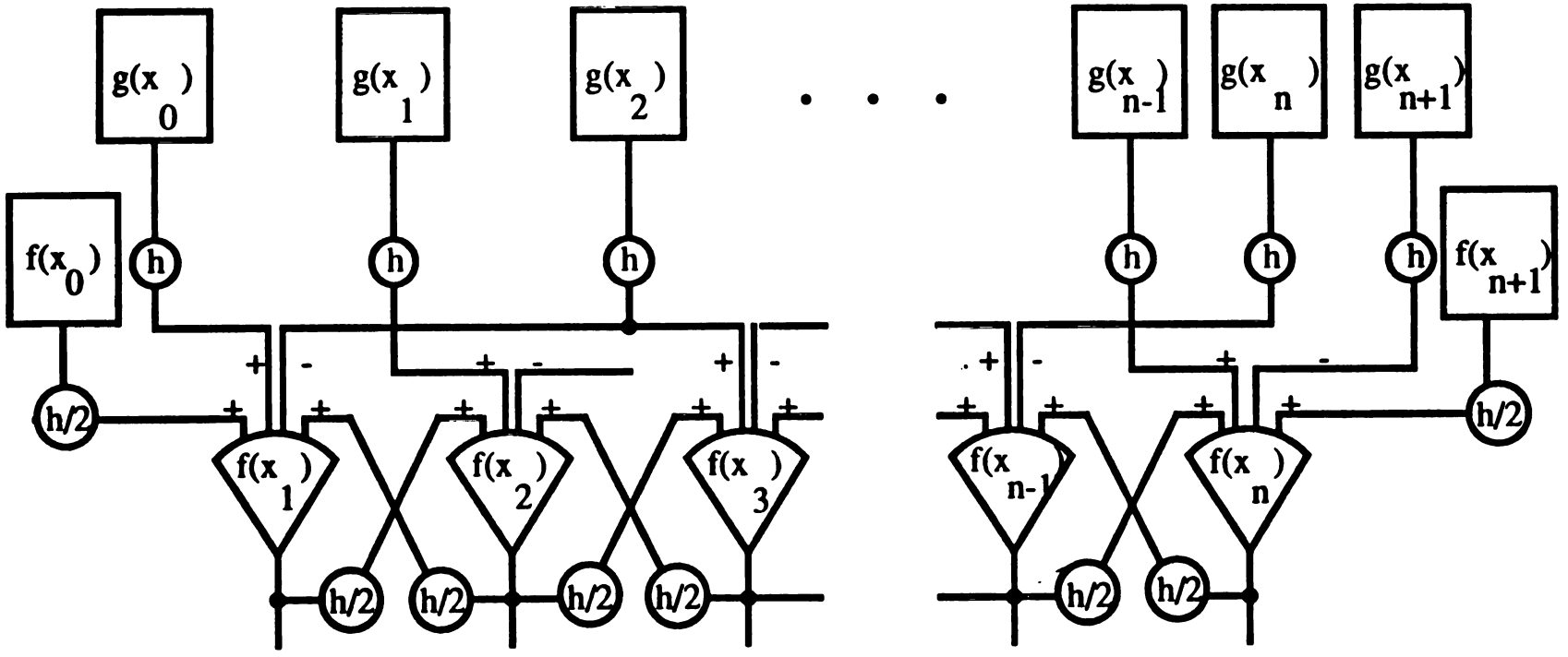

Figure 3.23: *Illustration of the Central Difference Parallel Summation Architecture*

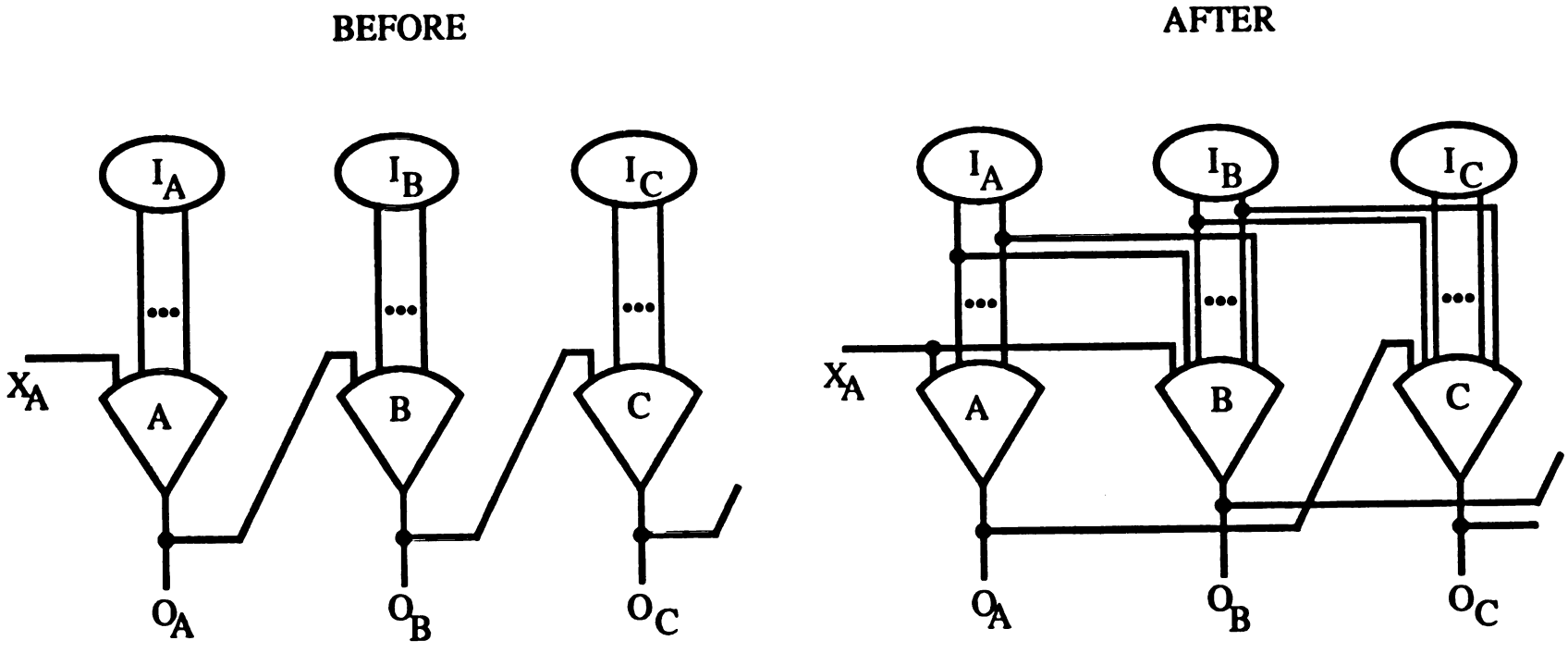

Figure 3.24: *Recursive Expansion of the Parallel Summation Architecture*

$$O_A = A(I_A, X_A) \tag{3.89}$$

$$O_B = B(I_B, O_A) \tag{3.90}$$

$$O_B = B(I_B, I_A, X_A) \tag{3.91}$$

$$O_C = C(I_C, O_B) \tag{3.92}$$

$$O_C = C(I_C, I_B, O_A) = C(I_C, B(I_B, O_A)) = C(I_C, O_B) \tag{3.93}$$

Thus, the recursive expansion by substitution. Figure 3.25 illustrates the architecture of this observation for the central difference formula.

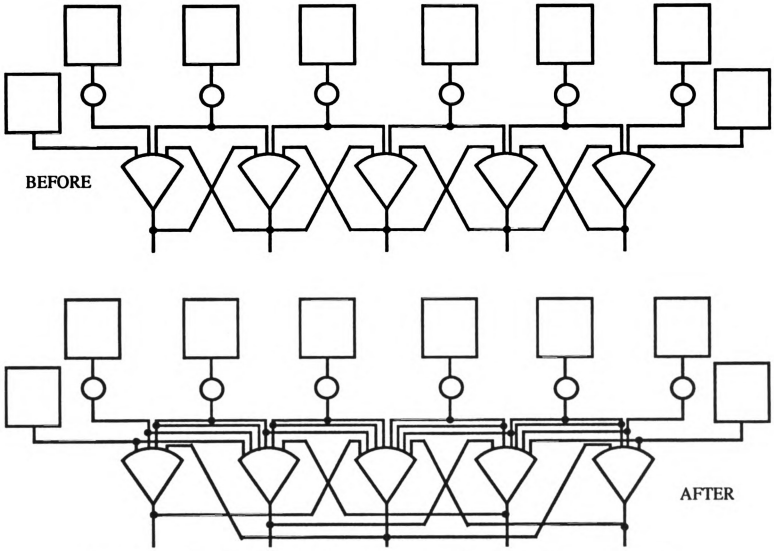

Figure 3.25: *Expansion of Central Difference Approximation for Parallel Summation*

Two-dimensional Parallel Summation

With the development of the central difference formula for bidirectional parallel summation, and the observation of the recursive expansion, the final step in the construction of multidimensional parallel summation can be accomplished. Using the five point approximation template as a *kernel*, the template can be "unfolded" as many times as desired in a recurrent fashion, obtaining a larger connection neighborhood, which takes into account the local $C_x(x,y)$ and $C_y(x,y)$ in the solution. Different kernels generate different connection patterns.

Recalling the weight expressions for the model from Table 3.2, let them be written as the following equations

$$w_x^+(i,j) = \frac{1}{4}\left[1 + \frac{\sigma_x}{2\sigma}(i,j)\right] \tag{3.94}$$

$$w_x^-(i,j) = \frac{1}{4}\left[1 - \frac{\sigma_x}{2\sigma}(i,j)\right] \tag{3.95}$$

$$w_y^+(i,j) = \frac{1}{4}\left[1 + \frac{\sigma_y}{2\sigma}(i,j)\right] \tag{3.96}$$

$$w_y^-(i,j) = \frac{1}{4}\left[1 - \frac{\sigma_y}{2\sigma}(i,j)\right] \tag{3.97}$$

The general equation for the neuron input function is

$$u(i,j) = w_x^+(i,j)v(i+1,j) + w_x^-(i,j)v(i-1,j) + w_y^+(i,j)v(i,j+1) + w_y^-(i,j)v(i,j-1) \tag{3.98}$$

which upon recursive expansion (for $k = 2$) is

$$
\begin{aligned}
u(i,j) &= w_x^+(i,j)[w_x^+(i+1,j)v(i+2,j) + w_x^-(i+1,j)v(i,j) \\
&+ w_y^+(i+1,j)v(i+1,j+1) + w_y^-(i+1,j)v(i+1,j-1)] \\
&+ w_x^-(i,j)[w_x^+(i-1,j)v(i,j) + w_x^-(i-1,j)v(i-2,j) \\
&+ w_y^+(i-1,j)v(i-1,j+1) + w_y^-(i-1,j)v(i-1,j-1)] \\
&+ w_y^+(i,j)[w_x^+(i,j+1)v(i+1,j+1) + w_x^-(i,j+1)v(i-1,j+1) \\
&+ w_y^+(i,j+1)v(i,j+2) + w_y^-(i,j+1)v(i,j)] \\
&+ w_y^-(i,j)[w_x^+(i,j-1)v(i+1,j-1) + w_x^-(i,j-1)v(i-1,j-1) \\
&+ w_y^+(i,j-1)v(i,j) + w_y^-(i,j-1)v(i,j-2)] \qquad (3.99)
\end{aligned}
$$

Figure 3.26 shows the first two expansions for a 5-point mesh architecture. An algorithm which implements the connection weights for any value of k is shown in Figure 3.27. In this manner, new connection weight values are defined. This expansion continues until the desired amount of connectivity is achieved. The maximum connectivity is obtained when the expansion for each node has recursively used all other nodes. This will scale directly with the length of the largest dimension of the multidimensional space.

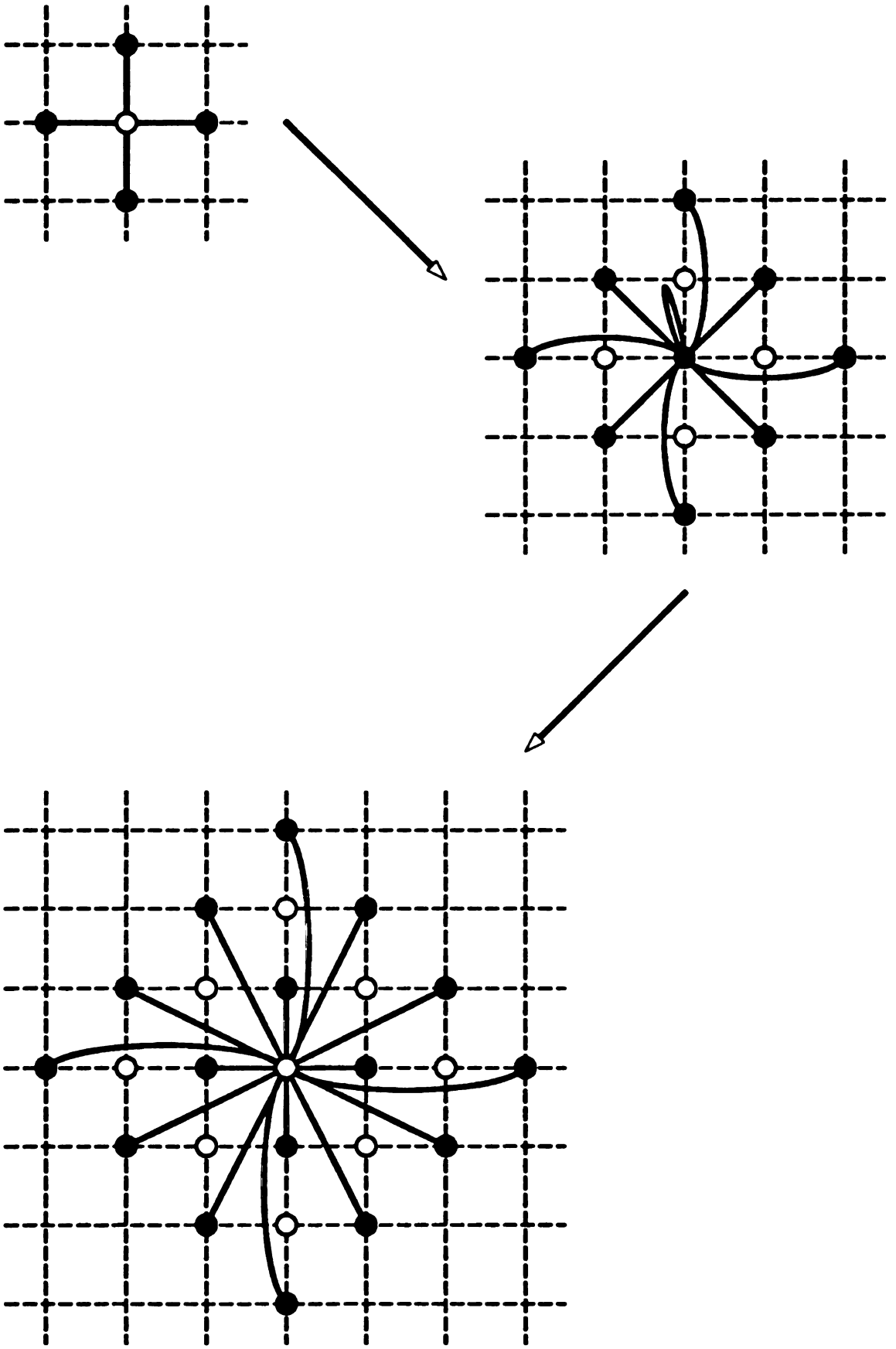

Figure 3.26: *Two Expansions of the 5-pt. Kernel Approximation Template*

```
1.    begin F(k,v,i,j)
2.        if k = 0 then
3.            F ⟵ v(i,j);
4.        else
5.            F_x^+ ⟵ w_x^+(i,j)F(k − 1,v,i + 1,j);
6.            F_x^− ⟵ w_x^−(i,j)F(k − 1,v,i − 1,j);
7.            F_y^+ ⟵ w_y^+(i,j)F(k − 1,v,i,j + 1);
8.            F_y^− ⟵ w_y^−(i,j)F(k − 1,v,i,j − 1);
9.            F ⟵ F_x^+ + F_x^− + F_y^+ + F_y^−;
10.       endif
11.   end
```

Figure 3.27: *Algorithm to Generate the Parallel Summation Neuron Weights*

3.4 Complexity Analysis of the Neural Network

The neural network computes a finite difference approximation, which can be expressed in matrix form. The matrix equation for the neural network is

$$\boldsymbol{u}^{(k+1)} = \boldsymbol{A}\boldsymbol{u}^{(k)} \tag{3.100}$$

where $\boldsymbol{u}^{(j)}$ is the j^{th} update to the $(n^2 \times 1)$ neuron vector

$$\boldsymbol{u} = \begin{bmatrix} u_{1,1} \\ u_{1,2} \\ \vdots \\ u_{1,n} \\ u_{2,1} \\ \vdots \\ u_{n,n} \end{bmatrix} \tag{3.101}$$

$\boldsymbol{A}$ is the tri-diagonal square $(n^2 \times n^2)$ matrix defined as

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{T} & \frac{1}{2}\boldsymbol{B} & & 0 \\ \boldsymbol{Y_N} & \boldsymbol{T} & \boldsymbol{Y_P} & \\ \ddots & \ddots & \ddots & \\ & & & \boldsymbol{Y_P} \\ 0 & & \frac{1}{2}\boldsymbol{B} & \boldsymbol{T} \end{bmatrix} \tag{3.102}$$

along with the square $(n \times n)$ matricies

$$\boldsymbol{B} = \boldsymbol{I} \tag{3.103}$$

$$\boldsymbol{T} = \begin{bmatrix} 0 & 1/2 & & 0 \\ (1 - \frac{C_x}{2})/4 & 0 & (1 + \frac{C_x}{2})/4 & \\ \ddots & \ddots & \ddots & \\ & (1 - \frac{C_x}{2})/4 & 0 & (1 + \frac{C_x}{2})/4 \\ 0 & & 1/2 & 0 \end{bmatrix} \tag{3.104}$$

$$\boldsymbol{Y_P} = \begin{bmatrix} (1 + \frac{C_y}{2})/4 & & & 0 \\ & (1 + \frac{C_y}{2})/4 & & \\ & & \ddots & \\ 0 & & & (1 + \frac{C_y}{2})/4 \end{bmatrix} \tag{3.105}$$

$$\boldsymbol{Y}_N = \begin{bmatrix} (1-\frac{C_x}{2})/4 & & & 0 \\ & (1-\frac{C_x}{2})/4 & & \\ & & \ddots & \\ 0 & & & (1-\frac{C_x}{2})/4 \end{bmatrix} \tag{3.106}$$

Each iteration of Equation 3.100 requires $O(n^4)$ operations on a Von Neumann processor, but only a single cycle on the neural network. The neural network implements the five point finite difference approximation as a point iterative Jacobi[39] algorithm (all points are computed simultaneously in parallel, however). Lapidus and Pinder show[39] that the rate of convergence of the iterations of Equation 3.100 is inversely proportional to the square of the discretization of the grid $(1/n)$. Therefore, the convergence of the neural network approximation is $O(n^2)$. The complexity of the PPPM algorithm is then

$$O(n^2) + O(\text{all paths}) = O(n^2) + \text{no. of paths} \cdot O(\text{path length}) \tag{3.107}$$

The O(path length) is $O(n^2)$ since the worst case length includes all nodes of the neural network. The number of paths computed is linear in heading at the start node, so the overall complexity of the algorithm PPPM is $O(n^2)$.

3.5 Summary

This chapter has developed the details of a massively parallel architecture for path planning. Electrostatic field theory was used to model the path planning problem. Multiple path solutions are provided by this model through the field lines which describe the distribution of current. Numerical approximation provided the means to directly map the solution of the Laplacian partial differential equation of the model to a massively parallel architecture through a finite

difference approximation. An artificial neural network was designed to compute the finite difference approximation by using the approximation template to define the synaptic weights of the network. A parallel summation technique was developed which provided for a variable connectivity in the neural network. This technique recursively applies the approximation template to define the synaptic weights of additional connections in the neural network. Finally, the complexity of the neural network architecture was shown to be $O(n^2)$ for a network with n^2 neurons.

Chapter 4

Path Planning Application Results

In Chapter 3, an architecture design was developed for the electrostatic model of path planning. The architecture used a neural network computing model to solve the partial differential equation for the electrostatic model. The neural network was simulated to examine the usefulness of this approach to path planning. In this chapter, the results of the simulation runs are discussed.

Experimental verification of the solution and performance characteristics of the massive parallel architectures was accomplished using simulations on a VAX and a Connection Machine. The results and performance of the neural network which implements the electrostatic model is reviewed, including the simulations on the Connection Machine. The solutions generated by the massively parallel architecture have been checked against known admissible algorithm results.

The effects of variable connectivity on the results of the massively parallel architecture is also discussed. The amount of interconnection between computational units in the massively parallel architecture affects the accuracy and convergence of the network solution. It is shown that as the amount of connectivity in the network increases, the convergence rate improves.

The results using the electrostatic model show an average difference of eight percent in computing the best cost paths relative to sequential best first search. Connectivity curves are included which show the convergence of the Taylor series based templates and the parallel summation technique templates at various degrees of connectivity in the neural network. These curves show that the parallel summation technique templates converge faster with higher connectivity. The analog circuit simulation on the 16k CM-2 Connection Machine resulted in a real-time performance projection of 18 milliseconds for Test Case B whose neural network had 3,600 neurons.

4.1 Test Case Descriptions

Several test cases were used to experiment with the architecture simulations in this research. Table 4.1 lists the features of each of the test cases. The column

Table 4.1: *Path Endpoints and Optimal Costs of the Test Cases*

CASE	N: Size	S: Start	G: Goal	P*: Optimal Cost
A	30	(3, 23)	(17, 2)	0.33
B	60	(20, 10)	(45, 55)	5.34
C	50	(8, 42)	(37, 11)	0.71
D	65	(14, 35)	(50, 50)	0.45
E	21	(13, 5)	(11, 17)	18.14
F	21	(17, 5)	(5, 17)	17.56
G	26	(3, 4)	(24, 24)	38.99

labelled "N" is the size of the square grid used for the test case. The total number of grid points in each test case is N^2. The next column, labelled "S", contains the coordinates of the start node on the grid. The column labelled "G" lists the coordinates of the goal node. To establish the minimum cost path and its cost

value as a benchmark against which the simulated architecture results could be compared, a branch-and-bound search for the test case with the specified start and goal node was conducted. The last column, labelled "P*" shows the value of the accumulated cost for the best path between S and G of each case.

An example of the cost functions for the test cases is displayed in Figure 4.1 which is the cost function for Test Case B. The cost function value is illustrated as the contour lines of a surface whose height corresponds to the function value. Contour plots of the cost function for all of the test cases are illustrated in Appendix A. A detailed description of each test case is included in the appendix. These test cases collectively contain contrasting features in the cost function. These contrasting features in the test cases included single versus multiple cost regions, discrete versus continuous cost regions, dense versus sparse cost regions, and synthetic versus real-world cost model regions. In addition to their contrasting features, the various sizes serve to investigate the scalability of the neural network architecture. Many neural network investigations have used at most 100 neurons in there test cases. This test set included networks with several thousand neurons.

4.2 Electrostatic Model Results

Table 4.2 shows the results of the electrostatic model neural network architecture on the test cases. The average difference for all of the test cases was 8.3 percent.

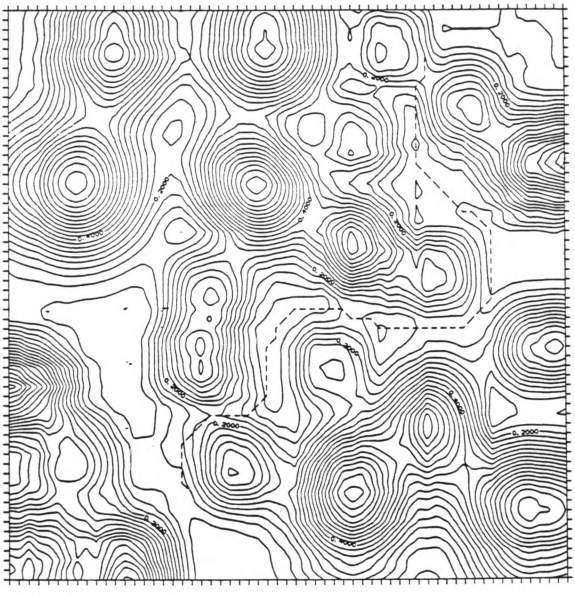

Figure 4.1: *Test Case B Showing a Cost Function Surface by Contours*

Table 4.2: *Test Case Results Using the 5-pt. Approximation Neural Network*

CASE	Network Cycles	P: Least Cost Path Value	% Difference from P*
A	4,000	0.33	0.0
B	3,483	7.67	+43.6
C	5,460	0.84	+18.3
D	6,767	0.43	-4.4
E	566	19.00	+4.7
F	370	18.00	+2.5
G	2,610	36.50	-6.4
Average			+8.3

4.2.1 Results Contrasted with Sequential Search

In examining the data, an obvious question is: How can there be a negative difference? This is due to differences in the method for generating the reference path and for extracting the path from the electrostatic model. The former used a branch-and-bound search using the nodes of the test case and their cost values. Recall that the gradient descent algorithm (Figure 3.10) for extracting paths used a fractional step in the direction of the gradient which resulted from a bivariate interpolation of the potential values at the neural network nodes.

