A BINARY RELATION INFERENCE NETWORK FOR CONSTRAINED OPTIMIZATION

By

Crystal Jinghua Su

B. A. Sc. and M. A. Sc., Tsinghua University, China, 1985 and 1988

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES

DEPARTMENT OF ELECTRICAL ENGINEERING

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

Sept. 1992

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

(Signature) _____

Department of Electrical Engineering

The University of British Columbia
Vancouver, Canada

Date Dec 11, 1992

Abstract

Constrained optimization is an essential problem in artificial intelligence, operations research, robotics and very large scale integration hardware design, etc. Many constrained optimization problems are computation-intensive, and yet require fast solutions. This thesis presents a novel parallel computing network designed for some optimization problems which can be represented by binary relations and the calculation upon them.

Traditional graph search techniques provide sequential solutions to constraint satisfaction, but their speed is limited by the computational ability of a single processor. Parallel processing offers faster solutions by dividing its work load among many processors. A parallel algorithm can be developed from its sequential counterpart by decomposing the unfinished task dynamically according to the availability of processors. These kind of algorithms usually have difficulty in distributing work load evenly among processors [69]. Many distributed parallel computing models have been proposed for various types of problems. A distributed computation model usually embodies the data structure for a specific type of problem [16]. Many of them work in the discrete time domain [71, etc]. Neural network approaches have been proposed for optimization problems [28], however, optimal solutions are often not guaranteed when the gradient descent scheme is used. The determination of the link weights in a Hopfield-type neural network can be elaborate and convergence for large problems remains a problem [2, 90].

This thesis proposes a new parallel computing network — a binary relation inference network. The network topology is set up to match conventional optimization procedures. The network transforms some optimization problems (shortest path, transitive closure, assignment, etc.) into network convergence problems and divides the work load

evenly among all processors. Link weights are known. It can operate in synchronous discrete-time, asynchronous discrete-time, and continuous-time domains. Inference network algorithms are derived directly from problem formulation, and are effective for a large range of problem sizes.

The inference network consists of interconnected components; each component is composed of a unit, a number of sites attached to the unit, and links from the unit to sites on other units. Each unit is a simple computational engine. The topology of the inference network matches naturally to many optimization problems, since its deduction sites produce candidates to compete for the optimum and its decision-making units select the optimum from the site values. Either directly or through a transformation mapping to a systolic structure, discrete inference network algorithms can be implemented on a commercially available parallel processing facility, such as the Connection Machine. Analog inference network can be built using integrated circuits which solve problems efficiently in the continuous-time domain.

In this thesis, mathematical analysis and numerical simulation were conducted for the synchronous, asynchronous, and analog inference network. Various applications has been discussed. The inference network has shown to solve the all-pair shortest path problem efficiently in various updating schemes. The inference network algorithm for the transitive closure problem was derived straightforwardly from the formulation of the problem and the topology of the inference network. Efficient continuous-time solution was obtained from a non-synchronized logical circuit. The convergence of the network for the shortest path or transitive closure problems was shown to be independent of the problem size. It was demonstrated that the inference network can solve the assignment problem in a way similar to the Hopfield net's solution to the traveling salesman problem. However, the former was shown to converge for all test cases with problem size up to 128.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I am grateful to my supervisor Dr. K.P. Lam for giving me inspiration, providing effective direction, allowing me to explore the areas I am interested in, and offering advice even when he is working outside of UBC.

I would like to thank Dr. H. Alnuweiri sincerely for taking over the duties as my research supervisor at a time of need. My thanks also go to my supervisory committee members Dr. P.D. Lawrence and Dr. R. Ward for spending time with me despite of their busy schedules. These people help greatly in this thesis presentation.

I appreciate the Institute of Advanced Computer Studies at University of Maryland and the Thinking Machine Corporation Network Service for granting the access to their Connection Machines.

Chapter 1

Introduction

1.1 Constrained Optimization

A *Constrained Optimization Problem* [56] has four major components: V, D, C and f, where V is a set of variables $\{v_1, v_2, \cdots, v_n\}$, and $|V| = n$ (the cardinality of V is n); $D = D_1 \times D_2 \times D_3 \cdots \times D_n$ is the vector of domains for all the variables in V, and D_i is the domain of v_i — a set of values from which v_i can take one; C is the set of constraints for subsets (of arbitrary sizes) of variables in V, and a constraint on a subset of variables is a restriction on the values that they can take simultaneously; f is an objective function to be optimized. Constraints can also be an integral part of the objective function, where the function measures the compatibility between the constraints and the solution.

A *valid* solution for constrained optimization is an assignment of one value to each of the variables in V such that all the constraints are satisfied. An *optimal* solution is a valid solution which optimizes the objective function. The goal of constrained optimization is, therefore, to find the optimal solution. However, when there is no efficient way to get the optimal solution (exact solution), a good valid solution (approximate solution) is also acceptable.

Many practical problems can be interpreted as constrained optimization problems. The assignment problem, to be discussed in Chapter 6, is a typical example. Given N persons, N jobs, and the benefit of each person getting each job, an one-to-one person-job assignment is to be found to maximize the total benefit. Here we may take persons as

variables and jobs as their domains. The constraint is each person getting a distinct job, and the objective function calculates the total benefit.

The all-pair shortest path problem (Chapter 4) can also be formulated into a constrained optimization problem, although not as obvious as the assignment problem. Given a map of N cities and road distances between some of them, how to find the shortest path between two cities? The variables in this problem are city pairs, and their domains are all the combinations of possible intermediate cities. The objective function is the total distance between these city pairs. An implicit constraint is that if the shortest path from city one to city three passes city two, then this path must be the concatenation of the shortest paths from city one to two and from city two to three [11]. The transitive closure problem is a path-finding problem related with the shortest path problem, and therefore can also be considered as an optimization problem (Chapter 5).

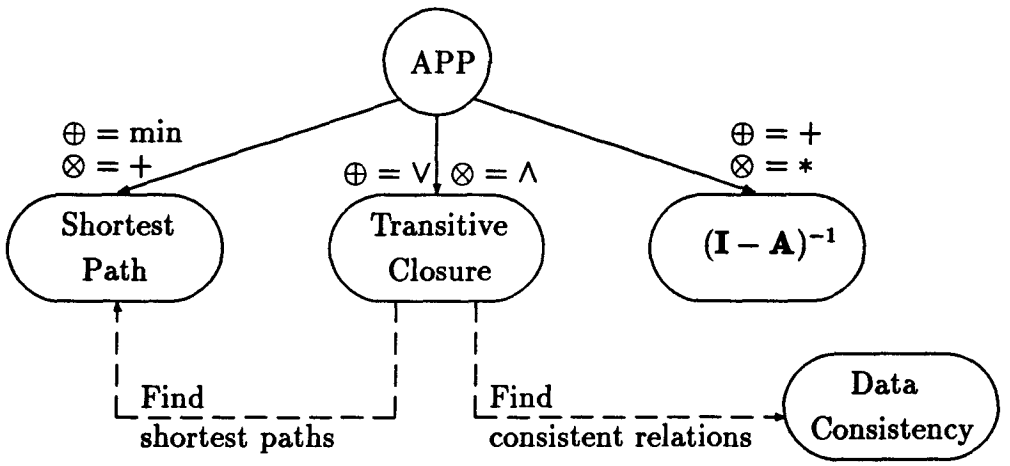


Figure 1.1: Relationship between some optimization problems.

Figure 1.1 outlines some interesting relationships between the problems of shortest path (page 41), transitive closure (page 62), data consistency checking ,(page 119) and

matrix inversion. First, the Algebraic Path Problem (abbreviated to APP), is a generalization of several conventional problems [1, 57, 58, 71, 95]. Substituting the general operators $\oplus$ and $\otimes$ with specific ones, an algorithm for the APP [71, 70] can give solutions to the shortest path problem, the transitive closure problem, and computing $(\mathbf{I} - \mathbf{A})^{-1}$. Second, transitive closure, shortest path, and data consistency problems are related. The transitive closure problem determines whether a binary relation is defined for two nodes in a graph. The data consistency problem examines the consistency of a given set of time/location referencing data, and subsequently derives a consistent set of data based on the given data. The shortest path and data consistency problems, in addition to the determination of the existence of a binary relation, find a specific — the shortest or consistent — relation from a number of given or derived relations concerning the two given nodes.

1.2 Traditional Approaches for Constrained Optimization

The procedure of finding a valid solution to a constrained optimization problem is usually called *constraint satisfaction* [56]. Symbolic Artificial Intelligence (AI) procedures for constraint satisfaction can be used in constrained optimization: generate all valid solutions using constraint satisfaction techniques, evaluate the objective function for each solution, and find the optimal. *Generate and test* [56] is the most straightforward approach for constraint satisfaction. It simply generates *all* possible combinations of the variable assignments and tests their compatibility with the constraints. *Generate and Test* is very inefficient and valid only when the variable domains are discrete and finite.

A predicate is a function that maps an argument into a truth value (TRUE or FALSE). *Backtracking* [56] is a basic graph search algorithm which systematically explores variable domains by sequentially evaluating predicates in a specific order. As soon as any

predicate has all its variables evaluated, its truth value is determined. Whenever a predicate is evaluated as false, the backtracking procedure moves back to the last variable with unassigned values remaining in its domain (if any) and evaluate its next value. When the size of a problem is large, backtracking is inefficient due to the combinatorial explosion of computations [67]. Searching with the help of *heuristic information*, for example A^* search [64], may significantly reduce the computational cost. However, good heuristic information is often unavailable. Mackworth [55] and Waltz [87] proposed to systematically reduce the search space by eliminating local inconsistency. Although these algorithms can often significantly reduce the sizes of the variable domains, they usually have to be combined with backtracking to get final solutions (refer [55] and [87]).

Simulated annealing [35] has shown significant potential for solving large-scale combinatorial optimization problems. It is an iterative improvement procedure using controlled uphill step to achieve a global minimum. This procedure was used in the traveling salesman problem [35, 68]. *Linear programming* [68] is a well-studied area for optimization problems and has many established algorithms to perform relaxation [9]. *Probability relaxation* is a labeling procedure used in artificial intelligence and especially in computer vision [4, 59]. When a variable domain is finite and discrete, each variable is assigned a label which is associated with a weight or probability. Relaxation procedure moves the label vector around in order to satisfy all the constraints, as well as optimize the objective function [4]. *Dynamic programming* [11] provides a general approach for optimization problems that can be broken down into smaller but similar problems. Dreyfus [11] explores the mathematical formulations for problems like equipment replacement, resource allocation, shortest path, and traveling salesman, etc.

All these approaches have limitations. The major difficulties with symbolic AI constraint satisfaction are its sequential nature and the combinatorial explosion. Symbolic

AI algorithms usually can not be parallelized straightforwardly without serious communication overhead (refer [69]). When the size of a problem is large, it is impossible to explore and evaluate all the valid solutions. Simulated annealing is a sequential procedure and does not guarantee the solution to be optimal [68]. Linear programming and dynamic programming are not general methods for all optimization problems; their efficiency depends critically on the recurrent formulation of the problem. Some of the algorithms can be parallelized; others are computational-intensive and have a sequential nature (refer [11]).

1.3 Parallel Approaches

The design of efficient techniques for parallel computers is strongly influenced by the assumption of the parallel computation model used. There are different classifications of parallel machines; one of them is to divide parallel machines into shared-memory models [75] and distributed-memory models [75]. The main difference between the two models is the way in which communication among processors is handled.

In a shared memory system, processors write and read data to and from a single shared memory. The memory serves both as a communication medium as well as a storage medium. The PRAM (Parallel Random Access Machine) model assumes that all processors can perform a single access to the shared memory in $O(1)$ time, regardless of the data access pattern [18]. Within a single memory access period, all processors can perform either a read operation or a write operation but not both.

A parallel algorithm can be developed from its sequential counterpart by decomposing its computation dynamically to multiple processors. Quinn [69] proposed synchronous and asynchronous models to perform backtracking (page 3). In the synchronous model, all unexamined subproblems are kept in a single priority queue stored in the memory

of a master processor; whereas in the asynchronous model, each processor keeps its own priority queue. Quinn's results show that the synchronous model predicts an eventual decrease in speedup as the number of processors increases, due to the increased communication overhead. Quinn's asynchronous model has better performance. However, when the number of processors increases, theoretical speedup saturates and experimental speedup decreases [69].

In a distributed-memory model, each processor has a local memory. Data exchange is accomplished through inter-processor connections. Communication among processors is restricted by the network topology, bandwidth, data movement pattern, and other related factors [50]. To obtain good performance, small diameter (the maximum number of times a message has to be forwarded from one processor to its destination), uniform communication, extensibility, short wires, and redundant paths are recommended [26]. On the other hand, in order to reduce cost, minimum number of wires, efficient layout, fixed degree, and fitting to available technology are necessary [26].

In the attempts to solve problems requiring a large amount of communication, various interconnection topologies have been proposed [6]. Some examples are crossbar, ring, tree, grid, torus and hypercube. Reconfigurable networks are used to achieve higher network throughput. An example is given in [3].

Connectionist networks, neural networks and systolic arrays are some intensively-studied distributed-memory networks. Shastri [72] showed that structured connectionist knowledge representation can handle problems that have proven difficult for logic-based approaches. The connectionist approach uses distributed representation which reflects the way a human brain works [14]. Constrained optimization problems can be solved by using such a connectionist network, in which constraints are represented by links and their weights [16]. Unit outputs $\mathbf{v}$ must be given proper interpretations so that a possible stable state of the network corresponds to a meaningful state of the problem (refer [28]).

The presence and weight of the links, denoted by $\mathbf{T}$, are pre-defined according to the known convergence requirements such that the network will arrive at a desired state eventually. Computational energy of the network $E(\mathbf{T}, \mathbf{v})$ can be defined indicating how much the state of the network disagrees with the given constraints (refer [28]). Optimization procedures move the outputs $\mathbf{v}$ in the space of possible states representing solutions, progressing in the direction that tends to decrease the energy. When used for polynomial-complexity problems, a connectionist network can be a vehicle to parallelize an otherwise sequential algorithm.

The Hopfield network was used in solving the traditional traveling salesman problem [31, 2]. The objective function was composed of four terms; three of them represent the constraints and one represents the length of the tour to be minimized. The result is reasonably good for small size problems and its speed is acceptable. However, the validity and quality of the solution depend critically on the weight of each term in the objective function [90]. It is very hard to determine these weights for this problem with a large number of cities [90]. Combinatorial optimization can be done in an analog neural network with modified Hopfield's energy function and parameters [36]. The idea of transforming a constrained optimization problem into a network convergence problem was also used to solve the map coloring and Necker cube problems [15, 16].

Winner-Take-All (WTA) is a decision-making mechanism used in distributed computing situations. A WTA network is comprised of neurons connected by mutually inhibitory links [14]. Neurons in a WTA network compute their values based on such a rule: the neuron with the largest value keeps its value and the values of the rest of the neurons are set to zero. A maximum neural network is composed of a number of interconnected clusters where each cluster is a winner-take-all network. Maximum neural networks have been used for the bipartite subgraph problem [49] and the max cut problem [48]. It has been proven [48, 49] that the maximum neural network always generates valid (optimal

or near-optimal) solutions for these two problems and the link weights of the network are easy to determine. When the maximum neural network is used for the max cut problem, the solution quality improves as time elapses [48]. Unlike the energy function used in the Hopfield model, which is comprised of four weighted terms, the energy functions for these two problems only contain one term which covers the constraints as well as the quantity to be optimized. Since there is only one coefficient involved in the energy function, link weights can be easily determined by letting the energy decrease monotonically. Therefore, the two problems solved by Lee [48, 49] provide examples of a neural network with easy-to-determine link weights. The proposed inference network will be compared with some neural network models in Section 2.6 when the inference network topology is defined.

A systolic array is another approach to deal with some optimization problems. Several systolic architectures and algorithms were proposed for algebraic path problems (including shortest path, matrix inversion, and transitive closure) [71, 70, 12, 39, 23]. The definitions for the shortest path and transitive closure problems can be found on page 41 and page 62 respectively. A systolic architecture is a number of systematically connected processors; each processor calculates and transfers data rhythmically in a pre-defined way. A systolic array is a discrete-time system; processing elements in the array must be properly synchronized to ensure a correct solution. Unlike many connectionist and neural network approaches, systolic algorithms do not involve relaxation.

In distributed-memory models, a network topology can embody specific computational structures needed to solve a certain class of problems [16]. A systolic structure for a given problem is usually derived by analyzing the data dependencies of the corresponding sequential algorithm for the problem (refer [52]). Grid and torus connection are most commonly used in parallel computers because of their simple hardware implementation. However, parallel algorithms written for these architectures often have to be carefully

designed in terms of data delivery and synchronization. The popular hypercube topology has a higher space-compression factor and a lower communication factor than what is naturally needed in optimization problems (refer [43] or Section 2.7). For optimization problems, this thesis defines a parallel computing model in which both the evaluation of the candidates competing for the optimum and the selection of the optimum are explicitly represented by the values of network components. It is a distributed-memory model when it is used in synchronous discrete time domain.

1.4 Scope of this Thesis

This thesis presents a novel parallel computing network — the binary relation inference network which is suitable for some constraint optimization problems, such as the all-pair shortest path problem, the transitive closure problem, and the assignment problem. Constrained optimization is an essential problem in artificial intelligence, operations research, robotics, and very large scale integration hardware design, etc. Many constrained optimization problems are computation-intensive, and yet require fast solutions.

The approach proposed here is based on defining and using an inference network whose topology is derived from simple intuitive approach to optimization. It will be shown that the inference network algorithms are obtained from the formulation of the problems, and they can be executed in synchronous discrete-time, asynchronous discrete-time, or continuous-time domains. It will be shown that the link weights of the inference network are easy to determine. The network will be shown to produce optimal solutions to the shortest path (Chapter 4) and the transitive closure (Chapter 5) problems. It does not guarantee an optimal solution to the assignment problem (Chapter 6) because gradient descent will be used to decrease the network energy. The trade-off of eliminating synchronization is the large amount of communication between the elements in

the network. With these communication channels, each decision-maker (a unit in the network) always has access to all candidates competing for the optimum. Each element of the network is a very simple computational engine.

The thesis has the following contents. Chapter 1 is an introduction. Chapter 2 discusses the topology and operation of the inference network. It also shows several structural versions and updating mechanisms for different instances of optimization problems. Chapter 3 is about implementation issues. It discusses the components of the network and their interconnection. Both analog and discrete implementations are studied. A transformational mapping of the discrete-time inference network onto the Connection Machine is also introduced. The inference network algorithms for some constrained optimization problems are discussed in Chapter 4 to Chapter 6, and results are compared with other established methods. Chapter 4 gives the synchronous, asynchronous, and continuous-time algorithms for the shortest path problem and discusses their convergence. The network can also unify several other algorithms for the problem in terms of different inference network operations. The efficiencies of various algorithms are also discussed. Chapter 5 gives the inference network algorithm for the transitive closure problem and the continuous-time implementation of the algorithm using logic gates. Chapter 6 derives algorithms for the assignment problem and discusses network dynamics in pursuing approximate solutions. A decreasing weight is used in the energy function for the quantity to be optimized, which results in good network convergence. Finally Chapter 7 summarizes the contributions of this thesis and discusses future work.

Chapter 2

Topology of the Inference Network

2.1 Inference on Binary Relations

A node is an item. It can be a node in a graph, a city in a map, or a time instance in a schedule, etc. A relation associates two or more nodes. A *binary relation* $R(x, y)$ is either a qualitative or quantitative relation between two nodes x and y. A binary relation is more basic than a general m-ary relation $R(x_1, x_2, ..., x_m)$ for m distinct nodes, which can often be defined by a set of binary relations about the m nodes, together with a set of inference rules on the binary relations. Examples of inference rules are *Unary* and *binary* deductions which draw a conclusion from one or two premises respectively.

The relation '$x > y$' is a binary relation. The relation 'u *is the largest among* u, v, w' is a 3-ary relation which can be derived from binary relations '$u > v$' and '$u > w$'. Unary deduction obtains $y > x$ from $x < y$, whereas binary deduction obtains $x > z$ from $x > y$ and $y > z$. A relation between x and z may also be derived using another intermediate node u, for example $x > u$ and $u > z$ where $u \neq y$. The final inference result of the relation between x and z is determined based on all possible intermediate nodes.

Consider a problem with N nodes denoted as $1, 2, \ldots, N$ where the relations between the nodes can be fully represented using binary relations $R(i, j)$ ($1 \leq i, j \leq N$ are nodes). Operations on the relations follow the format: $R(i, j)$ and $R(j, k) \Rightarrow R(i, k)$ (deducing relation $R(i, k)$ from relations $R(i, j)$ and $R(j, k)$). There are other ways to deduce $R(i, k)$: from $R(i, m)$ and $R(m, k)$, $R(i, n)$ and $R(n, k)$, etc. Inference takes place

to determine relation $R(i,k)$ based on various deduction results.

2.2 A Novel Inference Network

The inference network is a novel topology for solving constrained optimization problems. To the best of the author's knowledge, no research has been reported on networks with the same architecture and operations.

The inference network architecture aims at representing optimization procedures explicitly. Its components are interconnected in such a way that a decision maker always has access to the candidates competing for the optimum.

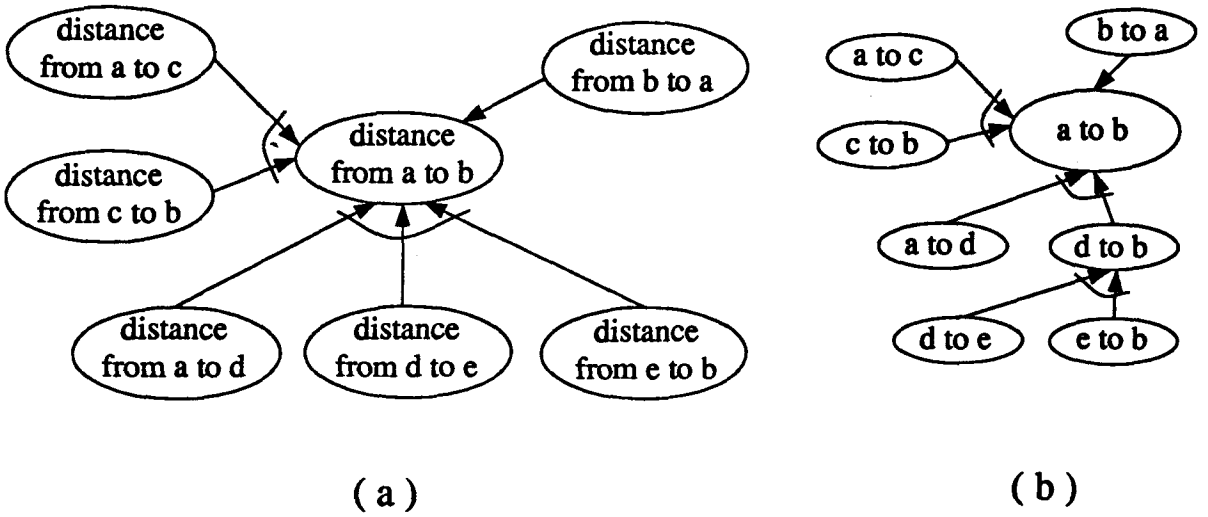


Figure 2.2: (a) An AND/OR graph indicating various ways to find the distance from *a* to *b*. (b) Substitute the 3-input AND relation in (a) with two 2-input AND relations.

The basic inspiration of the inference network topology came from a conventional AND/OR graph [64]. An AND/OR graph is a notation used in artificial intelligence to represent logical relations. As shown in Figure 2.2, arrows are tied up by arcs into groups. The logic between the relations tied up by an arc is AND — all the tied relations

are needed to derive a result. The logic between any two groups of relations is OR — each group represents an independent way to derive the result. Figure 2.2(a) shows various ways to calculate the distance from a to b. The distance may be obtained from the distance from b to a alone, OR, from the distances $a \rightarrow c$ AND $c \rightarrow b$, OR, from the distances $a \rightarrow d$ AND $d \rightarrow e$ AND $e \rightarrow b$. Since m-ary deduction can often be decomposed into a number of binary deductions, an AND/OR graph with only 2-input AND relations is often sufficient to represent the logic relation among a set of nodes. Fig. 2.2(b) is a variation of Fig. 2.2(a) and only uses 2-input AND relations.

The AND/OR graph is a scheme to describe logical relations. It is not a parallel computing network; however, it shows how to derive a conclusion from given data, and how the given data are to be associated. Based on an AND/OR graph, the inference network was proposed which is composed of units, sites and links. The correspondence and descriptions of the entries in the inference network and an AND/OR graph are given in Figure 2.3.

Figure 2.3 contains all the binary deductions which can be done to the three binary relations. In this figure, any distance can be calculated from the other two distances. On the other hand, any distance is used in evaluating another distance. Given a set of N nodes, there will be up to $N(N-1)$ binary relations between the nodes. Generally, a result $R(i,j)$ can be deduced from any two relations $R(i,l)$ and $R(l,j)$. If all the possible binary deductions for an N-node problem are shown in a graph, the graph will become a complicated, regularly connected, closed network with a large number of units. How can one ensure that such a closed network will not oscillate? The convergence of a parallel computing network can be analyzed by defining and examining the computational energy of the network. The Hopfield model has shown an example of using an energy function in convergence analysis.

The topological distinction between the inference network and neural networks or

other parallel computing models is that the inputs to an inference network unit are grouped into pairs (sites), and the evaluation of each pair (corresponding to site calculation) is independent of the evaluation of other pairs. Figure 2.5 shows a unit with its paired inputs, as well as a typical neuron with all inputs connecting to it directly. A typical neuron collects all of its inputs and produces an output accordingly. An inference network unit is able to update its value based on the change of any of its input pair. The reason to group the inputs is that the result obtained from one relation pair (for example $a \rightarrow c \rightarrow b$) can be used to update the relation ($a \rightarrow b$) without waiting for results from others ($a \rightarrow d \rightarrow e \rightarrow b$ or $b \rightarrow a$).

Another difference between the inference network and some other parallel computing models (for example a systolic array) is that the former can be updated synchronously, asynchronously, or in continuous-time. This is a feature related with the structure of the inference network. To be more specific, the grouping of unit inputs into pairs makes it possible to compute candidates for optimum independently. An AND/OR graph shows various paths to evaluate a relation using other relations, and the evaluations through distinct paths are independent. Again let us look at Figure 2.2, the calculation of distance $a \rightarrow c \rightarrow b$ is independent of the calculation of $a \rightarrow d \rightarrow b$. This indicates that the evaluation of sites can be parallelized. For some applications, the evaluation of all relations can be active at the same time. For example, distance $d \rightarrow b$ and $a \rightarrow b$ can be calculated simultaneously. When the distance $a \rightarrow d \rightarrow e \rightarrow b$ is not known, distance $a \rightarrow b$ is determined by $a \rightarrow c \rightarrow b$ and $b \rightarrow a$. After $a \rightarrow d \rightarrow e \rightarrow b$ is obtained, distance $a \rightarrow b$ can be updated (if necessary) based on the new information. This analysis shows that for some applications, both the units and the sites in the inference network can be updated synchronously or asynchronously (Section 2.4).

2.3 Inference Network Topology

The binary relation inference network consists of simple computational elements and operates by communicating between the elements. Each computational element is composed of a unit, a number of sites attached to the unit, and links from the unit to a site at another unit, refer Figure 2.4. Unit and site functions are usually obtained from application considerations. Just as described in the previous section, a site output derives a result from two relations, and sites attached to a certain unit provide different ways to derive a relation. A unit determines a binary relation based on its site values. In general, site functions represent deduction rules or calculate factors competing for the optimum; and unit functions resolve conflicts or perform optimization. The dynamic state of the network is determined by its link weights, unit and site functions, initial state, and updating mechanism. The network is named *inference network* because conclusions are obtained by binary deduction and multi-input conflict resolution.

We will use u_{ij} for the value of unit (i,j) and s_{ijl} for the value of site l at unit (i,j). Figure 2.4 shows unit (i,j) in the inference network. Relation u_{i1} and u_{1j} arrive at site 1 of unit (i,j) and produce a result s_{ij1} — relation between i and j according to site 1. At the same time, other results about i and j — s_{ij2}, $\cdots$, s_{ijN} — can also be derived. Unit (i,j) is responsible to derive a relation between i and j based on the site values s_{ij1}, s_{ij2}, $\cdots$, s_{ijN}. The key features of this topology which make it suitable for optimization problems are its deduction ability at each site and the comparing ability at each unit. A 2-input site performs binary deduction or calculates a candidate competing for the optimum, for example, it calculates the distance from i to j via a specific intermediate node. A multi-site unit provides the platform for multiple-input decision-making or optimization, for example, finds the shortest path from i to j.

For an N-node problem, the general inference network contains N^2 units. For all

$1 \leq i,j \leq N$, unit (i,j) has N sites, namely site l ($1 \leq l \leq N$). Unit (i,j) has $2N$ outgoing links leading to site j at units (i,l) and site i at units (l,j), where $1 \leq l \leq N$. Site l at unit (i,j) has two incoming links from unit (i,l) and unit (l,j). A unit is a simple computational engine which is able to calculate its value based on its N site values using a simple unit function. The ability required for a site is to evaluate a simple site function of two variables — its two inputs. Like a neuron in an artificial neural network, a unit does not require any sophisticated processor, nor does it require a large memory.

For many applications, diagonal units (i,i) can be omitted, hence the total number of units is reduced to $N(N-1)$, the number of sites at each unit reduced to $(N-2)$, and the number of outgoing/incoming links from/to each unit reduced to $2(N-2)$. The site label l at unit (i,j) satisfies $1 \leq l \leq N$ and $l \neq i \neq j$. For applications with undirected graphs, $d_{ij} = d_{ji}$ (symmetric distance matrix), the network size can be cut down by half: only $C(N,2)$ units in the network[1]. Each unit in this case still has $(N-2)$ sites and each site has two incoming links; *unit (i,j)* and *unit (j,i)* will be used to refer to the same unit in the network.

For clarity, Figure 2.4 only shows the sites and links for unit (i,j). However, other units also have N sites and each site also has two incoming links. If all the sites and links at each unit are put into Figure 2.4, the complete inference network will become a closed network with feedback. If the N^2 units of the network are placed into a 2-dimensional array on a plane, and unit (i,j) stays at row i and column j, then only the units in the same row or column communicate among themselves. There is no direct connection between unit (i,j) and unit (v,w) if i,j,v,w are four distinct nodes in the problem.

[1] $C(N,2) = N * (N-1)/2$ is the number of different 2-object combinations from N objects without considering the order.

2.4 Updating Schemes

The inference network works in both discrete and continuous-time domains. The discrete inference network is one which operates rhythmically with a clock. It can work in synchronous, asynchronous, or partially synchronous modes. In synchronous updating, all unit values are updated at the same time based on the unit values at the previous step. Asynchronous updating changes one unit at a time. The new value of the unit is then used in the updating of the next unit. Partially synchronous mode combines these two schemes — updating some of the units synchronously at a time. The number of iterations required to complete a task shows the speed of a discrete algorithm. An iteration is used to stand for the process that can be performed on the inference network, given the results of the previous iterations. Each iteration comprises the simultaneous binary deduction (evaluation of sites) at all units and their subsequent conflict resolution (evaluation of units).

In the analog inference network, all units constantly change their values until a stable state is reached. Updated unit values are used by other units right away. The time required for the network to arrive at the stable state indicates the speed of an algorithm. A unit value can be calculated after all its sites are evaluated in parallel. It is also possible to update the unit value as soon as one or more sites are evaluated. The reason behind this is that optimization can take place whenever a candidate is generated or updated; the unit value will always be the optimum-so-far. For analog implementation, it is not difficult to update a unit as soon as its site changes (refer Figure. 3.7).

2.5 Transfer Function

A site function $S()$ is a function of two unit values:

$$s_{ijl} = S(u_{il}, u_{lj}) \tag{2.1}$$

A discrete unit function $U()$ is a function of its previous unit value and all its site values:

$$u_{ij}^{(k+1)} = U(u_{ij}^{(k)}, s_{ij1}^{(k+1)}, s_{ij2}^{(k+1)}, \ldots, s_{ijN}^{(k+1)}) \tag{2.2}$$

A continuous-time unit function $U()$ is a function of its own value and all its site values:

$$u_{ij}(t+\delta) = U(u_{ij}(t), s_{ij1}(t+\delta), s_{ij2}(t+\delta), \ldots, s_{ijN}(t+\delta)) \tag{2.3}$$

For an optimization problem, the candidates competing for the optimum are usually independent of each other, which means the unit function is separable:

$$u_{ij} = U'(\ \ldots U'(\ U'(l_0, l_1),\ U'(l_2, l_3)\ )\ldots) \tag{2.4}$$

where $l_0, l_1, l_2, \ldots, l_N$ is an arbitrary permutation of s_{ij1}, s_{ij2}, ..., s_{ijN}, and u_{ij}. $U'()$ is a 2-variable function related with $U()$. For example, if $U()$ is a minimization of $N+1$ variables, then $U'()$ is a minimization of two variables. For many unit function $U()$, the $N+1$ variables in $U()$ can be associated randomly and minimized in an arbitrary order.

The state of the inference network can be described by all its unit values, which can be put into an N^2-dimensional vector

$$\mathbf{u} = (u_{11}, u_{12}, \ldots, u_{1N}, u_{21}, u_{22}, \ldots, u_{NN})^t$$

Updating of the network can be characterized by a transfer matrix $\mathbf{P}$ if:

- the network updates in synchronous discrete mode or continuous-time domain;
- unit function $U()$ and site function $S()$ are linear and are the same for all the unit and sites;
- unit function is separable.

