INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®

LOCALIZED ADAPTIVE NETWORKS FOR HYBRID SYMBOLIC AND SUBSYMBOLIC PROCESSING

by
YOUSUF C. H. MA
B.S., King Abdul Aziz University, 1987
M.S., University of Central Florida, 1991

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the School of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Summer Term
2000

Major Professor: Ronald F. DeMara

UMI Number: 9977820

Bell & Howell Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

ABSTRACT

The *Localized Self-Contained Adaptive Networks* (LSCAN) representation strategy is developed to integrate advantages of symbolic and subsymbolic approaches while supporting scalable distributed processing techniques. Although symbolic processing systems have been successfully implemented for several decades, the problems of ambiguity, brittle decision-making, and low availability parallelism have encouraged development of various subsymbolic approaches. However, subsymbolic representations lack variable binding capabilities and make symbolic composition and decomposition difficult. To address these deficiencies, a localized decision-making approach is designed and evaluated which integrates beneficial features from both paradigms. In LSCAN, the fundamental structures of entity nodes, AND evaluators, OR evaluators, and lateral links are used to allow more direct incorporation of cognitive rules while retaining the learning capabilities inherent in subsymbolic approaches. First, processing nodes are defined for constant, variable, functional, and decision assertive entities. Next, reasoning mechanisms are developed using scaled and shifted sigmoid evaluation functions to generate excitatory, inhibitory, and synchronization interactions among processing nodes. The evaluation functions can be readily adapted and tuned using supervised or unsupervised learning. It is shown that these structures are capable of performing classification and filter switching tasks by means of several

examples. The structures are then cascaded using weighted links to propagate belief factors defined by thresholding expressions derived through the training process.

LSCAN was applied extensively to two case studies to evaluate its performance relative to previous approaches using a network manager implemented in the C++ language. In the first case study, LSCAN is employed to classify hand-written digits provided in the NIST benchmark image database. Both pattern-based and rule-guided classification methods are developed and compared to previous approaches. The number of classes required to achieve desired levels of recognition accuracy is evaluated for numerous parameters of feature orientation, feature strength, pattern threshold, and bias values when using methods such as Gaussian feature extraction. In the second case study, LSCAN is used to implement routing protocols capable of handling dynamic network configurations in the context of the Border Gateway Protocol 4.0 (BGP4) methodology. It is shown that localized decision-making in LSCAN can provide a flexible, distributed processing approach to the integration of rule-guided techniques with self-adaptive representations.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

It is widely recognized that symbolic processing more directly emulates high-level human cognitive processes. Inputs to symbolic processing systems are typically representations in the form of comprehensible character strings. Hence, real world information at cognitive levels is more readily represented and handled. On the other hand, connectionist, or subsymbolic processing, replaces fixed symbols with dynamic of numerical values, and processes those numerical patterns among connected simple processing nodes. Each node constructs an output based on some contribution of its total input strengths. Connections between nodes are associated with numerical weights which can be adjusted through a systematic learning algorithm. Connectionist models provide advantages of learning, handling incomplete information, and parallel processing. However, inputs to subsymbolic processing systems consist of numerical data which are low-level representations not directly discernible nor related to human reasoning processes.

Symbolic and subsymbolic paradigms are distinguished by three major characteristics [7]: (1) the type of representation, (2) the style of composition, and (3) the paradigm's functional characteristics. During processing, the forms of the symbols themselves are never changed so that symbolic paradigms work equally well by replacing symbols with variables. Those variables bind to actual symbols at run-time. In

contrast, a subsymbolic paradigm encodes knowledge using numerical patterns which are distributed over a collection of processing units. Therefore, an entity in the subsymbolic representation might be a pattern of continuous values. For example, the concept of *car* might be expressed subsymbolically as (-1.525 +0.831 +0.928 -1.192). These values may vary as necessary during processing time, although the new pattern will still behave closely to the original pattern. This feature provides tolerance of variations of a specific symbol. Thus, the ambiguity of a symbol is resolved or well-tolerated. Connections between sub-symbols can be numerically-weighted links which are subject to adaptation. Therefore, subsymbolic paradigms possess learning capabilities. These fundamental representation differences influence directly the functional characteristics and compositional styles of each paradigm.

One of the main criticisms to against the subsymbolic paradigms is that the subsymbolic models are unable to represent useful *compositional structure* [16]. In symbolic models, symbols are combined into higher-order composite representations which maintain the integrity of the constituent elements without losing the relationships between those elements. These relationships form well-defined structures between symbols. On the other hand, subsymbolic paradigms employ layered nodes which are flat and do not exhibit useful compositional structure. The compositional representations of symbols are encoded into patterns similar in form to each individual symbol. This point was addressed by the distributed connectionist model, called Recursive Auto-Associative Memory (RAAM) [41]. The RAAM model encodes recursive data structures such as trees and lists into distributed representations. These representations of data structures are very different from the symbolic concatenative structures.

Symbolic and subsymbolic paradigms have their own advantages and disadvantages respectively. However, advantages of symbolic paradigms supplement disadvantages of subsymbolic paradigms. The same situation applies to the reverse relationship. Therefore, hybrid of symbolic and subsymbolic paradigms have been gaining more interesting to researchers [51] [58]. Hybrid systems attempt to resolve the disadvantages which appeared in each of symbolic and subsymbolic paradigms while maintain their advantages. From the symbolic viewpoint, hybrid systems are trying to resolve the problems such as:

- brittleness of decision making,
- resolution of symbolic ambiguities,
- decision making with incomplete input information,
- lack of learning mechanisms,
- maintaining multiple searching paths while eliminate backtracking,
- maintaining semantic structures, and
- maintaining reasonable processing time while knowledge base size increased, if parallel processing is implemented.

1.1 Strengths and Limitations of Symbolic Models

Symbolic representations utilize forms that humans can understand easily. Therefore, thes knowledgebases can be more readily maintained and revised. The symbolic processing of Artificial Intelligence has two computing directions: one is function-based computing and the other is logic-based computing. Function-based computing, e.g. LISP, takes a set of symbols as inputs to map to another symbols

as outputs. Logic-based computing is based on relations between objects and their properties. PROLOG is an example of a programming language using predicate logic to represent knowledge using two-valued logic, either TRUE or FALSE.

Probabilistic Reasoning [40] is one scheme which does not compute truth values using two-valued logic. It uses Bayesian formalism or *conditional probabilities* to compute beliefs between symbolic variables. Symbolic variables are connected into Bayesian Networks. Each Bayesian network node represents one symbolic variable. Related symbolic variables are linked together forming a Directed Acyclic Graph (DAG). Each DAG is an Independent-Map or I-map which showing minimal links required to make clear independence between symbolic variables for a given probability distribution P. The directed acyclic links represent direct dependency between symbolic variables: a symbolic variable depends on the symbolic variables which have outward links connected to it. To compute the value of belief of a symbolic variable, it needs information of the prior probabilities of the symbolic variable and the causal symbolic variables, also the likelihood probability. For example, in the propositional statement "*if X then Y*", X is the causal variable of Y. The prior probabilities of X and Y are *P(x)* and *P(y)* which are probabilities for $X = x$ and $Y = y$ alone. The likelihood probability is $\lambda(x) = P(x \mid y)$ which is the probability of how likely it happens while $Y = y$ and $X = x$. The belief of Y is computed as:

$$BEL(y) = P(y \mid x) = \frac{P(x \mid y)P(y)}{P(x)} = \frac{\lambda(x)P(y)}{P(x)} \tag{1.1}$$

If a third symbolic variable Z is caused by Y, then the likelihood probability $\lambda(x)$ has to include the affection of outcomes of Z into consideration:

$$\lambda(x) = P(x \mid y) = \sum_{y} P(z \mid y, x)P(y \mid x) = \sum_{y} P(z \mid y)P(y \mid x) = M_{y|x} \bullet \lambda(y) \tag{1.2}$$

where $M_{y|x}$ is the conditional probabilistic matrix of the two variables X and Y and $\lambda(y) = P(y \mid z)$ is the likelihood probability of $Y = y$ if $Z = z$. From observations of Equations 1.1 and 1.2, some points are concluded as:

- belief of a symbolic variable can be computed locally.
- belief of a symbolic variable can be affected if new events which have no direct dependency are added to a DAG.
- computational complexity increases while more symbolic variables are included in a DAG.
- computational complexity increases with more outcomes of each symbolic variable, and
- all probabilities of outcomes of each symbolic variable has to be known before definitive calculation of beliefs.

Computational complexity will be increased if more symbolic variables are causally related of other variables. Nonetheless, the structure of Bayesian Network is favorable for parallel processing.

Rule-based systems use a set of interrelated hypotheses. Each rule contains a premise and a conclusion. The premise is, in turn, made of one or more conditions. The general form of a rule can be expressed by:

If *premise* then *conclusion*;

A rule suggests that the conclusion follows the premise in an action-oriented way. Once the rule matches the premise part to known facts, then, the rule is proven or the conclusion is confirmed. Rule-based systems are limited by:

- lack of partial matching the premise,
- use of backtracking to find other searching paths, and
- poor conflict-resolution capability.

Associative networks are also known as semantic networks [43]. Concepts are represented as nodes with labeled links representing the relations between concepts. The original semantic network models increase the activation of some particular network nodes when an input is processed. Activations are spread to connected nodes and potentially numerous irrelevant spreading activations may be generated. Marker-passing, in contrast, uses discrete symbolic markers to control the spreading activation. Markers are passed among network nodes simultaneously and are evaluated by a set of rules to decide which node a marker is passed to. The symbolic information held in their markers and networks enable them to perform dynamic role-bindings. thus, high-level inference for applications such as natural language understanding can be achieved. Associative networks are the most connectionist of the symbolic models and have the potential for massive parallel processing [8] [34]. However, marker-passing semantic networks still lack learning capabilities. In summary, symbolic processing systems face some difficult problems:

- symbolic ambiguities are difficult to resolve,
- learning mechanisms are difficult to construct,
- best searching path is not promised while multiple searching paths exist,
- confidence factors are hard to maintain while backtracking is required, and
- throughput drops when knowledgebase size increased.

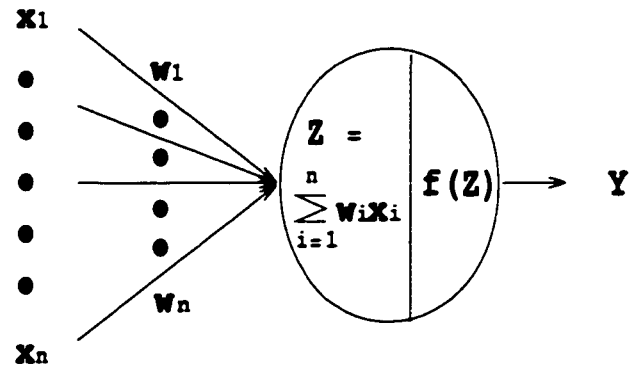


Figure 1.1: A Simple Artificial Neuron

1.2 Strengths and Limitations of Subsymbolic Models

Subsymbolic paradigms are built around the structure of Artificial Neural Networks [47] [54]. Artificial neural networks consist of input neurons, hidden neurons, and output neurons [47]. The input neurons accept information from the real world. However, the real world information has to be encoded into certain numerical formats, binary or real numbers for example, which are accepted by the input neurons. The hidden neurons receive inputs from the input neurons which have direct connections to the hidden neurons. The hidden neurons act as an intermediate stage to store intermediate state information about the network. The outputs of hidden neurons are inputs of the output neurons. An artificial neural network is tuned entirely through the network connections which store the mapping information between the input neurons and the output neurons. In order to make correct mapping information the network has to be trained by a set of input examples. The advantages of artificial neural networks are two-fold. One is at the neuron level and the other is at the network level. For an individual neuron, inputs are summed by each product of an input value and a connection weight to that neuron. The summation of products

is mapped to the output of a neuron by an activation function $f(Z)$, as illustrated in Figure 1.1. The activation function together with a threshold decide whether an artificial neuron can fire its output to other neurons. Beneficial features of this approach include:

- the output of an neuron is depends on the strengths of inputs, and
- the firing decision is made completely locally.

At the network level, the strength of all connections between neurons are regulated at the training time by a pre-selected input set. The regulation of connection weights are guided by learning algorithms. The relation information between neurons is stored in the network connections after the training phase to realize the following benefits:

- the reasoning system can work correctly even with incomplete information,
- learning capabilities are readily possible, and
- processing can be more directly implemented on massively parallel architectures.

Different subsymbolic models emphasizing these features, [14], [16], [19], [27], [41], [49], [54], [55], and [60], have been designed.

1.3 Needs for Hybrid Symbolic Processing

In hybrid symbolic systems, connectionist networks resolve problems such as belief propagation, multiple path searching, and disambiguation while using symbolic processing features to perform composition, decomposition, and variable bindings. A general model of distributed hybrid symbolic system is illustrated in Figure 1.2. This

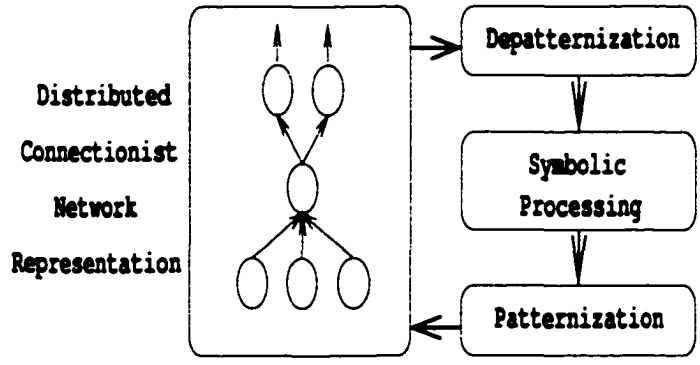


Figure 1.2: A General Model of a Hybrid Distributed Symbolic System

hybrid system [26] acts as a symbolic parser for natural language processing. The symbolic side contains a look-ahead buffer to hold a fixed number of constituents, a stack for manipulating embedded structures, an encoder to code the look-ahead buffer and the stack for the input layer of the network, and a decoder to evaluate an action from the output units. The subsymbolic part contains a feed-forward network which examines the contents of the buffer and stack and yields a preference for an appropriate action. The feed-forward network is trained with patterns which represent the encoding of the buffer positions and the top of the stack. The output of the network contains a series of units which represent actions to be implemented during processing. The input to the hybrid system is a natural language sentence. The output is a parse tree representing the intended meaning to be conveyed.

The second kind of hybrid system uses localist connections which is embedded into symbolic systems. Symbols are not encoded into numeric patterns as in the connectionist systems. A general model of hybrid localist symbolic system is illustrated in Figure 1.3. Each network node represents a symbol or a concept. A

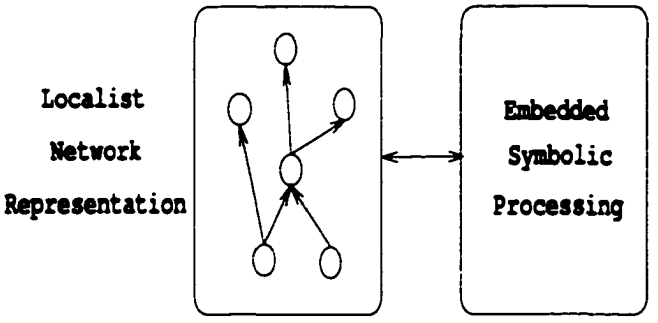


Figure 1.3: A Hybrid Localist Symbolic System

hybrid system, SC-Net [46], demonstrates use of localist and symbolic processing to integrate learning capabilities into rule-based systems. Rules are encoded as inputs to the network. Each intermediate and output cell contains a bias value which are subject to being regulated during the learning process. This model is different from others in that the learning does not occur on the weights between nodes but on the bias values. This model also shows how to build sub-networks for structured variable presentations, variable bindings, linguistic hedge evaluations and relational comparators. Two kinds of variables: structureless and structured variables are used in the SC-Net. Structureless variables have a simple form which includes a variable name and an uncertainty value associated with the variable. For example, a person's age $Age = 39$ with an uncertainty 0.9. Structured variables includes fuzzy and nominal variables. Fuzzy variables contain linguistic hedges and nominal variables do not. Each structured variable has its own sub-network with the attributes of the variable as the outputs of the sub-network. Each sub-network is evaluated by using the fuzzy logic. New sub-networks can be added into the existing network and retrained with the existing network.

The existing hybrid symbolic/subsymbolic models demonstrate various improvements over either pure symbolic or subsymbolic systems. However, there may be a need for long and intricate training periods [26]. Furthermore, the distributed networks are not necessarily scalable, as adding new examples requires retraining of the entire network. Although the SC-Net [46] shows good promise on scalability. However, the SC-Net is built on vertical relations and may lack in lateral relations between sub-networks. Symbolic composition and decomposition are another aspects which are not covered by most of the hybrid symbolic/subsymbolic models. Yet, symbols hold very important roles in high-level reasoning. They allow a system to be easily traced and understood which are important aspects to build an AI system which is accumulative [47], competitive [47], and restrictive [11]. In Chapter 3, a new hybrid symbolic/subsymbolic paradigm is proposed as a prototype model toward achieving the desired goals.

1.4 Outline of Dissertation

In the following chapters, a review of previous work is discussed in Chapter 2. Some of the earlier developed networks such as CONSYDERR [57], SC-Net [46], and $P_ARADISE$ [3], are presented in detail. In Chapter 3, the proposed architecture of an improved hybrid system called *Localized Self-Contained Adaptive Networks* (LSCAN) is introduced. The philosophy of knowledge representation is discussed initially. Next, the derived AND and OR feed-forward network units with lateral connections are discussed in detail. Learning mechanisms for both of feed-forward and lateral connections are discussed. The learning mechanisms can be used in ei-

ther supervised or unsupervised training modes. A comparison between LSCAN and CONSYDERR is also discussed. At the end of Chapter 3, the LSCAN knowledge acquisition and processing tool is introduced which implements the proposed model and techniques. The feed-forward learning mechanisms are further developed by using dual-input connections and inverters or NOT operators, to impart the functionalities of high-pass, low-pass, and band-pass filters. Illustrative examples are used to demonstrate possible applications and these feed-forward learning mechanisms are discussed in Chapter 4. In Chapter 5, a case study of hand-printed digit image classification is implemented and discussed in detail. Two classification methods, pattern-based and rule-guided classifications, are used to illustrate LSCAN functionality and performance. Chapter 6 is an additional case study of how discrete LSCAN networks can be designed to implement a network routing decision maker. A hypothetic routing network with 7 autonomous systems is presented and assessed. The corresponding LSCAN sub-networks are presented. Finally, the conclusions of this research are given in Chapter 7. The future work in this area is also expressed. The development of this dissertation is illustrated in Figure 1.4.

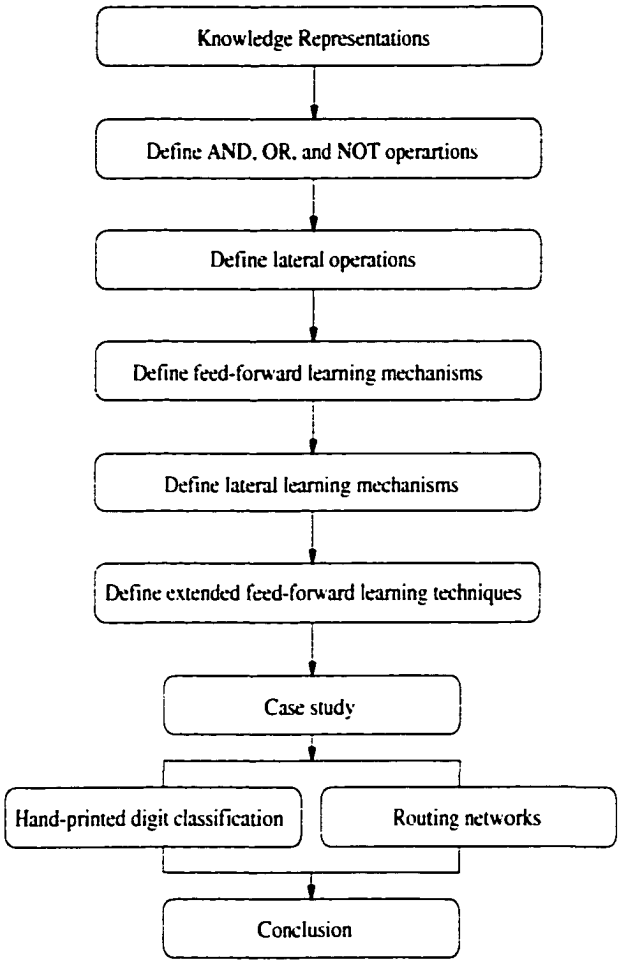


Figure 1.4: Dissertation Development Flow Chart

CHAPTER 2

PREVIOUS WORK

2.1 Overview

Due to the weaknesses of symbolic processing, various research began in the mid of 1980's to develop alternative methods in order to incorporate the advantages while eliminating the weaknesses of symbolic processing. This kind of processing blends together symbolic and subsymbolic techniques and therefore named hybrid symbolic and subsymbolic processing. In general, there are three main research paths to incorporate symbolic elements into subsymbolic processing systems. One approach is to allow symbolic elements in connectionist networks [10] [19]. The second approach associates localist networks with symbolic representations [52] [55]. The third approach mixes connectionist networks and localist networks in a novel symbolic and subsymbolic paradigm [57]. In the case of connectionist networks, symbols are encoded or patterned through the connectionist network nodes. In the case of localist networks, each node represents one symbolic element.

The network $\mu KLONE$ [10] was the first subsymbolic connectionist system designed for reasoning using high-level structured symbolic representations such as semantic networks. At any one time, an $\mu KLONE$ network represents a single hypothesis and relevant information. Each $\mu KLONE$ network consists of five modules,

three auxiliary modules which mediate some of the inter-module constraints, a variable number of modules used for input and output, and a variable number of auxiliary I/O modules. Individual symbols are patterned into a set of bit-patterns. A concept is a bit-pattern of micro-features. This kind of network falls into the first approach of injecting symbols into subsymbolic systems. One of the localist networks is the SC-Net [46] which uses an input node to present one premise of a rule. The CONSYDERR [57] networks can be considered as the third approach of injecting symbols into subsymbolic systems. The networks consist of two levels, the upper level is a localist network in which each network node represents a concept. The lower level consists of two-layer connectionist networks between subconcepts. Three promising paradigms, CONSYDERR [57], SC-Net [46], and $P_ARADISE$ [3], are reviewed in next three sections. Their main representation strategy and main principle of operation are outlined along with their advantages and limitations.

2.2 CONSYDERR

CONSYDERR [57] was designed using the concept of a two-level connectionist network. The upper level is a localist network from which each network node represents a concept or a conceptual rule. The upper level is called the *conceptual level*. This conceptual level is symbolic and structural. The links between the nodes represent the relations between the concepts. Each concept A is connected to a concept B which represents a conceptual rule $A \rightarrow B$. If the concepts A and B have features, then their features are represented as two groups of fine-grained elements on the lower level. Let F_A represent the features of concept A and F_B represent the features of concept B. Each node on the lower level represents a feature. Each node

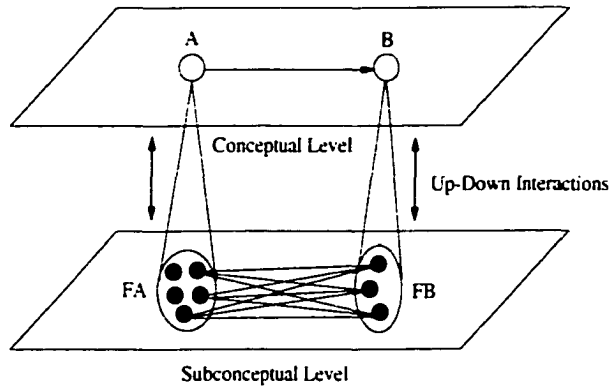


Figure 2.1: A CONSYDERR Network Structure

in F_A has links to all nodes in F_B if the conceptual rule $A \rightarrow B$ is established in the conceptual level. Otherwise, there is no link established between the nodes of F_A and F_B. A CONSYDERR network structure is shown in Figure 2.1. The lower level is called the *subconceptual level.* The main advantage of this subconceptual level is its similarity-based representation. The nodes represented on the two levels interact each other through three phases during an inference processing cycle. The three phases are the top-down phase, settling phase, and bottom-up phase. In the top-down phase, the feature nodes at the subconceptual level are activated by the conceptual nodes which have the highest activation values connected to the feature nodes. In the settling phase, nodes at each level interact with the nodes activated during the top-down phase through their weighted links. In the bottom-up phase, the nodes at the conceptual level will pick the highest activation values between their own original activation values and the activation summations of their corresponding feature nodes. The key points of these top-down and bottom-up processes are the top-down and the bottom-up inheritance.

The design of CONSYDERR combines two levels of reasoning structures. One is a high-level reasoning mechanism which resembles symbolic processing. The other is a low-level reasoning mechanism which resembles subsymbolic processing. Therefore, the CONSYDERR networks can maintain most advantages of symbolic processing at the top level while gain some advantages of subsymbolic processing at the bottom level. Some nodes at the top level can be activated due to the similarity processing at the bottom level. At the same time, the activated nodes at the top level inherit properties from some of the other nodes at the top level. The advantages of CONSYDERR networks is that they meet many of the desired features of hybrid symbolic and subsymbolic systems. However, the learning mechanisms at both levels are not well defined. This shortcoming may lead CONSYDERR networks into uncertain situations in some adaptive environments. For instance, consider a subconcept at the bottom level which needs to develop some subfeatures. In this case, there might be a need of having a third level to deal with this situation. The reasoning system may be complicated to support arbitrary interactions between levels.

2.3 SC-Net

The SC-Net [46] is a hybrid symbolic and subsymbolic system to integrate learning capabilities into expert systems, especially, rule-based systems. The system does its learning from examples that are encoded in much the same way that the other connectionist networks. The system can learn concepts where imprecision is involved. The network representation allows for variables in the form of (attribute, value) pairs to be used. Both numeric and scalar variables can be represented. They are provided to the system upon setting up for the domain. The learning algorithm

uses a network structure, which is configured based on the distinct problems presented to the system.

A SC-Net network consists of three types of cells: input, hidden, and output cells. Each cell has a bias associated with it, which lies on the real number scale. Cells are connected through links which are weighted. Each cell has an activation value within the range of [0, 1]. This corresponds to the fuzzy membership values of fuzzy sets. Some cells in the network act as *max* and *min* functions as fOR and fAND operations in fuzzy logic. Those cells are distinguished by using the sign of the bias of a cell to determine which of the two functions is to be modeled. Furthermore, a bias value of zero indicates when a cell should operate as an inverter, or fNOT operation. An example SC-Net network is shown in Figure 2.2. Each cell in a network can accommodate n inputs with associated weights CW_n. Every cell contains a bias value, which indicates what type of fuzzy function a cell models, and an absolute value which represents the rule range. Every cell C_i with a cell activation CA_i, except for input cells, computes its new cell activation according to the formula given in Equation 2.1. Let

CA_i = cell activation for cell C_i. CA_i in [0, 1].

CW_{ij} = weight for connection between cell C_i and C_j, CW_{ij} in R.

CB_i = cell bias for cell C_i, CB_i in [-1, +1].

$$CA_i' = \begin{cases} min_{j=0,\ldots,i-1,i+1,\ldots,n}(CA_j \times CW_{ij}) \mid CB_i \mid, & CB_i < 0 \\ max_{j=0,\ldots,i-1,i+1,\ldots,n}(CA_j \times CW_{ij}) \mid CB_i \mid, & CB_i > 0 \\ 1 - (CA_j \times CW_{ij}), & CB_i = 0 and CW_{ij} \neq 0 \end{cases} \quad (2.1)$$

An activation of 0 indicates no presence, 0.5 indicates unknown, and 1 indicates

true. In the initial topology contains an extra layer of two cells, denoted as the positive and the negative cell layer placed before every output cell. These two cells collect information in conjunction with the positive cell and against the presence of a conclusion from the negative cell. The final cell activation for the concluding cell is given as:

$$CA_{output} = CA_{positivecell} + CA_{negativecell} - 0.5 \tag{2.2}$$

In Figure 2.2, the unknown cell labeled as UK always propagates a fixed value of 0.5 and, therefore, acts on the positive cell and the negative cells as a threshold. The positive cell will only propagate an activation ≥ 0.5, whereas the negative cell will propagate an activation ≤ 0.5. Whenever there is a contradiction in the derivation of a conclusion, the fact will be represented in a final cell activation close to 0.5. For example, if $CA_{positivecell} = 0.9$ and $CA_{negativecell} = 0.1$, then $CA_{output} = 0.5$, which means it is unknown. If either $CA_{positivecell}$ or $CA_{negativecell}$ is equal to 0.5, then CA_{output} is equal to the others' cell activation, which indicates there is no contradiction. One example rule given as:

if and(or(s1, s2), s3) then d1 (0.8);

is translated into the SC-Net subnetwork as shown in Figure 2.3. There are two types of variables that can be used in the SC-Net systems. One type is the fuzzy variable and the other type is the scalar type variables. Fuzzy variables need special subnetworks to represent them.

The SC-Net networks showed the benefits of resolving the weakness of rule-based systems. A learning algorithm is also provided to adjust the bias values in order to change the outputs of the concluding nodes. However, using fAND operation or

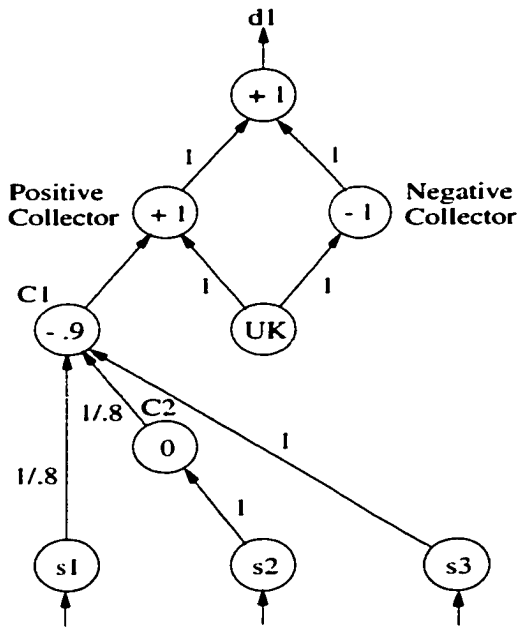


Figure 2.2: A SC-Net Network Architecture

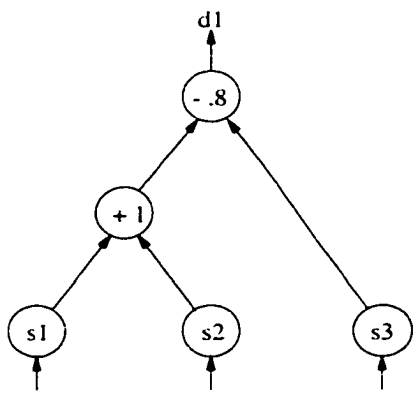


Figure 2.3: Rule in SC-Net Representation

min functions neglects the total influence of the inputs toward the concluding node. Furthermore, encoding rules into a network involves extra work to convert values into the range of [0, 1]. For variable binding, it needs a specific design for a particular fuzzy variable.

2.4 PARADISE

$P_A$ttern Recognition Architecture for Deformation Invariant Shape Encoding ($P_ARADISE$) [3] was designed as a network to detect patterns of visual images. The detected patterns can be used for image classification. $P_ARADISE$ is not a strictly hybrid symbolic and subsymbolic system. However, the network architecture and classification methods are related to the LSCAN networks. $P_ARADISE$ employs three stages of processing to achieve deformation invariant object recognition. The first stage is the feature extraction processing using either a Gaussian or Gabor filter. The second stage is pattern detection processing. The third stage is classification processing. As shown in Figure 2.4, the three layers, feature extraction layer, pattern detection layer, and classification layer work together to implement all of the three processing stages.

At the feature extraction layer, a Gaussian or Gabor filter is applied to a scanning window which moves through the entire input image. The filters are oriented in two directions. One is in the horizontal direction and the other is in the vertical direction. The two orientations result in two feature planes in the feature extraction layer as shown in Figure 2.4. In a feature plane, each location is influenced by the scanning window which covers a specific area on the input plane. After the window

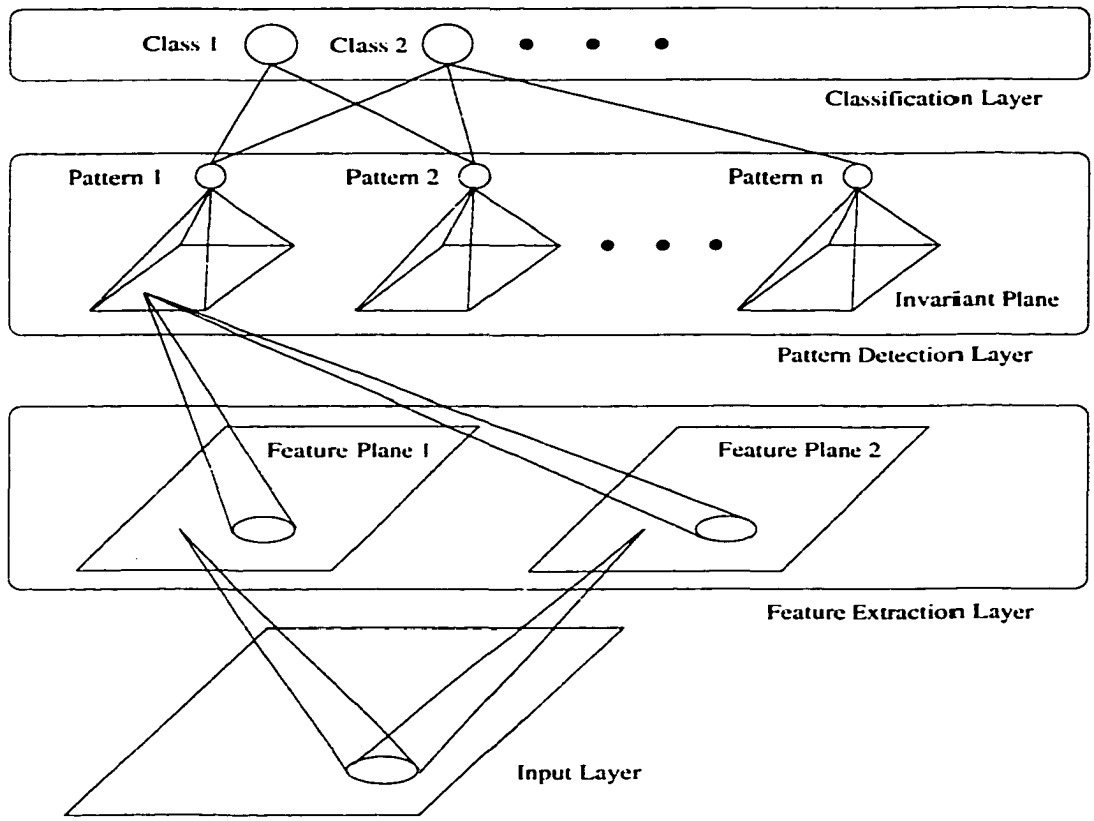


Figure 2.4: The $P_ARADISE$ Network Architecture

scanning process, the two oriented feature planes are ready for the next processing stage which is pattern detection processing.

At the pattern detection layer, a Pattern Detection Module (PDM) is shown in Figure 2.5. A PDM consists of two sublayers. The top sublayer has only one node which represents the winning node of the lower sublayer. The top sublayer node has connections to the classification nodes. The bottom sublayer or called as an invariant plane consists of $2\gamma + 1$ nodes, as explained below. Each node maps the same areas on the two feature planes. Each area on the feature plane has a size of $2\lambda + 1$. A pattern is formed if the average feature strength of the mapped areas is greater than a preset

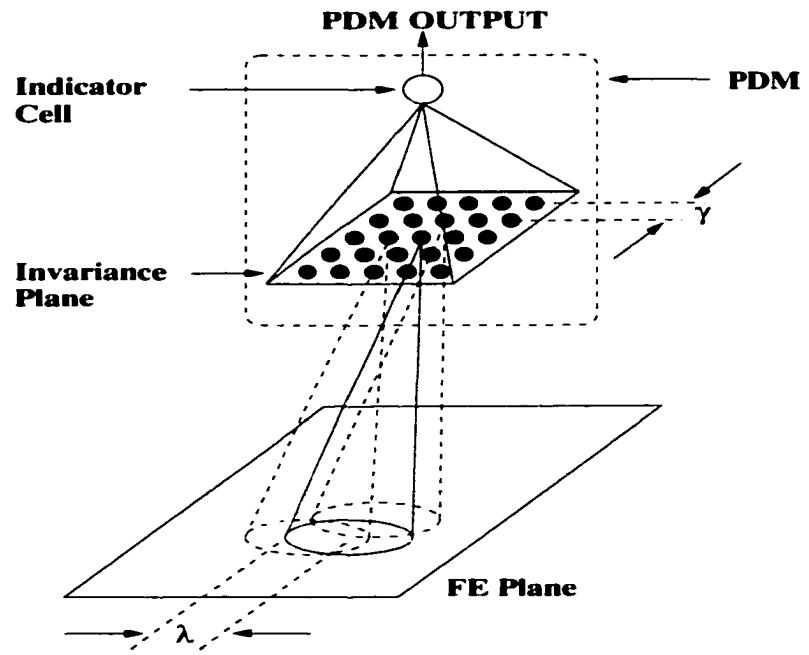


Figure 2.5: The Pattern Detection Module Structure

threshold. Then, a decision function, a modified Radial Basis Function (RBF) [3], is used to evaluate the selected pattern. If the output of the RBF is greater than the threshold then the corresponding invariant node is eligible to represent the selected pattern. At the beginning of the training, there is no PDM available in the pattern detection layer. New PDMs are created while a new class is created. The purpose of the invariant plane in a PDM is to detect potential patterns on the feature planes while their locations had been shifted to their neighborhood locations. The farthest distances of the neighborhood locations are decided by the size of γ. The size of γ also decides how much a pattern can be shared by different image objects.

At the classification layer, a classification node has input connections from some of the active PDMs. A classification node is activated if the node gains enough strength from the connected PDMs. If there are no active classification nodes to

represent the image object, a new classification node is created. While a new classification node is created, new PDMs are created if there are not enough PDMs to represent the extracted patterns from the feature planes. In the case of multiple active classification nodes representing the input image object, a hypothesis test is conducted to pick one of the active classification nodes.

$P_ARADISE$ networks had been previously applied to different image objects and showed reasonable results. The networks show three advantages over the other types of neural networks. One advantage is that the learning can be incremental which means new classification nodes can be added into the system at run-time. The second advantage is no need of pretraining of the networks. At run-time, learning happens only on the related PDMs. Therefore, learning is achieved in a relatively short time. The third advantage is that the networks can be tuned for solving specific image objects using the control parameters. A limitation of $P_ARADISE$ networks occurs when patterns are shared with many image objects. The final classification decision requires hypothetic tests which may lead to inaccuracy.

2.5 Other Research Works

There are other research works which aimed at combining symbolic and subsymbolic processing. Hendler [19] used connectionist networks to connect microfeatures to some symbolic elements in semantic networks. These symbolic elements become active through the microfeatures. On the top of the connectionist networks are semantic networks, which maintain the structures of the semantic elements. Kwasny and Faisal [26] used a connectionist network to solve symbolic parsing problem for

natural language processing. The main part of the system is a connectionist network which is trained with patterns which represent the encoding of the buffer positions and the top of the stack. The input to the hybrid system is a natural language sentence and the output is a parse tree. The patterns are encoded through a symbolic processor which handles the constituents of the natural language sentence. Shastri [52] designed the SHRUTI system to process predicate logic based systems. Each network node represents one predicate clause. The facts are connected to the related network nodes to initiate node firing. Eventually, node firing propagates through the entire network. SHRUTI networks can hence be classified as localist networks.