This distinction is further illustrated by an example shown in Figure 4.2. In the figure, the discritization of the cost function is depicted by the nodes of the grid. The solid nodes are those which are included in the reference path found by the sequential search algorithm. This algorithm expands the nodes of the search space (grid) sequentially and the path is defined by connecting the nodes of the path with a straight line segment. This is shown as the solid line segments. In contrast, an example of the type of path generated by the gradient descent algorithm is illustrated as the dashed line in the figure. It is usually a much

smoother path since it uses a gradient step size which is often defined as a fraction of the grid discritization value.

The fractional step size used in the neural network simulations was 0.5 (one-half of the grid step defined by the test case nodes), so this resulted in a higher resolution of the path points extracted than those of the reference path. With a higher resolution, some of the extracted paths in the test cases had a lower cost than the reference path which resulted in a negative difference.

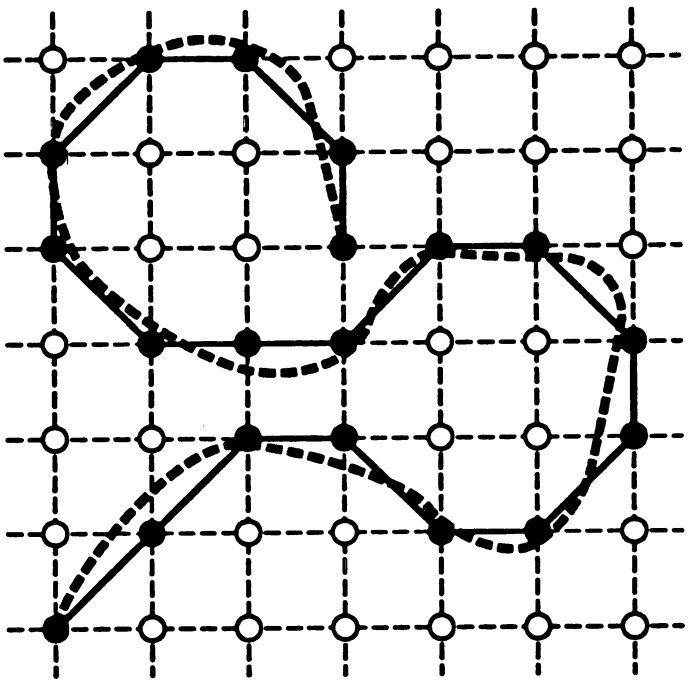

Figure 4.2: *Constrasting Sequential Search Path with Gradient Descent Path*

4.2.2 Results of the Parallel Search Architecture

Figure 4.3 shows an example of the multiple paths computed by the architecture along with the benchmark optimal path. The contour plot depicts the

cost function of the test case. In the figure, seven paths are illustrated which are separated at the start node by a 45 degree direction angle.

Figure 4.4 illustrates the solution cost from the simulated architecture. The path cost shown is that of the least cost path over 360 headings around the start node with a one degree increment. The algorithm for the parallel path planning method is shown in Figure 4.5. This algorithm implements the neural network simulation. At the end of each cycle of the simulated architecture, the path defined by the flux line at each of the 360 angles was computed and the cost of the path accumulated. The plotted cost is the minimum of all of these 360 possible paths at each pass of the simulation. This figure illustrates the convergence of the minimum cost path to asymptotic values for the electrostatic model. Figure 4.6 is a plot of the convergence over time of the simulated architecture. Each of the eight test cases exhibited a similar convergence profile. Figure 4.7 is a plot of the heading from the start node of the minimum cost path after each pass of the algorithm. This illustrates how the heading of the minimum cost path settles. All of the cases also exhibited a similar global profile in heading. For each of these plots, however, the local variations of the curves will change from one cost function to another.

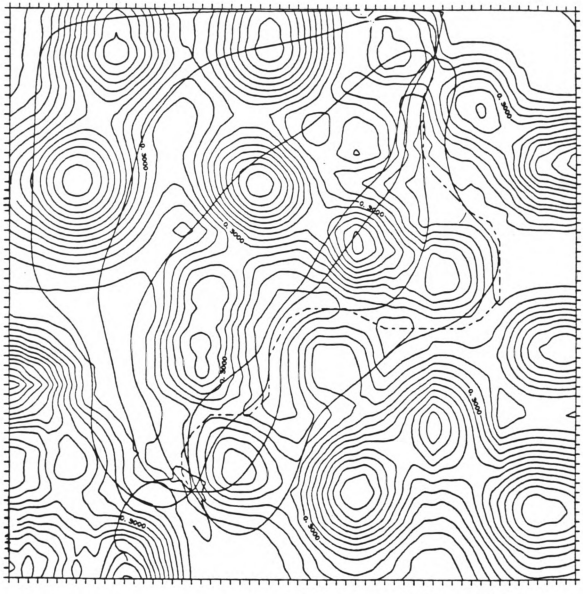

Figure 4.3: *Multiple Path Results of the Electrostatic Model on Test Case B*

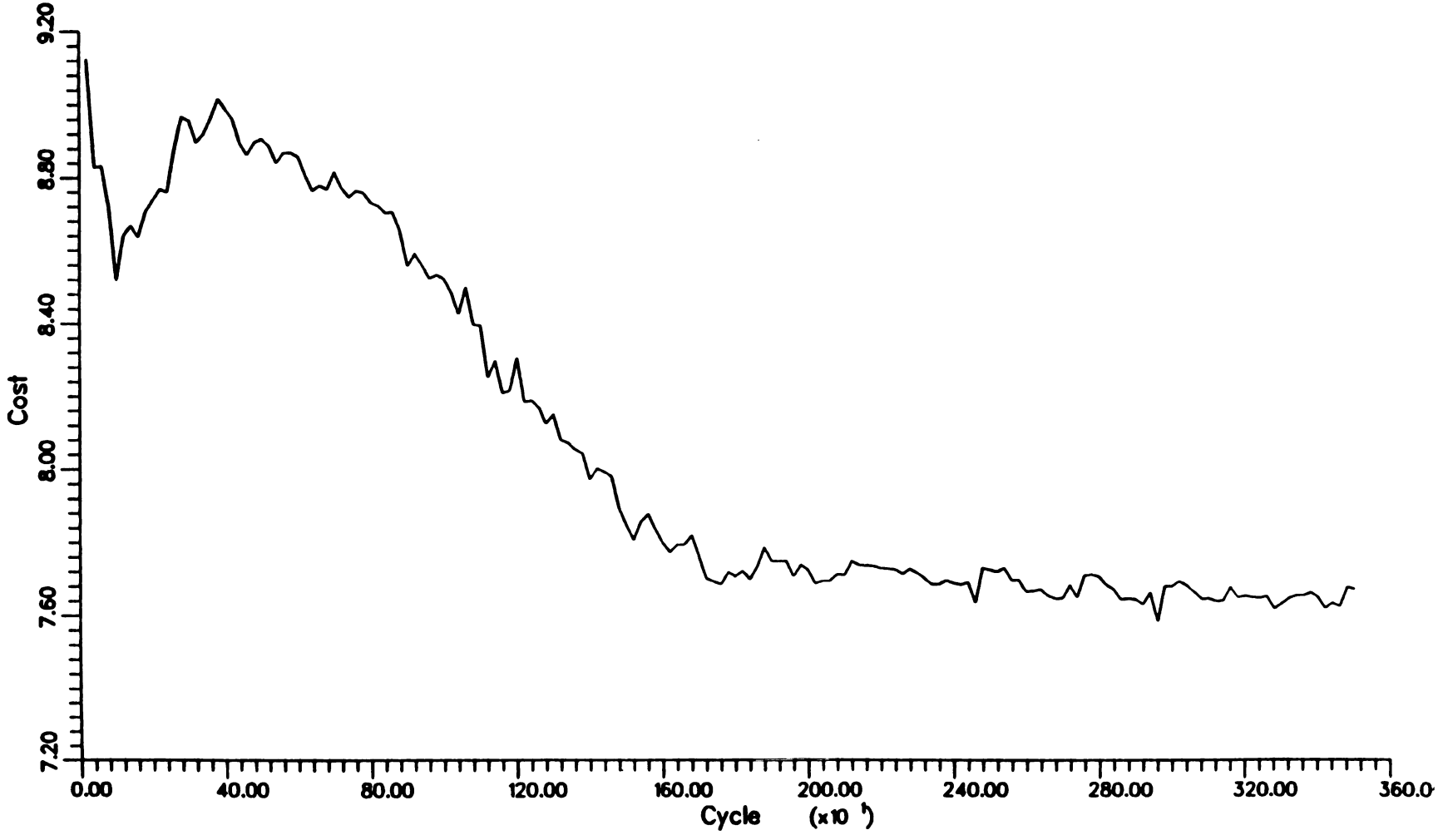

Figure 4.4: *Least Cost Path Value Found During the Neural Network Simulation*

```
1.    begin PPPM
2.        real totdiff : total sum-squared difference;
3.        real stest : convergence test criterion;
4.        integer ncycle : number of iterations;
5.        integer maxcycles : iteration limit;
6.        integer nsize : neural network square size;
7.        real u(nsize,nsize) : neural network outputs;
8.        real x(nsize,nsize) : previous u;
9.        real ssd : sum squared difference;
10.       integer j,k : general indicies;
11.       while ( totdiff > stest and ncycle < maxcycles )
12.           Compute neural network update;
13.           for j ⟵ 1 to nsize
14.               for k ⟵ 1 to nsize
15.                   ssd ⟵ u(j,k) − x(j,k);
16.                   ssd ⟵ ssd * ssd;
17.                   totdiff ⟵ totdiff + ssd;
18.               endfor
19.           endfor
20.           totdiff ⟵ totdiff/nsize/nsize;
21.           totdiff ⟵ SQRT(totdiff);
22.           ncycle ⟵ ncycle + 1;
23.           x ⟵ u;
24.           for j ⟵ 1 to 360;
25.               Perform gradient descent over u at angle j;
26.           endfor
27.       endwhile
28.   end PPPM;
```

Figure 4.5: *Algorithm of the Neural Network Parallel Path Planning Simulation*

Figure 4.6: *Convergence of Neural Outputs During Network Simulation*

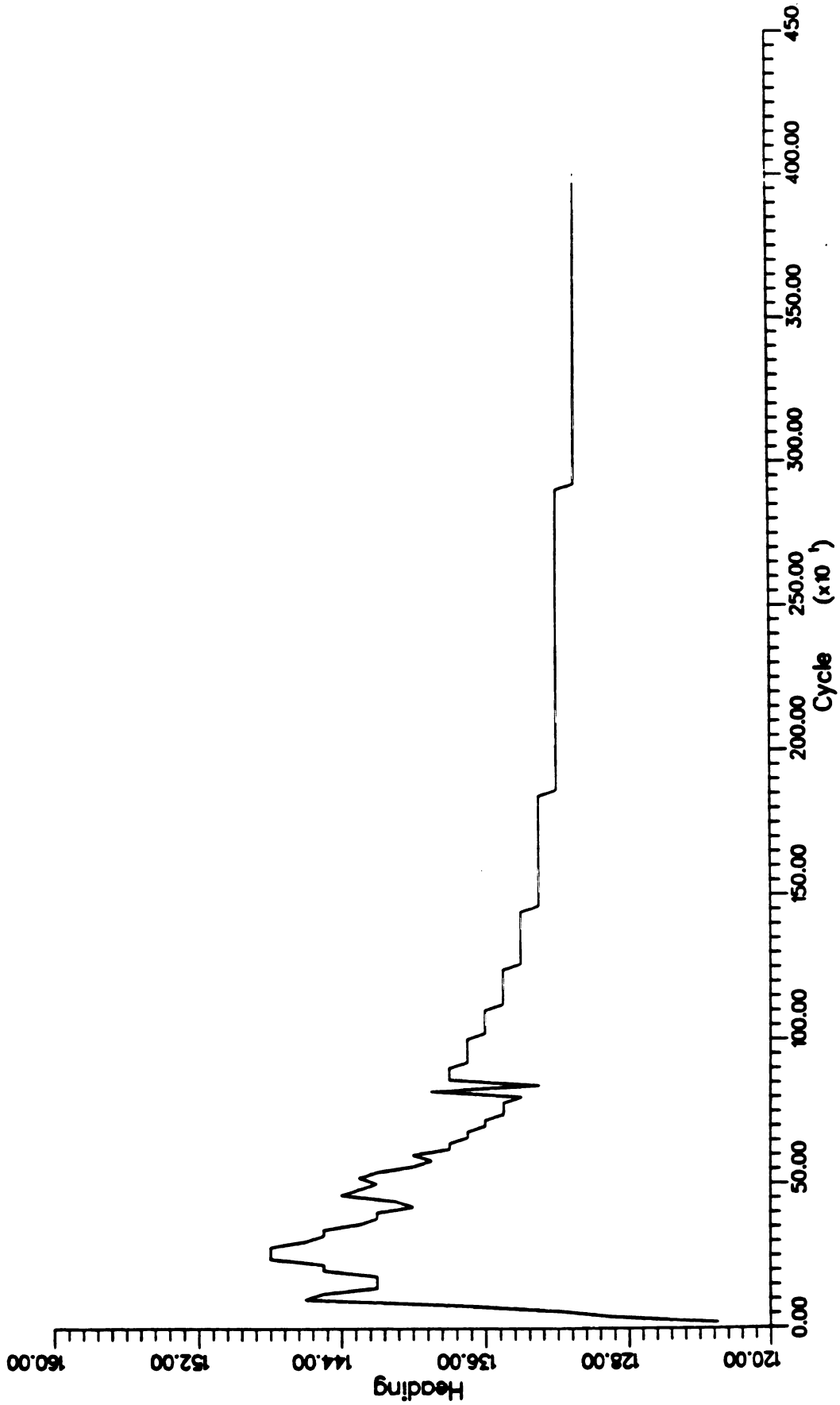

Figure 4.7: *Least Cost Path Heading Found During Neural Network Simulation*

The electrostatic model neural network becomes an element in a system which can be applied to the generation of aircraft trajectories. Such a trajectory generator is shown as a block diagram in Figure 4.8. Inputs to the neural network are the start and goal coordinates for the trajectory, and the explicit cost function values for each cell of the problem grid. In the PPPM algorithm, the network settling is checked against an input parameter for convergence. This check tests if the sum-squared difference of all the neuron outputs between successive iterations is less than the input parameter. When the neural network settles to its stable state, the neuron outputs define the potential surface solution for the electrostatic model. This surface is provided to the gradient descent procedure, along with the start and goal coordinates, and a discrete signal indicating that the network outputs are settled. The descent procedure also receives a direction angle as an external input. This angle is the heading at the start coordinate to use in initializing the descent procedure. The procedure then outputs the list of trajectory coordinates which are obtained from the descent starting with this heading. The accumulated cost along this trajectory is also provided with the list of coordinates.

Once the neural network has settled, new direction angles may be input to direct the trajectory generator to supply a new list of trajectory coordinates. This is used in the trajectory generation system which is illustrated in Figure 4.9. The trajectory generator of Figure 4.8 in contained in this system as shown in the block diagram. The current aircraft heading is supplied to a trajectory analysis function as an initial heading. The current aircraft location is supplied to the trajectory generator as the start coordinate. The trajectory analysis function passes the initial

heading as the current direction angle to the neural network. When the network is settled, the trajectory coordinates and the final accumulated cost of the trajectory is input to the analysis function. This function can then adjust the direction angle and receive a new set of trajectory coordinates and accumulated cost from the neural network, without a re-settling of the network, since the cost, start, and goal conditions have not been changed. This allows the analysis function to employ comparison techniques to converge on the minimum cost trajectory as a function of heading. An example of techniques to use could be a binary search over heading, or a branch and bound search over heading. When the analysis function finds the best trajectory according to its search criteria, it signals that the trajectory has been found and the corresponding trajectory coordinates are output.

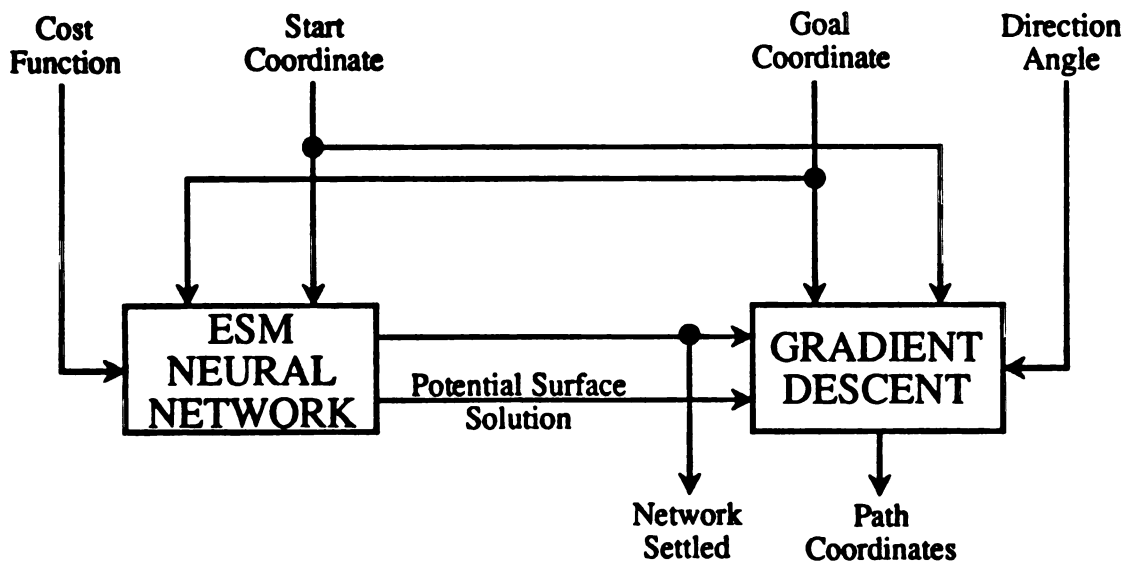

Figure 4.8: *System Block Diagram of the Electrostatic Model Neural Network*

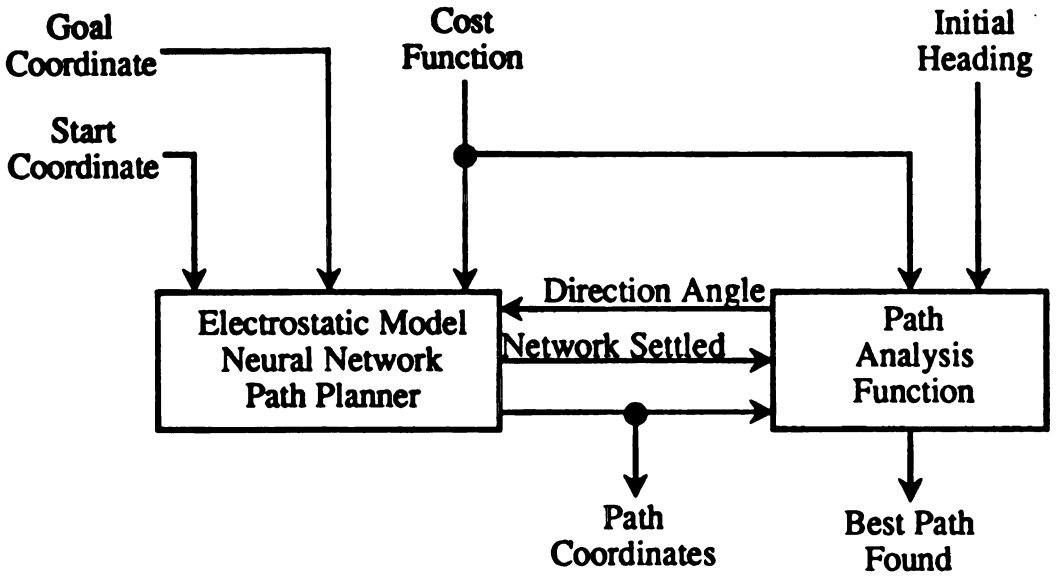

Figure 4.9: *Block Diagram of a Trajectory Generation System*

4.2.3 Multiple Path Results

One of the features of the Electrostatic Model for path planning is the generation of multiple paths in the solution space. Once the potential surface is computed by the neural network, gradient descent from the start node over the surface extracts the path. The initial direction angle, which is given as a heading, determines a unique path to the goal node. For all of the test cases, a one degree increment was used at the start node heading to create the unique paths. This section reviews the results of all the test cases for their multiple paths.

Table 4.3 shows the best ten paths for Test Case A. The costs of the first 10 paths had a range of 0.005 which is approximately 1.5 percent of the best path cost of 0.3287. The PPPM algorithm computed a total of 170 paths for Test Case A. The costs of these paths are plotted in sorted order from the best value to the worst value in Figure 4.10. The figure clearly indicates a linear cost increase of

Table 4.3: *Costs and Headings for 10 Best Path Values of Test Case A*

RANK	Path Cost Value	Heading
1	0.3287	131
2	0.3306	132
3	0.3312	133
4	0.3319	125
5	0.3319	126
6	0.3325	127
7	0.3325	128
8	0.3325	134
9	0.3331	129
10	0.3337	130

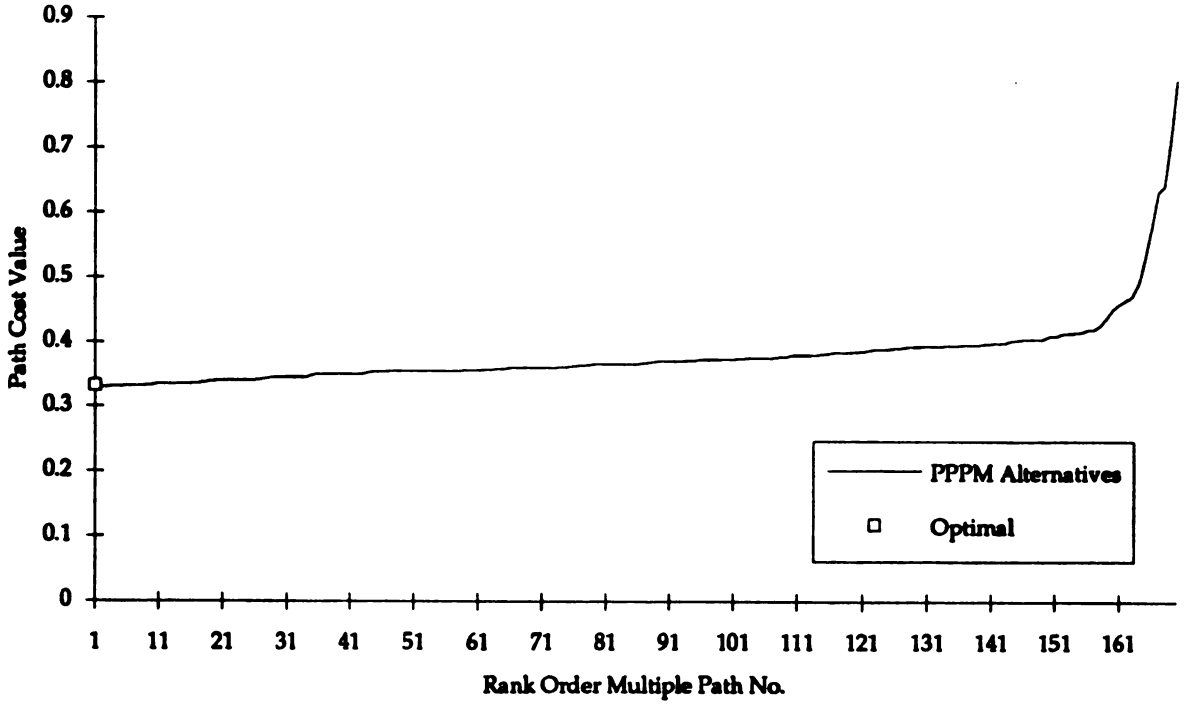

Figure 4.10: *Plot of Test Case A Multiple Path Values Ordered by Cost*

only 0.0006 units per path up through the first 157 paths which were generated. This is a mere 0.18 percent increase in the cost per path. Overall, the 157th path cost of 0.4206 represents only a 28 percent increase in the path cost from the best path value of 0.3287. Of the 170 paths generated for Test Case A, these 157 nearly linearly increasing path values account for 92 percent of the solutions which were generated by the PPPM algorithm for Test Case A.

