THE FLORIDA STATE UNIVERSITY
COLLEGE OF ARTS AND SCIENCES

HIERARCHICAL ART-BASED CLASSIFICATION NETWORKS

By
CATHERINE LEBLANC

A Thesis submitted to the
Department of Computer Science
in partial fulfillment of the
requirements for the degree of
Master of Science

Degree Awarded:
Spring Semester, 1995

The members of the Committee approve the thesis of Catherine LeBlanc defended on February 3, 1995.

S. I. Hruska
Professor Directing Thesis

R. C. Lacher
Committee Member

C. Kacmar
Committee Member

Approved:
R.C. Lacher, Chair, Department of Computer Science

Acknowledgements

I extend my thanks to Dr. Charles Katholi of the University of Alabama at Birmingham for lending his expertise to the problem of evaluation of a set of clusters and for answering other questions of statistics that I had. I also thank Dr. Thomas Unnasch of the University of Alabama at Birmingham for the original problem and for the use of the data.

Thanks to my committee, Dr. Kacmar and Dr. Lacher, for their careful reading of this thesis and useful suggestions.

I especially thank my major professor, Dr. Susan Hruska. Her excellent explanations, her humor and mostly, her listening skills were instrumental in keeping me sanely on the path to finishing this thesis.

My family has not seen me much in the past year and I thank them for being so patient and understanding. Finally, thanks to Evelyn Stiller, who has provided objective feedback, many home-cooked meals, many needed diversions and lots of laughter as well as tears over the course of the past two and half years.

Contents

List of Tables viii

List of Figures ix

Abstract x

1 Introduction 1

2 Adaptive Resonance Theory 3

2.1 Overview 3

2.2 Longterm Memory Updates 9

2.3 Vigilance Parameter 11

2.4 Design Principles 12

2.4.1 Stability-Plasticity Tradeoff 13

2.4.2 Search-Direct Access Tradeoff 14

2.4.3 Match-Reset Tradeoff 15

2.4.4 Self-scaling and Critical Features 15

3 Clustering Algorithms 17

3.1 The Leader Algorithm 17

3.2 The Modified Leader Algorithm 19

3.3 The K-means Algorithm 20

3.4 Distance Metrics 22
3.4.1 Euclidean Distance 23
3.4.2 Normalized Inner Dot Product 24

4 Hierarchical Clustering using HABclass Networks 25

4.1 Overview 25
4.1.1 *Onchocerca volvulus* 26
4.1.2 Data Representation 28
4.1.3 The Algorithm 29
4.2 Evaluation of the Set of Categories 30
4.2.1 Evaluation and Singleton Clusters 33
4.3 Choosing Vigilance Levels 34
4.4 Requirements of HABclass Networks 35
4.4.1 Aligned Sequences 35
4.4.2 Numeric Representation for Data 35
4.4.3 ART Parameter Tuning 36
4.5 Advantages of HABclass Networks 36
4.5.1 Unsupervised Learning 36
4.5.2 Easy Adaptability 37
4.5.3 Singleton Clusters 37

5 HABclass Clustering Results 39

5.1 Evaluation of One-Layer HABclass Nets 39
5.2 Two-Layer HABclass Nets 42

6 Comparisons of Algorithms **48**

6.1 HABclass Networks and the Leader Algorithm 48

6.2 HABclass Networks and the K-means Algorithm 53

6.3 Comparison Conclusions 58

7 Future Work **60**

8 Conclusions **64**

Appendices **69**

A *O. volvulus* DNA Sequences **69**

B Description of the *O. volvulus* Data **87**

C Changes Made to the ART2 Software **88**

C.1 Changes to **art2.c** 88

C.1.1 Changes to function **do_cycle** 88

C.1.2 Changes to function **BU_reset** 89

C.1.3 Changes to function **TD_reset** 90

C.1.4 Changes to function **update_Zs** 92

C.1.5 Changes to function **print_weights** 93

C.2 Changes to **in_vec.c** 95

C.2.1 Changes to function **get_inputs** 95

C.2.2 Changes to function **select_cycle** 96

C.3 Changes to **in_vec.h** 96

D Other Software used in the *O. volvulus* Application **97**

D.1 **project.c** 97

E Instructions for using Additional Software **101**

E.1 Necessary Software Files 101

E.2 Clustering Using HABclass Networks 102

E.3 Clustering Using Alternative Algorithms 105

Bibliography **107**

Vita **110**

List of Tables

5.1 Effect of changing vigilance level 46

7.1 Effect of presentation order (vigilance = 0.98) 62

List of Figures

2.1 ART2 Architecture 4

2.2 ART2 F1 Layer Node 5

2.3 Critical Feature Detection 16

4.1 HABclass Network 27

5.1 ART2 Clustering of Guatemala and Brazil Data - Vigilance = .98 40

5.2 ART2 Clustering of Guatemala and Brazil Data - Vigilance = .99 41

5.3 Sample Run of HABclass using Only Brazil and Guatemala Data 43

5.4 Sample Run of HABclass using Unlabeled Data 45

6.1 Simple Leader Algorithm with Vector Distance = .94 49

6.2 Leader Algorithm with Euclidean Distance Threshold = 4 . . . 50

6.3 Modified Leader Algorithm with Threshold = 4 51

6.4 Modified Leader Algorithm Classification 52

6.5 K-means Clustering of Brazil and Guatemala Data - 6 Clusters . 54

6.6 K-Means Classification of Unlabeled Data - 6 Clusters 55

6.7 K-Means Clustering of Brazil and Guatemala Data - 22 Clusters 56

6.8 K-Means Classification of Unlabeled Data - 22 Clusters 57

Abstract

Adaptive resonance theory (ART) describes a class of artificial neural network architectures that act as classification tools which self-organize, work in real-time, and require no retraining to classify novel sequences. This thesis describes Hierarchical ART-Based Classification Networks (HABclass networks), adapted from ART networks to provide support to scientists attempting to categorize DNA fragments from *Onchocerca volvulus*. In the HABclass network approach, sequence data are presented to multiple ART-based networks which are linked together into two (or more) tiers; the first tier provides coarse sequence classification while the subsequent tiers refine the classifications as needed. The overall rating of the resulting set of categories is measured using statistical techniques based on those introduced by Zimmerman, *et al.* (1994) to validate results from traditional phylogenetic analysis. New sequences can then be presented to the HABclass network for placement into these categories or into newly-created categories, if necessary. Tests of the Hierarchical ART-based Classification Network indicate its value as a fast, easy-to-use classification tool which adapts to new data without retraining on previously classified data.

CHAPTER 1
Introduction

Classification is an activity that has occupied scientists for centuries. The activity of classification has been useful in a variety of disciplines, from astronomy to zoology [20]. The very act of naming an object requires that the object be classified in some manner. From a scientific standpoint, classification generally implies that there are patterns whose class memberships are known a priori and will be used for creation of the initial classes. In other words, classification activities use supervision. When, on the other hand, such labelled patterns are not available, the activity of creating classes is generally called clustering, since patterns will be clustered together into groups based on the characteristics of the pattern data itself. Clustering, then, uses no supervision [8]. Because *a priori* knowledge concerning the pattern data may indeed be difficult to obtain, clustering is a more general activity than classification, although there is no rigid distinction between the two activities.

Many techniques have been developed to cluster objects. This thesis will introduce a technique called Hierarchical ART-based Classification Networks (HABclass Nets), based on adaptive resonance theory (ART) artificial neural network architectures developed by Carpenter and Grossberg [4] [3] [2]. Adaptive resonance theory attempts to use knowledge of the functioning of the human brain to cluster arbitrary pattern data that is input to the network. The development of HABclass Nets was prompted by the fact that ART networks do not

allow hierarchical classifications. The original motivation for the development of HABclass Nets came from applying the ART network clustering technique on a problem from molecular biology described in [22] and [15]. This problem involves the clustering of DNA sequences taken from the human parasite *Onchocerca volvulus*, which causes river blindness in humans. Once a set of clusters is developed, it might be useful to break one or more of the clusters into subclusters, thus, creating a hierarchical clustering of the DNA sequences. Our collaborators on the project have suggested numerous other situations where HABclass Nets might be an appropriate clustering tool.

One of the reasons that so many clustering algorithms have been developed is that each algorithm has its own strengths and weaknesses. Often, the most appropriate clustering algorithm to use depends entirely upon the nature of the data to be clustered.

In this thesis, HABclass networks will be compared to a number of other clustering algorithms with particular emphasis on the appropriateness of each technique for the DNA sequence classification problem. Finding the goodness of a set of clusters continues to be an open problem. This thesis will describe and illustrate a method for determining the goodness of a set of clusters resulting from applying the HABclass network technique to the DNA sequence problem [22] [15].

CHAPTER 2
Adaptive Resonance Theory

2.1 Overview

Adaptive resonance theory (ART) [4] [3] [6] [2] describes a class of artificial neural network architectures that use competitive, unsupervised learning to classify patterns. ART networks work in real-time and their developers have made great strides toward solving the stability-plasticity dilemma, that is, how the network can be plastic enough to learn significant new information yet remain stable enough to prevent new information from washing away previous learning [11].

Carpenter and Grossberg's first attempt at creating artificial neural networks that model the human brain was a class of architectures called ART1 [4]. These networks accepted only binary input. Since an artificial model of the brain must accept analog input, Carpenter and Grossberg next developed ART2 neural networks [3]. The discussions in this thesis will focus on ART2 and all experiments described later use a modified version of the ART2 software available via anonymous ftp from Boston University.