For synchronous discrete updating, the state of the network is determined by equation

$$\mathbf{u}^{(k+1)} = \mathbf{P}\mathbf{u}^{(k)} \tag{2.5}$$

For the analog network, network state can be determined by equation

$$\frac{d\mathbf{u}}{dt} = \mathbf{P}\mathbf{u} \tag{2.6}$$

where the transfer matrix $\mathbf{P}$ is an $N^2 \times N^2$ sparse matrix whose element at row $(iN+j)$ and column $(kN+l)$ is $p_{iN+j,kN+l}$. $p_{iN+j,kN+l}$ shows whether and how a binary relation concerning nodes i and j is related with another binary relation concerning nodes k and l.

$$p_{iN+j,kN+l} = a\delta(i-k)\delta(j-l) + b\delta(i-k) + c\delta(j-l) \tag{2.7}$$

where a, b, c are constants determined from applications and

$$\delta(x) = \begin{cases} 1 & x = 0 \\ 0 & x \neq 0 \end{cases}$$

The first term of $p_{iN+j,kN+l}$ shows that a unit has feedback from itself; the second and third terms of $p_{iN+j,kN+l}$ indicate that a unit interacts with the units in the same row $(i = k)$ or the same column $(j = l)$. When $i \neq k$ and $j \neq l$, $p_{iN+j,kN+l} = 0$, which means a unit does not communicate directly with units which are not in the same row nor in the same column with it.

For the assignment problem, to be discussed in Chapter 6, a transfer matrix $\mathbf{P}$ is defined whose elements satisfy Eq.(2.7). Both Eq.(2.5) and Eq.(2.6) hold for the application. The transfer matrix is used to examine network convergence for the problem (Chapter 6). For the shortest path and transitive closure problems, to be discussed in Chapter 4 and Chapter 5 respectively, a transfer matrix can also be defined indicating link weights. For these two applications, a, b and c in Eq.(2.7) are all one. However, since these two applications use non-linear site and/or unit functions, the updating of network state cannot be described by linear equations Eq.(2.5) or Eq.(2.6).

When the inference network is used to solve an optimization problem and is stabilized at a solution, the unit value for each unit (i,j) is denoted as u_{ij}^{exp} $\forall i,j$. Although u_{ij}^{exp} are unknown before the problem is solved, they can sometimes be expressed in a mathematical expression. For example, in the shortest path problem, $u_{ij}^{exp} = \min\{u_{ij}, \min_{\forall l}\{(u_{il} + u_{lj})\}\}$. When the expression for u_{ij}^{exp} is known as a function of unit values, the energy function of the network can have the following form:

$$E = \sum_i \sum_j (u_{ij} - u_{ij}^{exp})^2 \tag{2.8}$$

where u_{ij} is the current value of unit (i,j), and u_{ij}^{exp} is an expression of the expected value of unit (i,j) when the network is stabilized. The energy function indicates the distance of the current network state from its stable state. When the network arrives at the expected solution, $u_{ij} = u_{ij}^{exp}$ $\forall i,j$ and $E = 0$. When u_{ij} changes in continuous time domain, and $\lim_{t \to \infty} u_{ij} = u_{ij}^{exp}$, then energy E will be sufficiently close to zero when time t is sufficiently long. Plugging the expression for u_{ij}^{exp} into the energy function, the derivative of the energy can be used to analyze the convergence of the network.

In some applications, it is impossible to know u_{ij}^{exp} before solving the problem. However, the constraint of the problem can be put into a mathematical term E_c which equals to zero when the constraint is satisfied and is greater than zero otherwise. Meanwhile, the optimization problem is to minimize an objective function E_o under the constraint. Under this circumstance, the energy function can be defined as $E = A_c E_c + A_o E_o$ where A_c and A_o are constants. The objective is to decrease the energy until arriving at a state at which $E_c = 0$ and E_o is minimized. The Hopfield model is an example. Giving each term proper weight is critical here if gradient descent is used to decrease the energy. For some graph partition problems, Lee shows [48, 49] that both the problem constraint and the objective function can be expressed in a single term E_{co}, therefore the energy function $E = AE_{co}$ contains only one coefficient, and there is no need to adjust the weights

between the constraint and the objective function.

When linear unit and site functions are used in synchronous or continuous-time updating, the energy function has the form of

$$E = \mathbf{u}^t\mathbf{P}\mathbf{u} + \mathbf{Q}^t\mathbf{u} + \mathbf{R} \tag{2.9}$$

where $\mathbf{u}$ is a vector of all units, $\mathbf{P}$ is the above transfer matrix, $\mathbf{Q}$ and $\mathbf{R}$ are constant vectors. Refer Chapter 6 for the derivation of Eq.(2.9). Although units in the network are connected with *feedback*, oscillation will not occur in the electronically implemented network, as long as the energy function decreases monotonically, that is, in the mathematical term, $\frac{dE}{dt} \leq 0$.

2.6 Is the Inference Network a Neural Network?

An artificial neural network is an abstract computational model based on the structure of animal brains. Individual neurons do not transmit large amounts of symbolic information. Instead, they compute by being appropriately connected to a large number of similar neurons [14]. A number of neurons form a state and represent a concept (distributed representation, refer the Necker Cube example in [16]). The operation of each neuron is very simple. A neural network often has a uniform structure. The change of the state of a neuron aims at reducing network energy, and is often in the direction of gradient descent (for example, the Hopfield model for the traveling salesman problem). Neural network approaches focus on parallelism, robustness, adaptation, and learning. Neural networks usually solve an optimization problem right from its definition (define an energy function which is composed of constraints and the quantity to optimize). Algorithms for some types of neural networks involve a lot of learning and training to determine link weights [27]. The objective of network updating is to decrease its energy, and link weights

are set up to achieve this goal. Artificial neural networks have both digital and analog versions.

The inference network is a parallel computation model. When it is used synchronously, it is a distributed memory model. When it is used in continuous-time (without a clock), it is a circuit without the kind of memory in a von Neumann machine. Some electronic components in the circuit (capacitance, etc) act like memories to keep voltage values etc. Its topology is a reflection of an optimization process — evaluating candidates and determining the optimum. Each processor has a specific mission — an explicit reason why it performs the operation. Unit and site functions are defined to be essential procedures in optimization. Proper definition of the unit and site functions results in a decreasing energy function. The network has a uniform structure and units are identical. The network does not involve training and learning. All link weights are known. Algorithms are developed right from problem formulation. The network can operate in both discrete and continuous-time domains.

Based on the above information, the inference network can be regarded as a neural network with known link weights. However, in the inference network, the sites at a unit are independent of each other. Changes at any site (or sites) can be used to update a unit value. In other words, a unit does not have to collect all its site values to calculate its own value. When a new candidate arrives, the optimum-so-far will be compared with the new candidate to find a new optimum. In a neural network, a neuron usually collects all the inputs and then produces an output (for example, a weighted sum). The inputs are not explicitly divided into groups. Figure 2.5 compares an inference network component with a classical sigmoid artificial neuron.

It is interesting to compare the inference network with a specific type of neural network, namely, the maximum neural network [49]. In a maximum neural network, the clusters are responsible for decision-making, just like the units in the inference network.

However, all neurons in a cluster, except the one with the highest value, are set to zero once the decision is made; whereas a unit in the inference network chooses the optimum from site values, but it does not reset the site values. Each unit in the inference network has one output; whereas in a general maximum neural network, all neurons in a cluster send out their values. Neither the maximum neural network nor the inference network require a rigorous synchronization procedure. The inference network guarantees optimal solutions to the shortest path (Chapter 4) and transitive closure (Chapter 5) problems. Its algorithm for the assignment problem (Chapter 6) provides valid near-optimal solutions. The maximum neural network provides valid near-optimal solutions to the max cut and bipartite subgraph problems [48, 49].

An inference network with N^2 units can be considered as a sub-net of a maximum neural network with N^2 clusters. The former has less links. If the extra links are taken out from an N^2-cluster maximum neural network, and the *winner-take-all* rule for each cluster is defined as the unit function, the N^2 cluster maximum neural network will become an inference network. A fully connected N^2-cluster maximum neural network is more powerful than an N^2-unit inference network because the former uses more links.

Despite the similarities between the maximum neural network and the inference network, the inference network topology does not naturally match problems like max cut and bipartite subgraph, and the author is not aware of any solution on a maximum neural network to solve the shortest path, transitive closure or the assignment problems.

In Lee's solution to the bipartite subgraph problem [49], the value change of each neuron is based on the gradient descent of the function to be optimized, although winner-take-all scheme is used in each cluster. Even if the maximum neural network is used to solve the shortest path or the transitive closure problem, as long as the states of the neurons change in the direction of gradient descent, whether the solution is optimal is still questionable.

2.7 The Inference Network Cannot Be Treated as a Hypercube

A hypercube [26, 75] is a topology commonly adopted by many parallel machines. The inference network is a topology derived from simple intuitive procedures of optimization. Node connectivity is defined as the number of distinct links connected to a hypercube node or an inference network unit, which is a constant for both the hypercube and the inference network. The ratios between any two of node connectivity, total number of nodes (units), and total number of links represent some features of a topology. The inference network cannot be treated as a hypercube because its ratios are quite different from the corresponding ratios of a hypercube. Figure 2.6 shows the topology of a simple hypercube and a simple inference network.

If we define the space compression factor as the ratio of total number of nodes (units) to node connectivity, then the hypercube topology has a higher space-compression factor than the inference network. For a problem with size N, the inference network has N^2 units. If a hypercube with the same number of nodes is to be used ($2^M = N^2$), the dimension of the hypercube is

$$M = \log_2(N^2) \tag{2.10}$$

The node connectivities for the inference network and the hypercube are $2(N-1)$ and M respectively. For any $N > 2$, the M which satisfies Eq.(2.10) is smaller than $2(N-1)$. This indicates that the amount of communication required naturally by optimization problems is higher than the hypercube with the same number of processors can provide. The following table shows the connectivities of the inference network and a hypercube. Refer Figure 2.6 for illustration.

Inference network topology for directed problems

problem size (N)	3	4	5	6
connectivity ($2(N-1)$)	4	6	8	10
total units(N^2)	9	16	25	36
total links ($(N-1)N^2$)	18	48	100	180

Hypercube topology

hypercube dimension (M)	3	4	5	6
connectivity (M)	3	4	5	6
total nodes (2^M)	8	16	32	64
total links ($M * 2^{(M-1)}$)	12	32	80	192

Nevertheless, it is possible to implement the inference network on a piece of hardware which has a hypercube topology. In this case, a link in the inference network has to be implemented as an indirect path through several processors. The transformation mapping of the inference network to a systolic architecture, to be discussed in Section 3.3, is an example.

The communication requirement of the inference network is a major challenge to computer technology. Each unit in the N^2-unit network has links to $2N$ other units. This amount of communication cannot be accommodated by many existing digital parallel processing facilities. However, since modern technology is able to build a fully connected analog neural network chip (for example with 50 neurons), it is technically possible to build a VLSI chip for the inference network.

2.8 Examples and Potential Applications

Let us look at some application examples of the inference network. The objective of the shortest path problem is to find the shortest path between two cities in a given distance graph. In this application, a unit value represents a distance between two cities. Unit (i,j) looks for the shortest path from city i to city j. Site l on this unit calculates the distance from city i to city j via city l by adding two unit values. Therefore, each site on the unit provides the distance of one path and the unit obtains the shortest path by minimizing the site values.

Another example is to use the network for data consistency checking. Given N events and the time intervals between any two of them, the objective is to check whether these time intervals are consistent with each other. The inference network for this problem is composed of $\frac{N(N-1)}{2}$ units, and each unit (i,j) represents the time interval between event i and event j. At site k, the time interval between event i and event j is obtained from intervals from event i to event k and from event k to event j. The given data are consistent if the intervals between i and j from all sites are equal. In this application, site function is an addition and unit function checks consistency of site values.

The inference network topology can be extended to a weightless neural network for the consistency labeling problem. The weightless neural network consists of clusters of neurons. Each cluster (i,j) is more complicated than a unit (i,j), however, it also works for one binary relation — the consistency between the labels for object i and object j. The interconnection between the clusters is the same as the interconnection between the units: cluster (i,j) communicate with cluster (i,k) and cluster (k,j) for all k's. Cluster (i,j) eliminates inconsistent labels for object i and j, and whenever the eligible labels for object i (or object j) change, the information is passed to cluster (i,k) (or cluster (k,j)) for all k's. Details about this weightless neural network can be found in Appendix C.

The inference network can be applied to an N-node problem if the problem has the following features:

1. Since a unit only keeps a binary relation, the problem must be fully represented by a set of binary relations u_{ij}, where i and j are nodes of the problem. The nature of the relations u_{ij} are the same for all i and j;

2. With the links provided by the inference network, the state of the problem must progress through the interaction of the binary relations, and relation u_{ij} is only directly used in determining relation u_{il} and u_{lj};

3. Due to the structure of each site, candidates for the optimal u_{ij} are obtained from a binary function on u_{il} and u_{lj};

4. Deduction rules are defined to derive s_{ijl} from u_{il} and u_{lj};

5. Decision-making rules are defined to determine relation u_{ij} from $s_{ij1}, s_{ij2}, \cdots, s_{ijN}$;

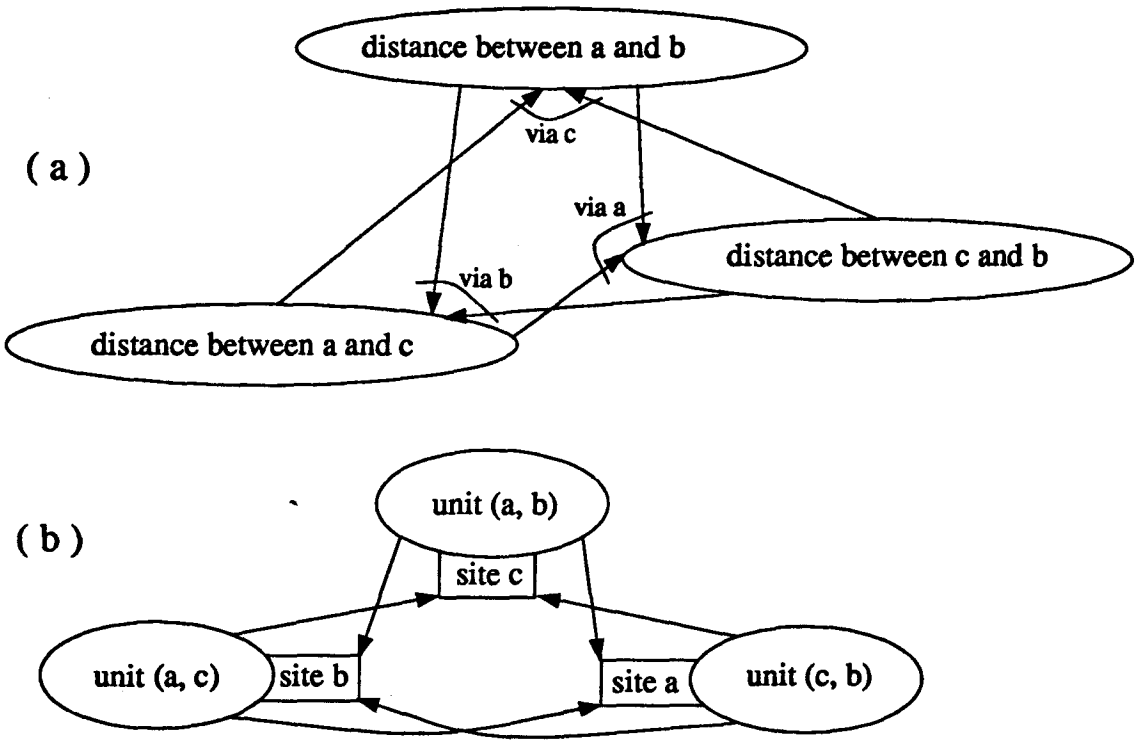
6. An energy function can be defined for the problem, and through the site and unit updating, it will eventually arrive at a global or local minimum when the network is stabilized.

Among the above features, the first two are the most important, and they can be used to determine if it is possible to solve a problem on the inference network platform. Although the longest path problem — finding the longest path in a general graph — sounds similar to the shortest path problem, it can not be solved on the inference network because the second and third requirements are violated. In the shortest path problem, the shortest path length from node i to j always equals to the total of the shortest path lengths from i to k and from k to j for all ks. However, the longest path from i to j often is not the concatenation of the longest path from i to k and from k to j for any

k. Despite that the shortest path problem is solved nicely on the inference network, the longest path problem cannot be solved easily on the same network. If a relation between i and j cannot be determined from the relation between i and k and the relation between k and j, the kind of deduction is not supported by the inference network.

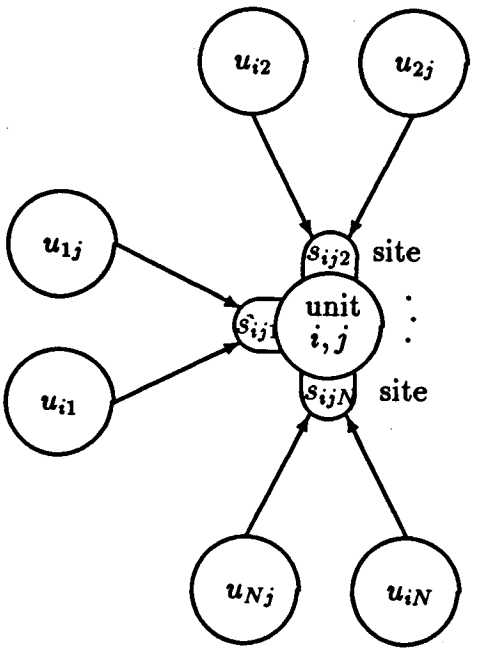
The inference network is not suitable for any problem that cannot be fully represented by a set of simple binary relations. For example, if there are more than one salesmen in the traveling salesman problem, and the salesmen's destinations are associated with some priorities, it is not easy to define binary relations u_{ij} $\forall i, j$ such that they can fully represent a state of the problem. On the other hand, even if it is possible to fully represent the complicated traveling salesman problem using a set of binary relations, the deduction upon the relations is probably too complicated for the inference network platform to support (the second and third requirements above).

For what kind of optimization problems can the inference network produce global optimal solutions? In the inference network, each unit performs local optimization based on its inputs and the unit function. The shortest path (Chapter 4), transitive closure (Chapter 5), data consistency checking (Appendix A), and consistency labeling (Appendix C) are a few examples with a common feature: local optimization of a binary relation is always advantageous to the optimization of other binary relations. For applications with this feature, the inference network produces globally optimal solutions. For example, in the shortest path problem, any unit with a non-optimal path continues to optimize and activates other units to change accordingly. Once all units have settled down, a solution to the all-pair shortest path problem is found.



Inference Network	AND/OR graph	Description
unit	oval node	a binary relation to be optimized
site	arc for AND	associating two relations for a result
link	arrow	showing dependence of relations
sites on a unit	untied OR logic	independent ways to derive a relation

Figure 2.3: (a) A complete AND/OR graph for three relations; (b) the corresponding inference network of (a).



General configuration:
$1 \leq i, j, l \leq N$
N^2 units, N sites per unit
$2N$ incoming links per unit

No-diagonal-units configuration:
$1 \leq i, j, l \leq N \quad l \neq i \neq j$
$N(N-1)$ units, $N-2$ sites per unit
$2(N-2)$ incoming links per unit

Undirected network configuration:
$1 \leq i, j, l \leq N \quad l \neq i \neq j$
unit (i,j) coincides unit (j,i)
$N(N-1)/2$ units, $N-2$ sites per unit
$2(N-2)$ incoming links per unit

Figure 2.4: Unit (i,j) in the binary relation inference network.

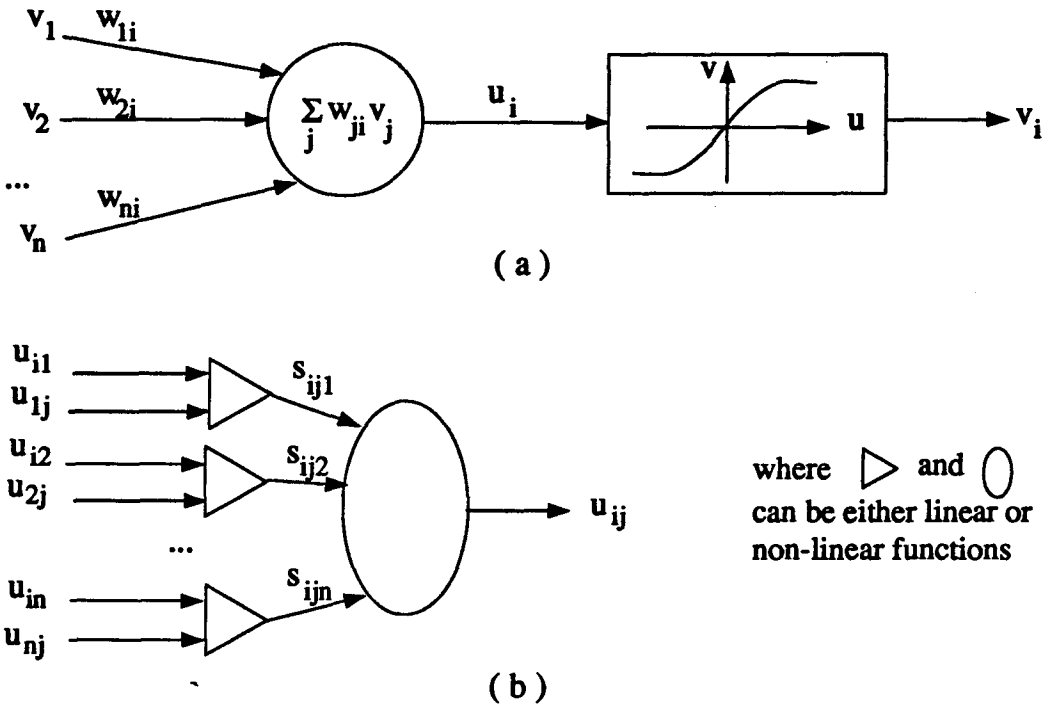


Figure 2.5: (a) A classical sigmoid artificial neuron; (b) a basic component in the inference network.

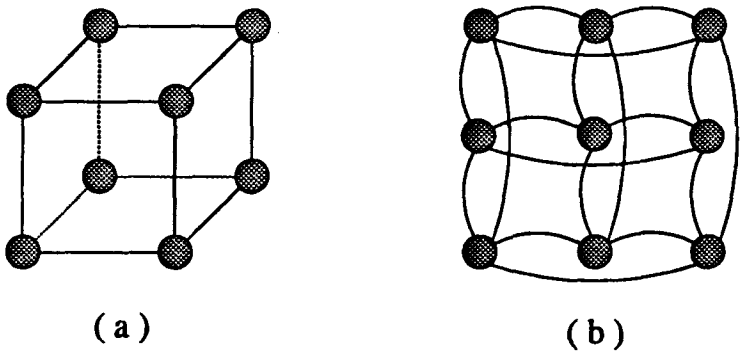


Figure 2.6: (a) Topology of a 3-dimensional hypercube with 8 processors and 3 bidirectional links on each processor; (b) a full $N = 3$ inference network with 9 units and 4 bidirectional links per unit.

Chapter 3

Inference Network Implementation

3.1 Analog Inference Network

Each unit in the inference network is a simple computational engine. The following chapters will show that for a variety of applications, operations involved in the site and unit functions involve only a few basic arithmetical operations. Therefore, a unit can be built by wiring up some operational amplifiers or logic gates operating in continuous time. An output of a site is sent to a single unit as input; an output of a unit drives a number of sites.

Figure 3.7 shows two units and the sites attached to them. Figure 3.7(a) is a network component for the shortest path problem. An analog unit for a 5 city shortest path problem was built and tested [91]. The value of a unit is the voltage measurement at the output of a unit. Each site is an operational amplifier to add two voltages. Each unit is composed of some voltage comparator to choose the minimum voltage among the site outputs. The inference network algorithm for this application will be given in Chapter 4. Figure 3.7(b) is a unit and its attached sites for the transitive closure problem. A 'high' unit value indicates that there is a path between the two nodes. Each site is an *AND* gate determining the existence of a specific path between two nodes. The unit is an *OR* gate determining the existence of any path between the two nodes. Detailed discussion can be found in Chapter 5.

What spatial configuration will the units form if the inference network is to be built

physically? Let us answer the question by looking at the connectivity of the units. First, if there is a link from the first unit to a site at the second unit, then there must be a link from the second unit to a site at the first unit, refer Figure 2.3. Therefore, bidirectional links can be physically used. Second, all units are not directly connected. If the N^2 units of the inference network are placed on a 2-dimensional array and unit (i, j) sits at row i and column j, only units within the same row or column have links in between.

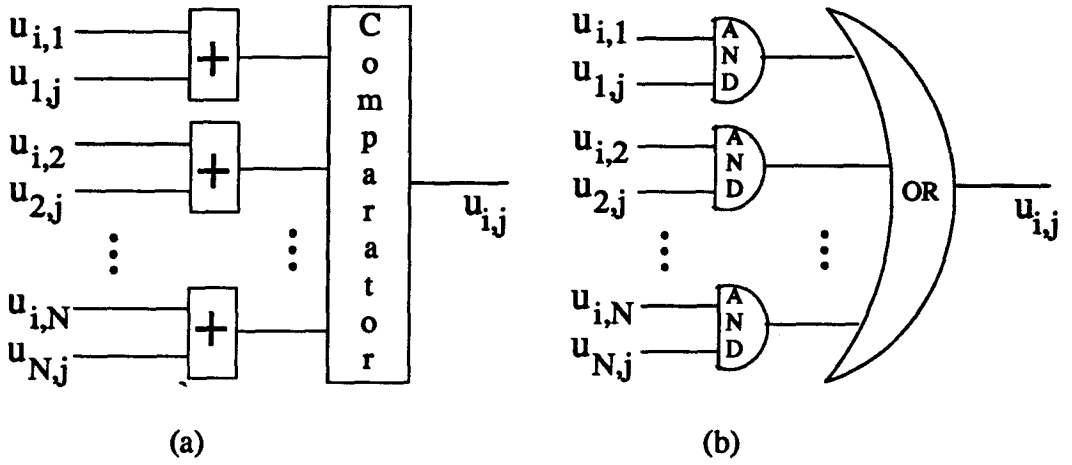


Figure 3.7: Examples of inference network components, (a) for the shortest path problem, (b) for the transitive closure problem.

In light of these features, a torus structure, as shown in Figure 3.8, can be used. It is formed by layered circular disks and a cylindrical surface wrapped on these disks. Each unit locates at the center of a layer, and communicates with all other relevant disks through bidirectional links in the cylindrical surface. For a directed problem with size N, there are $N^3(N-1)$ bidirectional links on the cylindrical surface.

Since only the units in the same row or column are connected, it is possible to use multiple buses, and each bus only contains unit signals in the same row or column. Figure 3.9 shows this structure. Each unit is connected with a vertical bus and a horizontal

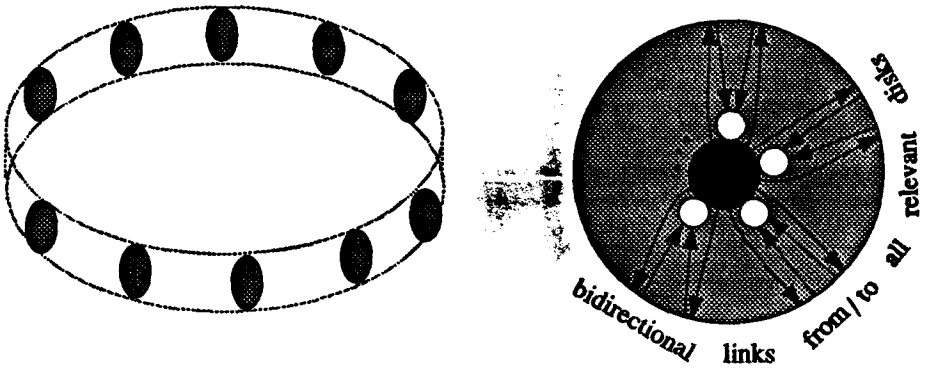


Figure 3.8: (a) Torus structure of the inference network; (b) a circular disk.

bus; each bus contains signals from units in a particular row or column. Units get inputs from two bus lines, calculate outputs, and send results back to the two bus lines to drive other units. The width of each bus is N and there are $2N$ buses in total.

Despite the feedback in the network, for each of the applications considered in this thesis, it is shown that the inference network has stable dynamic behavior and eventually it will converge. An energy function can be defined for each application and can be proved to be decreasing. The feedback links in the network will not cause oscillation.

A complete analog inference network system consists of a conventional computer, the inference network formed by circuits or VLSI chips, and D/A and A/D converters, as shown in Figure 3.10. The computer is a monitor as well as a storage medium. The analog inference network is composed of a physical configuration (torus, square, etc.), and units plugged in to it. The units can be multi-functional, which have switches to determine their functionality for the given application, or, they can be simple units for a specific application. Sites at a unit are evaluated in parallel.

The advantage of using analog units instead of digital ones is due to the simplicity of analog units. An analog system uses additional D/A and A/D, but it does not need a set

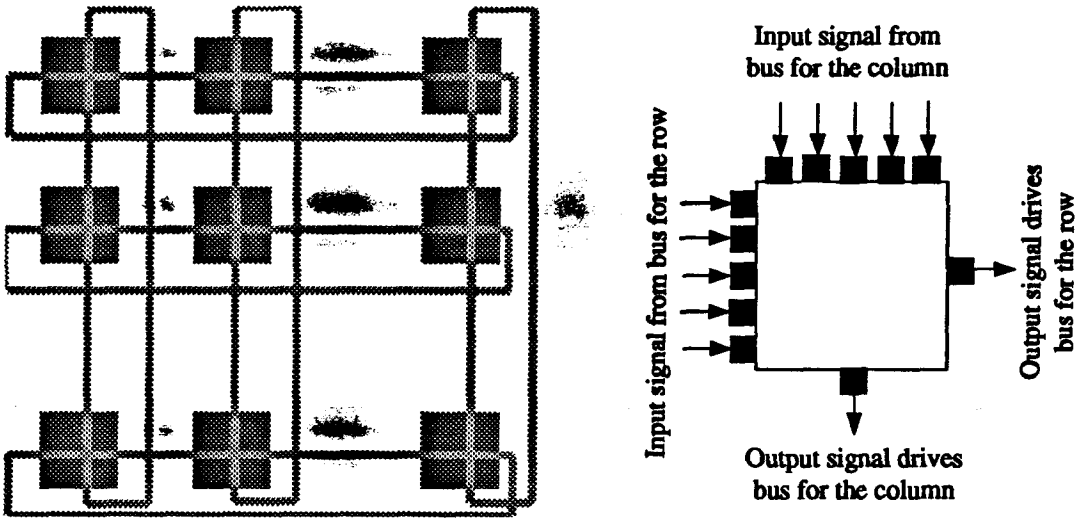


Figure 3.9: (a) A square structure with multiple sets of buses; (b) connection of a unit to bus lines.

of circuits for each bit of the data, whereas a digital system does. Besides, the units in an analog system do not need a clock to synchronize their operations. A potential problem with the above implementation is the limited driving power of the output signals. When the network size is large, additional driving circuits may be necessary.

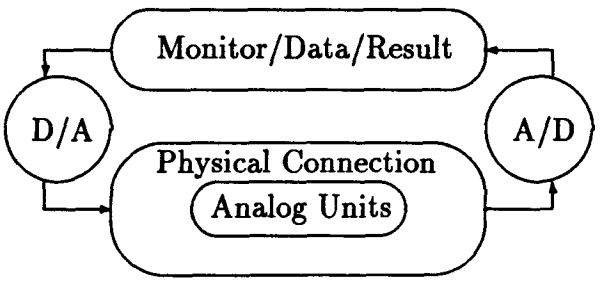


Figure 3.10: An analog inference network system.

3.2 Discrete-Time Inference Network

The computational requirement for the components of the inference network is simply the ability to perform basic arithmetical operations. A site produces an output based on two inputs; a unit collects the site values and generates a unit output. Neither a sophisticated CPU nor a big memory is needed. However, each unit does require a fairly big communication capacity: each unit has $2N$ outgoing links and $2N$ incoming links.

A discrete inference network can be built using digital components similar to those shown in Figure 3.7. Physical connection of units are also similar. The major difference is, in order to update the network synchronously, a global clock is required to synchronize the activities of each component. At the beginning of each time step, sites attached to a unit obtain inputs through its incoming links, calculate site outputs simultaneously, and then the unit performs its unit function and produces the output for the next time step.

The Connection Machine (CM) [26, 85] can be programmed as the inference network. The Connection Machine is a single instruction multiple data stream computer which consists of a large number of simple processors and is efficient for massively parallel computing. It has up to $16K$, $32K$ or $64K$ processors; each processor has up to $1MBit$ of memory. Its processors are connected logically into an n-cube topology and support highly efficient n-dimensional grid communications. An algorithm can be executed on the Connection Machine efficiently if it mainly requires regular and local communication. Non-local communication can be achieved with the trade-off in speed. If an application requires more processors than the CM physically has, virtual processors can be created. The CM uses a conventional computer, such as SUN or VAX, as a front-end machine. Several high-level languages for parallel programming are supported, such as *Lisp, C*, CM Fortran, which are parallel extensions of the corresponding sequential languages. CM provides the utility to measure computational time: CM busy time is the time the CM

is involved in calculation; CM elapsed time includes the time the CM is communicating with its front end, in addition to CM busy time.