The best 10 path values for the remaining test cases are contained in Table 4.4 through Table 4.9. The plots of the ordered path values are displayed in Figure 4.11 through Figure 4.16. These data show that similar results hold for all of the test cases. All test cases show that the increase in path cost values for the multiple paths starts out linearly, followed by a more dramatic rise in the path cost values. Over the linear region of these path cost values, the slope of the linear increase averaged only 0.22 percent of the best path value per new path. This linear portion of the multiple paths averaged to be 63 percent of the multiple paths generated by the algorithm PPPM over all of the test cases.

Table 4.4: *Costs and Headings for 10 Best Path Values of Test Case B*

RANK	Path Cost Value	Heading
1	7.672	92
2	7.678	96
3	7.694	95
4	7.701	93
5	7.703	94
6	7.725	89
7	7.729	97
8	7.742	90
9	7.790	91
10	7.793	99

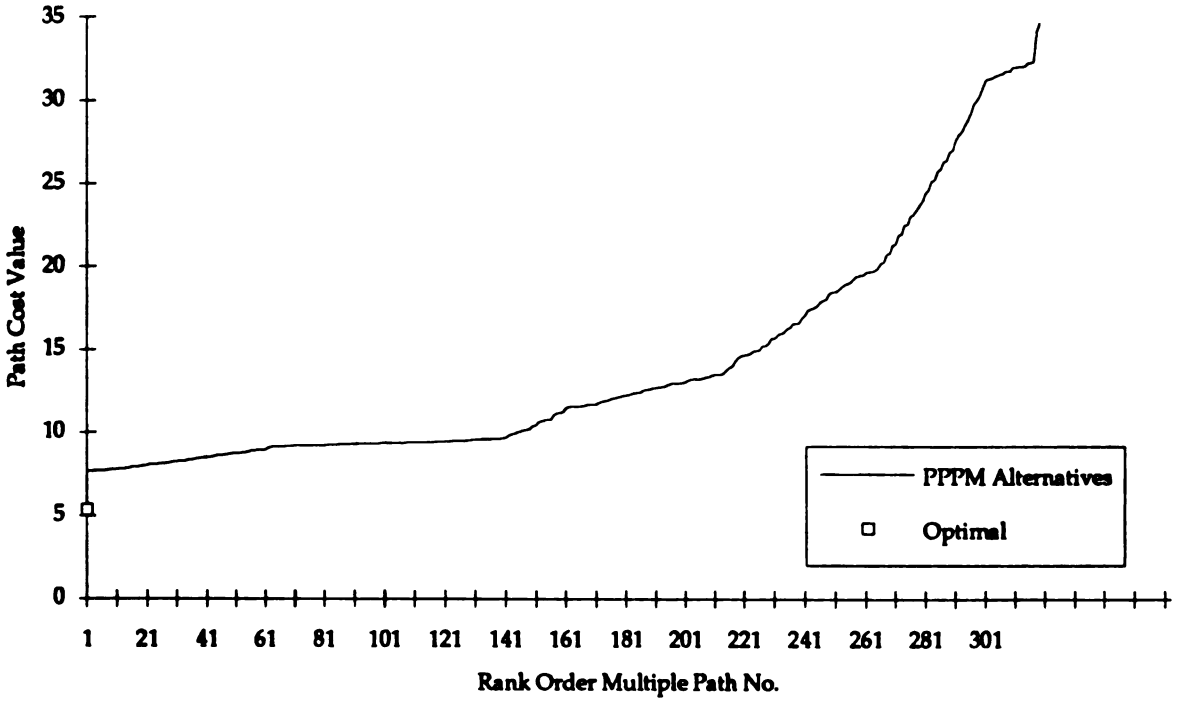

Figure 4.11: *Plot of Test Case B Multiple Path Values Ordered by Cost*

Table 4.5: *Costs and Headings for 10 Best Path Values of Test Case C*

RANK	Path Cost Value	Heading
1	0.8397	52
2	0.8404	51
3	0.8417	50
4	0.8418	77
5	0.8422	76
6	0.8428	75
7	0.8456	74
8	0.8459	72
9	0.8463	81
10	0.8464	37

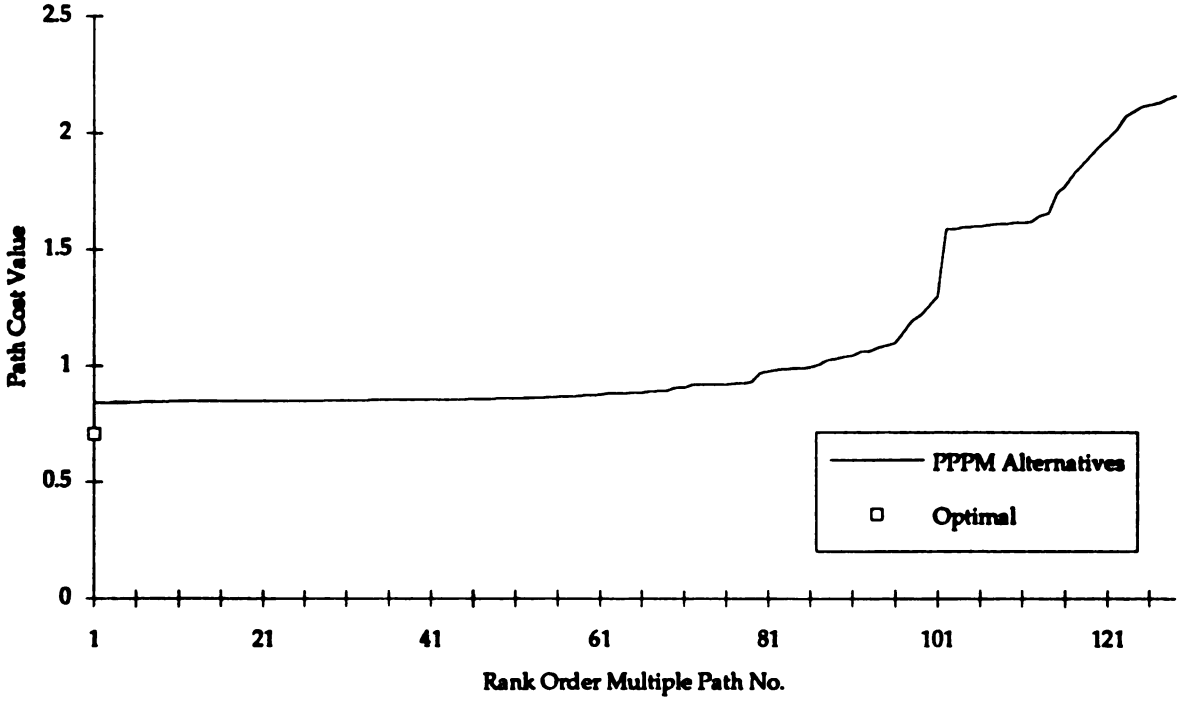

Figure 4.12: *Plot of Test Case C Multiple Path Values Ordered by Cost*

Table 4.6: *Costs and Headings for 10 Best Path Values of Test Case D*

RANK	Path Cost Value	Heading
1	0.4298	65
2	0.4304	64
3	0.4315	63
4	0.4320	62
5	0.4323	74
6	0.4324	73
7	0.4330	72
8	0.4331	61
9	0.4336	71
10	0.4338	60

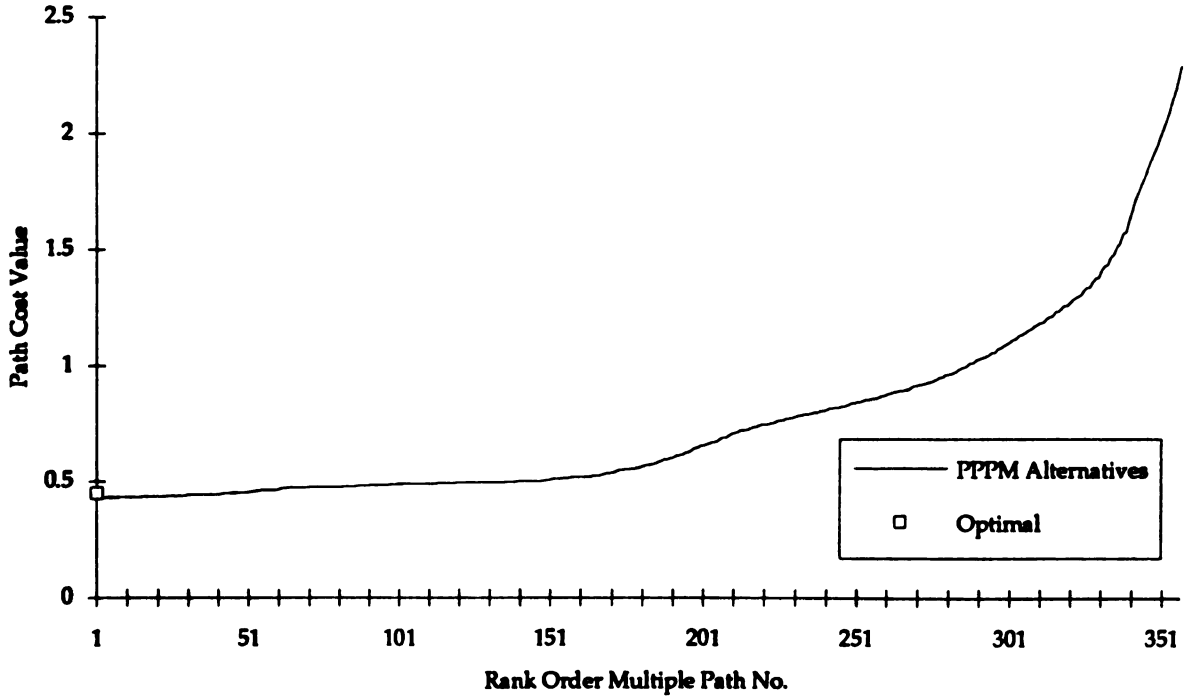

Figure 4.13: *Plot of Test Case D Multiple Path Values Ordered by Cost*

Table 4.7: *Costs and Headings for 10 Best Path Values of Test Case E*

RANK	Path Cost Value	Heading
1	19.00	71
2	19.50	72
3	19.50	74
4	19.50	75
5	20.00	68
6	20.00	76
7	20.00	77
8	20.00	78
9	20.00	79
10	20.50	69

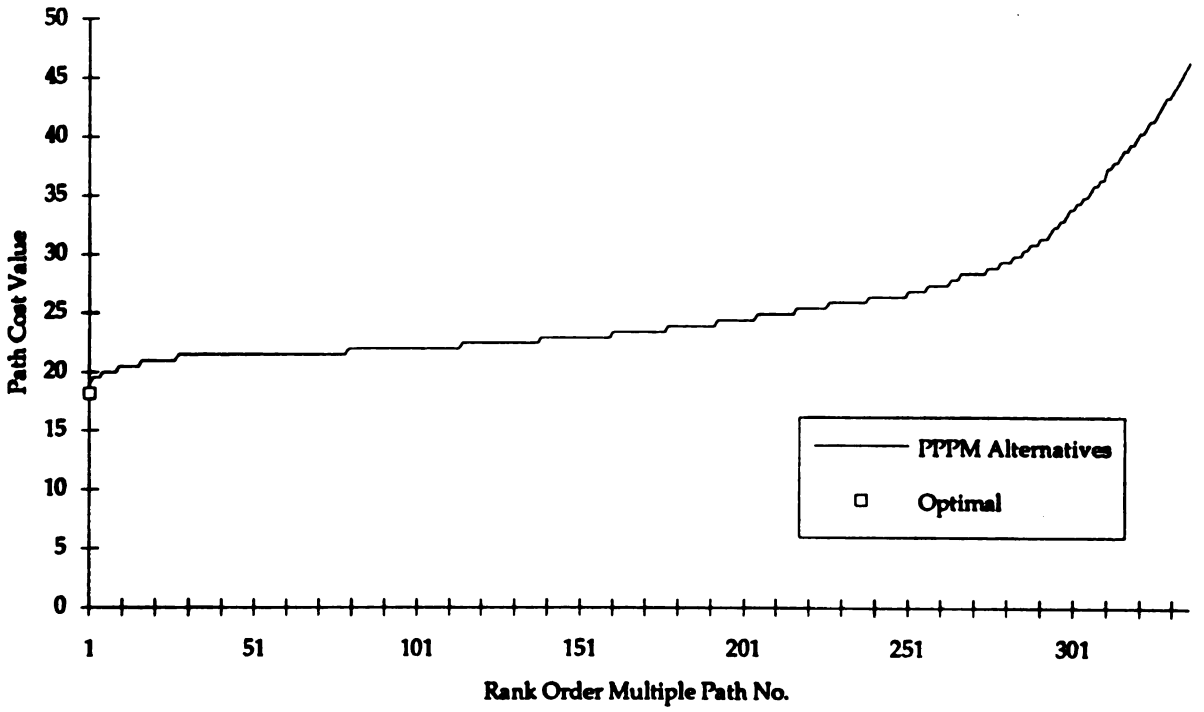

Figure 4.14: *Plot of Test Case E Multiple Path Values Ordered by Cost*

Table 4.8: *Costs and Headings for 10 Best Path Values of Test Case F*

RANK	Path Cost Value	Heading
1	18.00	336
2	18.50	337
3	18.50	338
4	18.50	339
5	18.50	340
6	18.50	341
7	18.50	342
8	18.50	343
9	18.50	344
10	18.50	345

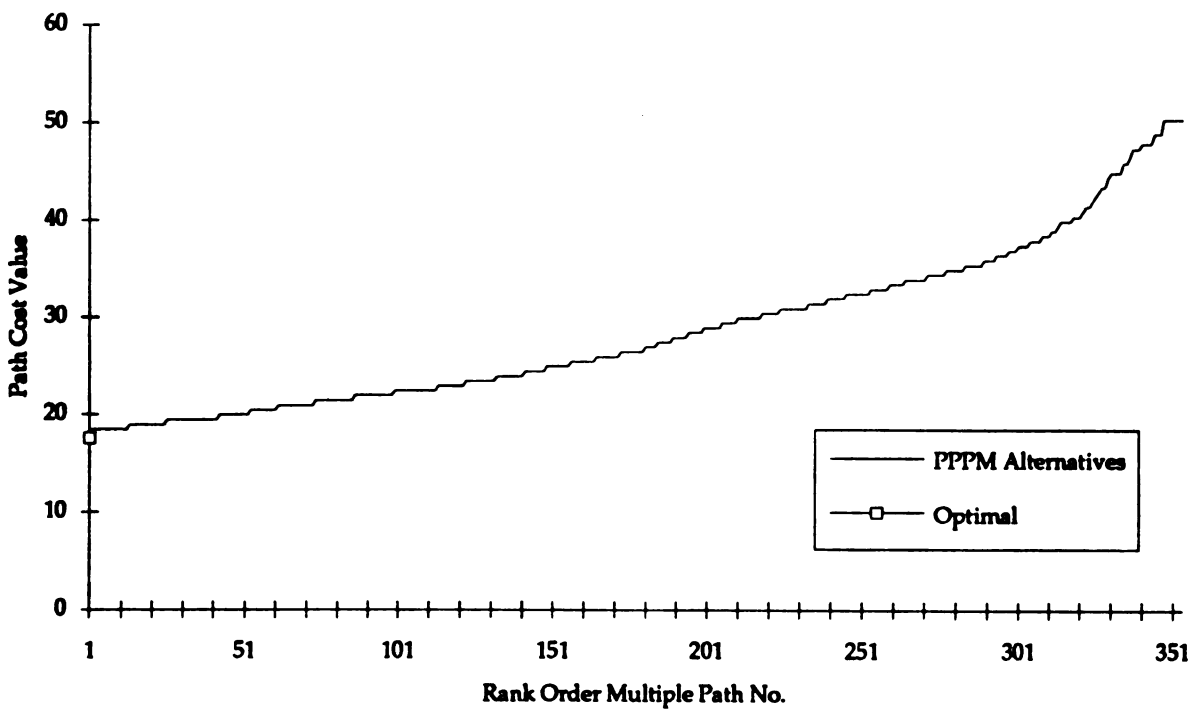

Figure 4.15: *Plot of Test Case F Multiple Path Values Ordered by Cost*

Table 4.9: *Costs and Headings for 10 Best Path Values of Test Case G*

RANK	Path Cost Value	Heading
1	36.50	13
2	38.00	12
3	38.00	14
4	38.00	15
5	38.00	16
6	38.00	20
7	38.00	21
8	38.50	9
9	38.50	10
10	38.50	11

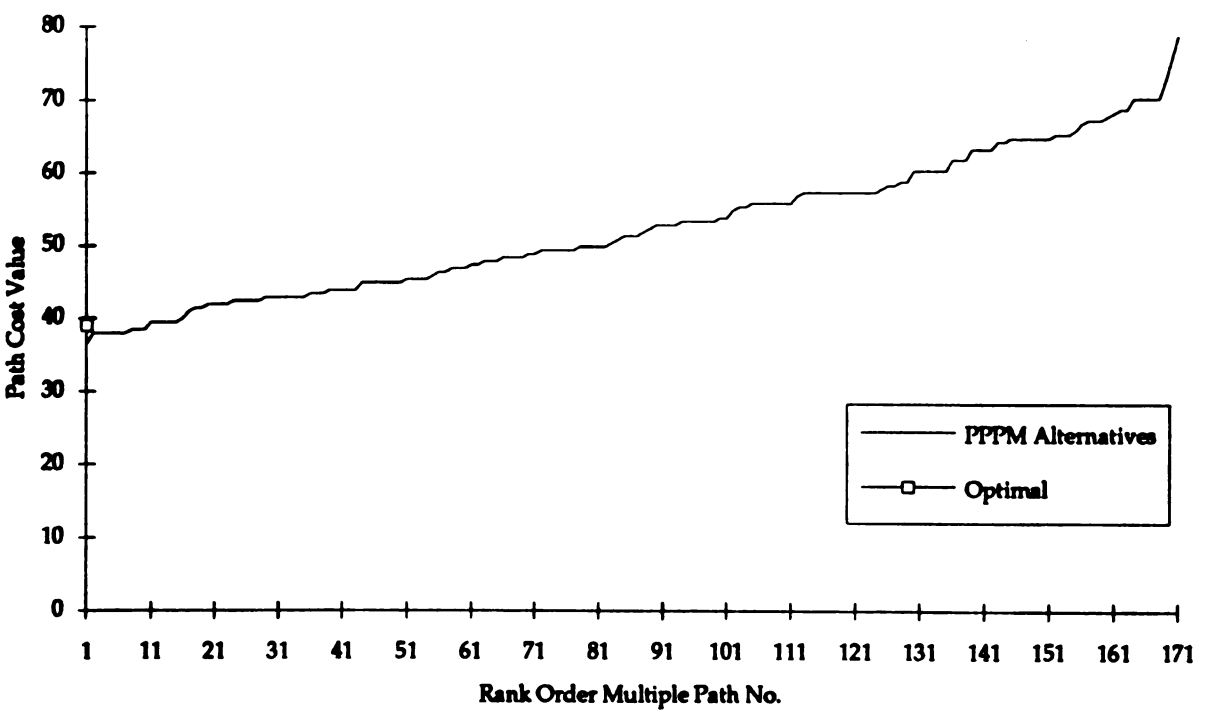

Figure 4.16: *Plot of Test Case G Multiple Path Values Ordered by Cost*

4.3 Variable Connectivity Results

4.3.1 Taylor Series Template Method

Table 4.10 shows the number of cycles for each simulation run of the neural network using the templates generated in Section 3.3.1. These runs were executed on a VAX 8600 until the convergence input parameter was satisfied at the end of each iteration of the simulation (or until 15,000 iterations of the simulation were executed). The sum-squared difference of all neuron outputs (`TOTDIFF` in the `PPPM` algorithm) is checked against the input parameter to test for convergence. For the simulation runs, the convergence input parameter was arbitrarily set to 0.001.

Table 4.10: *Neural Network Simulation Iterations Using Taylor Series Templates*

Template Size	Cycles per Case		
	A	B	C
5 point	4,000	3,484	5,460
13 point	3,647	2,251	4,015
25 point	3,621	2,612	4,518
41 point	6,947	15,000	10,140

In each case, it can be seen that fewer cycles of the parallel algorithm are required for convergence using the 13 point template over the 5 point template. While this improvement continues marginally for the 25 point template of Case A, in the other cases the number of cycles to achieve convergence increased. Continued expansion to the 41 point template resulted in a further, and rather significant degradation in convergence for all cases. Table 4.11 contains data for the final minimum cost path value at the end of the algorithm execution. Each

larger template failed to provide more accurate (smaller) cost values in each case. In fact, the cost values increase with the larger templates using the Taylor series method. The results reported here illustrate the failure of the Taylor series

Table 4.11: *Least Cost Path Values Using Taylor Series Templates*

Template Size	Path Cost per Case		
	A	B	C
5 point	0.329	7.672	0.840
13 point	0.331	7.599	0.937
25 point	0.331	7.631	1.006
41 point	0.333	7.826	1.068

template method which was reported in Section 3.3.4 and led to the development of the parallel summation template method.

Figure 4.17 through Figure 4.19 are composite plots of the minimum cost of successive algorithm iteration for the four increasingly complex Taylor series template connectivity patterns. Over 135 simulations were run and their results were plotted. These plots show the value of the minimum cost path found by the neural network during each cycle of the run. This was repeated for the more connected pattern, and each run was plotted for a particular test case. A family of curves results which shows the computed cost over computation time for the varying connectivity. These plots show the impact of increased connectivity on the quality of the computation results. Each figure illustrates that the convergence patterns are similar for all templates. The plots of different test cases have a different character or general shape. There are three considerations to explain this phenomenon. First, each test case had a different cost function. Not only was the size of the problems different, but the variability of the cost function was

also different. The cost functions included rather smooth surfaces, sharp edges, or frequent hills and valleys. The unique surfaces of these cost functions result in the different characteristics of these connectivity curves. The second reason is that the value plotted is for the minimum cost path found thus far in the algorithm iteration. This can be any flux line and the heading plot of Figure 4.7 shows how much this changes at the beginning of the algorithm. The final consideration is that the plot illustrates tentative results towards convergence. The path value depends on the latest potential surface computed by the neural network at that point in the algorithm convergence. Again, Figure 4.6 illustrates that during the early iterations of the algorithm, the potential surface has not converged to a stable configuration. These reasons considered together account for the differences in the curves between the test cases.

Upon close inspection of the plots, the desired result of improvement from using a more complex template connectivity pattern does not occur. Indeed, as the template connectivity is increased, there is actually a decrease in the convergence, exhibited by a higher final cost of the minimum cost path.

However, these connection patterns are very small on the scale of connectivity. In order to get more significant results at larger connectivity, much larger templates are required. Additional constraints are added to convert the template coefficient equations (Equation 3.46 through Equation 3.51) from an underdetermined system. These constraints are intuitive, but yet somewhat ad hoc, trying to account for behavior farther from the approximation point at the center of the template.

As the connectivity increases with the larger templates, it was hoped that the cost plot could be predicted to drop sooner for each increase in connectivity, reaching the minimum cost in faster time. The larger templates use more of the network node outputs. As changes to node outputs occur in a remote region of the network more distant from a local node of interest, these larger templates facilitate propagation of the remote changes to each local node's output computation. However, plots of the simulation results show that the larger templates are not providing a better approximation. An examination of Figure 4.17 through Figure 4.19 reveals that the higher connective templates are not, in general, converging faster.

The template defines the differential operator for the approximation. Expansion of the template would seem to be counter to the definition of a derivative in the limit, which would lead to contracting the template instead of expanding the template. As the cost function varies more, less accurate approximations appear with the larger templates, and argues for smaller templates in order to have more accurate approximations of the derivatives with higher frequency cost function variation.