An ART network contains two layers, F1 and F2, of computational units. The F1 layer accepts input patterns and is called the feature representation field while the F2 layer will represent the categories learned by the network and is called the category representation field. The network is fully connected. Each F1 node A_i $(i = 0 \ldots m)$ is connected to every F2 node B_j $(j = 0 \ldots n)$ by

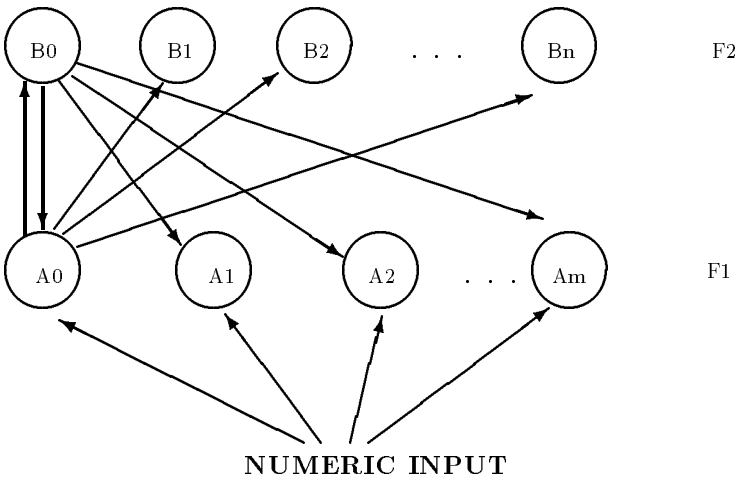

Figure 2.1: ART2 Architecture

means of a bottom-up weight or long term memory (LTM) trace, z_{ij}. Each F2 node B_j is connected to every F1 node A_i by means of a top-down weight or LTM trace, z_{ji}. Figure 2.1 shows the architecture of an ART2 network although, for the sake of simplicity, not all LTM traces are shown. The bottom-up LTM traces encode the input patterns while the top-down LTM traces encode learned expectations [3].

Since ART2 architectures are designed for the processing of analog, as well as binary, input patterns, the system needs to be able to pick out and enhance signals that are embedded in noisy data. In order for this to happen, each node of the feature representation field must include several processing layers, shown in Figure 2.2 below. I_i is the raw input to the network, that is, the i^{th} element of the input vector, where $1 \leq i \leq n$ and n is the total number of elements in the input vector.

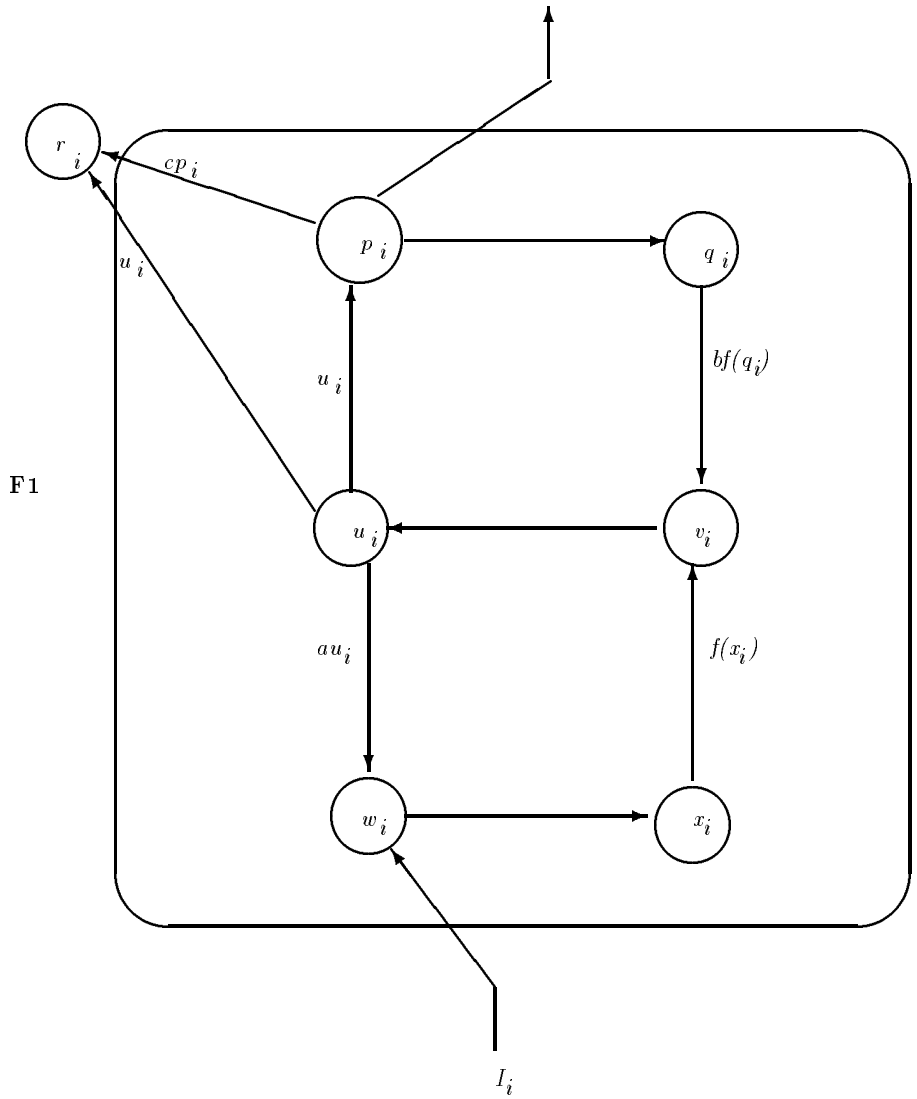

Figure 2.2: ART2 F1 Layer Node

The following formulas characterize the activity, called short term memory (STM), of the i^{th} F1 node:

$$p_i = u_i + \sum_j g(y_j) z_{ji} \tag{2.1}$$

$$q_i = \frac{p_i}{e + \| \mathbf{p} \|} \tag{2.2}$$

$$u_i = \frac{v_i}{e + \| \mathbf{v} \|} \tag{2.3}$$

$$v_i = f(x_i) + bf(q_i) \tag{2.4}$$

$$w_i = I_i + au_i \tag{2.5}$$

$$x_i = \frac{w_i}{e + \| \mathbf{w} \|} \tag{2.6}$$

where $\| \mathbf{V} \|$ denotes the norm of vector $\mathbf{V}$, y_j is the potential or STM activity of the j^{th} F2 node, z_{ji} is the LTM trace from the j^{th} F2 node to the i^{th} F1 node. The function f serves to suppress noise in a pattern and is typically of the form

$$f(x) = \begin{cases} 0 & \text{if } 0 \leq x \leq \theta \\ x & \text{if } x > \theta \end{cases} \tag{2.7}$$

The function g serves to gate the potential of the F2 nodes before they reach F1. This gating is a form of competition at the F2 layer. When a pattern is presented to the network at the feature representation field, F1, we want it to be placed into some category at the category representation field, F2. This category will be the one that receives the greatest total input from F1. That is, let T_j be the summed input from F1 to the j^{th} F2 node

$$T_j = \sum_i p_i z_{ij}. \tag{2.8}$$

The F2 node J that is maximally active is then chosen as the potential category for this pattern. The ART network will check to see whether the category represented by the F2 node J is close enough to the input pattern. "Close enough" will be represented by a vigilance parameter, ρ. If it is not close enough, this F2 node will be reset and the system will continue its search for the correct category. The main elements of the gating dynamics are represented by the function g

$$g(y_j) = \begin{cases} d & \text{if } T_J = \max\,(T_j| \text{ the } j^{th} \text{ F2 node has not} \\ & \text{been reset on the current trial)} \\ 0 & \text{otherwise} \end{cases} \tag{2.9}$$

The equations above use constants a,b,d, and e. In all cases that Carpenter and Grossberg have described in their various papers, the constant $e = 0$. The other constants must be chosen very carefully. The constant a is the internal F1 layer top-down feedback parameter and the constant d is the $F2 \rightarrow F1$ top-down feedback parameter. When either a or d is small, the importance of the $F2 \rightarrow F1$ expectation feedback weakens relative to the bottom-up input and coding of input patterns can no longer be guaranteed to be stable. In other words, at one time, an input pattern may be put into one category while during the next cycle, that same pattern may be put into a different category and during the next cycle, the pattern may be put back into the first category. The constant b is the $F1 \rightarrow F2$ bottom-up feedback parameter. If b is chosen to be small, unstable coding of input parameters may again occur because previous LTM patterns may be washed away by subsequent input patterns [3].

What has been described thus far constitutes the *attentional subsystem* of the ART2 architecture. The other subsystem, the *orienting subsystem* has been hinted at in the discussion of the vigilance parameter, ρ. The orienting subsystem allows the degree of match between an input pattern and a learned expectation to be computed. This degree of matching is determined by the vector $r = (r_1, \ldots, r_m)$ where

$$r_i = \frac{u_i + cp_i}{e + \| \mathbf{u} \| + \| c\mathbf{p} \|}. \tag{2.10}$$

The orienting subsystem will determine that a mismatch has occurred between the input pattern and the chosen category if

$$\frac{\rho}{e + \| \mathbf{r} \|} > 1 \tag{2.11}$$

where the vigilance parameter ρ is set between 0 and 1 [3]. The primary function of the orienting subsystem is to protect a category from being recoded by patterns which do not belong to it [6]. The parameter, c, in Equation 2.10, is used in the orienting subsystem to help ensure that learning of patterns will increase the network's sensitivity to mismatches between the input pattern and the previously learned expectation of the category node. In particular, c is constrained by the following equation:

$$\frac{cd}{1-d} \leq 1. \tag{2.12}$$

The closer $\frac{cd}{1-d}$ is chosen to 1, the more sensitive the sytem will be to mismatches, assuming all other things are equal [3].

When presented with a pattern to be classified, the network will search the category representation field, F2, for a potentially matching category. The

network will measure the similarity between the input pattern and the previously learned expectation of the category node. If the similarity is good enough, that is greater than ρ, the learned expectation will be changed to incorporate the input pattern. If the similarity is not good enough, F2 is searched for a better match [3].