When the Connection Machine is programmed as the inference network, a processor array can be declared in which each processor corresponds to a unit in the inference network. The network can operate in both synchronous and asynchronous modes. Each processor performs site operations as well as unit operation. To obtain site values of a unit, each processor is required to communicate with $2(N-1)$ other processors. Unit values can then be calculated within each processor. Although each CM processor has only four hardware links with its neighbors, arbitrary communication can be achieved from software level. For example, C^* code

$$[i][j]work = [k][l]work$$

sends the element [k][l] of vector variable *work* to position [i][j]. The programmer does not need to do any hardware reconfiguration. If the two elements are not hypercube neighbors, the message has to travel through other processors to get to its destination. However, since the cost of non-local communication increases rapidly with problem size N, this way of using CM is very inefficient.

3.3 Mapping to the CM Using Local Communication

Under the situation that existing hardware does not support the operations of the inference network efficiently, we propose a systolic transformation mapping of the network to the Connection Machine. In the previous section, all the sites of a unit constantly reside on the unit. However, since there is only one processor for these sites, site values are evaluated in serial, and site evaluation normally involves non-local communications. The high communication cost caused by directly programming the CM can be significantly reduced by proper arrangement of site calculations at each unit: each unit value passes

through a set of units (processors) and stops at each unit to evaluate one site of the unit; each unit calculates its site values in its own order.

Here are the details. The $N \times N$ inference network is mapped onto an $N \times N$ processor array on the CM, each processor for one unit. See Figure 3.11. Each processing element (PE) has a local memory keeping the unit value. Two data flows, one horizontal and one vertical, pass though each PE constantly. Each data flow consists of N unit values from a certain row or column. Since each unit only communicates with all the units in the same row or column, a horizontal (or vertical) data flow can be shared by all the units in the same row (or column). The data flows in the two directions are ordered in such a way that the two inputs of a certain site arrive at a PE at the same time. Eq.(3.11-3.12) is the initialization of the data flows:

$$h[i, \ (-i-j) \pmod N] := u[i,j] \tag{3.11}$$

$$v[(-i-j) \pmod N, \ j] := u[i,j] \tag{3.12}$$

where $h[k,l]$ and $v[k,l]$ are the data at $u[k,l]$ after the initialization. It takes N time units for the data flows to traverse the array. In this mapping, the N sites are not physically attached to a unit all the time, but are attached to the unit one after another rhythmically. For most optimization problems, the unit function is separable, as shown in Eq.(2.4), hence a PE does not need extra memory to keep any site values.

One synchronous iteration is composed of $N+1$ steps on the CM:

- form the vertical and horizontal data flows (prepare updated unit values for the site inputs) according to Eq.(3.11-3.12);
- repeat N times:
  - update local memory (calculate site values and revise the unit value);

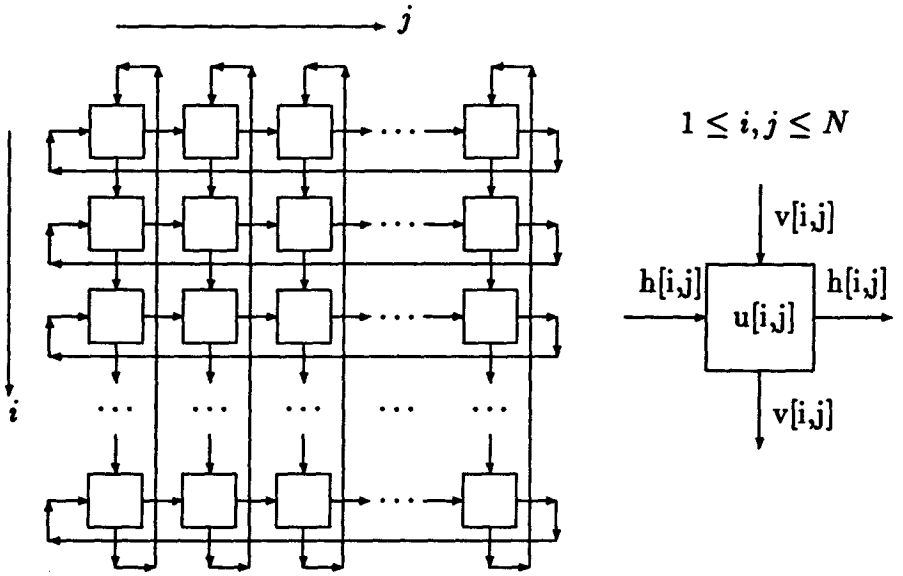


Figure 3.11: The systolic mapping of the inference network to the Connection Machine.

– propagate horizontal and vertical data.

The result of this mapping is that each unit only has to communicate with its four neighbors. Table 3.1 shows that the mapping is far more efficient than directly program the CM into the inference network, which uses non-local communication.

N	4	8	16	32	64	128	256	512
Direct program	0.062	0.313	0.818	1.828	5.957	15.72	133.04	1029.0
Mapping	0.087	0.113	0.176	0.296	0.701	1.88	8.21	37.1

Table 3.1: Connection Machine busy time (in seconds) to solve the all-pair shortest path problem. Direct program: the CM is simply programmed as the inference network; Mapping: the inference network is mapped to a processor array using local communication.

This mapping of the inference network to a square processor array on the Connection Machine makes good use of the available processors. Because of the restrictions of the

CM, a rectangular geometry must be declared on the CM if a systolic algorithm uses a parallelogrammic processor array. Therefore, the rectangular geometry, which completely covers the parallelogrammic array, results in a higher virtual processor ratio (defined as the number of virtual processors over the number of physical processors). Table 3.2 gives the Connection Machine busy time for the inference network (refer Appendix B) and a systolic algorithm [70] to complete a matrix inversion. The inference network uses a square processor array whereas the systolic algorithm uses a parallelogrammic one. The two algorithms have similar operations. When the problem size is large, the inference network is significantly more efficient. More details can be found in [80, 78]

N	4	8	16	32	64	128	256	512
Inference Network	0.22	0.39	0.88	1.21	1.91	3.86	17.73	115.35
systolic algorithm	0.09	0.17	0.38	0.89	2.31	8.27	37.29	210.38

Table 3.2: CM busy time (in seconds) for the inference network and a systolic algorithm to solve $(\mathbf{I} - \mathbf{A})^{-1}$ on a $16K$ CM.

Although the mapping of the inference network to the CM is more efficient than directly program the CM into the inference network, it must be made clear that this mapping is just a vehicle to utilize a piece of existing hardware. It is not the end of the inference network itself. As a matter of fact, the mapping introduces additional restrictions on the inference network: it requires the sites to be evaluated in specific orders and unit calculation to be strictly synchronized.

Chapter 4

Inference Network for the All-Pair Shortest Path Problem

This chapter discusses the inference network algorithm for the all-pair shortest path problem. For discrete synchronous updating, the algorithm was shown to converge to the global minimum state within $\lceil \log_2(N-1) \rceil$ iterations[1]. Unlike most existing discrete approaches and some neural network approaches, the algorithm was also shown to converge to the optimal solution for asynchronous or continuous-time updating. The inference network was proved to converge, independent of problem size, to the optimal solution for the problem. It was demonstrated that the inference network not only can provide an efficient platform to solve the shortest path problem efficiently, but also can unify many established algorithms for the problem by defining different unit and site functions.

4.1 The Shortest Path Problem

The shortest path problem has the following description. Consider a densely connected graph with N nodes denoted as $0, 1, \ldots, N-1$ and a set of directed arcs A. Every arc from node i to node j in A is associated with a distance (or length, cost, time etc.) d_{ij}. For those node pairs without an arc in between, define their distance as infinity. Assume further that $d_{ii} = 0$ if it is not otherwise specified. The length of a path is the total length of the arcs on the path. The problem is to find the shortest paths between all pairs of nodes. Rote [71] gives more formal definition of the problem using semiring theory.

[1] $\lceil x \rceil$ ($x \geq 0$) is the smallest integer greater or equal to x.

Generally, d_{ij} can be negative as well as positive. However, if a directed *loop* with negative length exists in the distance graph, the shortest paths will have negative infinite length — because the more a path repeats the loop, the shorter the path can be. In order to have a well defined problem, we restrict our discussion to graphs containing no negative-length loops. (Negative-length arcs may exist, as long as any closed loop in the graph has a positive or zero length.) A subset of these graphs are those containing undirected arcs with non-negative lengths, that is, in mathematical terms, $\forall i,j\ d_{ij} = d_{ji} \geq 0$ where $0 \leq i,j \leq N-1$.

Finding the shortest paths in graphs or trees is an intensively studied problem in artificial intelligence and operations research. Established discrete algorithms for the shortest path problem fall into three categories. Dijkstra's algorithm [10] (to be referred to as DIJKSTRA) is the most efficient one among the *single-pair* algorithms, which find the shortest path between a specific pair of nodes at a time. There are several algorithms which can find the shortest paths from a specific source node to all other nodes at the same time. Yen's [94], Spira's [73], and Moffat's[63] algorithms (to be referred to as YEN, SPIRA, MOFFAT, respectively) are examples of this kind of *single-source* algorithms; among them SPIRA is the most efficient. The third class of algorithms, *all-pair* algorithms, find the shortest paths between all the node pairs at the same time. This class includes the matrix and revised matrix algorithms [13, 32, 93], Floyd's [17], Dantzig's [9] and modified Dantzig's [81] algorithms, to be referred to as MATRIX, RMATRIX, FLOYD, DANTZIG, MDANTZIG, respectively. Systolic algorithms like Rote's [71] and Robert and Trystram's[70] (to be referred to as ROTE and ROBERT respectively) are also *all-pair* algorithms. All-pair algorithms are easier to be parallelized.

The shortest path problem is also solved by neural network approaches. Fritsch [19] discussed eigenvalue analysis method and proposed to use self-organizing feature map

and a modified Hopfield net to solve the problem. Thomopoulos [83] introduced a neural network algorithm run on a network structure similar to the Hopfield net. Like neural network approaches to other problems, these algorithms can not guarantee an optimal solution. The problem is also solved by the connectionist approach in digital environment[25]. It obtains the solution through relaxation with self-adapting step size.

The shortest path problem has applications in communication, transportation networks, integrated circuit layout, and control, etc. Applications of the single-pair shortest path problem are common in daily life. Single-source and all-pair solutions are useful in VLSI layout etc. For example, compaction is a process in VLSI layout aiming at minimize the layout area while meeting all the design rules. It is an optimization problem because masks can move around as long as spacing, minimum-size, and shape rules are met. One-dimensional compaction changes one coordinate of the masks while preserving the design rules and not altering the function of circuits. In two-dimensional compaction, both coordinates can change simultaneously to minimize the area. Solving a single-source shortest path problem is one of the ways to perform a one-dimensional compaction. The difficulty of two-dimensional compaction lies in determining how the two dimensions of the layout must interact to minimize the area. One way out of this difficulty is to do two one-dimensional compactions under some additional constraints. The test of the constraints involves solving two all-pair shortest path problems. Lengauer [51] gives detailed algorithms for them.

4.2 Inference Network Algorithm for the Shortest Path Problem

By giving proper unit and site functions, the inference network can be used for the all-pair shortest path problem. Each unit (i, j) $(0 \leq i, j \leq N-1)$ in the network works for the shortest path from node i to node j. Its value u_{ij} is the shortest path length between

i and j found so far. Site l at unit (i,j) records a path from node i to node l and then to node j. Its value s_{ijl} is the sum of the path lengths from i to l and from l to j. All links have weight 1. For undirected shortest path problem with no negative-distance path, initial value $u_{ij}^{(0)}$ is the given distance d_{ij}. For directed graph, $u_{ij}^{(0)}$ are obtained from distances d_{ij} using Eq.(4.13). If a path from node i to i exist with positive length, the shortest path has, by common sense, zero length. If the path from i to i has a negative length, by repeating this loop, the length of the shortest path will approach negative infinity.

$$u_{ij}^{(0)} := \begin{cases} d_{ij} & i \neq j \\ 0 & i = j, \quad d_{ij} \geq 0 \\ -\infty & i = j, \quad d_{ij} < 0 \end{cases} \tag{4.13}$$

Each site on a unit performs

$$s_{ijl} := u_{il} + u_{lj} \tag{4.14}$$

and all the units in the network perform

$$u_{ij} := \min\{u_{ij}, \min_{\forall l}\{s_{ijl}\}\} \tag{4.15}$$

Sites and units can be updated asynchronously, synchronously with a clock, or in continuous-time.

The algorithm for synchronous discrete time, asynchronous discrete time, and continuous-time domains are given in Algorithm 1, and Algorithm 2, Algorithm 3 respectively.

Algorithm 1 : *Shortest Path — Synchronous (to be referred to as* INFERENCE*)*

set initial unit values according to Eq.(4.13)

$k := 0$

while (1)

BEGIN

BEGIN PARALLEL for all i, j

BEGIN PARALLEL for all l

$$s_{ijl}^{(k)} := u_{il}^{(k-1)} + u_{lj}^{(k-1)}$$

END PARALLEL

$$u_{ij}^{(k)} := \min\{u_{ij}^{(k-1)}, \min_{0 \le l \le N-1}\{s_{ijl}^{(k)}\}\}$$

END PARALLEL

if $\forall i, j \;\; u_{ij}^{(k)} = u_{ij}^{(k-1)}$ *break;*

$k := k + 1$

END

Algorithm 2 : *Shortest Path — Asynchronous*

set initial unit values according to Eq.(4.13)

REPEAT

randomly pick a unit (i, j)

BEGIN PARALLEL for all l

$$s_{ijl} := u_{il} + u_{lj}$$

END PARALLEL

$$u_{ij} := \min\{u_{ij}, \min_{0 \le l \le N-1}\{s_{ijl}\}\}$$

UNTIL no unit value changed when any of the units is picked.

Algorithm 3 : *Shortest Path — continuous time*

- *Set initial unit values according to Eq.(4.13);*
- *All sites at each unit perform function* $s_{ijl} := u_{il} + u_{lj}$;
- *All units perform* $u_{ij} := \min\{u_{ij}, \min_{0 \le l \le N-1}\{s_{ijl}\}\}$;
- *Sites and units will eventually stabilize and each unit value represents the distance of a shortest path.*

4.3 Algorithm Convergence

There are two important considerations in using the inference network for the shortest path problem. First, will the network always converge to the desired solution? Second, what are the parameters or conditions that affect the convergence rate of the network? The answer to the first question is 'yes', because at the k-th iteration, *all* paths containing at most 2^k arcs take part in the minimisation; and the local minimisation performed at each distinct unit drives the network to a global minimum state — which is the desired solution. The convergence rate of the network is $O(log_2 N)$.

4.3.1 Synchronous Algorithm Converges in at most $\lceil \log_2(N-1) \rceil$ Iterations

Since all the shortest paths containing at most 2^p arcs can be found in p iterations and a shortest path in an N-node graph contains at most $(N-1)$ arcs, the inference network converges to the solution in at most $\lceil log_2(N-1) \rceil$ synchronous iterations.

In Algorithm 1, the inference network algorithm uses distances d_{ij}, which is the length of one arc, in the first iteration to calculate the shortest paths containing at most 2 arcs. In the second iteration, it uses the shortest paths containing 1 or 2 arcs to calculate shortest paths containing at most 4 arcs. Repeating this procedure p times, all the shortest paths containing at most 2^p arcs can be found. In other words, a shortest path containing q arcs can be found in $\lceil \log_2 q \rceil$ iterations. Since it is assumed that there is no negative loop in the graph and, by common sense, a shortest path does not contain a closed loop with non-negative length, shortest paths are all open paths. If an open path contains at least one infinite-distance arc, then the length of the path, which is infinity, can be determined right away. If there are a number of open paths between node i and j which are composed of finite-distance arcs only, the k-th of them contains $l_k(i,j)$ such arcs, then the number of iterations required, n_{ij}, to find the shortest path between node

i and j falls into range:

$$\lceil \log_2(\min_k \{ l_k(i,j) \}) \rceil \leq n_{ij} \leq \lceil \log_2(\max_k \{ l_k(i,j) \}) \rceil$$

The algorithm completes when the shortest paths between any two nodes are found, therefore, the number of iterations required, n, to find all shortest paths is given by

$$\lceil \log_2(\max_{i,j} \{ \min_k \{ l_k(i,j) \} \}) \rceil \leq n \leq \lceil \log_2(\max_{i,j} \{ \max_k \{ l_k(i,j) \} \}) \rceil$$

For an N-node problem, an open path consists of at most $N-1$ arcs, that is

$$\max_{i,j} \{ \max_k \{ l_k(i,j) \} \} \leq N-1$$

hence all shortest paths can be found in at most $\lceil \log_2(N-1) \rceil$ iterations. Usually, the network requires fewer iterations to converge than this upper bound. The number of iterations required depends solely on the maximum number of arcs in an open path in the graph; it is not directly related with graph density.

4.3.2 Convergence of the Asynchronous Algorithm

All the algorithms we found in literature for the shortest path problem are synchronous algorithms. However, it can also be proved that using the same updating rules asynchronously, the inference network Algorithm 2 converges to the global minimum state as well. For asynchronous updating, the unit which has the least chance to update is most likely the one which determines the completion time.

In synchronous evolution, $u_{ij}^{(k)}$ is the shortest path length after k-th iteration, and it gives the shortest path among the paths from i to j which contain at most 2^k arcs each. For asynchronous updating, we need a record n_{ij} to indicate that all the paths from i to j which contain no more than n_{ij} arcs have been considered in minimizing u_{ij}. Some, but not all, paths contain more than n_{ij} arcs may also contribute in the minimisation.

For any n_{ij}, u_{ij} in the asynchronous case is less or equal to $u_{ij}^{(\lceil \log_2 n_{ij} \rceil)}$ in the synchronous case. Site and unit functions for the asynchronous inference network are the same as in synchronous evolution for each individual unit and site. The initial value of n_{ij} is always 1, and it is updated according to

$$n_{ij} = \min_l \{n_{il} + n_{lj}\}$$

In each iteration, a random unit (i, j) is picked to be minimize. In addition to calculating and updating s_{ijl} and u_{ij}, n_{ij} can be recorded. Paths are all minimized if $\forall i, j \quad n_{ij} \geq N - 1$. If units are updated one by one systematically, and one round of network updating starts only when all the units are updated in the previous round, then it takes $N(N-1)\lceil \log_2(N-1) \rceil$ iterations to obtain the shortest paths for all node pairs. If updating is in a random order, minimizing a unit (i, j) with its n_{ij} greater than that of some other units just makes the updating process longer. Hence for random updating, the number of iterations required is at least $N(N-1)\lceil \log_2(N-1) \rceil$, and depends on the unit (i, j) which increases its n_{ij} value slowest. The asynchronous algorithm will eventually converge, that is, arrive at the state of $n_{ij} \geq N - 1 \ \forall i, j$, if each unit has a chance to update sometime.

It is worth point out that n_{ij} is used here to prove the convergence of the algorithm. However, it is not necessary to calculate them if only the shortest paths are to be found.

4.3.3 Convergence of the Analog Algorithm

In the analog shortest path algorithm, each unit performs

$$u_{ij} := \min\{u_{ij}, \min_{0 \leq l \leq N-1} \{s_{ijl}\}\} \tag{4.16}$$

Taking into account the transient period and assuming that each unit is a first-order system, the behavior of a unit can be described by

$$\frac{d\,u_{ij}}{d\,t} = -\lambda_{ij}u_{ij} + \lambda_{ij}\left(\min\{u_{ij}, \min_{0 \le l \le N-1}\{s_{ijl}\}\}\right) \tag{4.17}$$

where constant $\lambda_{ij} > 0$ is the open-loop pole of unit (i,j) which partially determines the speed at which u_{ij} changes.

The network energy for this problem can be defined as

$$E = \sum_i \sum_j \left(u_{ij} - \min\{u_{ij}, \min_{0 \le l \le N-1}\{s_{ijl}\}\}\right)^2$$

The global minimum state of the network is signified by $E = 0$, that is $\forall\, i,j,\ u_{ij} = \min_{0 \le l \le N-1}\{s_{ijl}\}$. At this state, unit values will not change any more and the network is stabilized. The following is a proof that the network will arrive at a state sufficiently close to this global optimal state as the elapse of time.

- From Eq.(4.16), $u_{ij} \ge \min_{0 \le l \le N-1}\{s_{ijl}\}$;
- If $u_{ij} = \min_{0 \le l \le N-1}\{s_{ijl}\}$, $\frac{d\,u_{ij}}{d\,t} = 0$, the unit keeps its value;
- If $u_{ij} > \min_{0 \le l \le N-1}\{s_{ijl}\}$, $\frac{d\,u_{ij}}{d\,t} = -\lambda_{ij}\left(u_{ij} - \min_{0 \le l \le N-1}\{s_{ijl}\}\right) < 0$, which drives u_{ij} towards $\min_{0 \le l \le N-1}\{s_{ijl}\}$;
- The farther u_{ij} is from $\min_{0 \le l \le N-1}\{s_{ijl}\}$, the faster u_{ij} decreases. Therefore, $\frac{d^2\,u_{ij}}{d\,t^2} > 0$;
- $\frac{d\,E}{d\,t} \le 0$ when the behavior of a unit is governed by Eq.(4.17):

$$\begin{aligned}
\frac{d\,E}{d\,t} &= \sum_i \sum_j \frac{d\,E}{d\,u_{ij}} \frac{d\,u_{ij}}{d\,t} \\
&= 2\sum_i \sum_j \left(u_{ij} - \min\{u_{ij}, \min_{0 \le l \le N-1}\{s_{ijl}\}\}\right) \frac{d\,u_{ij}}{d\,t} \\
&= -2\sum_i \sum_j \frac{1}{\lambda_{ij}} \left(\frac{d\,u_{ij}}{d\,t}\right)^2 \le 0
\end{aligned}$$

- $\frac{d^2 E}{d t^2} = -4 \sum_i \sum_j \frac{1}{\lambda_{ij}} \frac{d u_{ij}}{d t} \frac{d^2 u_{ij}}{d t^2}$ where $\lambda_{ij} > 0$. when $u_{ij} > \min_{0 \le l \le N-1} \{s_{ijl}\}$, $\frac{d u_{ij}}{d t} < 0$, and $\frac{d^2 u_{ij}}{d t^2} > 0$, hence $\frac{d^2 E}{d t^2} > 0$;

- $\frac{d E}{d t} = 0$ holds only when $\frac{d u_{ij}}{d t} = 0 \ \ \forall i, j$, that is, the network has settled down and $u_{ij} = \min_{0 \le l \le N-1} \{s_{ijl}\} \ \ \forall i, j$ holds (all the paths are optimized);

- if $\frac{d E}{d t} = 0$, then $u_{ij} = \min_{0 \le l \le N-1} \{s_{ijl}\} \ \ \forall i, j$, which means $E = 0$;

- $E \ge 0$, $\frac{d E}{d t} \le 0$, and $\frac{d^2 E}{d t^2} > 0$ show that the network energy is a decreasing convex function which has to approach its stable value zero with the elapse of time. The energy decreases quickly at the beginning, and the decreasing rate slows down when the value of E is close to zero;

- when $\lim_{t \to \infty} \frac{d E}{d t} = 0$, we have $\lim_{t \to \infty} u_{ij} = \min\{u_{ij}, \min_{0 \le l \le N-1} \{s_{ijl}\}\} \ \ \forall i, j$, that is, the network will eventually settle down at the global optimal solution.

4.4 Relationship to other Algorithms

This section addresses the relationship between the inference network algorithm and other existing algorithms, such as A*, MATRIX, FLOYD, and DANTZIG. The essential difference between the inference network algorithm and other algorithms is that the former is derived directly from the topology of the inference network. It is a parallel algorithm with the information of how to be executed on a parallel network. Those existing algorithms are sequential algorithms which, by themselves, do not bear information about how to be parallelized.

4.4.1 A* Algorithm

A* [64] is a heuristic search algorithm. It maintains two lists during the search; the open and closed lists contain nodes which are to be expanded and have been expanded

respectively. To determine the candidate to expand, it uses a heuristic guess of the length from the current node to the goal node. In searching for the shortest path from s to t, the algorithm chooses a node with the least estimated length $f(s,t,n)$ to expand, where

$$f(s,t,n) = g(s,n) + \hat{h}(n,t) \tag{4.18}$$

and $g(s,n)$ is the length of the partial solution from s to n, $\hat{h}(n,t)$ is an estimation of $h(n,t)$ — the shortest path length from n to t.

There is a remarkable degree of computational correspondence between INFERENCE and A*. If n is a node in the open list, unit values $u(s,n)$ and $u(n,t)$ in the inference network correspond to $g(s,n)$ and $\hat{h}(n,t)$ in A*, respectively. Of the four values, $g(s,n)$ is shortest path length between s and n; $u(s,n)$ and $u(n,t)$ are path lengths which may be further minimized; $\hat{h}(n,t)$ is heuristic information. One unit value in the inference network serves as the heuristic information of another unit. Both the heuristic information in A*, $\hat{h}(n,t)$, and the heuristic information used in INFERENCE, $u(s,n)$ and $u(n,t)$, have to be greater than the shortest path length to ensure a correct solution. The difference is that the heuristic information $\hat{h}$ in A* has to be provided prior to the search; whereas the inference network updates the shortest paths between two nodes in each iteration, and the results in one iteration are used in the next iteration as heuristic information until a global minimum is reached.

4.4.2 MATRIX Algorithm

MATRIX is an all-pair algorithm which can be expressed in matrix operations. It defines a matrix binary operation $\otimes$ called minaddition:

$$\mathbf{W} = \mathbf{A} \otimes \mathbf{B} = [w_{ij} = \min_k(a_{ik} + b_{kj})] \quad \text{where } \mathbf{A} = [a_{ij}],\ \mathbf{B} = [b_{ij}] \tag{4.19}$$

If $\mathbf{D}^1 = \mathbf{D}$ is the distance matrix $[d_{ij}]$, the elements of $\mathbf{D}^{N-1}$ gives the shortest path

lengths, where $\mathbf{D}^{N-1}$ is the minaddition of $N-1$ $\mathbf{D}$'s:

$$\mathbf{D}^{N-1} = \mathbf{D} \otimes \mathbf{D} \otimes \mathbf{D} \otimes \cdots \otimes \mathbf{D} \otimes \mathbf{D} \tag{4.20}$$

The divide and conquer version of MATRIX [8, 47] uses the fact that minaddition is associative:

$$\mathbf{D}^4 = \mathbf{D} \otimes \mathbf{D} \otimes \mathbf{D} \otimes \mathbf{D} = \mathbf{D}^2 \otimes \mathbf{D}^2 \tag{4.21}$$

With this feature, only $\lceil \log_2(N-1) \rceil$ $\otimes$-operations are required.

INFERENCE is also intimately related with MATRIX, although the former is derived directly from the inference network platform. It is not difficult to observe that, if we define another matrix binary operation $\perp$ as

$$\mathbf{W} = \mathbf{A} \perp \mathbf{B} = [w_{ij} = \min\{a_{ij}, b_{ij}\}] \quad \text{where } \mathbf{A} = [a_{ij}],\ \mathbf{B} = [b_{ij}] \tag{4.22}$$

then Eq.(4.14-4.15) can be effectively represented as

$$\mathbf{D}^k = \mathbf{D}^{k-1} \perp (\mathbf{D}^{k-1} \otimes \mathbf{D}^{k-1}) \quad \text{where } k = 1, 2, \ldots, \lceil \log_2(N-1) \rceil \tag{4.23}$$

Under the assumption of no negative-length loops, $d_{ii} = 0$, therefore Eq.(4.23) is equivalent to the divide and conquer version of MATRIX. However, the matrix algorithm itself does not carry information of how to implement it on parallel processing arrays, whereas the inference network provides a platform to solve the problem in parallel whose updating equations are mathematically equivalent to the divide and conquer scheme.

4.4.3 FLOYD Algorithm

Floyd's algorithm (FLOYD) constructs optimal paths by inserting nodes, when appropriate, into more direct paths. After k iterations, function $f_k(i,j)$ is evaluated to the optimal path length between i and j, where the intermediate nodes belong to the set of

nodes $\{0, 1, \ldots, k\}$. Its recurrent relation is

$$f_k(i,j) = \min\{f_{k-1}(i,j), f_{k-1}(i,k) + f_{k-1}(k,j)\} \tag{4.24}$$

with boundary condition $f_0(i,j) = d_{ij}$.

The difference between FLOYD and INFERENCE is that the former considers one distinct intermediate node at a time and iterates N times; while the latter consider all intermediate nodes in each iteration and iterates $\log_2(N-1)$ times.

4.4.4 DANTZIG and MDANTZIG Algorithms

Dantzig's algorithm (DANTZIG) constructs shortest paths using arcs whose nodes numbered between 0 and k at the k-th iteration. $g_k(i,j)$ $(i,j \leq k)$, the shortest path length from i to j in the k-th iteration, is obtained from $g_{k-1}(i,j)$, d_{ik} and d_{kj}. Dreyfus [11] gives the algorithm in recurrent form:

$$g_k(i,k) = \min_{0 \leq l \leq k-1}\{g_{k-1}(i,l) + g_{-1}(l,k)\} \quad (i = 0,1,\ldots,k-1), \tag{4.25}$$

$$g_k(k,j) = \min_{0 \leq l \leq k-1}\{g_{-1}(k,l) + g_{k-1}(l,j)\} \quad (j = 0,1,\ldots,k-1), \tag{4.26}$$

$$g_k(k,k) = \min\{0, \min_{0 \leq l \leq k-1}\{g_k(k,l) + g_k(l,k)\}\} \tag{4.27}$$

$$g_k(i,j) = \min\{g_{k-1}(i,j), g_k(i,k) + g_k(k,j)\} \quad (i,j = 0,1,\ldots,k-1) \tag{4.28}$$

where $g_{-1}(i,j) = d_{ij}$ for all i and j. The total number of basic operations required is $2N^2(N-1)$.

Modified Dantzig's algorithm (MDANTZIG) avoids unnecessary operations by further decomposing Eq.(4.25-4.26) into k recurrent steps. Define $g_k^0(i,k) = g_k^0(k,j) = \infty$, $g_k(i,k) = g_k^{k-1}(i,k)$ and $g_k(k,j) = g_k^{k-1}(k,j)$ can be obtained from:

$$g_k^l(i,k) = \begin{cases} g_k^{l-1}(i,k) & \forall 0 \leq i \leq k-1 \quad d_{lk} \geq g_k^{l-1}(l,k) \\ \min\{g_k^{l-1}(i,k), g_{k-1}(i,l) + d_{lk}\} & \forall 0 \leq i \leq k-1 \quad d_{lk} < g_k^{l-1}(l,k) \end{cases} \tag{4.29}$$

$$g_k^l(k,j) = \begin{cases} g_k^{l-1}(k,j) & \forall 0 \le j \le k-1 \quad d_{kl} \ge g_k^{l-1}(k,l) \\ \min\{g_k^{l-1}(k,j), d_{kl} + g_{k-1}(l,j)\} & \forall 0 \le j \le k-1 \quad d_{kl} < g_k^{l-1}(k,l) \end{cases} \tag{4.30}$$

A significant amount of operations can be saved when $d_{lk} \ge g_k^{l-1}(l,k)$ or $d_{kl} \ge g_k^{l-1}(k,l)$ is satisfied.

While the INFERENCE considers all node pairs and all intermediate nodes in each iteration, DANTZIG only considers node pairs whose indices are less than k in the k-th iteration. The former requires at most $\lceil \log_2(N-1) \rceil$ iterations, whereas the latter requires N iterations.

4.5 Unifying Some Established Algorithms Using an Inference Network

Many all-pair shortest path algorithms can be implemented in the inference network by defining corresponding unit and site functions. This section will summarize these functions. For more details, refer Appendix D.2.

4.5.1 Floyd's Algorithm

The inference network implementation for FLOYD needs N^2 units. The unit and site functions are defined as:

$$u_{ij}^{(k)} = \min\{u_{ij}^{(k-1)}, s_{ijk}^{(k)}\} \tag{4.31}$$

$$s_{ijk}^{(k)} = u_{ik}^{(k-1)} + u_{kj}^{(k-1)} \tag{4.32}$$

Initial unit values are $u_{ij}^{(-1)} = d_{ij}$. After N steps, $u_{ij}^{(N-1)}$ gives the shortest path length from node i to node j. Compared with INFERENCE, FLOYD only updates one *specific* site (indexed k) at each unit for each iteration; but it takes N instead of $\lceil \log_2(N-1) \rceil$ iterations to find all the shortest paths.

4.5.2 Revised Matrix Algorithm

Revised matrix algorithm (RMATRIX) updates matrix elements asynchronously to avoid unnecessary operations. Two distinct processes are defined: (i) a forward process

$$d_{ij}^{(f)} = \min_{0 \le l \le N-1} \{d_{il}^{(p)} + d_{lj}^{(q)}\} \quad \text{where } p = \begin{cases} 1 & j \le l \\ f & j > l \end{cases} \quad q = \begin{cases} 1 & i \le l \\ f & i > l \end{cases} \tag{4.33}$$

and (ii) a backward process

$$d_{ij}^{(b)} = \min_{0 \le l \le N-1} \{d_{il}^{(p)} + d_{lj}^{(q)}\} \quad \text{where } p = \begin{cases} f & l \le j \\ b & l > j \end{cases} \quad q = \begin{cases} f & l \le i \\ b & l > i \end{cases} \tag{4.34}$$

where $d_{ij}^{(1)}$ is the initial given distance. Yang [93] shows that the shortest paths can be obtained by one forward process followed by on backward process, one backward process followed by one forward process, three forward processes, or three backward processes.