2.6 Summary

In this chapter, different types of hybrid symbolic and subsymbolic were introduced. Three systems, *CONSYDERR*, *SC-Net*, and $P_ARADISE$, were discussed in more detail. Their advantages and limitations were also discussed. Work from other researchers, Hendler, Kwasny and Faisal, and Shastri, were introduced briefly. There are more research works in this related areas. Those models can be found in various aspects of the other researchers' works already mentioned.

After reviewing the previous hybrid symbolic and subsymbolic models, it appears that more research can be done in order to increase the benefit of a hybrid symbolic and subsymbolic system. This is the purpose of this research to investigate how a new paradigm can benefit this research area. The new paradigm is given in next chapter.

CHAPTER 3

LOCALIZED SELF-CONTAINED ADAPTIVE NETWORKS

This chapter introduces a new paradigm which handles symbolic values using a computational model based on artificial neurons. This processing paradigm is called the *Localized Self-Contained and Adaptive Networks* (LSCAN) representation and reasoning scheme. The knowledge entities are represented by interconnected network nodes, one node for each knowledge entity or concept. Each node has capabilities to process decisions at the symbolic level. The LSCAN system propagates numerical values among the network nodes while maintaining high-level symbolic structures. Each node in the LSCAN network is sufficient to make decisions from its own local inputs without help from global information. In addition to flexibilities of decision making possessed by localist networks, LSCAN systems provide learning capabilities. Furthermore, the LSCAN systems retain the power of symbolic processing such as: variable binding, symbol composition and decomposition, relational operators. At the node level, each node associates a function to derive the outputs from the input data. The following sections provide information about how knowledge is represented and how conclusions are made from various input information.

3.1 Knowledge Representation

A viable parallel knowledge representation formalism provides the following capabilities:

- information can be easily added or removed from the knowledge base,
- representational mechanisms can perform inferences readily from the available knowledge,
- information can be incorporated to guide the inferences mechanism in the most promising direction, and
- inference mechanisms can be directly implemented on parallel computers.

To achieve these goals, the LSCAN knowledge representation uses knowledge entities and entity relations. Each entity is named as an unique symbolic identifier. Therefore, the LSCAN knowledge representation is easily understandable and traceable at the level of human cognition. Knowledge elements are viewed as knowledge entities; each entity is either a conclusion or a composition of some other knowledge elements, which represents the AND relation, or each entity is either a conclusion or a composition of an alternative of several group of knowledge elements, which represents the OR relation. In this definition, a knowledge entity can be decomposed into fine elements. Many fine elements together can be composed into a high-level knowledge entity. The AND and OR relations have been used in many AI approaches such as Rule-Based knowledge systems, Predicate Logics, and Prolog Languages. Therefore, these AI practices can be easily mapped into the LSCAN networks. Furthermore, the AND and OR relations are universal applicable to represent relations between knowledge

elements, if the relations are well defined. Let n represents the number of entities in a specific knowledge domain. An entity vector $E(e_1, e_2, e_3, \ldots, e_n)$ includes all required entities in a knowledge domain. In a LSCAN network, subnetworks can be constructed across different knowledge domains, for example, using commonsense in a specific knowledge domain. The knowledge in a specific knowledge domain can be represented in the following forms:

1. $e_i \Longrightarrow e_j$.
2. e_1 **AND** e_2 **AND** e_3 **AND** $\ldots \Longrightarrow e_j$
3. e_1 **OR** e_2 **OR** e_3 **OR** $\ldots \Longrightarrow e_j$
4. (e_1 **AND** e_2 **AND** $\ldots$) **OR** (e_2 **AND** e_9 **AND** $\ldots$) **OR** $\ldots \Longrightarrow e_j$
5. (e_1 **OR** e_2 **OR** $\ldots$) **AND** (e_7 **OR** e_{10} **OR** $\ldots$) **AND** $\ldots \Longrightarrow e_j$
6. $e_i \Longrightarrow e_j, e_k, e_l, \ldots$
7. e_1 **AND** e_2 **AND** e_3 **AND** $\ldots \Longrightarrow e_j, e_k, e_l, \ldots$
8. e_1 **OR** e_2 **OR** e_3 **OR** $\ldots \Longrightarrow e_j, e_k, e_l, \ldots$

Where $1 \leq i, j \leq n$, and $\Longrightarrow$ means implication. Each of the above forms is called an *implication rule*. The word implication is used to indicate that the assertion of an entity is depending on the strengths of the input entities. However, to simplify the discussion, the word, *rule*, is used to represent a LSCAN knowledge representation form. Rule #1 uses a single entity. Rule #2 uses more than two entities combined by the operator AND. Rule #3 uses more than two entities combined by the operator OR. The rules #4 and #5 use mixed combinations of the rules #2 and #3. Thus,

representations of the rules #4 and #5 can be resolved by the representations of the rules #2 and #3. Rule #6 represents the one-to-many mapping. Rule #7 represents the many-to-many mapping using AND operator. Rule #8 represents the many-to-many mapping using OR operator. Therefore, the LSCAN knowledge representations can be represented in the forms of the rules #1, #2, #3, #6, #7, and #8. The AND operator represents the composite relation between an entity and its attributes or input entities. The OR operator represents optional relation between an entity and its attributes.

Each entity carries two kinds of information. One is the physical values of an entity. The other information is the belief factor about an entity. There are three types of physical values: a binary number, a real number, and a character string. The belief factor is a real number between 0 and 1. The belief factor with a value of 1, an entity is fully accepted. Otherwise, with a value of 0, an entity is fully rejected. The LSCAN inference mechanism can always process and propagate belief factors in the range of [0, 1]. A NOT operator can be applied for each entity. If the NOT operator is applied to an entity, the belief factor of that entity is subtracted from 1. For example, the belief of BOB_IS_TALL is 0.82, then, the belief of NOT(BOB_IS_TALL) is 0.18.

An entity can represent a relation between other entities. This kind entity is called *relational entity*. There are four relations between entities:

- one-to-one relation,
- one-to-many relation,
- many-to-one relation, and

- many-to-many relation.

For example, the one-to-one relation is a person's Social Security Identification Number which is unique for each person. On the other hand, one physics teacher has many students, which is an example of one-to-many relation. Conversely, the students having a particular physics teacher is an example of many-to-one relation. For the many-to-many relation, a proper example is words spoken in a natural language sentence and the intended concepts which the sentence delivers. These relations are represented by using the forms of #1, #6, #2, and #7 repectively.

The other kind of entity is called *assertive entity*. An assertive entity asserts something with a belief factor. For example, the entity BOB_IS_TALL is an assertive entity. Depending on the physical values asserted, there are four classes named as: *constant entity*, *variable entity*, *functional entity*, and *decision entity*. A constant entity has only one possible physical value. A variable entity has many possible physical values. A functional entity manipulates the physical values of its input entities. A decision entity asserts TRUE or FALSE according to its input entities. The processing of belief factors using the LSCAN knowledge entities is discussed in next section.

3.2 Reasoning Mechanisms

The reasoning mechanism of the LSCAN system is discussed in this section. As discussed in the previous section, a LSCAN knowledge entity carries two types of information. One type of entity information is the belief factor of an entity. The value of a belief factor is a real number in the range of [0, 1]. The other type of entity information is the physical values of an entity. The LSCAN reasoning mecha-

nism manipulates belief factors explicitly between knowledge entities. The assertion of the LSCAN reasoning mechanism is based on the values of belief factors among a set of entities. The largest belief factor is selected as the assertion of an information query. Some knowledge representations use two-valued logic. These knowledge representations are Predicate Logic, Frame-based Systems, and Database Systems. The belief factors for these systems are embedded in their knowledge representations. The reasoning mechanisms for these systems directly manipulate the physical values of the knowledge components. A belief factor has a value of one if a physical value is matched. Boolean Logic systems implement two-valued belief factors directly without physical values, or physical values equal to belief factors from the viewpoint of the LSCAN reasoning mechanisms. Production systems [39] process reasoning in two phases: matching premises of rules and combining confidence factors.

The LSCAN reasoning mechanisms are analogous to the Production Rule systems in the matter of forward processing. However, there are differences between the LSCAN reasoning mechanisms and the production systems on the ways of computing the belief factors and resolution of conflicting rules. Furthermore, in rule based systems, computational tractability problems are encountered when the size of knowledge base increased and the relations between the knowledge entities become more complicated. The reasoning mechanisms face harder decisions to select a correct direction among all possible searching paths. For processing efficiency, a reasoning mechanism may have to forego multiple search paths. On the other hand, LSCAN reasoning mechanisms attempt to meet the following goals:

- maintain multiple search paths,

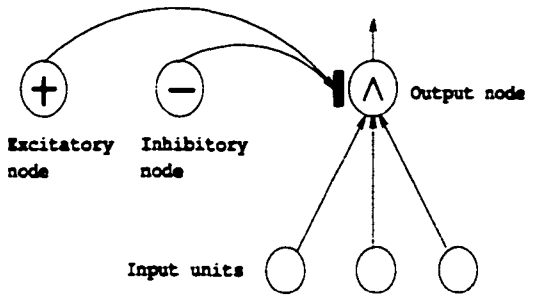


Figure 3.1: A Complete AND E-node Structure

- provide belief-factor propagation and two-valued logic processing,
- resolve rule conflicts and rule synchronization constraints,
- provide training and learning capabilities,
- tolerate incomplete information, and
- utilize massive parallelism during the reasoning process.

3.3 Low-Level System Structure: Evaluator Nodes

As mentioned in Section 3.1, the AND relation is required for all composite entities. The AND operator obtains the summation of the attribute entity strengths. Let ϵ represent the summation for the entity node e_i. Thus, the mathematics expression for the AND operator is:

$$\epsilon_i = \sum_{j=1..n} \xi_j w_{ji} \tag{3.1}$$

where $\xi_j, j = 1..n$, are n belief factors of the active entities connected to the entity e_i and w_{ji} are the connection strengths between the entities e_j and e_i. A complete AND evaluating node is illustrated in Figure 3.1 which contains feed-forward connected

input nodes and lateral interacting nodes.

Each entity is embedded in an evaluator node, called an E-node. Let ξ_i be the output strength of the entity e_i, and ξ_i is expressed as:

$$\xi_i = \left(\frac{1}{1 + e^{-(\varepsilon_i - \delta_{sh})}}\right)\left(\frac{\varepsilon_i}{\delta_{sf}}\right) \tag{3.2}$$

where ε_i is a normalized equation from Equation 3.1 based on the ratio of number of input connections d_{in} and the constant δ_{sf}; in mathematical expression:

$$\varepsilon_i = \epsilon_i \left(\frac{\delta_{sf}}{d_{in}}\right) \tag{3.3}$$

The constant δ_{sf} is a pre-selected constant number in order to scale the first factor in Equation 3.2 into the range of [0, 1]. An appropriate default value is $\delta_{sf} = 28.4$ which is just sufficient to saturate the function:

$$\frac{1}{1 + e^{-(\varepsilon - \delta_{sh})}} \tag{3.4}$$

The function 3.4 is a modified function of a regular Sigmoid function called *Shift-Right Sigmoid* function which shifts the center of an original Simoid curve to a positive offset, δ_{sh}, in the horizontal direction. Where δ_{sh} is equal to $\frac{\delta_{sf}}{2}$. The curve of the Equation 3.4 is illustrated in the Figure 3.2. Equation 3.2 is called a *Scaled-Shift-Right Sigmoid* function which maps inputs to an output value into the range of [0, 1]. The plot for Equation 3.2 is illustrated in the Figure 3.3.

The OR relation indicates sufficient conditions which are not always necessary. If an entity has OR relations with sets of ANDed entities then the entity picks up the strongest set as its belief factor. The OR operator is analogous to a

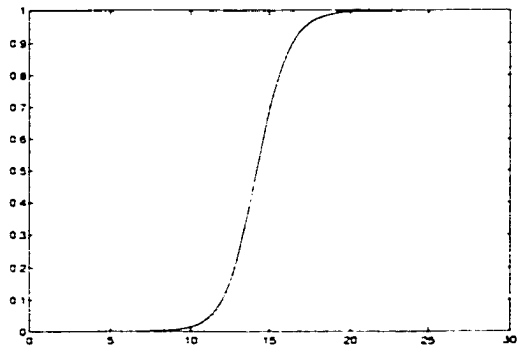


Figure 3.2: Shift-Right Sigmoid Function

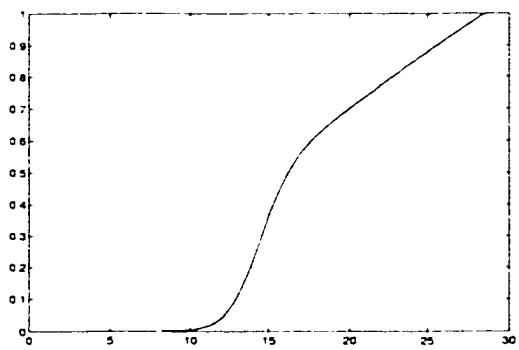


Figure 3.3: Scaled-Shift-Right Sigmoid Function

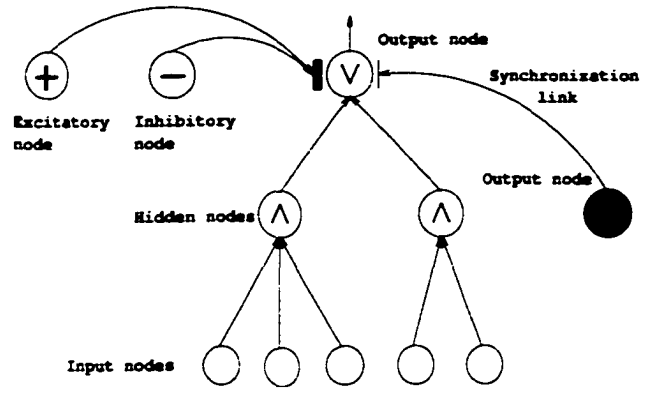


Figure 3.4: A Complete OR E-node Structure

local *winner-get-all* [48] operation. The locality of competition is among a small set of ANDed neural units. As illustrated in Figure 3.4, each hidden unit gets the summation of its input strengths by applying Equation 3.1. Next, the output strength is evaluated by using Equation 3.2. The output unit picks up the largest strength ξ_k from the hidden unit e_k. The mathematics expression for the OR operator is:

$$\xi_i = MAX_{k=1..m}\,(\xi_k W_{ik}) \tag{3.5}$$

where m is the number of hidden units an W_{ik} is the connection strength between the entity e_i and the hidden unit e_k. The default value for W_{ik} is 1. Different values can be assigned to W_{ik} in order to distinguish the preferences among the hidden units.

The NOT operator inverts the belief factor of an entity. The mathematics expression for the NOT operator is:

$$\overline{\xi}_j = 1 - \xi_j \tag{3.6}$$

where ξ_j is the belief strength of the entity e_j and $\overline{\xi}_j$ is the inverted belief strength of the entity e_j. LSCAN systems use these AND, OR and NOT operators together

with activation function of entities to process the belief factors. Through the LSCAN network, belief factors are propagated.

Comparing the *Scaled-Shift-Right Sigmoid* function to the *Shift-Right Sigmoid* function, the former has some advantages over the later as an activation function. The Scaled-Shift-Right Sigmoid function consists two parts in the plot of Figure 3.3. The lower half of the plotting curve is a near-exponential curve and the upper half is a near-linear curve. Unlike the Shift-Right Sigmoid function which separates activation strengths into two extremes, near 0 or near 1, the Scaled-Shift-Right Sigmoid activation function provides following features:

- Degrades activation strengths for weak inputs.
- Distinguishes activation strengths for strongly active inputs.

The property of the Scaled-Shift-Right Sigmoid activation function has the similar effect as the experimental results, frequency versus input current of a standard neuron, of Gerstner [17]. Firing an E-node is decided by an individual threshold θ_i. If the activation strength ξ_i is greater than the threshold θ_i then the E-node, e_i, is fired with the strength ξ_i. Each E-node is classified into two types of belief logic. One is the two-valued logic and the other is the real-valued logic which has values fall into the range of [0, 1]. For the two-valued logic, the threshold θ_i of an E-node is set at a value near to 1 so that output is generated if the evaluated expression indicates a TRUE condition.

There are also three lateral relations between the LSCAN E-nodes: excitatory, inhibitory, and synchronization interactions. The excitatory interactions permit

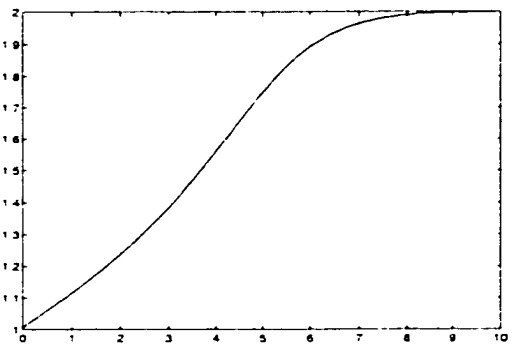


Figure 3.5: Enhancement Lateral Interaction

an E-node to excite other E-nodes. On the other hand, the inhibitory interactions discourage other E-nodes. As illustrated in the Figure 3.1 and Figure 3.4, both excitatory and inhibitory interactions can coexist resulting in the summation of the excitatory and inhibitory strengths. If the E-node e_i has p excitatory interactions and q inhibitory interactions, then the total influence of the other E-nodes on the E-node e_i is:

$$\rho_i = \sum_{j=1...p} \epsilon_j \omega_{ji} - \sum_{k=1...q} \epsilon_k \omega_{ki} \tag{3.7}$$

where ω_{ji} and ω_{ki} are lateral connection strengths between the E-node e_i and other E-nodes e_j and e_k.

If ρ_i is positive, then ρ_i is mapped into the range [1, 2] with the mapping factor η_i. Then, η_i is used to magnify the output strength of the E-node e_i by multiplying the mapping factor η_i. The mapping factor η_i is illustrated in the Figure 3.5 and is given as:

$$\eta_i = \left(\frac{-1}{1 + e^{-(\delta_{th} - \rho_i)}}\right)\left(1 - \frac{\rho_i}{\delta_{lf}}\right) + 2 \tag{3.8}$$

where δ_{lf} is a constant value which makes the curve in the Figure 3.5 saturation. A

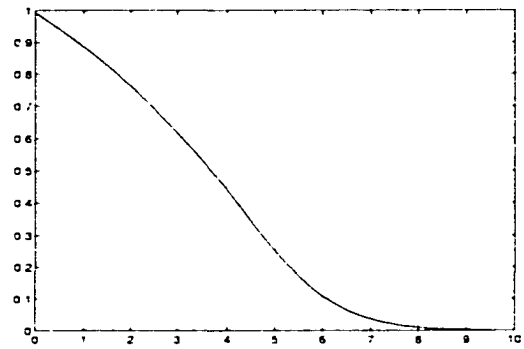


Figure 3.6: Degradation Lateral Interaction

good practical value for δ_{lf} is 10 so that $\delta_{lh} = \frac{\delta_{lf}}{2}$.

If ρ_i is negative, then ρ_i is mapped into the domain [0, 1] which is represented by the mapping factor η_i. The mapping factor η_i is multiplied to the output strength of the E-node e_i. Therefore, the E-node e_i is discouraged. The plot of η_i for the negative ρ_i is illustrated in the Figure 3.6 and is given as:

$$\eta_i = \left(\frac{1}{1 + e^{-(\delta_{lh} - \rho_i)}}\right)\left(1 - \frac{\rho_i}{\delta_{lf}}\right) \tag{3.9}$$

Equation 3.8 and Equation 3.9 are mirror functions of one another. The result of η_i is applied to Equations 3.2 and Equation 3.5. The output ξ_i of an E-node e_i becomes:

$$\xi_i = \xi_i \eta_i \tag{3.10}$$

The effects of lateral connections are two-fold. One benefit supports conflict-resolution. If two or more E-nodes are active at the same time and they conflict each other, the preferred E-nodes inhibit the others to be active. The other benefit is support for non-monotonic reasoning. If new information is added which contradicts previously established knowledge then the old information can be inhibited. This action results in no conclusion from affected E-nodes. Now consider the more com-

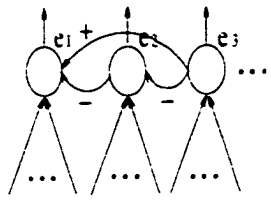


Figure 3.7: An Example of Chained Inhibitory Links

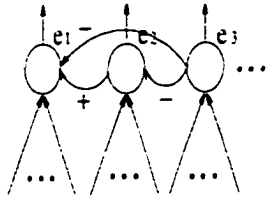


Figure 3.8: An Example of Chained Excitatory-Inhibitory Links

plicated cases which have e_1 to be degraded by e_2 which then to be degraded by e_3. The problem raised over here is that how e_1 knows it should be activated because e_2 is degraded by e_3. This scenario and solution are illustrated in the Figure 3.7. The node e_1 is enhanced by e_3 in order to cancel the effect of degradation from e_2. In this manner, e_1 is able to make decision by itself alone without looking at whether e_2 is degraded by e_3. In the case of that e_1 is enhanced by e_2 which then is enhanced by e_3. e_1 is not required to have an enhanced link from e_3. However, in the case of that e_1 is enhanced by e_2 which then is degraded by e_3, then e_1 is required to have an degraded link from e_3 in order to cancel the effect of enhancement from e_2 as illustrated in the Figure 3.8.

The lateral synchronization connection is always two-valued: a connection is either active or inactive. If an E-node e_i is active while a lateral synchronization connection is asserted from the other active E-node e_j, then the E-node e_i is deactivated regardless the activation strength of the E-node e_i. The lateral synchronization

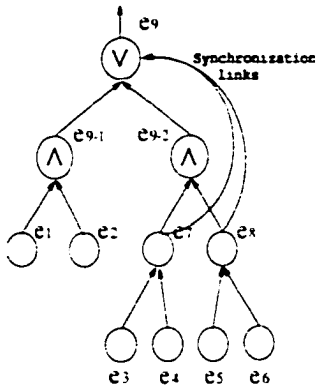


Figure 3.9: An Example of Using Synchronization Links

connection is required while an ORed subnetwork has different depths in its branches. Consider the scenario giving the following structure:

e_1 **AND** $e_2 \Longrightarrow e_9$

e_3 **AND** $e_4 \Longrightarrow e_7$

e_5 **AND** $e_6 \Longrightarrow e_8$

e_7 **AND** $e_8 \Longrightarrow e_9$

This structure is illustrated in the Figure 3.9. If all of the entities: e_1, e_2, e_3, e_4, e_5, and e_6 are activated, then e_9 can be activated through either the left or the right branch. The activation of e_1 and e_2 will activate e_9 directly. However, the activation of e_3, e_4, e_5, and e_6 will activate e_7 and e_8 which then activate e_9. Therefore, e_9 is activated at different time which may not give e_9 a chance to pick up the highest strength between the two branches. The solution to this problem is to use the synchronization connections between e_7, e_8, and e_9 as shown in the Figure 3.9. Once e_7 or e_8 is activated, e_9 is disabled to wait for one more processing cycle. If neither e_7 nor e_8 is activated, e_9 has no problem to be activated immediately through the left branch.

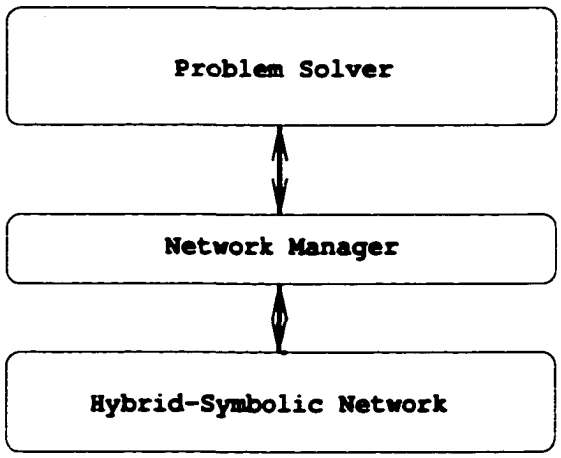


Figure 3.10: A System Diagram

3.4 A High-Level System Structure

This section discusses the network management process used during problem solving. A LSCAN system consists of an artificial localist network, a problem solver, and a network manager as shown in Figure 3.10. The LSCAN network provides multiple searching paths and local decisions based on the relations between an E-node and other E-nodes. Initially, some leaf E-nodes are activated by some facts. A leaf E-node has no input connections. Next, the activated leaf E-nodes activate some other E-nodes which are connected to them. This process continues until no E-nodes are activated. An entire firing cycle is defined as a processing period from the time of the first E-node is fired to the time of no E-node is fired. An entire firing cycle is composed of many firing stages. A firing stage is defined as the time of some E-nodes activated to the time of some other E-nodes which are activated. Once activated, each E-node is disabled from further activating within the same cycle. To generate the effects of lateral interactions between E-nodes, a firing stage is accomplished in

two phases. The first phase computes the output strength, ξ_i, from the inputs, and η_i using Equation 3.2 and Equation 3.9. The second phase recomputes the output strength, ξ_i using Equation 3.10, if any lateral interacting E-node is activated.

The network manager provides the services of controlling the firing cycle, changing thresholds of E-nodes, reporting results from any specific E-node, and tracing firing paths. The problem solver interprets a specific problem and converts the problem into forms which are implemented by the LSCAN networks. The final results inferred by the LSCAN network are also interpreted by the problem solver.

3.5 Learning Mechanisms

LSCAN systems support learning from the input units of an E-node and lateral connections. The inputs contribute to learning by allowing partial set of inputs to result in the activating of an E-node, the learning from incomplete information. Supervised learning and unsupervised learning schemes are both provided with the LSCAN systems. Lateral learning occurs when two or more conflicting E-nodes resolve the conflict by allowing one or more than one E-node to be activated. Learning schemes adjust the weights of the links between E-nodes, as discussed below.

3.5.1 Supervised Learning

Supervised learning is a guided scheme where both the facts and the expected results are provided at each firing cycle. Each expected firing E-node adjusts its own input connection weights based on the difference between the actual and the expected output strengths and the number of active inputs. If some of the inputs of an E-node

are all active, then this E-node is learning how to deal with incomplete information. Let an ANDed E-node, e_i, have n input units with the input connection weights, ω_{ji}. Also, let μ_i be the expected output strength and the actual output strength be ξ_i. At the time of firing the E-node e_i, the number of active input units is m and $m \leq n$. In order to adjust each weight of ω_{ji}, ν_i has to be computed from μ_i using Equation 3.2; ξ_i is replaced by μ_i and ε_i is replaced by ν_i. However, Equation 3.2 is not reversible, so the Newton-Raphson [42] method is used to find an approximate ν_i from μ_i. Let the difference between ν_i and ε_i be $\Delta\varepsilon_i = \nu_i - \varepsilon_i$. The adjustment of weights $\omega_{ji(new-active)}$ is proportional to the radios of $\frac{\nu_i}{\delta_{sf}}$ and $\frac{\Delta\varepsilon_i}{m\xi_{ave}}\frac{n}{\delta_{sf}}$. The final weight adjustment for active input units is:

$$\omega_{ji(new-active)} = \omega_{ji(old-active)} + \frac{n\nu_i}{\delta_{sf}^2 m}\frac{\Delta\varepsilon_i\lambda}{\xi_{ave}} \tag{3.11}$$

where ξ_{ave} is the average input strength and λ is a learning rate which limits the learning step size. The value of λ is less than 1 and greater than 0. In the case of $\lambda = 1$ only one learning step is required. The smaller λ creates a longer training period. On the other hand, weights of inactive input units are adjusted based on the total amount weight change for active input units. Then, for inactive input units, their weights are adjusted as:

$$\omega_{ji(new-inactive)} = \omega_{ji(old-inactive)} - \left(\sum_{j=1\ldots m}\left(\omega_{ji(new-active)} - \omega_{ji(old-active)}\right)\right)\frac{1}{n-m} \tag{3.12}$$

In the case when $n = m$ which implies all input units of an E-node are active and equation 3.12 is not necessary. The weight adjustment, can be based on each input strength: stronger input units gain more weight. On the other hand, weaker input units lose weights. Let ξ_{ave} be the average input strength for the E-node e_i. Then,

the weight adjustment is given as:

$$\omega_{ji(new)} = \omega_{ji(old)} + \left(\frac{\xi_j}{\xi_{ave}} - 1\right) \frac{\nu_i}{\delta_{sf}^2} \frac{\Delta\varepsilon_i \lambda}{\xi_{ave}} \tag{3.13}$$

The training process proceeds until all supervised E-nodes reach the expected outputs. The process trains input connection weights only for those E-nodes which are supervised while all other input connection weights are intact. For an ORed E-node, learning occurs on the winning hidden unit.

3.5.2 Unsupervised Learning

During unsupervised learning, expected values are not assigned. Equations 3.11, 3.12, and 3.13, are used. However, the expected output strength μ_i is replaced by the threshold θ_i. Furtheremore, an E-node is trained only when the actual output strength ξ_i is greater than and equal to $\alpha\theta_i$, where α is a factor to control whether an E-node has to be trained. The value of α is between 0 and 1. Otherwise, an E-node is not trained.

3.5.3 Learning with Bias Nodes

With the AND connections, any LSCAN node can add one or more bias nodes to it as shown in Figure 3.11. The purposes of bias nodes are two fold. One purpose is to stabilize the training process. In some cases, training process takes more training steps to train a LSCAN node while the strength of a LSCAN node is getting closer to the training threshold. Addition of bias nodes can help to train a node to reach the training threshold more accurately and use less training iterations.

The other purpose is to make bias nodes as active or inactive nodes by giving

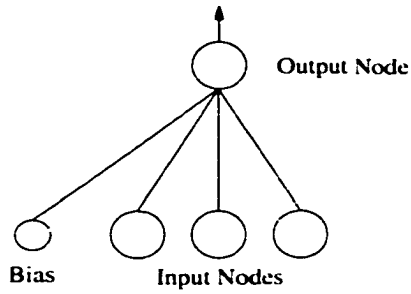


Figure 3.11: An AND E-node Structure with Bias Node

bias nodes with constant values of 0 or a value which is greater than θ, which is the threshold of a LSCAN AND node. Bias nodes are active while their values are greater than θ. In this case, the bias nodes act as connection weight sink. Some of the weaker connections to the LSCAN AND node lose more connection weights after training. Therefore, the weaker input nodes have lesser influence on the output node. On the other hand, bias nodes are inactive while their values are always 0. In this case, the bias nodes act as connection weight source. The connections of the bias nodes provide weights to the other in-bound connections of the LSCAN AND node. The bias nodes promote the other input nodes connected to the LSCAN AND node to make them having more influence on it. Also, bias nodes change the training Equation 3.13 to the Equations 3.11 and 3.12.

3.5.4 Competitive Learning

Competitive learning occurs at the lateral level. At each firing stage, multiple E-nodes are fired simultaneously. Conflicting E-nodes establish inhibitory connections from the preferred E-nodes to the conflict E-nodes. There are supervised and unsupervised competitive learning. For the supervised competitive learning, the LSCAN systems select one or more preferred E-nodes which can be continued to trigger other

E-nodes while the other firing E-nodes are inhibited. The inhibited E-nodes are degraded under their own thresholds in order to be not fired. Therefore, the inhibitory connections have weights to be regulated according to the total inhibitory strength. Let ω_{ik} be the inhibitory connection weights from e_i to e_k and $i = 1 \ldots \rho$. Then, the weights ω_{ik} are adjusted by:

$$\omega_{ik(new)} = \omega_{ik(old)} + \frac{\xi_i}{\sum_{j=1\ldots\rho} \xi_j} \frac{\varepsilon_{\mu k}}{\delta_{lf} \xi_{i(ave)}} \tag{3.14}$$

where $\varepsilon_{\mu k}$ is derived from θ_k / ξ_k by the Newton-Raphson [42] method and $\xi_{i(ave)}$ is the average inhibitory strength from all E-nodes $e_{i=1\ldots\rho}$ to e_k. Under supervised competitive learning, at one firing stage the LSCAN systems can inhibit some E-nodes from different non-inhibitory E-nodes.

Under unsupervised competitive learning, the maximum number of E-nodes that can be fired at a firing stage is specified as κ. The first κ E-nodes with greater output strengths are used to inhibit the others. Equation 3.14 is used to adjust the weights ω_{ik}. Under the unsupervised competitive learning, all E-nodes are inhibited by the same group of the preferred E-nodes.

3.5.5 Enhancement Learning

The LSCAN systems also support, at a firing stage, exaggeration of some E-nodes from others if they are fired at the same time. The LSCAN systems establish connections from some E-nodes $N_{i=1\ldots\rho}$ to the E-node N_k which is exaggerated. Alternately, some E-nodes can be exaggerated with respect to each other. This kind of learning is called enhancement learning and is supervised only. This kind of learning occurs when some conditions are asserted. In other words, activations of some

. Even an inactive E-node

CONSYDERR

orks are similarity and in-

LSCAN networks through

SCAN networks have sim-

eractions between the sub-

nstructed by using LSCAN

CAN approaches.

el is defined as the number

If A and B are the two

are represented as F_A and

f features of F_A is m_A and

pped features is m_{AB} which

ts A and B is measured as

similarity representation is

trength if the concept A is

ausal fact of B. In this case

$\times ACT_A$. If the concept A

activated with the strength

s of F_B. This measurement

E-nodes are exaggerated if some other clues are activated. Even an inactive E-node can be activated by one or more exaggerating E-nodes.

3.6 Comparison Between LSCAN and CONSYDERR

The main functionalities of CONSYDERR networks are similarity and inheritance. These two functionalities can be achieved by LSCAN networks through appropriately structured networks. As this view point, LSCAN networks have similar representation power as CONSYDERR has. The interactions between the sub-conceptual nodes in the sub-conceptual layer also can be constructed by using LSCAN networks. The following two sub-sections describe the LSCAN approaches.

3.6.1 Similarity

The similarity appeared on the subconceptual level is defined as the number of overlapping features between any two feature groups. If A and B are the two concepts represented in the conceptual level, their features are represented as F_A and F_B respectively in the subconceptual level. The number of features of F_A is m_A and the number of features of F_B is m_B. The number of overlapped features is m_{AB} which can be zero. Then the similarity between the two concepts A and B is measured as $s_{AB} = (A \sim B) \approx \frac{m_{AB}}{m_B}$. One of the advantages of the similarity representation is to allow the concept B to be activated to some level of strength if the concept A is activated with the strength ACT_A and A is not a direct causal fact of B. In this case the concept B is activated with the strength $ACT_B = s_{AB} \times ACT_A$. If the concept A is activated with a full strength of 1, then the concept B is activated with the strength $ACT_B = s_{AB}$ which is a fraction of the number of features of F_B. This measurement

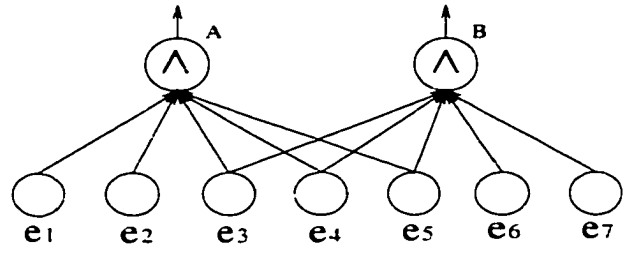


Figure 3.12: An Example of Weak-Bound Similarity

is very close to the results using the LSCAN's activation function (Equation 3.2). The outputs of this equation are plotted in the Figure 3.3 which shows that the activation strengths are near s_{AB} if the activation strengths are greater than 0.6. The activation strengths are dropped sharply if the similarity s_{AB} is less than 0.6. The Equation 3.2 can be used as a measurement of similarity of s_{AB} in a non-linear manner. This non-linear manner shows a good property to prevent weak similar concepts from being activated and propagated.

There are two cases for the concept B to be activated through the overlapped features with the concept A in LSCAN's structures. The first case is called *weak-bound-similarity* which is shown in the Figure 3.12. The activation strength of the concept B does not depend on the activation strength of A. This is due to the fact that features, or knowledge entities, of the concept A are not activated by A. The second case is called *strong-bound-similarity* which is shown in the Figure 3.13. The features of the concept A are activated by A. Therefore, the activation strength of the concept B depends on the activation strength of A. The similarity of B to A is measured only when the E-nodes e_3, e_4, and e_5 are activated while e_6 and e_7 are not active.

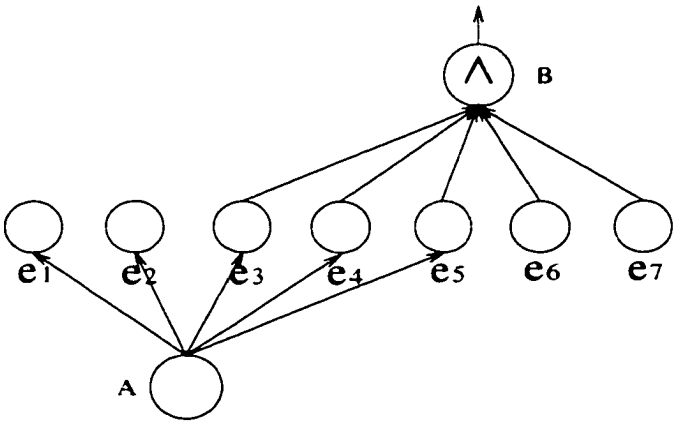


Figure 3.13: An Example of Strong-Bound Similarity

3.6.2 Inheritance

Consider a set of concepts A and B which A is the superclass of B. In the feature space, the features F_A are a subset of F_B. Similarly, C is the superclass of D and F_C are a subset of F_D. In the CONSYDERR, the following cases are handled:

- Suppose A has a property value C and B has no specified property value. If B is activated then C is activated to a certain extent from the top-down inheritance of A.

- Suppose B has a property value D and A has no specified property value. If A is activated then D is activated too from the bottom-up inheritance of B.

In LSCAN, the first case is handled as shown in Figure 3.14. In Figure 3.14, e_A is the superclass of e_B. E_A covers all features which include the subclass of e_B. E_C is activated only when $e_{1..7}$ are all activated. While e_B is activated, e_C is activated to a certain strength by e_4, e_5, and e_6. The second case is handled as shown in Figure 3.15. While e_A is activated, e_D is activated by e_4, e_5, and e_6.

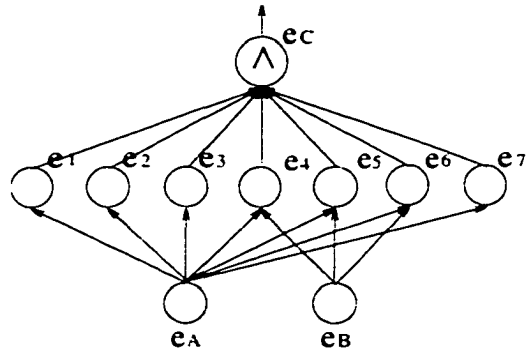


Figure 3.14: An Example of Inheritance for The Case 1

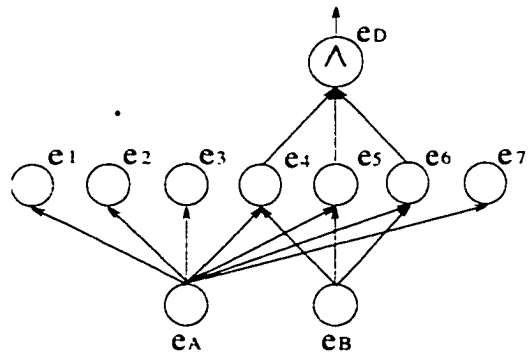


Figure 3.15: An Example of Inheritance for The Case 2

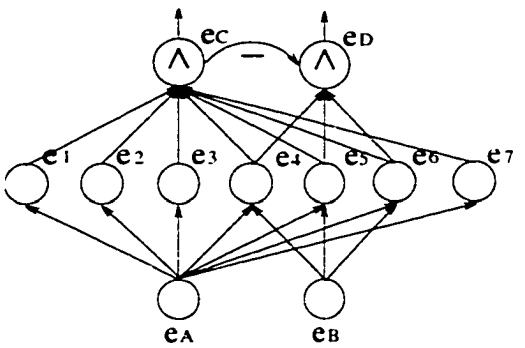


Figure 3.16: Inheritance Cancellation for The Cases 1 and 2

The above cases deal with only either A or B has property value. If both of A and B have property values C and D respectively then activation of A may activated both of C and D. Therefore, CONSYDERR deals with cancellation of property inheritance. The following four cases are discussed:

1. A has a property value C and B has a property value D and $D \neq C$. If A is activated then activation of C is stronger than D.