There will be one F2 layer node for each possible category in the application. Each input pattern is encoded in exactly one category in the network. Because competitive learning is used, generic ART networks are not applicable for use in creating hierarchical categorization, that is, for forming subcategories within categories [2].

2.2 Longterm Memory Updates

Between the two layers, F1 and F2, of an ART2 network are two filters, a bottom-up filter and a top-down filter. Each of these filters consists of a set of weights or longterm memory (LTM) traces. The bottom-up LTM traces encode the input pattern while the top-down LTM traces encode the expectation, or template, learned at a particular F2 node. Once the system has achieved *resonance*, that is, once the system has stabilized to a category (F2 layer node) for this input, the input pattern may be encoded in LTM traces between F1 and F2. The LTM trace update equations are

$$\text{Top-down}(F2 \rightarrow F1) : \frac{dz_{ji}}{dt} = g(y_j)(p_i - z_{ji}) \tag{2.13}$$

$$\text{Bottom-up}(F1 \rightarrow F2) : \frac{dz_{ij}}{dt} = g(y_j)(p_i - z_{ij}) \tag{2.14}$$

Equations 2.9, 2.13, and 2.14 imply that when F2 makes a choice of node J,

$$\frac{dz_{Ji}}{dt} = d(p_i - z_{Ji}) \tag{2.15}$$

and

$$\frac{dz_{iJ}}{dt} = d(p_i - z_{iJ}) \tag{2.16}$$

In addition, for all $j \neq J$, $\frac{dz_{ji}}{dt} = 0$ and $\frac{dz_{ij}}{dt} = 0$ [3].

Recovering the templates, or expectations, learned at the F2 node would be useful in a number of applications. Intuitively, these templates correspond to a set of *critical features* for that particular category. That is, in order to belong to this category, a pattern must have this set of critical features. Using the notion of an *index set*, Ω_J, of critical features for a category J, Carpenter and Grossberg describe the nature of these critical features. When an F2 node, J, has been chosen to represent a particular pattern, the index set for J is

$$i \epsilon \Omega_J \text{ iff } z_{Ji}^{(old)} > \frac{\theta}{1-d}, \tag{2.17}$$

where $\mathbf{z}_J^{(old)}$ is the top-down LTM vector at the beginning of the presentation of the input pattern. In other words, the algorithm will basically ignore those places in the template where $z_{Ji}^{(old)} < \frac{\theta}{1-d}$ [6].

Initially, the top-down LTM traces need to be chosen small so that when an uncommitted node is chosen to encode a pattern, a reset does not occur immediately. The bottom-up weights should also be small so that an input that first chose an uncommitted node cannot switch to other uncommitted nodes in the middle of a learning trial [3].

2.3 Vigilance Parameter

The vigilance parameter is used in orienting subsystem to determine when an F2 category node, J, should be reset. Such a reset will occur whenever the difference between the current learned expectation for J and the potential learned expectation (if this pattern were to be coded in category J) for J is too large. That is, by equation 2.11 and the fact that the constant $e = 0$, reset of node J will occur iff

$$\| \mathbf{r} \| < \rho. \tag{2.18}$$

In fact, Carpenter and Grossberg have shown that reset will occur iff

$$cos(\mathbf{I}, \mathbf{z}_J) < \rho^*, \tag{2.19}$$

where cos($\mathbf{x}$,$\mathbf{y}$) is the cosine of the angle between vector $\mathbf{x}$ and vector $\mathbf{y}$, $\mathbf{I}$ is the input vector, $\mathbf{z}_J$ is the learned expectation of F2 node J,

$$\rho^* = \frac{\rho^2(1+\sigma)^2 - (1+\sigma^2)}{2\sigma} \tag{2.20}$$

and

$$\sigma = \frac{cd}{1-d}. \tag{2.21}$$

Note that $\rho^* = 1$ iff $\rho = 1$ [6]. Because the vigilance parameter is being measured against the cosine of the angle between the two vectors, ρ must be greater than $\| \mathbf{r} \|$ for reset to occur. That is, the cosine of the angle between two identical vectors is 1 and so if this cosine is greater than the vigilance parameter, the two vectors are related to each other closely enough to put them in the same category.

There is some tuning of the vigilance parameter that must occur in order for the clustering to be useful. If the vigilance parameter is set to be very high, each pattern will be placed into its own cluster. If the vigilance parameter is set to be too low, all patterns will be placed into one large cluster. Clearly, neither of these extremes is likely to be very useful in a real clustering application. Thus, the user of an ART2 network must experiment with the data to determine an appropriate vigilance level for that data.

In the problem for which HABclass Nets were designed, the *Onchocerca volvulus* DNA sequences are very highly conserved. Thus, the vigilance levels used tend to be relatively high, usually in the .97-.99 range. Each application domain is sure to have its own optimal vigilance range. The user should experiment with the data to determine this range. A methodology for determining the vigilance range is a subject of ongoing research.

2.4 Design Principles

Adaptive resonance architectures are neural networks that use unsupervised learning to develop stable pattern recognition codes in response to arbitrary sequences of input patterns. ART2 is a class of adaptive resonance architectures which will respond rapidly to input patterns which are either analog or binary. The design principles that are solved in a variety of ways include the stability-plasticity tradeoff, the search-direct access tradeoff, the match-reset tradeoff and the self-scaling/critical features principle [11].

2.4.1 Stability-Plasticity Tradeoff

The fundamental stability-plasticity dilemma is that the system must be able to learn important new events but not at the expense of forgetting what it has already learned. Most researchers have dealt with this problem by either ignoring it or by providing an external force which determines when the plasticity should be shut off. Ignoring the problem is certainly no solution. Relying on an external force may result in plasticity being turned off too soon, in which case important information may never be learned, or too late, in which case, earlier learning begins to be washed away by subsequent learning. Other researchers have solved this problem by externally restricting their learning environment. This approach amounts to ignoring the problem in the design of the network and certainly leads to reduced network usefulness [11]. Clearly, neither of the above approaches is ideal and a useful neural network technique should attempt to solve this dilemma.

An artificial neural network technique that suffers from a lack of stability when confronted with new data is back-propagation [14]. When a back-propagation network that has been trained on a set of data is presented with a completely new set of data, eventually the new data will wash away any memory the network has of the old data. One way that researchers using back-propagation networks get around this problem is to turn off learning once the network has been trained adequately on the old data. Once this happens, the network is no longer plastic enough to learn new information that might be present in the new data. Such a solution requires a human to intervene in the learning process. The human may intervene too early in the process so that not

all the present information is learned or the human may intervene too late in the process so that the previously learned information has begun to be washed away. In any case, new information that is present in the data can no longer be learned once the learning process is turned off.

Carpenter and Grossberg have solved the dilemma in ART, which internally determines when to switch from its plastic mode to its stable mode [1]. An ART network will refine its already learned codes when new information warrants it by simply updating LTM traces. This is the network's plastic mode. If, however, a new pattern does not fall close enough to a previously learned category, the network will not place the new input there. This is the network's stable mode. Equation 2.11 determines when to switch from plastic to stable mode.

2.4.2 Search-Direct Access Tradeoff

Many classification tools involve some sort of searching. The tool must search the current categories for a match to the current input. In many of these search models, the search time grows as the learned category representation becomes arbitrarily more complex. ART2 carries out a search while learning occurs in that, as F2 categories are chosen, a match test will occur. If the match is very far off, the system will search for another appropriate category. Once a pattern becomes familiar, however, the system will choose the correct category without resets. The search time will then decrease as the pattern becomes familiar no matter how complex the learned category code has become. Unfamiliar patterns will create new categories [4].

2.4.3 Match-Reset Tradeoff

A classification tool must walk a fine line between the requirements of sensitive matching and the formation of new codes. Therefore, the system should be able to recognize and react to small differences between the input pattern and the LTM pattern encoded in the top-down LTM traces. It should also be able to choose an uncommitted F2 node without causing a reset to occur even though no match can occur between the input pattern and a category template (because there is not yet a category template). ART2 architectures are able to meet this principle as long as the top-down LTM traces are chosen appropriately before the training begins. In other words, when an uncommitted F2 node is chosen to represent a pattern, the top-down LTM traces must initially have been chosen to be equal to zero so that no reset will occur [3].

2.4.4 Self-scaling and Critical Features

One of the most important design principles of ART is that the system must be able to learn in noisy environments. This means that the system must be able to discover invariant, critical features in a pattern while treating the rest of the pattern as noise. In addition, the system must be able to self-scale, that is, in one context, a particular feature must be treated as noise, while in another context, that same feature must be treated as signal. ART2 has this self-scaling property. The vigilance parameter, ρ, and the learning algorithm both help to achieve this property [11]. In some of the literature, Carpenter and Grossberg call this feature contrast-enhancement [3].