The inference network for RMATRIX contains N^2 units. Initially, $u_{ij}^{(0)} = d_{ij}$. Site function for both forward and backward processes are the same:

$$s_{ijl}^{(k)} = u_{il}^{(k-1)} + u_{lj}^{(k-1)} \tag{4.35}$$

Unit functions for a forward process is:

$$u_{ij}^{(k)} = \begin{cases} \min_{0 \le l \le N-1} \{s_{ijl}^{(k)}\} & \text{when } k = f(i,j) \\ u_{ij}^{(k-1)} & \text{otherwise} \end{cases} \tag{4.36}$$

where

$$f(i,j) = \frac{\max\{i,j\}(\max\{i,j\} - 1)}{2} + \min\{i,j\} + 1 \tag{4.37}$$

Unit functions for a backward process is:

$$u_{ij}^{(k)} = \begin{cases} \min_{0 \le l \le N-1} \{s_{ijl}^{(k)}\} & \text{when } k = b(i,j) \\ u_{ij}^{(k-1)} & \text{otherwise} \end{cases} \tag{4.38}$$

where

$$b(i,j) = \frac{(\max\{N-i, N-j\}-1)(\max\{N-i, N-j\}-2)}{2} + \min\{N-i, N-j\} \quad (4.39)$$

The network updates two units at a time. To obtain all the shortest paths, the units of the network have to be updated in a specific order for $N(N-1)$ (one forward and one backward process) or $\frac{3}{2}N(N-1)$ iterations (three forward or three backward processes).

4.5.3 Dantzig's and modified Dantzig's Algorithms

Size of the inference network for DANTZIG is $N \times N$. The unit function for DANTZIG is

$$u_{ij}^{(2k)} = \begin{cases} \min\{u_{ij}^{(2k-1)}, \min\limits_{0 \le l \le k-1}\{s_{ijl}^{(2k)}\}\} & j < i = k \text{ or } i < j = k \\ u_{ij}^{(2k-1)} & \text{otherwise} \end{cases} \quad (4.40)$$

$$u_{ij}^{(2k+1)} = \begin{cases} \min\{u_{ij}^{(2k)}, s_{ijk}^{(2k+1)}\} & i < k \text{ and } j < k \\ \min\{0, \min\limits_{0 \le l \le k-1}\{s_{ijl}^{(2k+1)}\}\} & i = j = k \\ u_{ij}^{(2k)} & \text{otherwise} \end{cases} \quad (4.41)$$

where $0 \le k \le N-1$, $u_{ij}^{(-1)} = d_{ij}$, and $s_{ijl}^{(r)} = u_{il}^{(r-1)} + u_{lj}^{(r-1)}$ $\forall\, 0 \le r < 2N$. In the $2k$-th iteration, $2k$ units are updated, k sites at each unit are involved in the minimisation; in the $(2k+1)$-th iteration, a total number of k^2+1 units with a total of $k(k+1)$ sites are minimized. The total number of synchronous iterations required is $2N$.

Corresponding to the $2k$-th iteration in DANTZIG, the inference network implementation for MDANTZIG requires k iterations. In each of these iterations, $2k$ units are updated; and at each of these units, only one site is activated. So MDANTZIG requires a total number of $\frac{1}{2}N(N+1)$ iterations.

4.6 Comparison with Established Algorithms

4.6.1 Parallel Processing with Optimal Hardware

A variety of algorithms have been proposed for the shortest path problem, most of which are initially designed for sequential computation. It is interesting to observe that in many cases, an efficient sequential algorithm does not naturally lead to an efficient one in parallel. In this section, we will compare the completion time for various algorithms under the assumption that each algorithm can be executed on the hardware that optimizes its performance. In the next subsection, we will compare the efficiency of some algorithms on a specific hardware — the Connection Machine. It is assumed that our goal is to find *all* the shortest paths between any two nodes.

For single pair algorithms, such as DIJKSTRA and A*, $N(N-1)$ processors can be used. Each processor works for one shortest path between two distinct nodes. Since they are initially designed for finding the shortest path between a single pair of nodes, no inter-processor communication is needed. The completion time for the all-pair algorithm is determined by the processor which takes the longest time to find the one-pair solution.

Single-source algorithms, for example YEN, SPIRA, and MOFFAT, are designed to find the shortest paths from one specific node to all other nodes. N processors can be used to find *all* the shortest paths. There is no interprocessor communication among the N processors. The completion time is determined by the processor which has the heaviest work load.

All-pair algorithms are designed to find all shortest paths simultaneously. N^2 processors can be used. Rich communication between the processors is necessary. For FLOYD, MATRIX, RMATRIX, DANTZIG, MDANTZIG, in each iteration, each processor either performs an addition and a comparison or does nothing. The completion times is determined by the number of synchronized iterations they require.

Systolic algorithms, for example ROTE and ROBERT, are to be executed on their own systolic structures. Both the number of processors required and the number of time units required are determined by the systolic array.

The synchronous inference network algorithm requires $N(N-1)$ processors. Each processor has $2(N-2)$ outgoing and $2(N-2)$ incoming links, and performs the unit function and the simultaneous site functions. At most $\lceil \log_2(N-1) \rceil$ iterations are required.

Table 4.3 shows the numbers of processors required, number of links for each processor, and theoretical completion times for various algorithms. t_o is used to stand for the time to perform a basic operation (addition or comparison), and t_c is the time for a processor to send or receive a basic datum from another processor. Under the assumption that each algorithm can have its optimal hardware, t_o and t_c are the same for all the algorithms and are invariant with problem size N. The inference network algorithm has the shortest completion time since it requires the least number of synchronous iterations.

The inference network can also solve the shortest path problem in continuous-time. The neural network approach [19], the Hopfield network approach [83], and the connectionist approach [25] all use gradient descent in decreasing their network energy, therefore do not guarantee the solutions to be optimal. On the contrary, the inference network algorithm (Algorithm 3 on page 45) is derived from the simple intuitive procedure of finding the shortest paths, and it is an analog algorithm that guarantees an optimum solution.

4.6.2 Performance on Connection Machine

Now let us compare the efficiencies on a specific kind of computer — the Connection Machine. Table 4.4 shows the number of time units required by some all-pair shortest path algorithms when using $O(N^2)$ processors, assuming there are only four links on each

Algorithm	# of Proc.	Link/Proc.	Execution Time
DIJKSTRA	$N(N-1)$	0	$O(\frac{1}{2}N^2t_o)$
A*	$N(N-1)$	0	$< O(\frac{1}{2}N^2t_o)$
YEN	N	0	$\frac{3}{2}N^2t_o$
SPIRA	N	0	$N^2 \log_2 Nt_o$
MOFFAT	N	0	N^2t_o
MATRIX	N^2	$4N$	$\lceil log_2 N \rceil (2Nt_o + t_c)$
RMATRIX	N^2	$4N$	$\frac{\alpha}{2}N^2(2Nt_o + t_c)$
FLOYD	N^2	$4N$	$N(2t_o + t_c)$
DANTZIG	N^2	$4N$	$2N(2t_o + t_c)$
MDANTZIG	N^2	$4N$	$< 2N(\frac{N}{2}t_o + t_c)$
ROTE	$(N+1)^2$	6	$(7N-2)(2t_o + t_c)$
ROBERT	$N(N+1)$	4	$(5N-2)(2t_o + t_c)$
Algorithm 1 (page 44)	$N(N-1)$	$4(N-2)$	$< \lceil log_2 N \rceil (2t_o + t_c)$

Table 4.3: Execution time for shortest path algorithms executed on their optimal hardware configuration. α in RMATRIX is 2 or 3.

processor. Each *time unit* is approximately the time to perform one addition and one minimisation and to propagate two data.

Algorithm	ROBERT	INFERENCE-CM	MATRIX	FLOYD	DANTZIG
time units	$5N-2$	$< N \log_2(N-1)$	N^2	N^2	N^2

Table 4.4: The number of time units required to obtain the shortest paths when implemented on the Connection Machine.

ROBERT, FLOYD and DANTZIG algorithms consider one specific node at each iteration, hence their processor operations are time-dependent. INFERENCE-CM (mapping of INFERENCE on the Connection Machine, refer Appendix D.1) and the MATRIX share the same kind of time-independent PE operations. All but ROBERT occupies a square processor array on the Connection Machine. ROBERT occupies a rectangular geometry which is twice as large as the other's square array. Considering all these factors and the

number of time units required, the most efficient algorithms should be INFERENCE-CM or ROBERT. The performance of INFERENCE-CM, in comparison with ROBERT, is given in Table 4.5. For a randomly generated distance matrix and N up to 512, INFERENCE-CM gives shorter CM busy times except for $N = 4$. The worst-case CM busy times for the inference network (iterate exactly $\lceil \log_2(N-1) \rceil$ times), are close to but still shorter than that of ROBERT.

Within the problem size we tested, INFERENCE-CM outperforms ROBERT. There are three reasons for this: processor operation for INFERENCE-CM is time-independent; INFERENCE-CM has a smaller virtual processor ratio; and INFERENCE-CM usually takes less than $\lceil N \log_2(N-1) \rceil$ time units. When N increases further, however, according to the theoretical prediction, ROBERT will eventually be faster than INFERENCE-CM.

N	4	8	16	32	64	128	256	512
random	0.0874	0.1131	0.1759	0.2958	0.7012	1.8825	8.2125	37.1443
worst	0.0874	0.1181	0.1759	0.3248	0.7971	2.1143	8.8626	46.9628
Robert	0.0828	0.1688	0.3753	0.8965	2.3678	8.0302	36.9367	209.620

Table 4.5: CM busy time for INFERENCE-CM (random), the worst case of INFERENCE-CM (worst), and Robert and Trystram's algorithm (Robert) to solve the shortest path problem given random distance graphs.

4.7 Discussion

Although many algorithms have been proposed for the shortest path problem, the inference network algorithm has shown a special feature — it works in synchronous discrete-time, asynchronous discrete-time, and continuous-time domains.

In synchronous discrete-time, the algorithm was shown to take the least number of iterations and was mapped to the Connection Machine efficiently. For a large systolic

array, it is a technical headache to synchronize all the processing elements. The inference network algorithm was proved to converge in asynchronous updating as well as in synchronous mode. Therefore, global synchronization was not necessary. Simulation of the asynchronous algorithm was conducted and convergence was guaranteed as long as each unit has a chance to update.

The inference network was also proved to work in continuous-time using an analog circuit. The analog parallel implementation was shown to solve the N-node problem in a speed determined by the time constants of the first order system. A physical continuous time inference network for a 10-city shortest path problem is being built.

Although neural network algorithms can also solve the problem using analog techniques [83, 19], they do not guarantee an optimal solution, due to the use of gradient descent. The inference network algorithm was proved to produce the optimal solution in both discrete and continuous time domains for all problem sizes.

Chapter 5

Inference Network for the Transitive Closure Problem

This chapter outlines the application of the discrete and continuous-time inference network to the transitive closure problem. Simple binary *and* operation is required at each site; and each unit performs multi-input *or* operations. The time complexity of the algorithm was shown to be bounded by $\log_2 N$. The algorithm was proved to converge in continuous-time domain as well in discrete-time. It is demonstrated that the convergence is independent of the problem size, and the optimal solution is guaranteed. Theoretical analysis and numerical simulation of the circuit implementation was conducted. The algorithm was shown efficient and easy to implement. Using basic logic gates, the solution was obtained in nanoseconds range.

5.1 The Problem

The transitive closure of a graph indicates whether two nodes in the graph are connected by a path. A graph G has a set of nodes V, and a set of arcs E connecting some of these nodes. A directed graph G can be represented by its connection matrix $\mathbf{A} = [a_{ij}]$ in which $a_{ij} = 1$ or 0 means there is or is not an arc from node i to node j. The graph G^*, which has the same node set as G, but has an arc from v to w if and only if there is a directed path from v to w in G, is called the transitive closure of G. G^* can be similarly represented by its connection matrix $\mathbf{A}^*$.

Many systolic algorithms were proposed for the problem [71, 70, 52, 40, 39, 46, 65] which require about N^2 processors synchronized with a clock to perform operations in

pre-defined orders. Their time complexities are usually around $5N$ to $7N$ time units. Tamassia [82] proposed an algorithm to solve the problem in $O(\log N)$ time using $\frac{N}{\log N}$ processors in exclusive-read exclusive-write PRAM model. Toptsis [84] introduces a parallel algorithm for highly scalable multiprocessors. If a perfect hashing function is available, the speedup over sequential algorithm is proportional to the number of processors. Wang [88] proposes a constant time algorithm which solves the problem on N^6 processors.

Transitive closure is a basic problem in artificial intelligence, and have many applications in database management systems, redundancy elimination, image labeling, and graph theory, etc.

The inference network algorithm for the problem can be executed in both discrete and continuous time domain. In its discrete time version, the algorithm is straightforward and the time complexity is bounded by $\log_2 N$. Compared with other discrete-time transitive closure algorithms, the inference network one is not superior. However, the power of the algorithm is in its continuous-time implementation. For the continuous-time version, the network is composed of simple logic gates, and its completion time is in nanoseconds range. The network can be implemented using NAND gates. Simulation of this NAND-gate circuit is given. The network is easy to assemble; clock and global synchronization are not needed.

5.2 Unit and Site Functions

The inference network for the transitive closure of a directed graph has $N(N-1)$ units, each unit has $N-2$ sites. All unit and site values fall into range $[0,1]$. $u(i,j) = 1$ indicates that a path from node i to node j has been found; $u(i,j) = 0$ otherwise. Similarly, $s(i,j,l) = 1$ (or $s(i,j,l) = 0$) shows a path from node i to node l then to node j has (or has not) been found. An initial unit value is 1 if the two corresponding nodes

are connected by a direct arc in the graph. All link weights are 1. The sites and units can be updated asynchronously, synchronously with a clock, or in continuous-time. In the discrete version, the site and unit functions can be as simple as the following: each site performs binary *and*

$$s^{(k)}(i,j,l) = u^{(k-1)}(i,l) \wedge u^{(k-1)}(l,j) \quad \forall 0 \leq i \neq j \neq l < N \tag{5.42}$$

and a unit performs $(N-1)$-input *or*

$$u^{(k)}(i,j) = u^{(k-1)}(i,j) \vee (\bigvee_{\forall l} s^{(k)}(i,j,l)) \tag{5.43}$$

In the continuous time version, the following site and unit functions may be used (refer Figure 5.12):

$$s_t(i,j,l) = \begin{cases} 1 & u_t(i,l) > v_h, u_t(l,j) > v_h, t \geq t_0 + \tau_s \\ \frac{t-t_0}{\tau_s} & u_t(i,l) > v_h, u_t(l,j) > v_h, t < t_0 + \tau_s \\ 0 & u_t(i,l) \leq v_h \text{ or } u_t(l,j) \leq v_h \end{cases} \tag{5.44}$$

where $0 < v_h < 1$ and t_0 is the time when both $u_t(i,l)$ and $u_t(l,j)$ just exceed v_h.

$$u_t(i,j) = \begin{cases} 1 & u_{t-\delta t}(i,j) = 1 \text{ or } t \geq t_0 + \tau_u \\ \frac{t-t_0}{\tau_u} & \exists l \;\; s_t(i,j,l) > v_h, t < t_0 + \tau_u \\ 0 & \forall l \;\; s_t(i,j,l) \leq v_h \end{cases} \tag{5.45}$$

where t_0 is the time when the first site value exceeds v_h.

5.3 Convergence Consideration

Since both unit and site functions are non-decreasing, and their values are upper-bounded, the network must converge eventually. For the discrete version, if we define an $N \times N$ matrix $\mathbf{U}^{(k)} = [u^{(k)}(i,j)]$ and $u^{(k)}(i,i) \equiv 1 \; \forall i,k$, Eq. (5.42-5.43) can be effectively expressed by

$$\mathbf{U}^{(k)} = \mathbf{U}^{(k-1)} \odot \mathbf{U}^{(k-1)} = [\bigvee_{\forall l} (u^{(k-1)}(i,l) \wedge u^{(k-1)}(l,j))] \tag{5.46}$$

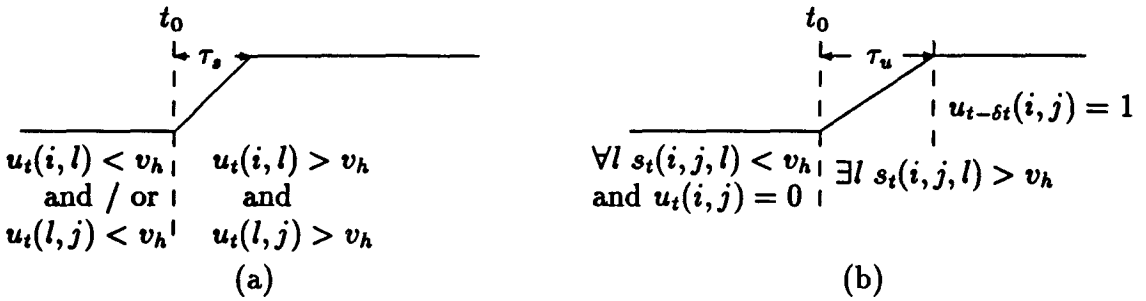


Figure 5.12: (a) Site and (b) unit functions for the transitive closure problem.

which is equivalent to

$$u^{(k)}(i,j) = u^{(k-1)}(i,j) \bigvee_{\forall l \neq i \neq j} (u^{(k-1)}(i,l) \wedge u^{(k-1)}(l,j))$$

If $i \rightarrow j$ is an arc in the original graph G, define $u^{(0)}(i,j) = 1$. After the first iteration, if $u^{(1)}(i,j) = 1$, there is a path from i to j composed of one or two arcs in G. In the second iteration, all directed paths containing at most 4 arcs are found and the corresponding unit values are updated to 1. Repeating this procedure p times, all the paths containing at most 2^p arcs can be found. In other words, a path containing q arcs can be found in at most $\lceil \log_2(q) \rceil$ iterations. Since the longest open path in an N-node graph contains at most $N-1$ arcs, all the paths between any two nodes can be found in at most n iterations, where

$$n = \lceil \log_2(N-1) \rceil \tag{5.47}$$

and the corresponding unit values in $\mathbf{U}^{(n)}$ contain the solution — $\mathbf{U}^{(n)}$ equals to the connection matrix of transitive closure G^*. Therefore, the number of iterations required for a synchronous discrete network to arrive at the solution is upper bounded by Eq.(5.47).

Eq.(5.44-5.45) for the continuous-time version are similar to Eq.(5.42-5.43), except that the transition periods from value 0 to 1 are taken into account. The convergence rate is now determined by the turn-on time τ_s and τ_u as well. Since τ_s and τ_u for different

sites and units can be different, unit values do not change simultaneously as they do in the discrete version. To measure the 'distance' between a network state and the final stable state, we can define the following network energy $E(t)$:

$$E(t) = \sum_i \sum_{j \neq i} (1 - \max_{\forall l \neq i \neq j} \{u_t(i,j), u_t(i,l) * u_t(l,j)\})^2 \tag{5.48}$$

The solution corresponds to the global minimum state, where $E_{\min}(t)$ equals to the number of 0's in the connection matrix of the transitive closure G^*. Since $0 \leq u_t(i,j), u_t(i,l) * u_t(l,j) \leq 1$

$$E(t) \leq \sum_i \sum_{j \neq i} (1 - u_t(i,j))^2 \tag{5.49}$$

since

$$\frac{dE(t)}{dt} = \sum_i \sum_{j \neq i} \frac{\partial E(t)}{\partial u_t(i,j)} \frac{du_t(i,j)}{dt} \tag{5.50}$$

$$\begin{aligned} \frac{\partial E(t)}{\partial u_t(i,j)} &\leq \frac{\partial}{\partial u_t(i,j)} \sum_i \sum_{j \neq i} (1 - u_t(i,j))^2 \\ &= -2(1 - u_t(i,j)) \leq 0 \end{aligned} \tag{5.51}$$

hence

$$\frac{dE(t)}{dt} \leq \sum_i \sum_j -2(1 - u_t(i,j)) \frac{du_t(i,j)}{dt} \tag{5.52}$$

Eq.(5.52) shows that the network energy does not increase as long as $\frac{du_t(i,j)}{dt} \geq 0$ — unit values do not decrease, which is guaranteed by Eq.(5.44-5.45).

Now we will see that the network has to arrive at the correct solution after Eq.(5.44-5.45) have been applied. At the minimum state of the network,

$$\frac{dE(t)}{dt} = 0 \tag{5.53}$$

holds. In Eq.(5.50), since $\frac{\partial E(t)}{\partial u_t(i,j)} \frac{du_t(i,j)}{dt} \leq 0$ for all i and j, Eq.(5.53) is equivalent to

$$\forall i,j \quad \frac{\partial E(t)}{\partial u_t(i,j)} = 0 \text{ or } \frac{du_t(i,j)}{dt} = 0$$

that is, from Eq.(5.51),

$$\forall i,j \;\; u_t(i,j) = 1 \text{ or } \frac{du_t(i,j)}{dt} = 0$$

At this stable state, a unit either keeps its original 0 value or has become 1. All the site values are also either 0 or 1. Since Eq.(5.44-5.45) were applied before the stable state was achieved, the following statements are true for the stable state ($t = \infty$):

1. $u_0(i,j) = 1 \;\Rightarrow\; u_\infty(i,j) = 1$;

2. $u_\infty(i,l) = 1 \;\wedge\; u_\infty(l,j) = 1 \;\Rightarrow\; u_\infty(i,j) = 1$;

3. $u_0(i,j) = 0 \;\wedge\; \forall l,t \; (u_t(i,l) < v_h \text{ or } u_t(l,j) < v_h) \;\Rightarrow\; u_\infty(i,j) = 0.$

From the first two statements, by mathematical induction, we know that for all i and j with a directed path in between, $u_\infty(i,j) = 1$; from the last statement, we can conclude that if there is no directed path from i to j, $u_\infty(i,j) = 0$, or in other words, for all i and j with $u_\infty(i,j) = 1$, there is a path from i to j. According to the definition, $u_\infty(i,j)$ gives the solution to the transitive closure problem. The above analysis also shows that unit function can be any non-decreasing monotonic functions.

A computational framework for solving the transitive closure problem can be formulated as follows:

1. Construct the $N(N-1)$-unit continuous-time inference network, where the dynamic behavior of each individual unit and site is governed by Eq.(5.44-5.45).

2. The network has a complex but regular interconnection structure, where each unit sends its output to a set of $2 * (N-2)$ units, and receives at the same time the outputs from this same set of units at its $N-2$ sites.

3. Initialize the network units with $u_0(i,j) = 1$ if there is a direct arc from i to j in graph G. Otherwise, set $u_0(i,j) = 0$.

4. The network will converge to a global minimum at a rate depending on the transition time of each unit. The converged output $u_\infty(i,j)$ of unit (i,j) corresponds to the element a^*_{ij} in the connection matrix $\mathbf{A}^*$.

5.4 Implementation Using Logical Circuits

Although the Connection Machine can be used to implement discrete-time inference network algorithms, a simple special purpose inference network can be built for the transitive closure problem, with NAND gates as its major components. From Eq.(5.42-5.43), we can see a unit performs

$$u(i,j) = u(i,j) \vee (\bigvee_{\forall l} s(i,j,l)) = \overline{\overline{u(i,j)} \wedge (\bigwedge_{\forall l} \overline{s(i,j,l)})}$$

where

$$\overline{s(i,j,l)} = \overline{u^{(k-1)}(i,l) \wedge u^{(k-1)}(l,j)}$$

A 2-input NAND gate, for example the kind in a 74LS00, can be a site, refer Figure 5.13(a). It has a simplified mathematical model of Eq.(5.54) with turn-off time τ_s, see Figure 5.14(a). An $(N-1)$-input NAND gate with a mathematical model of Eq.(5.55) can be used for each unit, see Figure 5.13(b) and 5.14(b). For a small-size problem with $N < 15$, a 13-input NAND gate 74S133 can be used as a unit. If N is large, a unit may consist of several m-input NAND gates, based on

$$\begin{aligned} u &= s_1 \vee s_2 \vee \cdots \vee s_M = \overline{\overline{s}_1 \wedge \cdots \wedge \overline{s}_m} \vee \overline{\overline{s}_{m+1} \wedge \cdots \wedge \overline{s}_{2m}} \vee \cdots \vee \overline{\cdots \wedge \overline{s}_M} \\ &= \overline{\overline{\overline{s}_1 \wedge \cdots \wedge \overline{s}_m}\ \overline{\overline{s}_{m+1} \wedge \cdots \wedge \overline{s}_{2m}}\ \cdots\ \overline{\cdots \wedge \overline{s}_M}} \end{aligned}$$

Since each site has two inputs coming from two distinct units, the fan-in capability of the gates is sufficient. However, the output of a unit has to go to $2(N-1)$ sites, a regular NAND gate may not have large enough fan-out for a unit. If this is the case, additional driving gates may be used at the output end of units.

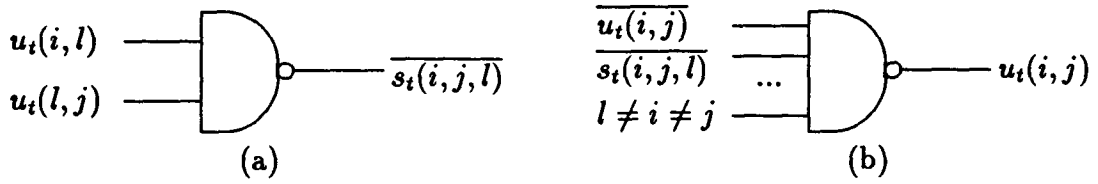


Figure 5.13: Essential components for a unit.

The following equations describe the state of the inference network. Site gates perform

$$\overline{s_t(i,j,l)} = \begin{cases} 0 & u_t(i,l) > v_h, u_t(l,j) > v_h, t \geq t_0 + \tau_s \\ 1 - \frac{t-t_0}{\tau_s} & u_t(i,l) > v_h, u_t(l,j) > v_h, t < t_0 + \tau_s \\ 1 & u_t(i,l) \leq v_h \text{ or } u_t(l,j) \leq v_h \end{cases} \tag{5.54}$$

where t_0 is the time when both $u_t(i,l)$ and $u_t(l,j)$ just exceed v_h — the minimum value to be considered as 'high', and τ_s is the turn-off time of the NAND gate. Each unit is governed by

$$u_t(i,j) = \begin{cases} 1 & u_{t-\delta t}(i,j) = 1 \text{ or } t \geq t_0 + \tau_u \\ \frac{t-t_0}{\tau_u} & \exists l \ \overline{s_t(i,j,l)} < v_l, t < t_0 + \tau_u \\ 0 & \forall l \ \overline{s_t(i,j,l)} \geq v_l \end{cases} \tag{5.55}$$

where t_0 is the time when the first input value goes below v_l — the maximum value to be considered as 'low', and τ_u is the total effective turn-on time of the entire unit. Eq.(5.54-5.55) are very similar to Eq.(5.44-5.45) except sites produce $\overline{s_t(i,j,l)}$ instead of $s_t(i,j,l)$, which enables the use of popular NAND gates. With a procedure similar to that in the last section, it can be proved that the solution given by this implementation is also correct. The transition time for each unit and site may be different. The only requirement to guarantee a valid solution is that during the transition period, the output of a gate changes monotonically.

In order to give an idea of the completion time, let us assume the transition time for each site and unit are τ_s and τ_u respectively. A site gate takes $(1 - v_l)\tau_s$ time to turn

off, and a unit gate takes $v_h\tau_u$ time to turn on, refer Figure 5.14. If τ_s or τ_u are identical for all sites and units, the total completion time for the network to get the solution is about $\lceil \log_2(N-1) \rceil ((1-v_l)\tau_s + v_h\tau_u)$. Use the inference network described above for an $N = 10$ problem, with 180 74LS00 (a total of 720 2-input NAND gates), 90 inverters in 74LS02, and 90 74S133 (13-input NAND gates, only 9 inputs of each gate are used), the network is expected to arrive at the solution within 50ns. Here we use the typical transition time for 74LS00 and 74S133 as 9.5ns and 3ns respectively, and $v_h = 0.6$ and $v_l = 0.2$ respectively.

This implementation overcomes a major limitations of systolic approaches — global synchronization — by updating the units and sites in continuous-time domain. Compared with the processing elements in a systolic array, each unit in the inference network only consists of a number of NAND gates. Physical implementation of the continuous-time inference network is much simpler than a systolic array. For comparison, a systolic algorithm running on the Connection Machine takes about 10 seconds CM elapsed time, including 0.2 second CM busy time for an $N = 10$ problem. The potential problem with this implementation, limited fan-out of a gate, may be overcome by using additional driving gates.

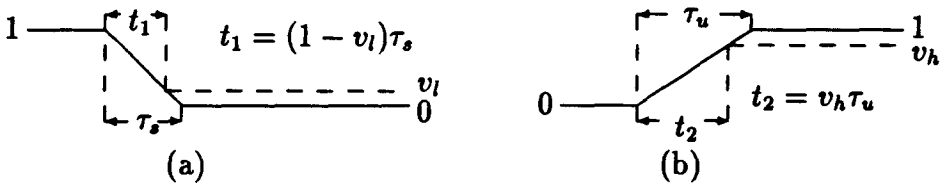


Figure 5.14: Site and unit time delay.

Besides the existence of a path between two nodes, if we are also interested in the path itself, a path indicator can be attached to each site. Figure 5.15 shows the circuit for such a path indicator. Under the condition that none of the site or unit value decreases (the

sufficient condition for convergence), an indicator at a site will turn on only when the site value becomes 'high' while the unit value is still 'low'. Therefore, when the network is stabilized, if the value of unit (i,j) is 'low', there is no path between the two nodes. If unit (i,j) has 'high' value, and none of the path indicator is on, there is a direct path from i to j. If indicator of site l at unit (i,j) is on and unit (i,j) has 'high' value, there is a path from node i to node j through node l. The path from i to l and from l to j can be derived recursively based on the path indicator at the corresponding units. Table 5.6 shows how the path indicator works.

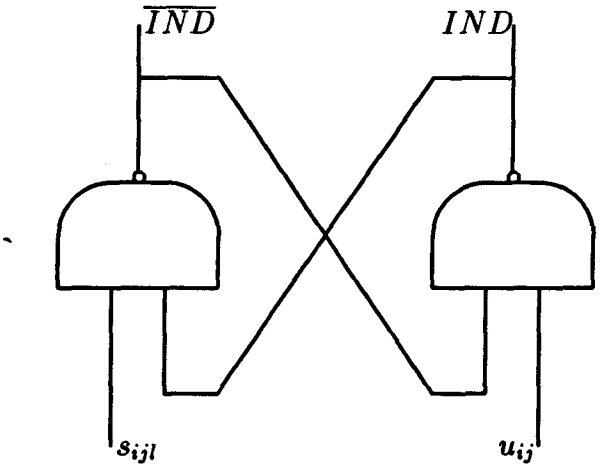


Figure 5.15: A path indicator for the transitive closure problem.

$t-\delta$				$t+\delta$				note
s_{ijl}	u_{ij}	IND	$\overline{IND}$	s_{ijl}	u_{ij}	IND	$\overline{IND}$	
0	0	1	1	↑	0	1	0	indicator turns on
0	0	1	1	0	↑	0	1	indicator turns off
0	1	0	1	↑	1	0	1	indicator keeps off
1	0	1	0	1	↑	1	0	indicator keeps on

Table 5.6: States of unit, site and path indicator.

5.5 An Example

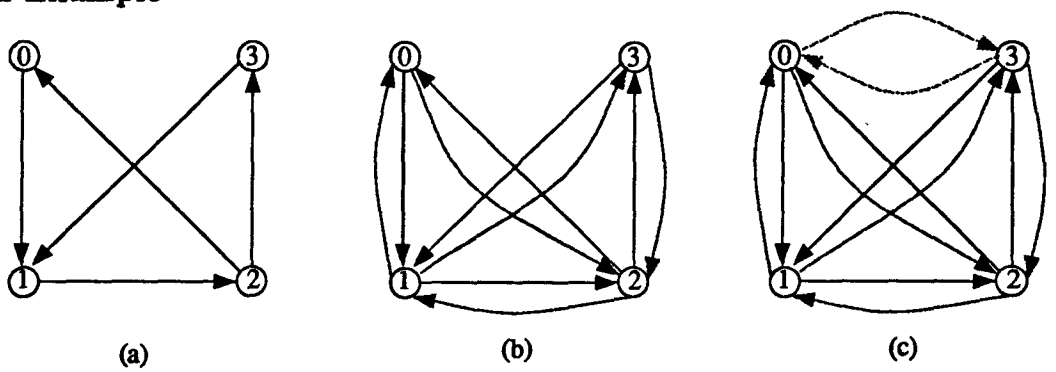


Figure 5.16: An example of a 4-node graph (a), the intermediate result (b) and the transitive closure (c).

To illustrate the algorithm and implementation of the network, let us go through the following $N = 4$ example. The connection matrix of a 4-node graph, Figure 5.16(a), is

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \tag{5.56}$$

An $N = 4$ problem requires 12 units, each with 2 sites. A unit is a 3-input NAND gate and a site is a 2-input NAND gate. Figure 5.17 shows the complete circuit. Initial values are applied to the inputs of the inverters at $t = 0$. It takes the transition time of an inverter and a 3-input NAND gate for 5 unit outputs become 'high' — corresponding to the 5 arcs in Figure 5.16(a). These high unit values are then sent to other units. 5 other unit outputs become high after approximately the transition time of two NAND gates. The paths found so far are show in Figure 5.16(b). After another transition period of two NAND gates, the last 2 unit outputs become high, and the transitive closure, Figure 5.16(c), is obtained. Figure 5.18 shows the unit outputs as functions of time. It

takes about 10 nanoseconds to obtain the final result.