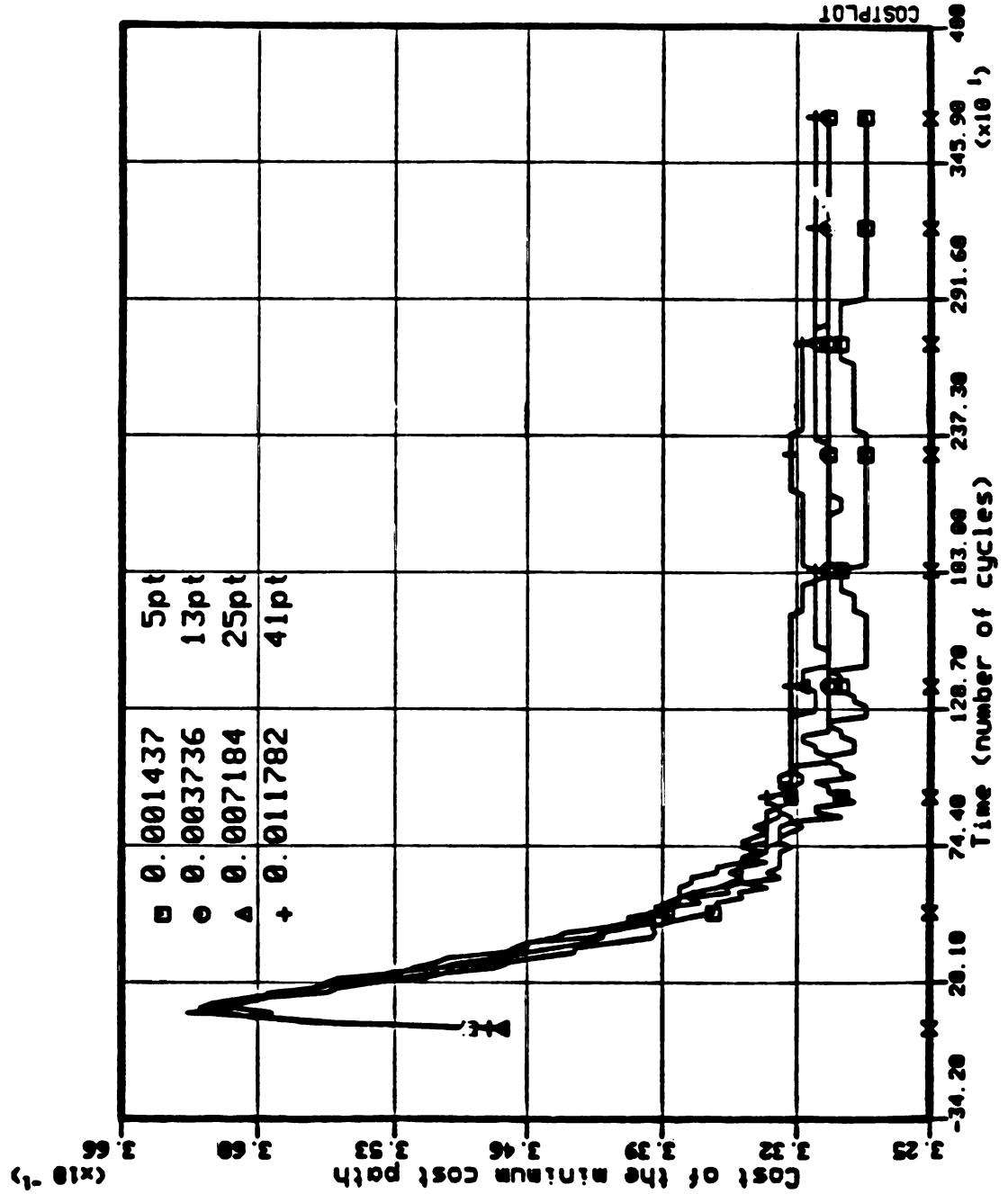

Figure 4.17: *Connectivity Curves for the Least Cost Path Found Using the Taylor Series Templates on Test Case A*

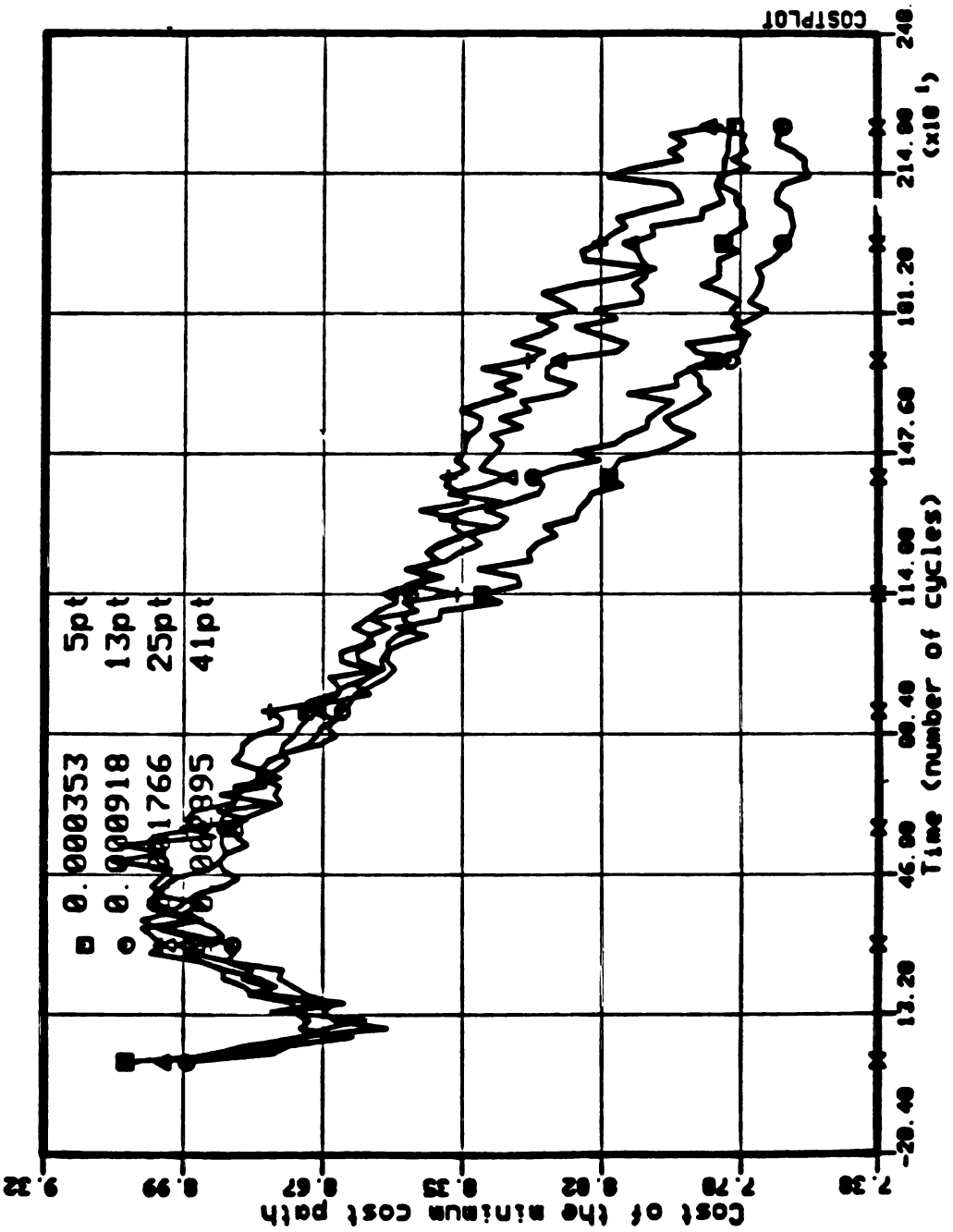

Figure 4.18: *Connectivity Curves for the Least Cost Path Found Using the Taylor Series Templates on Test Case B*

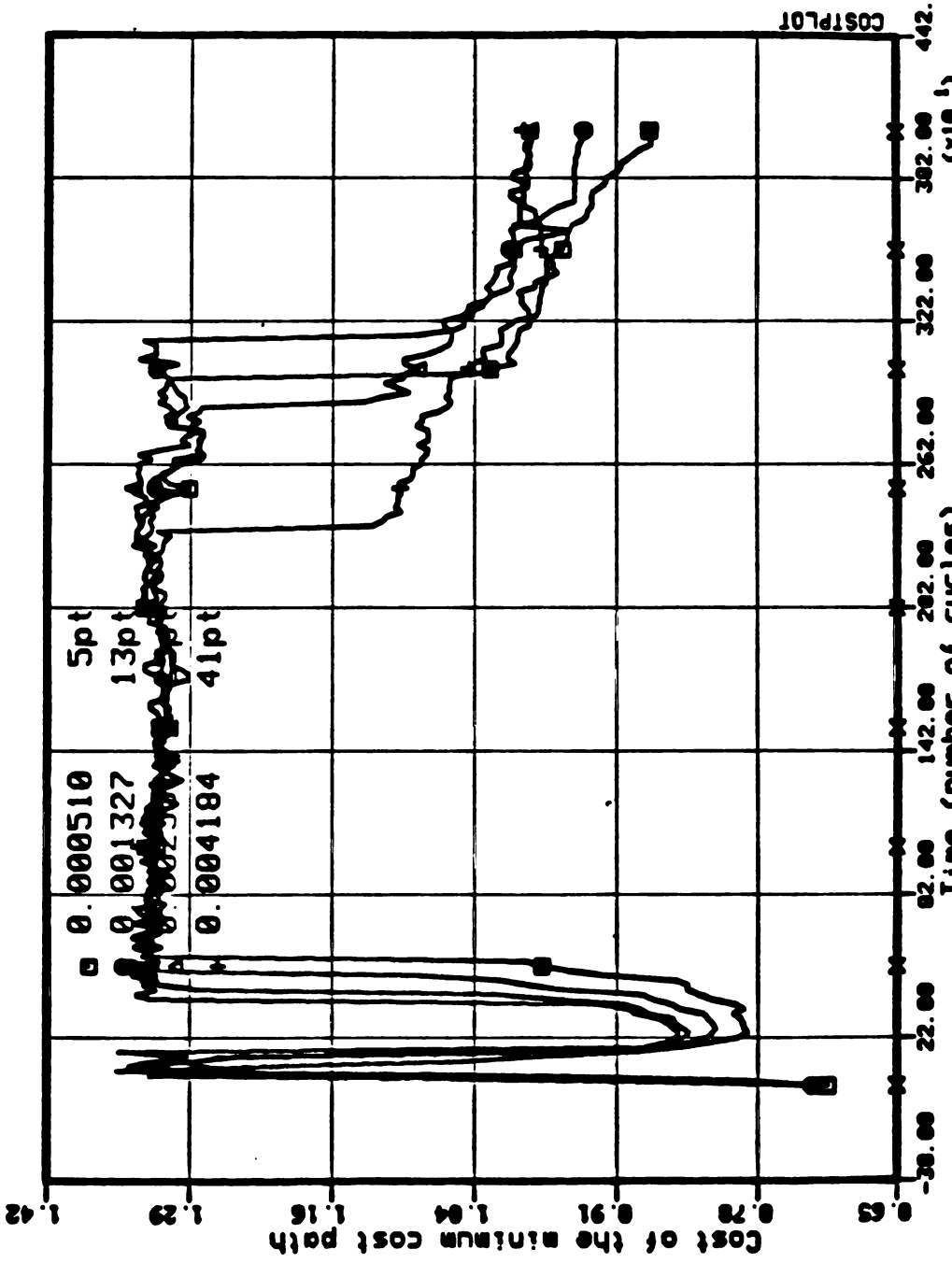

Figure 4.19: *Connectivity Curves for the Least Cost Path Found Using the Taylor Series Templates on Test Case C*

4.3.2 Parallel Summation Method

The parallel summation method provided a solution to the convergence and accuracy problems of the Taylor series templates of larger connectivity. The solution did not change the underlying accuracy of the five-point approximation to the derivative, but rather achieved increased connectivity through the computation architecture of the template.

Figure 4.20 through Figure 4.22 are composite plots of the minimum cost of successive algorithm iteration for the increasingly complex template connectivity patterns derived using the parallel summation architecture of Section 3.3.4. Each figure illustrates that the convergence patterns are similar for each template. However, unlike the previous template patterns derived from the Taylor series, these template patterns exhibit the improvement expected from using increased connectivity as hypothesized in Section 3.3.4. As the template connectivity is increased through more parallelism in the approximation summation, the network architecture accomplished a more rapid decrease in minimum cost paths. In each plot, the increased connectivity is indicated by a larger index, k. Each test case has a similar behavior. The higher the connectivity index, the faster the minimum cost drops during the algorithm iteration.

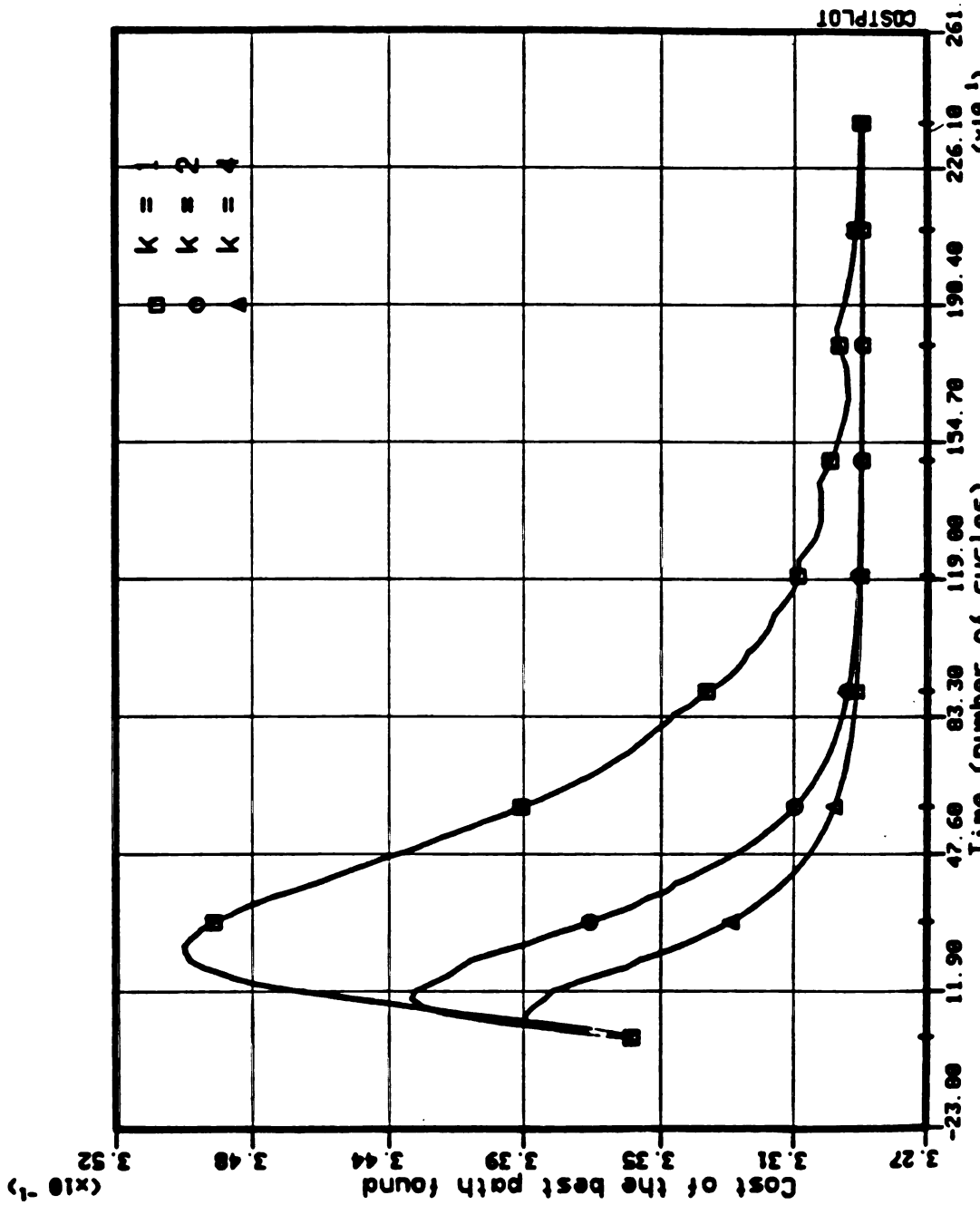

Figure 4.20: *Connectivity Curves for the Least Cost Path Found Using the Parallel Summation Templates on Test Case A*

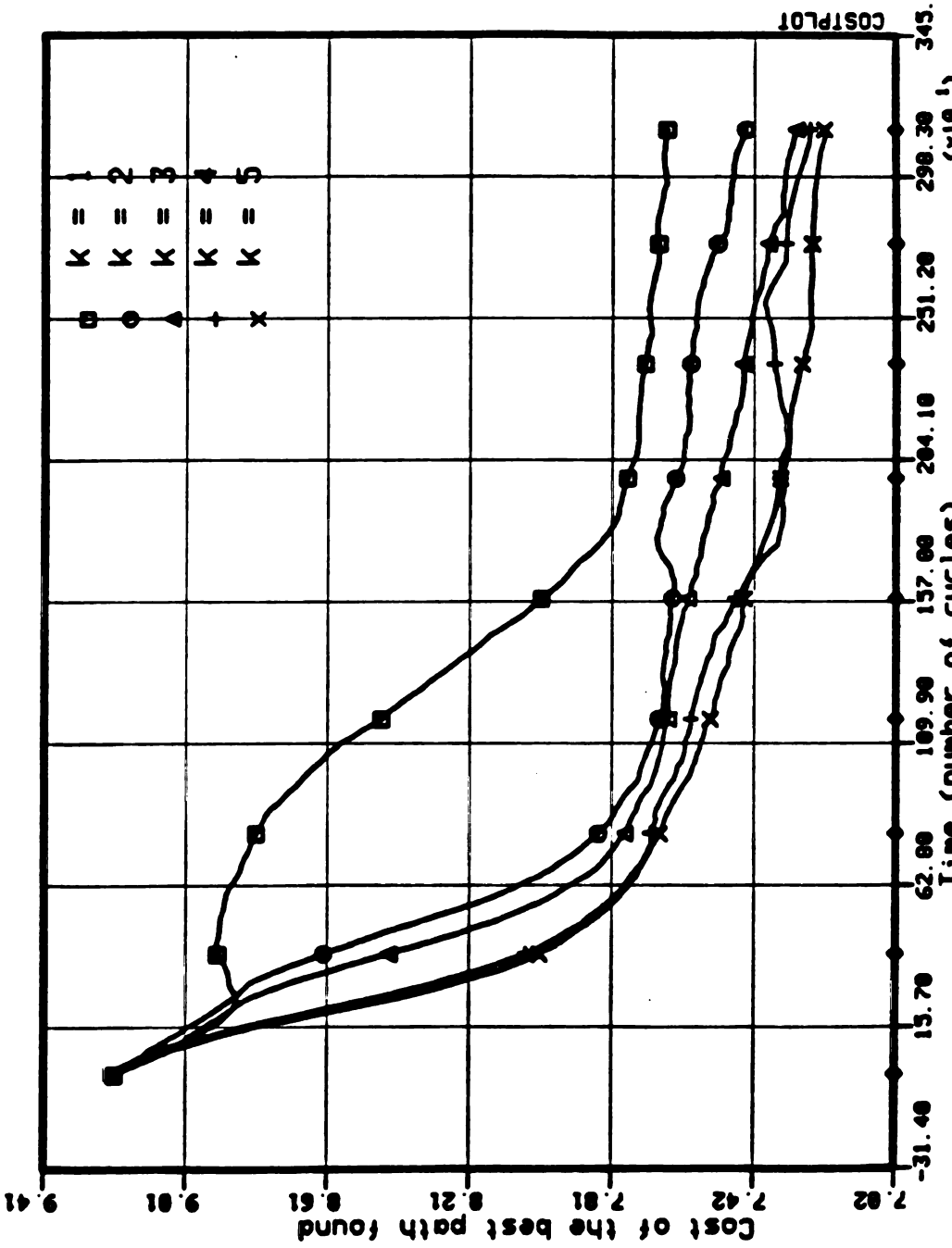

Figure 4.21: *Connectivity Curves for the Least Cost Path Found Using the Parallel Summation Templates on Test Case B*

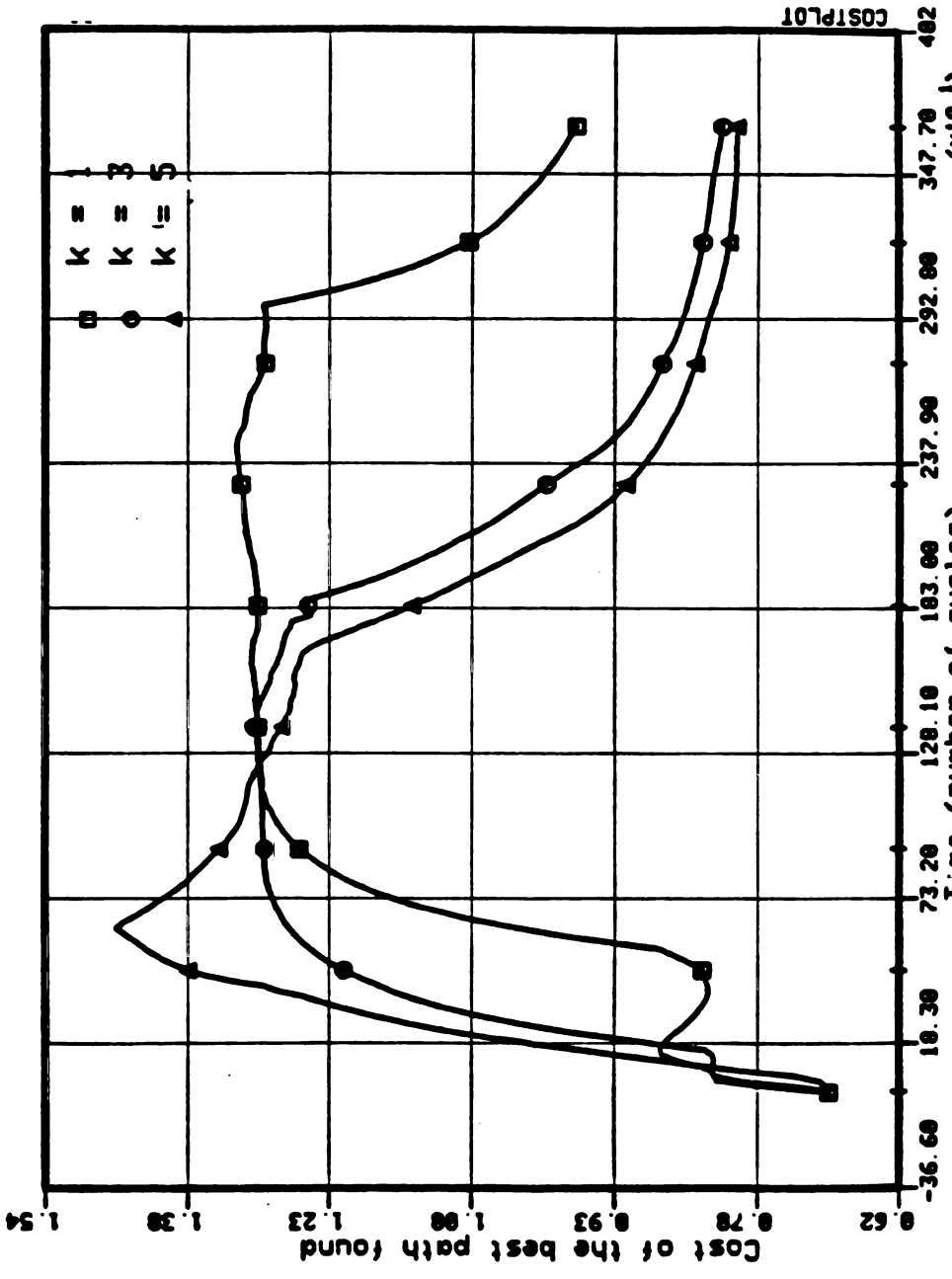

Figure 4.22: *Connectivity Curves for the Least Cost Path Found Using the Parallel Summation Templates on Test Case C*

4.4 Connection Machine Analog Simulation Results

Hardware implementation of neural networks have been offered[12, 15, 30, 31] using many technologies. Analog neural networks have received a lot of interest due to their speed and potential for VLSI implementation. This section reports on an investigation of an analog implementation of the neural network architecture for the electrostatic model. Because of the number of neurons involved in the test case (3,600), this required 7,200 operational amplifiers for an analog circuit. Rather than building such a massive circuit using breadboard techniques, a simulation of the analog circuit was developed for execution on a Connection Machine. The Connection Machine was selected because of the ease with which the simulation could be programmed for parallel execution and the resulting simplicity of the neuron model. Each processor of the Connection Machine was devoted to simulating a single neuron.

The analog circuit simulation is based upon the neuron model illustrated in Figure 3.9. For simulating the neural network architecture on the Connection Machine[28], a simple operational amplifier model[66] was employed as shown in Figure 4.23. A block diagram of the function of the inverting operational amplifier is shown in Figure 4.24. This block diagram computes the function

$$e_o = -\omega_H \int e\, dt$$

where ω_H is the open-loop gain of the amplifier. The digital simulation is implemented as shown in Figure 4.25.