2. A has a property value C and B has a property value D and $D \neq C$. If B is activated then activation of D is stronger than C.

3. A has a property value C and B has a property value D which is a subclass of C. If A is activated then activation of C is stronger than D.

4. A has a property value C and B has a property value D which is a subclass of C. If B is activated then activation of D is stronger than C.

In LSCAN, the cases 1 and 2 are handled as shown in Figure 3.16. For the case 1, e_A activates all of the belonging features $e_{1..7}$ which activate e_C and e_D to the same activation level. However, the degradation of e_D from e_C will cease the activation of

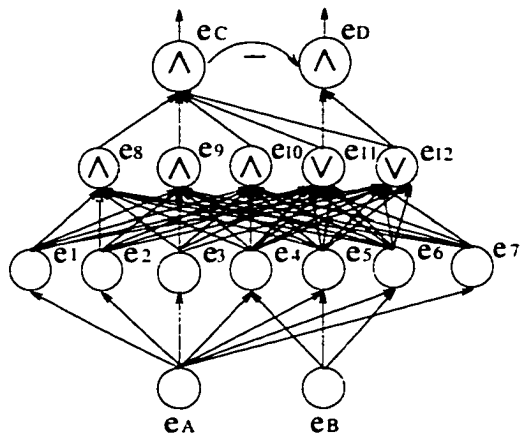


Figure 3.17: Inheritance Cancellation for The Cases 3 and 4

D. In case 2, e_B is activated. In this case, e_D is activated at its full strength and e_C is only activated at a fraction of its full strength. Therefore, the activation strength of e_D is stronger than e_C. The cases 3 and 4 are handled in Figure 3.17. In these two cases, e_C and e_D have their own features, $e_{8..12}$ and $e_{10,11}$, which interact with the features of e_A and e_B. While e_A is activated, both of e_C and e_D are activated through $e_{8..12}$. However, e_C degrades e_D to cease activity. If e_B is activated, then e_D is activated to its full strength through $e_{10,11}$. The activation strength of e_C is much weaker than the strength of e_D.

3.7 LSCAN Knowledge Acquisition and Processing Tool

The LSCAN simulation tool is a Windows-based software using windows standard user interface. This tool is a prototype to simulate LSCAN networks for discrete nodes, not aggregate nodes. However, aggregate nodes can be further developed. The purpose of the simulation tool is to examine the needs for LSCAN networks and further requirements for future developments.

The software was designed using MS Windows standard user interfaces to allow users to build LSCAN networks and implement them for testing and analysis. The first user interface is a dialog window as shown in Figure 3.18. Users can define the properties for an E-node, add a new E-node, or modify an existing E-node. The properties of an E-node include name, node type, value type, logic type, input link type, and other control parameters. A node type is one the six types: constant, variable, function, decision, digital (binary), and construct. A constant type means an E-node carries a constant value. A variable type means the values of an E-node can be binded with any values at processing time. A function type means that an E-node derives its own value from a given function. A decision type means that an E-node implements Boolean logic. A construct type means an E-node implements an external script. A value type can be an integer, a real number, a binary logic value which is either 0 or 1, a real logic value which is in the range of [0, 1], or a character string. The logic type is either a rigid logic or a soft logic. A rigid logic has a value of 0 or 1. A soft logic has a value between 0 and 1 inclusive. There are four kinds of input connections. The NIL link means an E-node has no inbound connections, which is a leaf node. The AND link means an E-node has AND connections, which implements the AND evaluation function. The OR link means an E-node has OR connections, which implements the OR evaluation function. The SINGLE link means an E-node has only one input connection. In the most cases, a SINGLE link E-node may be a variable to bind the value from its input node. The control parameters are self-explained from their labels.

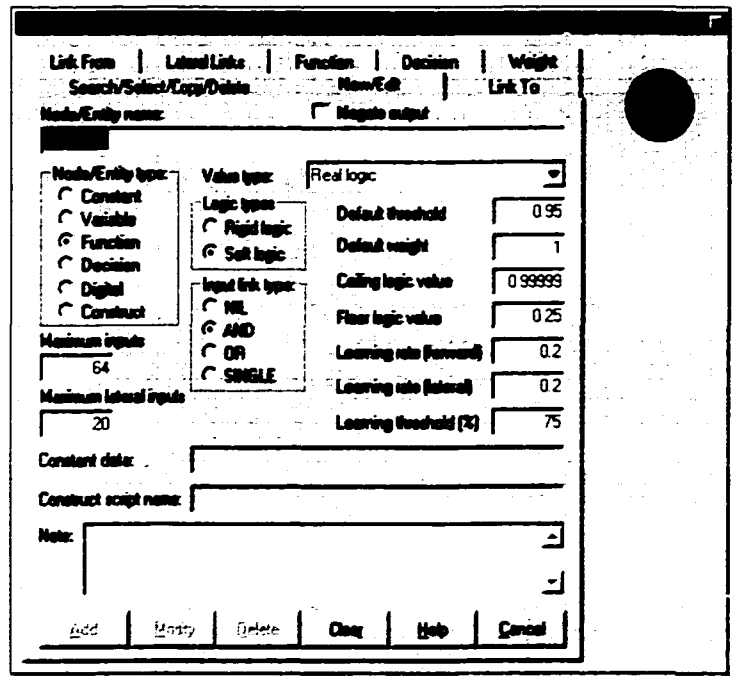


Figure 3.18: The New E-node and Edit E-node Interface

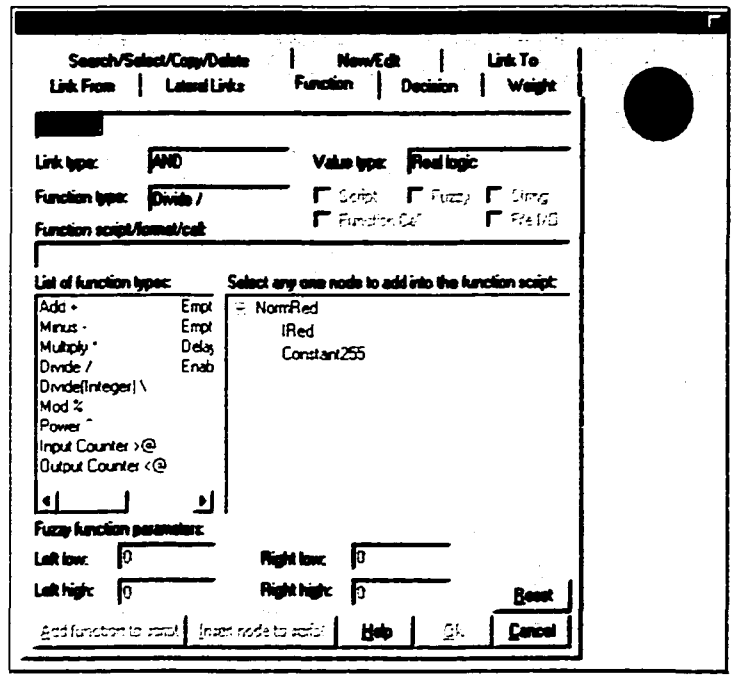


Figure 3.19: The Function Edit Interface

The second user interface shown in Figure 3.19 is a dialog window to define the function type for an E-node. LSCAN tool provides some build-in functions, or allows users to specify a function script, a formatting function, or an external function call using DLL techniques. If a function is selected as a Fuzzy logic function, the required Fuzzy logic parameters are inquired. The third user interface is shown in Figure 3.20. Through this dialog window, users can establish inbound connections for the specified E-node. Inbound connections can be added or delinked. The forth user interface is shown in Figure 3.21. Through this dialog window, users can establish outbound connections for the specified E-node. Outbound connections can be added or removed. Using these two dialog windows, users can double check whether the inbound and outbound connections are well established. The fifth user interface is the connection weights setup dialog window as shown in Figure 3.22. All default connection weights are 1. Through this dialog window, uers can assign a different weight value to a specific connection.

After a LSCAN network is established, users can view the entire network or networks through two view windows. One window shows the list view of the all E-nodes. Within this view window, some properties of the all E-nodes are displayed at the right side of the view window. One example list-view window is shown in Figure 3.23. The other view window is the tree view of the all E-nodes. The tree-view window shows how the E-nodes are connectioned. The tree-view allows users to expand or collapse any E-node if such an E-node has inbound connections. One example tree-view window is shown in Figure 3.24. There are many other features of this LSCAN tool. However, they are not included in this text due to space limitation.

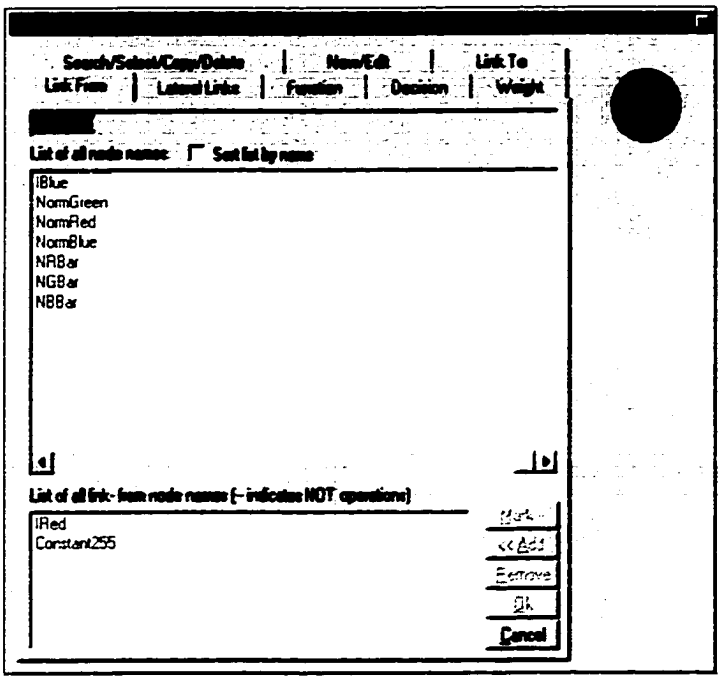


Figure 3.20: The Link-From Interface

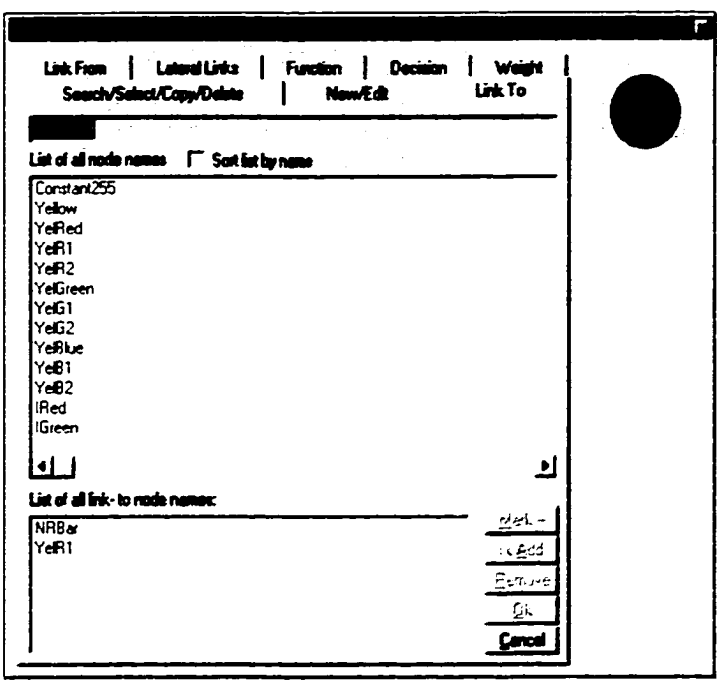


Figure 3.21: The Link-To Interface

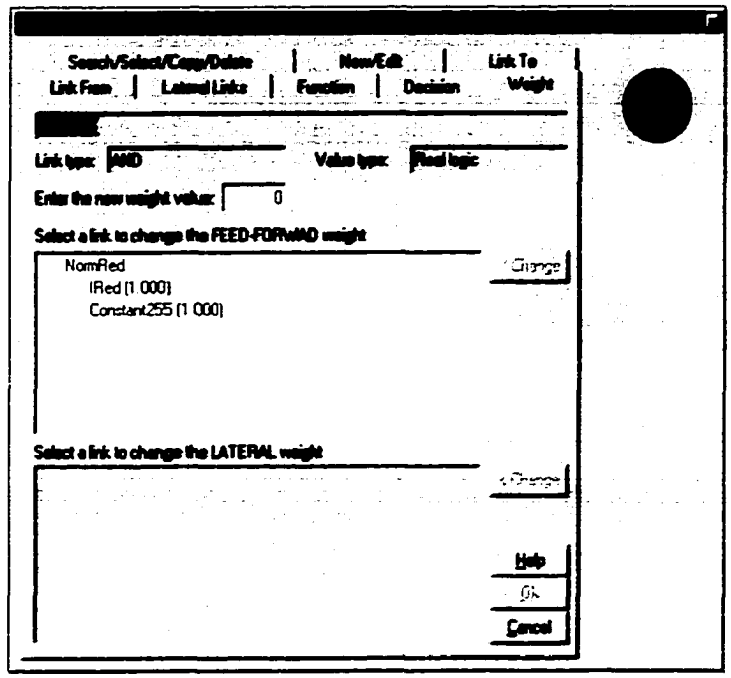


Figure 3.22: The Weight Edit Interface

3.8 Summary

This chapter introduced the proposed LSCAN paradigm. The background motivation of the deriving AND and OR evaluation nodes was presented in the formalization of knowledge representations. The complete AND and OR nodes include feed-forward connections and lateral connections. Next, their feed-forward evaluation functions and lateral interaction functions were defined. The NOT operators are the 1's complement of the logical outputs of any AND or OR nodes. Learning mechanisms for both of feed-forward and lateral connections were defined. For the feed-forward learning mechanisms, two cases would be considered. One case is when the input nodes are not all active. In this case, two learning mechanisms are used. One mechanism is for the active nodes and the other mechanism is for the inactive nodes. The active nodes gain more connection weights from the inactive node connections. The

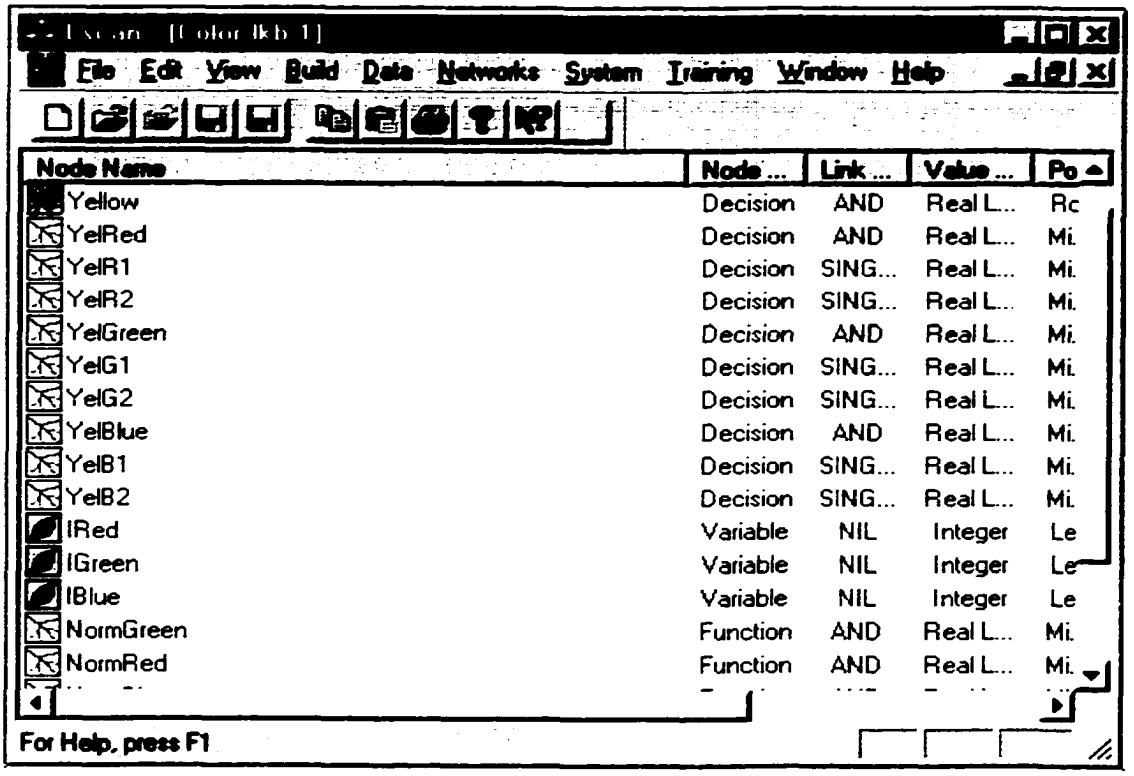


Figure 3.23: A Network List View

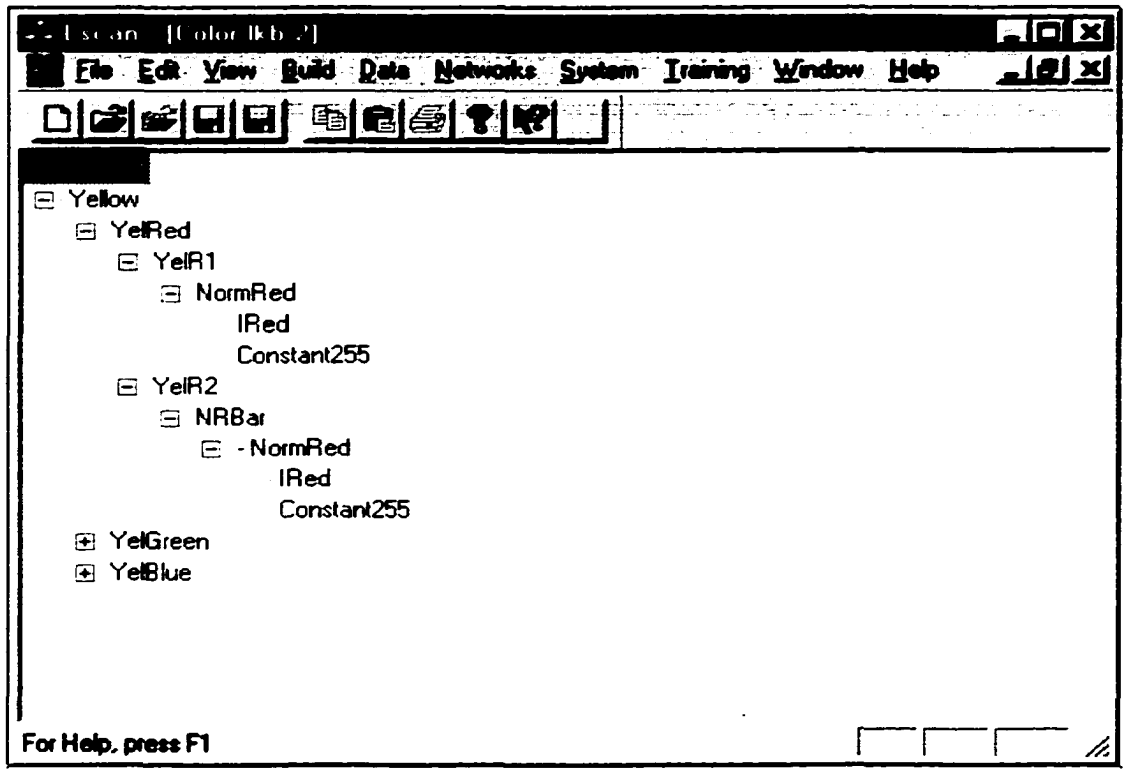


Figure 3.24: A Network Tree View

second case is when all of the input nodes are active. In this case, only one learning mechanism is used. The stronger active node connections gain more connection weight from the weaker node connections. Therefore, the stronger active nodes gain more activation than the weaker ones. All learning mechanisms can be implemented under the supervision or without supervision. The lateral interaction learning mechanisms were also defined. Two types of lateral learning mechanisms were provided. One is the competitive learning mechanism. In the competitive learning mechanism, the winner nodes prohibit activations of the other nodes, which depending on the lateral connections. Conversely, the type of the lateral learning mechanisms is the enhancement learning mechanism. Some active nodes would enhance the activations of some other nodes, which may be active or inactive. A LSCAN subnetwork can add some bias nodes to accelerate the learning process as discussed in Section 3.5.3.

Sun's CONSYDERR networks have some similarities to the LSCAN representations. The two aspects of similarity and inheritance can be compared to the LSCAN representations as discussed. Finally, the LSCAN knowledge acquisition and processing tool was presented with some explanations and illustrations.

CHAPTER 4

EXTENDED FEED-FORWARD LEARNING TECHNIQUES

This chapter demonstrates how to use the feed-forward learning method given in Section 3.5.1 to build networks as classifiers, switches, and pattern recognizers. These networks are considered as subsymbolic networks which can be integrated into the LSCAN hybrid structure. In order to build these networks, the concepts of high pass filter, low pass filter, and band pass filter are explained as the base of constructing the mentioned networks. Each category of these networks: classifiers, switches, and pattern recognizers, has been illustrated with an example.

4.1 Filters

This section demonstrates how to train a single input variable to be a high pass filter, a low pass filter, or a band pass filter. Refer to Section 3.5.1 on the training mechanism, an E-node is trained to be active while a sequence of inputs, which have values between 0 and 1, are applied. The inputs activate this trained E-node with a strength above the threshold. In this sense, the trained E-node is acting as a high pass filter. The Figure 4.1 illustrates the structure of the high pass filter. In a rapid training procedure, the network needs to be trained with the lowest value to form a

high pass filter. In Figure 4.1, the E-node *HPF* is trained with the value of 0.3 from the input E-node *Var*. The final weight of the link is 3.333.

On the other hand, the E-node *LPF* is trained with the one's complement of the value of the E-node *Var*, which is $\overline{Var} = 1 - Var$. The highest value of the E-node *Var* is the lowest value of the E-node $\overline{Var}$. As illustrated in the Figure 4.2, in the viewpoint of the E-node $\overline{Var}$, the E-node *LPF* is a high pass filter. However, in the viewpoint of the E-node *Var*, the E-node *LPF* is a low pass filter. The E-node *LPF* in Figure 4.2 is trained with the value of 0.4 from the E-node *Var*. The link weight is 1.6 between the E-node *LPF* and the E-node $\overline{Var}$. The *LPF* is active while the values of *Var* are less or equal to 0.4. The output strength of *LPF* reduces until failure to be active depending on where the threshold is set, if the values of *Var* exceed 0.4.

Merging the above two filters forms a band pass filter as shown in Figure 4.3. The E-node *HPF* is trained as a high pass filter with the value of 0.3 from the E-node *Var*. The trained weight is 3.333. The E-node *LPF* is trained as a low pass filter with the value of 0.5 from the E-node *Var*. The trained weight is 2.0. If the value of *Var* is lower then 0.3, the E-node *HPF* is inactive and the E-node *LPF* is active. In this case, the E-node *BPF* fails to be active. On the other hand, if the value of *Var* goes above 0.5, the E-node *HPF* is active and the E-node *LPF* is inactive. In this case, the E-node *BPF* fails to be active. However, if the value of *Var* is between 0.3 and 0.5, both E-nodes *HPF* and *LPF* are active. Therefore, the E-node *BPF* is activated and behaves a band pass filter. All examples used for the three filters can be extended for multiple inputs.

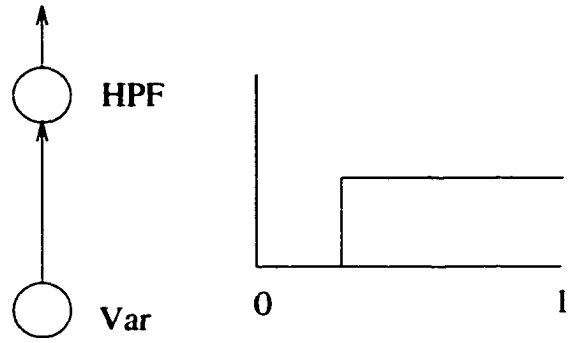


Figure 4.1: A High Pass Filter Structure

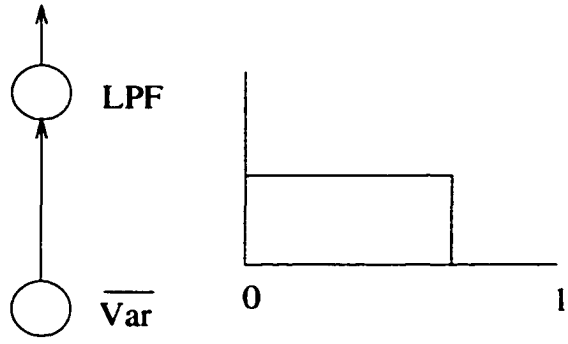


Figure 4.2: A Low Pass Filter Structure

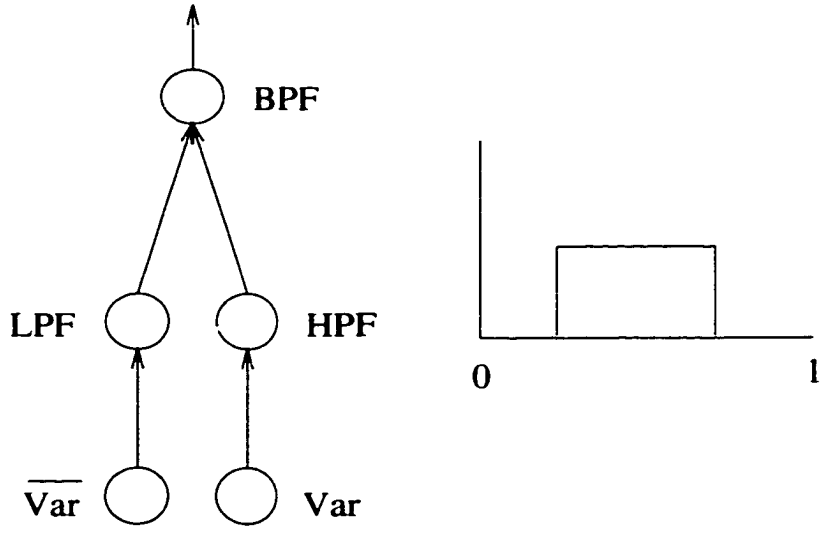


Figure 4.3: A Band Pass Filter Structure

4.2 Classifiers

Based on the concept of the filters, this section demonstrates design of a network to classify the color *Yellow* based on the strengths of the three elementary colors: Red, Green, and Blue. As shown in the Figure 4.4, the three input E-nodes: IR, IG, and IB, represent the three elementary colors and have integer values between 0 and 255. These input data are normalized by the E-nodes: *NormR*, *NormG*, and *NormB*, with the constant 255. These three normalization E-nodes generate the strengths of the three elementary colors with values between 0 and 1. The three E-nodes: $\overline{NormR}$, $\overline{NormG}$, and $\overline{NormB}$, act as the inverters of *NormR*, *NormG*, and *NormB*. Their outputs are the 1's complements of the normalization. The three E-nodes: *YelR1*, *YelG1*, and *YelB1*, are trained as three high pass filters with their input link weights which are 1.067, 1.063, and 51.0, respectively. The three E-nodes: *YelR2*, *YelG2*, and *YelB2*, are trained as three low pass filters with their input link weights which are 15.938, 17.0, and 1.02 respectively. The input data used for this training are: 239, 240, and 5, for the three elementary colors respectively. In this case, a single set of data is used to train the network. Therefore, the three E-nodes: *YelR*, *YelG*, and *YelB*, act as three *pulse* filters. The output E-node, *Yel* which represents the yellow color, is activated only when the input data has the values as the training data set. Otherwise, it fails to be active. For example, the testing data set is (230, 252, 11). The E-node *Yel* has an output strength of 0.639 which fails to be active. The other testing data set is (230, 252, 35). The E-node *Yel* has an output strength of 0.002 which is further weakened.

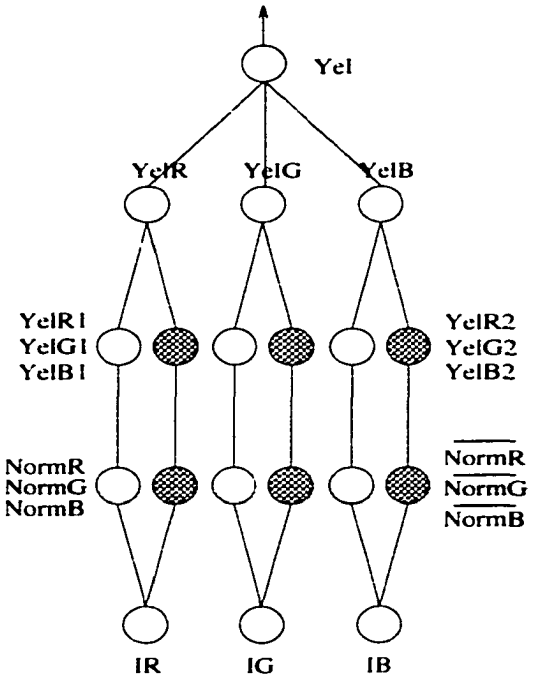


Figure 4.4: A Yellow Color Classifier

4.3 Switches

The color classifier illustrated in last section shows no significance with respect to the input locations, the three inputs can be arranged in any sequences, the results are the same. In this and next sections, the two examples demonstrate the results of the two networks are input location sensitive. As the example shown in the Figure 4.5, the network is trained to switch one of the four inputs to the assigned one of the four outputs. The four input E-nodes are: *Var1*, *Var2*, *Var3*, and *Var4*. The four output E-nodes are: *Dest1*, *Dest2*, *Dest3*, and *Dest4*. The four middle E-nodes, *Dest11*, *Dest21*, *Dest31*, and *Dest41*, are trained as high pass filters. The four middle E-nodes, *Dest12*, *Dest22*, *Dest32*, and *Dest42*, are trained as low pass filters. The training process takes four stages. For each set of the input data, a pair of the middle E-nodes of one of the four output E-nodes are trained as a pair of high pass and low

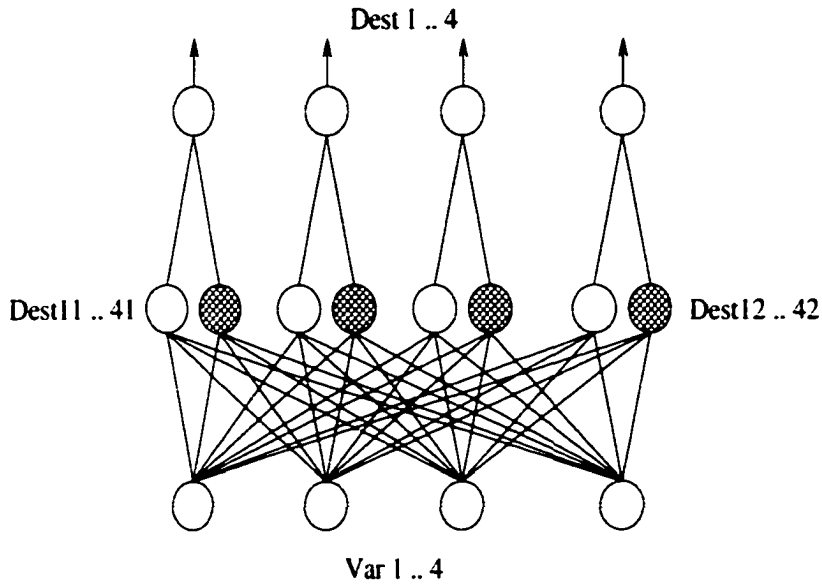


Figure 4.5: A Four To Four Switch

pass filters. Table 4.1 shows the input and output data sets and the trained weights for the middle E-nodes. From the trained weights, it shows that the network resists noisy inputs. For example, if the first input data set becomes 0110, then, both of the two E-nodes, *Dest11* and *Dest12* fail to be active.

4.4 Pattern Recognizers

This section demonstrates a network to recognize three different patterns in a 4×4 grids. There are three output E-nodes which represent the three patterns. The three patterns are shown in Figure 4.6. The third pattern has two variations as shown in Figure 4.6. For each pattern, a pair of the middle E-nodes are trained. The pattern recognizer network is shown in Figure 4.7. The training results are shown in Table 4.2.

Table 4.1: Training Results for The Switch Example

Input	Output	Weight				
0010	1000	Dest11	-7.805	-7.805	3.977	-7.805
		Dest12	1.323	1.323	-24.775	1.323
0001	0100	Dest21	-7.805	-7.805	-7.805	3.977
		Dest22	1.323	1.323	1.323	-24.775
0100	0010	Dest31	-7.805	3.977	-7.805	-7.805
		Dest32	1.323	-24.775	1.323	1.323
1000	0001	Dest41	3.977	-7.805	-7.805	-7.805
		Dest42	-24.775	1.323	1.323	1.323

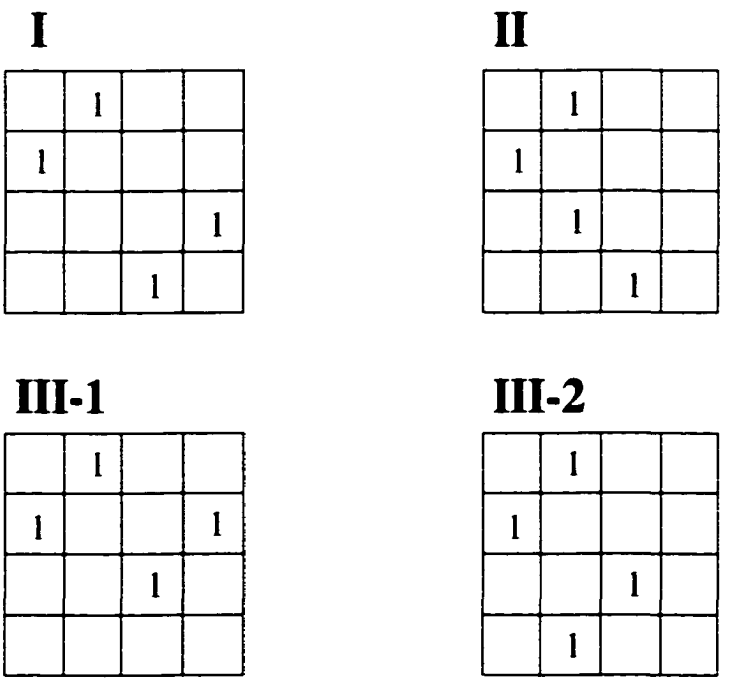


Figure 4.6: Three Training Patterns

Table 4.2: Training Results for The Pattern Recognizer Example

Pattern	Weight				
0100100000010010	Pat11	0.010	3.971	0.010	0.010
		3.971	0.010	0.010	0.010
		0.010	0.010	0.010	3.971
		0.010	0.010	3.971	0.010
	Pat12	1.323	-5.444	1.323	1.323
		-5.444	1.323	1.323	1.323
		1.323	1.323	1.323	-5.444
		1.323	1.323	-5.444	1.323
0100100001000010	Pat21	0.010	3.971	0.010	0.010
		3.971	0.010	0.010	0.010
		0.010	3.971	0.010	0.010
		0.010	0.010	3.971	0.010
	Pat22	1.323	-5.444	1.323	1.323
		-5.444	1.323	1.323	1.323
		1.323	-5.444	1.323	1.323
		1.323	1.323	-5.444	1.323
0100100100100000	Pat31	-0.322	4.966	-0.322	-0.322
0100100000100100		4.966	-0.322	-0.322	3.639
		-0.322	-0.322	4.966	-0.322
		-0.322	1.005	-0.322	-0.322
	Pat32	1.888	-16.716	1.888	1.888
		-16.716	1.888	1.888	-4.879
		1.888	1.888	-16.716	1.888
		1.888	-9.950	1.888	1.888

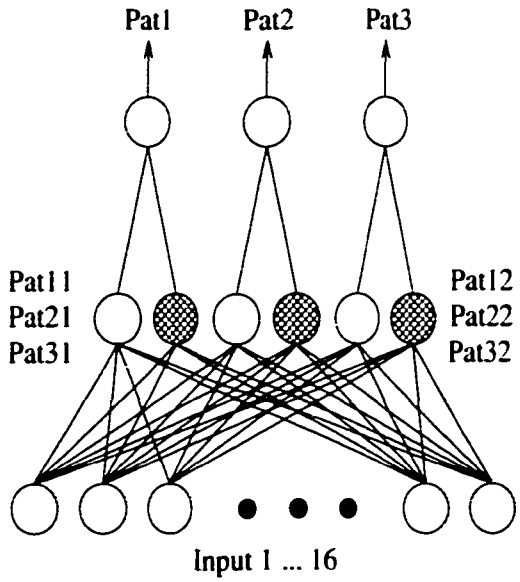


Figure 4.7: A Pattern Recognizer

4.5 Summary

A LSCAN subnetwork can learn a concept or event directly from its input nodes if the activation strength is equal to or greater than the threshold θ_h. After learning, all activation strengths above the threshold activate the output node of the subnetwork. In the concepts of filtering, the output node acts as a high pass filter. On the other hand, if all of the strength values of the input nodes are inverted by subtracting from 1 then the LSCAN subnetwork learns the complement of a concept by passing its activation strength over the threshold θ_l. In this case, the output node of this subnetwork acts as a low pass filter. If a LSCAN subnetwork uses both of positive and inverted input nodes, then the subnetwork can be trained as a band pass filter. By using these techniques, various applications can be developed according the needs of the problem space.

Sections 4.2, 4.3, and 4.4, showed three examples of applications. These subnetworks can be embedded into a larger LSCAN network to perform more complex reasoning tasks. Using the concept of a band pass filter, a subnetwork can be trained as a pulse filter as illustrated in Section 4.2. Therefore, a concept can be learned sharply (pulse filter), accurately (band pass filter), or loosely (high or low pass filter).

CHAPTER 5

APPLICATION TO HAND-PRINTED DIGIT CLASSIFICATION

This case study includes two methods, one uses pattern-based reasoning and the other uses rule-based reasoning, which demonstrate the flexibility of LSCAN to be used for variety classification methods and derive satisfactory recognition results. The pattern-based classification is constructed with a network in several layers which are similar to other feed-forward neural networks. The rule-guided classification method incorporates cognitive rules from the input data. The rules are used as inputs to the classification network which contains two layers, one input layer and one output layer.