In Fig. 5.18, unit outputs go up to 'high' in the sequence predicted by Fig. 5.16. However, due to the capacitance and inductance distributed among circuit elements, the waveforms are not as smooth as an ideal output from a digital system. No over-shooting of unit outputs is observed. Down-shooting of less than 0.25V occurs at some units which does not affect the overall updating of the network. Smoother waveforms can be achieved by optimizing the layout of the units within the chip and wiring of elements in each unit. Additional built-in diodes at the output of units can regulate the down-shooting of waveforms to be within an acceptable range.

5.6 Discussion

The inference network was shown to provide a platform to solve the transitive closure problem. The discrete version algorithm was shown to produce a solution within at most $\lceil \log(N-1) \rceil$ synchronous iterations. The analog network was composed of logic gates which operated in non-synchronized fashion. The completion time for the analog implementation was shown in nanoseconds range. A potential problem for a large size network is the fan-out limitation of the gates. Additional driving gates may be used to increase the fan-out ability.

Since a logic gate is formed by some analog circuit, and the logic circuit solution to the transitive closure problem does not require a clock, analog circuits (or analog simple neurons) can be used to substitute the logic gates for sites and units. Therefore, the circuit for the transitive closure problem becomes indeed an analog circuit (or analog neuron network).

The transitive closure problem can also be solved using systolic array or constant-time algorithms. They are synchronous approaches: the calculation at each processing

element has to follow some predesigned orders which are derived through tedious analysis of the data dependencies among processors. The inference network structure is obtained from simple optimization procedures, therefore, network algorithm for transitive closure can be derived directly from the problem formulation.

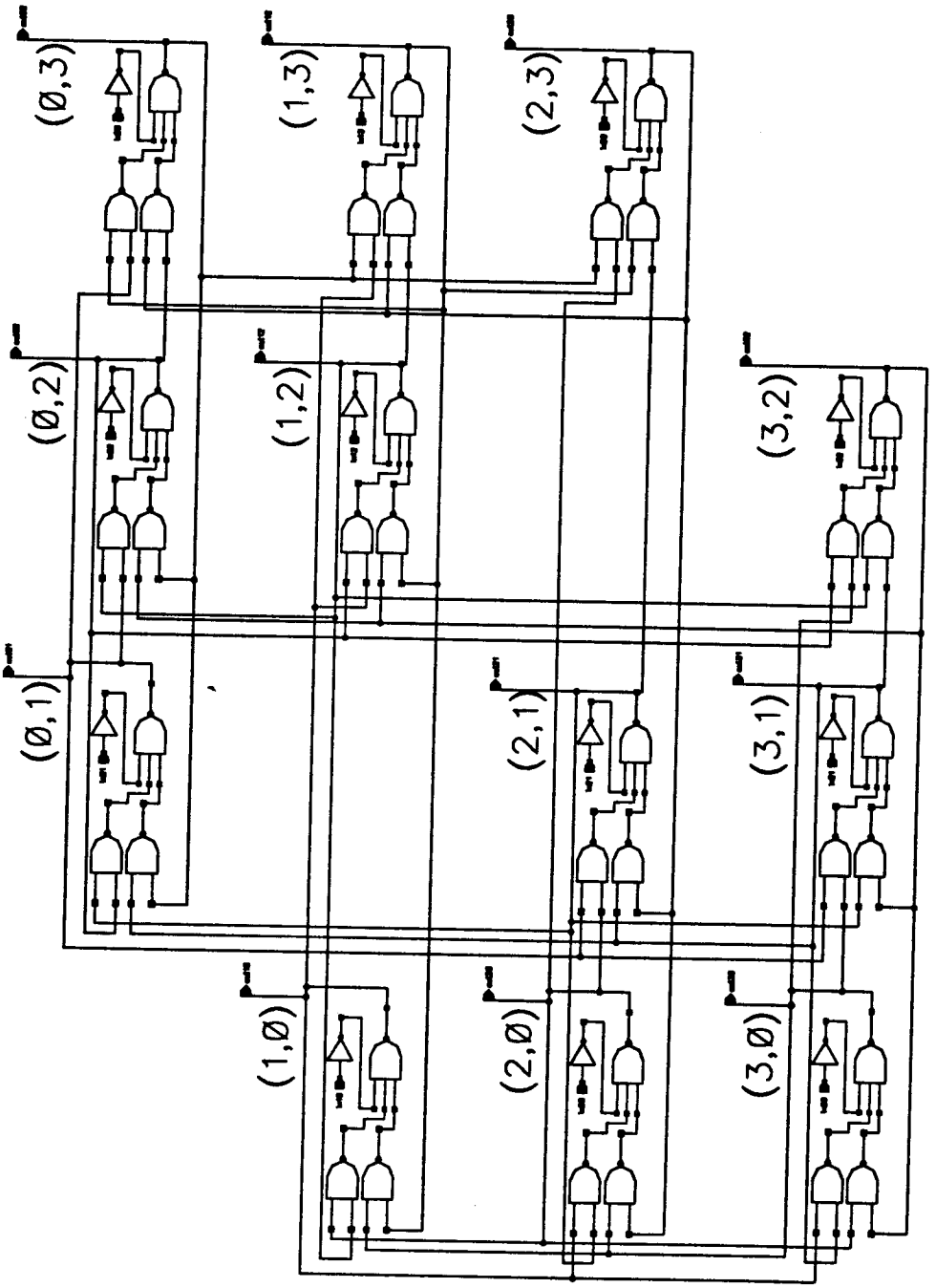


Figure 5.17: Circuit for an $N = 4$ transitive closure problem.

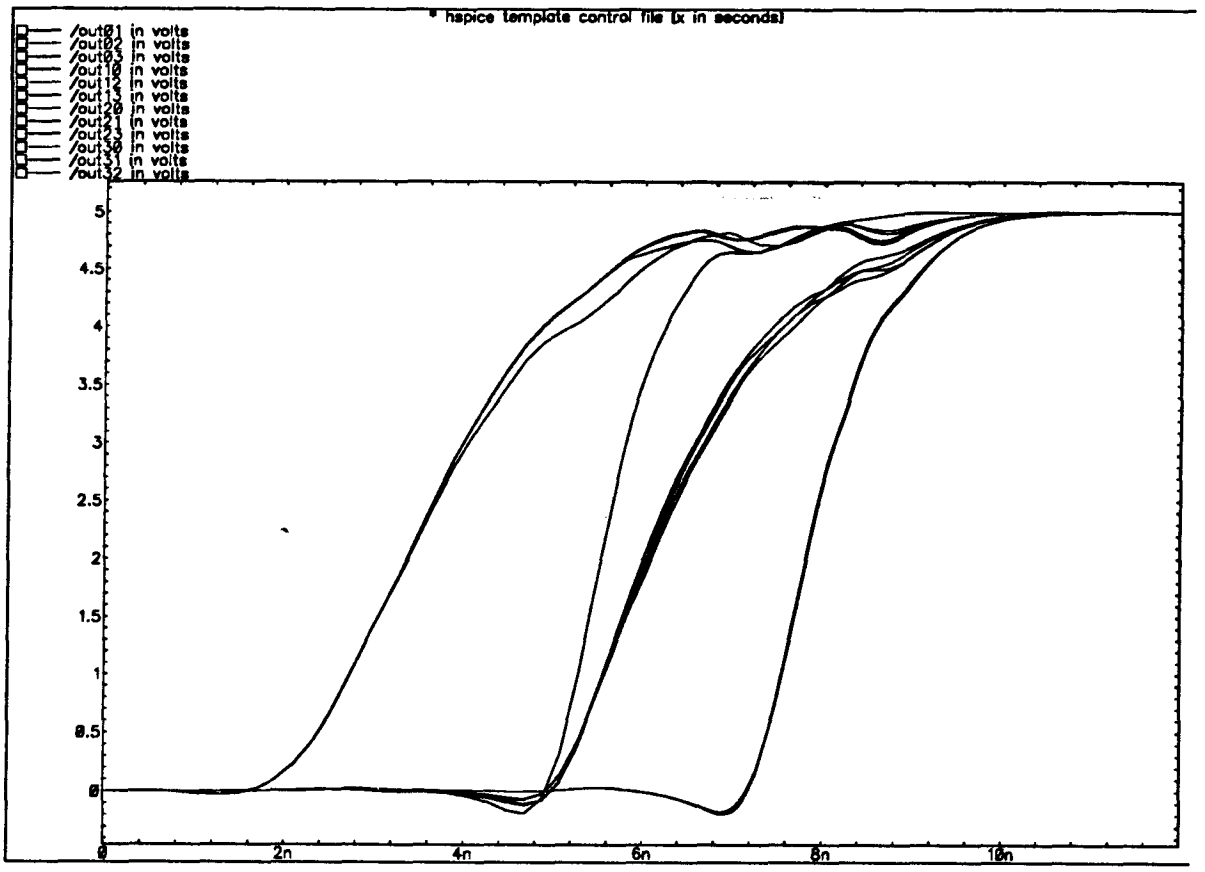


Figure 5.18: Hspice simulation of the inference network for a 4-node transitive closure problem.

Chapter 6

Inference Network for the Assignment Problems

This chapter discusses the behavior of the inference network when it is used in optimization through gradient descent of an energy function — an approach used by many neural network algorithms. The application of the inference network to the assignment problem was discussed. The dynamics of the network was shown similar to that of the Hopfield net for the traveling salesman problem (TSP). The inference network has less links than the Hopfield net, however, the eigenvectors for the two networks were proved similar. Due to the nature of the assignment problem and the use of a decreasing weight for the objective function, the network was shown to converge for a much larger range of problem size than the Hopfield net for the TSP. For all the test cases used with size up to 128, it was demonstrated that the network converges to a near-optimal valid solution.

6.1 The Assignment Problem

6.1.1 The Problem

The assignment problem is a classical combinatorial optimization problem. Consider N objects to be assigned to N persons. Both persons and objects are denoted by an integer $0, \cdots, N-1$. Given r_{ij} $(0 \leq i, j < N)$ as the benefit of assigning object j to person i, the objective is to assign each person a distinct object so that the resulting one-to-one correspondence maximizes the total benefit.

The Hungarian method, proposed by Kuhn in 1955 [38], is a traditional solution to

the problem and a special case of linear programming [66]. It requires $O(N^4)$ operations. In 1979, Bertsekas [5] proposed the auction algorithm which guarantees a global optimal solution with a worst-case sequential complexity of $O(NM\log(NM))$, where M is the largest *integer* benefit. Wein and Zenios [89] developed a hybrid implementation of the auction algorithm on the Connection Machine to solve large assignment problems. Kosowsky and Yuille [37], motivated by statistical physics, solve the problem using a continuous dynamical system. Kim [34] uses a modified Hopfield's model to solve the assignment problem, and claims achieving better stability and accuracy. (The paper is in Korean.)

The assignment problem has many applications. In VLSI layout, it mostly occur in semi-customer design styles such as gate-array or standard-cell design. In gate assignment, the slots are library cells and the circuit elements are gates. Supposed each cell can accommodate exactly one gate and each gate has a set of legal locations to preserve the function of the circuit, the assignment is to minimize cost. Similarly, optimal assignment is also used for determining pin locations in global and area routing. Refer [51] for details.

6.1.2 Optimization Formulation

The assignment problem can be formulated into a constrained optimization problem. A valid solution satisfies the constraint of one object for each person. The valid solution which maximizes the total benefit is defined as an optimal solution. Let us define $0 \leq u_{ij} \leq 1$ for all $0 \leq i, j < N$. $u_{ij} = 1$ means person i is assigned to object j. $u_{ij} = 0$ means person i does not get object j. Based on these definitions, $\sum_j \sum_{l \neq j} u_{ij}u_{il} = 0$ guarantees that person i does not get two distinct objects. Similarly, $\sum_j \sum_{l \neq j} u_{ji}u_{li} = 0$ guarantees that object i is not assigned to two distinct persons. $\sum_i u_{ij} = 1$ and $\sum_j u_{ij} = 1$ ensure that only one person gets object j, and person i only gets one object. If u_{ij}'s are put

into an $N \times N$ array, then a valid solution is an array with exactly one 1 in each row and column, all the rest elements are 0. The objective function to be minimized is:

$$E = A\sum_i\sum_j\sum_{l\neq j} u_{ij}u_{il} + B\sum_i\sum_j\sum_{l\neq j} u_{ji}u_{li} + C\sum_i[(\sum_j u_{ij} - 1)^2 + (\sum_j u_{ji} - 1)^2] - D\sum_i\sum_j u_{ij}r_{ij} \quad (6.57)$$

where $0 \leq u_{ij} \leq 1$, $0 \leq i, j, l < N$, $A, B, C, D > 0$ and $r_{ij} \geq 0$ $\forall i, j$. In Eq.(6.57), the sum of term A, B, C gets a minimum of 0 when the constraint is satisfied. Term D is the total benefit. Due to the symmetry of the constraint, the first two terms should be equally weighed. Therefore $A = B$. A, B and C must not be zero. If A and B are zero, Eq.(6.57) may get a minimum state where all unit values are close to $\frac{1}{N}$. If C is zero, the minimum state of Eq.(6.57) is $u_{ij} = 0$ $\forall i, j$, which is not a valid solution either.

The assignment problem can have some further constraints. For example, a certain person must or must not get a certain object. If person i has to get object j, u_{ij} equals to 1 constantly, which reduces the problem dimension by one. If person i must not get object j, $u_{ij} = 0$ can be set initially and be kept unchanged for the entire calculation.

Neural network approach is also used in other optimization applications. Sriram [74] introduced a neural network solution to an NP-complete two dimensional assignment problem. Its energy function is the sum of three terms. Two of the terms represent the constraint, and the third one measures the quantity to be optimized. Constants in the energy function are determined experimentally. Wacholder [86] introduced a neural network-based optimization algorithm for the static weapon-target assignment problem. The similarity of all neural network approaches for constrained optimization is that their energy functions are composed of terms for constraints and the quantities to be optimized. However, distinct energy functions correspond to different dynamic behaviors of the network.

6.2 Unit and Site Functions for Optimization

The inference network for the assignment problem consists of N^2 units, and N sites on each unit; unit value u_{ij} has the same interpretation as in Eq.(6.57). The algorithm is based on gradient descent — to decrease the value of the objective function Eq.(6.57) monotonically. The following site and unit functions can guarantee that $\Delta E = E^{(k+1)} - E^{(k)} \leq 0$ is satisfied for small changes. The site l at unit (i,j) has a site function of

$$s_{ijl} = u_{il} + u_{lj} \tag{6.58}$$

For all units (i,j), the unit function is defined as

$$u_{ij}^{(k+1)} = f(u_{ij}^{(k)} + K_d \Delta u_{ij}^{(k)}) \tag{6.59}$$

where

$$f(x) = \begin{cases} 1 & x > 1 \\ x & 0 \leq x \leq 1 \\ 0 & x < 0 \end{cases} \tag{6.60}$$

$$\Delta u_{ij}^{(k)} = (A+B)u_{ij}^{(k)} - \sum_l ((A+C)u_{il}^{(k)} + (B+C)u_{lj}^{(k)}) + 2C + \frac{D}{2} r_{ij} \tag{6.61}$$

$$= 2Au_{ij}^{(k)} - \sum_l (A+C)s_{ijl} + 2C + \frac{D}{2} r_{ij} \tag{6.62}$$

A, B, C, D are the constants in Eq.(6.57), $K_d > 0$ is a constant determining the magnitude of the change. $B = A$ is used in Eq.(6.62). $D < 0$ is also allowed.

Summarize Eq.(6.59-6.61), it can be observed that each unit, together with its sites, act very similar to a neuron in the Hopfield net. See Figure 6.19. Each neuron in the Hopfield net is connected with all other neurons. However, each unit in the inference network is only connected with the units in the same row or column.

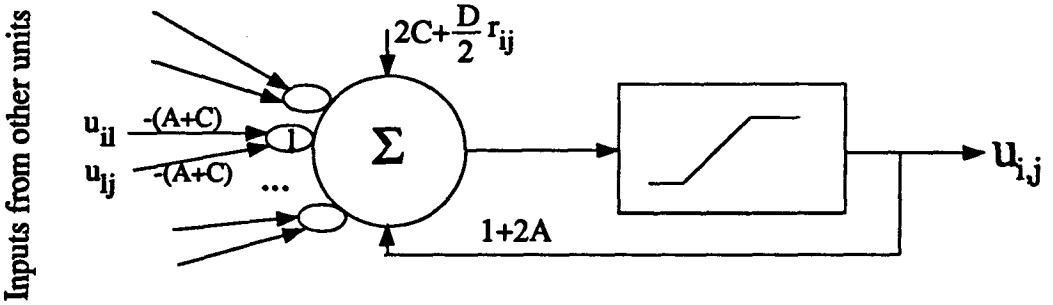


Figure 6.19: Unit and sites in the inference network act like a neuron.

Continuous-time unit and site functions for the assignment problem can be similarly defined:

$$u_{ij} = f(K_c v_{ij}) \tag{6.63}$$

where v_{ij} satisfies

$$\frac{d\, v_{ij}}{d\, t} = (A+B)u_{ij} - \sum_l ((A+C)u_{il} + (B+C)u_{lj}) + 2C + \frac{D}{2}r_{ij} \tag{6.64}$$

$$= 2Au_{ij} - \sum_l (A+C)s_{ijl} + 2C + \frac{D}{2}r_{ij} \tag{6.65}$$

and $f(x)$ and s_{ijl} are defined the same as in Eq.(6.60) and Eq.(6.58).

6.3 The Decreasing Objective Function

The value of the objective function decreases monotonically for $K_d > 0$ and any values of A, B, C, and D. According to the definition of function $f(x)$ in Eq.(6.60),

$$\begin{aligned}
&\text{if } 0 \le u_{ij}^{(k)} + K_d \Delta u_{ij}^{(k)} \le 1, \quad u_{ij}^{(k+1)} - u_{ij}^{(k)} = K_d \Delta u_{ij}^{(k)}; \\
&\text{if } u_{ij}^{(k)} + K_d \Delta u_{ij}^{(k)} > 1, \quad u_{ij}^{(k+1)} - u_{ij}^{(k)} = 1 - u_{ij}^{(k)} \ge 0 \quad \& \quad K_d \Delta u_{ij}^{(k)} > 1 - u_{ij}^{(k)} \ge 0; \\
&\text{if } u_{ij}^{(k)} + K_d \Delta u_{ij}^{(k)} < 0, \quad u_{ij}^{(k+1)} - u_{ij}^{(k)} = -u_{ij}^{(k)} \le 0 \quad \& \quad K_d \Delta u_{ij}^{(k)} < -u_{ij}^{(k)} \le 0.
\end{aligned}$$

In any of the cases,

$$(u_{ij}^{(k+1)} - u_{ij}^{(k)})K_d\Delta u_{ij}^{(k)} \geq 0 \tag{6.66}$$

holds. The derivative of the objective function Eq.(6.57) can be derived:

$$\frac{d\,E}{d\,u_{ij}} = -2\left((A+B)u_{ij} - \sum_l((A+C)u_{il} + (B+C)u_{lj}) + 2C + \frac{D}{2}r_{ij}\right) \tag{6.67}$$

When u_{ij} changes slowly, ($u_{ij}^{(k+1)} - u_{ij}^{(k)}$ is sufficiently small)

$$E^{(k+1)} - E^{(k)} = \sum_i\sum_j \frac{dE}{du_{ij}}(u_{ij}^{(k+1)} - u_{ij}^{(k)})$$

Notice that

$$\frac{d\,E}{d\,u_{ij}}K_d\Delta u_{ij}^{(k)} = -2K_d\left((A+B)u_{ij} - \sum_l((A+C)u_{il} + (B+C)u_{lj}) + 2C + \frac{D}{2}r_{ij}\right)^2 \leq 0$$

Using Eq.(6.66), we can get

$$\frac{d\,E}{d\,u_{ij}}(u_{ij}^{(k+1)} - u_{ij}^{(k)}) \leq 0 \quad \forall\, i,j$$

we thus proved that

$$E^{(k+1)} - E^{(k)} \leq 0 \tag{6.68}$$

It can also be proved that network energy decreases for continuous-time updating as well. According to the definition of function $f(x)$ in Eq.(6.60),

$$\begin{array}{llll} \text{when} & 0 \leq v_{ij} \leq 1 & u_{ij} = K_c v_{ij} & \frac{d\,u_{ij}}{d\,t} = K_c\frac{d\,v_{ij}}{d\,t} \\ \text{when} & v_{ij} > 1 & u_{ij} = 1 & \frac{d\,u_{ij}}{d\,t} = 0 \\ \text{when} & v_{ij} < 0 & u_{ij} = 0 & \frac{d\,u_{ij}}{d\,t} = 0 \end{array}$$

therefore

$$\frac{d\,E}{d\,t} = \sum_i\sum_j \frac{d\,E}{d\,u_{ij}}\frac{d\,u_{ij}}{d\,t} = K_c\sum_i\sum_j \frac{d\,E}{d\,u_{ij}}\frac{d\,v_{ij}}{d\,t}$$

using Eq.(6.67) and Eq.(6.64), we have

$$\frac{d\,E}{d\,t} \leq 0$$

The sigmoid function

$$\frac{1}{2}\left(1+\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}\right) \tag{6.69}$$

can be used instead of the piece-wise linear function Eq.(6.60) and still keep Eq.(6.68) hold. However, Eq.(6.60) makes the analysis of the network dynamics easier and results similar performance as using the sigmoid function, as long as K_d is kept small.

6.4 TSP, the Hopfield Model and Comparison

6.4.1 The Hopfield Model for the TSP

The traveling salesman problem (TSP) has the following simple description: given a complete digraph of N nodes with an $N \times N$ matrix $||d_{ij}|| \geq 0$ defining the length of arc (i,j), find a minimum length circuit (or *tour*) which goes through each node exactly once. Distances d_{ij} are symmetric, that is, $d_{ij} = d_{ji}$. Also assume $d_{ii} = 0$. The length $d(T)$ of a tour T is given by $d(T) = \sum_{(i,j)\in T} d_{ij}$.

The problem has been studied extensively for the past few decades and many algorithms have been proposed for its exact solution. Since it is an NP-complete problem, one is therefore also interested in approximate algorithms which take less time to get a good, but not guaranteed optimal, solution. Hopfield and Tank [30] gave $0 \leq v_{ij} \leq 1$ the following interpretation for all $0 \leq i,j < N$: $v_{ij} = 1$ means city i is the j-th stop in the salesman's tour; and $v_{ij} = 0$ means city i is not the j-th stop. A valid tour is one with exactly one 1 in each row and column of matrix $[v_{ij}]$. The energy function to be minimized is

$$\begin{aligned} E \;=\; & \frac{A}{2}\sum_{x}\sum_{i}\sum_{j\neq i} v_{xi}v_{xj} + \frac{B}{2}\sum_{i}\sum_{x}\sum_{y\neq x} v_{xi}v_{yi} + \frac{C}{2}\left(\sum_{x}\sum_{i} v_{xi} - N\right)^{2} \\ & + \frac{D}{2}\sum_{x}\sum_{y\neq x}\sum_{i} d_{xy}v_{xi}(v_{y,i+1} + v_{y,i-1}) \end{aligned} \tag{6.70}$$

where $A, B, C, D > 0$. The first and second term indicate that only one 1 is allowed in each column and row. The third term means the total number of 1's must be N. The fourth term is the tour length.

6.4.2 Comparison of the Assignment Problem with the TSP

Although a lot of studies and improvements have been made to the original Hopfield net, we will compare the inference network with it due to the similar appearance of the energy functions. From Section 6.1.2 and Section 6.4.1, we have observed some similarities and differences between the assignment problem and the TSP, and their optimization formulations; they can be summarized as follows:
Similarities:

- Constraints can be formulated into the same restrictions: one 1 in each row and each column;
- The A and B terms in objective function Eq.(6.57) is the same as those in energy function Eq.(6.70);
- Both Eq.(6.57) and Eq.(6.70) are to be minimized;
- Both use gradient descent which may end up in local minimum state;
- When D term is ignored, the eigenvalues of the connection matrices for both problems are similar, as we will see later and in [2].

Differences:

- Each valid assignment has one and only one distinct set of u_{ij} values; however, a single tour corresponds to 2 different sets of v_{ij} values, since a circular tour can be interpreted in two opposite directions;

- The C term in Eq.(6.70) restricts the total number of neurons on; C term in Eq.(6.57) specifies the number of on-neurons in each row and column;
- D terms for total benefit and total tour length are different due to the interpretation of neurons.

Because of the similarities of the two problems, many dynamical behaviors of the network for the assignment problem are similar to those of the Hopfield model for the TSP. Using a negative D, the objective function Eq.(6.57) for assignment can also be interpreted for the TSP, in which $u_{ij} = 1$ means city j is immediately after city i in the salesman's tour. Since this formulation often yields disconnected sub-circles, additional constraints prohibiting sub-tours must be used. Xu [92] studied this approach.

The major reason responsible for the Hopfield model's invalid tours does not exist in the similar model for assignment. A fundamental problem with the Hopfield TSP model is its degeneracy. Since a tour can be clockwise or anti-clockwise, it has 2 distinct v_{ij} valid states. The network aims at one tour but proceeds towards the 2 corresponding states, and finally it often ends up with an invalid state. In the assignment problem, each valid assignment corresponds to a unique u_{ij} array, hence the network evolves towards a particular state.

The constraint that there are exactly N 1's in the matrix $[u_{ij}]$ is represented differently in the neural models. The Hopfield model has a tendency of getting two or zero 1's in a row or column [90] while the total number of 1's is still N. C term of Eq.(6.57) specifically restricts the number of 1's in each row or column to be one. This reduces the chance of assigning one object to two persons or two objects to one person.

6.5 Dynamics of the Network

6.5.1 Connection Matrix

Let $\mathbf{u}$ be an $N^2 \times 1$ vector and u_{ij} be its $(iN+j)$-th element. Eq.(6.59-6.62) can be represented by the following matrix operation:

$$\mathbf{u}^{(k+1)} = f'(\mathbf{u}^{(k)} + K_d\mathbf{P}\mathbf{u}^{(k)} + K_d\mathbf{Q}) \tag{6.71}$$

where $\mathbf{P}$ is an $N^2 \times N^2$ symmetric matrix. Its element at row $iN+j$ and column $rN+s$ is (when $B = A$)

$$p_{iN+j,rN+s} = 2A\delta(i-r)\delta(j-s) - (A+C)\delta(i-r) - (A+C)\delta(j-s) \tag{6.72}$$

$\mathbf{Q}$ is an $N^2 \times 1$ vector, its element at row $iN+j$ is

$$q_{iN+j} = 2C + \frac{D}{2}r_{ij} \tag{6.73}$$

and

$$f'(\mathbf{X}) = f'((x_1, x_2, \cdots, x_m)^t) = (f(x_1), f(x_2), \cdots, f(x_m))^t$$

$$\delta(x) = \begin{cases} 1 & x = 0 \\ 0 & x \neq 0 \end{cases} \tag{6.74}$$

function $f(x)$ is defined in Eq.(6.60).

The determinant of $|s\mathbf{I} - \mathbf{P}|$, though lengthy, can be derived (see Appendix E.2 for details):

$$|s\mathbf{I} - \mathbf{P}| = (s + 2(N-1)A + 2NC)(s-2A)^{(N-1)^2}(s + (N-2)A + NC)^{2(N-1)}$$

Therefore, $\mathbf{P}$ has three distinct eigenvalues:

$$\lambda_1 = -2(N-1)A - 2NC \tag{6.75}$$

$$\lambda_2 = 2A \tag{6.76}$$

$$\lambda_3 = -(N-2)A - NC \tag{6.77}$$

since $A,\ C > 0$, we have $\lambda_1 < 0$, $\lambda_2 > 0$ and $\lambda_3 < 0$. The eigenvector corresponding to λ_1 is

$$\mathbf{e}_1 = (1, 1, \cdots, 1)^t \tag{6.78}$$

or

$$\hat{\mathbf{e}}_1 = (\frac{1}{N}, \frac{1}{N}, \cdots, \frac{1}{N})^t \tag{6.79}$$

The eigenvectors corresponding to λ_2 and λ_3 form $(N-1)^2$- and $2(N-1)$-dimensional sub-spaces $\Re_2$ and $\Re_3$ respectively. For any $A,\ C > 0$, $\lambda_1 \neq \lambda_2 \neq \lambda_3$, therefore,

$$\mathbf{e}_1 \perp \Re_2 \perp \Re_3$$

The objective function Eq.(6.57) can be written using matrices $\mathbf{P}$ and $\mathbf{Q}$ as:

$$E = -\mathbf{u}^t\mathbf{P}\mathbf{u} - 2\mathbf{Q}^t\mathbf{u} + 2C \tag{6.80}$$

6.5.2 Eigenvalues Move the Network Towards the Valid Sub-space

Since Eq.(6.71) contains non-linear function f', it is hard to analyze its dynamic behavior. However, when K_d is small enough to keep $f'(\mathbf{x}) = \mathbf{x}$, Eq.(6.71) can be simplified into

$$\mathbf{u}^{(k+1)} = \mathbf{u}^{(k)} + K_d\mathbf{P}\mathbf{u}^{(k)} + K_d\mathbf{Q} \tag{6.81}$$

Let $\mathbf{u}$ be decomposed into three components $\mathbf{u^1}$, $\mathbf{u^2}$ and $\mathbf{u^3}$ corresponding to direction $\mathbf{e}_1$, sub-space $\Re_2$ and $\Re_3$ respectively.

$$\mathbf{u} = \mathbf{u^1} + \mathbf{u^2} + \mathbf{u^3} \tag{6.82}$$

where $\mathbf{u^1} = (\mathbf{u} \cdot \hat{\mathbf{e}}_1)\hat{\mathbf{e}}_1$. Since λ_1, λ_2, and λ_3 are eigenvalues, we have

$$\mathbf{P}\mathbf{u}^{(k)} = \lambda_1\mathbf{u}^{1(k)} + \lambda_2\mathbf{u}^{2(k)} + \lambda_3\mathbf{u}^{3(k)} \tag{6.83}$$

$\mathbf{Q}$ in Eq.(6.81) has component of r_{ij}, hence, $\mathbf{Q}$ can not be pre-decomposed into the three sub-spaces. In order to analyze network dynamics, let us assume $D = 0$ for the moment, hence $\mathbf{Q} = 2C\mathbf{e}_1 = 2CN\hat{\mathbf{e}}_1$. With these assumptions and decomposition, we have, from Eq.(6.81),

$$\mathbf{u}^{1(k+1)} = \mathbf{u}^{1(k)} + K_d\lambda_1\mathbf{u}^{1(k)} + 2CNK_d\hat{\mathbf{e}}_1 = (1 + K_d\lambda_1)\mathbf{u}^{1(k)} + 2CNK_d\hat{\mathbf{e}}_1 \tag{6.84}$$

$$\mathbf{u}^{2(k+1)} = \mathbf{u}^{2(k)} + K_d\lambda_2\mathbf{u}^{2(k)} \tag{6.85}$$

$$\mathbf{u}^{3(k+1)} = \mathbf{u}^{3(k)} + K_d\lambda_3\mathbf{u}^{3(k)} \tag{6.86}$$

On the other hand, according to Eq.(6.80), we have,

$$\begin{aligned} E &= -\lambda_1|\mathbf{u}^1|^2 - \lambda_2|\mathbf{u}^2|^2 - \lambda_3|\mathbf{u}^3|^2 - 4CN|\mathbf{u}^1| + 2C \\ &= -\lambda_1(|\mathbf{u}^1| + \frac{2CN}{\lambda_1})^2 - \lambda_2|\mathbf{u}^2|^2 - \lambda_3|\mathbf{u}^3|^2 + \frac{4C^2N^2}{\lambda_1} + 2C \end{aligned} \tag{6.87}$$

Recall that $\lambda_1,\ \lambda_3 < 0$, and $\lambda_2 > 0$, in order to minimize E, what are required are

1. $|\mathbf{u}^1| = -\frac{2CN}{\lambda_1} = \frac{2CN}{|\lambda_1|}$
2. $|\mathbf{u}^2|^2$ increases with the increase of k, and
3. $|\mathbf{u}^3|^2$ decreases with the increase of k

From Eq.(6.84), we can get

$$\mathbf{u}^{1(k+1)} = (1 + K_d\lambda_1)^k(\mathbf{u}^{1(0)} + \frac{2CN}{\lambda_1}\hat{\mathbf{e}}_1) - \frac{2CN}{\lambda_1}\hat{\mathbf{e}}_1$$

for small K_d, we can have $-1 < K_d\lambda_1 < 0$, hence

$$\mathbf{u}^{1(\infty)} = -\frac{2CN}{\lambda_1}\hat{\mathbf{e}}_1$$

therefore,

$$|\mathbf{u}^{1(\infty)}| = \frac{2CN}{|\lambda_1|}$$

which ensures that the first requirement will eventually be satisfied. Since $\lambda_2 > 0$, Eq.(6.85) ensures $|\mathbf{u}^2|$ increasing with time. Similarly, the negative λ_3 ensures $|\mathbf{u}^3|$ decreasing with time.

For a valid solution, $\sum_i \sum_j u_{ij}^{(\infty)} = N$ holds, hence,

$$\mathbf{u}^{1(\infty)} \cdot \hat{\mathbf{e}}_1 = \mathbf{u}^{(\infty)} \cdot \hat{\mathbf{e}}_1 = \frac{1}{N}(\mathbf{u}^{(\infty)} \cdot \mathbf{e}_1) = \frac{1}{N}N = 1$$

and $\mathbf{u}^{2(\infty)}$ will be very big (without the non-linear function $f(x)$) and $\mathbf{u}^{3(\infty)}$ will be zero. The sub-space formed by eigenvectors corresponding to λ_2 contains valid solutions, and the sub-space formed by eigenvectors corresponding to λ_3 contains invalid solutions.