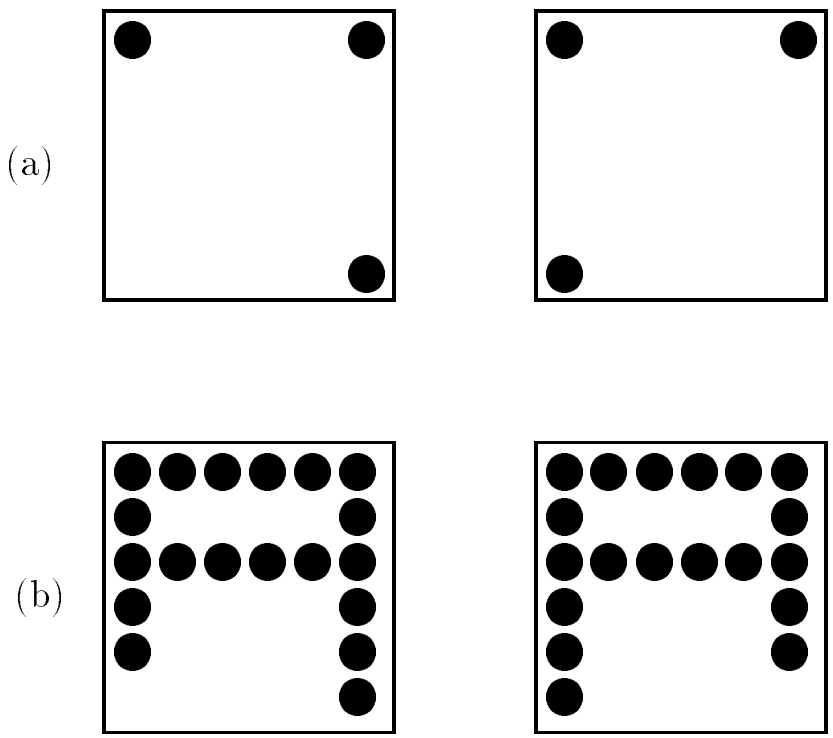

Figure 2.3: Critical Feature Detection

The self-scaling property, which discovers critical features in a context sensitive manner, is illustrated in Figure 2.3. In Figure 2.3 (a), the two patterns mismatch at one feature and this mismatch is probably enough to cause the two patterns to organize into two separate categories (depending, of course, upon the vigilance level). In Figure 2.3 (b), the two patterns mismatch at the same feature as Figure 2.3 (a), but in this case, this mismatch should probably be treated as noise (again, depending on the vigilance level) so that the two patterns are organized into the same category [4] [11].

CHAPTER 3
Clustering Algorithms

There are many other algorithms available for clustering data points. Two such algorithms are the leader algorithm and the K-means algorithm. Each algorithm has its own strengths and weaknesses and is useful in certain situations while not so useful in other situations [7] [13] [19].

3.1 The Leader Algorithm

The leader algorithm is a very fast method for clustering data points. It requires one pass through the data to put each point in a particular cluster or group of data points. Associated with each cluster is a "leader", which is one pattern against which new patterns will be compared to determine whether the new pattern belongs to this particular cluster. Presentation order of the patterns is very important. For example, the first pattern presented will always be the leader of the first cluster. The second pattern will be compared to the leader of the first cluster. If the second pattern is close enough to this leader (as determined by a user-supplied threshold), the second pattern is placed in the first cluster. If the second pattern is not close enough to this leader (again, as determined by a user-supplied threshold), the second pattern will become the leader of the second cluster. The next pattern will then be compared to the leader of the first cluster, and if close enough, is placed in that cluster, and if not close enough, compared to the leader of the next cluster. The pattern will

either be placed into a cluster or, after having been compared to all existing cluster leaders and found to be not close enough to any of them, will become the leader of a new cluster. The next pattern goes through the same process and so on, until each pattern has been placed into some cluster [13].

The leader algorithm works as follows:

1. Begin with the first pattern. Classify this pattern into the first cluster (so the number of clusters, K, is 1). The leader of cluster 1 is, thus, the first pattern.

2. Begin working with the first cluster, $J = 1$.

3. For each remaining pattern, I, let $D(I, J)$ be the distance between pattern I and the leader of cluster J.

4. If $D(I, J)$ is less than the user-supplied threshold, then pattern I is assigned to cluster J. Return to step 3.

5. For each remaining cluster, let $D(I, J)$ be the distance between pattern I and the leader of cluster J. Return to step 4.

6. When all clusters have been examined for using pattern I and pattern I has not been placed in any of the clusters, a new cluster must be created. Increase K by 1 and the leader of the new cluster is pattern I. Return to step 3 [13].

This algorithm requires one pass through the data for every pattern to be placed into a cluster and is, thus, a very fast algorithm. On the other hand, the algorithm is not invariant to presentation order. For example, the first pattern presented will always be a cluster leader. In addition, the clusters that are created first will tend to be very large since a pattern will always be compared to them first and be placed in the first cluster to which it is close enough [13].

3.2 The Modified Leader Algorithm

The modified leader algorithm attempts to deal with one of the major problems of the leader algorithm described above. To determine into which cluster a new pattern will fall, the algorithm will calculate which of the cluster leaders is closest (according to some user-defined distance metric) to the pattern. If that closest cluster leader is close enough (as determined by the user-supplied threshold), then the pattern belongs to that cluster. If the cluster leader is not close enough (again, as determined by the user-supplied threshold), the pattern becomes the leader for a new cluster. In this way, each cluster has an equal chance at having the new pattern fit into it rather than clusters that are created earlier having an undue advantage. Of course, this algorithm is still not invariant to presentation order since the first pattern will again always be the leader of the first cluster. This algorithm is also slower than the simple leader algorithm since distances between the pattern and *every* cluster leader must be calculated before the pattern can be placed into any cluster.

The modified leader algorithm works as follows:

1. Begin with the first pattern. Classify this pattern into the first cluster (so the number of clusters, K, is 1). The leader of cluster 1 is, thus, the first pattern.
2. Begin working with the first cluster, $J = 1$.
3. For each remaining pattern, I, let $D(I, J)$ be the distance between pattern I and the leader of cluster J. Find the cluster for which $D(I, J)$ is smallest.
4. If $D(I, J)$ is less than the user-supplied threshold, then pattern I is assigned to cluster J. Return to step 3.
5. If $D(I, J)$ is greater than the user-supplied threshold, then a new cluster

must be created. Increase K by 1 and the leader of the new cluster is pattern I. Return to step 3 [13].

In both versions of the leader algorithm, the user must guess the correct threshold level and various threshold values may yield completely different results. It may not be completely clear which is an optimal threshold level and often measures of goodness of clusters require some sort of tradeoff to be made. The choice of this threshold is critical. A very large threshold will result in all patterns assigned to one cluster. A very small threshold will result in each pattern assigned to its own individual cluster. In order to find the natural clusters inherent in the data, the threshold must be larger than the typical within-cluster distances and smaller than the typical between-cluster distances [7].

3.3 The K-means Algorithm

In the K-means algorithm, the user specifies the number of clusters desired, K. The optimal clustering of the data into K clusters is determined. First, the patterns are put into an initial set of K clusters. There are a variety of ways to create the initial clustering. A set of clusters in which patterns are randomly placed in the clusters is one method. Another method is to put each of the first K patterns into a cluster and put all of the remaining patterns into the first cluster. The initial clustering seems to only have an effect on the length of time the algorithm will take to determine the optimal clustering. Once the initial clustering is created, the mean of each pattern element for each cluster is determined. The error of this clustering is then determined by subtracting each variable for each pattern in a cluster from the mean for that cluster, squaring it and adding it to the error measure. Once this measure has been determined

for the entire clustering, each pattern is examined to see whether moving it to a new cluster would reduce the error. If the error would decrease as a result of the move, the pattern is moved and the means and error are recalculated. If there is no cluster to which the pattern can be moved so that the error is decreased, the next pattern is examined. This process continues until no pattern can be moved to decrease the error for the clustering. The resulting set of clusters are optimal for that set of patterns [13].

The K-means algorithm works as follows:

1. Create an initial set of clusters for the p patterns using one of the methods described above. Compute the cluster means M_{LJ} for each cluster L and each each pattern element J. Compute the initial partition error

$$E = \sum_{i=1}^{p} D(i, M(L(i)))^2, \tag{3.1}$$

where p is the number of patterns, $D(x, y)$ is the Euclidean distance between x and y and $M(L(i))$ is the mean of the cluster that contains pattern i.

2. For the first pattern, for every cluster, compute the change in error if the pattern were to be moved from the cluster it is currently in to the new cluster. That is, compute

$$C_L = \frac{N(L)D(1, M(L))^2}{N(L)+1} - \frac{N(L(1))D(1, L(1))^2}{N(L(1))-1)}, \tag{3.2}$$

for all clusters $L \neq L(1)$ where $N(x)$ is the number of cases in cluster x, $L(y)$ is the cluster containing pattern y, and $D(x, y)$ is the Euclidean distance between x and y. If the minimum C_L is negative, transfer the first pattern from cluster $L(1)$ to the cluster L where C_L is minimum, adjust the cluster means of $L(1)$ and the new cluster L, and add C_L (which is negative) to the error E.

3. Repeat step 2 for all remaining patterns.

4. If no pattern is moved from one cluster to another, stop. Otherwise, repeat step 2 [13].

The major difficulty with the K-means algorithm is that the number of clusters must be guessed by the user. In addition, the algorithm will put all patterns into these K clusters so that if there is one outlier, that pattern will probably be put into a cluster by itself and the remaining patterns will be forced into K-1 clusters. Once the clusters have been created, if a new pattern is seen (one that was not part of the original set of patterns), either the clusters will have to be created with the original patterns plus the new pattern or the new pattern will have to be placed into one of the existing clusters, no matter how well it fits into that cluster. That is, if the new pattern belongs to a heretofore unseen cluster, the user would not be given this information. Thus, the K-means algorithm is not very adaptable to new data.

3.4 Distance Metrics

Since the goal of clustering is to find the natural groupings in a set of data, the choice of an appropriate distance metric is not a trivial task. The groupings may change depending upon the metric chosen. Ideally, the metric should be invariant to the kinds of transformations that might occur in measurement of the data at different times. There is a wide variety of ways to measure the distance between two patterns. Two of the most commonly used are Euclidean distance and the inner dot product [7]. Both measures require numeric data and view the patterns as vectors. Each is appropriate in certain situations but not in all. The beauty of both metrics lies in their simplicity.

At this point, it should be mentioned that the calculation of the distance between two vectors always involves combining values of their components. For some pattern recognition tasks, this may introduce problems into the task [7]. For example, if the vectors represent amino acid sequences, one component may be which protein family the sequence belongs to while another component may be a measure of the hydrophobicity of the sequence. There is no natural or logical way to combine these two components. This is one of the major difficulties in determining which distance metric to choose for clustering [7].