The training and testing data were obtained from NIST (National Institute of Standards and Technology) database [36] which contains 3,471 hand-printed digits from 49 individuals. Each digit was scanned as a binary image in 32 by 32 matrix. Figure 5.1 shows 88 hand-printed digits from a single person. Both of the pattern-based and the rule-guided classification networks use on-line training. On-line training contains no classes or patterns at the initial state and has no knowledge about the input digits. This kind of learning is called unsupervised learning. The networks train themselves by growing the required number of classes while there remains some unclassified digits. In the pattern-based network approach, the number of patterns

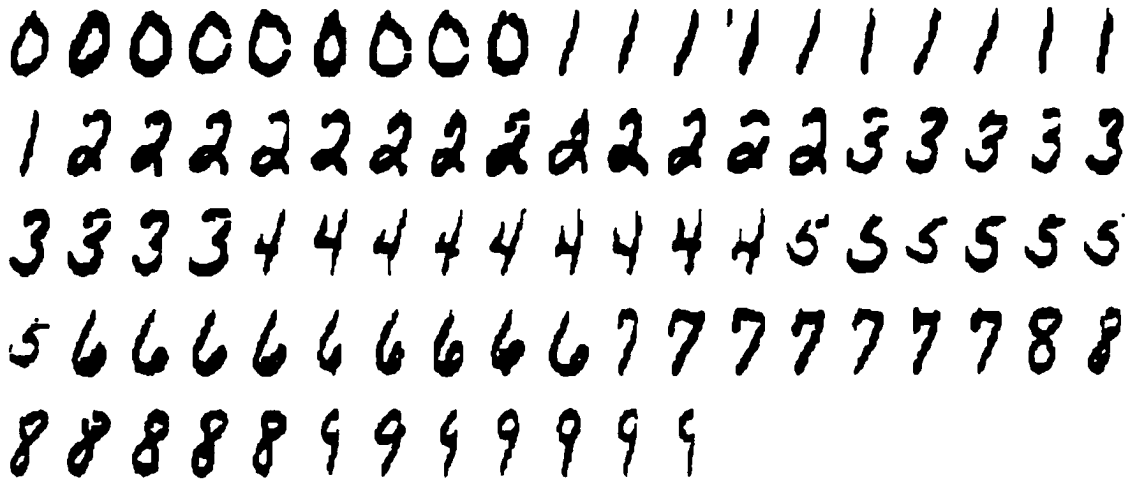


Figure 5.1: Examples of Some Hand-Printed Digit Images

grows while additional new classes are recognized. In the rule-guided network approach, the number of rules are fixed in the initial state. Therefore, the number of input nodes are not changed during execution.

5.1 Pattern-Based Classification

Pattern-based classification has been implemented by Banarse [3] in his dissertation. In this case study, a similar approach is adapted to extract features from the input image and to form patterns from the extracted features using LSCAN. The LSCAN pattern-based networks has similar structure as Banarse's P_ARADISE networks. However, they differ in the activation functions of the output nodes at each layer and in some of the control parameters. Furthermore, there are some bias nodes connected to each of the classification nodes.

5.1.1 Network Structure

The network structure contains four layers: input layer, feature layer, pattern layer, and classification layer, as illustrated in Figure 5.2. The input layer contains a 32x32 node matrix which read the binary input image. The feature layer contains two feature planes which represent the two orientations of the feature extractions. The two orientations are 0 and 90 degrees. The 0-degree orientation filter extracts features for vertical lines and the 90-degree orientation filter extracts features for horizontal lines. The pattern layer contains no pattern nodes at the beginning of training. A new pattern is created if all of the existing patterns failed to represent the new pattern which is extracted from the two feature planes. Each pattern unit structure contains two sub-layers. The output sub-layer contains one pattern node P_m which represents the pattern. The pattern node input sub-layer is composed of a set of invariant pattern evaluation nodes or an invariant plane as shown in Figure 2.5. Each invariant node PI_{ij}, where $i = 1...2\gamma+1$, $j = 1...2\gamma+1$, maps its inputs from the specific areas of the two feature planes. The size of each invariant plane is controlled by the variable γ which can be either an odd or an even number. The purpose of the invariant plane is to detect the pattern while the corresponding features shift near the location where the pattern was first detected. The size of γ also decides how much of a pattern is shared by the other digit images. Which means more pattern sharing, or larger γ, then the less number of patterns generated. The classification layer has inputs from the pattern layer. Initially, there are no classes in the initial state. While an input digit image is presented to the network, if there is no class which can best represent the input digit then a new class is created with the all input connections to the activated patterns.

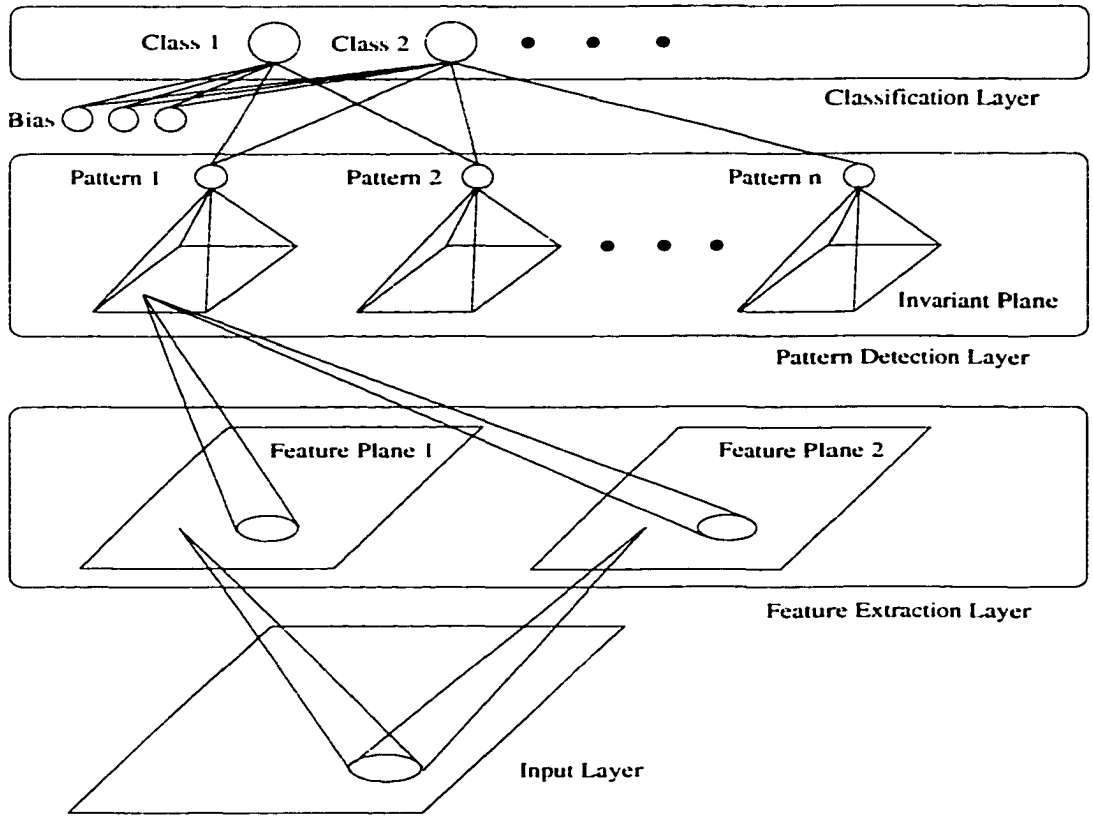


Figure 5.2: The Pattern-Based Network

5.1.2 Feature Extraction

The features of a digit image are extracted by using Gaussian filters. A Gaussian filter uses a $\beta \times \beta$ window which scans through the input image plane as shown in Figure 5.3. The size of β is always picked as an odd number. The results of the scanning window are recorded in the feature planes. Each feature plane has the same size as the input image plane. The feature point F_{xy} at (x, y) in a feature plane is mapped to the region R_{xy} in the input image plane. The region R_{xy} has the same size as the scanning window W_β. Therefore, the region R_{xy} covers the area with locations I_{uv}, where $u = y - (\beta - 1)/2...y + (\beta - 1)/2$ and $v = x - (\beta - 1)/2...x + (\beta - 1)/2$. The Gaussian filter is applied to all regions R_{xy}. A Gaussian filter is a 2D Gaussian function plus orientations. The function is given as:

$$G_{uv} = exp(-\frac{(v\cos\theta + u\sin\theta)^2}{2\sigma_v^2} - \frac{(-v\sin\theta + u\cos\theta)^2}{2\sigma_u^2}) \tag{5.1}$$

where θ is the orientation. In this case study, two orientations, 0 and $\frac{\pi}{2}$ are used. Figure 5.4 and Figure 5.5 show the plots for the two orientations of Gaussian filters using $\beta = 13, \sigma_u = 4$, and $\sigma_v = 2$. The Gaussian feature strength at the location (x, y) in a feature plane is given:

$$F_{xy} = \sum_{u=y-(\beta-1)/2}^{y+(\beta-1)/2} \sum_{v=x-(\beta-1)/2}^{x+(\beta-1)/2} I_{uv}G_{uv} \tag{5.2}$$

5.1.3 Pattern Detection

A pattern is formed by the areas which have the same locations on the feature planes as shown in Figure 5.2. As shown in Figure 2.5, each area on the feature plane has a radius λ. To select a pattern is to pick up the areas where have the maximum

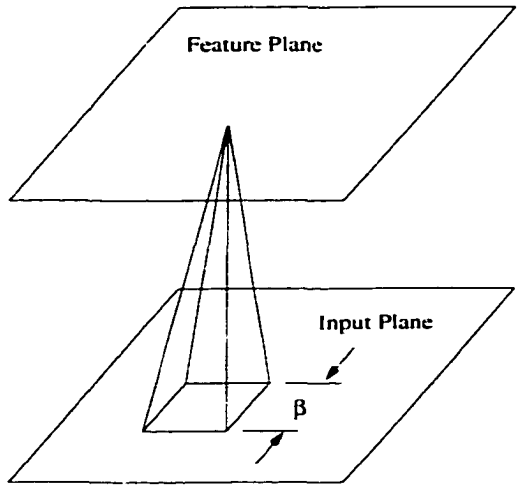


Figure 5.3: A Structure for Feature Extractions

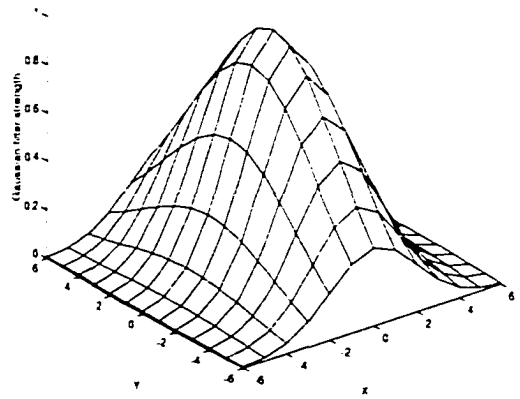


Figure 5.4: Gaussian Filter with 0-Degree Orientation

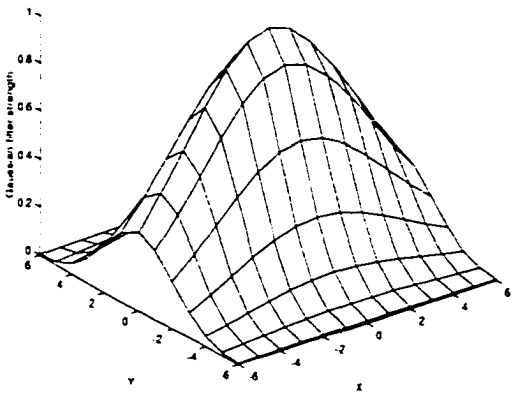


Figure 5.5: Gaussian Filter with 90-Degree Orientation

average feature strength. The pattern selection function is given:

$$Ps_{xy} = \frac{\sum_{k=1}^{K} \sum_{m=-\lambda}^{\lambda} \sum_{n=-\lambda}^{\lambda} F_{mn}^{k} g_{mn}}{K(2\lambda+1)^2} \tag{5.3}$$

where K is the number of feature planes and g_{mn} is the Gaussian weight which is given as:

$$g_{mn} = e^{\{-\frac{m^2+n^2}{2\sigma^2}\}} \tag{5.4}$$

If $Ps_{xy} > \nu$, where $0 < \nu < 1.0$, then the pattern is selected. Once a pattern is selected, the corresponding areas on the feature planes are masked with the 1's complement of the Gaussian weight function 5.4. The 1's complement of the Gaussian weight function is expressed as

$$1 - g_{mn} \tag{5.5}$$

The purpose of this masking is to prevent the same areas on the feature planes to be selected as another pattern.

If there are some pattern modules in the pattern detection layer, the selected pattern from the feature planes is examined by the existing pattern modules. If the

selected pattern is not presented by any of the existing pattern modules, then a new pattern module is created at the time of a new class creation. A new pattern module as illustrated in Figure 2.5 is created with one set of connections, a size of $K(2\lambda+1)$. The weights of the connections cast the product of the feature strengths and the Gaussian weight function 5.4 as given by

$$w_{mn}^{k} = \rho F_{mn}^{k} g_{mn} \tag{5.6}$$

where ρ is a factor to control the feature strengths. The values of ρ are between 0 and 1.

Detecting a pattern is done by finding a representing node PI_{ij}^{win} in the invariant plane of a pattern detection module (PDM) for the selected pattern drawn from the feature planes. The representing node PI_{ij}^{win} is the winning node which has the maximum output strength among the all invariant nodes in the invariant plane. Furthermore, the strength of PI_{ij}^{win} has to pass the preset threshold ϑ_p. Therefore, the activation decision of PI_{ij}^{win} is given by

$$PI_{ij}^{win} = \begin{cases} active & \text{if} \geq \vartheta_p - \tau_p \\ inactive & \text{otherwise} \end{cases} \tag{5.7}$$

where the factor τ_p is less than ϑ_p and is used to control pattern sharing between numerals. Each invariant node PI_{ij} is evaluated as

$$PI_{ij} = \xi(\varepsilon_{ij}) \tag{5.8}$$

where ξ is a function of ε as expressed in Equation 3.2 and 3.3. The factor ϵ in Equation 3.3 is evaluated as

$$\epsilon_{ij} = \sum_{k=1}^{K} \sum_{m=-\lambda}^{\lambda} \sum_{n=-\lambda}^{\lambda} (\vartheta_p - dif_{mn}^{k}) \tag{5.9}$$

where dif^k_{mn} is the difference between the connection weight w^k_{mn} and the input feature strength $\rho F^k_{mn} g_{mn}$ at the k_{th} feature plane. It is expressed as

$$dif^k_{mn} = abs(w^k_{mn} - \rho F^k_{mn} g_{mn}) \tag{5.10}$$

If all of dif^k_{mn} equal to 0, then the selected pattern Ps_{xy} has a perfect match with the corresponding PI_{ij} in the invariant plane.

Pattern sharing between numerals can be controlled in two manners: one in a spacial aspect and the other in a pattern strength aspect. In a spacial aspect, pattern sharing is controlled by the variable γ. The larger size of γ then the larger chances are for a pattern to be shared with the other numerals. This result is due to increase in the total areas covered by the invariant pattern nodes in a PDM. In a pattern strength aspect, pattern sharing is controlled by the variable τ_p which lowers down the activation threshold as indicated in Equation 5.7. Therefore, more invariant pattern nodes PI_{ij} can be activated.

5.1.4 Classification

A class node C_n is activated by a set of active pattern nodes $P_{m(active)}$ which are connected to C_n. C_n is evaluated as:

$$C_n = \xi(\varepsilon_n) \tag{5.11}$$

where ξ is a function of ε as expressed in Equation 3.2 and 3.3. The factor ϵ in Equation 3.3 is evaluated as:

$$\epsilon_n = \sum_{i=1}^{M} P_i \tag{5.12}$$

C_n is activated if

$$C_n = \begin{cases} active & \text{if} \geq \vartheta_c \tau_c \\ inactive & \text{otherwise} \end{cases} \tag{5.13}$$

where the factor τ_c is a percentage of ϑ_c and is used to increase the number of active classes if the value of τ_c is smaller. The purpose of using τ_c is due to the pattern sharing problem. Patterns are shared between the numerals. Therefore, a class can be mis-activated by a different digit numeral. Thus, it is not sufficient to select an active class to represent the input digit image alone. One needs to examine another factors to make the decision of whether an active class node can represent the input numeral. There are two factors used in the experiments. One factor is the ratio of the number of active patterns connected to the class node C_n and the total number of patterns connected to the class node C_n. This ratio μ_c is expressed as:

$$\mu_c = \frac{P_{(total-active-in-C_n)}}{P_{(total-pattern-in-C_n)}} \tag{5.14}$$

The other factor is the ratio of the total number of patterns connected to the class node C_n and the total number of feature patterns of the input digit image. This ratio μ_f is expressed as:

$$\mu_f = \frac{P_{(total-patterns-in-C_n)}}{P_{f(total-feature-patterns)}} \tag{5.15}$$

The final selected class node C_f has to satisfy the conditions of $\nu_c > \vartheta_{\mu c}$ and $abs(1 - \nu_f) < \vartheta_{\mu f}$. Where $\vartheta_{\mu c}$ and $\vartheta_{\mu f}$ are preset thresholds for μ_c and μ_f respectively.

5.1.5 Class Creation

A new class is created at the beginning of the on-line learning and while there is no existing node which can represent the input digit image. A new class is formed by a set of active patterns which include the new patterns. A new class is

trained with the connected active patterns by adjusting the connection weights using the Equations 3.11 and 3.12. Once a new class is trained then no further training is required.

5.1.6 Classification Experiments

In this case study, the digit images shown in Figure 5.1 were used for the classification experiments. All experiment results were one-pass learning results unless otherwise specified. Two kinds of variables, spacial variables and functional variables, have influence on the outcomes of the experiments. The follows describe the effectiveness of the two kind of variables:

- Spacial variables:

  θ: The value of θ changes feature orientations. Two θ values, 0 and $\frac{\pi}{2}$, are used in this case study.

  β: The size of β changes the strengths of features.

  λ: The size of λ changes the number of patterns extracted from the feature planes.

  γ: The size of γ changes pattern sharing possibilities.

- Functional variables:

  ρ: Changing feature strengths may get patterns similar to each other, therefore, fewer patterns are generated.

  ν: Feature strength average threshold changes the number of patterns extracted from the feature planes.

ϑ_p: The pattern node threshold is preset at 0.70.

ϑ_c: The class node threshold is preset at 0.85.

$\vartheta_{\mu c}$: Thresholds for μ_c change the number of active class nodes.

$\vartheta_{\mu f}$: Thresholds for μ_f change the number of active class nodes.

A large number of experiments were implemented for this small data set in order to observe the changes of outcomes of the experiments while changing the values of the spacial and functional variables.

Some experiments were implemented to observe the influence of the functional variables. First, the spacial variables were fixed at the values of $\beta = 15, \lambda = 5$, and $\gamma = 7$. Some of the functional variables had values at $\nu = 0.10, \vartheta_{\mu c} > 0.7$, and $\vartheta_{muf} < 0.3$. Figure 5.6 shows the results of variations of controlling of the variable τ_c with $\rho = 0.175$ and $\tau_p = 0.3$. Figure 5.7 shows the results of variations of controlling of the variable τ_p with $\rho = 0.175$ and $\tau_c = 0.3$. Figure 5.8 shows the results of variations of controlling of the variable ρ with $\tau_c = 0.5$ and $\tau_p = 0.3$. All figures were plotted for the number of images against the percentages. The curves labeled as *Class* are the ratios of the number created classes over the number of input images. The curves labeled as *Accuracy* are the ratios of $\frac{I-M_c}{I}$, where I is the number of input images and M_c is the number of misclassified classes. All ratios are expressed in percentages. For spacial variables, γ, λ, and β, Table 5.1 shows the experiment results. The functional variable values were fixed at: $\rho = 0.16, \nu = 0.08, \tau_c = 0.3$, and $\tau_p = 0.4$.

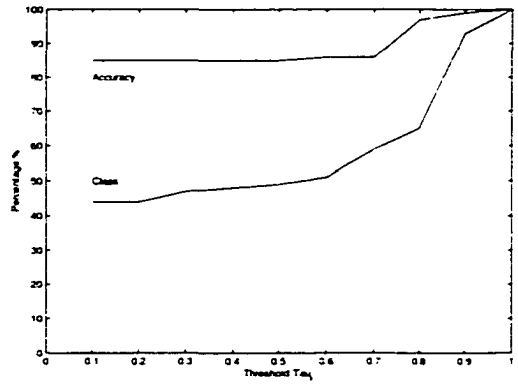


Figure 5.6: Experiment Results for The Variable τ_c

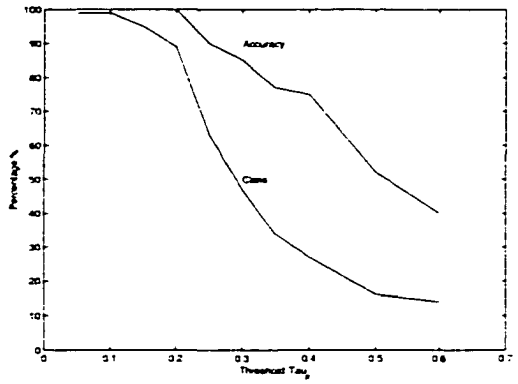


Figure 5.7: Experiment Results for The Variable τ_p

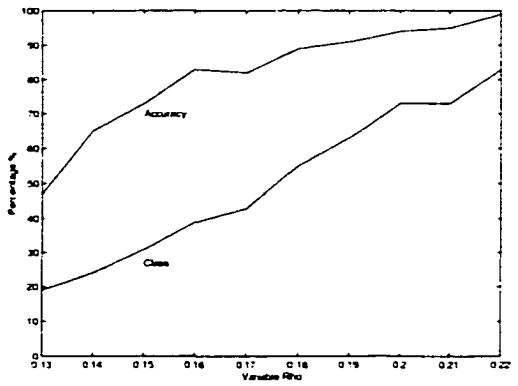


Figure 5.8: Experiment Results for The Variable ρ

Table 5.1: Experiment Results for Spacial Variables

$(2\gamma + 1) \times (2\lambda + 1) \times 3$	Class/Missed	Class/Accuracy %
13x11x13	19/25	22/72
13x13x13	8/55	9/38
13x15x13	6/58	7/34
15x11x13	13/27	15/58
15x13x13	6/66	7/25
15x15x13	5/65	6/26
13x11x15	19/26	22/70
13x13x15	9/54	10/39
13x15x15	5/64	6/27
15x11x15	15/31	17/65
15x13x15	5/72	6/18
15x15x15	4/59	5/33

Table 5.2: Experiment Results for High Accuracy

	$\vartheta_{\mu f}$						
$\vartheta_{\mu c}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7 - 1.0
0 - 0.6	64/3	57/4	51/6	46/15	41/22	38/27	38/27
0.7	67/4	60/5	54/6	47/17	42/24	39/27	37/28
0.8	72/2	65/3	59/6	55/15	50/20	47/20	45/21
0.9	84/0	82/0	79/0	78/1	75/4	73/6	72/7

Experiments With Bias Nodes

With bias nodes added to the classification layer, the new class nodes are trained to change their connection weights using the Equations 3.11 and 3.12. The bias nodes have constant values of 0. The purpose of the bias nodes is to stabilize the patterns connected to each classification node. The results of the experiments were tuned for either higher accuracy or less classes generated, but lower accuracy. Table 5.2 shows the results for higher accuracy and Table 5.3 shows the results for lower accuracy with less classes required. The values for the variables were: $\beta = 13, \lambda = 5, \gamma = 6, \rho = 0.17, \tau_p = 0.4, \nu = 0.16, \vartheta_p = 0.3, \vartheta_c = 0.85$ for high accuracy, and $\vartheta_c = 0.60$ for low accuracy.

Experiments With Large Image Set

For the large image set, all of the 3,471 images were used to train the LSCAN networks. The experiments were implemented in 10 stages. Each stage had an increment of 350 images starting from 350 images. For each set of images, two experiments

Table 5.3: Experiment Results for Low Accuracy

	$\vartheta_{\mu f}$					
$\vartheta_{\mu c}$	0.1	0.2	0.3	0.4	0.5	0.6 - 1.0
0 - 0.4	32/19	23/20	17/31	16/34	16/35	13/43
0.5	33/19	23/20	18/30	16/35	16/36	13/44
0.6	35/19	24/20	19/28	17/35	17/36	15/41
0.7	43/18	32/18	25/27	21/34	21/34	18/40
0.8	54/11	42/13	40/18	36/24	34/27	31/32
0.9	79/2	75/1	72/2	71/6	69/6	68/7

were undertaken. The first experiment utilized one-pass training. The second experiment used the constructed classes to test the same image set, but in a different input sequence. The results showed that during the second experiment there were no new classes created. The accuracy was also improved slightly. The results showed that LSCAN networks were trained well during the first pass. The network was not tuned for the best conditions for either high accuracy or low number of classes. The purpose of these experiments was to observe the trends of accuracy and number of created classes while increasing the size of an image set. The control parameters used in these experiments were $\beta = 15$, $\lambda = 6$, $\gamma = 7$, $\nu = 0.08$, $\rho = 0.14$, $\tau_p = 0.14$, $\tau_c = 0.85$, $\vartheta_p = 0.3$, and $\vartheta_c = 0.7$. The results are shown in Figure 5.9.

5.1.7 Conclusion

The experiments had demonstrated that LSCAN networks have capabilities to learn patterns to perform classification tasks. The LSCAN networks can be tuned to

Figure 5.9: The Experiments for Large Image Sets

perform high accuracy demanded tasks as indicated in Table 5.2 at the cost of higher number of classes. However, the experiments showed that the LSCAN networks displayed good learning capabilities. One-pass learning is very important in some working environments, if there is only one chance to learn the input patterns. In the cases of the experiments for larger image sets, the accuracies were improved slightly. These were due to the shared patterns had better representations for some particular digit images at the second pass. If this is the case, then, improving the pattern representations can improve the accuracy and reduce the number of created classes.

5.2 Rule-Guided Classification

Rule-guided classification uses rules to classify the class types of the input digit images. There are different ways to extract rules from the digit images. In this case study, a scan-path is used to develop cognitive rules. The scan-path method scans each image directly at a specific area to evaluate a specific rule. This method is simple to develop and demonstrates the ability to incorporate rules into LSCAN models.

5.2.1 Network Structure

The network structure contains two layers as shown in Figure 5.10. The rule layer contains a fixed number of rules. These rules are predefined. Their inputs are from the input image plane. Some rule extraction algorithms are used to detect the existence of rules. A rule node has an input of 1 if there is a rule extraction algorithm detects a rule. At the rule layer, there are some bias nodes which are connected to each classification node at the classification layer. These bias nodes have no inputs from the image plane and always have the values of 1. The purpose of the bias nodes is two fold. One purpose is to represent a class if there has no rule node which is activated by a rule extraction algorithm. The other purpose is to make each of the classification nodes distinguishable from each other if they have different combinations of the rule-node activations. This is required based on the results of the training of each classification node.

At the classification layer, there is no classification node in the initial state. A new classification node is created if there is no existing classification node which is activated by the input rule nodes. However, the classification node which represents any unclassified digit images is created only once.

5.2.2 Rule Extraction

The rule extraction algorithms are used to detect the features of a digit image as follows:

- change from one segment to two segments with continuity,
- change from two segments to one segment with continuity,

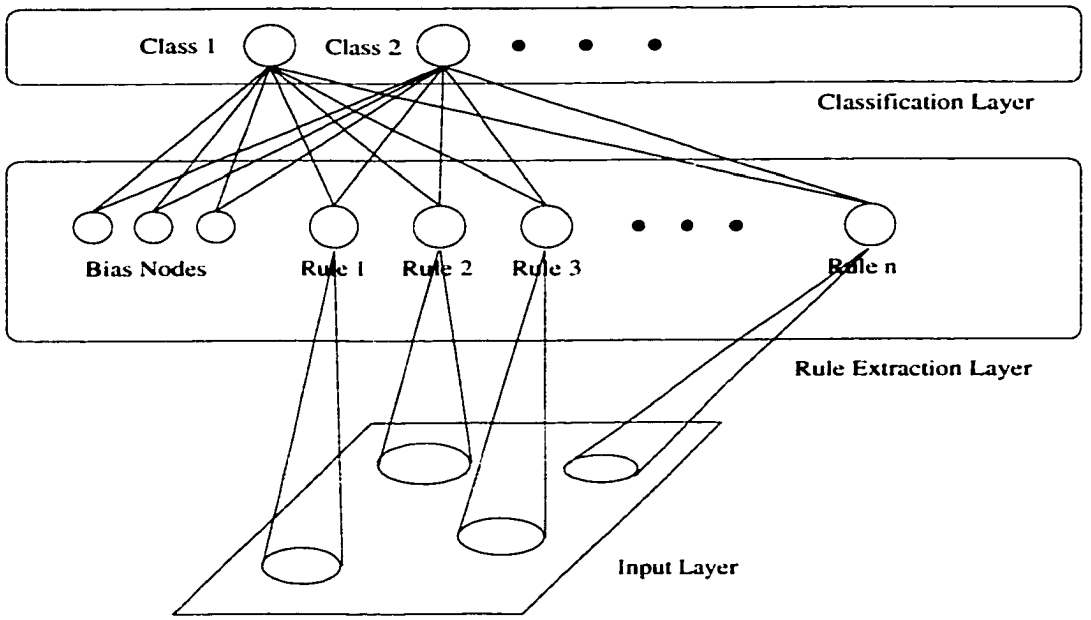


Figure 5.10: The Proposed Rule-Guided Network

- change from two segments to one segment without continuity.
- change from one segment to two segments with no connection.
- only one segment from top to bottom.
- only one segment from left to right.
- bottom half has a short stroke.
- upper right has a short stroke.
- upper left has a short stroke.
- vertical center has three segments, and
- down left side has an angle.

Based on the features of all of the digit images, the following 15 rules are used to detect the features in the input digit images:

1. Only one segment from top to bottom.
2. Only one segment from left to right.
3. Upper half changes from one segment to two segments with continuity.
4. Bottom half changes from one segment to two segments with continuity.
5. Left half changes from one segment to two segments with continuity.
6. Right half changes from one segment to two segments with continuity.
7. Upper half changes from two segments to one segment with continuity.
8. Bottom half changes from two segments to one segment with continuity.
9. Vertical center has three segments.
10. Upper half changes from two segments to one segment without continuity.
11. Upper left side has a short stroke.
12. Bottom half has a short stroke.
13. Bottom left side has an angle.
14. Upper right side has a short stroke.
15. Upper half has two vertical strokes.

Table 5.4: The Purposes of The Rules

Rule No	Good for the digits
1	1
2	1
3	0, 2, 3, 7, 8, and 9
4	0, 3, 5, 6, and 8
5	0, 4, 6, and 9
6	0, 2, 6, 7, and 9
7	8 and 9
8	6 and 8
9	2, 3, 5, 6, and 8
10	2, 3, and 7
11	6
12	1, 7, and 9
13	4
14	5
15	4

The purpose of each rule is shown in Table 5.4. From Table 5.4, it shows several digits share one rule. However, using of a set of combinational rules can identify one digit from the others. If each rule can perfectly detect the feature for its own purpose, then the 15 rules are sufficient and efficient to classify all digits. The sufficiency means no more rules are needed. The efficiency means less classes are generated with high accuracy of classifications. However, hand-printed digits do not perfectly obey these rules. Here, the qualities of ideal rules are discussed before looking into the real problems.

The outcomes of rules have strong influence on the performance of the systems. Some guidelines are required to establish the rules. Ideal rules have the following qualities:

- each unique rule represents only one unique feature.
- for each feature of a digit there is a rule to detect it.
- each rule is exclusive from the other rules, and
- no rule leakage for each rule.

In the actual case of the NIST hand-printed digit images shown in Appendix D, extracting ideal rules may be infeasible. For example, in Figure 5.11 shows some zeros with broken curves. Rules 3 and 6 fail to detect the change from one segment to two segments with continuity. All rules face unpredictable problems:

- orientations of the same feature,
- locations of the same feature.

Figure 5.11: Examples of Defective Digits

- availability of a feature,
- relations of a feature with the others,
- broken writing,
- thickness of writing,
- noise, ink infusion, and
- incomplete images.

The rules have to be tuned to find the best fit for some potential problems. However, all of the rules can not cover all of the problems. Table 5.5 shows the experiments implemented for each rule alone against the same digit. The numbers in the table represent the number of misdetected cases, or features not available, of a rule for the corresponding digit. The data show no rules which are perfectly exclusive and all rules have leakage. The data in Table 5.5 are also plotted in Figures 5.12 to 5.26. The point here is that LSCAN can learn to accommodate these deficiencies.

Table 5.5: The Rule Misdetections

Rule	0/406	1/404	2/378	3/384	4/331	5/209	6/369	7/347	8/304	9/339
1	405	5	368	376	329	186	353	311	302	327
2	406	35	377	384	313	209	369	344	304	338
3	38	404	139	93	317	140	361	107	265	289
4	36	404	334	185	214	97	158	347	280	309
5	28	397	289	380	225	202	72	343	214	162
6	58	403	320	299	279	204	287	53	201	147
7	74	404	268	378	323	203	353	347	132	129
8	49	404	336	372	327	199	294	347	49	242
9	347	401	219	15	254	63	124	314	80	70
10	403	404	94	90	194	136	369	81	301	224
11	367	4	272	272	301	65	9	291	290	323
12	397	5	336	335	80	166	299	2	255	13
13	393	382	347	353	24	175	354	156	201	17
14	394	395	374	383	300	22	206	346	269	319
15	387	400	329	315	50	106	64	304	237	282

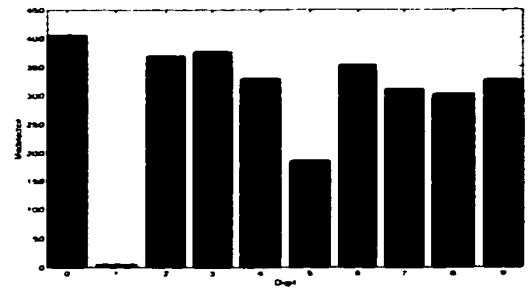


Figure 5.12: Misdetections for Rule 1

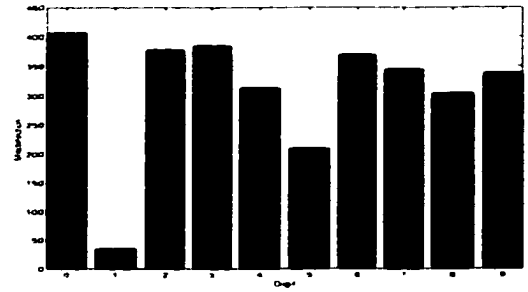


Figure 5.13: Misdetections for Rule 2

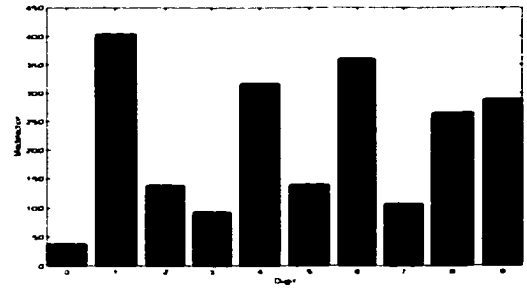


Figure 5.14: Misdetections for Rule 3

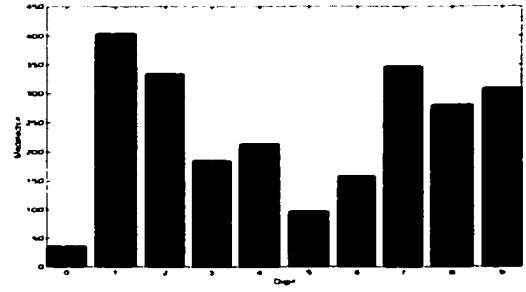


Figure 5.15: Misdetections for Rule 4

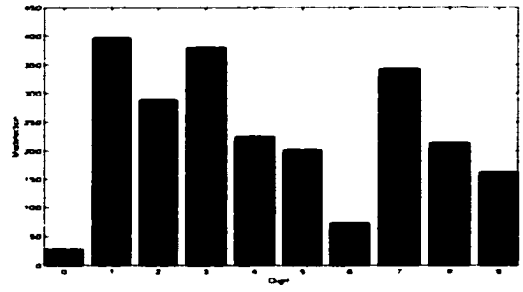


Figure 5.16: Misdetections for Rule 5

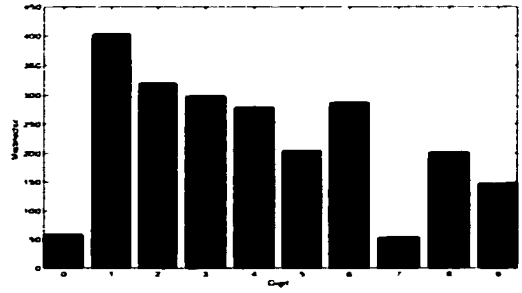


Figure 5.17: Misdetections for Rule 6

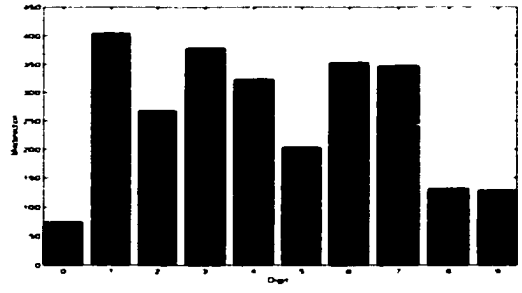


Figure 5.18: Misdetections for Rule 7

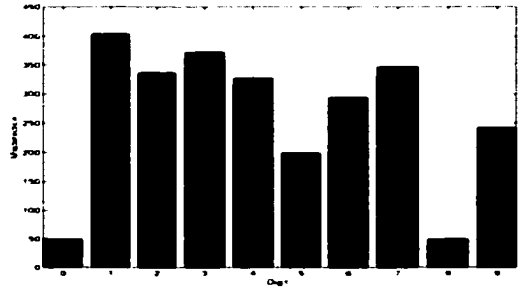


Figure 5.19: Misdetections for Rule 8

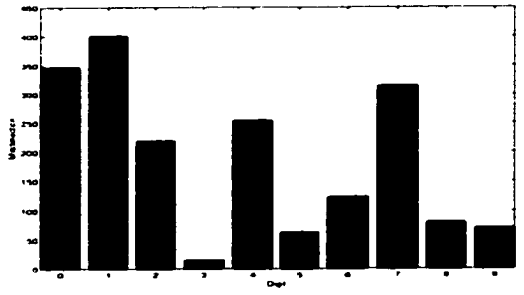


Figure 5.20: Misdetections for Rule 9

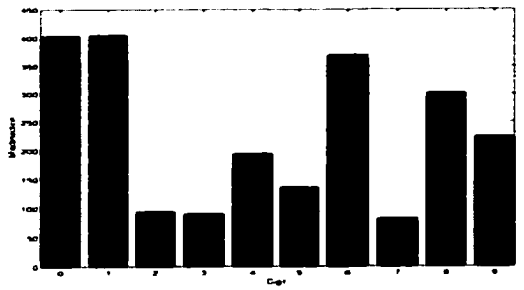


Figure 5.21: Misdetections for Rule 10

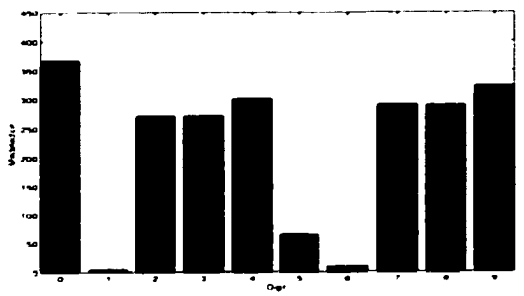


Figure 5.22: Misdetections for Rule 11

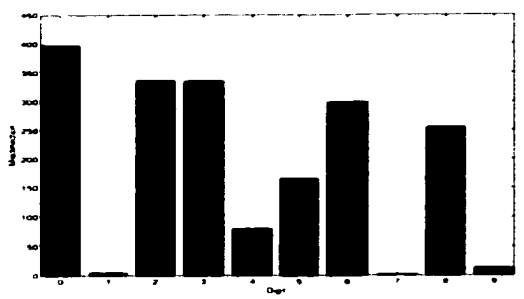


Figure 5.23: Misdetections for Rule 12

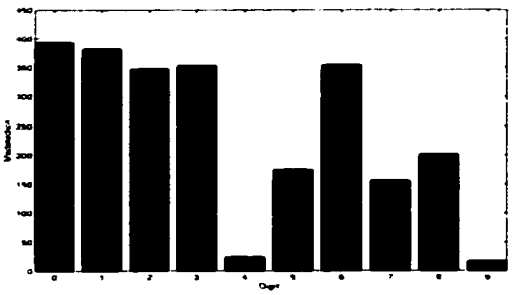


Figure 5.24: Misdetections for Rule 13

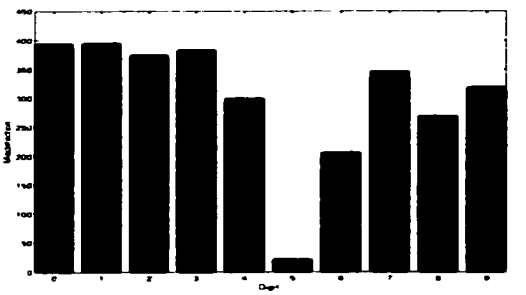


Figure 5.25: Misdetections for Rule 14

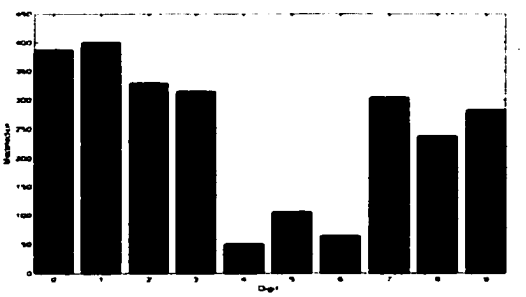


Figure 5.26: Misdetections for Rule 15

5.2.3 Class Creation

A new class is created in two cases. One case is when there are no classes in the classification layer. The other case is that there are no existing classes activated by the activation rules. A new class has connections to the all rule nodes. Initially, all of the connections have the weights of 1. The new class node is trained at the time of creation by using the Equations 3.11 and 3.12. The activation threshold is set at 0.75. After training, each classification node uses the Equation 3.2 to determine whether the activated input rule nodes are sufficient to activate the corresponding classification node.