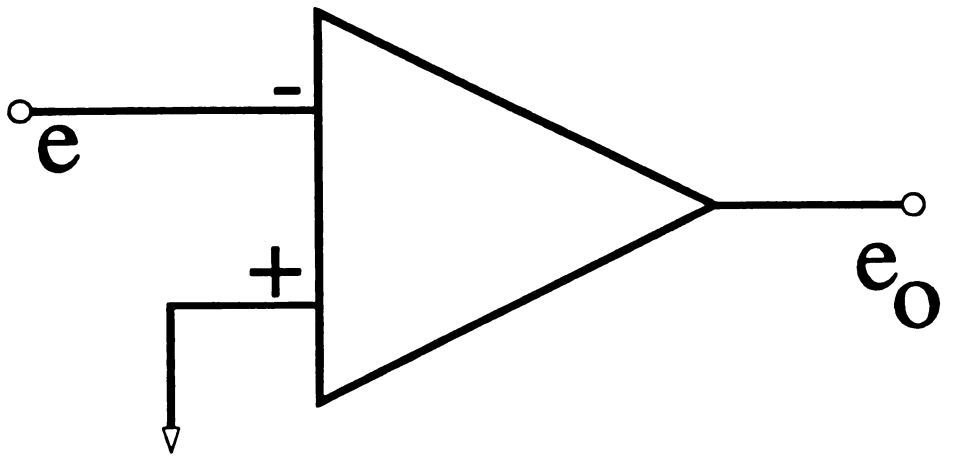

Figure 4.23: *Operational Amplifier Schematic*

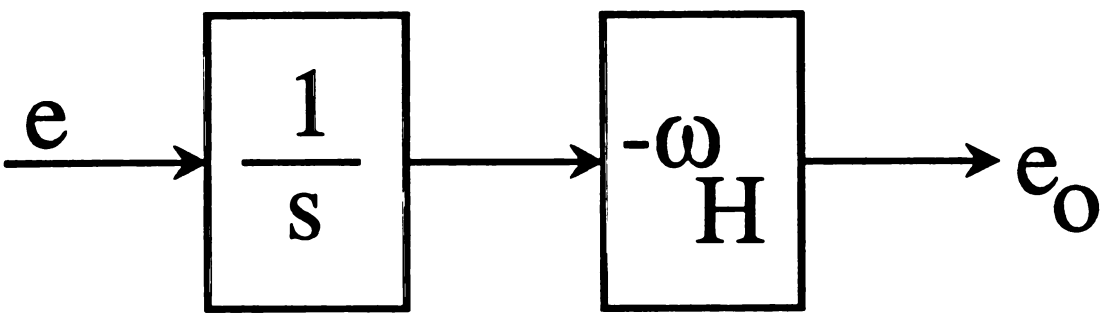

Figure 4.24: *Operational Amplifier Block Diagram*

The configuration employed in the implementation of the model using inverting operational amplifiers is shown in Figure 4.26. The operational amplifier attempts to match the inputs. Since the positive input is grounded, the output of the amplifier will be such that the negative input will be driven to match to ground state through the feedback path. The equations for the configuration begin with applying Kirkoff's current law to the junction at the negative input to the op amp, labelled as e. Since there is a very high impedance at the input to the op amp,

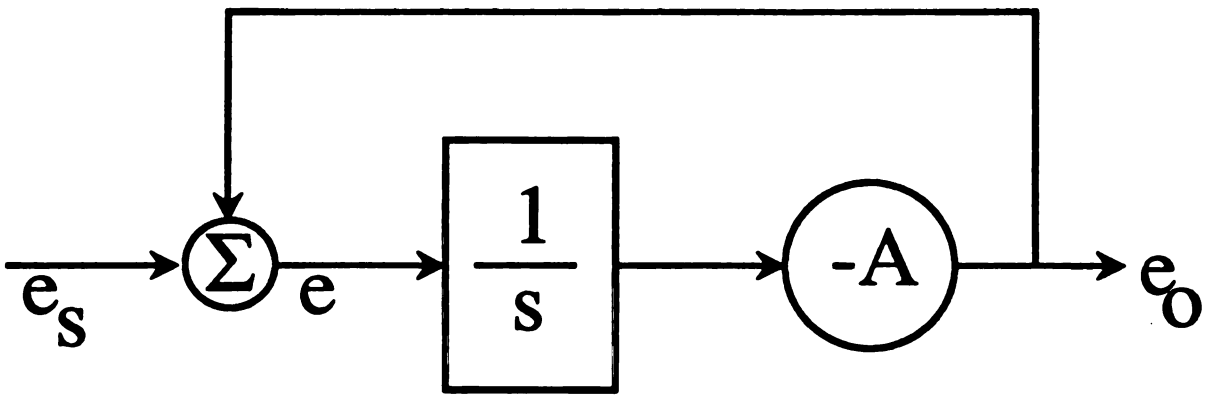

Figure 4.25: *Op Amp Digital Simulation Block Diagram*

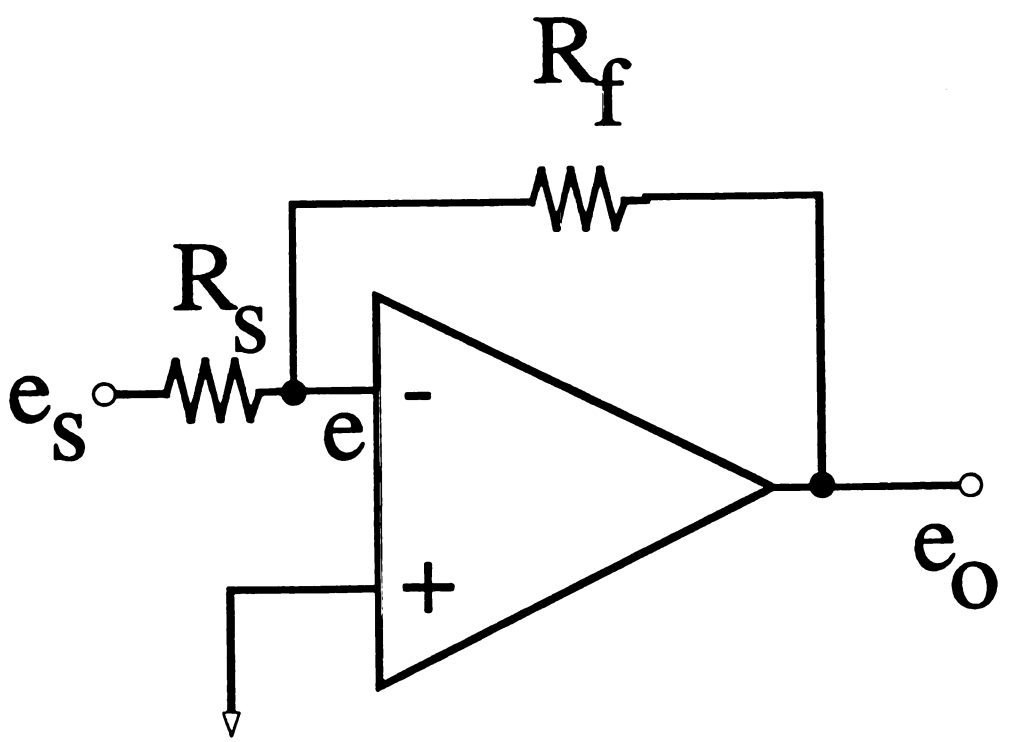

Figure 4.26: *Configuration of an Op Amp Adder with Gain*

any current flow to or from the circuit node e and the op amp at the negative input can be ignored. So the first equation needed is

$$\frac{e_s - e}{R_s} + \frac{e_o - e}{R_f} = 0 \tag{4.1}$$

Solving for e, gives

$$e = \frac{R_f}{R_f + R_s} e_s + \frac{R_s}{R_f + R_s} e_o \tag{4.2}$$

But since the operational amplifier is driving its output so that its inputs match and the positive input is ground, $e = 0$, and Equation 4.2 reduces to

$$e_o = -\frac{R_f}{R_s} e_s \tag{4.3}$$

Thus, the closed loop gain provided by the operational amplifier is

$$\text{GAIN} = \frac{e_o}{e_s} = -\frac{R_f}{R_s} \tag{4.4}$$

For n inputs, $e_1, \ldots, e_n$, rather than the single input e_s, this configuration generalizes to

$$e_o = -R_f \left(\frac{e_1}{R_1} + \frac{e_2}{R_2} + \cdots + \frac{e_n}{R_n} \right) \tag{4.5}$$

Figure 4.27 is an illustration of the op amp schematic for implementing the 5-point finite difference approximation template which computes the weighted sum of its four neighbors.

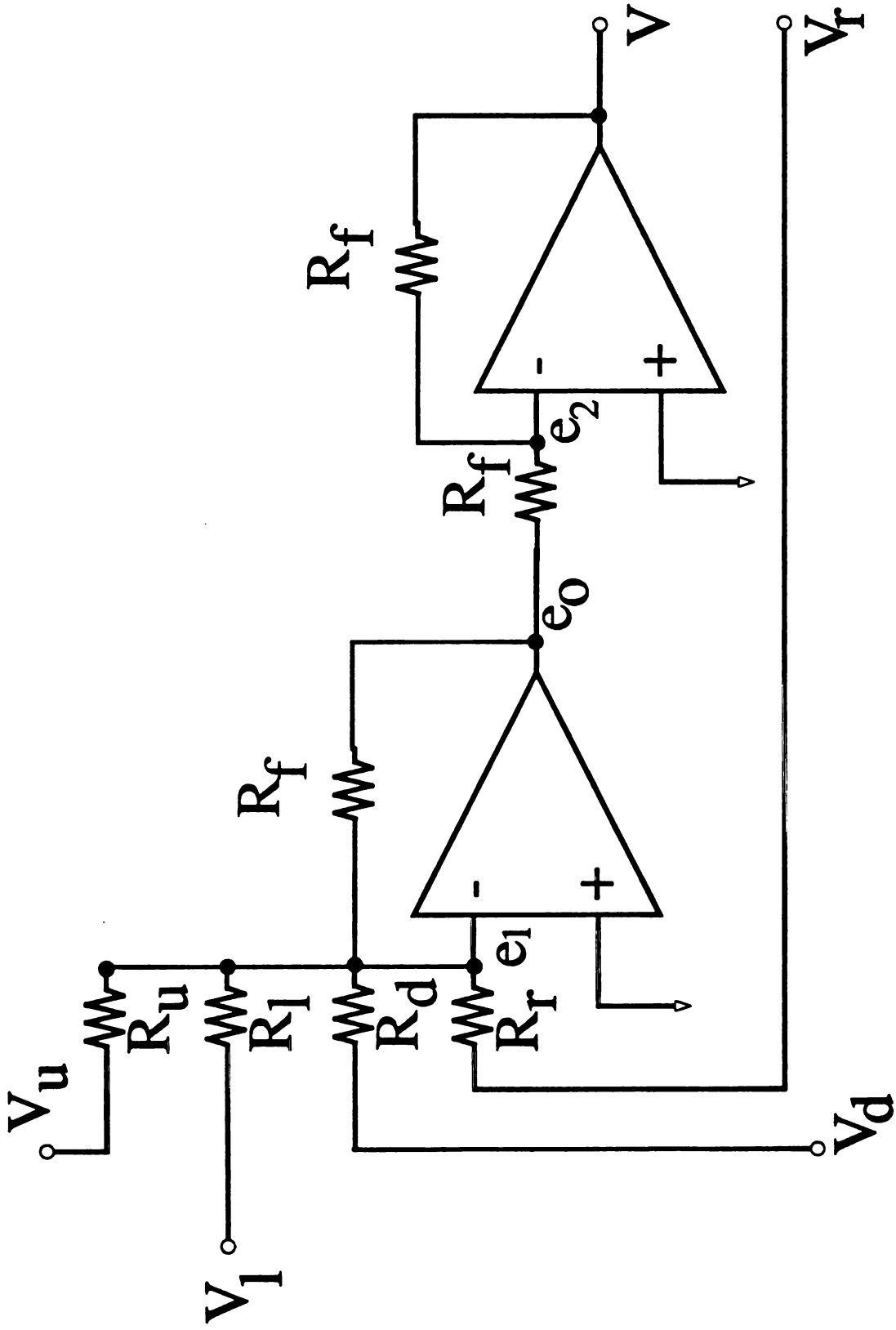

Figure 4.27: *Neuron Schematic for 5-pt. Approximation using Op Amps*

To set up for the simulation of the operation amplifier neural network, a conductance, G, for each input connection is computed. This conductance, G, is the inverse of R. In the following procedure, the subscripts R, L, U, D denote (respectively), the right, left, up, and down neighbor nodes in the finite difference approximation template. The procedure is as follows:

1. Establish the reference conductance, G_f

2. For each input connection weight, w_i, compute its corresponding conductance value $G_i = G_f w_i$, where $w_i = \alpha_i/\alpha_o$, as computed in Equation 3.59 through Equation 3.63 and shown in Table 3.2.

3. Compute the voltage divider coefficients

$$VDC_R = \frac{G_R}{G_R + G_L + G_U + G_D + G_f}$$
$$VDC_L = \frac{G_L}{G_R + G_L + G_U + G_D + G_f}$$
$$VDC_U = \frac{G_U}{G_R + G_L + G_U + G_D + G_f}$$
$$VDC_D = \frac{G_D}{G_R + G_L + G_U + G_D + G_f}$$
$$VDC_f = \frac{G_f}{G_R + G_L + G_U + G_D + G_f}$$

4. Implement the operational amplifier by computing

$$e_1 = V_R VDC_R + V_L VDC_L + V_U VDC_U + V_D VDC_D + e_o VDC_f$$

$$e_o = -A \int e_1 \, dt$$

$$e_2 = \frac{e_o}{2} + \frac{V}{2}$$

$$V = -A \int e_2 \, dt$$

Figure 4.28 and Figure 4.29 both illustrate the response of the simulated operational amplifier used in the Connection Machine simulation of the neural network architecture.

Figure 4.30 illustrates the convergence of the neural network simulation conducted on the Connection Machine. The simulation time step is 200 nanoseconds. This simulation converged to within 0.000001 of the results of the digital simulation of the network in approximately 18 milliseconds of real time.

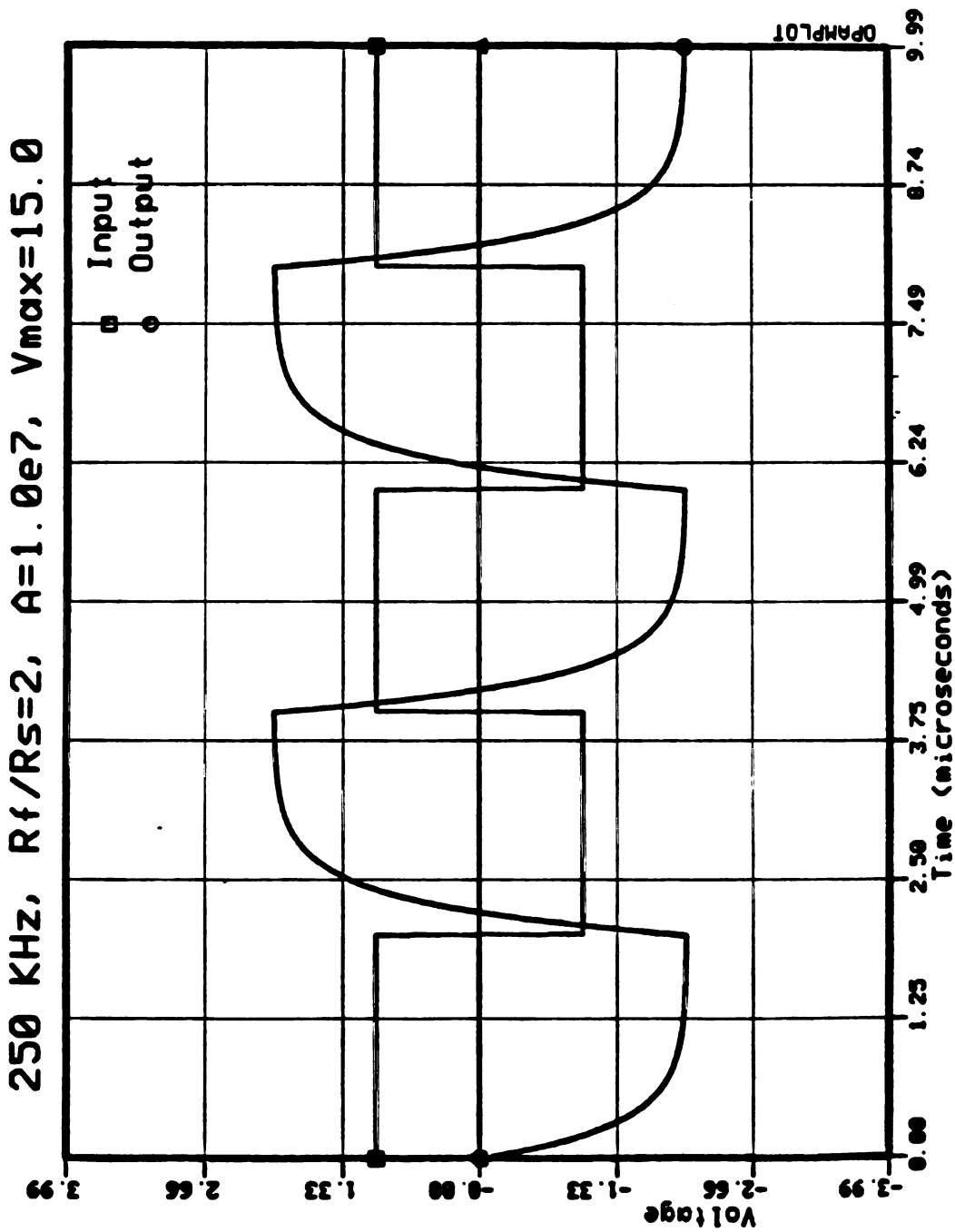

Figure 4.28: *Op Amp Simulation Using a 1 Volt Input With a Gain of 2*

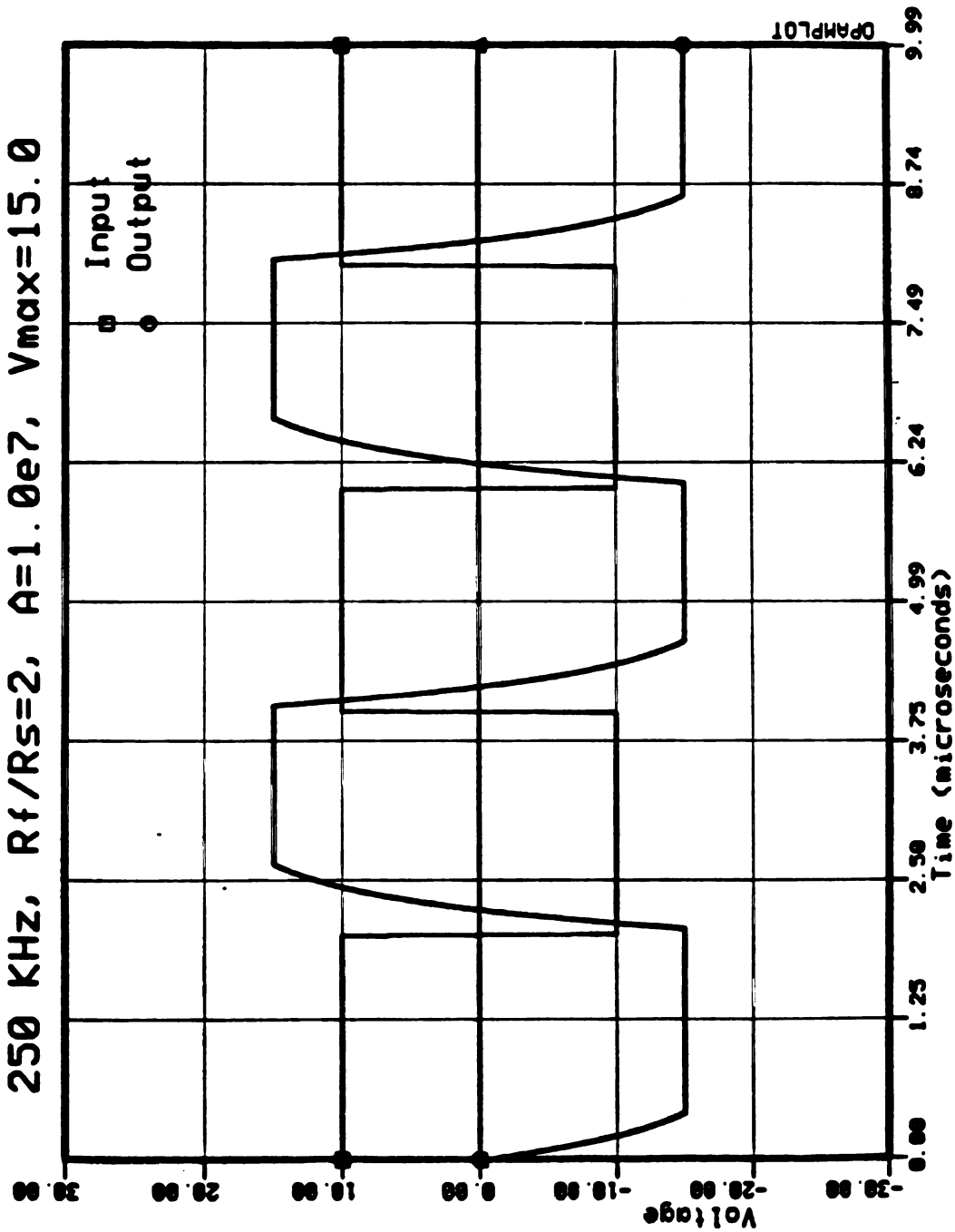

Figure 4.29: *Op Amp Simulation Using a 10 Volt Input With Gain of 2 and 15 Volt Limit*

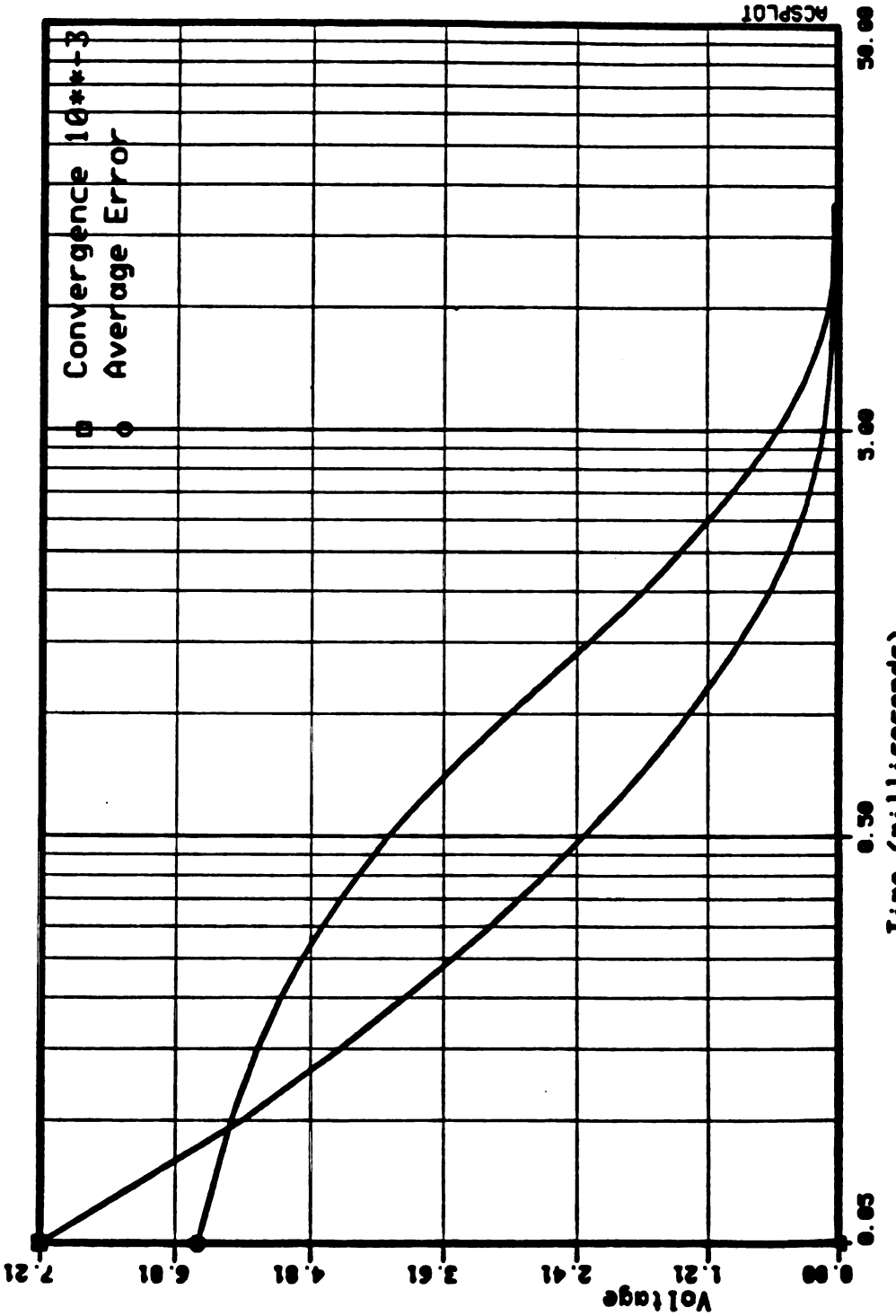

Figure 4.30: *Analog Circuit Simulation of 5-pt. Neural Network on CM-2 Showing Projected Real-Time Convergence*

4.5 Timing Results

Table 4.12 shows the timing results for the electrostatic model running on a VAX 4000 Workstation with a 6 MFLOPS performance for each of the test cases. The second column shows the size of the problem defined by the total number of neurons which correspond to the number of states in the problem. The third column is the time in milliseconds for the network to compute the potential for all nodes of the network. This time is divided by the network size and the result in contained in the fourth column as the average computation time per neuron for the parallel architecture. The average for all cases was 99.7 microseconds with a standard deviation of 8.2 microseconds. This data provides a baseline for comparison with the Connection Machine implementation and the projected analog circuit performance. It also indicates that the network scales linearly on the VAX (not surprising since the algorithm loops on the neuron indicies). More important is that this shows results for scaling with neural networks of significant sizes (441 neurons up to 3,600 neurons). This is eight times larger at the larger network. Many of the neural networks studied in the past have only dealt with neurons in the quantity of at most a few hundred neurons.