6.5.3 The Role of the Non-Linear Function

When K_d is set sufficiently small, $K_d \Delta u_{ij}^{(k)}$ can be very small. If $0 < u_{ij}^{(k)} < 1$ also holds, Eq.(6.81) will describe the exact behavior of the network. In this case, vector $\mathbf{u}$ moves in the direction towards the valid solution space. Only when $\mathbf{u}$ is sufficiently close to a corner of the N-dimensional hypercube, the non-linear function has its impact. But since $K_d \Delta \mathbf{u}$ is kept small, the result $f(\mathbf{u} + K_d \Delta \mathbf{u})$ can only be slightly different from $\mathbf{u} + K_d \Delta \mathbf{u}$. Therefore the result vector still points very close a hypercube corner. In other words, the solution is almost determined before the non-linear function starts to affect the change; the non-linear function only keeps $|\mathbf{u}|$ within a certain range.

6.5.4 The Complete Q Term

$D = 0$ was assumed in the last two subsections. However, D term is a major part of vector $\mathbf{Q}$. Since this term is related to r_{ij}, it may have components in all of $\mathbf{e}_1$, $\Re_2$ and $\Re_3$ subspaces. For a positive D, this term moves the vector $\mathbf{u}$ in a direction in favor of large r_{ij}, although this movement can be either towards or away from the valid solution space. However, if the speed of moving the vector towards the valid space is properly rated with

the speed towards a large benefit, this term can move the vector approximately within the valid space in order to arrive at a near-optimal solution. Like all the other gradient descent algorithms, vector **u** will fall into local minimum if constants A, B, C and D are not properly given.

6.5.5 Initialization, Updating Rate and Termination

Ideally, initial values $u_{ij}^{(0)}$ do not affect the final solution, since the solution is determined by $|\mathbf{u}^1| \rightarrow -\frac{2CN}{\lambda_1}$, $|\mathbf{u}^2| \rightarrow \infty$, $|\mathbf{u}^3| \rightarrow 0$, non-linear function $f(x)$, together with the D term implication. A simple and unbiased initialization is $u_{ij}^{(0)} = \frac{1}{N}$ for all i, j. This initialization can keep Δu_{ij} roughly invariant with N, so that K_d can be almost the same for different N's.

A uniform initialization may cause problems for some specific benefit matrices. For example, when the benefit matrix $[r_{ij}]$ has two rows or two identical columns, the problem has multiple symmetric solutions. In order to break the symmetry, small noise may be added to the initial values. For example,

$$u_{ij}^{(0)} = \frac{1}{N} * s \tag{6.88}$$

where s is a random number distributed between 0.9 and 1.1. A large value of K_d can speed up the converging process. However, a too large K_d may introduce the non-linear effect too early and cause oscillation. K_d should keep $K_d \Delta u_{ij}$ within a small percentage of the initial values $u_{ij}^{(0)}$ so that non-linearity does not occur too early.

Three different criteria are used to determine when a solution is achieved or whether the calculation should be terminated. The first of these is the discovery of a valid solution. Since u_{ij} represents assignment, a valid solution is obtained if and only if there is exactly one neuron in each row and column with *high* value and all the rest have *low* values. To determine whether a particular neuron is high or low, a threshold rule can be applied.

For example, those neurons with values greater than $T_h = 0.65$ are considered *high*, and those below $T_l = 0.30$ are considered *low*.

The second criterion is to check whether the network is still changing. If the total change of the u_{ij} values between two successive iterations is less than $T_2 = 10^{-4}$, the neurons are considered stabilized, and the network reaches a stable final state.

The final termination criterion is a time-out test. If after $T_3 = 1500$ iterations, neither of the above criteria has been satisfied, the system is considered divergent.

6.6 Determining Constants

From Eq.(6.57-6.62), we know that constants A, B, C, and D are related. B can be set equal to A due to the symmetry of the constraint. In Eq.(6.62), if we assume $B = A$, $u_{ij} = \frac{1}{N}\ \forall i,j$, $\Delta u_{ij} = \frac{D}{2} r_{ij} - 2A\frac{N-1}{N}$ can be obtained. In order to get a D invariant with N, we choose $A = B = \frac{N}{N-1}$.

From the previous discussion, we see that constants A and C determine the eigenvalues, Eq.(6.85-6.86) show that vector $\mathbf{u}$ moves towards the valid sub-space with a speed $|\lambda_2 K_d| = 2AK_d$, and moves away from the invalid sub-space with a speed of $|\lambda_3 K_d| = ((N-2)A+2NC)K_d$. If $D = 0$ and K_d is sufficiently small, any positive A and C lead to a valid solution. From Eq.(6.57), if $\frac{C}{A}$ is too large, then C term overweighs A term, hence the network may arrive at a stable state where all u_{ij}'s are close to $\frac{1}{N}$. On the other hand, if $\frac{C}{A}$ is too small, the network may converge to the all zero state, where A and B terms are minimized and C term is not. Simulation shows that the ratio $\frac{C}{A}$ can vary in a certain range to obtain a valid solution. However, the range is approximately anti-proportional to N.

Term D moves the vector $\mathbf{u}$ towards the direction in favor of the large r_{ij}'s. However, this direction may or may not be the direction of the valid sub-space. If r_{ij} are given

such that the direction of large r_{ij}'s is close to the direction of invalid sub-space, a large D may force the vector stay in the invalid space. If the direction of large r_{ij} is close to the direction of valid sub-space, but still with some difference, a not-to-large D leads to a solution at a corner of the hypercube; whereas a too-large D bends the vector away from the corner. The larger D is, the farther the vector is from the corner. However, the optimization problem is meaningless if D is too small. From simulation we noticed that the direction of the solution vector $\mathbf{u}$ is usually determined at the early stage of the calculation. In the beginning of the calculation, the large D term contributes by moving the vector towards larger total benefit state; when the calculation is near the end, it often tends to draw the vector away from a hypercube corner. Based on this observation, we start with a large D_0 and reduce its value by a small percentage in each iteration: $D := pD$, where $0 < p < 1$. With a decreasing D, the network can always arrive at a valid solution, since when D term is negligible, the minimum state of Eq.(6.57) is always a valid assignment.

The value of D is also related with the distribution of r_{ij}. In a valid solution, $\frac{N-1}{N}$ of the elements are expected to settle down at 0. Suppose benefits r_{ij} have probability distribution of $g(r)$, and $\tilde{r}$ satisfies $\int_{-\infty}^{\tilde{r}} g(r)dr = \frac{N-1}{N}$, suppose further $\sum_l u_{il} + u_{lj} = 2$, and $u_{ij} = \frac{1}{N}$, the solution D_0 from

$$\Delta u_{ij} = 2A\frac{1}{N} - 2(A + C) + 2C + \frac{D_0}{2}\tilde{r} = 0$$

can be a guideline in determining initial value D_0.

6.7 Simulation Results

The overall performance of the algorithm is satisfactory. For all the examples we used with N varies from 2 to 128 and even distribution of benefits r_{ij}, the network always converges to a valid assignment as long as the constants are chosen based on the above

guidelines. For a particular set of benefits r_{ij}, the solutions corresponding to different values of $\frac{C}{A}$, $\frac{K_d}{A}$ are usually the same. The value $\frac{D}{A}$ sometimes changes the solution for large problems, but the total benefit does not change significantly. For all the examples we tested with $N < 12$, the inference network algorithm gives the same result as exhaustive search.

6.7.1 Small Size Examples

The following example with $N = 10$ illustrate some properties of the network. The benefits r_{ij} are evenly distributed in $[0, 1]$:

```
0.053 0.914 0.827 0.302 0.631 0.785 0.230 0.011 0.567 0.350
0.307 0.339 0.929 0.216 0.479 0.703 0.699 0.090 0.440 0.926
0.032 0.329 0.682 0.764 0.615 0.961 0.273 0.275 0.038 0.923
0.540 0.443 0.837 0.368 0.746 0.469 0.505 0.328 0.480 0.424
0.678 0.139 0.763 0.959 0.707 0.242 0.663 0.759 0.332 0.455
0.685 0.716 0.136 0.720 0.832 0.751 0.681 0.106 0.379 0.719
0.381 0.919 0.163 0.219 0.639 0.261 0.040 0.144 0.941 0.872
0.569 0.972 0.364 0.684 0.931 0.423 0.927 0.594 0.182 0.611
0.401 0.868 0.680 0.538 0.940 0.512 0.289 0.621 0.970 0.668
0.693 0.352 0.940 0.208 0.571 0.579 0.821 0.963 0.724 0.762
```

Using exhaustive search (takes about one hour on Sun/3), the optimal assignment with total benefit of 9.053 was found:

```
0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0 0 0
```

```
0 0 0 1 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0
```

The inference network algorithm gives the above optimal solution almost instantly. For this example, we used initial value $u_{ij}^{(0)} \in [\frac{0.9}{N}, \frac{1.1}{N}]$, and D decreases by 0.6% in each iteration ($p = 0.994$), the ranges of C, D_0, K_d that keep the network to converge are:

$$0.05 \leq C \leq 9.9 \quad \text{with } A = B = \frac{N}{N-1},\ D_0 = 4A,\ p = 0.994,\ K_d = 0.01$$

$$0.5 \leq D_0 \leq 11 \quad \text{with } A = B = \frac{N}{N-1},\ C = \frac{75}{N},\ p = 0.994,\ K_d = 0.01$$

$$0.0005 \leq K_d \leq 0.014 \quad \text{with } A = B = \frac{N}{N-1},\ C = \frac{75}{N},\ D_0 = 4A,\ p = 0.994$$

With $A = B = \frac{N}{N-1}$, $C = \frac{75}{N}$, $D_0 = 4A$, and $K_d = 0.01$, D's decreasing rate p can be within the range of $p \in [0.75, 1]$ to keep the solution optimal. K_d is the main factor determining converging speed. Figure 6.20 shows the number of iterations required for the network to settle down with respect to different values of the constants.

Additional constraints, such as somebody has to get a certain object or somebody must not get a certain object, can be introduced to the algorithm without much extra efforts. The same non-zero K_d is used for all the neurons which do not involve any additional constraints; and $K_d = 0$ is used for all the neurons with additional constraints. If person i has to get object j, set $u_{ij}^{(0)} = 1$; if person i must not get object j, set $u_{ij}^{(0)} = 0$. In the above $N = 10$ example, if we specifically require $u_{0,1} = 0$, $u_{6,8} = 0$ and $u_{2,2} = 1$, $u_{8,5} = 1$, our algorithm gives the following result with total benefit of 7.886:

0 0 0 0 0 0 0 0 1 0

0 0 0 0 0 0 0 0 0 1

0 0 1 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0 0 0

0 0 0 1 0 0 0 0 0 0

1 0 0 0 0 0 0 0 0 0

0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 1 0 0 0

0 0 0 0 0 1 0 0 0 0

0 0 0 0 0 0 0 1 0 0

If the benefit matrix $[r_{ij}]$ has two identical rows, the assignment for the two corresponding persons can be exchanged without changing the total benefit. These two assignments are considered symmetric. Similarly, if there are two identical columns in the benefit matrix, there will also be two symmetric assignments. Even two rows or columns have only part of them identical, it is still possible to have symmetric optimal assignments. For example, the following matrix (a) has two optimal assignments (b) and (c), both having total benefits of 3.0.

0.4	0.6	0.9	0.1		0	0	1	0		0	1	0	0
0.4	0.6	0.9	0.1		0	1	0	0		0	0	1	0
0.1	0.2	0.4	0.8		0	0	0	1		0	0	0	1
0.7	0.4	0.2	0.9		1	0	0	0		1	0	0	0
	(*a*)					(*b*)					(*c*)		

If initial values are all identical, we will get $u_{0j} = u_{1j}$ for all j and all iterations, therefore, the network will not arrive at a valid solution; instead, it will stay at a state with $0 < T_l < u_{01} = u_{11}, u_{02} = u_{12} < T_h < 1$, $u_{23} = 1.0$, $u_{30} = 1.0$, and all the rest are

0. However, if small amounts of noise are applied to $u_{ij}^{(0)}$ with Eq.(6.88), the network is observed to arrive at either solution (b) or (c) quickly.

The number of iterations required for the neural algorithm to arrive at its solution is mainly determined by the distribution of benefits r_{ij}'s rather than their range (as for the auction algorithm). If larger r_{ij} values are scatted in different rows and columns, the network arrives at the solution fast. On the other hand, if large benefits are concentrated in some rows and columns, the convergence speed is slower. For the following random benefit matrix (a)

0.053	0.914	0.827		0	1	0
0.302	0.631	0.785		0	0	1
0.230	0.011	0.567		1	0	0
	(a)				(b)

with $A = B = \frac{N}{N-1}$, $C = \frac{75}{N}$, $D_0 = 4A$, $p = 0.994$, and $K_d = 0.01$, the optimal assignment (b) is obtained in 70 iterations. Using the same constants for another benefit matrix (c),

0.1	0.8	0.11	0	0	1	0	1	0	0	1	0	1	0	0
0.9	0.85	0.9	1	0	0	1	0	0	0	0	1	0	0	1
0.16	0.87	0.17	0	1	0	0	0	1	1	0	0	0	1	0
	(c)			(d)			(e)			(f)			(g)	

it takes 168 iterations to arrive at the optimal assignment (d). Since matrix (c) is a tricky example with three near-optimal valid assignments (e), (f), and (g). If the noise on the initial values are too big, the network may stabilized at a sub-optimal state.

The number of iterations required increases gently with N. Figure 6.21 shows the results. The range of C is also related with problem size N. The larger N is, the smaller the C range will be to keep proper ratios of the A, B, C terms. Figure 6.22 shows the maximum C values for different N's which yield valid solutions.

6.7.2 Solve Large Problems on the CM

Simulations had been conducted for the cases of $N = 64$ and $N = 128$ on the $8K$ and $16K$ Connection Machine respectively. The algorithm is outlined in Algorithm 4. Table 6.7 and Table 6.8 gives the CM time and total benefits obtained. All benefits r_{ij} are evenly distributed in $[0,1]$. By using a largest converging K_d, the CM time can be made shorter. Changing D's decreasing rate p may result in a different assignment with slightly different total benefit. Statistically, given i evenly distributed numbers in $[0,1]$, the mathematical expectation of the largest number is $\frac{i}{i+1}$. If we pick the largest number in the first row and then delete the column it is in, pick the largest number in the second row and delete the column it is in, and so on, the mathematical expectation of the total benefit is $E_N = \sum_{i=1}^{N} \frac{i}{i+1}$. For $N = 64$ and $N = 128$, the expectations are $E_{64} = 60.24$ and $E_{128} = 124.56$, respectively. Although the inference network algorithm does not guarantee an optimal solution, the average total benefits we obtained are greater than these expectations. For comparison, optimal total benefits from the Hungarian method are also given.

Algorithm 4 : *Assignment on the CM*

for all PE, $u[i,j] := u_0$, *where* $u_0 \in [\frac{0.9}{N}, \frac{1.1}{N}]$

while (terminating criteria not met) (refer Section 6.5.5)

BEGIN PARALLEL for all i, j

$h[i,(-i-j)mod(N)] := u[i,j]$, $v[(-i-j)mod(N),j] := u[i,j]$

FOR $k = 0$ *TO* $k = N-1$

BEGIN

update $u[i,j]$ *using Eq.(6.58-6.62)*

propagate: $h[i,j] := h[i,(j-1)mod(N)]$ $v[i,j] := v[(i-1)mod(N),j]$

END

END PARALLEL

random problem	1	2	3	4	5	6	7
CM busy time	31.5	37.7	37.9	31.4	37.7	33.8	40.6
benefit–Inference	61.832	62.367	62.047	62.232	61.168	61.634	61.461
benefit–Hungarian	62.450	62.367	62.693	62.443	62.666	62.294	62.559
random problem	8	9	10	11	12	13	14
CM busy time	41.7	32.4	38.6	32.0	45.8	43.4	38.0
benefit–Inference	61.717	61.656	61.109	61.598	60.093	62.127	61.492
benefit–Hungarian	62.653	62.551	62.220	62.509	62.594	62.613	62.633
random problem	15	16	17	18	19	20	*average*
CM busy time	41.5	30.5	39.3	36.5	35.0	31.2	36.8
benefit–Inference	61.417	61.614	60.900	61.065	61.263	62.002	61.5317
benefit–Hungarian	62.294	62.460	62.357	62.443	62.486	62.659	62.4957

Table 6.7: CM busy times (in seconds) and assignment results for 20 random $N = 64$ problems using inference network ($A = B = \frac{N}{N-1}, C = \frac{75}{N}, D_0 = 4, p = 0.996, K_d = 0.008$) and corresponding result from the Hungarian method.

random problem	1	2	3	4	5	6	7
CM busy time	118.8	112.4	94.7	121.3	101.6	115.1	99.2
benefit–Inference	124.846	125.232	125.295	124.723	124.165	124.697	124.983
benefit–Hungarian	126.503	126.478	126.318	126.509	126.276	126.463	126.187
random problem	8	9	10	11	12	13	14
CM busy time	109.6	120.6	99.4	106.7	117.3	133.9	114.4
benefit–Inference	124.131	124.851	125.742	125.293	124.835	125.710	124.180
benefit–Hungarian	126.323	126.227	126.368	126.486	126.465	126.478	126.517
random problem	15	16	17	18	19	20	*average*
CM busy time	120.4	135.3	117.1	111.8	113.5	124.6	114.4
benefit–Inference	124.493	124.825	124.880	124.584	124.104	125.109	125.0321
benefit–Hungarian	126.293	126.464	126.362	126.411	126.386	126.587	126.4051

Table 6.8: CM busy times (in seconds) and assignment results for 20 random $N = 128$ problems using inference network ($A = B = \frac{N}{N-1}, C = \frac{75}{N}, D_0 = 4, p = 0.998, K_d = 0.007$) and corresponding result from the Hungarian method.

6.8 Discussion

The auction algorithm can be parallelized. Different from the Hungarian method and the inference network algorithm, the complexity of the auction algorithm depends on the range of the benefits: its complexity is $O(NM\log(NM))$, where M is the largest *integer* benefit. The auction algorithm is efficient at the beginning when many people bid for assignments. However, it usually has a long 'tail' when only a few people are left unassigned. This tail results in a relatively long completion time. Wein etc. [89] proposed a hybrid auction algorithm for the CM in which tails are truncated and processors are dynamically assigned to the active bidders. The algorithm can shorten the tailing effect when N is large (for example $N \geq 256$) and virtual processor ratio is greater than 1. It can solve an $N = 1000$ or $N = 2000$ problem with maximum benefit $M = 1000$ on the $16k$ CM in 22.2 and 96.3 seconds respectively. Therefore, the hybrid auction algorithm is more efficient than the inference network algorithm for a large size assignment problem.

In all our simulations with N up to 128, the inference network algorithm was shown to produce valid assignments and the results were usually close to optimal; the smaller the problem size was, the more likely an optimal solution would be reached. 20 different benefit sets for each problem size N were tested. The results obtained are given in Table 6.9. The optimum solutions in the table were obtained from the Hungarian method.

problem size N	4	8	16	32	64	128
# of optimal solutions obtained	20	20	13	2	1	0
average difference from optimum	0.0%	0.0%	0.5%	1.1%	1.5%	1.1%

Table 6.9: Solution quality vs. problem size. 20 distinct random examples were used for each N.

The computational complexity of the Hungarian method is $O(N^4)$. It takes about two minutes for the Hungarian method to produce an $N = 128$ assignment on a Sparc station. With a much slower clock rate (7 MHz), the inference network algorithm on the CM also takes about two minutes.

The Hopfield's model for the traveling salesman problem presented an approach for constrained optimization different from traditional AI methods. Unfortunately, it often yields invalid solutions and does not work well for large problems (for example $N >$ 15) [90]. Despite the similarities, the inference network algorithm for the assignment problem did not yield invalid solutions as the Hopfield model did for the TSP, due to the formulation of the problem and the setting of the constants (Simulations were conducted for random problems with size up to 128).

Both the assignment problem and the TSP are formulated to find an on-off neuron pattern to optimize an objective function. In the inference network, each valid assignment corresponds to a single state satisfying the constraint — exactly one neuron on in each row and column. However, the Hopfield model for the TSP is degenerated: in the Hopfield net, each valid tour corresponds to two distinct states, and both satisfy the constraint. Therefore, although the Hopfield net is a great idea for combinatorial problems, the TSP may need to be remodeled into some other constraints on neurons.

The D term in the objective (energy) function measures the quantity to be optimized. Therefore this term must be significantly large to achieve the goal of optimization. On the other hand, large D terms lead to divergence; a small D is advantageous for the network convergence. A compromise between these two aspects is to use a decreasing D term. Our simulation has shown that using a decreasing D, the network converges for all the test cases with problem size up to 128. Since the Hopfield net has similar eigen characteristics to the inference network, it can be expected that the Hopfield net can also have better convergence if a decreasing D is used. However, a decreasing D will not

solve the degeneracy problem of Hopfield's model for the TSP.

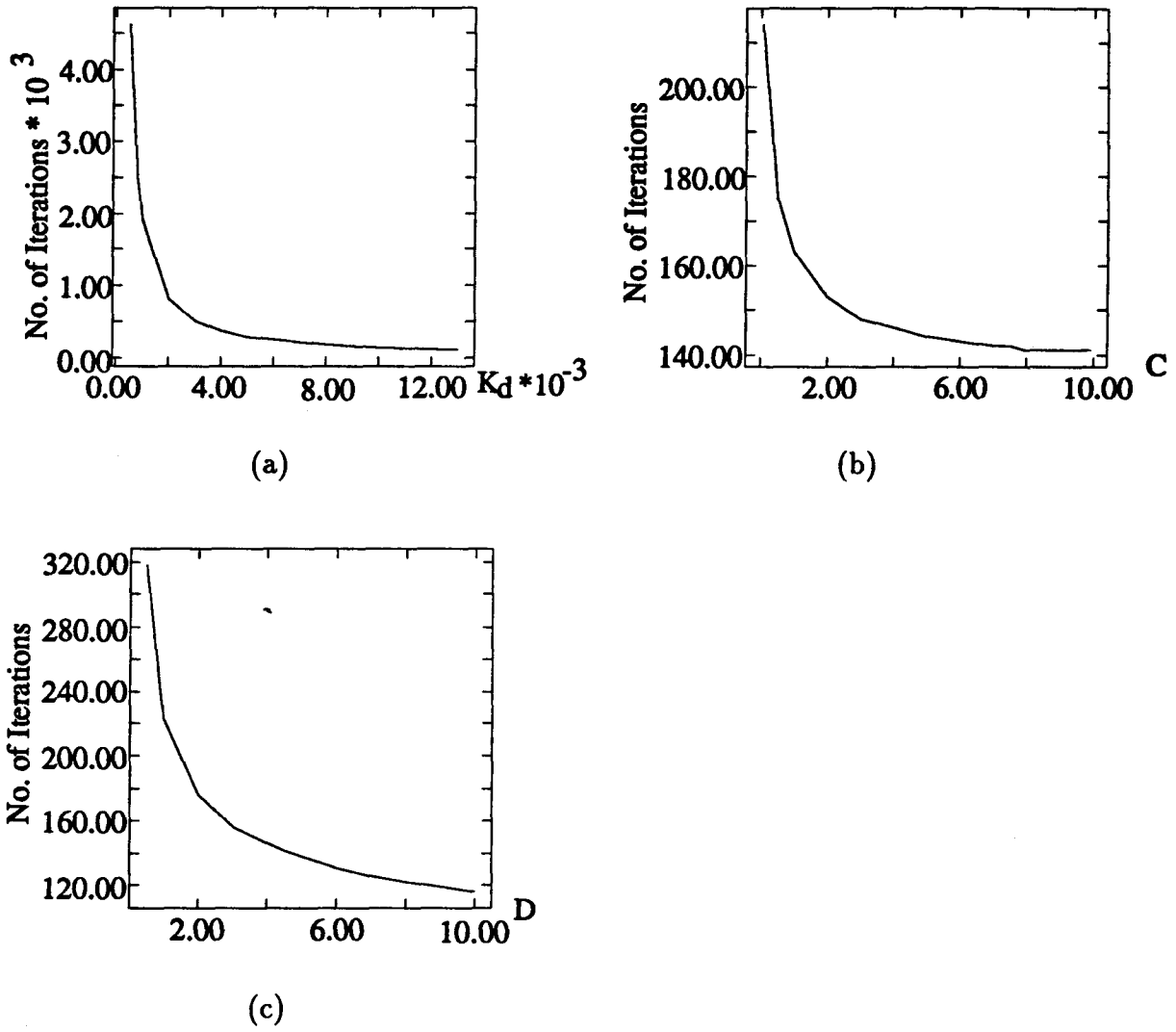


Figure 6.20: (a) $A = B = \frac{N}{N-1}$, $C = \frac{75}{N}$, $D_0 = 4A$, $p = 0.994$, the larger K_d is, the faster the network converges; (b) $A = B = \frac{N}{N-1}$, $D_0 = 4A$, $p = 0.994$, $K_d = 0.01$, the network converges slowly when C is small, and converging speed is almost the same within a certain range of C; (c) $A = B = \frac{N}{N-1}$, $C = \frac{75}{N}$, $p = 0.994$, $K_d = 0.01$, the larger D_0 is, the faster it converges.

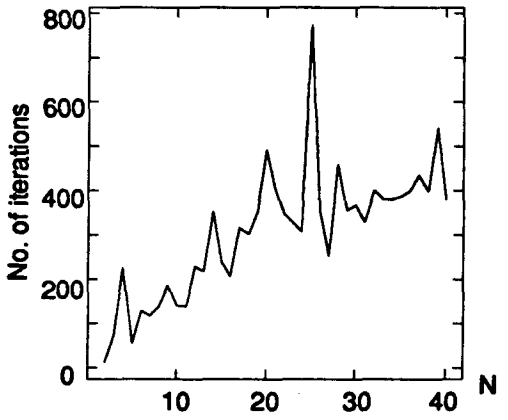


Figure 6.21: Number of iterations vs. problem size. All benefits r_{ij} are randomly distributed in $[0, 1]$. $A = B = \frac{N}{N-1}$, $C = \frac{75}{N}$, $D_0 = 4A$, $K_d = 0.01$, $u_{ij}^{(0)} \in [\frac{0.9}{N}, \frac{1.1}{N}]$.

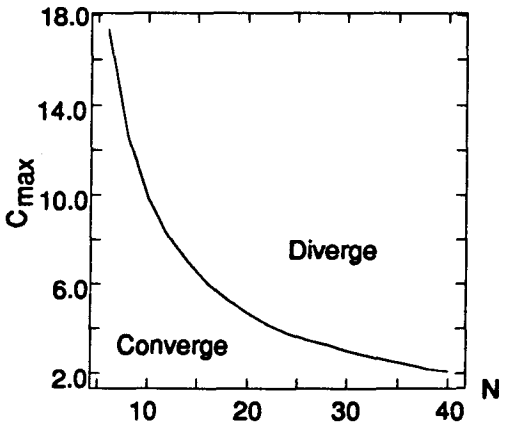


Figure 6.22: Maximum C for the algorithm to converge vs. problem size. All benefits r_{ij} are randomly distributed in $[0, 1]$. $A = B = \frac{N}{N-1}$, $D_0 = 4A$, $K_d = 0.01$, $u_{ij}^{(0)} \in [\frac{0.9}{N}, \frac{1.1}{N}]$.

Chapter 7

Conclusions

7.1 Summary

This thesis introduced a novel parallel computing platform — the binary relation inference network, which embodies time-efficient parallel computing structures to perform a class of optimization problems. Candidates competing for the optimum were explicitly represented by the sites at a unit; the selection of the optimum was conducted by the unit functions. The inference network algorithms were derived directly from suitable problem formulations.

Sites in the inference network perform binary deduction and produce candidates. Units collect the candidates and make decisions based on the optimization criteria. The complete inference network is composed of N^2 units, each unit has a fan-in of N and a fan-out of $2N$. The network has logarithmic convergence rate for the graph problems discussed. Each individual unit is a simple computing device. The inference network is a uniform structure with a high degree of redundancy, since the nature of the optimization problem is to make decisions based on multiple choices. The connectivity of the network units grows with problem size N.

Inference network algorithms in both discrete and continuous-time domains for the shortest path problem, the transitive closure problem, and the assignment problem were discussed. The synchronous inference network algorithm for the shortest path problem was mathematically equivalent to the divide and conquer version of the sequential matrix

algorithm for the problem. Compared with all other shortest path algorithms discussed in Chapter 4, the inference network was shown to solve the problem with the least number of synchronous iterations. Therefore, it will be the fastest if every algorithm runs on the kind of hardware that optimizes its performance. Implementing the discrete algorithm on the Connection Machine resulted in a shorter completion time than one of the fastest systolic algorithms for $N \leq 512$ (the maximum size investigated). It was also demonstrated that the network can unify many existing all-pair shortest path algorithms by defining different unit functions. The inference network algorithm, which was based on simple procedures for finding the shortest path, was proved to produce optimal solution for synchronous, asynchronous, and continuous-time updating, and its convergence was independent of the problem size.

The inference network algorithm for the transitive closure problem was derived directly from the problem formulation. The solution was obtained efficiently using combinational logic without a clock. No gradient descent was involved. Theoretical analysis proved that the convergence of the algorithm to the optimal solution was independent of the problem size. The structure of the continuous-time network and simulation of its performance were given.

The dynamic behavior of the inference network and the impact of its energy constants were discussed through the example of the assignment problem. The constants in the energy function (corresponding to link weights) were determined for the problem. Compared with results from exact solutions, the assignment results from the inference network were found to be near optimal for all the test cases with N up to 128. Compared with the Hopfield model for the TSP, the inference network was shown to converge for a larger size of problems, due to the use of a decreasing weight for the objective function as well as the formulation of the problem.

The inference network was also applied to data consistency checking (Appendix A),

consistency labeling (Appendix C), and matrix inversion (Appendix B). For data consistency checking or consistency labeling, the inference network finds or eliminates all inconsistency. Among the applications discussed, the assignment problem was solved using a different strategy from the other problems. For the assignment problem, the inference network aims at decreasing the network energy through gradient descent, which is very similar to other neural network optimization approaches. For other problems (shortest path, transitive closure, data consistency checking, and consistent labeling), the inference network performs simple optimization procedures for the problem, and the decreasing of the network energy was just a result of optimization.

For all the applications discussed except matrix inversion, synchronization of the units was not require. In matrix inversion, the inference network was used as a systolic array.

7.2 Contribution

The major contribution of this thesis is the architecture of the inference network. The network was designed to implement the optimization procedure in a direct manner. Each decision making unit in the network always has access to all candidates competing for the optimum and these decision-making units can operate asynchronously. Once a computation has been started, the decision making units actively optimize their values until an optimum is reached.

For the class of graph problems considered in this thesis (shortest path, transitive closure, consistent checking), the inference network competes with previously formulated neural network models. The inference network has the advantage of requiring no training or weight determination. Additionally, the inference network units have much smaller fan-in and fan-out than the neurons of a typical neural network. These features are fundamental to cost-effective hardware implementation.

The inference network has shown some of the advantages of a neural network and some of the advantages of a typical parallel processing array. Like many neural networks, the inference network does not require a rigorous synchronization and its units are simple computational engines. When applied to the assignment problem, the inference network behaves like a Hopfield-type neural network. Similar to a parallel processing array, the inference network can achieve optimal solution bounds for many problems (for example, shortest path, transitive closure, and consistent labeling) and the inter-processor links carry no weights. It can also be used synchronously to perform matrix inversion like a systolic array.

The inference network was shown to work in both synchronous and asynchronous modes, in both discrete-time and continuous-time domains. Traditional AI algorithms, systolic array algorithms, and dynamic programming algorithms concentrate on the detailed recurrence formulations. They are usually discrete-time algorithms pursuing exact solutions. A systolic array must have a clock to synchronize all the processing elements to ensure the correctness of the results. The inference network for most of the problems discussed was shown not to require rigorous synchronization, therefore a clock was not necessary. It was shown that the inference network can be built using simple integrated circuits (for shortest path), logic gates (for transitive closure) and simple neurons (for consistency labeling). The analog inference network operating in a continuous-time domain was shown to be easy to implement since it does not need a clock and it does not need one set of circuits for each bit of data.

Many neural network optimization approaches define an energy function in the form of a weighted sum and pursue the solution by decreasing the energy in gradient descent. Optimal solutions are not guaranteed and, if the energy function has more than one weight, determination of the weights in the energy function (and consequently the link weights in the network) is not a easy job. For all the applications discussed in the

thesis except the assignment problem, no gradient descent was involved in the inference network.

The inference network architecture was derived from simple optimization procedures. For problems in which the optimization of a binary relation is always advantageous to the optimization of other binary relations, (for example the shortest path, transitive closure, data consistency checking, and consistent labeling), the network remains active until all the units reach their optimum and the global optimal solution is obtained. In these cases, network links carry no weights.

The detailed structure difference between the inference network and neural networks or other parallel computing models is that the inputs to an inference network unit are grouped into sites, and the evaluation of each site is independent of the evaluation of other sites. This gives a unit the freedom to update the unit value whenever a new candidate for the optimum is produced.