5.2.4 Classification Experiments

Input digit images were arranged in random sequences for all of the experiments. For one size of digit images, only one random sequence was generated. Two data sets were used for small and large data set experiments. The small data set uses the 88 images as shown in Figure 5.1. The large data set uses all 3,471 digit images. The small data set was conducted for the experiments to observe how effectively the rules change the number of classes created and the number of misclassified classes while more rules were used. Table 5.6 shows the experiment results. The row of "Class" shows the number of classes created. The row of "Missed" shows the number of misclassified classes. The "%" rows show the percentage of created classes and the percentage of accuracy against the total input images. Table 5.6 also indicates which rule has a better improvement. For example, when the rule 5 added into the group of rules from 1 to 4, the accuracy jumped from 40% to 63%. At the right end of this table, when the accuracy reached to 99%, no more improvement can be done.

Table 5.6: The Effectiveness of The Rules

	Number of rules applied													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Class	2	2	4	6	10	14	17	20	26	34	38	42	43	46
%	2	2	5	7	11	16	19	23	30	39	43	48	49	52
Missed	77	68	60	53	33	32	25	23	15	6	4	1	1	1
Accuracy %	13	23	32	40	63	64	72	74	83	93	95	99	99	99

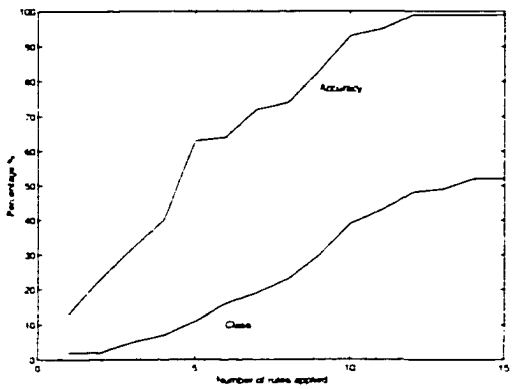


Figure 5.27: Experiments for Increasing Number of Rules

However, more rules applied the higher accuracy for the large data set. The graphical illustration is shown in Figure 5.27.

The experiments for the large data set were observed for what was the effectiveness of increasing the number of input digit images while the number of rules were fixed. The input digit images were increased for every 350 images started from 350 images. The experiments used 14, 15 rules, and 9 selective rules (1, 3, 5, 7, 9, 10, 11, 12, 14). The Tables 5.7, 5.8, and 5.9 show the experiment results. The graphical

Table 5.7: Experiment Results Using The First 14 Rules

	Number of input images									
	350	700	1050	1400	1750	2100	2450	2800	3150	3471
Class	157	249	318	369	416	439	476	506	536	554
%	45	36	30	26	24	21	19	18	17	16
Missed	35	74	120	170	249	370	370	386	468	470
Accuracy %	90	89	89	88	86	82	85	86	85	86

illustrations are shown in Figure 5.28, Figure 5.29, and Figure 5.30. These three experiments show that the number of created classes decreased in percentage. From Table 5.9, the number of classes stopped to increase while the number of input digit images reached 2,450. From the trends of the three experiments, for a fixed number of rules, the number of classes created might be saturated while the number of input digit images continues to increase. The accuracy in percentage shows a small variation in the three experiments. It appears to be stable while input images increased into larger numbers.

5.2.5 Conclusion

From the implemented experiments, rule-guided classification showed an efficient way to perform this kind of recognition task. The performance of the classification system depends on the rule-extraction algorithms which in turn rely on the qualities of the input digit images. LSCAN can learn to provide very good performance if the rule-extraction algorithms are even just reasonably accurate.

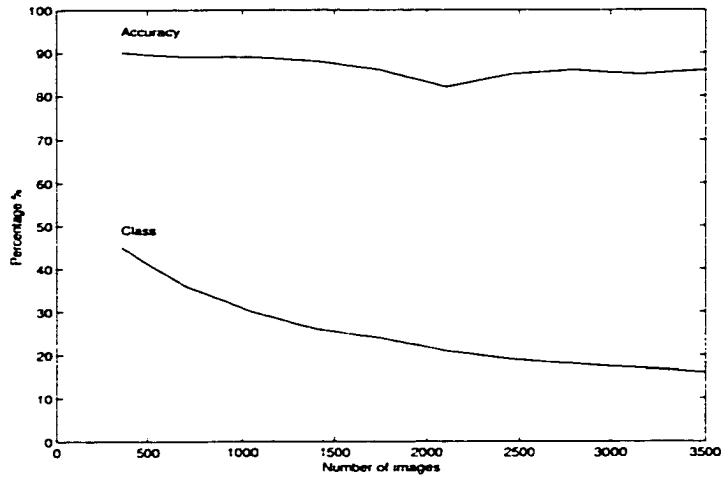


Figure 5.28: Experiments Using The First 14 Rules

Table 5.8: Experiment Results Using All 15 Rules

	Number of input images									
	350	700	1050	1400	1750	2100	2450	2800	3150	3471
Class	168	264	349	406	459	495	541	581	611	638
%	48	38	33	29	26	24	22	21	19	18
Missed	30	68	105	154	223	338	344	357	417	420
Accuracy %	91	90	90	89	87	84	86	87	87	88

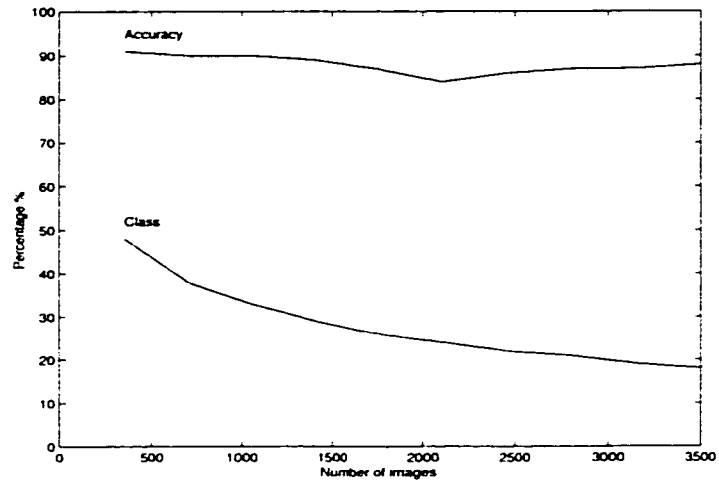


Figure 5.29: Experiments Using All 15 Rules

Table 5.9: Experiment Results Using The 9 Selective Rules

	Number of input images									
	350	700	1050	1400	1750	2100	2450	2800	3150	3471
Class	65	101	108	122	125	129	139	144	144	144
%	19	14	10	9	7	6	6	5	5	4
Missed	97	189	251	383	526	620	838	707	822	1003
Accuracy %	72	73	76	73	70	70	66	75	74	71

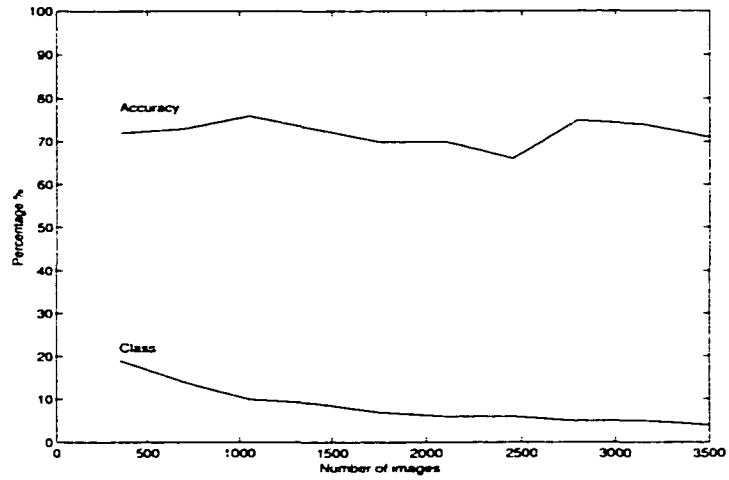


Figure 5.30: Experiments Using The 9 Selective Rules

5.3 Compare Pattern-Based to Rule-Guided Classification

From the viewpoint of network architecture, both pattern-based and rule-guided classification networks are similar to each other at the top layer. From the viewpoint of performance, both have advantages and disadvantages. Pattern-based classification can use the control parameters to tune the network to have high or low classification accuracy. The number of created classes varies accordingly. Rule-guided classification networks have no such control parameters. Once the rule extraction algorithms have completed, there is not much room for tuning. Nonethelss, different combinations of rules can change the classification accuracies and the numbers of created classes. Pattern-based classification systems have one more advantage over rule-guided classification systems. Pattern-based classification systems are free from image noise. Image noise are ignored at the process of pattern formation. On the other hand, image noise can make rule algorithms perform more poorly.

As the count of number of connections is concerned, rule-guided classification networks use less connections. Each classification node has connections of the number of rules plus the number of bias nodes. Each classification node has a fixed number of connections. On the other hand, each pattern-based classification node has a variable connections to a number of PDMs, which depending on the number of patterns extracted from the feature planes. Furthermore, each PDM has $K(2\lambda + 1)$ connections to the feature planes, where K is the number of feature planes. The larger size of λ, refer to the Figure 2.5, the larger number connections is required.

At the feature extraction layer, pattern-based classification networks require more computational power than rule-guided classification networks. Each rule needs to scan one time on a particular area on each input image. Let the size of the input image be MN pixels. For each rule, it needs $M'N'$ computational operations, where $M' \leq M$ and $N' \leq N$. The total number of computational operations is $RM'N'$, where R is the number of rules. For feature extractions, each location on a feature plane has 3^2 computational operations. Then the total number of computational operations is $KMN3^2$.

5.4 Summary

In this Chapter, pattern-based and rule-guided classification methods were applied to classify hand-printed digit images. Both approaches showed the the flexibilities of LSCAN networks to perform one task in different ways. The LSCAN networks were configured here to meet the needs of large-scale learning environments.

P_ARADISE [3] is one of the pattern-based classification networks. It was first used in Banarse's dissertation for digit image classifications. LSCAN can be used for a similar approach to implement the same task. The architecture of P_ARADISE is compatible with LSCAN E-nodes. Each classification node and PDM are trained individually. This kind of approach resembles localist networks. There are no hidden layers need to be trained. Therefore, LSCAN is a good candidate to implement the task. In alternative, LSCAN networks can implement rule-guided classifications. These two approaches demonstrate the flexibilities of LSCAN networks which can be configured to meet different requirements.

CHAPTER 6

APPLICATION TO ROUTING NETWORKS

The purpose of this case study is to demonstrate the capabilities of LSCAN to build LSCAN networks to replace the regular routing tables using Border Gateway Protocol version 4 (BGP4). The knowledge of routing information develops while new routes are detected. The LSCAN routing networks not only provide the best path for each IP network destination, they also provide the second best path for the same IP network destination. The second best path is not part of BGP4 standard. It is provided for the illustrative purposes in this case study. Nonetheless, the LSCAN networks built in this case study do not cover all aspects of BGP4.

6.1 Introduction

The Border Gateway Protocol (BGP), defined in RFC 1771, allows network administrators to create loop-free inter-domain routing between autonomous systems. An autonomous system (AS) is a set of routers under a single technical administration. Routers in an AS can use multiple interior gateway protocols to exchange routing information inside the AS and an exterior gateway protocol to route packets outside the AS. BGP version 4 is a classless routing protocol. Each BGP router uses aggregated IP addresses to route network messages. An IP address can be aggregated into an any bit length between 1 and 32 to specify a network IP address.

BGP uses TCP as its transport protocol (port 179). Two BGP speaking routers form a TCP connection between one another (peer routers) and exchange messages to open and confirm the connection parameters. BGP routers will exchange network reachability information, this information is mainly an indication of the full paths (BGP AS numbers) that a route should take in order to reach the destination network. This information will help in constructing a graph of ASs that are loop-free and where routing policies can be applied in order to enforce some restrictions on the routing behavior.

Any two routers that have formed a TCP connection in order to exchange BGP routing information are called peers, they are also called neighbors. In this text, peer and router may be used as synonyms. BGP peers will initially exchange their full BGP routing tables. From then on incremental updates are sent as the routing table changes. BGP keeps a version number of the BGP table and it should be the same for all of its BGP peers. The version number will change whenever BGP updates the table due to some routing information changes. Keep-alive packets are sent to ensure that the connection is alive between the BGP peers and notification packets are sent in response to errors or special conditions.

If one AS has multiple BGP speakers, it could be used as a transit service for other ASs. It is necessary to ensure reachability for networks within an AS before sending the information to other external ASs. This is done by a combination of Internal BGP (IBGP) peering between routers inside an AS and by redistributing BGP information to Internal Gateway Protocols (IGP) running in the AS. All BGP speakers running between any two different ASs use Exterior BGP (EBGP).

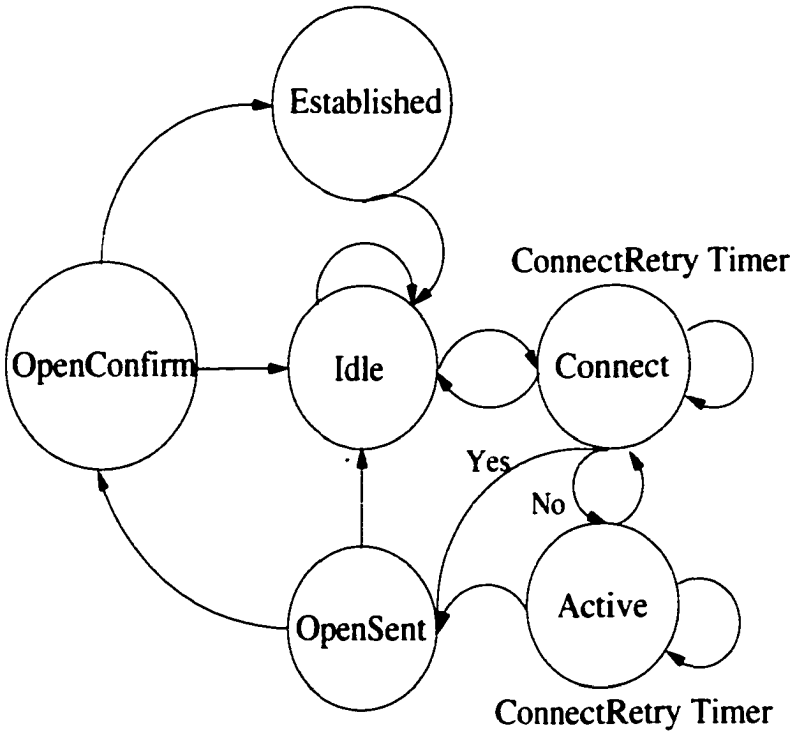


Figure 6.1: The BGP Finite State Machine

6.1.1 BGP Finite State Machine

This section briefs BGP operation in terms of a Finite State Machine (FSM). One BGP peer in order to make a connection with its neighbor peer goes through different states. Initially, a BGP is in the Idle state. Once the connection between two peers is completed, the BGP is in the Established state. Figure 6.1 illustrates relations of all states. A brief description for each state is given as:

Idle state: In this state, BGP refuses all incoming BGP connections. No resources are allocated to the peer. In response to the Start event (initiated by either system or operator) the local system initializes all BGP resources, starts the Connect-Retry timer, initiates a transport connection to other BGP peer, while

listening for connection that may be initiated by the remote BGP peer, and changes its state to Connect. The exact value of the Connect-Retry timer is a local matter, but should be sufficiently large to allow TCP initialization.

Connect state: In this state, BGP is waiting for the transport protocol connection to be completed. If the transport protocol connection succeeds, the local system clears the Connect-Retry timer, completes initialization, sends an OPEN message to its peer, and changes its state to OpenSent.

If the transport protocol connect fails (e.g., retransmission timeout), the local system restarts the Connect-Retry timer, continues to listen for a connection that may be initiated by the remote BGP peer, and changes its state to Active state.

Active state: In this state, BGP is trying to acquire a peer by initiating a transport protocol connection. If the transport protocol connection succeeds, the local system clears the Connect-Retry timer, completes initialization, sends an OPEN message to its peer, sets its Hold Timer to a large value, and changes its state to OpenSent. A Hold Timer value of 4 minutes is suggested.

In response to the Connect-Retry timer expired event, the local system restarts the Connect-Retry timer, initiates a transport connection to other BGP peer, continues to listen for a connection that may be initiated by the remote BGP peer, and changes its state to Connect.

If the local system detects that a remote peer is trying to establish BGP connection to it, and the IP address of the remote peer is not an expected one, the local system restarts the Connect-Retry timer, rejects the attempted connec-

tion, continues to listen for a connection that may be initiated by the remote BGP peer, and stays in the Active state.

OpenSent state: In this state, BGP waits for an OPEN message from its peer. When an OPEN message is received, all fields are checked for correctness. If the BGP message header checking or OPEN message checking detects an error, or a connection collision the local system sends a NOTIFICATION message and changes its state to Idle.

If there are no errors in the OPEN message, BGP sends a KEEPALIVE message and sets a Keep-Alive timer. The Hold Timer, now is replaced with the negotiated Hold Time value. If the negotiated Hold Time value is zero, then the Hold Time timer and Keep-Alive timers are not started. If the value of the Autonomous System field is the same as the local Autonomous System number, then the connection is an "internal" connection; otherwise, it is "external". This effects UPDATE processing as described below. Finally, the state is changed to OpenConfirm.

If a disconnect notification is received from the underlying transport protocol, the local system closes the BGP connection, restarts the Connect-Retry timer, while continue listening for connection that may be initiated by the remote BGP peer, and goes into the Active state.

If the Hold Timer expires, the local system sends NOTIFICATION message with error code Hold Timer Expired and changes its state to Idle.

OpenConfirm state: In this state, BGP waits for a KEEPALIVE or NOTIFICATION message. If the local system receives a KEEPALIVE message, it changes

its state to Established.

If the Hold Timer expires before a KEEPALIVE message is received, the local system sends NOTIFICATION message with error code Hold Timer Expired and changes its state to Idle.

If the local system receives a NOTIFICATION message, it changes its state to Idle. If a disconnect notification is received from the underlying transport protocol, the local system changes its state to Idle.

Established state: In the Established state, BGP can exchange UPDATE, NOTIFICATION, and KEEPALIVE messages with its peer. If the local system receives an UPDATE or KEEPALIVE message, it restarts its Hold Timer, if the negotiated Hold Time value is non-zero.

If the local system receives a NOTIFICATION message, it changes its state to Idle.

If the local system receives an UPDATE message and the UPDATE message error handling procedure detects an error, the local system sends a NOTIFICATION message and changes its state to Idle.

If a disconnect notification is received from the underlying transport protocol, the local system changes its state to Idle.

If the Hold Timer expires, the local system sends a NOTIFICATION message with Error Code Hold Timer Expired and changes its state to Idle.

6.1.2 BGP Decision Making

While all BGP peers are in the Established state, each peer receives new route or updated route information using the update messages from its neighbor peers. Each new route or updated route is examined against the existing routes which have the same network ID and initial router. The best route is selected and advertised to its neighbor peers. One route which is advertised by a peer to its neighbor peers might be returned to the same peer. In this case, the original advertising peer detects its own AS ID in the AS Path parameter in order to avoid route loops. BGP uses several parameters to make the decision of which route is the best route according to the BGP configurations. The parameters which are used for the best route choice are briefly described as:

Weight: This is a Cisco defined attribute. The weight is assigned locally to the router. It is not propagated or carried through any of the route updates. A weight can be a number from 0 to 65535. Paths that the router originates have a weight of 32768 by default and other paths have a weight of zero.

Local Preference: Local preference is an indication to the AS about which path is preferred to exit the AS in order to reach a certain network. A path with a higher local preference is more preferred. The default value for local preference is 100. Unlike the weight attribute which is only relevant to the local router, local preference is an attribute that is exchanged among routers in the same AS.

AS path: Whenever a route update passes through an AS, the AS number is preappended to that update message. The AS-Path attribute is actually the list of

AS numbers that a route has traversed in order to reach a destination. An AS-SET is an ordered mathematical set of all the AS that have been traversed.

Origin code: The origin is a mandatory attribute that defines the origin of the path information. The origin attribute can assume three values:

IGP: Network Layer Reachability Information (NLRI) is interior to the originating AS. This normally happens when we use the BGP network command or when IGP is redistributed into BGP, then the origin of the path information will be IGP. This is indicated with an "i" in the BGP table.

EGP: NLRI is learned via EGP (Exterior Gateway Protocol). This is indicated with an "e" in the BGP table.

INCOMPLETE: NLRI is unknown or learned via some other means. This usually occurs when we redistribute a static route into BGP and the origin of the route will be incomplete. This is indicated with an "?" in the BGP table.

MED: Multi_exit_discriminator (MED for BGP4), or Metric attribute, is a hint to external neighbors about the preferred path into an AS. This is a dynamic way to influence another AS on which way to choose in order to reach a certain route given that we have multiple entry points into that AS. A lower value of a metric is more preferred.

Unlike local preference, metric is exchanged between ASs. A metric is carried into an AS but does not leave the AS. When an update enters the AS with a certain metric, that metric is used for decision making inside the AS. When the same update is passed on to a third AS, that metric will be set back to 0. The

Metric default value is 0.

The decision making sequence is given as:

1. If NextHop is inaccessible do not consider it.
2. Prefer the largest Weight.
3. If same weight prefer largest Local Preference.
4. If same Local Preference prefer the route that the specified router has originated.
5. If no route was originated prefer the shorter AS path.
6. If all paths are external prefer the lowest origin code (IGP¡EGP¡INCOMPLETE).
7. If origin codes are the same prefer the path with the lowest MED.
8. If path is the same length prefer the External path over Internal.
9. If IGP synchronization is disabled and only internal path remain prefer the path through the closest IGP neighbor.
10. Prefer the route with the lowest ip address value for BGP router ID.

Figure 6.2 illustrates the sequence of how a best route is selected. The last decision making picks the path coming from the BGP peer which has the lowest IP address no matter what conditions are.

6.2 Network Topology

The hypothetic back-bone routing network contains six Autonomous Systems as shown in Figure 6.3. The six AS are numbered as: AS10, AS20, AS30, AS40,

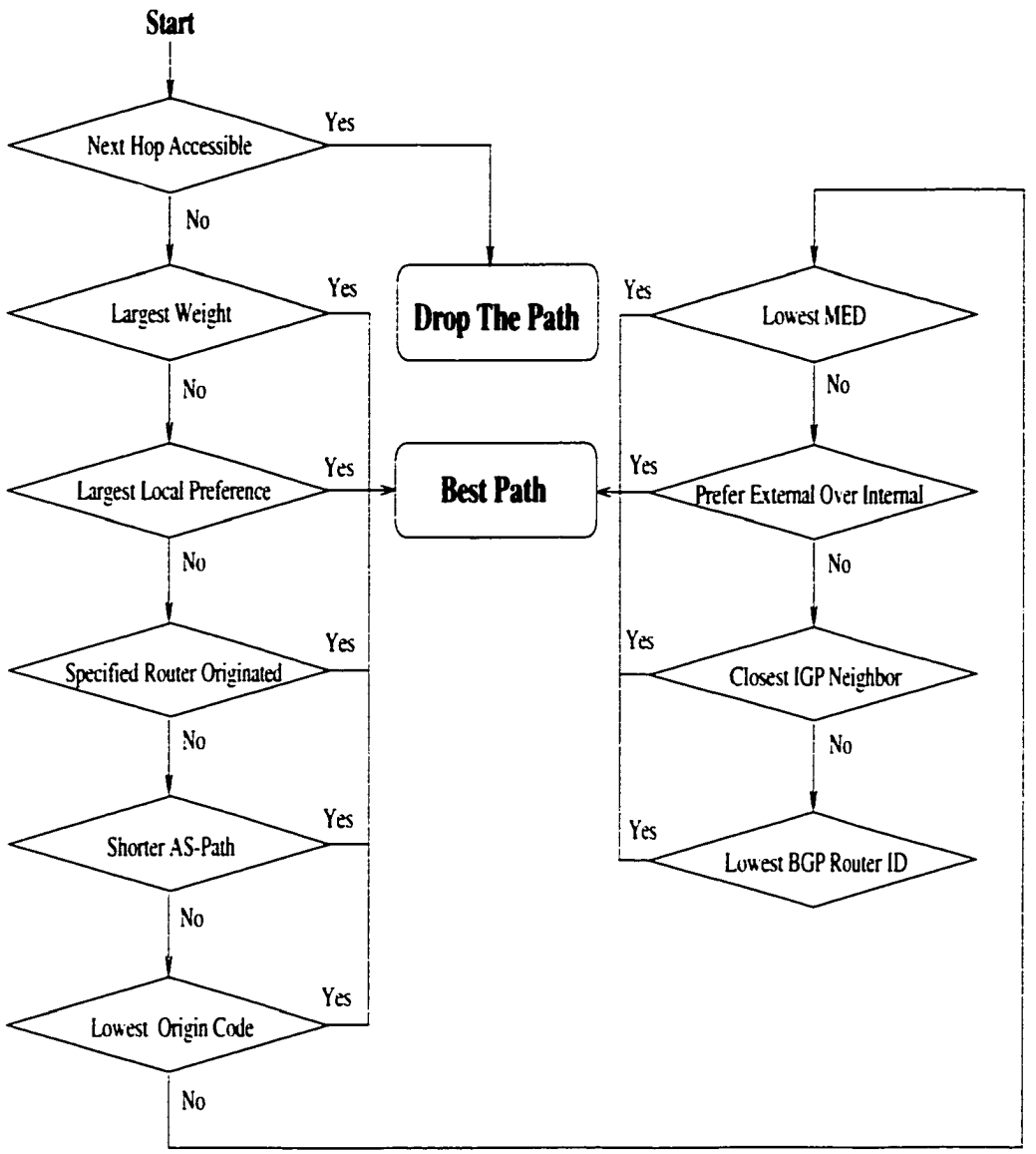


Figure 6.2: The BGP Decision Making Flow Chart

AS50, and AS60. AS10 has three routers: RTA, RTB, and RTF. RTF is an internal router which does not speak to any router outside of AS10. RTF is not considered in this case study. The other autonomous systems, AS20 through AS60, contain one BGP router in each AS. EBGP is used between the BGP peers.

Within each autonomous system, each internet address is prefixed with a letter of *E*, *L*, or *S*. The letter *E* represents the interface between the two routers using Ethernet. The letter *L* represents the interface between the two routers using LoopBack connection. LoopBack interface is a software-only interface which emulates an interface that is always up. The letter *S* represents the interface between the two routers using Serial connections. The first one-digit number following the letter is the interface slot number on the router. The second one-digit number following the first one-digit number is the interface port number on the router. For example, S2/1 means serial interface with slot number 2 and port number 1.

6.3 BGP Configurations

BGP routing policy is controlled by the configuration of each router. Through the configuration, each router makes the decision of selecting the best path for each destination. The configuration controls many aspects of how each route flows among BGP peers through BGP commands and attributes. The complete configurations of each router in Figure 6.3 are given in Appendix A. Here, some of the BGP commands and attributes are given in brief explanation.

ip address *ip-address mask*: This is an interface command that configures an interface with an IP address/mask tuple.

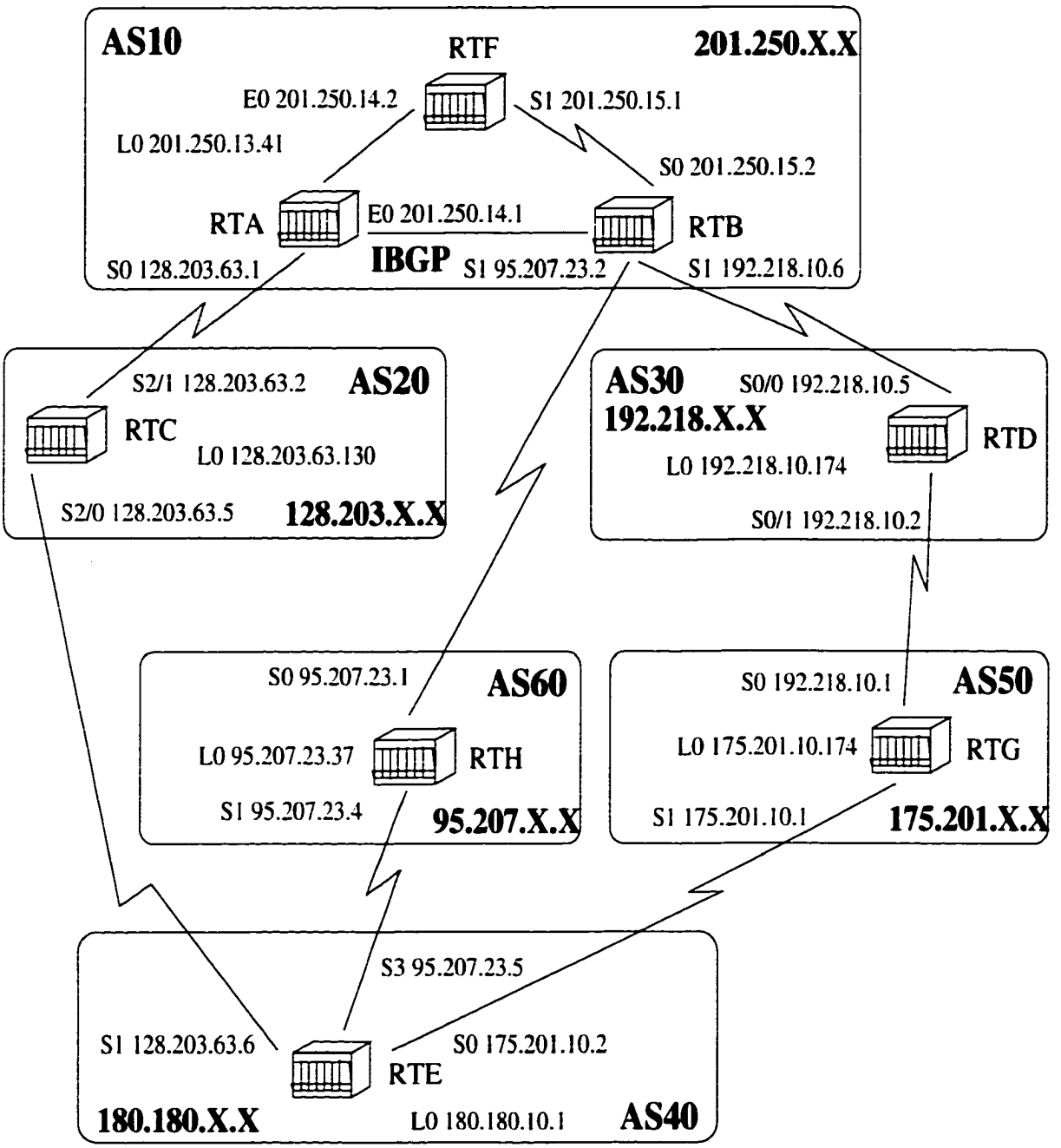


Figure 6.3: Case Study Back-Bone Network Topology

ip classless: This command enables the router to forward packets that are destined for unrecognized subnets of directly connected networks.

ip subnet-zero: This global configuration command is necessary in case the interfaces of the subnet-zero subnets are configured.

neighbor: This command is used to define the BGP neighbor connection parameters and policies between this router and its peers.

network *network-number network-mask*: The network command controls what networks are originated by this AS. This command is not trying to run BGP on a certain interface, rather it is trying to indicate to BGP what networks it should originate from this AS. The mask portion is used because BGP4 can handle subnetting and supernetting. A maximum of 200 entries of the network command are accepted.

remote-as: This command, when used with the BGP neighbor command, specifies the AS number of the remote BGP peer.

route-map: Route map is a method used to control and modify routing information. This is done by defining conditions for redistributing routes from one routing protocol to another or controlling routing information when injected in and out of BGP.

synchronization: If an autonomous system is passing traffic from another AS to a third AS, BGP should not advertise a route before all routers in the transit AS have learned about the route via IGP. BGP will wait until IGP has propagated

the route within the AS and then will advertise it to external peers. This is called synchronization.

update-source *interface*: This command, when used with the BGP neighbor command, specifies the interface to be used as a source IP address of the BGP session with the neighbor.

6.4 BGP Path Flow

During a BGP session, each active BGP router receives update messages. For any IP network within an AS, a BGP router originates the advertisement of a specific IP network to its peers. Then this IP network with its attribute and path information are examined at each receiving peer. This incoming IP network is advertised to the neighbors of the receiving peer in two cases:

1. The incoming IP network is new to the receiving peer.

2. The path of the IP network is the best path compared to the other paths of the same IP network.

In the first case, a new IP network to the receiving peer is accepted and advertised automatically. This is because there has no other paths to be compared. The complete path flow information for each router is given in Appendix B. Each path carries the information of Origin, Network, Next Hop, Local Preference, Weight, Metric, and AS-Path. Under the Origin column, two information are shown. On the left side, it shows the router names. On the right side, some special characters are used to mark the status of each path. Those special characters are explained as:

">" This character marks the path which is the best path and is advertised to the neighbor peers.

"@" This character marks the path which is the second best path and is added into the routing table. This option is for this case study only and it is not part of BGP. The second best path in not advertised by the router to its neighbor peers.

"!" This character marks the path which is dropped out of advertisement.

"*" This character marks the path which is a valid path.

"^" This character marks the path as a default path.

The network column contains the network ID or IP addresses where BGP routers originate them. Each network ID contains one IP address and a number of mask bits. For example, 128.203.0.0/16 has 16 mask bits or the mask is 255.255.0.0. The mask is used to routing all packets with more specific IP addresses to the network 128.203.0.0.

The next hop column contains the neighbor peers from which the paths flow into the receiving router. The next hop is also used to routing a data packet to the destination. However, for clarity purpose, the router RTB (refer to the AS10 in Figure 6.3) is the next hop of the router RTA and the same for the router RTA. Since RTA and RTB both are in the same autonomous system, they communicate with each other using IBGP. In BGP, any next hop is outside of the AS. For example, the router RTB learns external paths from the router RTC through the router RTA. The next hop of RTB is RTC, not RTA.

The AS-Path column shows how many AS have been passed through for each path. When an IP network is advertised by a BGP router, the router preappends its own AS ID in the AS-Path attribute. If a path flows back to any router which had advertised the path, the router rejects this path. Therefore, it prevents a loop to be happened. The letter "i" denotes that the path is learned through IBGP.

6.5 Construct LSCAN Networks

The main goal of constructing LSCAN networks is to establish the feasibility and performance for a learning network in this environment. The final routing networks for the router RTA is shown in Figure 6.4. The other routing networks are provided in Appendix C.

In this case study, the router RTA is taken as the example to build all related networks. At routing time, each destination IP address is required to compare to the all IP network addresses which were originated by the BGP routers. Taking an example in Figure 6.5, the destination IP address Des_NetID may contain a more specific IP address which is a subnetwork of the network which RTA originated. Then the network mask Mask_RTA1 has to be applied to Des_NetID in order to generate a super-network IP address Des_MaskNetID. Then, the Des_MaskNetID is compared to the originated network Net_201.250.13.0 which has a value of 201.250.13.0. If Net_201.250.13.0 and Des_MaskNetID are matched, the E-node MatchNet_RTA1 is fired. If Net_201.250.14.0 and Des_MaskNetID are matched, the E-node MatchNet_RTA2 is fired. Both MatchNet_RTA1 and MatchNet_RTA2 are ORed onto the

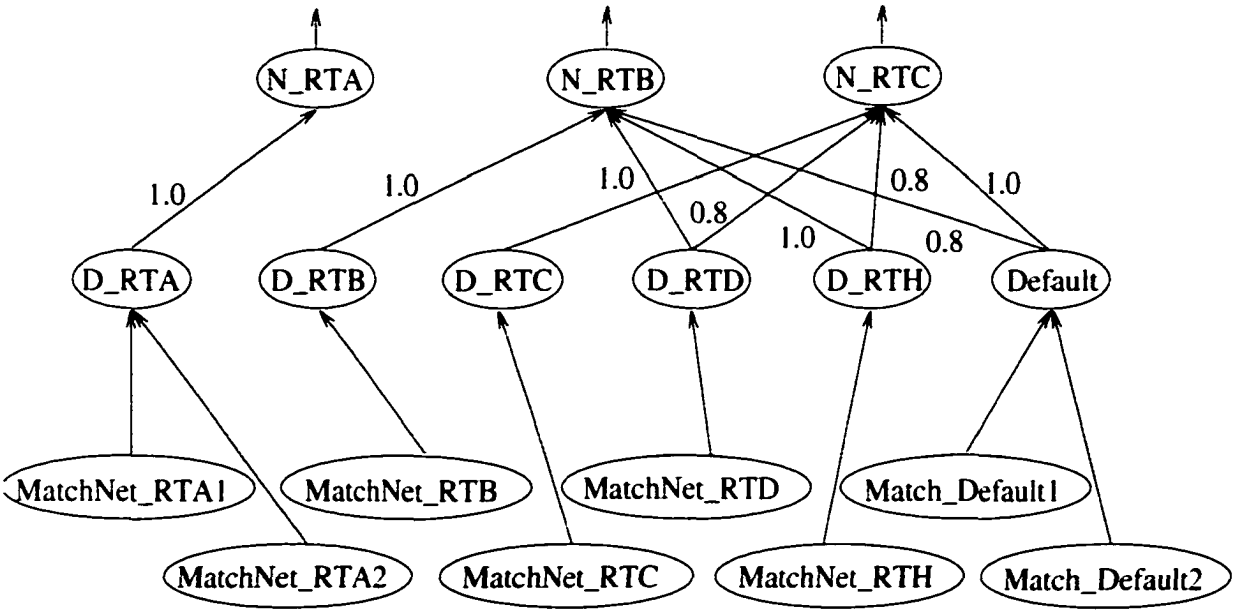


Figure 6.4: The RTA Routing Networks

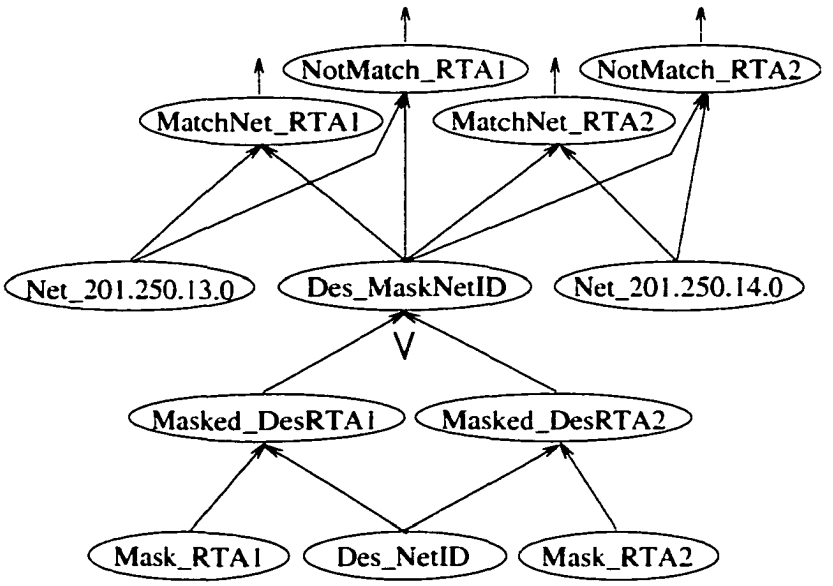


Figure 6.5: The RTA Destination Matching Networks

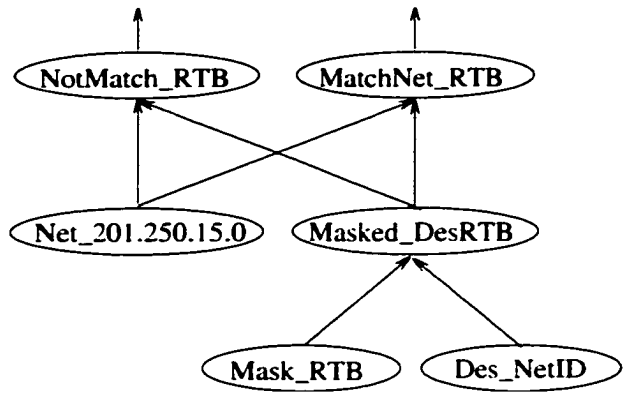


Figure 6.6: The RTB Destination Matching Networks

E-node D_RTA which leads the routing to the router RTA as shown in Figure 6.4.