3.4.1 Euclidean Distance

The Euclidean distance between two n-length vectors $\mathbf{x}$ and $\mathbf{y}$ is

$$\sum_{i=1}^{n}(x_i - y_i)^2. \tag{3.3}$$

Clusters that have been created using Euclidean distance will be invariant to rotations or other rigid-body motions of the data but not invariant to linear transformations, that is, transformations that distort the distance relationships of the data points. If rescaling or other linear transformations are unnatural to the data, this limitation of the Euclidean distance metric is of no concern [7].

Often, to reduce the effects of rescaling of the data on the distance metric, the data can be normalized. For example, the axes for the data might be translated and scaled in order to obtain invariance to displacement and scale changes. These kinds of normalization may not be desirable, however. If the larger distances are due to simple scaling affects in the way that measurements have been taken, normalization may be entirely appropriate. If, on the other hand, these larger distances are due to the presence of subclasses within the

data, such normalization may obscure that fact and the normalization procedure should be avoided [7].

3.4.2 Normalized Inner Dot Product

The normalized inner dot product between two n-length vectors $\mathbf{x}$ and $\mathbf{y}$ is

$$\frac{\mathbf{x}^t\mathbf{y}}{\| \mathbf{x} \| \| \mathbf{y} \|}. \tag{3.4}$$

This measure, which is the cosine of the angle between $\mathbf{x}$ and $\mathbf{y}$, is invariant to rotation but not invariant to general linear transformations. This measure is more often called a *similarity* measure than a distance measure [7].

If the features that are stored in the vectors $\mathbf{x}$ and $\mathbf{y}$ are binary valued (that is, 0 or 1), this similarity measure represents the number of features that are shared by the two vectors. In other words, if $x_i = 1$ means that vector $\mathbf{x}$ possesses the i^{th} feature, then $\mathbf{x}^t\mathbf{y}$ is simply the number of features $\mathbf{x}$ and $\mathbf{y}$ have in common, and $\| \mathbf{x} \| \| \mathbf{y} \|$ is the geometric mean of the number of features possessed by $\mathbf{x}$ and $\mathbf{y}$. Therefore, in this situation, the normalized inner dot product is a measure of the relative possession of shared features [7].

CHAPTER 4

Hierarchical Clustering using HABclass Networks

4.1 Overview

ART networks allow patterns to be classified into one and only one category. For some applications, hierarchical classifications are useful. For example, if one were attempting to classify foods, it might be useful to first create categories for fruits and vegetables and then further break fruits into apples, pears, etc.. A macintosh apple would then belong first to the fruit category and then within the fruit category, to the apple category. This hierarchy can be an arbitrary number of layers in depth. In the problem of clustering DNA fragments from the *O. volvulus* parasite, the ability to develop hierarchical clusters is important as well. Because ART networks would not allow such layering of categories, I developed the Hierarchical ART-based Classification Network tool for use on the problem of clustering *O. volvulus* DNA fragments.

The overall goal of adaptive classification is to find a network for hierarchical categorization of a set of DNA sequence fragments in order to analyze the similarities and differences among fragments in those categories and to serve as a classification instrument for new sequences of unknown origin. The overall technique used in HABclass is to start by presenting a set of training sequences to an ART-based network with a vigilance parameter set relatively low. From this first network, a set of categories is obtained that places fragments that are

fairly similar into the same category. Some of these categories will be targeted for further breakdown, and new ART-based networks are trained for each of these categories. This second network runs on only the sequences in the category of interest and will create a set of subcategories for each category of interest. The set of categories and subcategories, taken collectively, is then evaluated to rate the performance of the classification network. Figure 4.1 shows the HAB-class network tool. This method may be applied a number of times to the same set of data, with presentation order, vigilance parameters, etc., changed [15].

4.1.1 *Onchocerca volvulus*

One of the leading causes of infectious blindness in humans is human onchocerciasis, which is caused by *Onchocerca volvulus*, a parasite found in sub-saharan Africa, South America, and Central America [22] [21]. The need for rapid, automated, and adaptable analysis and classification of *Onchocerca volvulus* DNA fragments was the original motivation of the work reported in this thesis. While the resulting methodology is more widely applicable to the class of problems in classification of DNA sequences, this thesis will focus on its development and testing in this specific problem domain.

The specific problem that is addressed in this thesis regards the spread of *Onchocerca volvulus* around the world. Zimmerman, et al. [22] used statistical techniques to suggest that the spread of *O. volvulus* from Africa to the New World occurred relatively recently, probably as a result of the slave trade. The statistical techniques involved in this study proved successful although time-consuming and computationally-intensive, and allowed statistical assessment of results in addition to traditional phylogenetic analysis. Because the evolutionary

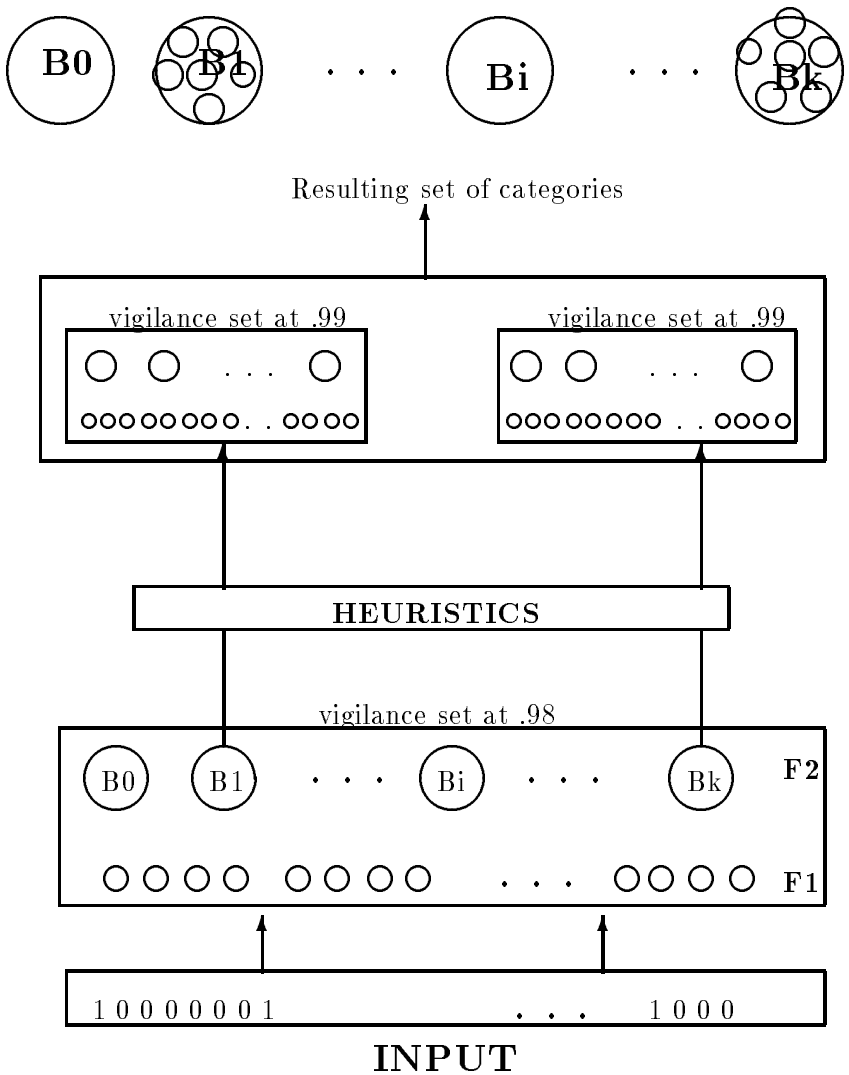

Figure 4.1: HABclass Network

history of a parasite such as *O. volvulus* is important in determining the source of illnesses as well as cures, a faster, easier to use tool is needed.

In determining evolutionary history, we begin with a set of aligned DNA fragments taken from *O. volvulus* from around the world [22]. Our goal is to determine which fragments are related to which other fragments. This problem is, first, one of sequence categorization. Researchers have used a variety of techniques to categorize biological sequences, particularly protein sequences [12] [9]. Most of these techniques disregard the positional information of subfragments because the presence or absence of a particular sequence of amino acids is more important than where that sequence of amino acids might be found. In the problem presented in this thesis, the position of a subfragment is very important. The adaptability and speed of artificial neural networks are particularly suited to such categorization problems.

4.1.2 Data Representation

In the data set used for this work [1], each DNA fragment is an aligned sequence of 106 nucleotides, i.e., a sequence of symbols from the set {`A, C, T, G, blank` (for skips) }. Since ART networks require numeric input, each nucleotide in the sequence is transformed into a series of three zeroes and one one, where the position of the one designates which nucleotide resides at that position in the nucleotide sequence. For this data, an `A` is encoded as 1000, a `G` as 0100, a `T` as 0010, and a `C` as 0001. A skip (deletion) in a particular position is encoded as 0000. Therefore, each sequence of 106 nucleotides will be represented as a

[1] The DNA sequences have been deposited in GENBANK under accession number U02590 to U02594 and U02731 to U02875.

sequence of 106*4=424 zeroes and ones. The F1 layer of each of the layered ART-based networks will have 424 nodes. Of course, in general, the F1 layer of the first ART-based network may have any number of nodes.

4.1.3 The Algorithm

Initially, there are p sequences which are used to train the first ART-based network, N_0. These sequences are presented to N_0 until the network has stably categorized all p sequences. The vigilance parameter, ρ_0, for N_0 is set relatively low for the application domain so that sequences which are relatively similar will be clustered into one category. Once A_0 has been trained, there will be a resulting set of k categories, Ω. Note that $k \leq p$ and that if $k = p$, each sequence has been put into its own individual category, indicating too high a setting for vigilance.