Table 4.13 shows the results of the average Connection Machine (CM-2) time per cycle update for the parallel network simulation. The results are in milliseconds. The average cycle time for all cases was 5.4 milliseconds with a standard deviation of 0.3 milliseconds. This parallel computation occurred using 16,384 processors for an actual network of 16,384 neurons. Using the results of 99.7

microseconds per neuron for the VAX, it would require $0.0997 \times 16384 = 1633.5$ milliseconds per cycle to simulate the 16,384 node network on the VAX. The ratio between these implementations is

$$\text{Speedup} = \frac{\text{VAX time for 16,384 neurons}}{\text{CM time for 16,384 neurons}} = \frac{1633.5 \text{ msec}}{5.4 \text{ msec}} \approx 303$$

Thus the Connection Machine implementation results in a performance improvement for the neural network of 300 times over the VAX due to its explicit parallelism. The Connection Machine employed was a 16K CM-2 at the Argonne National Lab. It is a data parallel computing system which associates one processor with each data element of an array. There are two parallel processing units, each of which has 16,384 data processors. Each data processor has 64K bits of bit-addressable memory and an ALU. The ALU is a bit processor, and arithmetic is a bit-serial process. One floating point accelerator exists for every 32 data processors. The performance specification for single precision floating point is 2500 MFLOPS[69]. Comparing this with the 6 MFLOPS performance of the VAX yields an expected speedup of 416. This is consistent with the results obtained above considering that timing for the neural network included other processing. This additional processing of the neuron outputs included saving previous values, range limiting, and clamping boundary conditions.

Since neural networks are fundamentally different from Von Neumann processors, they have their own performance criteria. Rather than using a measure of instructions per second (IPS), neural networks have been rated using the measure of connections per second (CPS). Table 4.14 gives the data for the neural network

Table 4.12: *VAX Timing Results for the Neural Network Simulation*

CASE	Network Size (n^2 Neurons)	Average Network Time (milliseconds)	Normalized Neuron Time (microseconds/neuron)
A	900	91.6	101.8
B	3,600	369.3	102.6
C	2,500	256.5	102.6
D	4,225	436.3	103.6
E	441	45.1	102.3
F	441	45.7	103.6
G	676	54.9	81.2

Table 4.13: *CM-2 Timing Results for the Neural Network Simulation*

CASE	Average Cycle Time (milliseconds)
A	5.5
B	5.9
C	5.5
D	5.2
E	5.1
F	5.5
G	5.1

performance in terms of connections per second. The Time/Neuron column is

Table 4.14: *Analysis of Connections per Second on the VAX*

CASE	Neurons	Connections	Time/Neuron (microsecs)	KCPS	Time/16k (millisecs)	B&B Time (seconds)
A	900	3,600	101.8	39	1,668	71
B	3,600	14,400	102.6	39	1,681	696
C	2,500	10,000	102.6	39	1,681	319
D	4,225	16,900	103.6	39	1,697	846
E	441	1,764	102.3	39	1,681	26
F	441	1,764	103.6	39	1,697	26
G	676	2,704	81.2	49	1,330	54

given in microseconds and has an average for all cases of 99.7 microseconds with a standard deviation of 8.2 microseconds. The KCPS column is the kilo-connections per second rate with an average of 40 KCPS and a standard deviation of 4 KCPS. The sixth column is the time required in milliseconds for a 16,384 node VAX cycle update. Its average is 1,636 milliseconds with a standard deviation of 134 milliseconds. Finally, the last column shows the time for the branch and bound algorithm to find the optimal path for each test case.

Recall that the time for the Connection Machine to update 16,384 neurons with 65,536 connections was 5.4 milliseconds. This gives an effective performance rate which would be defined in terms of the VAX performance as 12 million connections per second. This is approximately 300 times better than the VAX performance of 40 KCPS, and again, consistent with the 16k CM-2 performance.

Chapter 5

Discussion of Results

This chapter discusses the results obtained in the research. The first section presents a general result of the research concerning the approach for the design of massively parallel architectures. Following the discussion of this design paradigm, the path planning results in the areas of multiple path generation, variable connectivity, and network performance is discussed.

5.1 Design for Massively Parallel Architectures

Three distinct computer architectures are the Von Neumann processor, the multiprocessor network, and the massively parallel network. The Von Neumann architecture is characterized as a single processor (CPU) with a stored program memory. An example of the Von Neumann processor is a microprocessor such as the Motorola 68020 or the Intel 80386. The multiprocessor network is a group of Von Neumann processors communicating through an interconnection mechanism, such as a common bus. The interconnection topology is a federated arrangement, or sometimes a communications switching network. The neural network is an example of massive parallelism where an enormous number of

simple non-Von Neumann processing units are locally connected to many other processing units.

5.1.1 The Characteristics of Parallelism

A distinguishing feature of an architecture is its degree of parallelism. Parallelism exists at many levels within the complex structure of even a Von Neumann processor. At the electrical circuit level, parallelism is exhibited within semiconductors as analog signals such as the simultaneous propagation of current throughout the device. At the next level, these semiconductors are brought together into a digital device. Parallelism also is illustrated here as discrete voltage levels are changing simultaneously at various places within a digital circuit. Even at the hardware function level, such as I/O processing, instruction fetching, or cache memory operation, parallel activity occurs.

However, considering software as the means to specify computation activity, there is no parallelism in the algorithms programmed for a single CPU since the processing unit can decode and execute only one instruction at a time. More complex computations are involved with a single CPU than with the parallelism which is described next. For multiprocessor networks, the granularity of parallelism is coarse. Coarse granularity implies a low level of communication activity between the processors. These processors are performing simpler computations on the algorithm than in the single CPU. However, there is increased processing to handle the communications. The massively parallel architecture has a fine granularity and a high communication to computation ratio due to simple computations, and

a high degree of communication between adjacent computing units. This architecture has the algorithm "impressed" onto its medium in such a way that the algorithm solution emerges from local processing.

Consider the design approach in using these three architectures to solve problems. For the single CPU, classic design methodologies are used. These include flowcharts, stepwise decomposition, structured analysis and design, for example. Many of these methods are also used for multiprocessor designs, but additional complexity requires careful attention to algorithm requirements. Control and data partitioning of algorithms for process allocation over several multiprocessors is required. These requirements have to be carefully distinguished and managed to avoid inaccurate results and other harmful side effects.

Advancements in computer architectures and hardware technology have led to practical implementations of massively parallel computing[12, 15, 16, 28, 33]. The first fundamental question for this research from Section 1.2 asked if, in designing massively parallel architectures, any unifying principle exists for the distinct roles of application, algorithm, and architecture. Because of its characteristic of collective computation, a new way of thinking about the role of application, algorithm, and architecture in the design of massive parallelism was needed. The design challenge was to determine an appropriate model of the interdependence between the *application* problem to be solved, the *algorithm* which solves the problem, and the massively parallel *architecture* which solves the problem using collective computations. A key discovery of this research was a new design paradigm for massively parallel architectures which combine these interrelated

issues.

5.2 A Massively Parallel Architecture Design Paradigm

A new design paradigm of natural parallelism has been discovered in the course of this research. This paradigm is an architecture design concept for massively parallel architectures. The motivation for this paradigm was to meet a design need in the evolution of computer architectures towards massive parallelism. The paradigm of natural parallelism is hypothesized to meet this need. The results of experimental research presented in this dissertation have verified the usefulness of the paradigm.

A design paradigm needs to unify the issues of application, architecture, and algorithm in massive parallelism. This new method determines a mapping onto the processing elements and interconnections of the massively parallel architecture from the application model. This new paradigm for massively parallel architecture design is called *Natural Parallelism*. The natural parallelism paradigm uses phenomena in nature to suggest analogies for the computational model of the application problem.

5.2.1 The Paradigm of Natural Parallelism

Recent developments in massively parallel architecture technology has increased their capacity while shrinking their size. To capitalize on the advantages offered by these new architectures a new approach to their design is needed. This new approach is based on the increased density of hardware implementations

for massively parallel architecture technology. Processing units of the architecture are dedicated to explicit, unique state assignments of the problem, given a suitable state variable resolution.

Consider how discovery of nature is done. Nature is observed and a hypothetical model is conjectured to express natural law. This model is an approximation of the physical phenomenon which is observed, and expressed as mathematical equations in physics. When computing solutions to these equations, a numerical simulation of nature's behavior is often performed. Now consider this viewpoint reversed. Nature is the perfect analog computer for these same equations. Nature *is* the analogy for solving problems described by the mathematical equations of physics. Thus, the new approach for massively parallel design consists of 1) determining a natural phenomenon which is an analog of the abstract problem, 2) creating an analytical model of the natural phenomenon, 3) constructing a mapping of the analytical model to the abstract problem, and 4) implementing the massively parallel architecture for the mapping.

Recall that in Chapter 1, problems were described as state transition digraphs. Nature can also be described by states and operations which transform these states. This provides another analogy to massively parallel computation. Rather than the traditional sequential computer system in which a single processor fetches data from memory, modifies the data, and puts the data back into memory, consider memory which has a computation capability. This computation is accomplished by exchanging and modifying values among the data elements of the memory. When the data elements of the memory represent state assignments of

a problem, the memory values are possible state variable solutions. This structure provides an explicit mapping between processing nodes of a massively parallel architecture and state assignments of a physics problem. The mapping between state assignments of the natural analogy and the processing nodes occurs by a sampling of the state space into a computing grid with appropriate sampling resolution.

In the suggested analogy, the states of nature correspond to computing nodes, as described in the reversed viewpoint above. The explicit representation of the application problem states define the processing elements in the computing structure. Application solutions are computed numerically in a massively parallel architecture which is implemented in a grid structure corresponding to a sampling of states in the problem space.

5.2.2 Understanding the Natural Parallelism Paradigm

The natural parallelism paradigm is a synergistic concept. Synergism describes phenomena where elements of a system work cooperatively in such a way that the total effect of the system exceeds the sum of the effects of the independent elements. Similar concepts are available from other disciplines. Holism was defined in 1926 by evolutionists as the orderly grouping of unit structures into whole bodies or organisms in nature[67]. Holism was argued to be a synthesis which makes parts act as one. According to John Hopfield, the physiological computation of biological neural networks can be viewed holistically as spontaneous behaviour of the whole network:

> The bridge between simple circuits and the complex computational properties of higher nervous systems may be the spontaneous emergence of new computational capabilities from the collective behaviour of large numbers of simple processing elements[29].

The synergy described above is represented as collective behavior of simple processing elements. Similarly, the natural parallelism paradigm has a synergy arising from its own defined elements of application, algorithm, and architecture. Each of these elements contributes to a holistic effect of the design paradigm.

The relationship between the three elements of the natural parallelism paradigm is shown in Figure 5.1. Each element of the paradigm influences the other two elements. While individual descriptions of its elements do not characterize the synergy of the paradigm, it does help for understanding the concept of the paradigm. In this figure the aim is to get a sense of the unifying character of the paradigm. The paradigm tries to address massively parallel design in a holistic manner.

The three elements of the paradigm are algorithm, architecture, and application. The application is the problem requiring a solution, which is the purpose of the system. The algorithm is the model for using natural parallelism in the system. The architecture is the implementation of the system.

Consider the interaction between the elements of the paradigm as depicted by the arrows in Figure 5.1. The application concept suggests an analogy in nature which is the basis of the algorithmic model. The inverse relationship characterizes the type of problems which can be solved by the model. The algorithm defines the mapping needs for the architecture. The architecture, in turn, specifies the neuron and weight functions to the algorithm. The influence of the application on

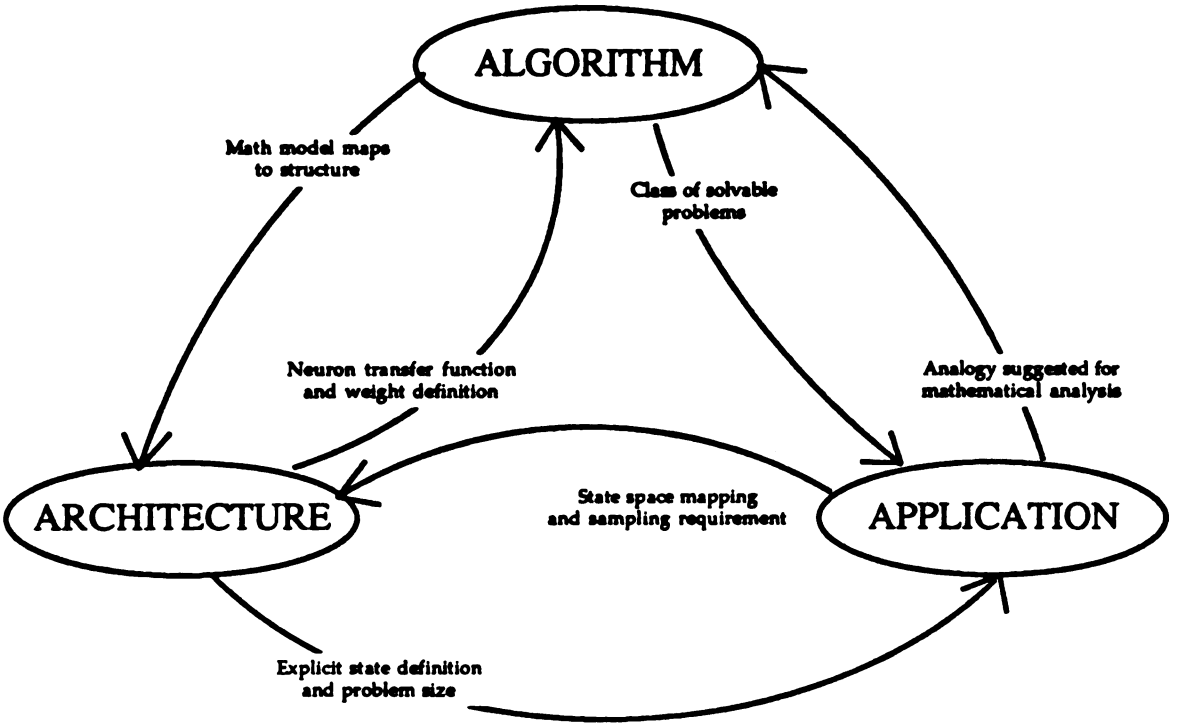

Figure 5.1: *Relationship Among Elements of the Natural Parallelism Paradigm*

the architecture develops state space mapping requirements such as the spatial sampling resolution. Conversely, explicit definition for the application problem states is provided by the architecture and its problem size is scoped.

5.2.3 Using the Natural Parallelism Paradigm

In Figure 5.2 the process of applying the paradigm of natural parallelism is illustrated. The process begins with the selection of some problem of interest. The first step in the process is to look for analogies in nature for the selected problem. When an appropriate analogy is found, the *natural parallelism concept* is formed. This concept from the paradigm is the key to the massively parallel design. The relationship of the concept to the application is understood and is

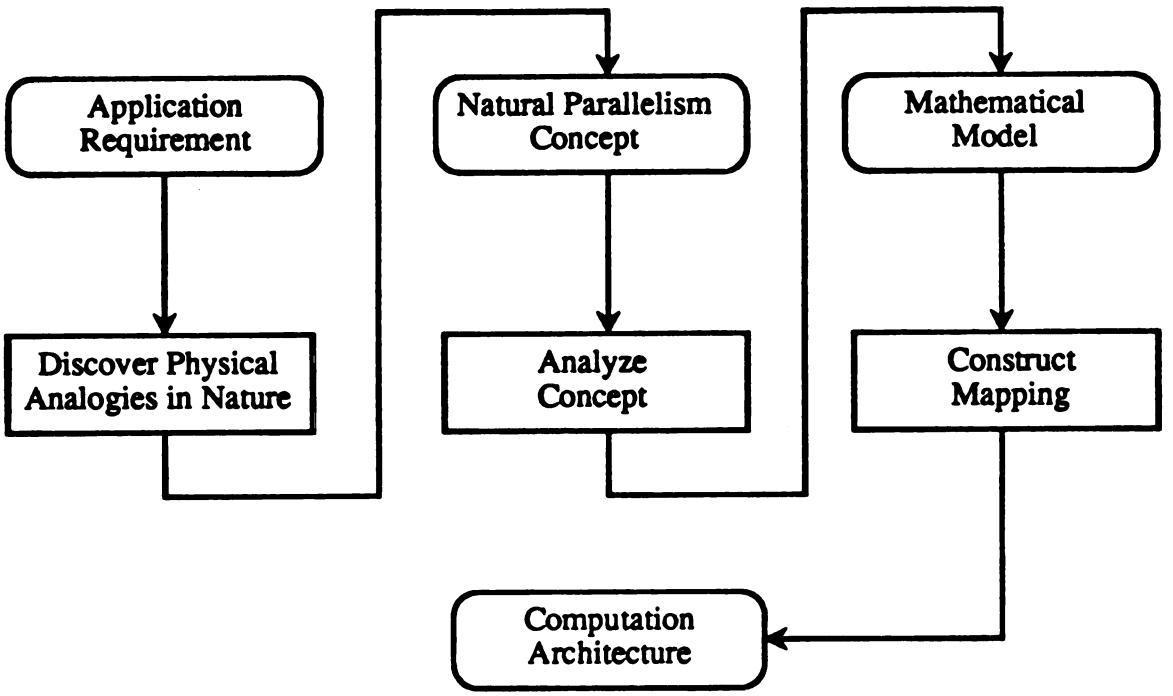

Figure 5.2: *Applying the Natural Parallelism Paradigm*

explainable. The algorithm is identified with the natural law which applies to the concept. The architecture is mapped through analysis of the concept to define the computational structure of the network. The natural law for specifying the algorithmic solution to the problem is a result of the analysis of the physics model used in the natural parallelism concept. The mathematical model of the natural law is used in constructing the computation structure of the architecture.

In the introduction to this dissertation, a central thesis was posed. It stated that a massively parallel computer architecture could be defined where the computational units correspond to explicit problem states converging to problem solutions. The natural parallelism design paradigm has achieved this objective.

5.3 Discussion of Path Planning Architecture Results

Chapter 4 presented the results of the artificial neural network architecture designed for path planning. Assessment of the data from all of the test cases showed that the value of the paths generated by the neural network had an average difference of 8.3 percent over the value of the optimal path for the test case. Even if 8.3 percent is a higher penalty for using the neural network than desired in the path planning application, the benefits of the multiple paths generated are the compensation of the method. The data revealed a significant effect from the multiple path generation. This effect was that the costs of the top-ranked paths generated by the neural network increased linearly for at least the first half of the alternative paths. Furthermore, the rate of increase averaged to be only 0.22 percent of value of the least cost path found by the neural network. So although the value of the least cost path may be higher than the optimal path, the architecture will generate many additional paths which do not significantly degrade in cost.

Chapter 6

Research Conclusions

The research of this dissertation dealt with issues of architecture design in massive parallelism. In Chapter 1, the requirements associated with large complex problems created the need to have a closer match between the application problem, and the massively parallel architectures and algorithms. The discovery of a paradigm to guide the design of massively parallel architectures is one of the significant results of the research. This paradigm represents the development of a working synergy between the traditionally distinctive elements of application, algorithm, and architecture in computer system design. A second significant result of this research was the use of the paradigm to design massively parallel architectures. An artificial neural network was designed which can solve a class of partial differential equations. These designs were used as examples to test the paradigm. A third significant result of this research was showing how the architectures can be applied to path minimization problems.

6.1 Path Planning Using the Electrostatic Model

One of the most difficult tasks of this research was to develop a model of path planning search which had a true parallelism inherent in the model. This interest came about since strictly sequential algorithms were not directly applicable to artificial neural network architectures. Parallelized versions were not of interest either since they are targeted for loosely coupled multiprocessor systems. The path planning model required the feasibility of a more direct use on tightly coupled architectures such as artificial neural networks.

The second interest, and perhaps a more compelling requirement for path planning, was a desire for the resulting system to produce alternative sub-optimal solutions. These alternative paths provide choices at the starting node to the user of the path planning system. For various operational considerations, users often desire alternative choices when employing automated planning systems. But this is a very difficult proposition for classical sequential search algorithms. The principle of optimality does not support the consideration of simultaneous, sub-optimal decisions.

It was at this point that the electrostatic model was conjectured to satisfy these requirements for a directly parallelized model and alternative sub-optimal paths. The idea was simple enough — nature "automatically" distributes electrical current through conducting material in an optimal manner, although this optimization criterion is different from that for a least cost path. Could the current flux lines then be used as the alternate paths? The result is that they can as was shown in

Section 3.2. These alternative paths make intuitive sense. They avoid regions of high cost and prefer regions of low cost.

The paths are also parallel alternatives which have an incremental variation between the adjacent paths. Neighboring (adjacent) paths are defined using the theory of the calculus of variations. The paths are defined by a function. This function, when adjusted by a "variational" increment, defines a new path neighboring the original path[7]. This variational increment is very similar to a differential, only applied to functionals rather than to variables. Consider a uniformly conducting material with dipole charges. The electrical current flux lines for this material look very similar to the magnetic field lines which form between the north and south poles of a bar magnet. As the conducting material is modified so that its conductivity is no longer homogeneous, the flux lines will warp and bend simultaneously to accommodate the changes in the conductivity of the material. Yet they remain adjacent to each other in a variational sense, preserving the alternative choices for the user. Thus, the electrostatic model was perceived as a viable approach to modeling this path planning problem.

What was needed next was an approach to implement this model on an artificial neural network.

6.2 Designing the Neural Network Architecture

The electrostatic model provided another important benefit in the research — a sound mathematical basis. This was due to the nonhomogeneous partial differential equation (Laplacian) of the electrostatic potential field. With the boundary

conditions applied, the flux lines could be extracted by a gradient descent over the potential surface from the source node (maximum potential boundary condition) to the sink node (minimum potential boundary condition). Therefore, the challenge was to compute a solution to the partial differential equation using an artificial neural network.

The means to accomplish this was a finite difference approximation to the partial differential equation developed in Section 3.3.1. A study of the finite difference approximation formulas revealed a striking similarity to the familiar equations for the artificial neural network. Thus, by an appropriate mapping of the coefficients of the finite difference approximation formulas into the weights of the artificial neural network, the neurons could compute the numerical solution to the partial differential equation. This design was successfully accomplished in this research and described in Section 3.3.2.

A significant result of this approach is that the resulting design benefits from the classical numerical analysis of finite difference approximations for partial differential equations. Since the artificial neural network directly implements the finite difference approximation (in a truly parallel architecture), it retains the convergence and accuracy properties of the finite difference approximation. Additionally, the artificial neural network scales directly with the size of the problem just as is the case with the finite difference approximation. With this architecture design in place, the influence of the amount of interconnections could be investigated.

6.3 Influence of Variable Connectivity

The finite difference approximation formulas map into simple interconnections between adjacent neurons in the artificial neural network. Improvements to the convergence of the network were conjectured to result from increasing the interconnections between neurons beyond the locally adjacent neurons to larger regions of the network. This required a way to increase the number of input connections to each neuron from other neurons, and yet preserve the integrity of the finite difference approximation.