The destination matching networks for the router RTB is shown in Figure 6.6. The destination matching networks for the routers RTC, RTD, RTE, RTG, and RTH are given in Appendix C.

6.5.1 Initialize LSCAN Networks

In order to build the routing and the destination matching networks for each router, some of the LSCAN networks are constructed at the beginning based on the knowledge given in the configurations. The first network to be built is the All_NextHops network as shown in Figure 6.7. This network contains generic E-nodes for the neighbor peers. If the E-node All_NextHops is activated, it triggers the E-nodes from NextHop1 to NextHopN. The values of NextHopX are the names of the routers. For example, RTA has two neighbors, RTB and RTC. Next, the value of NextHop1 is RTB and the value of NextHop2 is RTC. Each E-node of NextHopX activates its own attribute E-nodes: NH_IDX, NH_LPrefX, and NH_MEDX which represent network ID, local preference, and MED respectively as shown in Figure 6.8.

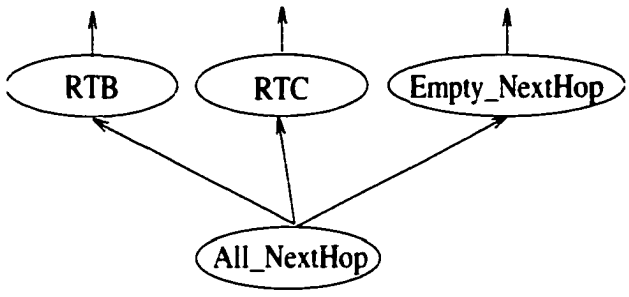


Figure 6.7: The All Next Hop Network

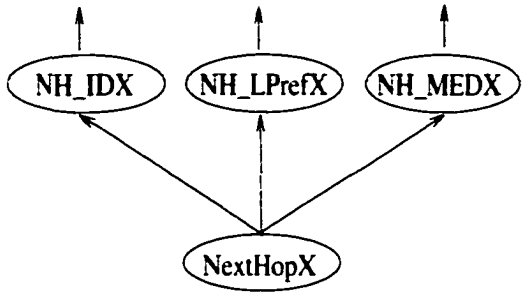


Figure 6.8: The Next Hop X Network

The E-node Empty_NextHop in Figure 6.7 is a functional E-node to detect if there is no valid next hop which is connected to this host router. The E-node All_NextHops carries a count for how many next hops are connected.

Refering the Section 6.1.2, the first condition of the decision making is the accessibility of the next hop. The Find Next Hop Network is required to make up the decision of whether an incoming path should be dropped out of consideration. Figure 6.9 illustrates the network.

The next step is to build an All_Routes network as shown in Figure 6.10. Initially, this network has only two E-nodes, the All_Routes E-node and the Empty_Route E-Node, only. Empty_Route is active only if the value of All_Routes is 0. This network grows as more new routes are received. Each valid new route is added into this

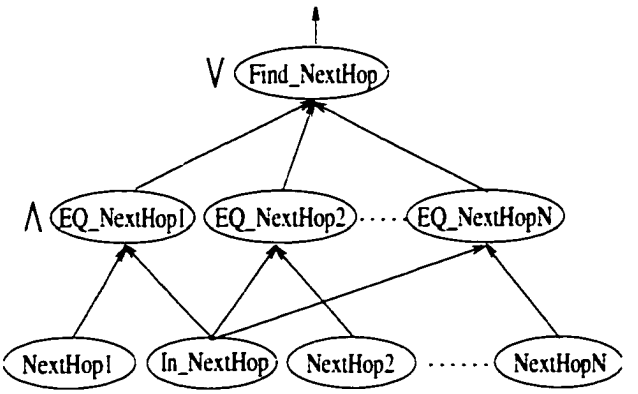


Figure 6.9: The Find Next Hop Network

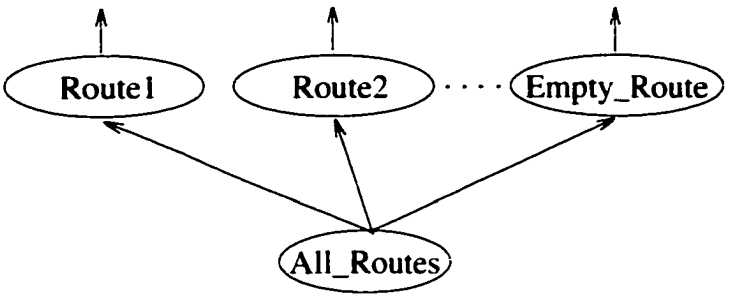


Figure 6.10: The All Route Network

network as illustrated in Figure 6.10, the route is added as Route1, Route2, ...etc. Next, each route is constructed as in Figure 6.11. In order to construct a new RouteX network, the incoming route has to be compared to all existing routes. The two routes are same if they have the same origin, come from the same next hop, and have the same AS paths as shown in Figure 6.12. If there is no existing route matches the incoming route, then a new route network is constructed by the construct route network as shown in Figure 6.13. The E-node Construct(Route) is a system E-node which builds the RouteX network, Find Route network, and All_Routes network based on the script of Route. The Route script carries all required information to

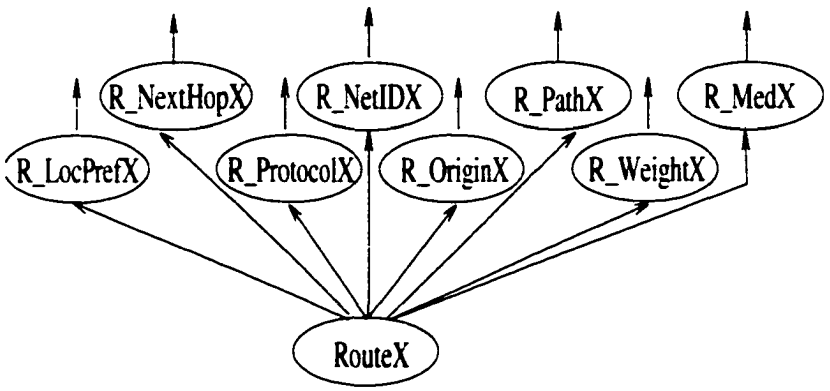


Figure 6.11: The Route X Network

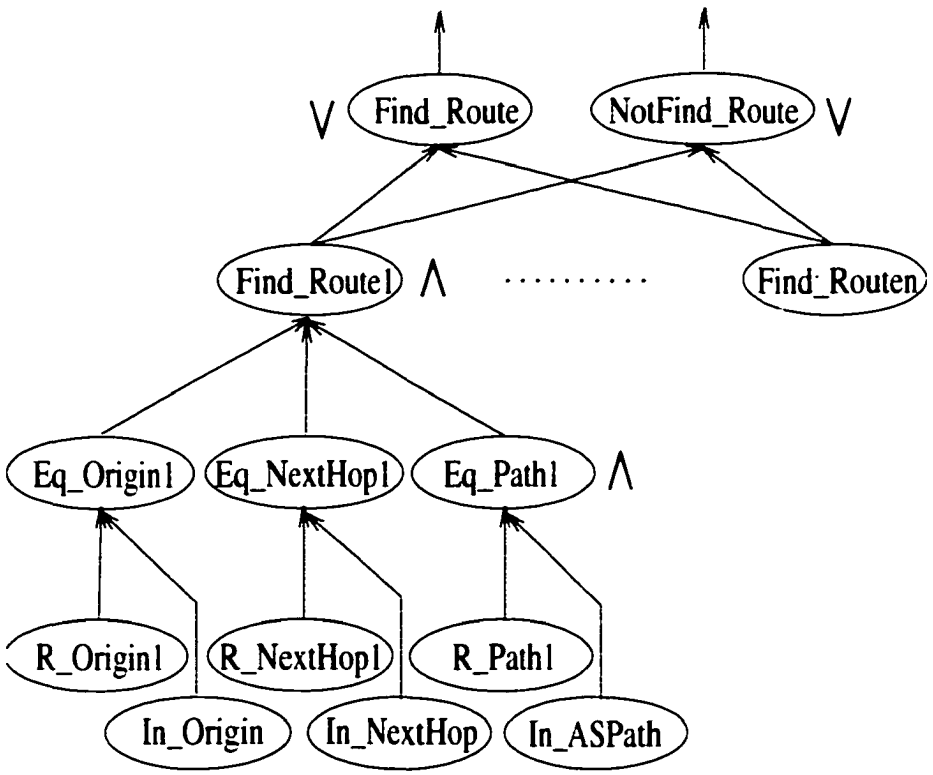


Figure 6.12: The Find Route Network

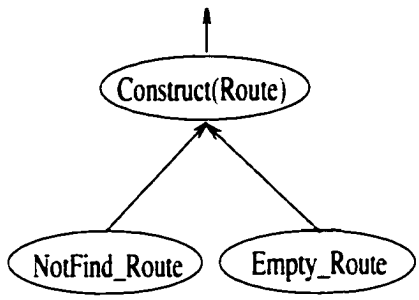


Figure 6.13: The Construct Route Network

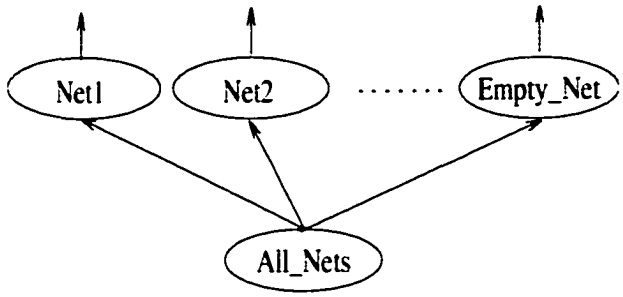


Figure 6.14: The All Net Network

build the mentioned networks.

For each route or path, if there is one network ID which is discovered as a new network ID to the host router, then this new network ID is added into the corresponding destination matching network. The all-nets network, find-net network, and construct-net network are for this purpose as shown in Figures 6.14, 6.15, and 6.16.

The network builds the construct-default network. If the incoming network ID is equal to the default network ID and the default network is not defined yet, then the E-node Construct(Default) is activated as shown in Figure 6.17. The activities of Construct(Default) generate the default network and the default matching network as shown in Figures 6.18 and 6.19. The script Default gives all information to build

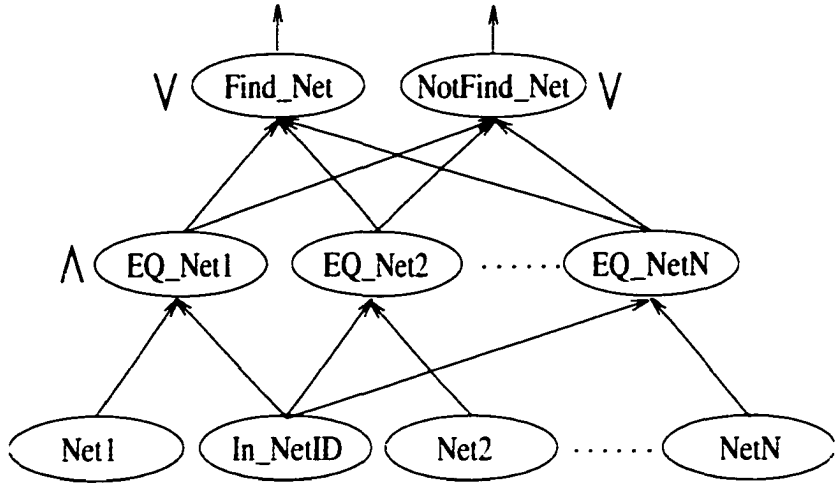


Figure 6.15: The Find Net Network

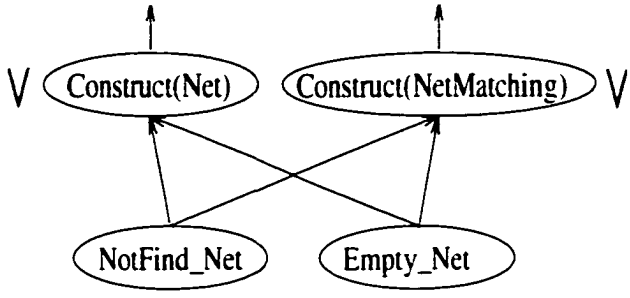


Figure 6.16: The Construct Net Network

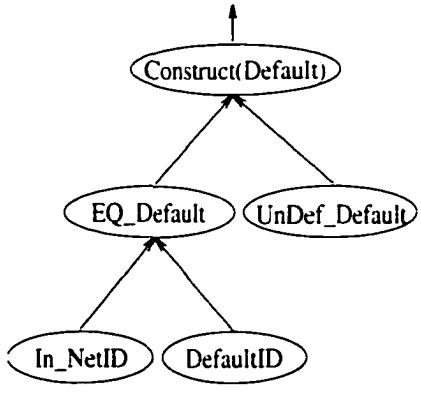


Figure 6.17: The Construct Default Network

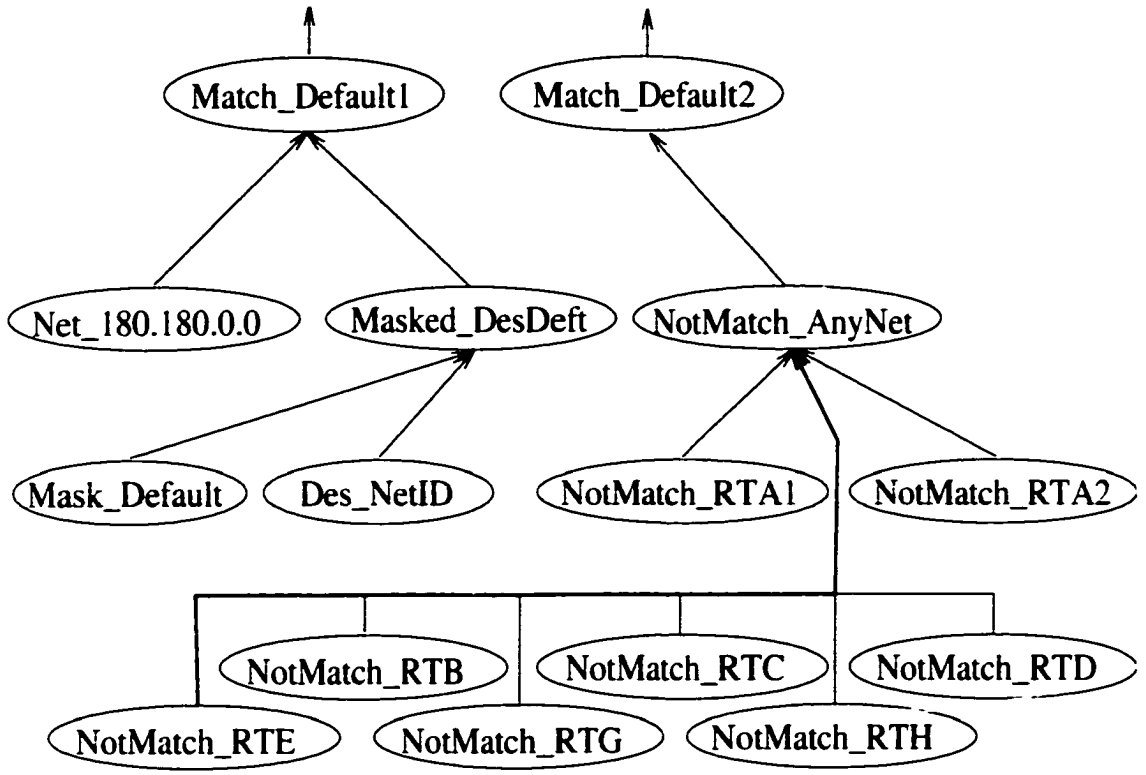


Figure 6.18: The Default Matching Network

these two networks.

One of the feature of BGP4 is to prevent loops occurring while a router learns a path and readvertises the path to its peers. This feature is done by the find-my-AS network shown in Figure 6.20. The attribute In_ASPath is matched against My_AS which is the AS number where the router belongs to. The E-node My_AS_InPath evaluates the matching. If My_AS_InPath is fired, it prevents the incoming route to be considered.

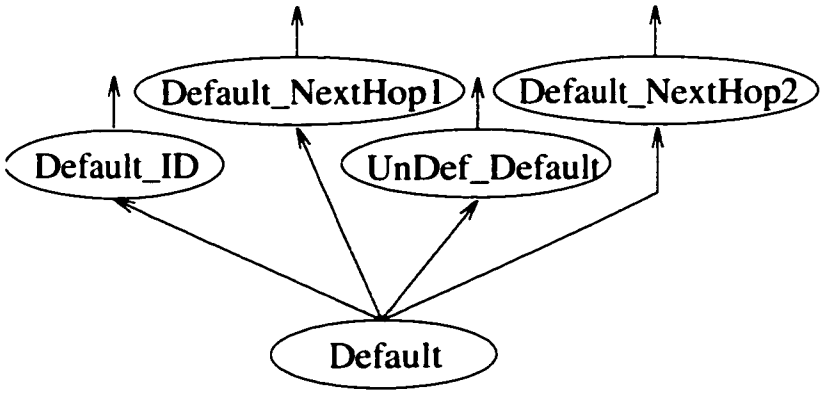


Figure 6.19: The Default Network

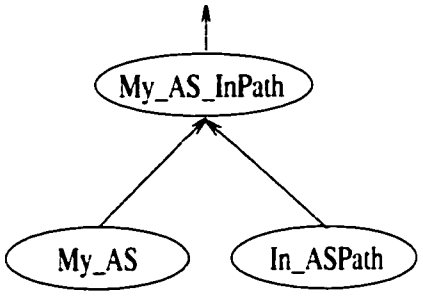


Figure 6.20: The Find My AS Network

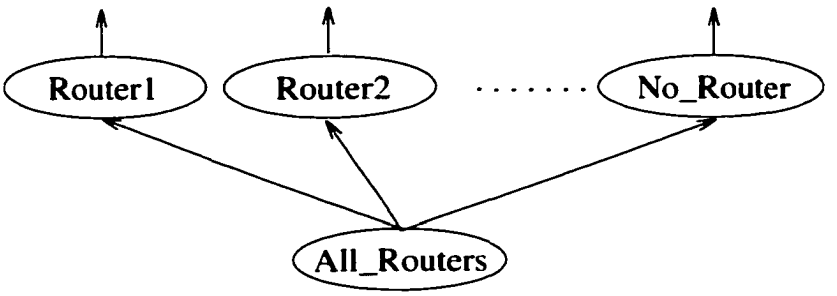


Figure 6.21: The All Routers Network

6.5.2 Build the Best Path Networks

For each destination router, there is at least one best route associated with one of the peers of the host router. Every incoming route is examined to against the current best path. Refering to Figure 6.4, the best path connection between a destination router and a neighbor peer may be changed from time to time if a new path is picked up as the best path based on the decision making criteria. There are two situations for a path becoming the best path for a destination router:

1. a path carries a new orienting router, or

2. the incoming path is better than the current best path.

In order to identify the originating router is new to the host router, the find-router-X network is used to identify the originating router which is noted as an E-node In_Origin as shown in Figure 6.22. Initially, the all-routers network is constructed with the two E-nodes, All_Routers and No_Router, only. Each time a new originating router is discovered, the new router is added into the all-routers network as illustrated in Figure 6.21. Adding a new router into the all-routers network is done by

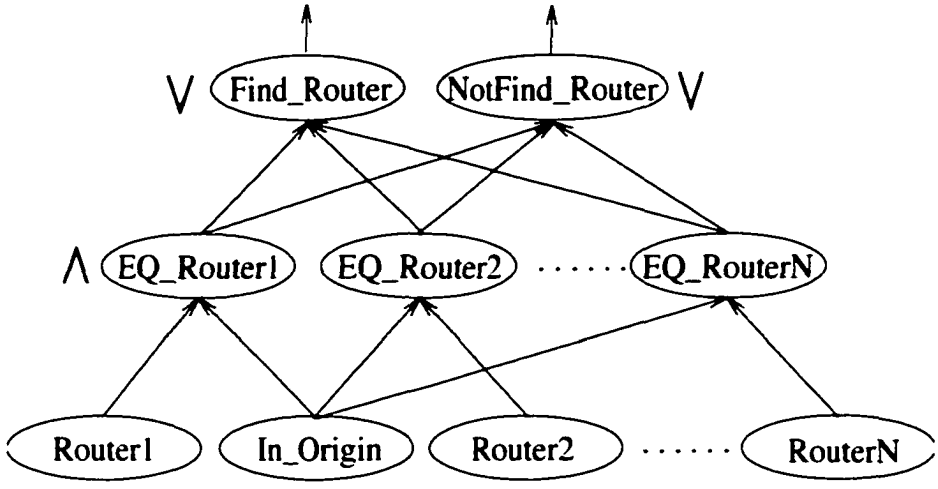


Figure 6.22: The Find Router X Network

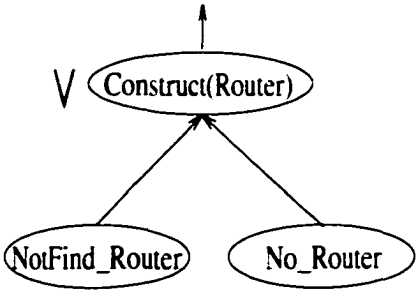


Figure 6.23: The Construct Router X Network

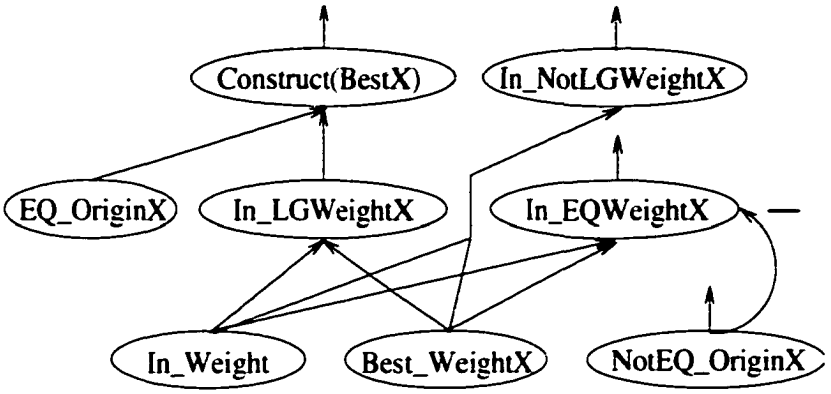


Figure 6.24: The Largest Weight X Network

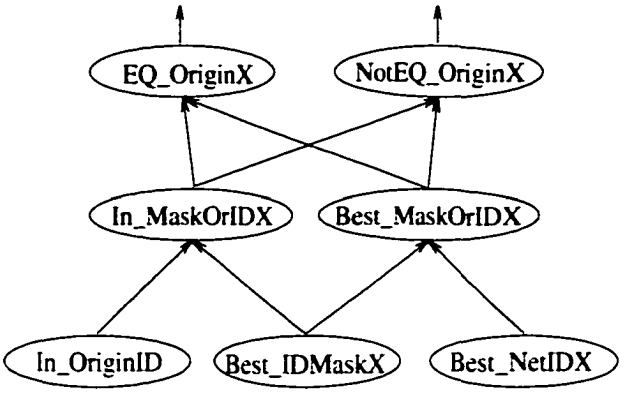


Figure 6.25: The Equal Origin X Network

the construct-router-X network as shown in Figure 6.23.

For the best path selection in this case study, there are attributes: weight, local preference, and AS-path which are considered. According to the decision making sequence, refering to Section 6.1.2, the weight is considered initially. Next, the local preference is the second and the AS-path is the third place to be considered. Figure 6.24 illustrates how a path with a larger weight is selected. The best-X network is constructed or modified only when the originating router is the same as the best origin X. To detect the same origin router is done by the equal-origin-X network as

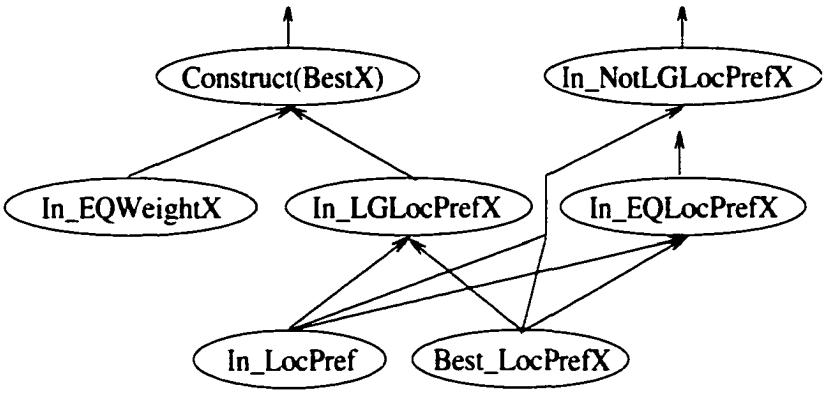


Figure 6.26: The Largest Local Preference X Network

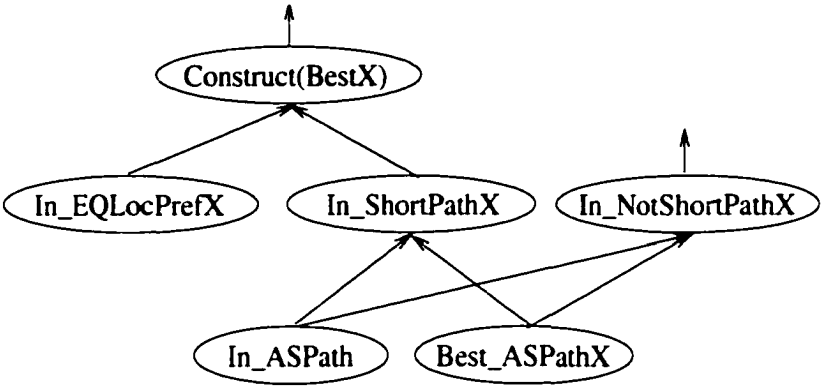


Figure 6.27: The Shortest AS Path X Network

shown in Figure 6.25.

If the largest weight is failed, the path is dropped out with no further consideration. If the two weights are the same, then the local preference is considered to compare to the best-local-preference-X as shown in Figure 6.26. In this network, the activation of the E-node Construct(Best_X) is restricted by the E-node In_EQWeightX.

If the largest local preference is failed, the path is dropped out with no further consideration. If the two local preferences are the same, then the AS path

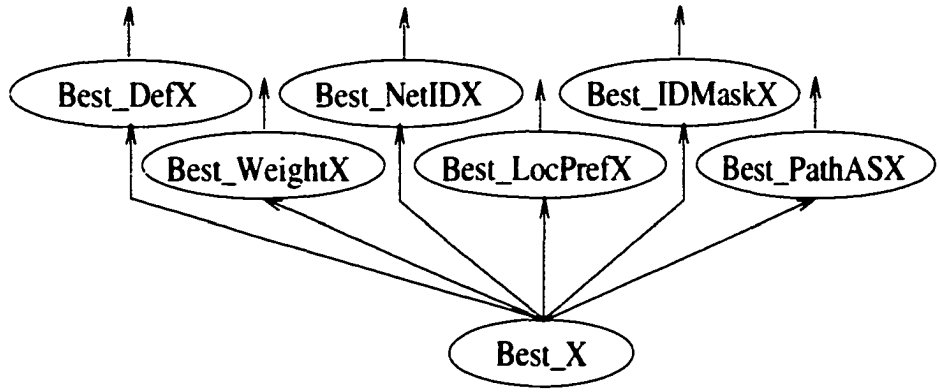


Figure 6.28: The Best X Network

is considered to compare to the best-As-path-X as shown in Figure 6.27. In this network, the activation of the E-node Construct(Best_X) is restricted by the E-node In_EQASPathX. The best-X network is shown in Figure 6.28. The script BestX instructs how to update the link for the destination router to the best neighbor peer.

6.5.3 Build the Second Best Path Networks

The second best path information which is added into the routing networks is a feature for this case study only in order to demonstrate how an alternative routing option can be implemented using LSCAN. As same as the best path selections, there are two conditions for a path to be the second best path:

1. a path is not the best path and there are only two paths for an orienting router,

   or

2. the incoming path is better than the current second best path.

The first path attribute consideration is the weight. As shown in Figure 6.29, the weight was failed in the largest weight network is the condition of selecting the second largest weight. If the weight is larger than the current second largest weight, the path

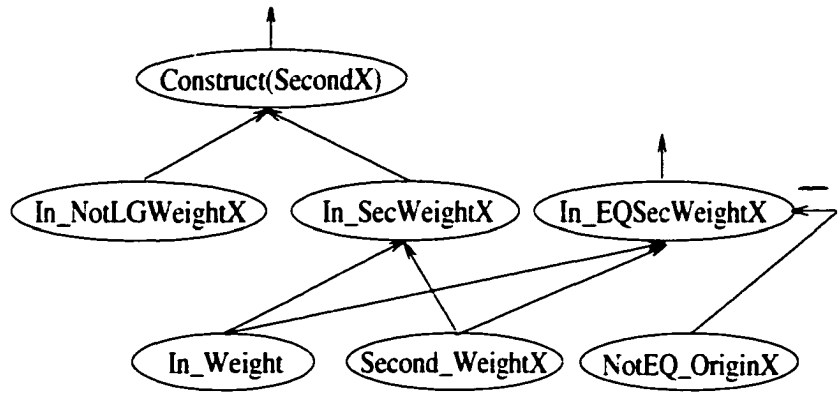


Figure 6.29: The Second Largest Weight X Network

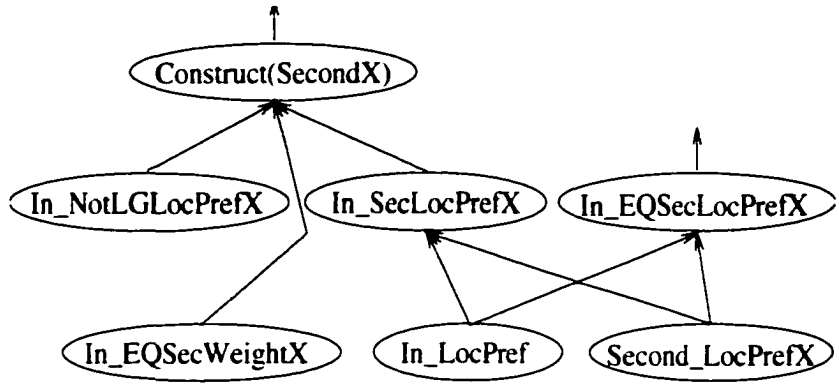


Figure 6.30: The Second Largest Local Preference X Network

is selected as the second best path. If the weight is smaller than the current second largest weight, the path is no longer considered. If the weight is equal to the current second largest weight, then the local preference is further considered.

Figure 6.30 illustrates how the path is selected as the second best path based on the local preference attribute. If the local preference is larger than the current second largest local preference, the path is selected as the second best path. If the local preference is smaller than the current second largest local preference, the path is no longer considered. If the local preference is equal to the current second largest

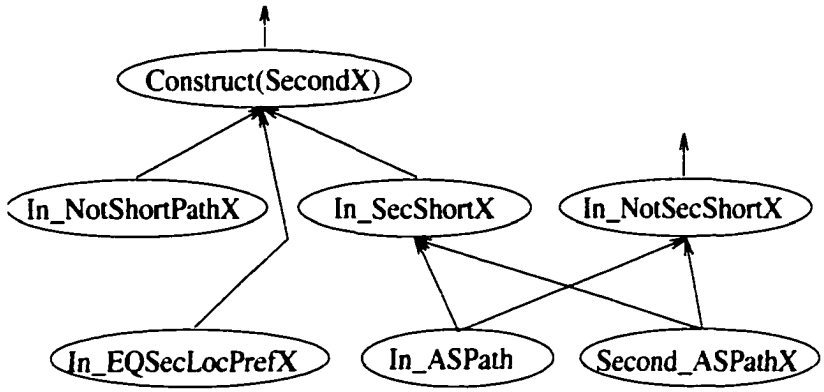


Figure 6.31: The Second Shortest AS Path X Network

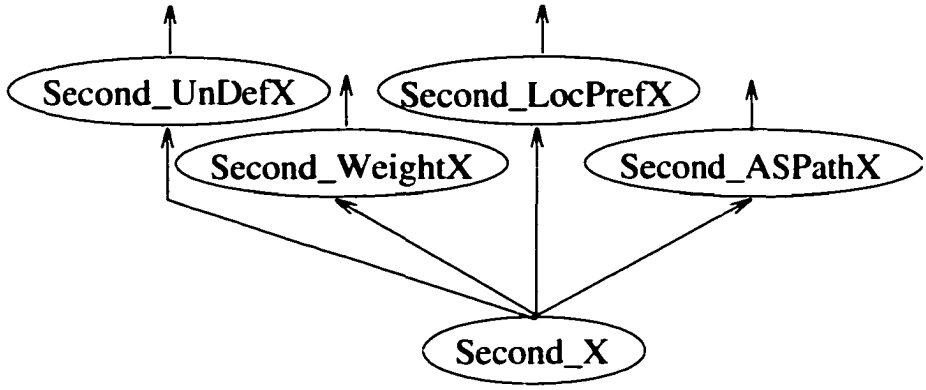


Figure 6.32: The Second Best X Network

local preference then the AS path is further considered.

Figure 6.31 illustrates how the path is selected as the second best path based on the AS path attribute. The path with a shorter AS path is selected as the second best path. Otherwise, the second best path is not changed. The second-best-X network is shown in Figure 6.32. The script SecondX instructs how to update the link for the destination router to the second best neighbor peer.

6.6 Summary

This chapter introduced a case study for applying LSCAN networks to the BGP4 network routing protocols. This case study explored the requirements for building LSCAN networks to perform the routing tasks without external processing. This means that all LSCAN subnetworks can self-support required functionalities, which include calling external procedures to implement some specific computations, establishing subnetwork nodes and connections through some external specification scripts, and even creating a new subnetwork through some external scripts for definitions. All of these kinds of functionalities are embedded in the processing of the nodes. Based on the given networks, the main required subnetworks were provided. Those subnetworks form the entire networks for any one BGP4 router. A small amount of learning or training is required on the routing subnetwork as shown in Figure 6.4.

The main differences between the networks built for the BGP4 routing processing and for the hand-printed digit classifications are in two aspects. The first aspect is the learning processing. In the hand-printed digit classification networks, a massive learning processing is involved. In the routing processing networks, learning processing happens only on a small portion of the entire subnetworks. The learning processing is more direct compared to the hand-printed digit classification networks. The second aspect is the structure of the subnetworks. The hand-printed digit classification networks contain more uniformed subnetworks. PDMs and classification nodes are created in an identical way. Conversely, the subnetworks for the BGP4 routing networks are more unique than the others. Subnetwork nodes creation benefits from the guidance of external scripts.

CHAPTER 7

CONCLUSION AND FUTURE WORK

This research was motivated by providing solutions to symbolic processing with learning and distributed computing capability. Furthermore, the proposed networks should maintain the structures of causal relations between symbolic entities. Symbolic entity composition and decomposition are also facilitated. LSCAN networks are representations of knowledge formalisms expressed in Section 3.1. Variable bindings are embedded into the knowledge formalism expressions, therefore, they are represented in LSCAN networks.

The activation function of the LSCAN networks is a general purpose evaluation function. The activation function can derive a decision making with any combination of rigid-logic and soft-logic from the input nodes. At the output level, the strength of a decision making can be maintained as a soft-logic or a rigid-logic. In feed-forward learning techniques, LSCAN networks can be structured using dual-input connections for each input node in order to extend the learning techniques. Using these extended feed-forward learning techniques discussed in Chapter 4, more applications can be explored. For example, sequential learning can gain advantages from these extended feed-forward learning techniques.

Similarity between any two symbolic entities can be measured using the LSCAN activation function, if the two symbolic entities have similar inputs. Re-

ferring to Figure 3.3, while the output strength is above 0.6, the output strength maintains a linear relation with the input effect. The output strength drops sharply while the ratio $\frac{\varepsilon}{\delta_{sf}}$, refer to Equation 3.2, is lower than 0.62. This implies the linearity of the similarity is maintained with the ratio above 0.62.

7.1 Summary of Case Study Results

In Chapter 5, the LSCAN networks implemented classification for the NIST hand-printed digit images. The experiments showed that the LSCAN networks were capable to achieve the goal of the recognition task. However, compared to the experiment results from the P_ARADISE networks as shown in Table 7.1 using Gaussian filters and all of the 3,471 digit images, the LSCAN networks generated a larger number of classes. For example, at the level of the accuracy of 71 %, the LSCAN networks created 819 classes. This is 14.89 times of 55 classes as shown in Table 7.1. The possible reason is that under the values of the control parameters, the LSCAN networks could not make more pattern sharing among different numerals. At the same time, there were many patterns shared by different meta-classes, which lead to lower accuracy. However, the LSCAN networks needed only to be trained one time. At the second training pass, there were no new classes created. At the same time, the accuracy ratio was improved. The rule-guided classification showed another important benefit of the LSCAN representation strategy. The experiments showed that the LSCAN networks were able to perform the classification task. When the supporting algorithms in the rule-guided classifications are improved, the LSCAN networks attain better performance directly from the cognitive rules even if the rules are imperfect.