The next step is to separate Ω into two subsets, Ω_I, the "interesting" categories and Ω_U, the "uninteresting" categories, for purposes of further subcategorization. The tagging of interesting categories is currently performed using a heuristic combination of the number in each class (with a tendency to target larger classes for further subdivision) and the within category variation, M, described in the Evaluation section below. Categories with elevated variability may be targeted for further breakdown. Ω_I now contains c categories, where $0 \leq c \leq k$, and for $i = 1 \ldots c$, $p_i \leq p - k + 1$, where p_i is the number of sequences in category i.

For each of the categories in Ω_I, a new ART-based network, N_i, $i = 1 \ldots c$, is trained on the p_i sequences using slow learning until stable categorization has

occurred. For each N_i, the vigilance parameter, ρ_i, is set greater than ρ_0 from network N_0. In this way, for each category i in Ω_I, a set of subcategories, Ω_{s_i}, is defined.

The result is then a new set of categories to be evaluated for performance in categorizing new DNA sequences:

$$\Omega_{New} = \Omega_U \bigcup (\bigcup_{i \epsilon I} \Omega_{s_i}).$$

Iterative application of the process can be used to yield even more finely discriminated categories.

4.2 Evaluation of the Set of Categories

There are two goals in evaluation of the category sets produced by HABclass networks. The first is to determine whether or not a set of categories derived using this technique is truly significant. Since we are using an ART-based mechanism, different runs through the data (with different presentation orders, for example) may produce different classification networks. The second evaluation goal is then to determine which among several possible classification networks is the best. One indication of a good classification is a low measure of variability within clusters compared to the variability between clusters.

A modified F statistic is used to evaluate whether the sum of the square errors for some classification network is significant or not [22]. For a particular set of p sequences, a consensus sequence pattern, $Y_{*,*}$, which uses a positional nucleotide mode of the observed frequency distribution of symbols, is created. That is, we determine which nucleotide occurs most frequently in each position of the sequences and use that nucleotide in the consensus sequence. For example,

if there are ten sequences for which we are attempting to determine a consensus sequence pattern and eight of those sequences have an `A` in the first position while two have a `T`, the consensus pattern would have an `A` in the first position. In the case of ties in the frequency count for two nucleotides in a particular position, we randomly choose one of the two nucleotides for the consensus pattern.

Next, a consensus pattern, $Y_{i,*}$, $i = 1 \ldots k$, is formed for each category containing two or more patterns, using the same method described above (singleton categories are disregarded). The variation among categories, T, is

$$T = \sum_{i=1}^{k} p_i [d(Y_{i,*}, Y_{*,*})^2] \tag{4.1}$$

where p_i is the number of patterns in category i, $d(W, Z)$ is the Hamming distance between sequence W and sequence Z, and k is the number of categories. The variation within categories, M, is

$$M = \sum_{i=1}^{k} \sum_{j=1}^{p_i} [d(Y_{i,j}, Y_{i,*})^2] \tag{4.2}$$

where $Y_{i,j}$ is the j^{th} sequence of the i^{th} category. The test statistic, F, is given by

$$F = \frac{T/(k-1)}{M/(p-k)}. \tag{4.3}$$

To determine whether a particular set of categories is significant, we generate a set of random groupings (with the number of categories, k, fixed) and calculate the modified F statistic for those random groupings. The number of F values greater than the F value for this categorization is calculated. The ratio of this number to the total number of random groupings used is taken as an estimate of the probability of observing a test statistic larger than that

for this categorization (i.e., the p-value for the test of the null hypothesis of no difference in the classification networks) [22].

The above method can only be used to compare networks that result in the same number of categories. Since the ART-based networks generate varying numbers of categories depending upon, among other things, the vigilance parameter, a different method must be used to compare categorizations from various HABclass networks. Suppose we want to compare two sets of categories, X and Y, each determined by different runs of the hierarchical ART network classification method described above. We first calculate the within category variation, M, for both X and Y. Call these M_X and M_Y. Both X and Y have a certain number of categories, k_X and k_Y where $k_X \neq k_Y$. We will then run the randomization test described above for X. For each of the random groupings, we will calculate the within category variation, M.

Examination of the observed values of M for each such random trial has shown that the distribution of the observed M under the randomization process is well approximated by a chi-square distribution. The degrees of freedom (df) for this approximation are estimated by the method of moments, i.e., the degrees of freedom are taken to be the nearest integer to the mean of the observed values of M.

The process is then repeated for the set of clusters in categorization Y. Suppose that the within category variation for categorization X is larger than the within category variation for categorization Y, that is, $M_X > M_Y$. To determine whether M_X is significantly greater than M_Y, use the ratio

$$\frac{M_X/df_X}{M_Y/df_Y}$$

compared to values from a standard F table to check for significance. It should be noted that this technique for assessment of what is best is in the nature of a heuristic statistical procedure rather than a formal statistical test [15].

4.2.1 Evaluation and Singleton Clusters

Note that in the evaluation procedure above, singleton clusters, that is, clusters which have only one sequence in them, are disregarded. For such clusters, the within category variation, M, defined in equation 4.2, will be zero since M is the sum of the squared error between the consensus pattern for the cluster and each of the patterns in the cluster. The consensus pattern for a singleton cluster is the single pattern that is in the cluster. The squared error between a pattern and itself is zero. Therefore, the singleton cluster has no within category variation at all.

As a result, a set of clusters in which every pattern forms its own singleton cluster would have the smallest total squared error since, for such a categorization, there would be no error at all. Clearly, creation of a singleton cluster for each pattern is not a useful solution in most situations. Therefore, some sort of balance must be reached between having a small squared error and having a large number of singleton clusters. This balance will help determine the level of vigilance that is optimal for the data. The vigilance level should produce a small enough squared error but not produce too many singleton clusters. How to determine when the vigilance is set at the correct level to achieve such a balance is an area of ongoing research but at this point, requires the use of some heuristics by expert users of the HABclass net.

4.3 Choosing Vigilance Levels

At each layer of the HABclass net, a vigilance level must be chosen. In the first layer of the HABclass net, the vigilance should be set at a relatively low level to allow coarse categorization of the input data. At each subsequent layer of the net, the vigilance should become increasingly stricter to allow increasingly finer categorizations. The choice of these vigilance levels will determine the resulting categorizations.

The question of how to choose these vigilance levels is not a trivial one and is the subject of ongoing research. For the sample runs described in chapter 5, the initial vigilance was determined through experimentation of the type described in section 4.1.3. The results of such experimentation were examined by our collaborator and expert in the field, biologist Dr. Thomas Unnasch [2]. From his comments, a good initial vigilance of .98 was chosen. Further discrimination in subsequent layers of the net was achieved by further experimentation and examination. Feedback from our collaborator indicates the usefulness of being able to identify certain categories for further refinement while leaving other categories intact. As explained in section 4.1.3, the creation of singleton clusters at this time precludes the use of some objective measure of error but again, this is an area of ongoing research.

[2]University of Alabama, Birmingham, Division of Geographic Medicine, Department of Medicine, Birmingham, AL 35294

4.4 Requirements of HABclass Networks

4.4.1 Aligned Sequences

In order to find the natural structure inherent in a set of sequences, HABclass nets have a number of requirements that must be met. First, patterns that are to be classified by a HABclass net must be aligned. For example, if a HABclass net were to be used to classify amino acid sequences where the conserved regions might be located at various places in the different sequences, these conserved regions would previously have to be aligned by running some multiple sequence alignment program on the sequences. HABclass net are not designed to find the structure if the sequences are not aligned. A corollary to this requirement is that the sequences to be aligned must all be of the same length since this technique is ultimately based on artificial neural networks which have a fixed number of nodes in the input layer.

4.4.2 Numeric Representation for Data

In addition, all input to the HABclass net must be numeric. This, again, results from the fact that a HABclass net is an artificial neural network which uses mathematical combinations of the input and some internal weights to determine categories. As a result, an appropriate data representation must be determined before the net can work on the data.

It is important to note that, unless care is taken, data representation can accidentally introduce unwanted relationships between sequences. For example, consider a data representation where the four bases are represented using two bits rather than the four as described. That is, let `A` be 00, `G` be 01, `C` be 10

and T be 11. In this representation, A is closer to G than it is to T because A differs from G by one bit while it differs from T by two bits. In addition, G is closer to A and T than it is to C because it differs from A and T by one bit each and from C by two bits. These distances do not represent reality and should be avoided. This is why the data representation described above, although it may seem inefficient, was chosen. Note that in the data representation described above, a skip (deletion) is closer to each base than any base is to any other base. This situation represents is reflective of the physical world since many would argue that it is easier to introduce a skip in a DNA sequence than it is to transform from one base to another base.

4.4.3 ART Parameter Tuning

The final requirement of the use of HABclass nets for clustering involves a difficulty with ART networks in general. As a model of the human brain, ART networks are very complex, with many components and parameters. The internal functioning of ART networks is difficult to track. The tuning of the many parameters in an ART network is not a trivial task and thus, there may be applications where an ART network behaves erratically until the parameters can be adequately tuned.

4.5 Advantages of HABclass Networks

4.5.1 Unsupervised Learning

When the correct groupings for the given data are unknown, an unsupervised approach to learning is warranted. An unsupervised learning method will attempt to find the natural dependencies in the data without the bias of the user

coming into play and then will use that information to classify new sequences, again without the bias of the user being an issue. Of course, this lack of supervision has a price. To determine which sequences belong in the same category together, the vigilance level must first be found. This may not be a trivial task for some applications and usually involves some trial and error.

4.5.2 Easy Adaptability

At the heart of the HABclass net is an artificial neural network. The particular networks used, based on adaptive resonance theory [3], are very adaptable in the face of new events. This may be extremely important in some applications, especially when presentation of a completely new event must be signalled somehow. An example of such an application might be the monitoring of machinery. A HABclass net could be trained to recognize normal machine functioning. When some sort of catastrophic event, such as machine failure, occurs, the event should not simply be clustered with the closest normal cluster. Instead, this event should be recognized as a completely unfamiliar one and clustered as such. HABclass nets provide such easy adaptability.