The solution was found in the parallel summation technique that was developed in Section 3.3.4. This approach preserved the fundamental finite difference approximation formula as a template kernel so that the approximation of the derivative would be consistent in larger interconnection patterns. The technique consisted of recursively unfolding the kernel to rewire the input connections of each neuron directly to those neurons whose outputs were directly connected to the neuron whose outputs were the source of the input connections. Each recursive unfolding of the approximation template resulted in a higher amount of interconnections. This was shown in Section 4.3.2 to successfully reduce the convergence time and yet preserve the accuracy of the approximation. The benefit of the variable connectivity is to give the neural network designer tradeoffs between the convergence time, accuracy, and network costs (measured in terms of the implementation complexity through connectivity). For a given accuracy and time requirement, this tradeoff then specifies the amount of interconnectivity that

is necessary in the artificial neural network. Conversely, for an embedded implementation of an artificial neural network with a given amount of connectivity, the design tradeoff can be used to project the accuracy obtainable from the network in a specified time interval.

6.4 The New Paradigm of Natural Parallelism

A new principle to guide the design of massively parallel architectures was conceived in this research. The paradigm which was conceived is as unique to massively parallel architectures as the sequential thinking process is unique to Von Neumann architectures. This paradigm was called natural parallelism and can be understood as nature acting as a massively parallel analog architecture. Nature serves as the ultimate massively parallel architecture. As such, natural phenomena can be used as analogs for abstract computation problems. Once discovered, these natural analogs can be analyzed to develop the mathematical model foundations upon which massively parallel architectures can be constructed.

This concept provided a synergy that unified the distinctive Von Neumann treatment of application, algorithm, and architecture in computer design.

6.4.1 Natural Parallelism Related to Algorithms

The natural law which was employed to investigate the paradigm was a class of partial differential equations in mathematical physics. These partial differential equations describe physical phenomena in field theory. Using natural parallelism,

these physical phenomena were shown to describe certain optimization problems. Nature was viewed as an optimization process, exhibiting equivalent behavior for which mathematical expressions are available (that is, the partial differential equations of field theory).

A neural network architecture was designed in the research to solve a class of partial differential equations. The significance of this design was a neuron transfer function and weight formulas for computing the finite difference approximation.

6.4.2 Natural Parallelism Related to Architectures

The partial differential equations were mapped onto the massively parallel architecture using finite difference approximations. The numerical solutions of the approximation formulas effectively simulated the analog computation of nature. An architecture was designed to compute a five-point finite difference approximation solution to these equations. The five-point finite difference approximation formula provided for the definition of interconnection weights in the architecture.

The massively parallel computing architecture was defined using explicit states of the problem as the computing nodes. This architecture computed solutions by exchanging information along interconnections defined in a mesh topology. As this exchange continued, each node's output converged to a value which represented the corresponding state's value in the problem solution. These state outputs are the scalar potential field solution to the partial differential equation. The approximation templates for the finite difference formula defined the interconnection weights for the particular architecture configuration.

This research resulted in a parallel summation technique using variable connectivity to perform the integration necessary for the finite difference approximation in the neural network. This technique was significant in its ability to define varying degrees of interconnection between the neurons. This was necessary to accomplish and assess the convergence of the network with different amounts of interconnection. The results of the connectivity study verified the hypothesis that increased convergence occurs as the interconnections increase.

6.4.3 Natural Parallelism Related to Applications

The application of primary interest was path planning. This required searching for a path in a space where a cost function was defined. The path end points were defined, and the path with the least total cost along its length was the minimum cost solution path to the problem.

The paradigm was employed to define a massively parallel architecture to solve this problem. Natural parallelism provided a model in field theory as an analog to this problem. This was the electrostatic model. The architecture design for the electrostatic model used an interconnection definition which supported variable connectivity. This architecture was shown to provide multiple, alternative solution paths. The architecture also was used to study the effect of connectivity on the solution convergence.

This research has shown the use of the natural parallelism paradigm to solve path planning applications. Artificial neural networks designed using the natural parallelism concept were shown to provide paths which correspond to a natural

analogy of field theory in electrostatics. The electrostatic model architecture was shown to compute multiple, alternative paths. The cost of these multiple path solutions showed only a linear increase among a significant number of the available alternative solutions.

6.5 Summary

In Chapter 1, several questions were raised to motivate and direct this research. As a result of this research, several observations and answers can be made on the issues they address.

The finite difference approximation method was used to define a numerical solution to the problem. The approximation template coefficients provided the definitions for the weights of the neural network. The template also determined the number of interconnections between the neurons, and their interconnection pattern. The results on variable connectivity illustrated the influence of the number of interconnections on the convergence of the neural network to stable states. The approximation also defined the neuron transfer function.

Regarding the design process questions, the natural parallelism paradigm solves the dilemma on the interaction of algorithm, architecture, and application issues. It also provided for an underlying mathematical model for the architecture from the natural analogy in physics. The model further provided for the mapping of the problem onto the neural network through the finite difference approximation.

6.5.1 Suggestions for Further Research

An interesting followup to this research would be to further test the concept by implementing the architectures in a path planning system environment. A first step would be to include the architectures in a real-time environment such as an aircraft simulation.

Further development could be pursued by the incorporation of the designs into hardware prototypes. Various hardware implementations are suggested. The simulation of the designs can be written in the VHSIC Hardware Design Language (VHDL). This would provide a behavioral circuit description of a network engine implementation. Or, a fully parallel VLSI design could be developed to fabricate a chip set to realize the networks. Other technologies such as optical computing can be examined for implementation.

The robustness of the design paradigm could be further examined by using the application of constrained optimization. Constraints should be modeled into the analogy as is the basic model. One example is constraining paths to a certain sector about the current vehicle state vector. The analogy for the electrostatic model can use zero conductivity regions behind the vehicle whose radii correspond to the constraint in order to restrict the field lines to the desired sector ahead of the vehicle.

Another improvement to the architecture design concerns the approximation technique. This research used finite difference formulas for the numerical approximation templates. Finite element modeling is another technique that could be

investigated to implement the architecture design.

Another issue is the refinement of the architecture design to develop adaptive mesh topologies for the interconnection topology and node definitions for the grid. Node definitions and interconnection weight adjustments would need to be defined with adaptive mesh generation algorithms. The self-organizing map is another neural network architecture which adjusts its connection weights in order to learn patterns in data. This technology may be a useful approach to map the variable structure of the cost functions into an adaptive mesh configuration.

Explicit representation of the state space in the architecture assumes the availability of the cost function values. However, this requires tremendous I/O capability. This presents a problem for implementation in supporting such a large bandwidth for data. In addition, not all cost functions can be explicitly defined in a spatial organization.

There are certainly other applications besides path planning that can benefit from the natural parallelism paradigm. Other applications should be examined to develop their natural analogies for mapping onto massively parallel architectures. Certainly the mathematical models of the analogies themselves can be solved using the architectures developed in this research. One possible application is the solution to computational fluid dynamics problems. Another example is the use of the architecture to solve some linear algebra problems.

Finally, other design paradigms may also be appropriate for the development of massively parallel architectures. The design paradigm of natural parallelism is just one of the possible techniques for envisioning solutions to the synergy

problem associated with these architectures. Hopefully, this research has enlightened the subject of massively parallel architecture design to suggest alternative methodologies in their analysis and development.

Appendix A

Description of the Test Cases

In this appendix, the test cases used to conduct the experiments in this research are described. There were eight different test cases used, as shown in Table A.1. The column labelled "N" is the size of the square grid used for the test.

Table A.1: *Problem Space Size and Path Endpoints*

CASE	N: Size	S: Start	G: Goal
A	30	(3, 23)	(17, 2)
B	60	(20, 10)	(45, 55)
C	50	(8, 42)	(37, 11)
D	65	(14, 35)	(50, 50)
E	21	(13, 5)	(11, 17)
F	21	(17, 5)	(5, 17)
G	26	(3, 4)	(24, 24)

Thus, the total number of grid points in each test case is N^2. The next column, labelled "S", contains the coordinates of the start node on the grid. The column labelled "G" lists the coordinates of the goal node.

A.1 Test Case A

A diagram of the cost layout for Test Case A is located in Figure A.1. Test Case A is a 30 by 30 grid with five regions containing different uniform cost

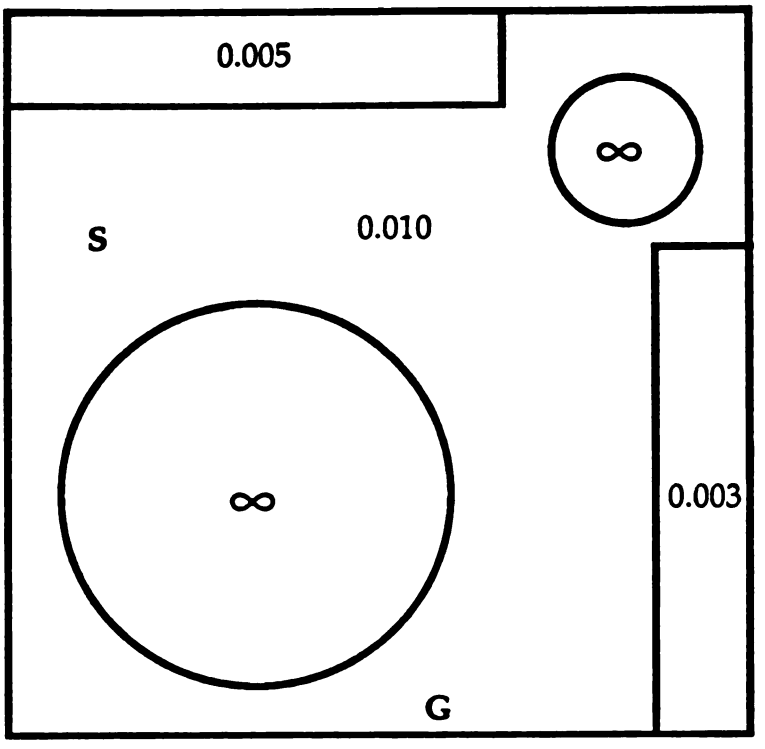

Figure A.1: *Diagram of the Cost Layout for Test Case A*

values. Each region in the diagram contains a number which is the cost for each cell in that region. The background region has a cost of 0.010, and contains the start node at (3,23) and the goal node at (17,2). Between the start and goal node is a circular region of infinite cost. At the top and right of the grid are two rectangular regions with lower costs of 0.005 and 0.003, respectively. Another smaller circular region of infinite cost is located in the upper right corner of the grid between the smaller cost strips. These regions were designed to bring a nonuniform distribution of cost regions to bear on the warping of the multiple path solutions. The cost function contour plot with the optimal path is shown in Figure A.2.

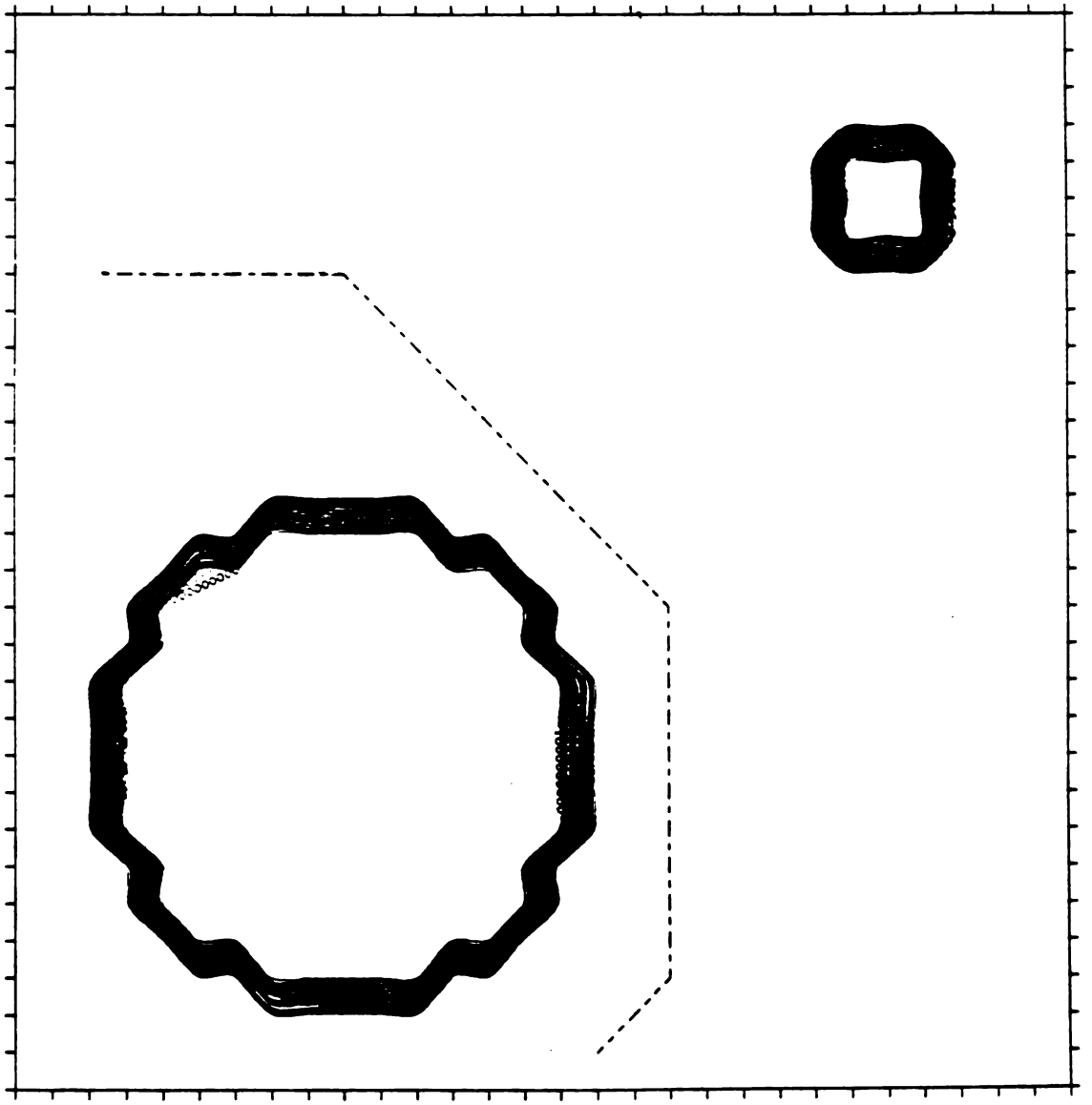

Figure A.2: *Test Case A Cost Function Contours and Optimal Path*

A.2 Test Case B

Test Case B is a threat penetration scenario using synthetic terrain elevation and threat coverage. The terrain which was generated is shown in Figure A.3. In the perspective view of the figure, the nearest corner is the southwest corner of the grid and the farthest corner is the northeast corner of the grid. This terrain covers an area of 3600 square nautical miles. The grid is measures 60 by 60 where each cell is one square mile of the region. The terrain was generated to establish a scenario where a mountainous region had a valley which provided a route along which defensive threats could be deployed. Threats with different effective ranges and risks were then added into the region, taking the terrain into account. The combined risk of these threats was then analyzed by line-of-sight calculations, taking the effects of terrain masking into accout, to generate the cost function for the search. The costs were obtained for a low altitude flight which was constrained to stay 1000 feet above the terrain. The resulting cost function is used in Test Case B, and shown in Figure A.4 along with the optimal path for the start and goal nodes.

A.3 Test Case C

Test Case C is another threat penetration scenario, but this case uses actual terrain elevation data. The region of the data is near the town of Fulda, Germany. This data has been used in several studies[53, 54, 55] because many military analysts believe this was the location where Soviet tanks would pour into the

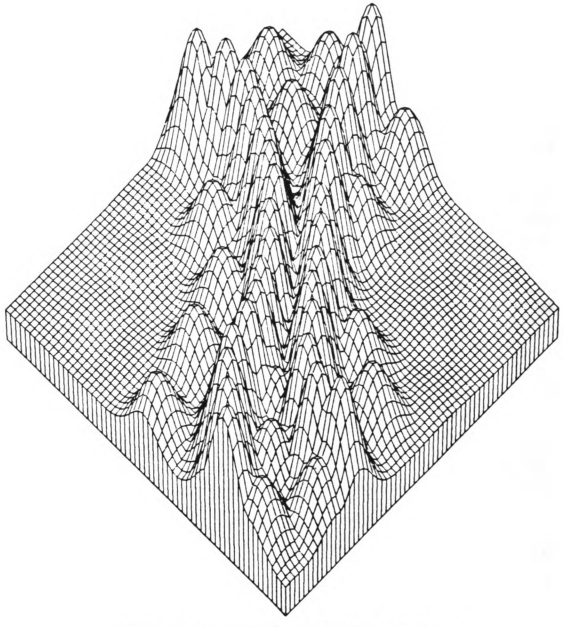

Figure A.3: *Synthetic Terrain Data for Test Case B*

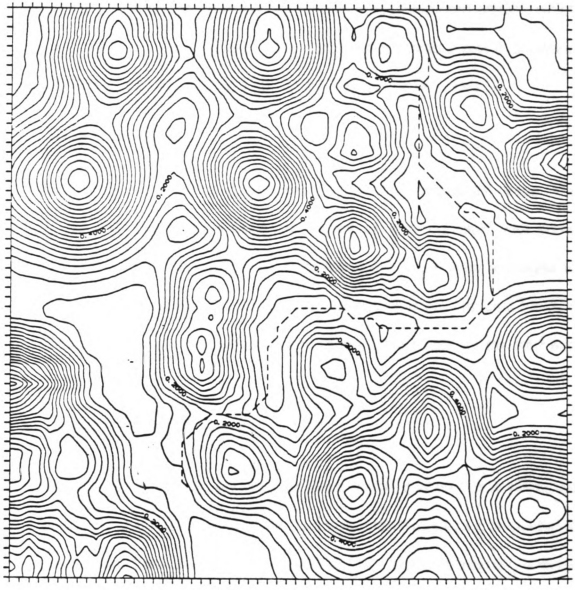

Figure A.4: *Test Case B Cost Function Contours and Optimal Path*

West at the beginning of a major offensive by the Warsaw Pact. Representative models of surface-to-air missle radar sites are used in the generation of the risk data for the cost function. The grid for this test case is a 50 nautical mile by 50 nautical mile area, with the grid cells at one nautical mile increments. One features of the cost function is that it has moguls in a checkboard pattern, which increase in value from west to east. The cost function contour plot is shown in Figure A.5 along with the optimal path for the start and goal nodes.

A.4 Test Case D

Test Case D uses a smoother cost function than the previous test cases. The data for the cost function is a spherical surface of large curvature with some smaller, local bumps placed on top of the surface. These were averaged to generate a smooth surface with some local variation in cost. The cost function contour plot is shown in Figure A.6 along with the optimal path for the start and goal nodes.

A.5 Test Cases E, F, and G

Test Cases E, F, and G, are cost functions which contain highly patterned regions of high costs. A diagram of the cost layout for Test Case E is located in Figure A.7. Test Case E is a 21 by 21 grid with two regions containing uniform cost values. The background region has a cost of 1.0, and contains the start node at (13,5) and the goal node at (11,17). Between the start and goal node is a rectangular strip with a cost of 4.0. The cost function contour plot with the

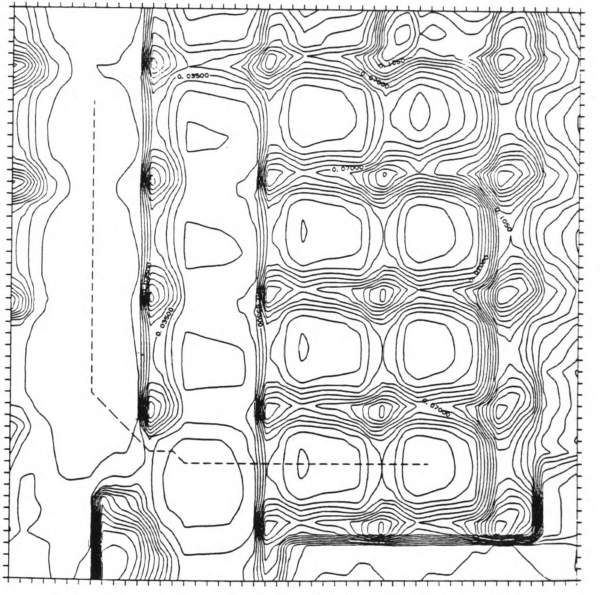

Figure A.5: *Test Case C Cost Function Contours and Optimal Path*

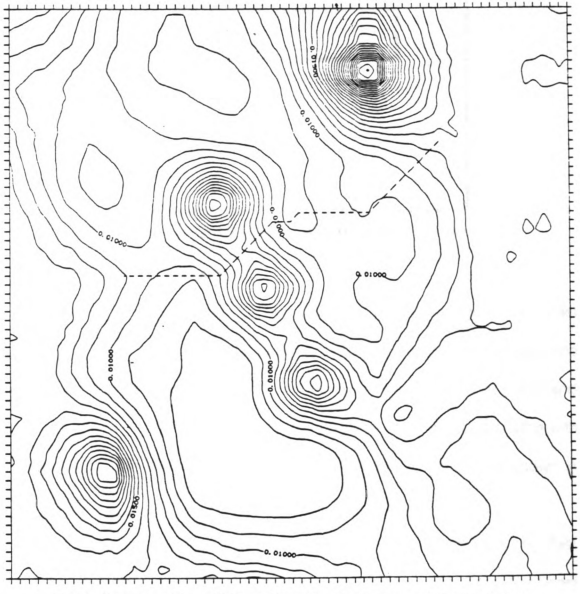

Figure A.6: *Test Case D Cost Function Contours and Optimal Path*

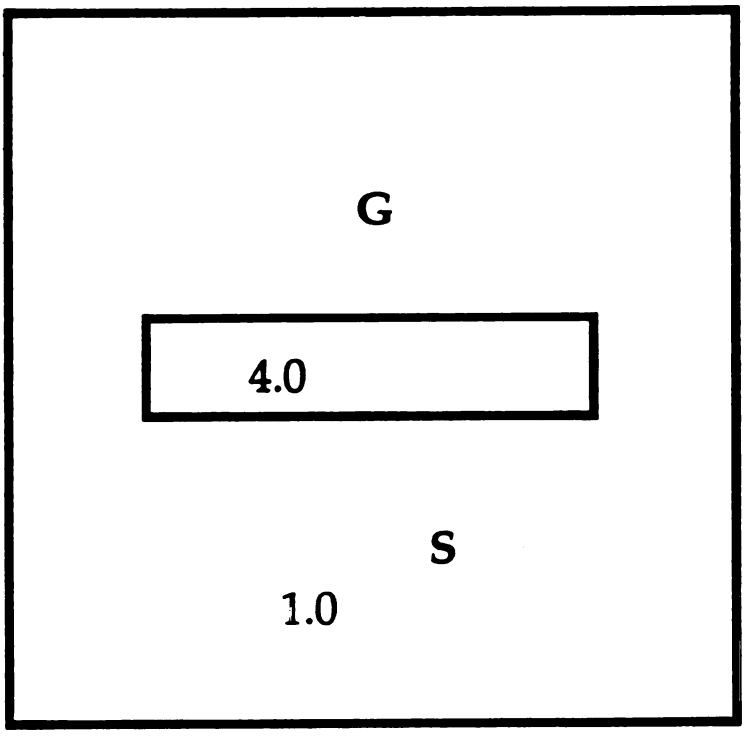

Figure A.7: *Diagram of the Cost Layout for Test Case E*

optimal path is shown in Figure A.8.

A diagram of the cost layout for Test Case F is located in Figure A.9. Test Case F is a 21 by 21 grid with two regions containing uniform cost values. The background region has a cost of 1.0, and contains the start node at (17,5) and the goal node at (5,17). Between the start and goal node is a triangular strip with a cost of 4.0. The cost function contour plot with the optimal path is shown in Figure A.10.