The main problem in the pattern-based classification is the sharing problem. The LSCAN networks used many patterns shared between different numerals. This problem causes the hypothesis test failure which leads to more classes. There are three ways to enhance the results of classification:

- In the experiments of pattern-based classification, new patterns were created only when a new class was created. In this situation, a pattern may be not the best pattern to represent the pattern extracted from the feature planes. If all new patterns are created once they are detected, even there is no new class discovered. Later, a new class is created with more representable patterns which leads to higher accuracy. Therefore, the number of created classes is also reduced.
- Using cognitive rules which identify an input digit image for specific group tags on each created pattern can be evaluated. A pattern represents the extracted pattern from the feature planes only if the tag of a pattern is in the same group of the input digit image. In this way, a pattern is more accurate to represent the extracted pattern from the feature planes. Therefore, it reduces the number of classes.
- A method is to use tags for the created patterns as the previous method does, however, the differences are using global similarity tests instead of rules. A global similarity test is applied to each input digit image in order to identify the group of the input digit image. While a new pattern is created, a group tag is added to the new pattern. In order to increase the accuracy, the pattern represents the extracted pattern from the feature planes only when these two patterns have the same tag.

Table 7.1: Some Experiments Results from $P_ARADISE$

Number of classes	Classes per metaclass	Accuracy (%)
55	5.5	71.2
78	7.8	67.3
125	12.5	81.5

The three suggested methods can lead to better results with higher accuracy and lower number of classes.

The case study in chapter 6 showed a different aspect of using and developing LSCAN networks. This case study explored the needs for establishing LSCAN networks to implement network routing tasks. Some new features and requirements such as external function calls and script-guided network creation were implemented.

7.2 Future Work

LSCAN networks can be viewed in two aspects. One aspect is the viewpoint of connectionist networks. The case study showed in Chapter 5 that hand-printed digit classification can utilize a kind of connectionist network, which has massive learning capability. On the other hand, LSCAN networks can deal with discrete cases such as the case study showed in Chapter 6. In discrete cases, learning happens within the network entities. Some subnetworks need to be trained. Based on these two aspects, LSCAN networks are beneficial in integrating neural and cognitive approaches, but more improvements are possible.

Capitalizing on the advantages of LSCAN networks, it is worth investigating two aspects. One aspect is to seek enhanced learning mechanisms. In a broad view, LSCAN networks can adapt another learning techniques as in the case of pattern learning in Chapter 5. Dealing with different problems may involve benefits from different learning mechanisms. The architectures of LSCAN networks are suitable for these kind of changes. The other aspect is to seek the required mechanisms for extended system integrations. LSCAN networks handle well for discrete events. Integration with the other connectionist networks is highly worthwhile. Those connectionist networks can be merged with LSCAN networks using different learning mechanisms and activation functions. Furthermore, the lateral connections and learning may be employed to readily achieve these goals.

APPENDIX A

BGP4 ROUTER CONFIGURATIONS

This appendix lists the configurations for all BGP routers.

RTA#

```
hostname RTA

ip subnet-zero

interface Loopback0
 ip address 201.250.13.41 255.255.255.0

interface Ethernet0
 ip address 201.250.14.1 255.255.255.0

interface Serial0
 ip address 128.203.63.1 255.255.255.252

router ospf 10
 redistribute bgp 10 metric 2000 subnets
 passive-interface Serial0
 network 201.250.0.0 0.0.255.255 area 0
 network 128.203.0.0 0.0.255.255 area 0
 default-information originate metric 2000

router bgp 10
 no synchronization
 network 201.250.13.0
 network 201.250.14.0
 neighbor 128.203.63.2 remote-as 20
 neighbor 128.203.63.2 route-map setlocalpref in
 neighbor 201.250.15.2 remote-as 10
 neighbor 201.250.15.2 update-source Loopback0

ip classless
ip default-network 180.180.0.0
```

```
route-map setlocalpref permit 10
 set local-preference 20
```

RTF#

```
hostname RTF

ip subnet-zero

interface Ethernet0
 ip address 201.250.14.2 255.255.255.0

interface Serial1
 ip address 201.250.15.1 255.255.255.252

router ospf 10
 network 201.250.0.0 0.0.255.255 area 0

ip classless
```

RTB#

```
hostname RTB

ip subnet-zero

interface Loopback1
 ip address 201.250.15.10 255.255.255.252
```

```
interface Serial0
 ip address 201.250.15.2 255.255.255.252
!
interface Serial1
 ip address 192.218.10.6 255.255.255.252

router ospf 10
 redistribute bgp 10 metric 1000 subnets
 passive-interface Serial1
 network 201.250.0.0 0.0.255.255 area 0
 network 192.218.10.6 0.0.0.0 area 0
 default-information originate metric 1000
!
router bgp 10
 no synchronization
 network 201.250.15.0
 neighbor 192.218.10.5 remote-as 30
 neighbor 192.218.10.5 route-map localonly in
 neighbor 95.207.23.1 remote-as 60
 neighbor 95.207.23.1 route-map localonly in
 neighbor 201.250.13.41 remote-as 10
!
ip classless
ip default-network 192.218.10.0
ip as-path access-list 1 permit ^30$
ip as-path access-list 2 permit ^60$

route-map localonly permit 10
 match as-path 1
 match as-path 2
 set local-preference 30
```

RTC#

```
hostname RTC

ip subnet-zero

interface Loopback0
 ip address 128.203.63.130 255.255.255.192

interface Serial2/0
 ip address 128.203.63.5 255.255.255.252
!
interface Serial2/1
 ip address 128.203.63.2 255.255.255.252

router bgp 20
 network 128.203.0.0
 aggregate-address 128.203.0.0 255.255.0.0 summary-only
 neighbor 128.203.63.1 remote-as 10
 neighbor 128.203.63.1 distribute-list 1 out
 neighbor 128.203.63.6 remote-as 40

ip classless
access-list 1 deny   175.201.0.0 0.0.255.255
access-list 1 permit any
```

RTD#

```
hostname RTD

ip subnet-zero

interface Loopback0
 ip address 192.218.10.174 255.255.255.192
!
```

```
interface Serial0/0
 ip address 192.218.10.5 255.255.255.252
!
interface Serial0/1
 ip address 192.218.10.2 255.255.255.252

router bgp 30
 network 192.218.10.0
 neighbor 192.218.10.1 remote-as 50
 neighbor 192.218.10.6 remote-as 10
```

RTE#

```
hostname RTE

ip subnet-zero

interface Loopback0
 ip address 180.180.10.1 255.255.255.0

interface Serial0
 ip address 175.201.10.2 255.255.255.252

interface Serial1
 ip address 128.203.63.6 255.255.255.252

router bgp 40
 network 180.180.10.0
 aggregate-address 180.180.0.0 255.255.0.0 summary-only
 neighbor 128.203.63.5 remote-as 20
 neighbor 175.201.10.1 remote-as 50
 neighbor 95.207.23.4 remote-as 60
 neighbor 95.207.23.4 route-map setlist out
```

```
ip classless
access-list 1 deny 175.201.0.0 0.0.255.255
!
route-map setlist permit 10
 match ip address 1
```

RTG#

```
hostname RTG

ip subnet-zero

interface Loopback0
 ip address 175.201.10.174 255.255.255.192

interface Serial0
 ip address 192.218.10.1 255.255.255.252

interface Serial1
 ip address 175.201.10.1 255.255.255.252

router bgp 50
 network 175.201.10.0
 aggregate-address 175.201.0.0 255.255.0.0 summary-only
 neighbor 192.218.10.2 remote-as 30
 neighbor 192.218.10.2 send-community
 neighbor 192.218.10.2 route-map setcommunity out
 neighbor 175.201.10.2 remote-as 40
!
ip classless
access-list 1 permit 175.201.0.0 0.0.255.255
access-list 2 permit any
```

```
!access-list 101 permit ip 175.201.0.0 0.0.255.255 host 255.255.0.0
!
route-map setcommunity permit 10
 match ip address 1
 set community no-export

route-map setcommunity permit 20
 match ip address 2
```

RTH#

```
hostname RTH

ip subnet-zero

interface Loopback0
 ip address 95.207.23.37 255.255.255.248

interface Serial0
 ip address 95.207.23.1 255.255.255.252

interface Serial1
 ip address 95.207.23.4 255.255.255.248

router bgp 60
 network 95.207.23.0
 aggregate-address 95.207.0.0 255.255.0.0 summary-only
 neighbor 95.207.23.2 remote-as 10
 neighbor 95.207.23.5 remote-as 40
 neighbor 95.207.23.5 route-map localonly in
!
ip classless
ip as-path access-list 1 permit ^40$
```

```
route-map localonly permit 10
 match as-path 1
```

APPENDIX B

LISTS OF NETWORKS AND ROUTES

Local Preference Default: 100
Metric Default: 0
Weight Default: 32786 for paths that the router originates and 0 for all others paths.
Status: $* - valid, > - best, @ - second, \wedge - default, ! - drop$
AS-Path: $i - internal$

RTA: Default Network 180.180.0.0

Origin		Network	Next Hop	Loc. Pref.	Weight	Metric	AS-Path
RTA	>	201.250.13.0/24	0.0.0.0		32768	0	i
RTA	>	201.250.14.0/24	0.0.0.0		32768	0	i
RTA	!*	201.250.13.0/24	RTC	200	0	0	20 10
RTA	!*	201.250.14.0/24	RTC	200	0	0	20 10
RTB	>	201.250.15.0/24	RTB	100	0	0	i
RTB	!*	201.250.15.0/24	RTC	100	0	0	20 40 60 10
RTC	>	128.203.0.0/16	RTC	200	0	0	20
RTC	!@	128.203.0.0/16	RTB	300	0	0	10 60 40 20 i
RTD	>	192.218.10.0/24	RTB	300	0	0	10 30 i
RTD	!@	192.218.10.0/24	RTC	200	0	0	20 40 50 30
RTE	!\wedge@	180.180.10.0/16	RTB	300	0	0	10 60 40 i
RTE	> \wedge	180.180.10.0/16	RTC	200	0	0	20 40
RTH	>	95.207.23.0/24	RTB	300	0	0	10 60 i
RTH	!@	95.207.23.0/24	RTC	200	0	0	20 40 60

RTB: Default Network 192.218.10.0

Origin		Network	Next Hop	Loc. Pref.	Weight	Metric	AS-Path
RTA	>	201.250.13.0/24	RTA	100	32768	0	i
RTA	>	201.250.14.0/24	RTA	100	32768	0	i
RTA	!*	201.250.13.0/24	RTD	100	0	0	30 50 40 20 10
RTA	!*	201.250.14.0/24	RTD	100	0	0	30 50 40 20 10
RTA	!*	201.250.13.0/24	RTH	100	0	0	60 40 20 10
RTA	!*	201.250.14.0/24	RTH	100	0	0	60 40 20 10
RTB	!*	201.250.15.0/24	RTD	300	0	0	30 10
RTB	!*	201.250.15.0/24	RTH	300	0	0	60 10
RTC	>	128.203.0.0/16	RTA	200	0	0	10 20 i
RTC	!*	128.203.0.0/16	RTD	300	0	0	30 50 40 20
RTC	!@	128.203.0.0/16	RTH	300	0	0	60 40 20
RTD	> ^	192.218.10.0/24	RTD	300	0	0	30
RTD	!^	192.218.10.0/24	RTA	300	0	0	10 20 40 50 30
RTD	!@	192.218.10.0/24	RTH	300	0	0	60 40 50 30
RTE	!*	180.180.10.0/16	RTA	200	0	0	10 20 40 i
RTE	>	180.180.10.0/16	RTH	300	0	0	60 40
RTE	!@	180.180.10.0/16	RTD	300	0	0	30 50 40
RTH	>	95.207.23.0/24	RTH	300	0	0	60
RTH	!*	95.207.23.0/24	RTA	200	0	0	10 20 40 60 i
RTH	!@	95.207.23.0/24	RTD	300	0	0	30 50 40 60

RTC: Deny 175.201.0.0/16 out to RTA

Origin		Network	Next Hop	Loc. Pref.	Weight	Metric	AS-Path
RTA	>	201.250.13.0/24	RTA	100	0	0	10
RTA	>	201.250.14.0/24	RTA	100	0	0	10
RTA	!@	201.250.13.0/24	RTE	100	0	0	10 60 40 i
RTA	!@	201.250.14.0/24	RTE	100	0	0	10 60 40 i
RTB	>	201.250.15.0/24	RTA	100	0	0	10
RTB	!@	201.250.15.0/24	RTE	100	0	0	40 60 10
RTC	!*	128.203.0.0/16	RTA	100	0	0	10 20
RTC	!*	128.203.0.0/16	RTE	100	0	0	40 20
RTD	>	192.218.10.0/24	RTA	100	0	0	10 30 i
RTD	!@	192.218.10.0/24	RTE	100	0	0	40 50 30
RTE	>	180.180.10.0/16	RTE	100	0	0	40
RTG	!*	175.201.10.0/16	RTE	100	0	0	40 50
RTH	!@	95.207.23.0/16	RTA	100	0	0	10 10 60 i
RTH	>	95.207.23.0/16	RTE	100	0	0	40 60

RTD:

Origin		Network	Next Hop	Loc. Pref.	Weight	Metric	AS-Path
RTA	>	201.250.13.0/24	RTB	100	0	0	10 i
RTA	>	201.250.14.0/24	RTB	100	0	0	10 i
RTA	!@	201.250.13.0/24	RTG	100	0	0	50 40 20 10
RTA	!@	201.250.14.0/24	RTG	100	0	0	50 40 20 10
RTB	>	201.250.15.0/24	RTB	100	0	0	10
RTB	!@	201.250.15.0/24	RTG	100	0	0	50 40 60 10
RTC	>	128.203.0.0/16	RTB	100	0	0	10 20 i
RTC	!@	128.203.0.0/16	RTG	100	0	0	50 40 20
RTD	!*	192.218.10.0/24	RTB	100	0	0	10 30
RTD	!*	192.218.10.0/24	RTG	100	0	0	50 30
RTE	!@	180.180.10.0/16	RTB	100	0	0	10 10 20 40
RTE	>	180.180.10.0/16	RTG	100	0	0	50 40
RTG	>	175.201.10.0/16	RTG	100	0	0	50
RTH	>	95.207.23.0/16	RTB	100	0	0	10 60
RTH	!@	95.207.23.0/16	RTG	100	0	0	50 40 60

RTE:

Origin		Network	Next Hop	Loc. Pref.	Weight	Metric	AS-Path
RTA	>	201.250.13.0/24	RTC	100	0	0	20 10
RTA	>	201.250.14.0/24	RTC	100	0	0	20 10
RTA	!*	201.250.13.0/24	RTG	100	0	0	50 30 10 i
RTA	!*	201.250.14.0/24	RTG	100	0	0	50 30 10 i
RTA	!@	201.250.13.0/24	RTH	100	0	0	60 10 i
RTA	!@	201.250.14.0/24	RTH	100	0	0	60 10 i
RTB	!@	201.250.15.0/24	RTC	100	0	0	20 10 i
RTB	!*	201.250.15.0/24	RTG	100	0	0	50 30 10
RTB	>	201.250.15.0/24	RTH	100	0	0	60 10
RTC	>	128.203.0.0/16	RTC	100	0	0	20
RTC	!*	128.203.0.0/16	RTG	100	0	0	50 30 10 20 i
RTC	!@	128.203.0.0/16	RTH	100	0	0	60 10 20 i
RTD	!*	192.218.10.0/24	RTC	100	0	0	20 10 30 i
RTD	>	192.218.10.0/24	RTG	100	0	0	50 30
RTD	!@	192.218.10.0/24	RTH	100	0	0	60 10 30
RTE	!*	180.180.10.0/16	RTC	100	0	0	20 40
RTE	!*	180.180.10.0/16	RTD	100	0	0	50 40
RTE	!*	180.180.10.0/16	RTH	100	0	0	60 40
RTG	>	175.201.10.0/16	RTG	100	0	0	50
RTH	>	95.207.23.0/16	RTH	100	0	0	60
RTH	!@	95.207.23.0/16	RTC	100	0	0	20 10 60 i
RTH	!*	95.207.23.0/16	RTG	100	0	0	50 30 10 60

RTG:

Origin		Network	Next Hop	Loc. Pref.	Weight	Metric	AS-Path
RTA	>	201.250.13.0/24	RTD	100	0	0	30 10 i
RTA	>	201.250.14.0/24	RTD	100	0	0	30 10 i
RTA	!@	201.250.13.0/24	RTE	100	0	0	40 20 10
RTA	!@	201.250.14.0/24	RTE	100	0	0	40 20 10
RTB	>	201.250.15.0/24	RTD	100	0	0	30 10
RTB	!@	201.250.15.0/24	RTE	100	0	0	40 60 10
RTC	!@	128.203.0.0/16	RTD	100	0	0	30 10 10 20 i
RTC	>	128.203.0.0/16	RTE	100	0	0	40 20
RTD	>	192.218.10.0/24	RTD	100	0	0	30
RTD	!@	192.218.10.0/24	RTE	100	0	0	40 60 10 30
RTE	!@	180.180.10.0/16	RTD	100	0	0	30 10 60 40
RTE	>	180.180.10.0/16	RTE	100	0	0	40
RTG	!*	175.201.10.0/16	RTD	100	0	0	30 50
RTG	!*	175.201.10.0/16	RTE	100	0	0	40 50
RTH	!@	95.207.23.0/16	RTD	100	0	0	30 10 60
RTH	>	95.207.23.0/16	RTE	100	0	0	40 60

RTH:

Origin		Network	Next Hop	Loc. Pref.	Weight	Metric	AS-Path
RTA	>	201.250.13.0/24	RTB	100	0	0	10 i
RTA	>	201.250.14.0/24	RTB	100	0	0	10 i
RTA	!@	201.250.13.0/24	RTE	100	0	0	40 20 10
RTA	!@	201.250.14.0/24	RTE	100	0	0	40 20 10
RTB	>	201.250.15.0/24	RTB	100	0	0	10
RTB	!@	201.250.15.0/24	RTE	100	0	0	40 20 10 i
RTC	!@	128.203.0.0/16	RTB	100	0	0	10 20 i
RTC	>	128.203.0.0/16	RTE	100	0	0	40 20
RTD	>	192.218.10.0/24	RTB	100	0	0	10 30
RTD	!@	192.218.10.0/24	RTE	100	0	0	40 50 30
RTE	!@	180.180.10.0/16	RTB	100	0	0	10 30 50 40
RTE	>	180.180.10.0/16	RTE	100	0	0	40
RTG	!@	175.201.10.0/16	RTB	100	0	0	10 30 50
RTG	>	175.201.10.0/16	RTE	100	0	0	40 50
RTH	!*	95.207.23.0/16	RTB	100	0	0	10 60
RTH	!*	95.207.23.0/16	RTE	100	0	0	40 60

APPENDIX C

ROUTING NETWORKS

This appendix shows the routing networks for the routers: RTB, RTC, RTD, RTE, RTG, and RTH. These routing networks are shown in Figures C.1, C.2, C.3, C.4, C.5, and C.6. At routing time, the destination matching networks for the routers RTC, RTD, RTE, RTG, and RTH are shown in Figures C.7, C.8, C.9, C.10, and C.11.

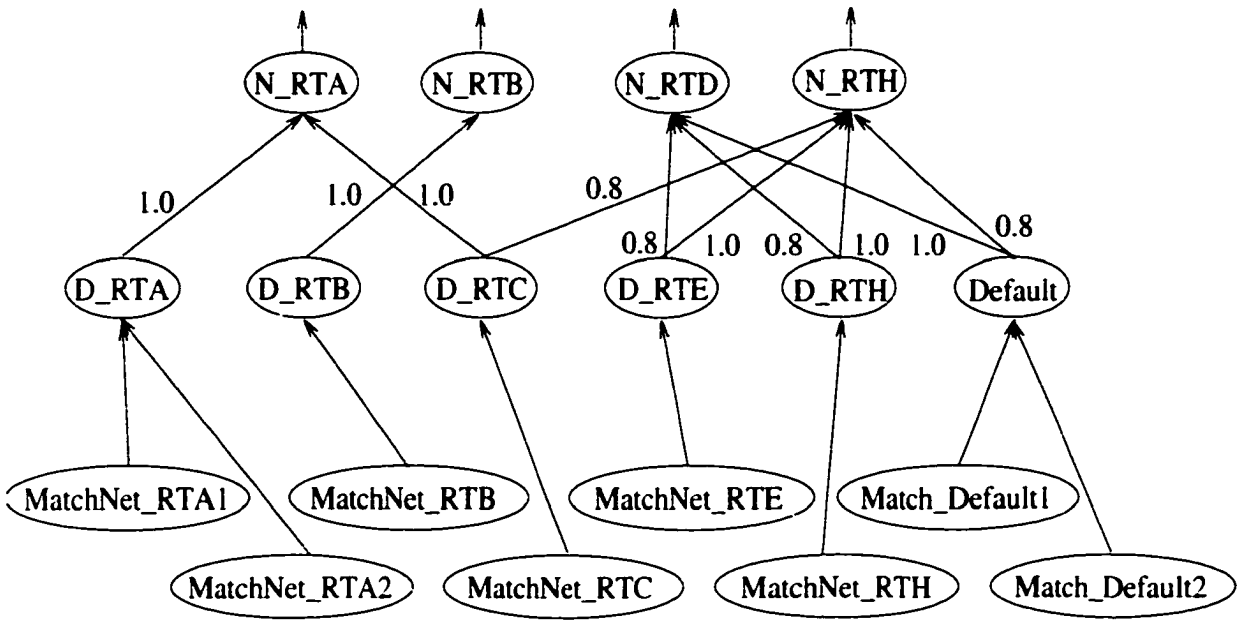


Figure C.1: The RTB Routing Network

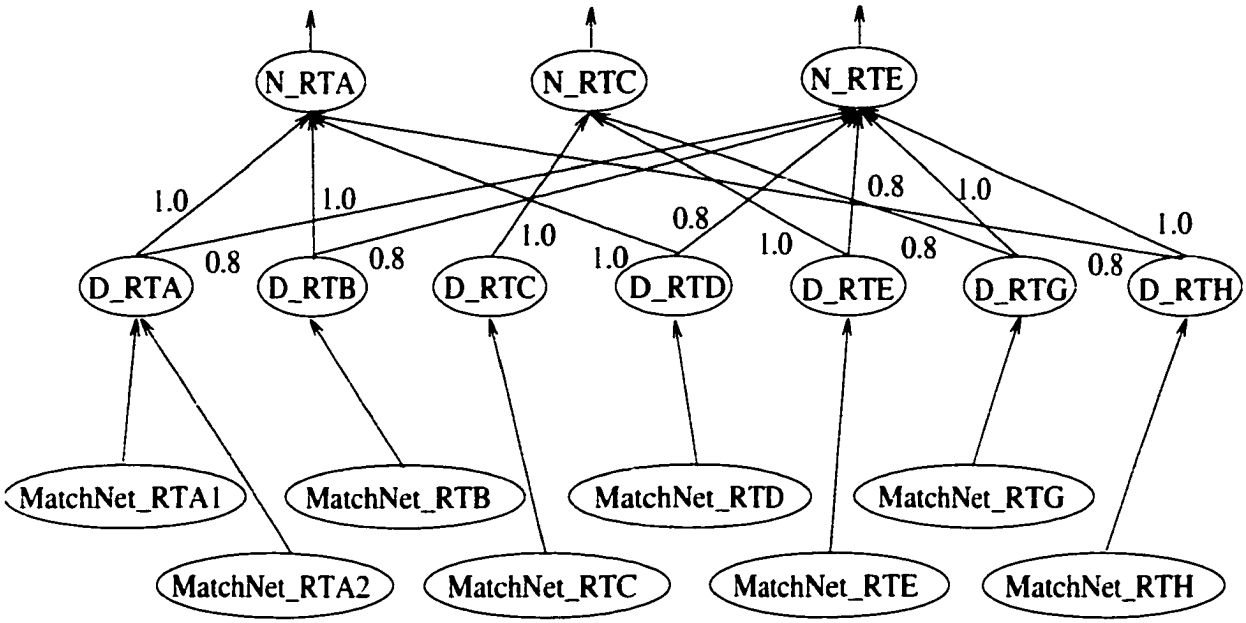


Figure C.2: The RTC Routing Network

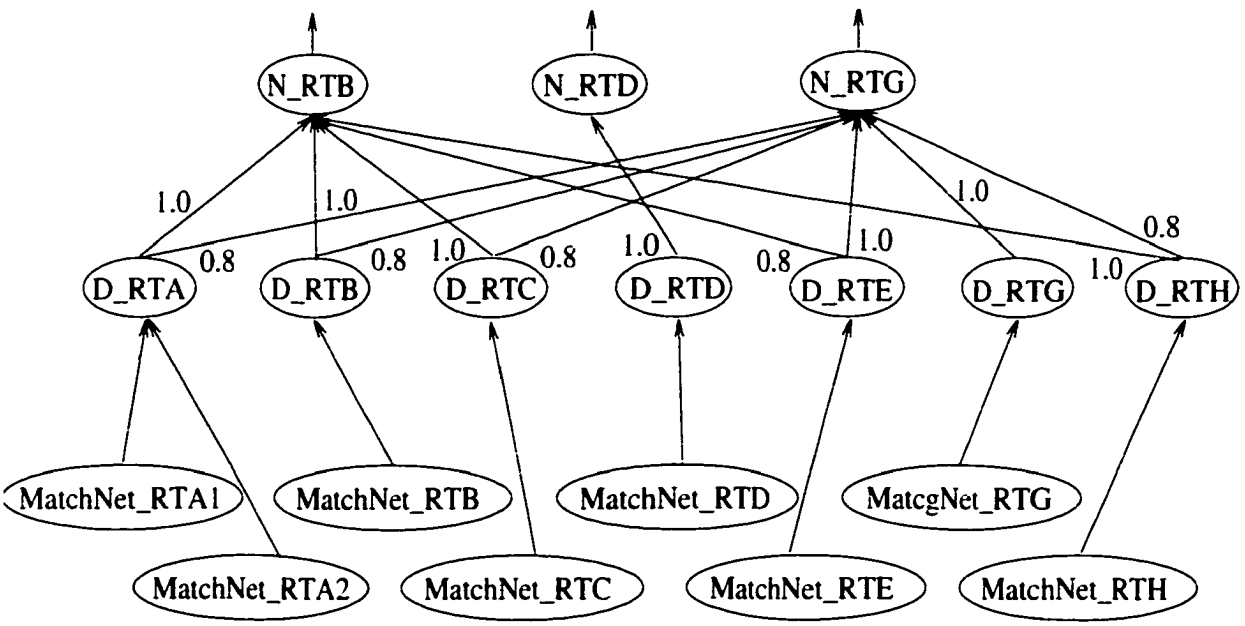
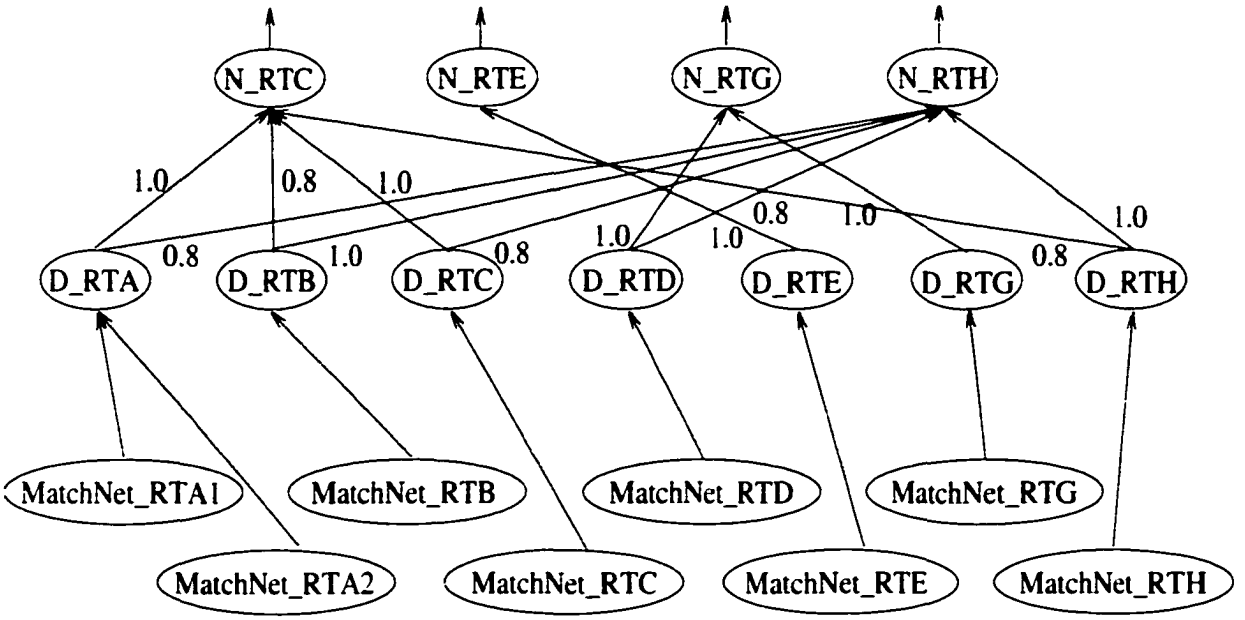
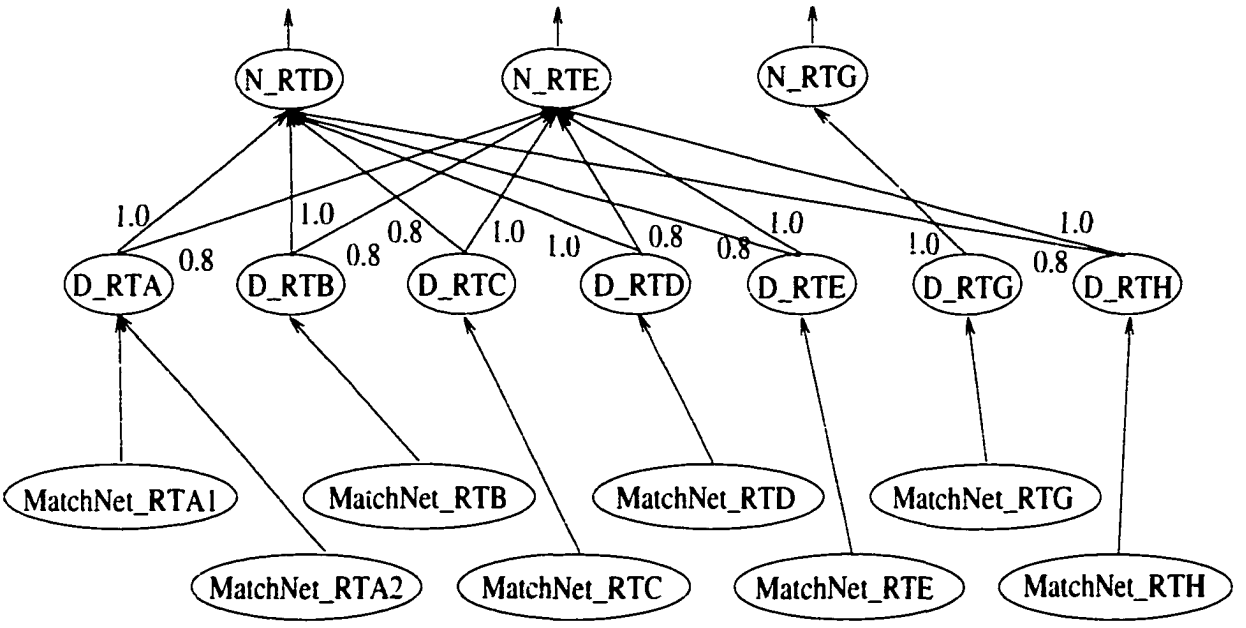


Figure C.3: The RTD Routing Network



All inbound links are OR links

Figure C.4: The RTE Routing Network



All inbound links are OR links

Figure C.5: The RTG Routing Network

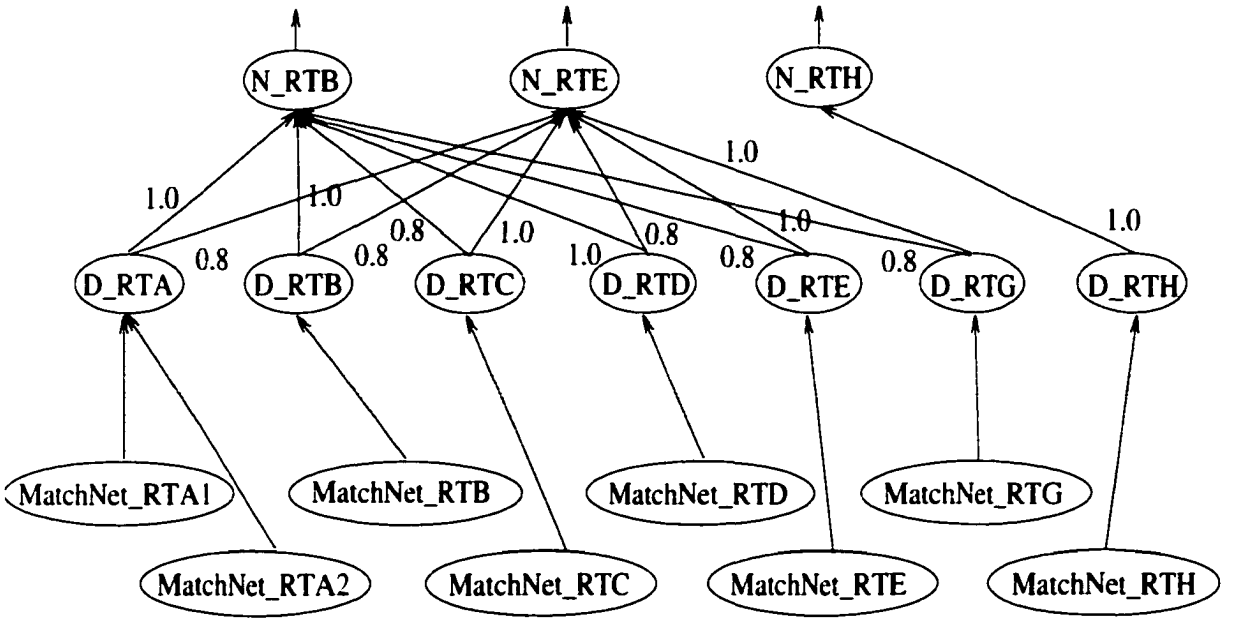


Figure C.6: The RTH Routing Network

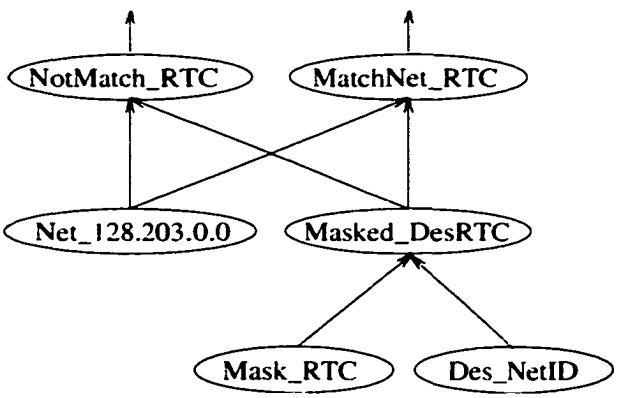


Figure C.7: The RTC Destination Matching Networks

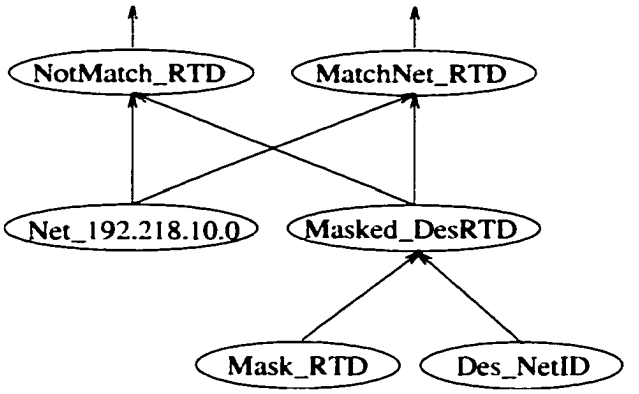


Figure C.8: The RTD Destination Matching Networks

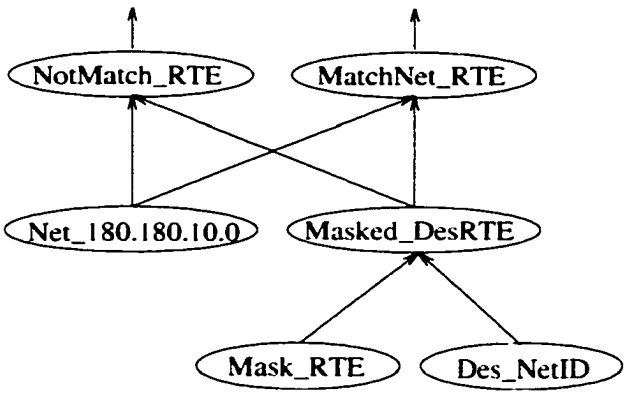


Figure C.9: The RTE Destination Matching Networks

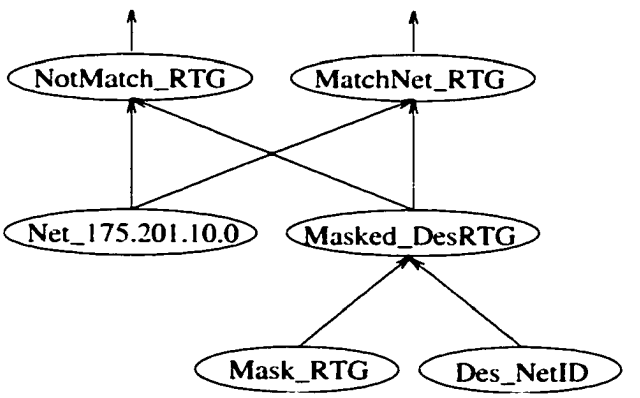


Figure C.10: The RTG Destination Matching Networks

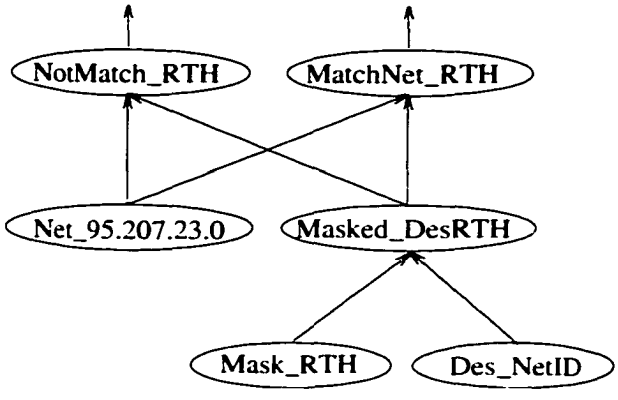


Figure C.11: The RTH Destination Matching Networks

APPENDIX D

NIST HAND-PRINTED DIGIT IMAGES

This appendix displays all 3471 hand-printed digits sorted in numerical orders.