7.3 Open Issues and Future Work

One of the open issues about the inference network is the number of units it requires. For example, classical neural network approaches for the N-city traveling salesman problem require N^2 neurons [30]. The maximum neural network model [49] solves the K-partite subgraph problem with $N * K$ neurons. The neural network approaches for the shortest path problem also use N^2 neurons [19, 83]. A question that arises is whether it is necessary for the inference network to have N^2 units for a problem of size N (e.g. a graph with N nodes). Each unit in the inference network represents a binary relation. If the objective is to obtain a value for each of the N^2 binary relations concerning N nodes, having less than N^2 units means that the final results must be obtained in some order and some kind of synchronization is required. Therefore, it is not very likely that the number of

units in the inference network can be reduced while preserving its asynchronous feature.

When the problem size is large, the divide and conquer scheme may be used to solve a large problem in a number of iterations. The transitive closure problem is an example that can be tackled by a divide and conquer scheme. After applying the inference network to a subgraph of the original problem, all the connected nodes can be merged and treated as one single node in the rest of the graph. Therefore, a smaller inference network can be applied to a larger graph and the transitive closure of the original graph will be obtained after using the inference network iteratively.

Another open issue is what other problems can be solved on the inference network? To shed some light on this issue, the problems of the shortest and longest path will be examined. The shortest path and the longest path problems can be similarly represented by a set of binary relations. The shortest path problem can be solved on the inference network while the longest path cannot. The basic reason is not that the former is a polynomial-complexity problem and the latter is an NP-complete [21] problem. In the shortest path problem, the shortest path length from node i to j always equals to the total of the shortest path lengths from i to k and from k to j for all k's. However, the longest path from i to j is often not the concatenation of the longest path from i to k and from k to j for any k. The question is which problem can be naturally represented by binary relations and solved through binary operations on these relations (as discussed in Section 2.8).

Can the inference network be applied to an NP-complete problem? Although none of the applications discussed in this thesis are NP-complete, this does not exclude the possibility of applying the inference network to an NP problem. However, since the inference network consists of a polynomial number of processors, if a polynomial completion time is expected to solve an NP-problem, the solution can not be guaranteed optimal. The author has attempted to implement the algorithm proposed by Press [68] for the

traveling salesman problem on the inference network. The algorithm combines simulated annealing with route rearrangement. The inference network supports the communication requirement for the kind of route rearrangement used. However, since the algorithm often converges to a local minimum and route rearrangement is sequential in nature, the performance of the inference network for this algorithm is not outstanding. It remains open to find efficient mapping of NP-complete problems on the inference network.

Is it possible for the inference network to produce an optimal solution to an NP-complete problem if non-polynomial time is allowed? This also remains open for further study. For the polynomial-complexity shortest path problem, Section 4.3.3 has shown that with our definition of unit function and network energy, the energy function is a decreasing concave function with only one minimum. This allows the inference network to produce the optimal solution to the shortest path problem in asynchronous or continuous-time domains. For an NP-complete problem, the shape of the search space is so complicated that it is very hard to define the unit functions of the inference network so that the network energy has only one minimum.

The inference network can be applied to dynamic programming, often a process of sequential decision making. Computing shortest path is a typical problem that can be solved by dynamic programming [11]. Since the shortest path problem can be solved efficiently on the inference network, this opens up the possibility of applying the non-sequential inference network to some dynamic programming problems. However, whether and how a specific problem can be solved in a non-sequential way requires further study.

Some future work which may be done to the inference network includes:

- Use the inference network platform for logical analysis that can be represented by an AND/OR graph. Chapter 1 discussed the relation between the inference network topology and an AND/OR graph. In artificial intelligence, many logical

reasoning problems can be represented by an AND/OR graph. Therefore, if the reasoning procedures can be parallelized, it is possible to use the inference network to carry out the analysis.

- Derive other optimization applications for the inference network, for example:
  - *Weapon-Target Association*: The costs of using any of N weapons for any of N targets are given. The objective is to minimize the total cost of associating the weapons to the targets. The problem is similar to the assignment problem except that an advanced weapon may be able to kill several targets and a particularly dangerous threat may require targeting by several weapons;
  - *Minimum Spanning Tree*: It is a fundamental problem in artificial intelligence and intimately related with the assignment and traveling salesman problems;
- Map the inference network onto a 3-dimensional (instead of current 2-dimensional) processor array in the Connection Machine, where the third dimension corresponds to the sites at each unit. This will increase the number of virtual processors required, however, site calculation will be parallelized. Refer Section 3.3;
- Solve the traveling salesman problem by adding additional constraints to the assignment algorithm to eliminate sub-tours, refer Xu [92];
- Design the general or special purpose inference network using VLSI circuits and test its performance;
- Relate the algorithms to practical applications in robotics, image processing, VLSI layout, and control etc.

Bibliography

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Pub. Co., 1974

[2] S.V.B. Aiyer, M. Niranjan, and F. Fallside, 'A Theoretical Investigation Into the Performance of the Hopfield Model', *IEEE Trans. on Neural Networks,* Vol. 1(2), pp. 204-215, 1990.

[3] H.M. Alnuweiri, 'Constant-Time Parallel Algorithms for Image labeling on a Reconfigurable Network of Processors', Technical report CICSR-TR92-004, Dept. of Electrical Engineering, Univ. of British Columnbia

[4] D.H. Ballard and C.M. Brown, *Computer Vision*, Prentice-Hall, Englewood Cliffs, NJ, 1982

[5] D.P. Bersekas, 'The Auction Algorithm: A Distributed Relaxation Method for the Assignment Problem', *Annals of Operations Research*, vol. 14, pp. 105-123, 1988

[6] G. Broomell and J. Robert Heath, 'Classification Categories and Historical Development of Circuit Switching Topologies' *Computing Surveys*, 15(2) 95-133, June 1983

[7] W.K. Chen, *Theory of Nets: Flows in Networks*, John Wiley & Sons, Inc. 1990

[8] Cormen, Leiserson and Rivest, *Introduction to Algorithms*, McGraw-Hill, 1990

[9] G.B. Dantzig, 'All Shortest Routes in a Graph', *Operations Research House, Stanford Univ. Tech. Rep.*, pp. 66-3, (1966)

[10] E.W. Dijkstra, 'A Note on Two Problems in Connexion with Graphs', *Numer. Math.*, 1, (1959) pp. 269-271

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

[12] A. El-Amawy, 'A Systolic Architecture for Fast Dense Matrix Inversion', in *IEEE Computer,* Vol. 38, No. 3, 1989

[13] B.A. Farley, A.H. Land and J.D. Murechland, 'The cascade algorithm for finding all shortest distances in a directed graph', *Management Sci.*, 14, pp. 19-28 (1967)

[14] J.A. Feldman and D.H. Ballard, 'Connectionist Models and Their Properties', Cognitive Science, Vol. 6, pp. 205-254, 1982

[15] J.A. Feldman, M.A. Fanty, N.H. Goddard, and K. Lynne, 'Manual of Rochester Connectionist Simulator', 1986.

[16] J.A. Feldman, M.A. Fanty, N.H. Goddard, and K. Lynne, 'Computing with Structured Connectionist Networks', *IEEE Trans. Computer,* pp. 91-102, March 1988.

[17] R.W. Floyd, 'Algorithm 97, Shortest Path', *Comm. ACM*, Vol. 5 (1962), pp. 345

[18] S. Fortune and J. Wyllie, 'Parallelism in Random Access Machines', *Proc. of 10th Annual ACM Symp. on Theory of Computing*, pp. 114-118, Mar. 1978

[19] T. Fritsch and W. Mandel, 'Communication Network Routing using Neural Nets - Numerical Aspects and alternative Approaches', in *IEEE*, 1991

[20] *Tutorial text book*, The 2nd Int'l Conf. on Fuzzy Logic and Neural Networks, July 17-22, 1992, IIZuka, Japan

[21] M.R.Garey and D.S. Johnson, *Computers and Intractability: A guide to the Theory of NP-Completeness*, W.H. Freeman and Company, 1979

[22] N.H. Goddard, M.A. Fanty, and K, Lynne, *The Rochester Simulator,* Tech. Rep. 233, Computer Science Dept., Univ. of Rochester, 1987.

[23] L.J. Guibas, H.T.Kung, and C.D. Thompson, 'Direct VLSI Implementation of Combinational Algorithms', in *Proc. Caltech Conference on VLSI,* California Institute of Technology, Pasadena, 1979, pp. 509-525

[24] S. Hammering, 'A Note on Modifications to the Given's Plane Rotation', in *J. Inst. Math. Appl.* Vol. 13, pp. 215-218, 1974

[25] R. Helton, 'A self-adaptive connectionist shortest-path algorithm utilizating relaxation methods', *Proc. Int'l Joint Conf. on Neural Networks*, vol. 3, pp. 895-901, 1991

[26] W.D. Hillis, *The Connection Machine*, The MIT Press, 1985

[27] , 'Connectionist Learning Procedures', in *Artificial Intelligence*, Vol. 40, pp. 185-234, 1989

[28] J.J. Hopfield, 'Neural Networks and Physical Systems with Emergent Collective Computational Abilities', *Proc. Nat'l Acad. Sci. USA*, Vol. 79, pp. 2554, 1982.

[29] J.J. Hopfield, 'Neurons with Graded Response Have Collective Computational Properties Like Those of a Two-State Neurons', *Proc. Nat'l Acad. Sci. USA*, Vol. 81, pp. 3088, 1984.

[30] J.J. Hopfield, and D.W. Tank, 'Neural Computation of Decisions in Optimization Problems', *Biolog. Cybern.,* Vol. 52, pp. 1-25, 1985.

[31] J.J. Hopfield, and D.W. Tank, 'Computing with Neural Circuits: A Model', *Science,* Vol. 233, pp. 625-632, 1986.

[32] T.C Hu, 'Revised matrix algorithms for shortest paths', *SIAM J. Applied Math.*, 15 (1967), pp. 207-218

[33] Y.N. Huang, J.P. Cheiney, 'Parallel computation of direct transitive closures', *Proc. of 7th Int'l Conf. on Data Engineering*, pp. 192-199, 1991

[34] G-H Kim, H-Y Hwang, and C-H Lee, 'A modified Hopfield network and its application to the layer assignment', *Trans. of the Korean Institute of Electrical Engineers*, vol. 40, Iss. 2, pp. 234-237, 1991, in Korean

[35] S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi, 'Optimization by Simulated Annealing', *Science*, 220, pp. 671-680, 1983

[36] H. Kita, O. Sekine and Y. Nishikawa, 'A design Principle of the Analog Neural Network for Combinatorial Optimization: Findings from a Study on the Placement Problem', poster presentation during the Int'l Joint Conf. of Neural Network, July, 1991, Seattle.

[37] J.J. Kosowsky and A.L. Yuille, 'Solving the Assignment Problem with Statistical Physics', *Int'l Joint Conf. on Neural Networks*, vol. 1, pp. 159-164, July 1991, Seattle

[38] H.W. Kuhn, 'The Hungarian Method for the Assignment Problem', *Naval Research Logistics Quarterly*, 1955

[39] S.Y. Kung, and S.C. Lo, 'A spiral systolic architecture /algorithm for transitive closure problems', *IEEE Int'l Conf. on Computer Design*, ICCD'85, New York, USA, pp. 622-626

[40] S.Y. Kung, P.S. Lewis, and S.C. Lo, 'On Optimally Mapping Algorithms to Systolic Arrays with Application to the Transitive Closure Problem', *IEEE Int'l. Symposium on Circuits and Systems*, vol. 3, 1986

[41] K.P. Lam, 'Using Hybrid Techniques for a Continuous-Time Inference Network', Proc. of the IEEE Symposium on Circuits and Systems, pp. 1721-1724, 1991.

[42] K.P. Lam, 'A Continuous-Time Inference Network for Minimum-Cost Path Problems', *Int'l Joint Conf. on Neural Networks*, July 1991, Seattle, pp. 367-372

[43] K.P. Lam, and C.J. Su, 'A Binary Relation Inference Network', *Proc. of the Fifth Int'l Parallel Processing Symposium*, Anaheim CA, April 30 - May 2, 1991, IEEE Press, pp. 250-255

[44] K.P. Lam, 'Time-range Approximate Reasoning for Microprocessor Systems Design', CICSR Technical Report TR 90-4, The University of British Columnbia, Feb. 1990, 33 pages, accepted for publication, Proc. of IEE, Part E, 1992

[45] K.P. Lam and C.J. Su, 'Inference Network for Optimization Applications', *Proc. of the second Int'l Conf. on Fuzzy logic and Neural Networks*, July 1992, pp. 189-192, Iizuka, Japan

[46] H. Lang, 'Transitive Closure on an Instruction Systolic Array', *Systolic Array Processors, Proc. of Int'l Conference on Systolic Arrays*, 1988

[47] Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart, and Winston, 1976

[48] K.C. Lee, Y. Takefuji, and N. Funabiki, 'A Maximum Neural Network for the Max Cut Problem', *Int'l Joint Conf. on Neural Networks*, July 1991, Seattle, pp. 379-384

[49] K.C. Lee, N. Funabiki and Y. Takefuji, 'A parallel Improvement Algorithm for the Bipartite Subgraph Problem', *IEEE Trans. on Neural Networks*, vol. 3, No. 1, 1992

[50] F.T. Leighton, *Introduction to Parallel Algorithms and Architectures*, Morgan-Kanfman, 1992

[51] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, John Wiley & and Sons Ltd, 1990

[52] P.S. Lewis and S. Y. Kung, 'Dependence Graph Based Design of Systolic Arrays for the Algebraic Path Problem', *Proc. of Ann. Asilomar Conference on Signal, System and Computers*, pp. 13-18, 1986

[53] V. Li, *Knowledge Representation and Problem Solving for an Intelligent Tutor Advisor*, M.A.Sc. thesis, Univ. British Columbia, 1990.

[54] W-M. Lin and V.K. Prasanna, 'Parallel Algorithms and Architectures for Discrete Relaxation Technique', in *IRIS No. 274, Institute for Robotics and Intelligent System, Univ. of Southern California, Los Angeles, California*

[55] A.K. Mackworth, 'Consistency in Network Relations', *Artificial Intelligence*, 8, pp. 99-118, 1977

[56] A.K. Mackworth, 'Constraint Satisfaction', *Encyclopedia of Artificial Intelligence*, Ed. by J. Shapiro, Wiley Publishers, vol. 1, pp. 205-211, 1987

[57] B. Marh, 'Semiring and Transitive Closure', in *Technische Universität Berlin, Fachbereich 20, report 82-5*, 1982

[58] B. Marh, 'Iteration and Summability in Semirings', in *Algebraic and Combinational Methods in Operations Research (Proc. of the Workshop on Algebraic Structure in Operations Research)* Ed. by R.E. Burkard, R.A. Cuninghame-Green, and U. Zimmermann. Ann. Discrete Math. vol. 19, pp. 229-256, 1984

[59] D. Marr, *Vision*, W.H. Freeman, San Francisco, 1982

[60] J.S.B. Mitchell and D.M.Keirsey, 'Planning Strategic Paths thought Variable Terrain Data', *SPIE applications of artificial Intelligence*, Vol. 485, pp. 172-179, 1984

[61] C. Mead, *Analog VLSI and Neural System*, Addison-Wesley, 1989

[62] J.J. Modi, *Parallel Algorithms and Matrix Computation*, Clarendon Press, Oxford, 1988

[63] A. Moffat and T. Takaoka, 'An all pairs shortest path algorithm with expected time $O(n^2 logn)$', *SIAM J. Computing*, 6, pp. 1023-1031, 1987

[64] N.J. Nilsson, *Principles of Artificial Intelligence*, Spinger-Verlag, 1980

[65] F.J. Núñez and M.Valero, 'A Block Algorithm For the Algebraic Path Problem And Its Execution On A Systolic Array', *Systolic Array Processors, Proc. of Int'l Conference on Systolic Arrays*, pp. 265-274, 1988

[66] C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall, Inc., 1982

[67] J. Pearl, *Heuristic: intelligent search strategies for computer problem solving*, Addison-Wesley Pub. Co., 1984

[68] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling, *Numerical Recipes*, Cambridge Univ. Press, 1986

[69] M.J. Quinn, 'Analysis and Implementation of Branch-and-Bound Algorithms on a Hypercube Multicomputer', *IEEE trans. on Computers*, Vol. 39 No. 3, 1990

[70] Y. Robert and D. Trystram: 'Systolic Solution of the Algebraic Path Problem', in *Systolic Arrays: Papers Presented at the First Int'l Workshop on Systolic Arrays*, Ed. W. Moore, A. McCabe and R. Urquhart, 1986

[71] G. Rote, 'A Systolic Array Algorithm for the Algebraic Path Problem (Shortest Paths; Matrix Inversion)', in *Computing* Vol. 34, pp. 191-219

[72] L. Shastri, *Sementic Nets:Evidential Formalization and Its Connectionist Realization*, Morgan-Kaufman, Los Altos/Pitman Pub., London, Feb, 1988

[73] P.M. Spira, 'A New Algorithm for Finding All Shortest Paths in a Graph of Positive Arcs in Average Time $O(n^2 log^2 n)$),' *SIAM J. Comput.* 2 (1973), 28-32

[74] K.B. Sriram and L.M. Patnaik, 'Neural Network Approach for the Two-Dimensional Assignment Problem', *Electronics Letters*, vol. 26, No. 12, 1990, pp. 809

[75] H.S. Stone, *High-Performance Computer Architecture*, Addison-Wesley Pub. Co., 1987

[76] C.J. Su and K.P. Lam, 'Implementation of All-Pair Shortest Path Algorithms with a Binary Relation Inference Network', 10 pages, submitted for publication, February 1991

[77] C.J. Su and K.P. Lam, 'Evaluation of an Inference Network Algorithm for the Shortest Path Problem', 15 pages, submitted for publication, May. 1992

[78] C.J. Su and K.P. Lam, 'A Wavefront Array for the Algebraic Path Problem', 16 pages, submitted for publication, June. 1991

[79] C.J. Su and K.P. Lam, 'Systolic Mapping of the Inference Network for Shortest Path Problems', in *Proc. of Int'l Joint Conf. of Neural Network*, Singapore, Nov., 1991, pp. 917-922

[80] C.J. Su and K.P. Lam, 'Dense Matrix Inversion on Connection Machine', 18 pages, to be submitted.

[81] Y. Tabourier, 'All shortest distances in a graph. An Improvement to Dantzig's Inductive Algorithm', *Discrete Math.*, 4 pp. 83-87 1973

[82] R. Tamassia, J.S. Vitter, 'Optimal parallel algorithms for transitive closure and point location in planar structures', *Proc. of ACM Symposium on Parallel Algorithms and Architectures*, pp. 399-408, 1989

[83] S.C.A. Thomopoulos, L. Zhang, and C.D. Wann, 'Neural Network Inplementation of the Shortest Path Algorithm for Traffic Routing in Communication Networks', in *Proc. of Int'l Joint Conf. on Neural Networks*, vol. 2, pp. 591, 1989

[84] A.A. Toptsis, 'Parallel transitive closure computation in highly scalable multiprocessors', *Proc. of Int'l Conf., Advances in Computing and Information*, pp. 197-206, 1991

[85] Technical Summary or User's Guide of the Connection Machine from the Thinking Machine Corporation, 1991

[86] E. Wacholder, 'A Neural Network Based Optimization Algorithm for the Static Weapon-Target Assignment Problem', *ORSA Journal of Computing*, Vol. 1, No. 4, 1989, pp.232

[87] D.L. Waltz, 'Waltz Filtering', *Encyclopedia of Artificial Intelligence*, Ed. by J. Shapiro, Wiley Publishers, vol. 2, pp. 1162-1165, 1987

[88] B.F. Wang, and G.H. Chen, 'Constant Time Algorithms for the Transitive Closure and Some Related Graph Problems on Processor Array with Reconfigurable Bus Systems', *IEEE Trans. on Parallel and Distributed Systems*, TPDS-1, 4, 1990, pp. 500-507

[89] J.M. Wein, and S. Zenios, 'Massively Parallel Auction Algorithms for Assignment Problem', *1990 Third Symposium On the Frontier of Massively Parallel Computation*, IEEE Computer Society Press, pp. 90-99

[90] G.V. Wilson and G.S. Pawley, 'On the Stability of the Travelling Salesman Problem Algorithm of Hopfield and Tank', *Biological Cybernetics*, vol. 58, pp. 62-70, 1988

[91] Kenny Wu, 'Analog Implementation of a Continuous-Time Inference Network for 5 cities Shortest Path Problems', Elec 475 project report, University of British Columbia, April, 1991

[92] X. Xu and W.T. Tsai, 'Effective Neural Algorithm for the Traveling Salesman Problem', *Neural Networks*, Vol. 4, pp. 193-205, 1991

[93] L. Yang and W.K. Chen, 'An extension of the revised matrix algorithm', *IEEE Proc. Int. Symp. on Circuits and Systems*, pp. 1996-1999 (1989)

[94] J.Y. Yen, 'Finding the Lengths of All Shortest Paths in N- Node Nonnegative-Distance Complete Networks Using $\frac{1}{2}N^3$ Additions and N^3 Comparisons', *J. Assoc. Comput. Mach*, 19 (1972), pp. 423-424

[95] U. Zimmermann, 'Linear and combinational optimization in ordered algebraic structures', (Chapter 8), in *Discrete Math.* Vol. 10, 1981

Appendix A

Data Consistency Checking Using the Inference Network

The inference network can be used to check consistency of a set of relations, derive binary relations from a consistent relation set, infer binary relations from an inconsistent relation set based on the reliability of the given relations. We will consider symmetric relations - $R(i,j) = R(j,i)\ \forall\ i,j$ - since these algorithms are mainly used in time or location referencing applications. Because of the symmetry of the binary relations, the inference network has $N(N-1)/2$ units.

For N nodes, there are up to $N*(N-1)$ binary relations in existence representing the relations between any two of them. The number of independent binary relations required to uniquely define all the binary relations in the network is $(N-1)$. Given some of the binary relations, the number of iterations required for the inference network to obtain all the remaining ones is determined by the structure and consistency of the relations. A set of binary relations can be kept in the inference network, unit (i,j) for relation $R(i,j)$. A set of relation is consistent if the inference network representing the relations is consistent; a network is consistent if for all the units, all site outputs for its different sites are identical to the unit output.

Given inference rules, The consistency of the relations can be checked. Consistency may be declared only when all the units and sites are defined. In checking the consistency,

unit output is evaluated according to:

$$\text{Unit output} = \begin{cases} \text{Previous unit output} & \text{if unit output is defined and} \\ & \text{all site outputs are identical to it.} \\ \text{Site output} & \text{if unit output is undefined and} \\ & \text{site outputs are all identical.} \\ \text{Inconsistent Message} & \text{all other cases.} \end{cases} \tag{A.89}$$

A site output is defined as the deduction of the two inputs from it's incoming links. If no *Inconsistent Message* is obtained after $\lceil \log_2(N-1) \rceil$ iterations, the relation set is consistent. Here N is the number of nodes involved in the relation set.

When the given relations are inconsistent, at some units, sooner or later, the outputs from its $(N-2)$ sites will be different. The network must have a ***conflict resolution mechanism*** in order to proceed further inference. There are various possibilities on conflict resolution, among which some reasonable ones are:

- using an average operator(minimizing high frequency noise),
- using the *max* operator(corresponding to ORing the inference results),
- using the *min* operator(corresponding to ANDing the inference results),
- using a weighted average operator.

Given initial relations and their corresponding certainty factor (reliability, hardness of the constraint), an inference network can

1. arrive at a stable consistent state, with each unit output being identical to all its site outputs;
2. arrive and stay at a metastable inconsistent state, with at least one unit output different from some of its site outputs;

3. become unstable with its degree of inconsistency growing bigger and bigger when the conflict resolution schemes are used.

As an example, we define site and unit functions as:

$$s_{ijl}^{(k+1)} = u_{il}^{(k)} + u_{lj}^{(k)} \tag{A.90}$$

and

$$u_{ij}^{(k)} = (1 - r_{ij}) * av_{ij}^{(k)} + r_{ij} * u_{ij}^{(k-1)} \tag{A.91}$$

in which

$$av_{i,j}^{(k)} = \frac{1}{N-2} \sum_{l=0, l \neq i \neq j}^{N-1} s_{ijl}^{(k)} \tag{A.92}$$

where r_{ij} is the certainty factor of relation $R(i,j)$ (represented by u_{ij}). If $R(i,j)$ is fully constrained, $r_{ij} = 1$. Otherwise, r_{ij} varies within $[0,1)$.

A global energy function can be defined to measure the network consistency. For the inference network involving N nodes ($N \geq 3$), unit (i,j) $(0 \leq j < i \leq N-1)$ is one of the $C(N,2)$ units in the network and site l $(0 \leq l \neq i \neq j \leq N-1)$ is one of the $(N-2)$ sites attached to unit (i,j), the energy function is defined as:

$$E^{(k)} = \sum_{i=1}^{N-1} \sum_{j=0}^{i-1} \sum_{l=0, l \neq i \neq j}^{N-1} \left(s_{ijl}^{(k+1)} - u_{ij}^{(k)} \right)^2 \tag{A.93}$$

where $E^{(k)}$ is the network's global energy after k iterations, $s_{ijl}^{(k+1)}$ is the output of site l at unit (i,j) after $(k+1)$ iterations, $u_{ij}^{(k)}$ is the output of unit (i,j). The value of $s_{ijl}^{(k+1)}$ is the deduction from $u_{il}^{(k)}$ and $u_{lj}^{(k)}$. The value of $u_{ij}^{(k)}$ is the result of conflict resolution operated on its site outputs.

Series $\{E\}$ shows the state of network:

1. The network converges, if $\{E\}$ decreases monotonically;

2. The network diverges, if $\{E\}$ increases monotonically;

3. The network converges to a metastable inconsistent state, if $\{E\}$ converges to a non-zero constant;

4. The network converges to a stable consistent state, if $\{E\}$ converges to zero.

When units are updated asynchronously, the network always converges. Given a single change at any unit (x, y) in a stabilized network, using the definitions (A.93 - A.92), it can be derived that

$$\Delta E^{(k)} = E^{(k)} - E^{(k-1)} = -3 * (N-2) * (1 - r_{xy}^2) * (av_{xy}^{(k)} - u_{xy}^{(k-1)})^2 \leq 0 \qquad \text{(A.94)}$$

'=' holds only when (1) $r_{xy} = 1$, i.e. $R(x, y)$ is fully constrained or (2) $av_{xy}^{(k)} = u_{xy}^{(k-1)}$, i.e. no change.

For synchronous updating, after the k-th iteration, the change of network energy can be derived as:

$$\begin{aligned}
\Delta E^{(k)} &= \sum_{i=1}^{N-1}\sum_{j=0}^{i-1} \Delta E_{ij}^{(k)} + 6 * \sum_{i=2}^{N-1}\sum_{j=1}^{i-1}\sum_{l=0}^{j-1}(d_{ij}^{(k)} d_{jl}^{(k)} + d_{jl}^{(k)} d_{li}^{(k)} + d_{li}^{(k)} d_{jl}^{(k)}) \\
&= \sum_{i=1}^{N-1}\sum_{j=0}^{i-1} -3 * (N-2) * (1 - r_{ij}^2) * (av_{ij}^{(k)} - u_{ij}^{(k-1)})^2 \\
&\quad +6 * \sum_{i=2}^{N-1}\sum_{j=1}^{i-1}\sum_{l=0}^{j-1}(d_{ij}^{(k)} d_{jl}^{(k)} + d_{jl}^{(k)} d_{li}^{(k)} + d_{li}^{(k)} d_{ij}^{(k)}) \qquad \text{(A.95)}
\end{aligned}$$

$\Delta E \leq 0$ does not always hold for general synchronous updating. ΔE can be greater than 0, which means for some given combinations of initial relations and certainty factors, the network may diverge.

Appendix B

Inference Network Algorithm for Matrix Inversion

The inference network algorithm for matrix inversion is developed based on *Crout's* algorithm [68]. The processor array is composed of $N \times N$ PEs connected in a two dimensional grid, as shown in Figure 3.11. There are three types of PEs in the array. Type A, type B, and type C refer to PEs whose coordinates satisfy $i > j$, $i = j$ and $i < j$ respectively. The matrix elements are copied to the processing elements in a pre-designed way so that when data are propagated in the array, the effect are just like the data organization in Figure B.23. It takes $5N - 4$ time units to complete the inversion. Each processing element requires a local memory 'c' to calculate a summation and a control counter 'k' to trigger and complete the calculation and propagation. Initially, the control counter for PE $[i, j]$ is set to $k := -i - j$. Algorithm 5 gives the detail. Result of the inversion is kept in $c[i, j]$ after $5N - 4$ time units. For more in-depth discussion about an $N \times N$ systolic array for matrix inversion, see author's paper [80, 78].

Algorithm 5 : *Matrix Inversion on the CM:*

BEGIN PARALLEL for all i, j *(initialization)*

$h[i,\ (-i-j) \pmod N] := v[(-i-j) \pmod N,\ j] := a_{ij}$

$k[i,j] := -i-j \qquad c[i,j] := 0$

END PARALLEL

FOR count $= 1$ *TO count* $= 5N-4$ *DO*

BEGIN PARALLEL for all i, j

$h[i,j] := h[i,\ (j-1) \pmod N] \qquad v[i,j] := v[(i-1) \pmod N,\ j]$

$k[i,j] := k[i,j] + 1$

IF $(0 \leq k[i,j] < \min\{i,j\})$

  $c[i,j] := c[i,j] + h[i,j] * v[i,j]$

ELSE IF $(k[i,j] = \min\{i,j\})$

  IF $(type = A)$ $h[i,j] := c[i,j] := (h[i,j] - c[i,j])/v[i,j]$

  ELSE IF $(type = B)$ $h[i,j] := v[i,j] := c[i,j] := v[i,j] - c[i,j]$

  ELSE $v[i,j] := c[i,j] := v[i,j] - c[i,j]$

ELSE IF $(k[i,j] = \max\{i,j\})$

  IF $(type = A)$ $v[i,j] := c[i,j]$

  ELSE IF $(type = C)$ $h[i,j] := c[i,j]$

ELSE IF $(k[i,j] = N + \min\{i,j\})$

  IF $(type \triangleq B)$ $h[i,j] := v[i,j] := c[i,j] := 1/v[i,j]$

  ELSE IF $(type = C)$ $c[i,j] := h[i,j] * v[i,j]$

ELSE IF $(N + \min\{i,j\} < k[i,j] < N + \max\{i,j\})$

  $c[i,j] := c[i,j] + h[i,j] * v[i,j]$

ELSE IF $(k[i,j] = N + \max\{i,j\})$

  IF $(type = A)$ $v[i,j] := c[i,j] := -c[i.j]$

  ELSE IF $(type = C)$ $h[i,j] := c[i,j] := -c[i.j]/v[i,j]$

ELSE IF $(k[i,j] = 2N + \min\{i,j\})$

  IF $(type = A)$ $h[i,j] := c[i,j]$

  ELSE IF $(type = C)$ $v[i,j] := c[i,j]$

ELSE IF $(k[i,j] = 2N + \max\{i,j\})$

  IF $(type = A)$ $c[i,j] := h[i,j] * v[i,j]$

ELSE IF $(2N + \max\{i,j\}) < k[i,j] < 3N)$

  $c[i,j] := c[i,j] + h[i,j] * v[i,j]$

END IF

END PARALLEL

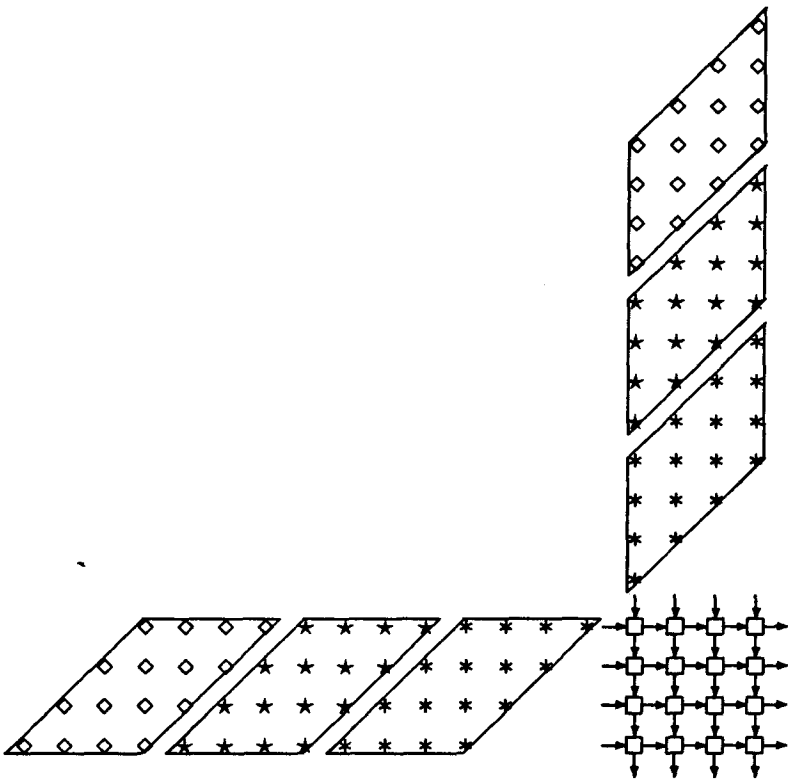


Figure B.23: Data flow in calculating $\mathbf{A}^{-1}$. Each parallelogram contains data for one phase. $*$ is an element in the original matrix; $\star$ is an element in $\mathbf{L}$ or $\mathbf{U}$; $\diamond$ is an element in $\mathbf{L}^{-1}$ or $\mathbf{U}^{-1}$.

Appendix C

A Weightless Neural Network for Consistent Labeling Problems

The objective of a consistency labeling problem is to label a set of N objects using a set of M labels which satisfy some compatibility constraints. Only unary or binary constraints are considered here. A unary constraint prohibits an object to have a certain label. A binary constraint restricts the combination of the labels for two objects. A weightless neural network is proposed here to eliminate all inconsistent labels for each object based on unary and binary constraints.