4.5.3 Singleton Clusters

In many applications, the most interesting data is that which is different from anything else seen. Such an outlier should form a singleton cluster, that is, a cluster in which there is only one pattern, namely, the outlier. This singleton cluster should not come at the expense of creating other clusters which are abnormally large, which might happen, for example, in the case of the K-means

algorithm where the number of clusters, K, must be specified *a priori*. That is, in the K-means algorithm, the creation of a singleton cluster will force the remaining patterns to be placed into K-1 clusters.

Singleton clusters may signal an event that is completely new to the problem or they may simply signify that the data sampling done was not broad enough. In the *O. volvulus* DNA problem described above, it was found that some singletons clusters created with subsets of the labeled sequences were no longer singletons once the unlabeled data was clustered. Other singleton clusters in that example remained singletons once the rest of the data was clustered. This phenomenon will be discussed further in chapter 5.

HABclass nets create and retain singleton clusters if such clusters are natural to the structure of the data for a given vigilance level. The difficulties that such singleton clusters cause in attempting to evaluate a set of clusters was discussed in section 4.1.3 while the benefits of retaining such clusters will be discussed in chapter 5.

CHAPTER 5
HABclass Clustering Results

5.1 Evaluation of One-Layer HABclass Nets

The techniques described in Section 4 were tested on 29 sequences, each containing information on 106 nucleotides [22]. These sequences represent a subset of the total 150 labeled sequences available for testing but I chose them because of their probably high level of similarity. These 29 sequences come only from Guatemala and Brazil. The sequences from the forest region of Africa, the savannah region of Africa and the "common" sequences [22] have been left out of the following tests. Sequences with such a high level of similarity present a particularly difficult clustering task. Tests for significance of the sets of categories from HABclass networks yielded excellent results, with significance levels typically at 0.0001 or smaller. These levels of significance are similar to those reported for the manually-derived categorization in [22]; significance of the categorization resulting from use of HABclass is clear.

In comparing the performance of two classification networks, the testing methodology is illustrated by the following example test: Setting the vigilance parameter to .98, the top-level HABclass network yielded six categories for the inputs, with three singletons falling out, as shown in Figure 5.1. The within category variation for this clustering was $M_{5.1} = 202$. With the vigilance parameter set to .99, the input sequences fell into ten categories with within category

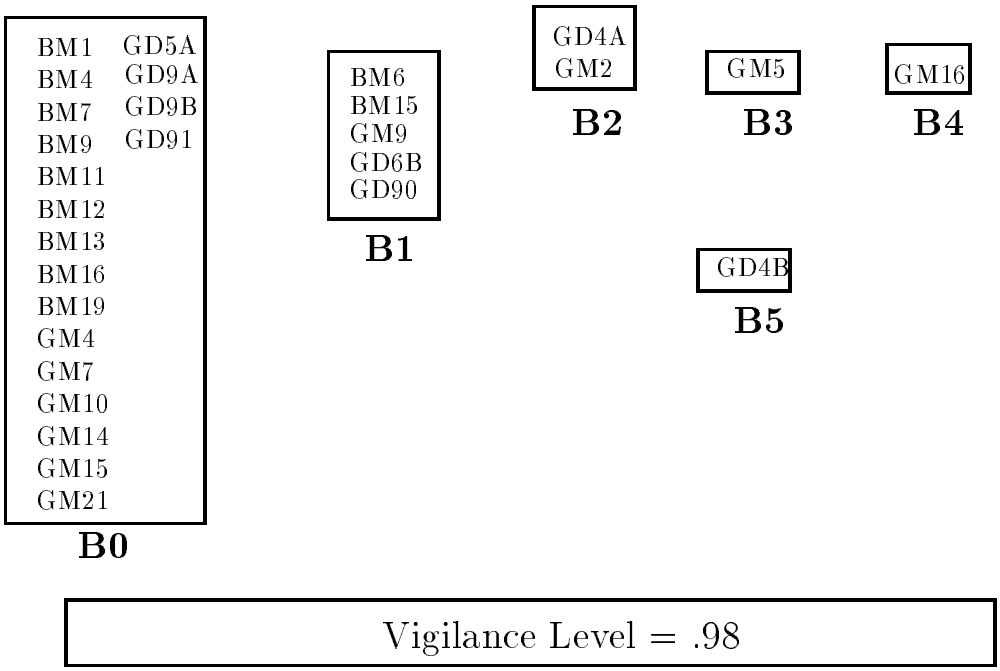

Figure 5.1: ART2 Clustering of Guatemala and Brazil Data - Vigilance = .98

variation, $M_{5,2} = 46$, and seven singletons falling out. Figure 5.2 shows these categories.

The clustering shown in Figure 5.2 differs from that shown in Figure 5.1 in a number of ways. First, the large category, **B0**, created with vigilance .98 has been broken into two categories, **B0** and **B2**, when the vigilance is pushed up to .99. In addition, category **B1** created with vigilance .98 is broken into a number of smaller categories. Sequences BM6 and GM9 remain together while sequences BM15 and GD6B each form their own singleton categories. Sequence GD90 falls into cluster **B2** along with the large majority of sequences from the

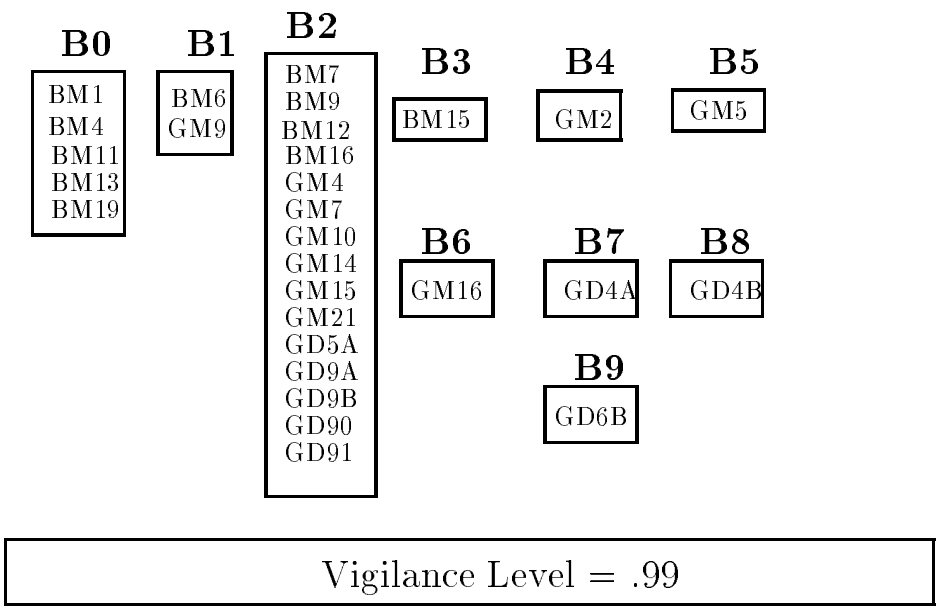

Figure 5.2: ART2 Clustering of Guatemala and Brazil Data - Vigilance = .99

.98 vigilance level category **B0**. Finally, category **B2**, which contains only two sequences when created using vigilance .98, is split into two singleton categories, **B4** and **B8**, using vigilance .99. These results suggest that perhaps a vigilance of .98 is not quite strict enough for category **B0** but is strict enough for the remaining categories. However, as explained earlier, an ART2 network requires a *global* vigilance and thus, cannot handle situations where some of the data should be clustered with one vigilance level while the rest of the data should be clustered with the a different vigilance level. This was the original motivation for the development of the HABclass net tool.

Notice that the within cluster variation of the network shown in Figure 5.1, that is, $M_{5.1} = 202$, is significantly greater than the within cluster variation

of the network shown in Figure 5.2, that is, $M_{5.2} = 46$. This points to one of the main problems with the way the evaluation of the clustering is done at this time. The evaluation procedure favors clusterings that have a large number of singleton clusters so that the best clustering would be one in which each sequence is placed into its own, singleton cluster. Notice that when one is attempting to determine evolutionary history of DNA sequences, such a clustering is not useful. The balancing of a large number of singleton clusters with a low measure of within cluster variation is a subject of on-going research.

5.2 Two-Layer HABclass Nets

Figure 5.3 shows an example of using a HABclass net for clustering of the 29 sequences described in Section 5.1. First, the 29 sequences are sent through an ART2 network with the vigilance set to .98. As described in Section 5.1, six categories with three singletons result. Two of the non-singleton categories are then chosen for further refinement. Category **B0** is sent through a second layer ART2 network with vigilance set to .99, resulting in two subcategories, neither of which is a singleton. Category **B1** is also sent through a second-layer ART2 network, also with vigilance set to .99, resulting in three categories, two of which are singletons. These two categories were chosen for further refinement on the basis of their size, that is, they are the two largest clusters, and on the basis of their relatively large within cluster variations, which are 117 for category **B0** and 72 for category **B1**. In reality, any heuristic may be used to choose categories for further refinement. Automation of some of these heuristics is a subject of ongoing investigation.