A diagram of the cost layout for Test Case G is located in Figure A.11. Test Case G is a 26 by 26 grid with nine regions containing two different uniform cost values. The background region has a cost of 1.0, and contains the start node at (3,4) and the goal node at (24,24). Between the start and goal node are several rectangular regions with a cost of 4.0. The cost function contour plot with the

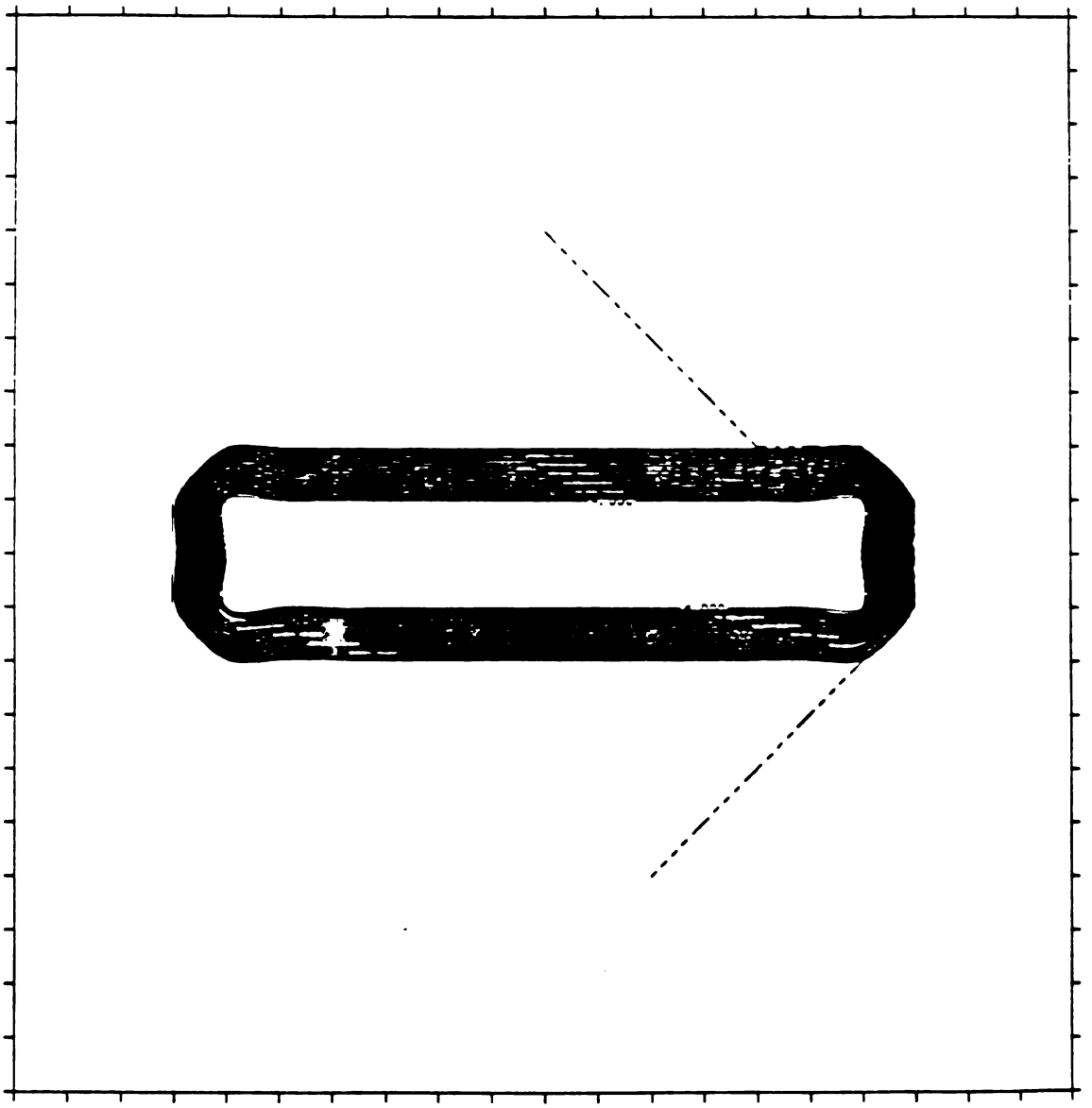

Figure A.8: *Test Case E Cost Function Contours and Optimal Path*

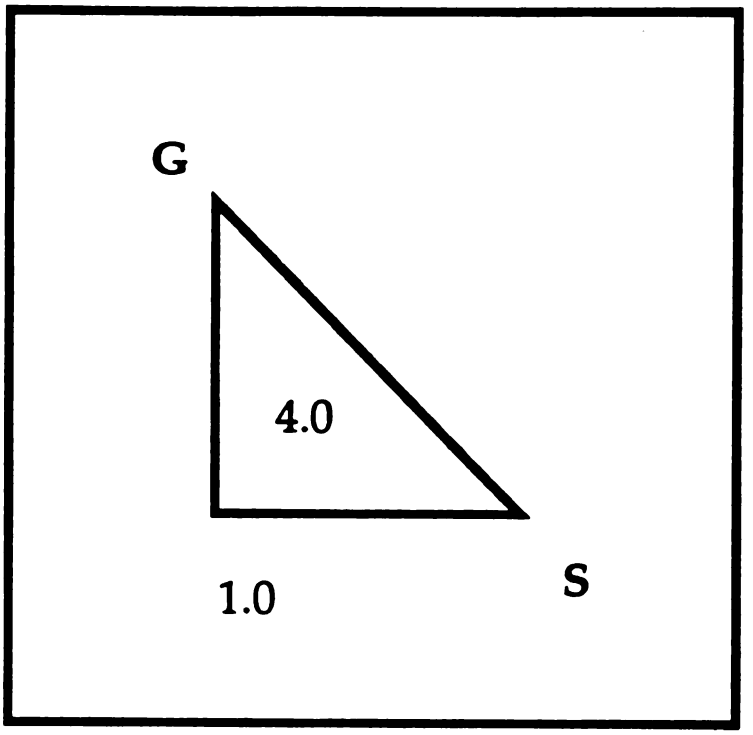

Figure A.9: *Diagram of the Cost Layout for Test Case F*

optimal path is shown in Figure A.12.

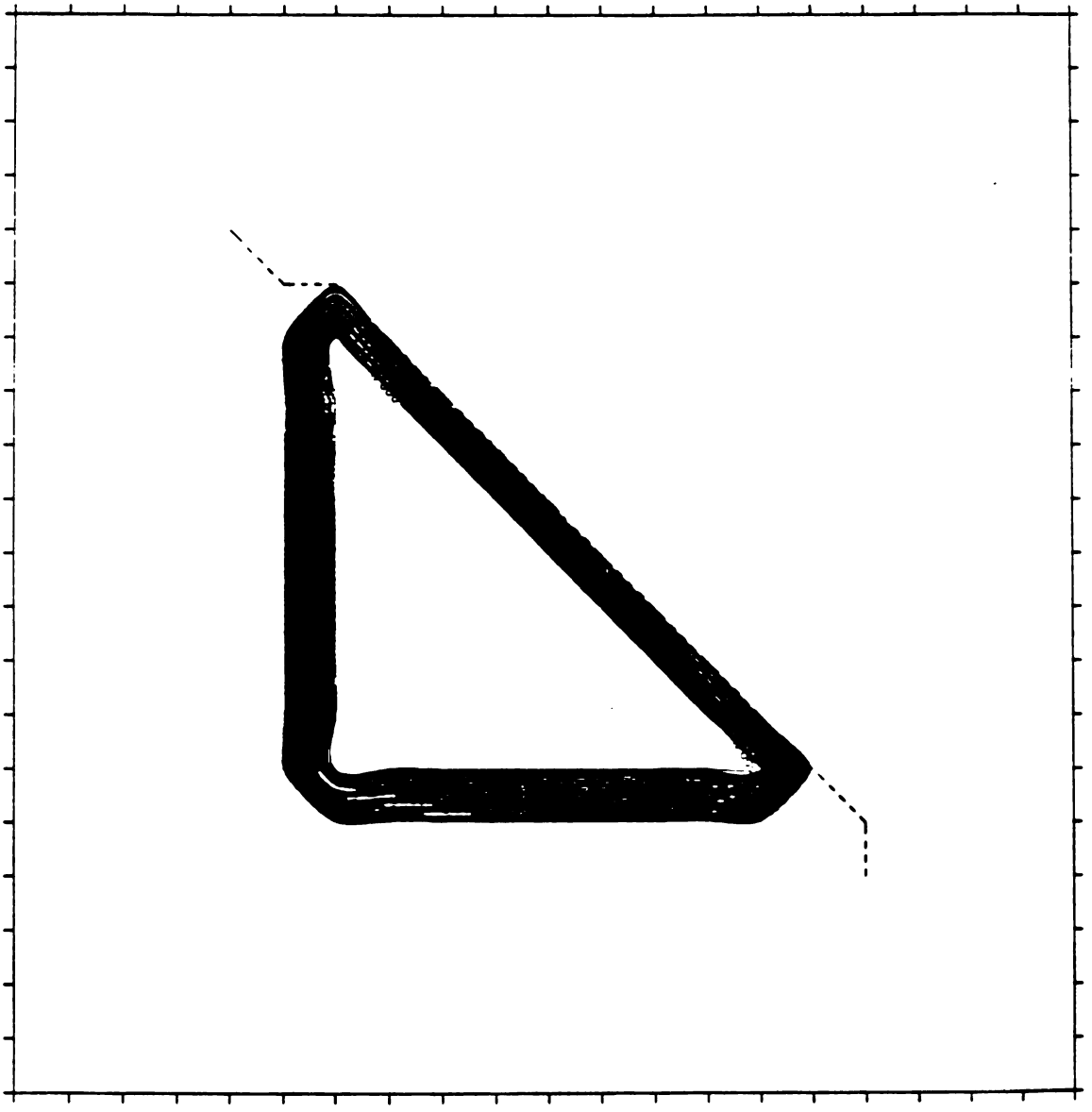

Figure A.10: *Test Case F Cost Function Contours and Optimal Path*

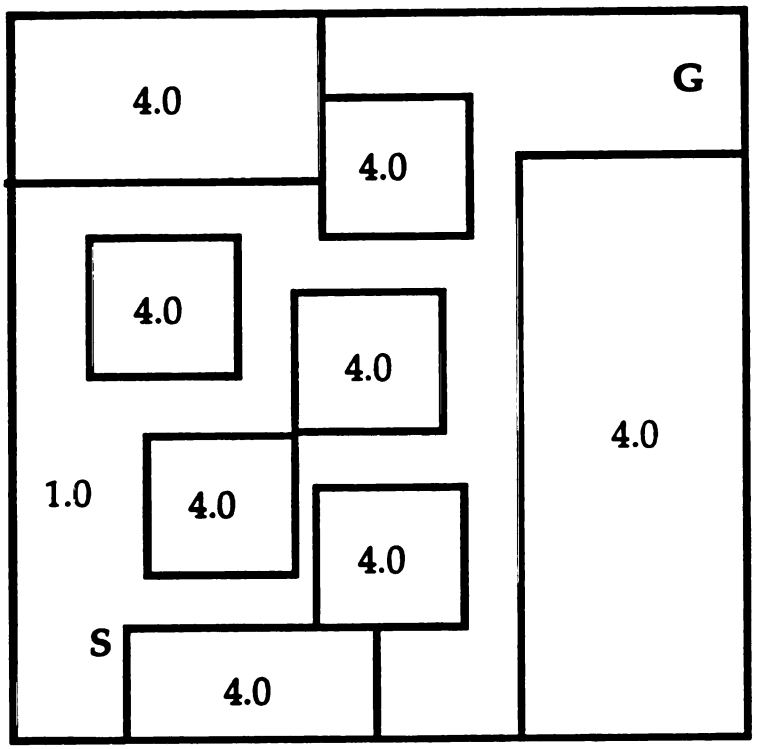

Figure A.11: *Diagram of the Cost Layout for Test Case G*

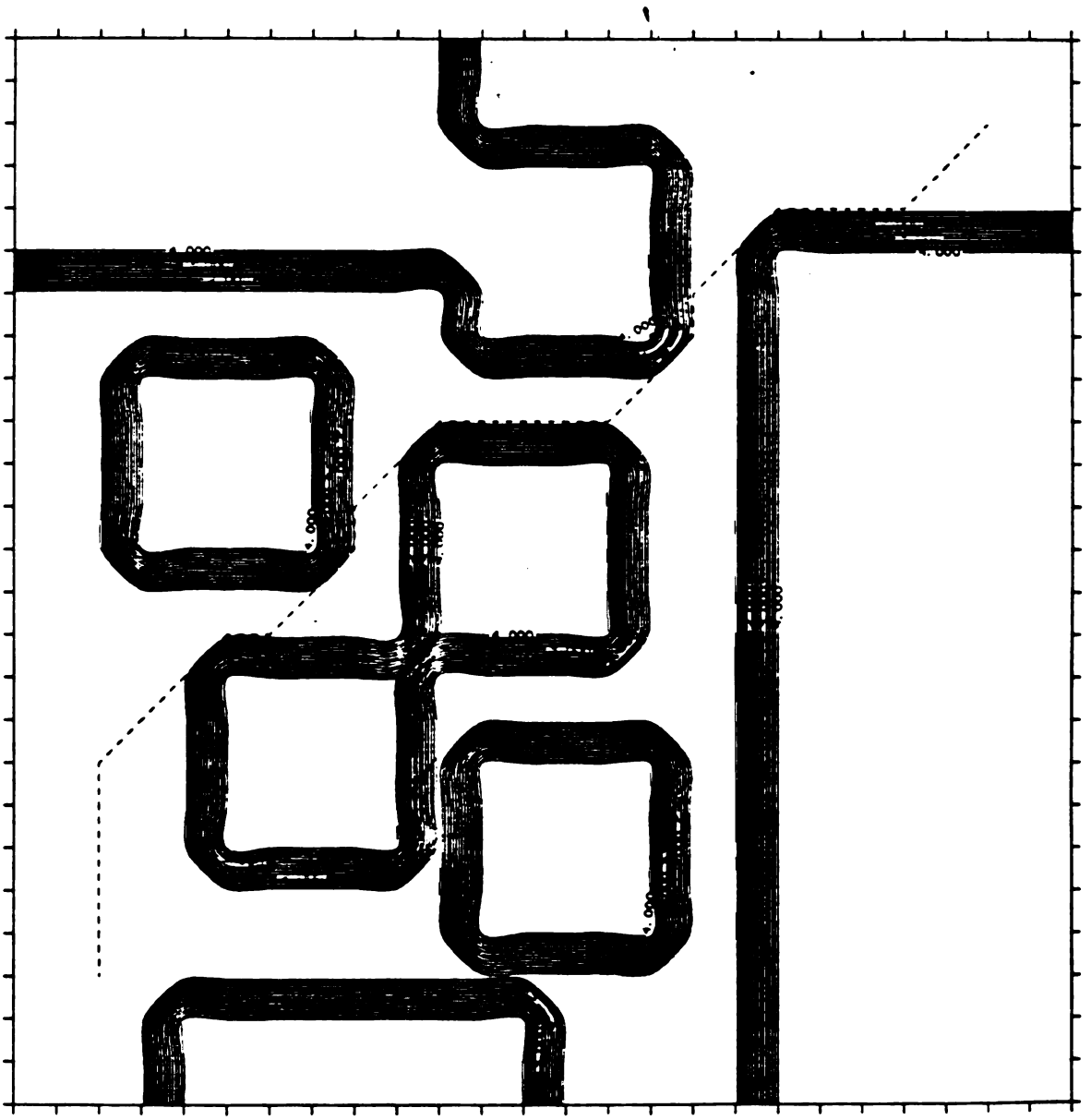

Figure A.12: *Test Case G Cost Function Contours and Optimal Path*

Bibliography

Bibliography

[1] Jeffrey Michael Abram. *Shortest-Path Algorithms with Decentralized Information and Communication Requirements*. DSc.thesis, Washington University, 1981.

[2] L. A. Akers, M. R. Walker, D. K. Ferry, and R. O. Grondin. Limited interconnectivity in synthetic neural systems. R. Eckmiller and C. von der Malsburg, editors, In *Neural Computers*, pages 407–416, Springer, Berlin, Heidelberg, 1988.

[3] A. Barr and E.A. Feigenbaum. *Handbook of Artificial Intelligence*. Kaufmann, 1981.

[4] R.E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[5] R.E. Bellman and S.E. Dreyfus. *Applied Dynamic Programming*. Princeton University Press, 1962.

[6] M. Bird, M. Olinger, R. Denton, and J. Jones. The trajectory generator for tactical flight management. In *Proceedings of the 1983 IEEE National Aerospace and Electronics Conference*, 1983.

[7] Gilbert A. Bliss. *Lectures on the Calculus of Variations*. University of Chicago Press, 1946.

[8] Oskar Bolza. *Lectures on the Calculus of Variations*. Dover Publications, 1961.

[9] C. Carathéodory. *Calculus of Variations and Partial Differential Equations of the First Order*. Holden-Day, 1965.

[10] Lothar Collatz. *The Numerical Treatment of Differential Equations*. Springer-Verlag, New York, NY, third edition, 1966.

[11] R. Courant and D. Hilbert. *Methods of Mathematical Physics*. Interscience Publishers, 1953.

[12] Judith E. Dayhoff. *Neural Network Architectures*. Von Nostrand Reinhold, 1990.

[13] R.V. Denton and J.P. Marsh. Applications of autopath technology to terrain/obstacle avoidance. In *Proceedings of the 1982 IEEE National Aerospace and Electronics Conference*, 1982.

[14] S.E. Dreyfus and A.M. Law. *The Art and Theory of Dynamic Programming*. Academic Press, 1977.

[15] R. Eckmiller, editor. *Advanced Neural Computers*. North-Holland, Amsterdam, 1990.

[16] R. Eckmiller, G. Hartmann, and G. Hauske, editors. *Parallel Processing in Neural Systems and Computers*. North-Holland, Amsterdam, 1990.

[17] R. Eckmiller and C. von der Malsburg, editors. *Neural Computers*. Springer, Berlin, Heidelberg, 1988.

[18] Omer Egecioglu, Terence R. Smith, and John Moody. Computable functions and complexity in neural networks. In *Real Brains, Artificial Minds*, pages 135–164, Elsevier Science Publishers, 1987.

[19] Philip H. Enslow, Jr. *Multiprocessors and Parallel Processing*. John Wiley & Sons, 1974.

[20] H. Erzberger and H. Lee. Constrained optimum trajectories with specified range. *Journal of Guidance and Control*, 3(1):78–85, 1980.

[21] S. E. Fahlman and G. E. Hinton. Connectionist architectures for artificial intelligence. *Computer*, 100–109, January 1987.

[22] Richard P. Feynman. *The Feynman Lecture Series on Physics*. Addison-Wesley Publishing Co., 1963.

[23] George E. Forsythe and Wolfgang R. Wasow. *Finite-Difference Methods for Partial Differential Equations*. John Wiley & Sons, Inc., New York, NY, 1960.

[24] J.A.B. Fortes and B.W. Wah. Systolic arrays — from concept to implementation. *Computer*, 20(7):12–17, 1987.

[25] L. Fox. *Numerical Solution of Ordinary and Partial Differential Equations*. Pergamon Press, Oxford, England, 1962.

[26] Jun Gu. *Parallel Algorithms and Architectures for Very Fast AI Search*. PhD thesis, University of Utah, 1989.

[27] Shailesh U. Hedge, Jeffrey L. Sweet, and William B. Levy. Determination of parameters in a hopfield/tank computational network. In *Proceedings of the 1988 Connectionist Models Summer School*, pages 211–216, Morgan Kaufmann Publishers, 1988.

[28] W. D. Hillis. *The Connection Machine*. MIT Press, Cambridge, MA, 1985.

[29] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Acadamy of Sciences*, 79:2554–2558, April 1982.

[30] J. J. Hopfield and D. W. Tank. "neural" computation of decisions in optimization problems. *Biological Cybernetics*, 52:141–152, 1985.

[31] John J. Hopfield and David W. Tank. Computing with neural circuits: a model. *Science*, 233:625–633, 1985.

[32] Jenq-Neng Hwang. *Algorithms/Applications/Architectures of Artificial Neural Nets*. PhD thesis, University of Southern California, 1988.

[33] K. Hwang and F.A. Briggs. *Computer Architecture and Parallel Processing*. McGraw-Hill, 1984.

[34] Leah H. Jamieson, Dennis Gannon, and Robert J. Douglass, editors. *The Characteristics of Parallel Algorithms*. MIT Press, 1987.

[35] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[36] H.T. Kung and C.E. Leiserson. Systolic arrays (for vlsi). *Sparse Matrix Proceedings*, 256–282, 1978.

[37] R.A. Kupferer and D.J. Halski. Tactical flight management — survivable penetration. In *Proceedings of the 1984 IEEE National Aerospace and Electronics Conference*, May 1984.

[38] C. Lanczos. *The Variational Principles of Mechanics*. Dover Publications, Inc., New York, NY, fourth edition, 1970.

[39] Leon Lapidus and George F. Pinder. *Numerical Solution of Partial Differential Equations in Science and Engineering*. John Wiley & Sons, Inc., New York, NY, 1982.

[40] Juo-Jie Li. *Parallel Processing of Combinatorial Search Problems*. PhD thesis, Purdue University, 1985.

[41] Richard P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, 4–22, April 1987.

[42] Chia-Yiu Maa and Michael A. Shanblatt. Adjustment of parameters for signal decision networks. In *Proceedings of the 1989 International Joint Conference on Neural Networks*, 1989.

[43] Chia-Yiu Maa and Michael A. Shanblatt. Stability of linear programming neural network for problems with hypercube feasible region. In *Proceedings of the 1990 International Joint Conference on Neural Networks*, 1990.

[44] Henry Margenau and George Moseley Murphy. *The Mathematics of Physics and Chemistry*. D. Van Nostrand Company, Inc., Princeton, NJ, second edition, 1956.

[45] James Clerk Maxwell. *A Treatise On Electricity and Magnetism*. Volume 1, Dover Publications, Inc., New York, NY, 1954. An unabridged and unaltered republication of the third edition of 1891.

[46] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 9:127–147, 1943.

[47] Carver Mead and Lynn Conway. *Introduction to VLSI Systems*. Addison-Wesley Publishing Co., 1980.

[48] D. H. Menzel. *Mathematical Physics*. Dover Publications, Inc., New York, NY, 1961.

[49] A. R. Mitchell and D. F. Griffiths. *The Finite Difference Method in Partial Differential Equations*. John Wiley & Sons, Inc., New York, NY, 1980.

[50] N.J. Nilsson. *Principles of Artificial Intelligence*. Tioga, 1980.

[51] Allen Nussbaum. *ELECTROMAGNETIC THEORY for Engineers and Scientists*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1965.

[52] John O'Donnell. Supporting functional and logic programming languages through a data parallel vlsi architecture. In *VLSI for AI*, pages 49–60, Kluwer Academic Publishers, 1989.

[53] M.D. Olinger and M.W. Bird. *Tactical Flight Management Phase 1 Final Report*. Technical Report ID-04R-0683, Lear Siegler, Inc. Instrument Division, 4141 Eastern Avenue, S.E., Grand Rapids, MI, 49518, 1983.

[54] M.D. Olinger and M.W. Bird. *Tactical Flight Management Phase 2 Final Report*. Technical Report ID-03R-0484, Lear Siegler, Inc. Instrument Division, 4141 Eastern Avenue, S.E., Grand Rapids, MI, 49518, 1984.

[55] M.D. Olinger and M.W. Bird. Tactical flight management threat penetration design. In *Proceedings of the 1984 IEEE National Aerospace and Electronics Conference*, 1984.

[56] R. H. J. M. Otten and L. P. P. P. vanGinneken. *The Annealing Algorithm*. Kluwer Academic Publishers, 1989.

[57] Judea Pearl and Richard E. Korf. Search techniques. *Annual Review of Computer Science*, 2:451–467, 1987.

[58] Robert Plonsey and Robert E. Collin. *Principles and Applications of Electromagnetic Fields*. McGraw-Hill, 1961.

[59] J. Ramanujam and P. Sadayappan. Parameter identification for constrained optimization using neural networks. In *Proceedings of the 1988 Connectionist Models Summer School*, pages 154–161, Morgan Kaufmann Publishers, 1988.

[60] L. A. Reibling. *Research and Development in Neural Network Technology and Applications*. Annual Report GR-09P-0989, Smiths Industries Aerospace & Defense Systems, Inc., October 1989.

[61] L. A. Reibling. A neural network implementation of parallel search for multiple paths. In *Proceedings of the 1990 International Joint Conference on Neural Networks*, pages 675–678, January 1990.

[62] L. A. Reibling. Neural network solutions to mathematical models of parallel search for optimal trajectory generation. In *Proceedings of the 61st Symposium on Machine Intelligence for Aerospace Electronic Systems*, Advisory Group for Aerospace Research & Development, 1991.

[63] Elaine Rich. *Artificial Intelligence*. McGraw-Hill, 1983.

[64] Mario G. Salvadori and Melvin L. Baron. *Numerical Methods in Engineering*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1961.

[65] G. D. Smith. *Numerical Solution of Partial Differential Equations*. Oxford University Press, New York, NY, 1965.

[66] John Smith. *Modern Operational Circuit Design*. John Wiley, 1971.

[67] J. C. Smuts. *Holism and Evolution*. Macmillian, London, 1926.

[68] William R. Smythe. *Static and Dynamic Electricity*. McGraw-Hill, 1950.

[69] Thinking Machines Corporation. *Connection Machine Model CM-2 Technical Summary*. Thinking Machines Technical Report HA87-4, April 1987.

[70] P.J.M. VanLaarhoven and E.H.L. Aarts. *Simulated Annealing: Theory and Applications*. D. Reidel Publishing Co., Dodrecht, Holland, 1987.

[71] Ernst Weber. *Electromagnetic Fields. Volume I — Mapping of Fields*, John Wiley & Sons, Inc., New York, NY, 1950.

[72] Patrick H. Winston. *Artificial Intelligence*. Addison-Wesley Publishing Co., 2nd edition, 1984.