The inconsistency elimination algorithm is based on the sequential algorithm discussed in [54]. Unary constraints are used to determine candidate labels for each object. Binary constraints are used to set up initial restriction of labels between object pairs. These restrictions are passed to other related object pairs and subsequently reduce candidate labels for objects.

The proposed neural network consists of $\frac{N(N+1)}{2}$ clusters of neurons, denoted as $G(i,j)$ $(1 \leq j \leq i \leq N)$. Each cluster $G(i,j)$ with $i > j$ detects inconsistent labels between object i and j. Initial unary and binary relations are used to set up initial states of the nuerons in cluster $G(i,i)$ and $(G(i,j)$ (for all i and j) respectively. If any inconsistent label is found for object i (or j) in cluster $G(i,j)$, a signal is sent to cluster $G(i,i)$ (or $G(j,j)$). When cluster $G(i,i)$ receives the signal indicating an inconsistent label for object i, it activates $G(i,j)$ $(j = 1,2,\cdots,i-1)$ and $G(j,i)$ $(j = i+1, i+2,\cdots,N)$ to eliminate the inconsistent label from the candidate labels for object i.

There are M independent neurons in cluster $G(i,i)$. The initial state of neuron m

($1 \leq m \leq M$) is determined by the unary constraints for object i. If l_m is a unary-consistent label for object i, neuron m in cluster $G(i,i)$ is initially set 'on'. Neuron m will turn off if label l_m is later found inconsistent for object i. If all neurons in a cluster $G(i,i)$ are off, the labeling process will be terminated since there is no consistent label for object i. When all the inconsistent labels are eliminated, if neuron m in cluster $G(i,i)$ is on, object i can be labeled l_m.

Each neuron cluster $G(i,j)$ ($i > j$) consists of two layers of neurons. The lower layer has $M \times M$ neurons. Neuron (p,q) ($1 \leq p,q \leq M$) has two inputs from neuron p in cluster $G(i,i)$ and neuron q in cluster $G(j,j)$. Neuron (p,q) is initially off if simultaneous labeling of l_p for object i and l_q for object j is prohibited by a binary constraint. After the network starts to update, neuron (p,q) remains 'on' if and only if both neuron p in cluster $G(i,i)$ and neuron q in cluster $G(j,j)$ are on. The upper layer has $2M$ neurons, each has M inputs from a row (or a column) of neurons in the lower layer and an output to a neuron in cluster $G(i,i)$ (or $G(j,j)$). If all neurons in a row (or a column) in the lower layer are off, the corresponding neuron in the upper layer will be turned off, and a signal will be sent to cluster $G(i,i)$ (or $G(j,j)$) indicating an inconsistent label for object i (or j).

The network will settle down when all inconsistent labels are eliminated. No oscillation can occur since neurons can only switch from 'on' to 'off' after initialization, or remain in an 'on' state or an 'off' state. The correctness and completeness of the elimination process is ensured by the corresponding sequential algorithm [54]. Links in the network only transmit an on/off state, therefore, all link weights can be set to one.

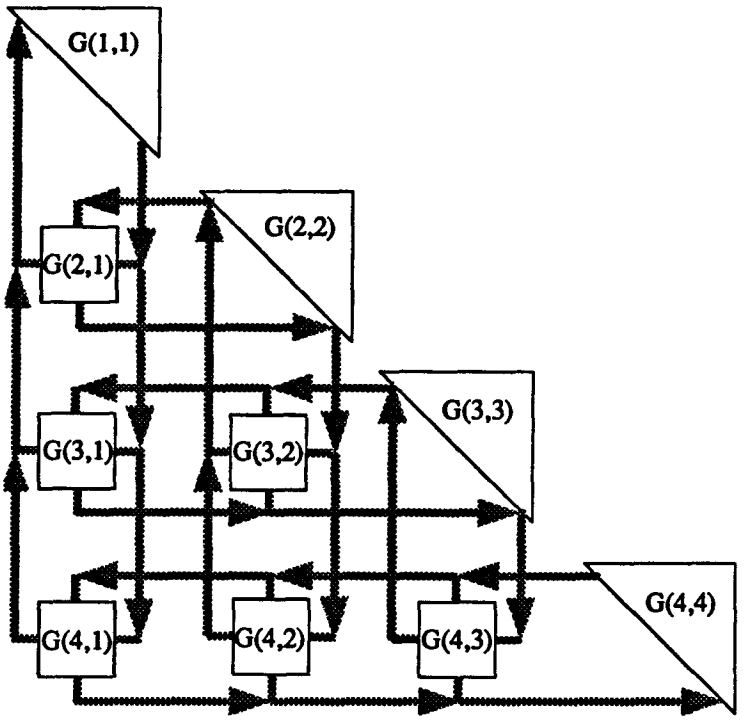


Figure C.24: Interconnected clusters form a weightless neural network for consistency labeling problems.

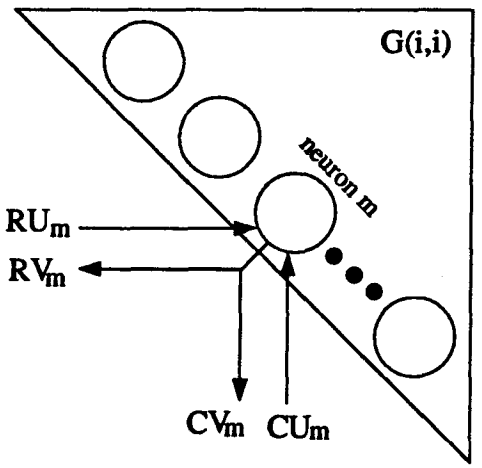


Figure C.25: A diagonal cluster $G(i,i)$ in the weightless neural network.

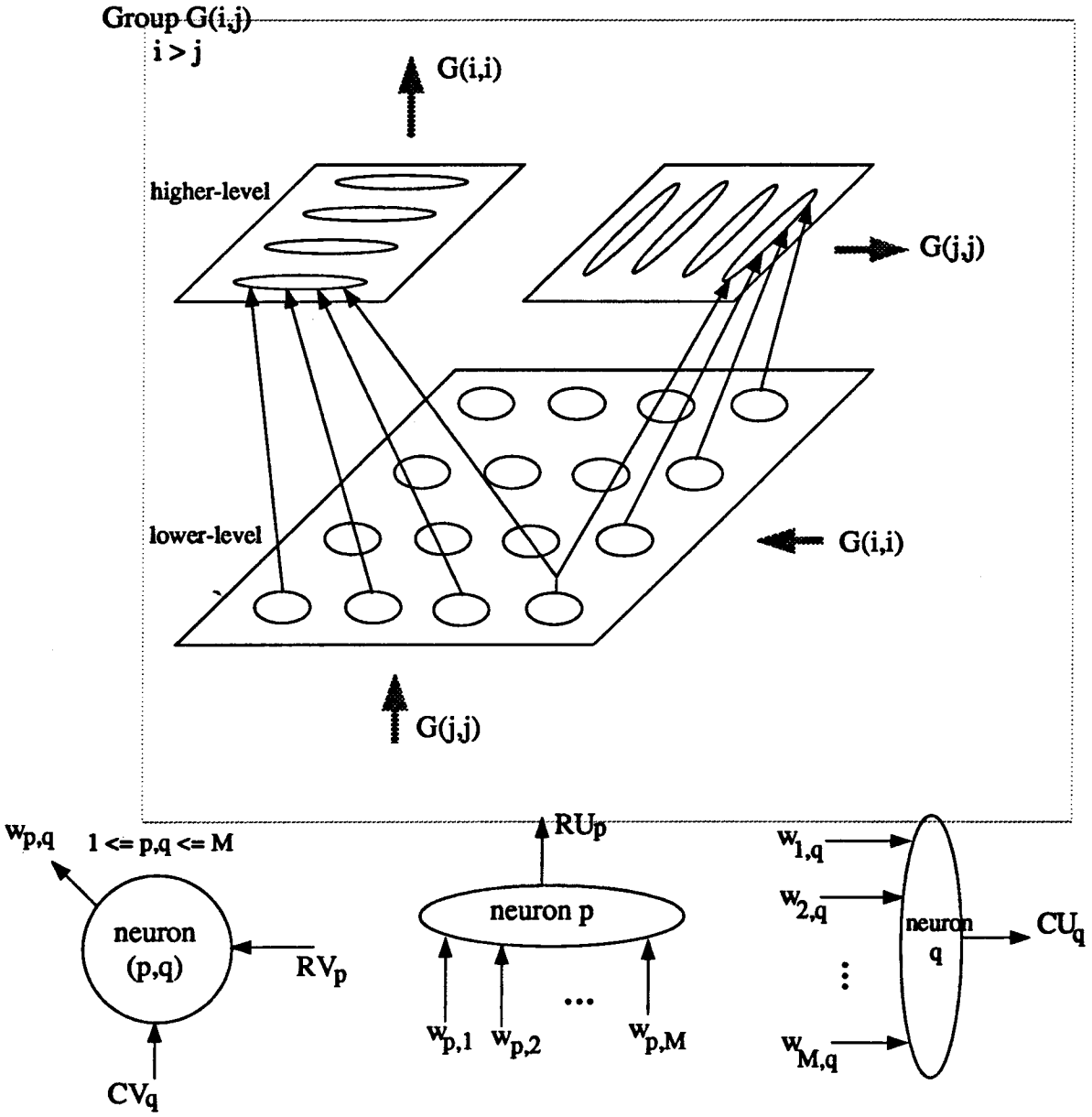


Figure C.26: A non-diagonal cluster $G(i,j)$ $(i > j)$ in the weightless neural network.

Appendix D

Algorithms Related with the Shortest Path Problem

D.1 Mapping the Shortest Path Algorithm onto the Connection Machine

Chapter 3 has shown that the Connection Machine is an efficient platform available to map the inference network algorithm on to. After initializing the processor array according to Eq.(4.13), the shortest paths can be obtained in at most $\lceil \log_2(N-1) \rceil$ iterations, or, $N\lceil \log_2(N-1) \rceil$ time units. As discussed in Section 3.3, each iteration consists of updating the data flows, evaluating site values, and calculating unit values. Mapped onto the CM, INFERENCE becomes:

Algorithm 6 : *Shortest Path on the CM (to be referred to as* INFERENCE-CM*)*

for all PE, $u[i,j] := u^{(0)}(i,j)$

count $:= \lceil log_2(N-1) \rceil$

while (count > 0*)*

BEGIN

BEGIN PARALLEL for all i, j

$h[i,(-i-j)mod(N)] := u[i,j]$, $v[(-i-j)mod(N),j] := u[i,j]$

$u_{old}[i,j] := u[i,j]$

FOR $k = 0$ *TO* $k = N-1$

BEGIN

update: $u[i,j] := min\{u[i,j], h[i,j] + v[i,j]\}$

propagate: $h[i,j] := h[i,(j-1)mod(N)]$ $v[i,j] := v[(i-1)mod(N),j]$

END

END PARALLEL

if $\forall i, j \;\; u[i,j] = u_{old}[i,j]$, *break;*

$count := count - 1$

END

It is not difficult to prove the equivalence of INFERENCE-CM and INFERENCE by comparing the corresponding unit values after each iterations.

D.2 Unifying Some Established Algorithms Using an Inference Network

All-pair shortest path algorithms can be implemented by defining proper unit and site functions in an inference network. In the following, we shall describe several algorithms and derive their inference network implementation for finding all shortest paths in a general directed graph.

D.2.1 Floyd's Algorithm

Floyd's [17, 11] algorithm (FLOYD) constructs optimal paths by inserting nodes when appropriate, into more direct paths. After k iterations, function $f_k(i,j)$ evaluates to the optimal path length between i and j, where the intermediate nodes belong to the set of nodes $\{1, 2, \ldots, k\}$. Its recurrent relation is

$$f_k(i,j) = min\{f_{k-1}(i,j), f_{k-1}(i,k) + f_{k-1}(k,j)\} \tag{D.96}$$

with boundary condition $f_0(i,j) = d_{ij}$. The total number of basic operations required is $2N^3$.

Since FLOYD requires $f_k(i,i)$ to detect negative cycles, its inference network implementation needs N^2 units. The unit and site functions are defined as:

$$u^{(k)}(i,j) = min\{u^{(k-1)}(i,j), s^{(k)}(i,j,k)\} \quad \text{(D.97)}$$

$$s^{(k)}(i,j,k) = u^{(k-1)}(i,k) + u^{(k-1)}(k,j) \quad \text{(D.98)}$$

Initial unit values are $u^{(0)}(i,j) = d_{ij}$. When there is no negative loop, the network settles down after $k = N$, and $u^{(N)}(i,j)$ gives the shortest path length from node i to node j, and diagonal units (i,i) have zero values. In the presence of negative loops, there will be at least one 'diagonal' unit having a negative value. Iteration has to stop after exactly N steps, in order to obtain shortest paths which do not traverse the same arc for more than once. Compared with INFERENCE, this algorithm only needs to update one *specific* site (indexed by k) at each unit for each iteration; but it takes N instead of $n \leq \lceil log_2(N-1) \rceil$ iterations to find all the shortest paths.

D.2.2 Matrix Algorithms

Matrix algorithm (MATRIX) defines a matrix binary operation $\otimes$ called minaddition:

$$\mathbf{W} = \mathbf{A} \otimes \mathbf{B} = [w_{ij} = \min_k (a_{ik} + b_{kj})] \quad \text{where } \mathbf{A} = [a_{ij}],\ \mathbf{B} = [b_{ij}] \quad \text{(D.99)}$$

If $\mathbf{D^1} = \mathbf{D}$ is the length matrix $[d_{ij}]$, then the elements of $\mathbf{D}^N$ gives the shortest path lengths, where $\mathbf{D}^N$ is the minaddition of N $\mathbf{D}$'s:

$$\mathbf{D}^N = \mathbf{D} \otimes \mathbf{D} \otimes \mathbf{D} \otimes \cdots \otimes \mathbf{D} \otimes \mathbf{D} \quad \text{(D.100)}$$

This algorithm is valid when there are no negative loops in the network. Minaddition is associative, for example,

$$\mathbf{D}^4 = \mathbf{D} \otimes \mathbf{D} \otimes \mathbf{D} \otimes \mathbf{D} = \mathbf{D}^2 \otimes \mathbf{D}^2 \quad \text{(D.101)}$$

Eq.(4.14-4.15) uses this property to make the inference network converge in no more than $\lceil log_2(N-1) \rceil$ iterations. In fact, the INFERENCE algorithm is intimately related with the MATRIX algorithm, although the former is derived directly upon an inference network platform. It is not difficult to observe that, if we define another matrix binary operation $\perp$ as

$$\mathbf{W} = \mathbf{A} \perp \mathbf{B} = [w_{ij} = min\{a_{ij}, b_{ij}\}] \quad \text{where } \mathbf{A} = [a_{ij}],\ \mathbf{B} = [b_{ij}] \tag{D.102}$$

then Eq.(4.14-4.15) can effectively be represented as

$$\mathbf{D}^{(k)} = \mathbf{D}^{(k-1)} \perp (\mathbf{D}^{(k-1)} \otimes \mathbf{D}^{(k-1)}) \tag{D.103}$$

Revised matrix algorithm (RMATRIX) updates matrix elements asynchronously to avoid unnecessary operations. Two distinct processes are defined: (i) a forward process

$$d_{ij}^{(f)} = \min_{1 \le l \le N} \{d_{il}^{(p)} + d_{lj}^{(q)}\} \quad \text{where } p = \begin{cases} 1 & j \le l \\ f & j > l \end{cases} \quad q = \begin{cases} 1 & i \le l \\ f & i > l \end{cases} \tag{D.104}$$

and (ii) a backward process

$$d_{ij}^{(b)} = \min_{1 \le l \le N} \{d_{il}^{(p)} + d_{lj}^{(q)}\} \quad \text{where } p = \begin{cases} f & l \le j \\ b & l > j \end{cases} \quad q = \begin{cases} f & l \le i \\ b & l > i \end{cases} \tag{D.105}$$

[93] shows that the shortest paths can be obtained by one forward process followed by on backward process, one backward process followed by one forward process, three forward processes, or three backward processes. So the total number of basic operations required is $4N(N-1)(N-2)$ or $6N(N-1)(N-2)$.

The inference network for RMATRIX contains $N(N-1)$ units (diagonal units are not necessary). The matrix elements are updated in an order shown in Figure D.27. Site function for both forward and backward processes are the same:

$$s^{(k)}(i,j,l) = u^{(k-1)}(i,l) + u^{(k-1)}(l,j) \tag{D.106}$$

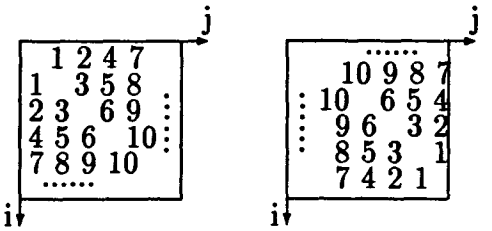


Figure D.27: The order of updating length matrix in a forward (left) and backward (right) process

Unit functions for forward process is:

$$u^{(k)}(i,j) = \begin{cases} min\{u^{(k-1)}(i,j), \underset{1 \le l \le N, l \ne i, l \ne j}{min}\{s^{(k)}(i,j,l)\}\} & \text{when } k = f(i,j) \\ u^{(k-1)}(i,j) & \text{otherwise} \end{cases} \tag{D.107}$$

where

$$f(i,j) = \frac{max\{i,j\}(max\{i,j\}-1)}{2} + min\{i,j\} \tag{D.108}$$

Unit functions for backward process is:

$$u^{(k)}(i,j) = \begin{cases} min\{u^{(k-1)}(i,j), \underset{1 \le l \le N, l \ne i, l \ne j}{min}\{s^{(k)}(i,j,l)\}\} & \text{when } k = b(i,j) \\ u^{(k-1)}(i,j) & \text{otherwise} \end{cases} \tag{D.109}$$

where

$$b(i,j) = \frac{max\{N-i, N-j\}(max\{N-i, N-j\}+1)}{2} + min\{N-i, N-j\} + 1 \tag{D.110}$$

The network updates one or two units at a time. To obtain all the shortest paths, the units of inference network have to be updated in a specific order for $N(N-1)$ (one forward and one backward process) or $\frac{2}{3}N(N-1)$ iterations (three forward or three backward).

D.2.3 Dantzig's Algorithm

Dantzig's algorithm [9] (DANTZIG) constructs shortest paths using arcs whose nodes numbered between 1 and k for the k-th iteration. $g_k(i,j)$ $(i \leq k, j \leq k)$, the shortest path length from i to j in the k-th iteration, is obtained from $g_{k-1}(i,j)$, d_{ik} and d_{kj}. [11] gives the algorithm in the recurrent form:

$$g_k(i,k) = \min_{1 \leq l \leq k-1} \{g_{k-1}(i,l) + g_0(l,k)\} \quad (i = 1,2,\ldots,k-1), \tag{D.111}$$

$$g_k(k,j) = \min_{1 \leq l \leq k-1} \{g_0(k,l) + g_{k-1}(l,j)\} \quad (j = 1,2,\ldots,k-1), \tag{D.112}$$

$$g_k(k,k) = min\{0, \min_{1 \leq l \leq k-1} \{g_k(k,l) + g_k(l,k)\}\} \tag{D.113}$$

$$g_k(i,j) = min\{g_{k-1}(i,j), g_k(i,k) + g_k(k,j)\} \quad (i,j = 1,2,\ldots,k-1) \tag{D.114}$$

where $g_0(i,j) = d_{ij}$ for all i and j. The total number of basic operations required is $2N^2(N-1)$.

The structure of inference network for DANTZIG is the same as that for FLOYD, which has N^2 units. The unit function for DANTZIG is

$$u^{(2k+1)}(i,j) = \begin{cases} min\{u^{(2k)}(i,j), \min_{1 \leq l \leq k-1} \{s^{(2k+1)}(i,j,l)\}\} & j < i = k \text{ or } i < j = k \\ u^{(2k)}(i,j) & \text{otherwise} \end{cases} \tag{D.115}$$

$$u^{(2k+2)}(i,j) = \begin{cases} min\{u^{(2k+1)}(i,j), s^{(2k+2)}(i,j,k)\} & i < k \text{ and } j < k \\ min\{0, \min_{1 \leq l \leq k-1} \{s^{(2k+2)}(i,j,l)\}\} & i = j = k \\ u^{(2k+1)}(i,j) & \text{otherwise} \end{cases} \tag{D.116}$$

where $2 \leq k \leq N$, $u^{(4)}(i,j) = d_{ij}$, and $s^{(r)}(i,j,l) = u^{(r-1)}(i,l) + u^{(r-1)}(l,j)$ $\forall$ $5 \leq r \leq 2N+2$ In the $(2k+1)$-th iteration, $2(k-1)$ units are updated, $k-1$ sites at each unit are involved in the minimization; in the $(2k+2)$-th iteration, a total number of $(k-1)^2+1$ units with a total of $k(k-1)$ sites are minimized. The total number of synchronous iterations required is $2(N-1)$.

Modified Dantzig's algorithm [81] (MDANTZIG) avoids unnecessary operations by further decomposing the first two recurrent relations into $k-1$ recurrent steps. Define $g_k^0(i,k) = g_k^0(k,j) = \infty$, $g_k(i,k) = g_k^{k-1}(i,k)$ and $g_k(k,j) = g_k^{k-1}(k,j)$ can be obtained from:

$$g_k^l(i,k) = \begin{cases} g_k^{l-1}(i,k) & \forall 1 \le i \le k-1 \quad d_{lk} \ge g_k^{l-1}(l,k) \\ min\{g_k^{l-1}(i,k), g_{k-1}(i,l) + d_{lk}\} & \forall 1 \le i \le k-1 \quad d_{lk} < g_k^{l-1}(l,k) \end{cases} \tag{D.117}$$

$$g_k^l(k,j) = \begin{cases} g_k^{l-1}(k,j) & \forall 1 \le j \le k-1 \quad d_{kl} \ge g_k^{l-1}(k,l) \\ min\{g_k^{l-1}(k,j), d_{kl} + g_{k-1}(l,j)\} & \forall 1 \le j \le k-1 \quad d_{kl} < g_k^{l-1}(k,l) \end{cases} \tag{D.118}$$

A significant amount of operations can be saved when $d_{lk} \ge g_k^{l-1}(l,k)$ or $d_{kl} \ge g_k^{l-1}(k,l)$ is satisfied.

Corresponding to the $(2k+1)$-th iteration in DANTZIG, the inference network implementation for MDANTZIG requires $k-1$ iterations. In each of these iterations, $2(k-1)$ units are updated; and at each of these units, only one site is activated. So MDANTZIG requires a total number of $\frac{1}{2}(N+2)(N-1)$ iterations.

Appendix E

Determinants and Eigenvalues

E.1 The Determinants of Some Matrices

The following matrix are all $n \times n$.

$$\begin{vmatrix} a & b & b & \cdots & b \\ b & a & b & \cdots & b \\ b & b & a & \cdots & b \\ & \cdots & & \cdots & b \\ b & b & b & \cdots & a \end{vmatrix} = \begin{vmatrix} a & b-a & b-a & \cdots & b-a \\ b & a-b & 0 & \cdots & 0 \\ b & 0 & a-b & \cdots & 0 \\ & & \cdots & & 0 \\ b & 0 & 0 & \cdots & a-b \end{vmatrix}$$

$$= a(a-b)^{n-1} - b(b-a)(a-b)^{n-2}(n-1)$$

$$= (a+(n-1)b)(a-b)^{n-1}$$

$$\begin{vmatrix} s_1 & a_1 & a_1 & \cdots & a_1 \\ a_2 & s_2 & a_2 & \cdots & a_2 \\ a_3 & a_3 & s_3 & \cdots & a_3 \\ & \cdots\cdots\cdots & & & \\ a_n & a_n & a_n & \cdots & s_n \end{vmatrix} = \begin{vmatrix} s_1-a_1 & 0 & 0 & \cdots & a_1 \\ 0 & s_2-a_2 & 0 & \cdots & a_2 \\ 0 & 0 & s_3-a_3 & \cdots & a_3 \\ & & \cdots\cdots\cdots & & \\ a_n-s_n & a_n-s_n & a_n-s_n & \cdots & s_n \end{vmatrix}$$

$$= s_n \prod_{i=1}^{n-1}(s_i-a_i) + \sum_{j=1}^{n-1} a_j \prod_{i=1, i\neq j}^{n} (s_i-a_i)$$

$$= \prod_{i=1}^{n}(s_i-a_i) + \sum_{j=1}^{n} a_j \prod_{i=1, i\neq j}^{n} (s_i-a_i)$$

$$\begin{vmatrix} s-a_1 & -a_1 & -a_1 & \cdots & -a_1 \\ -a_2 & s-a_2 & -a_2 & \cdots & -a_2 \\ -a_3 & -a_3 & s-a_3 & \cdots & -a_3 \\ & & \cdots\cdots\cdots & & \\ -a_n & -a_n & -a_n & \cdots & s-a_n \end{vmatrix} = \begin{vmatrix} s-a_1 & 0 & 0 & \cdots & a_1 \\ 0 & s-a_2 & 0 & \cdots & a_2 \\ 0 & 0 & s-a_3 & \cdots & a_3 \\ & & \cdots\cdots\cdots & & \\ -s_n & -s_n & -s_n & \cdots & s-a_n \end{vmatrix}$$

$$= (s-a_n)s^{n-1} + \sum_{i=1}^{n} ss^{n-2}a_i$$

$$= (s - \sum_{i=1}^{n} a_i)s_i^{n-1}$$

E.2 Eigenvalues of the Inference Network Connection Matrix

Let $\mathbf{P}$ be an $n^2 \times n^2$ symmetric matrix. Its element at row $iN+j$ and column $rN+s$ is

$$p_{ij,rs} = u\delta(i-r)\delta(j-s) - v\delta(i-r) - v\delta(j-s) \quad u, v \neq 0$$

where

$$\delta(x) = \begin{cases} 1 & x = 0 \\ 0 & x \neq 0 \end{cases}$$

Matrix $\mathbf{P}$ can be divided into n^2 $n \times n$ submatrix:

$$\mathbf{P} = \begin{vmatrix} \mathbf{P}_{11} & \mathbf{P}_{12} & \mathbf{P}_{13} & \cdots & \mathbf{P}_{1n} \\ \mathbf{P}_{21} & \mathbf{P}_{22} & \mathbf{P}_{23} & \cdots & \mathbf{P}_{2n} \\ \mathbf{P}_{31} & \mathbf{P}_{32} & \mathbf{P}_{33} & \cdots & \mathbf{P}_{3n} \\ & & \cdots\cdots\cdots & & \\ \mathbf{P}_{n1} & \mathbf{P}_{n2} & \mathbf{P}_{n3} & \cdots & \mathbf{P}_{nn} \end{vmatrix}$$

$$\text{where } \mathbf{P}_{ii} = \begin{vmatrix} -u-2v & -v & -v & \cdots & -v \\ -v & -u-2v & -v & \cdots & -v \\ -v & -v & -u-2v & \cdots & -v \\ & & \cdots\cdots\cdots & & \\ -v & -v & -v & \cdots & -u-2v \end{vmatrix} \quad \forall\, 1 \le i \le n$$

$$\text{and } \mathbf{P}_{ij} = \begin{vmatrix} -v & 0 & 0 & \cdots & 0 \\ 0 & -v & 0 & \cdots & 0 \\ 0 & 0 & -v & \cdots & 0 \\ & & \cdots\cdots\cdots & & \\ 0 & 0 & 0 & \cdots & -v \end{vmatrix} \quad \forall\, 1 \le i \ne j \le n$$

$$|s\mathbf{I} - \mathbf{P}| = \begin{vmatrix} s_1 & v & \cdots & v & v & 0 & \cdots & 0 & \cdots & v & 0 & \cdots & 0 \\ v & s_1 & \cdots & v & 0 & v & \cdots & 0 & \cdots & 0 & v & \cdots & 0 \\ & & \cdots & & & & \cdots & & \cdots & & \cdots & & \\ v & v & \cdots & s_1 & 0 & 0 & \cdots & v & \cdots & 0 & 0 & \cdots & v \\ \\ v & 0 & \cdots & 0 & s_1 & v & \cdots & v & \cdots & v & 0 & \cdots & 0 \\ 0 & v & \cdots & 0 & v & s_1 & \cdots & v & \cdots & 0 & v & \cdots & 0 \\ & & \cdots & & & & \cdots & & \cdots & & \cdots & & \\ 0 & 0 & \cdots & v & v & v & \cdots & s_1 & \cdots & 0 & 0 & \cdots & v \\ & & & & \cdots & & \cdots & & \cdots & & & & \\ v & 0 & \cdots & 0 & v & 0 & \cdots & 0 & \cdots & s_1 & v & \cdots & v \\ 0 & v & \cdots & 0 & 0 & v & \cdots & 0 & \cdots & v & s_1 & \cdots & v \\ & & \cdots & & & & \cdots & & \cdots & & \cdots & & \\ 0 & 0 & \cdots & v & 0 & 0 & \cdots & v & \cdots & v & v & \cdots & s_1 \end{vmatrix}$$

$$
= \begin{vmatrix}
s_2 & v & \cdots & v & s_2 & v & \cdots & v & \cdots & s_2 & v & \cdots & v \\
v & s_2 & \cdots & v & v & s_2 & \cdots & v & \cdots & v & s_2 & \cdots & v \\
 & & \cdots & & & & \cdots & & \cdots & & & \cdots & \\
v & v & \cdots & s_2 & v & v & \cdots & s_2 & \cdots & v & v & \cdots & s_2 \\
 & & & & & & & & & & & & \\
v & 0 & \cdots & 0 & s_1 & v & \cdots & v & \cdots & v & 0 & \cdots & 0 \\
0 & v & \cdots & 0 & v & s_1 & \cdots & v & \cdots & 0 & v & \cdots & 0 \\
 & & \cdots & & & & \cdots & & \cdots & & & \cdots & \\
0 & 0 & \cdots & v & v & v & \cdots & s_1 & \cdots & 0 & 0 & \cdots & v \\
 & & & & \cdots & & \cdots & & \cdots & & & & \\
v & 0 & \cdots & 0 & v & 0 & \cdots & 0 & \cdots & s_1 & v & \cdots & v \\
0 & v & \cdots & 0 & 0 & v & \cdots & 0 & \cdots & v & s_1 & \cdots & v \\
 & & \cdots & & & & \cdots & & \cdots & & & \cdots & \\
0 & 0 & \cdots & v & 0 & 0 & \cdots & v & \cdots & v & v & \cdots & s_1
\end{vmatrix}
$$

$$
= \begin{vmatrix}
s_2 & v & \cdots & v & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 & \cdots & 0 \\
v & s_2 & \cdots & v & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 & \cdots & 0 \\
 & & \cdots & & & & \cdots & & \cdots & & \cdots & & \\
v & v & \cdots & s_2 & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 & \cdots & 0 \\
 & & & & & & & & & & & & \\
v & 0 & \cdots & 0 & s_3 & v & \cdots & v & \cdots & 0 & 0 & \cdots & 0 \\
0 & v & \cdots & 0 & v & s_3 & \cdots & v & \cdots & 0 & 0 & \cdots & 0 \\
 & & \cdots & & & & \cdots & & \cdots & & \cdots & & \\
0 & 0 & \cdots & v & v & v & \cdots & s_3 & \cdots & 0 & 0 & \cdots & 0 \\
 & & & & & & & & & & & & \\
 & & & & \cdots & & \cdots & & \cdots & & & & \\
 & & & & & & & & & & & & \\
v & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & \cdots & s_3 & v & \cdots & v \\
0 & v & \cdots & 0 & 0 & 0 & \cdots & 0 & \cdots & v & s_3 & \cdots & v \\
 & & \cdots & & & & \cdots & & \cdots & & \cdots & & \\
0 & 0 & \cdots & v & 0 & 0 & \cdots & 0 & \cdots & v & v & \cdots & s_3
\end{vmatrix}
$$

$$
= \begin{vmatrix}
s_2 & v & \cdots & v \\
v & s_2 & \cdots & v \\
 & \cdots & \cdots & \\
v & v & \cdots & s_2
\end{vmatrix}
\begin{vmatrix}
s_1 & v & \cdots & v \\
v & s_1 & \cdots & v \\
 & \cdots & \cdots & \\
v & v & \cdots & s_1
\end{vmatrix}^{n-1} \tag{E.119}
$$

where $s_1 = s + u + 2v$, $s_2 = s + u + (n+1)v$, and $s_3 = s + u + v$, using the determinants derived in Appendix E.1,

$$|s\mathbf{I} - \mathbf{P}| = (s + u + 2nv)(s + u)^{(n-1)^2}(s + u + nv)^{2(n-1)}$$

P has three distinct eigenvalues:

$\lambda_1 = -(u + 2nv)$, its corresponding eigenvector is

$$\mathbf{e}_1 = (1, 1, \cdots, 1)^t \quad or \quad \hat{\mathbf{e}}_1 = (\frac{1}{N}, \frac{1}{N}, \cdots, \frac{1}{N})^t$$

$\lambda_2 = -u$, its $(n-1)^2$ eigenvectors form a $(n-1)^2$-dimensional subspace;

$\lambda_3 = -(u + nv)$, its $2(n-1)$ eigenvectors form a $2(n-1)$-dimensional subspace. Since $\lambda_1 \neq \lambda_2 \neq \lambda_3$, $\mathbf{e}_1 \perp \Re_2 \perp \Re_3$.