Figure D.1: Digit 9 with 339 Images

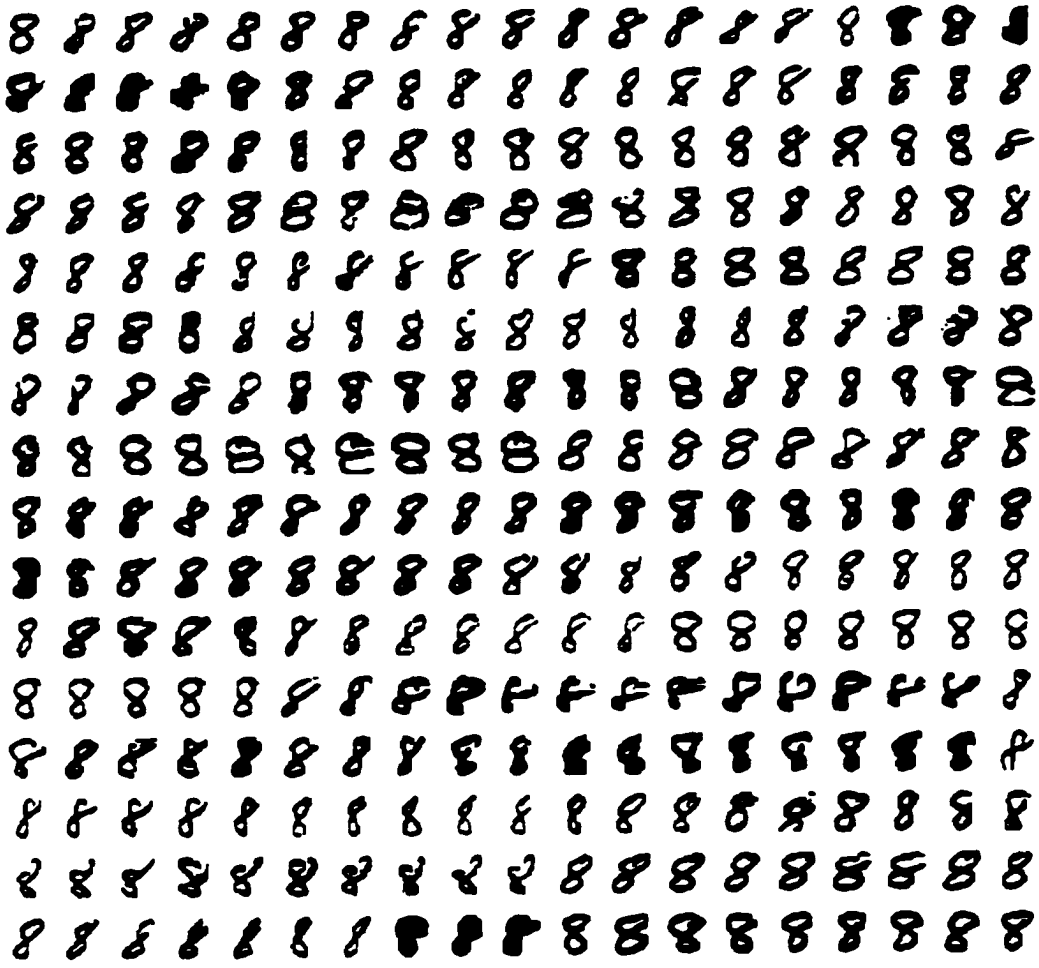


Figure D.2: Digit 8 with 304 Images

Figure D.3: Digit 7 with 347 Images

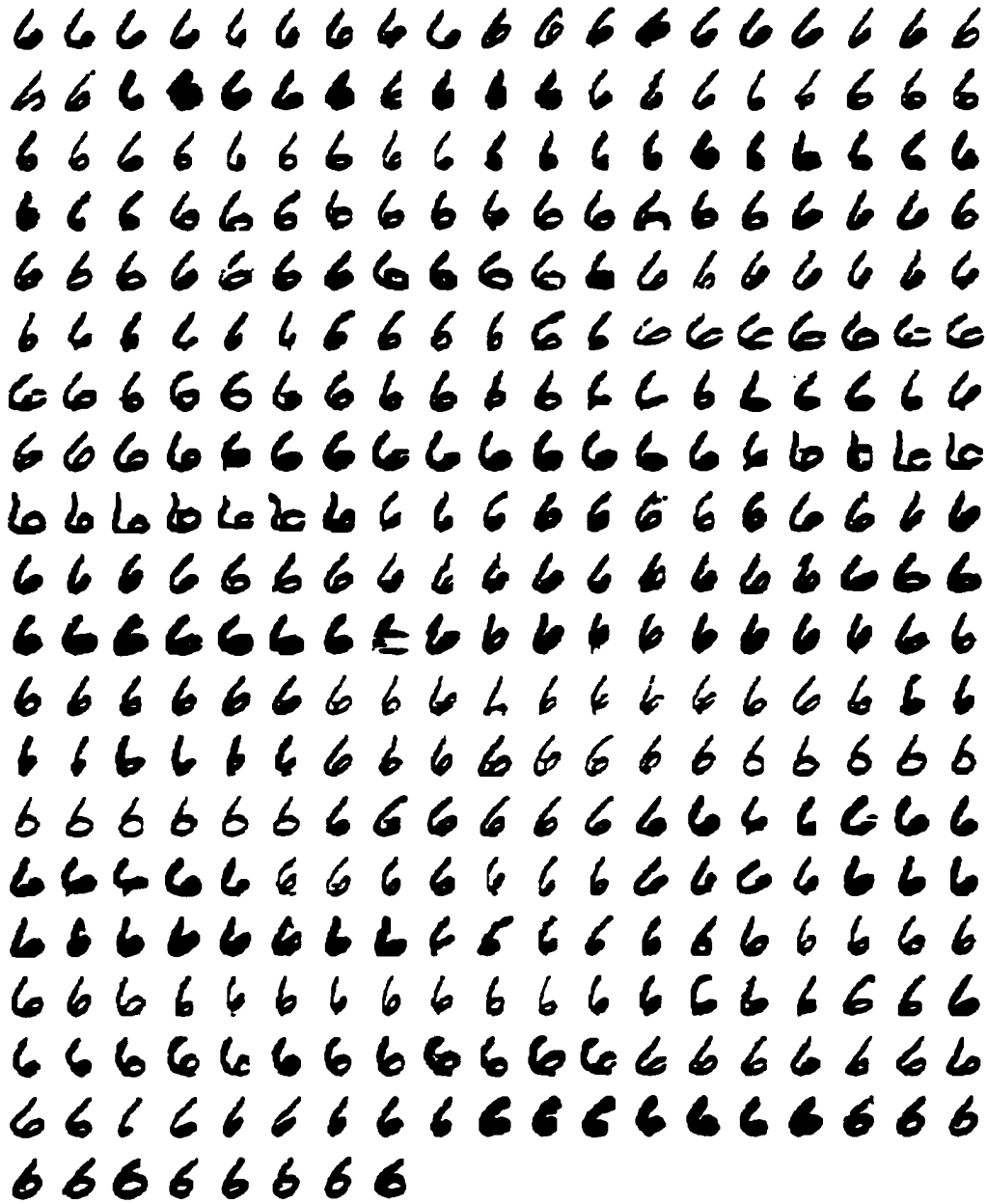


Figure D.4: Digit 6 with 369 Images

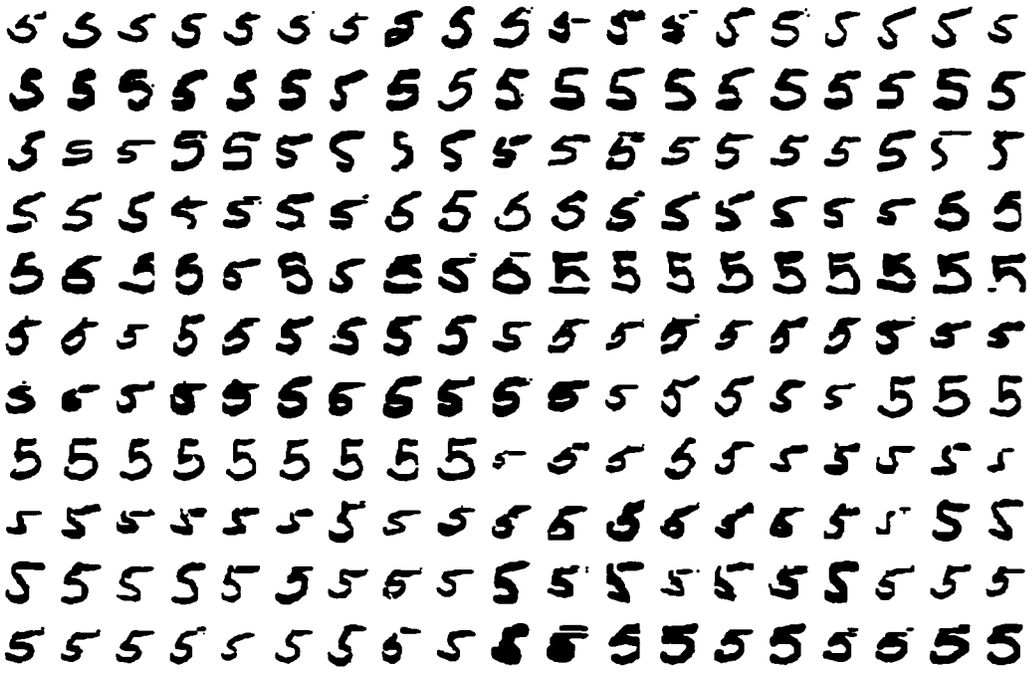


Figure D.5: Digit 5 with 209 Images

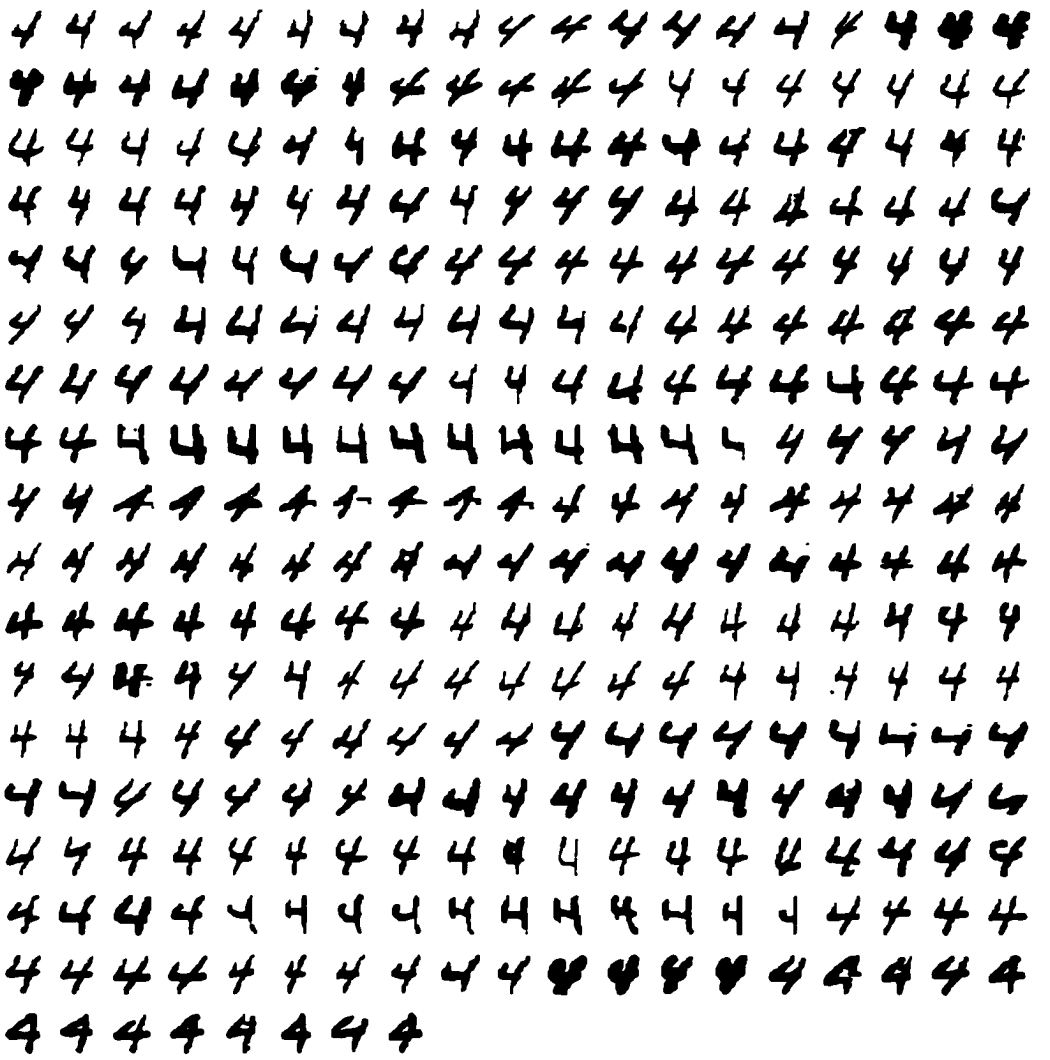


Figure D.6: Digit 4 with 331 Images

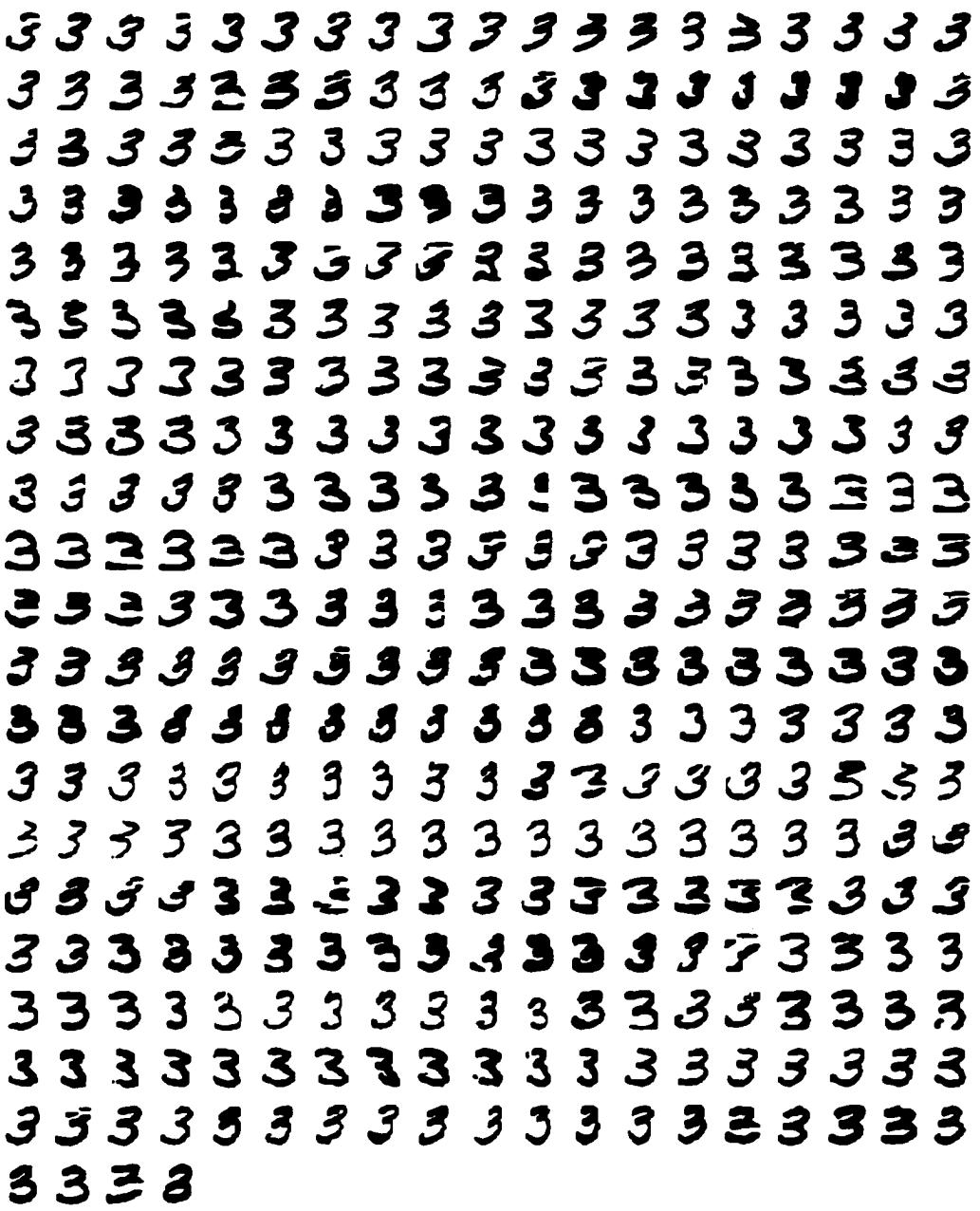


Figure D.7: Digit 3 with 384 Images

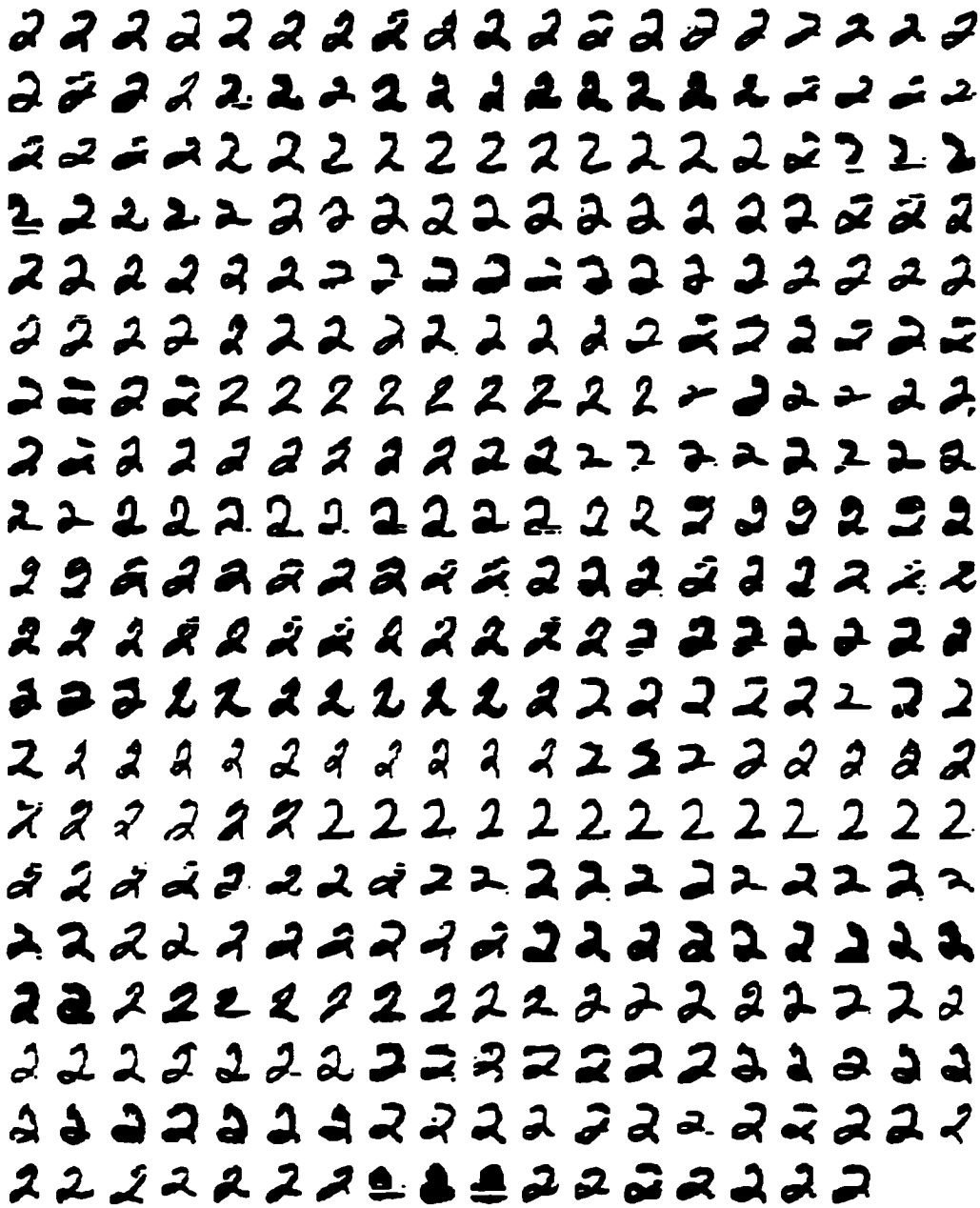


Figure D.8: Digit 2 with 378 Images

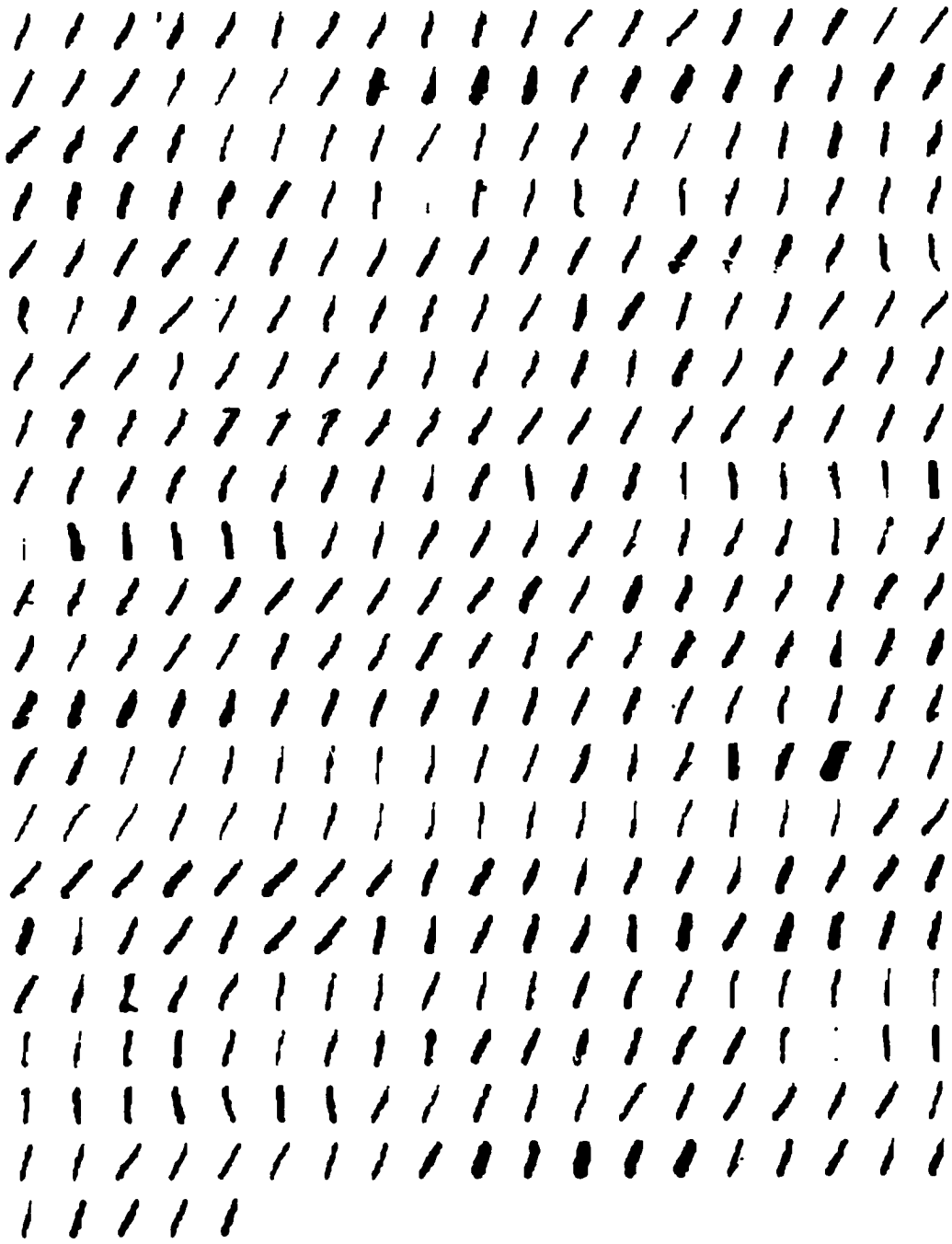


Figure D.9: Digit 1 with 404 Images

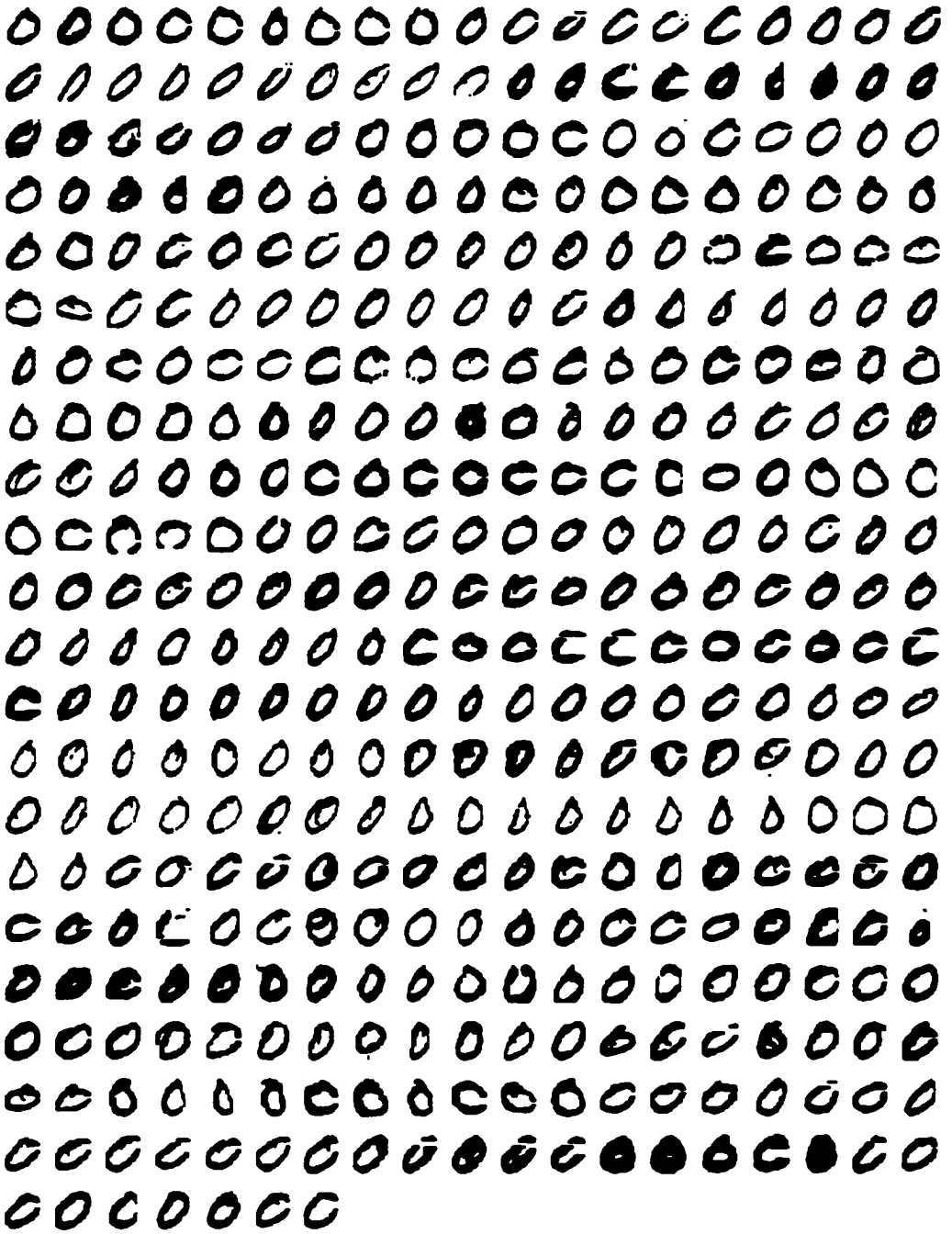


Figure D.10: Digit 0 with 406 Images

LIST OF REFERENCES

[1] J. R. Anderson and G. H. Bower, Human Associate Memory. Washington, DC: V. H. Winston and Sons, 1973.

[2] John A. Barden, "The Right of Free Association: Relative-Position Encoding for Connectionist Data Structures", Proc. of 10th Conference of Cognitive Science Society, Lawrence Erlbaum, Hillsdale, N. J., pp. 503-509, 1988.

[3] D. S. Banarse, A Generic Neural Network Architecture for Deformation Invariant Object Recognition, University of Wales, Bangor, UK, PhD Thesis, 1997.

[4] Thomas B. Berg and Howard Jay Siegel, "Instruction Execution Trade-Offs for SIMD vs. MIMD vs. Mixed Mode Parallelism", Proceeding of The 5th International Parallel Processing Symposium, pp. 301-308, 1991.

[5] Ira B. Black, Information in The Brain: A Molecular Perspective, Cambridge, MA: The MIT Press, 1994.

[6] Douglas S. Blank, "Differences between Representational Schemes in Connectionism and Classical Artificial Intelligence; or Why a Rose by a Symbolic Name Might not Smell as Sweet", In J. Dinsmore and T. Koschmann, (Eds.), Proceedings of the Second Midwest Artificial Intelligence and Cognitive Science Society, Huntsville, AL: Intergraph Corporation, pp. 8-14, 1990.

[7] Douglas S. Blank, Lisa A. Meeden and James B. Marshall, "Exploring the Symbolic/Subsymbolic Continuum: A Case Study of RAAM", In John Dinsmore (Ed.), The Symbolic and Connectionist Paradigms: Closing the Gap. Lawrence Erlbaum Associates, Publishers, pp. 113-148, 1992.

[8] Ronald F. DeMara and Dan Moldovan, "The SNAP-1 Parallel AI Prototype", IEEE Transactions on Parallel and Distributed Systems, Vol. 4, No. 8, August 1993.

[8] Ronald F. DeMara and Dan Moldovan, "The SNAP-1 Parallel AI Prototype" , IEEE Transactions on Parallel and Distributed Systems, Vol. 4, No. 8 , August 1993.

[9] Mark Derthick, Mundane Reasoning by Parallel Constraint Satisfication. Pitman, London, Morgan Kaufman Publishers, Inc., San Mateo, Ca. 1990.

[10] Mark Derthick, A Connectionist Architecture for Representing and Reasoning about Structured Knowledge, The 9th Annual Conference of The Cognitive Science Society, pp. 131-164.

[11] John E. Dowling, Neurons and Networks: An Introduction to Neuroscience, Cambridge, MA: The Belknap Press of Harvard University Press, 1992.

[12] J. Doyle, "A Truth Maintenance System", *Artificial Intelligence*, 12(3), 1979.

[13] Rick Evertsz, "The Automated Analysis of Rule-Based Systems, Based on their Procedural Semantics", Proceeding of The Joint International Conference on Artificial Intelligent (JICAI), 1991.

[14] Matthew P. Evett, William A. Anderson and James A. Hendler, "Massively Parallel Support for Efficient Knowledge Representation", Proceedings of The 13th International Joint Conference on Artificial Intelligence, pp. 1325-1330, 1993.

[15] Burden Faires, Numerical Analysis, 5th Ed. PWS-KENT.

[16] J. A. Fodor and Z. Pylyshyn, "Connectionism and Cognitive Architecture: A Critical Analysis", Cognition, 28, pp. 3-71, 1988.

[17] Wulfram Gerstner, "Associative Memory in a Network of Biological Neurons", In: Advances in Neural Information Processing Systems 3 . Lippmann RP, Moody JE, and Touretzky DS (Eds.), Morgan Kaufmann Publishers, San Mateo, pp. 84-90

[18] Carl W. Helstrom, Probability and Stochastic Processes for Engineers, Macmillan, 2nd Edition, 1991.

[19] James A. Hendler, "Marker-passing over Microfeatures: Towards a Hybrid Symbolic/Connectionist Model", Cognitive Science, 13, pp. 79-106, 1989.

[20] William Tsun-Yuk Hsu and Pen-Chung Yew, "An Effective Synchronization Network for Large Multiprocessor Systems", Proceeding of The 5th International Parallel Processing Symposium, pp. 309-317, 1991.

[21] Hiroaki Kitano, Hideto Tomabechi, Teruko Mitamura, and Hitoshi Iida "A Massively Parallel Model of Speech-to-Speech Dialog Translation: A Step Toward Interpreting Telephony", The DmDialog Project: Collection of Papers in 1989.

[22] J. de Kleer, "An Assumption-Based TMS", *Artificial Intelligence*, 28(2), pp. 127-162, 1986.

[23] Peter M. Kogge, The Architecture of Symbolic Computers, McGraw Hill, 1991.

[24] Benjamin Kuipers, Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge, The MIT Press, 1994.

[25] Steve Kuo and Dan Moldovan, "Performance Comparison of Models for Multiple Rule Firing". Proceeding of The Joint International Conference on Artificial Intelligent (JICAI), 1991.

[26] Stan C. Kwasny and Kanaan A. Faisal, "Symbolic Parsing Via Subsymbolic Rules", in John Dinsmore (Ed.), The Symbolic and Connectionist Paradigms: Closing the Gap. Lawrence Erlbaum Associates, Publishers, pp. 209-235, 1992.

[27] Trent E. Lange and Michael G. Dyer, "High-Level Inferencing in a Connectionist Network", Connection Science, 1(2), pp. 181-217, 1989.

[28] Trent E. Lange, "Hybrid Connectionist Models: Temporary Bridges Over the Gap Between the Symbolic and the Subsymbolic", in John Dinsmore (Ed.), The Symbolic and Connectionist Paradigms: Closing the Gap, Lawrence Erlbaum Associates, Publishers, pp. 237-289, 1992.

[29] Yousuf C. H. Ma, "A Survey of The Affect of Noise on Several Selected Artificial Neural Networks", Master Thesis, University of Central Florida, 1991.

[30] Yousuf C. H. Ma and Ronald F. DeMara, "Localized Self-Contained Adaptive Networks for Hybrid-Symbolic Reasoning", Forth Joint Conference on Information Sciences, Vol.3, pp. 81-86, October 23-28, 1998.

[31] John Malpas, Prolog: A Relational Language and Its Applications, Prentice-Hall, Inc., 1987.

[32] Timothy Masters, Signal and Image Processing with Neural Networks, John Wiley and Sons, Inc., New York., 1994.

[30] Yousuf C. H. M a and R onald F. DeMara. "Localized Self-Contained A daptive Networks for Hybrid-Symbolic Reasoning" , Forth Jo in t Conference on Inform ation Sciences, Vol.3, pp. 81-86, O ctober 23-28, 1998.

[33] D. A. McAllester, An Outlook on Truth Maintenance, Tech. Rep. AI Memo 551, MIT Artificial Intelligence Lab, Cambridge, MA., 1980.

[34] Dan Moldovan, Wing Lee and Changhwa Lin, "SNAP: A Marker-Propagation Architecture for Knowledge Processing", IEEE Transactions on Parallel and Distributed Systems, Vol.3, No.4, July 1992.

[35] Simon A. Newell, H. A. and J. C. Shaw, "Empirical Explorations with the Logic Theory Machine: A Case Study in Heuristics". In E. A. Feigenbaum and J. Feldman (Eds.), Computers and Thought. New York: McGraw-Hill, 1963.

[36] NIST database fl3, file://sequoyah.ncsl.nist.gov/pub/databases/data/fl3.tar.Z, 1997.

[37] D. A. Norman and D. E. Rumelhart, Explorations in Cognition. San Francisco, CA: Freeman, 1975.

[38] David W. Opitz and Jude W. Shavlik, "Heuristically Expanding Knowledge-Based Neural Networks", Proceedings of The 13th International Joint Conference on Artificial Intelligence, pp. 1325-1330, 1993.

[39] Kamran Parsaye and Mark Chignell, Expert Systems for Experts, John Wiley & Sons, Inc., 1988.

[40] Judea Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann, 1988.

[41] J. B. Pollack, "Recursive Auto-Associative Memory: Devising Compositional Distributed Representations". In Proceedings of the 10th Annual Conference of the Cognitive Science Society, Hillsdale, NJ: Lawrence Erlbaum Associates, pp. 33-39, 1988.

[42] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, Numerical Recipes in C: The Art of Scientific Computing, Second Edition, Cambridge University Press, 1992.

[43] M. R. Quillian, "Semantic Memory". in M. Minsky (Ed.), Semantic Information Processing. Cambridge, MA: MIT Press, pp. 227-270, 1968.

[44] Elaine Rich and Kevin Knight, Artificial Intelligence, 2nd ed. McGraw Hill, 1991.

[45] Steve G. Romaniuk and Lawrence O. Hall. "FUZZNET: Towards a Fuzzy Connectionist Expert System Development Tool", in Proc. IJCNN, pp 483-486, 1990.

[46] Steve G. Romaniuk and Lawrence O. Hall, "Injecting Symbol Processing Into a Connectionist Model", in Branko Soucek (Ed.), Neural and Intelligent Systems Integration for Fifth and Sixth Generation Integrated Reasoning Information Systems. John Wiley & Sons, Inc., 1991.

[47] David E. Rumelhart and James L. McClelland, (Eds.) Parallel distributed processing: Explorations in the microstructure of cognition. Vol. 1, The MIT Press, 1989.

[48] David E. Rumelhart and D. Zipser, "Feature Discovery by Competitive Learning", in David E. Rumelhart and James L. McClelland, (Eds.) Parallel distributed processing: Explorations in the microstructure of cognition, Vol. 1, The MIT Press, 1989.

[49] Lokendra Shastri, "A Connectionist Approach to Knowledge Representation and Limited Inference", Cognitive Science, 12, pp. 331-392, 1988.

[50] Lokendra Shastri and Dean J. Grannes, "Dealing With Negated Knowledge and Inconsistency in a Neurally Motivated Model of Memory and Reflexive Reasoning", ICSI TR-95-041, 1995.

[51] Lokendra Shastri and Dean J. Grannes, "A Connectionist Treatment of Negation and Inconsistency", The Proceedings of the Eighteenth Conference of the Cognitive Science Society, 1996.

[52] Lokendra Shastri, "Temporal Synchrony, Dynamic Bindings, and SHRUTI - a representational but non-classical model of reflexive reasoning", Behavioral and Brain Sciences Vol. 19, No.2, 331-337, 1996.

[53] R. M. Stallman and G. J. Sussman, "Forward reasoning and dependency-directed back-tracking in a system for computer-aided circuit analysis, Artificial Intelligence, 9(2), 1977.

[54] Ronald A. Sumida, Michael G. Dyer, and Margot Flowers, "Integrating Marker Passing and Connectionism for Handling Conceptual and Structural Ambiguities", The 10th Annual Conference of the Cognitive Science Society, pp. 146-152, 1988.

[55] Ron Sun, "A Discrete Neural Network Model for Conceptual Representation and Reasoning", Proc. of 11th Cognitive Science Society Conference, 1989.

[56] Ron Sun, "Rules and Connectionism", Proc. of International Neural Network Conference, Paris, 1990.

[57] Ron Sun and David Waltz, "A Neurally Inspired Massively Parallel Model of Rule-Based Reasoning", in Branko Soucek (Ed.), Neural and Intelligent Systems Integration for Fifth and Sixth Generation Integrated Reasoning Information Systems. John Wiley & Sons, Inc., 1991.

[58] Ron Sun, Integrating Rules and Connectionism for Robust Commonsense Reasoning, John Wiley & Sons, Inc., 1994.

[59] Akikazu Takeuchi, Parallel Logic Programming, John Wiley & Sons, Inc., 1992.

[60] David S. Touretzky and Geoffrey E. Hinton, "Symbols among Neurons: Details of a Connectionist Inference Architecture", Proc. of 9th IJCAI, pp. 238-243, Los Angeles, Calif., August 1985.

[61] David S. Touretzky and Geoffrey E. Hinton, "A distributed connectionist production system", Cognitive Science 12(3), pp. 423-466, 1988.

[62] David S. Touretzky and Shai Geva, "A distributed connectionist representation for concept structures", Proceedings of the 9th Annual Conference of the Cognitive Science Society, pp. 155-164, 1987.

[63] David S. Touretzky, "BoltzCONS: Dynamic Symbol Structures in a Connectionist Network", Artificial Intelligence, 46, pp. 5-46, 1990.