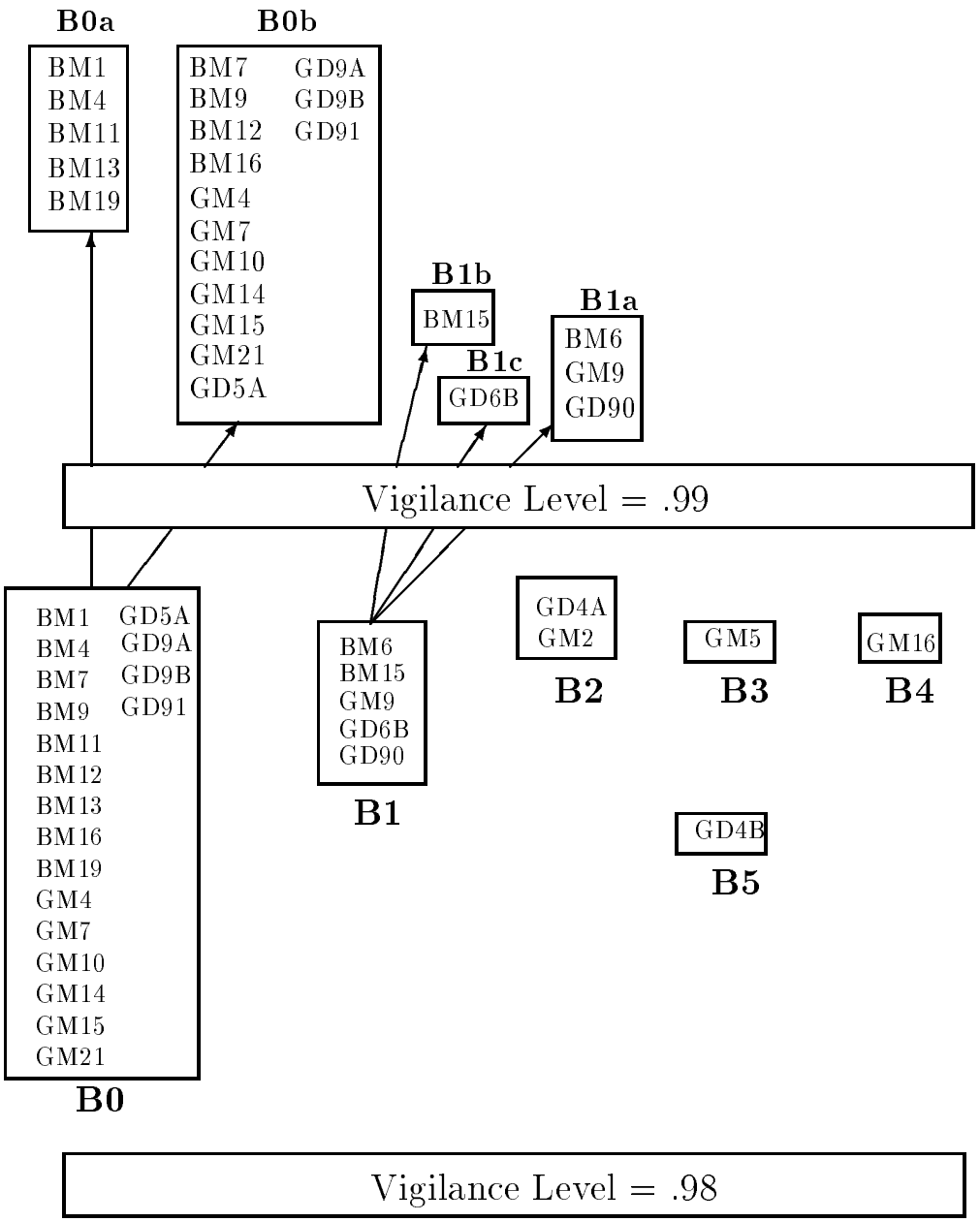

Figure 5.3: Sample Run of HABclass using Only Brazil and Guatemala Data

For the set of clusters reported in Figure 5.3, the within cluster variation, $M_{5.3}$ is 53. Notice that sequence GD90 falls into category **B1a** in Figure 5.3. When all of 29 sequences were clustered with vigilance .99, as shown in Figure 5.2, this sequence fell into a category equivalent to **B0b** in Figure 5.3. Figure 5.2 suggests that sequence GD90 is actually closer to the template of category **B0b** than that of category **B1a**, where it has been clustered. In fact, the within cluster variation, $M_{5.3^*}$, increases to 59 if sequence GD90 is moved from category **B1a** to category **B0b**. Of course, the HABclass net in Figure 5.3 will never place sequence GD90 in category **B0b** in the second layer because that sequence does not fall into category **B0** in the first layer. In fact, GD90 differs from sequence BM6 by 4 bases and from sequence GM9 by 2 bases while GD90 differs from the sequences in category **B0b** by anywhere between 0 and 6 bases.

Once a HABclass net has been trained using some set of data, additional data may be classified using the net. Figure 5.4 shows the use of the network from Figure 5.3 for classification of the 61 unlabeled *O. volvulus* sequences (see Appendix A). One singleton category from Figure 5.3, **B3**, is no longer a singleton category. This suggests that perhaps this category has not been sampled adequately. In other words, sequence GM5 is not simply an anomoly but instead represents a category into which other sequences, 4-6 for example, also fall. If the only data available were the original sequences from Brazil and Guatemala, one might conclude that GM5 is simply noise of some type.

In addition, many of the 61 unlabeled sequences shown in Figure 5.4 represent truly different clusters and the HABclass network creates new clusters to handle these sequences rather than simply placing the sequences into the closest

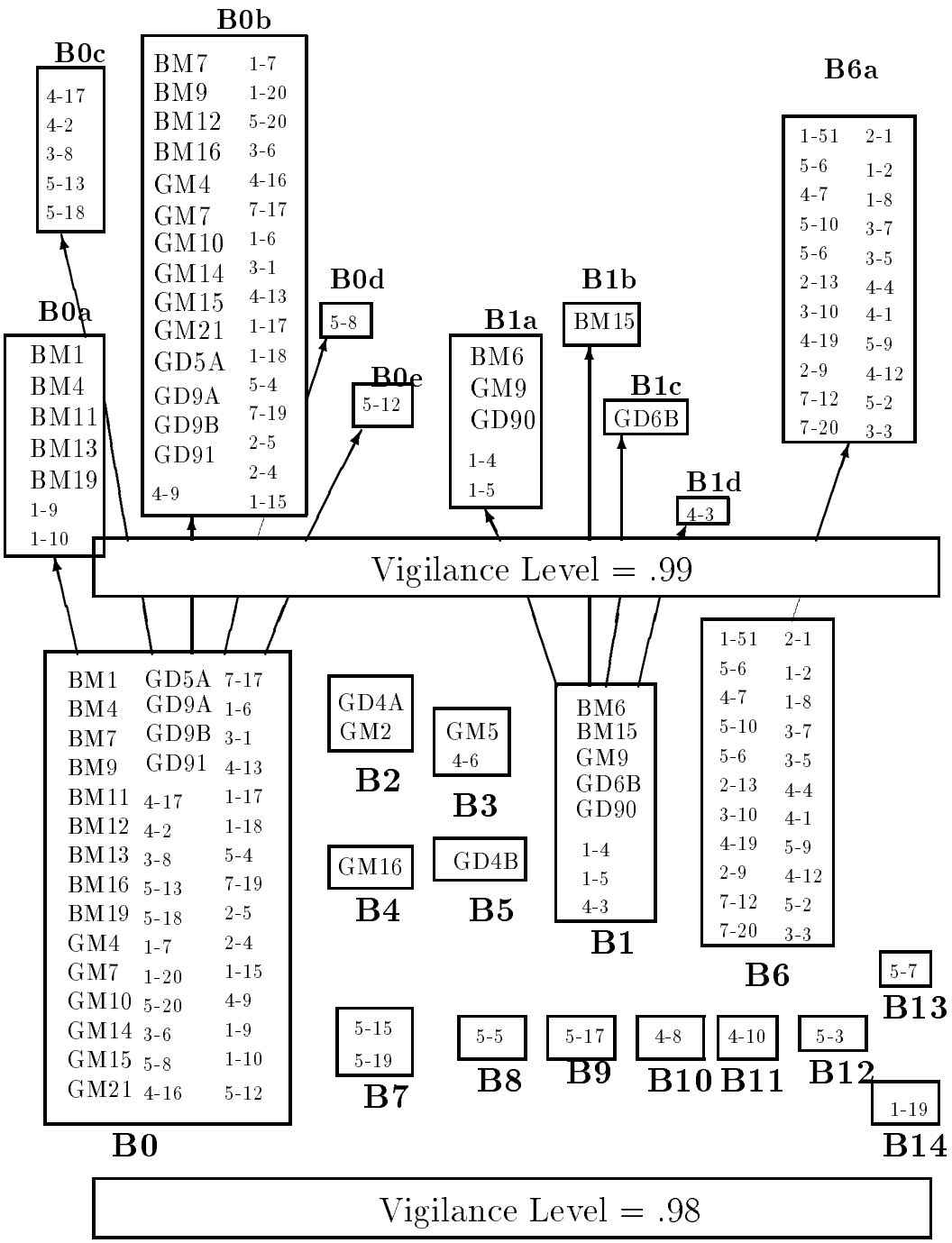

Figure 5.4: Sample Run of HABclass using Unlabeled Data

Table 5.1: Effect of changing vigilance level

Vigilance Level	Number of Categories	Number of Singletons	Error/ df
.95	2	0	.42023
.97	8	4	.22690
.98	19	9	.06539
.99	32	23	.02565

cluster. The unlabeled sequences that fall into category **B0** are sent through the second layer ART2 network from Figure 5.3 for category **B0**. Although a large number of them fall into existing subcategories, **B0a** and **B0b**, many of the sequences also cluster into new categories, including two singleton categories, **B0d** and **B0e**. In addition, those unlabeled sequences that fall into category **B1** are sent through the second layer ART2 network from Figure 5.3 for category **B1**. Finally, a new cluster, **B6**, is chosen, again using the size heuristic, for further refinement. After being sent through a second layer ART2 network with vigilance set to .99, no subcategories are created. This indicates a very high level of similarity among the sequences that fall into this cluster. The within category variation for the first layer of Figure 5.4 is $M_{5.4_1} = 806$ while the within cluster variation for the entire, two-layer network is reduced to $M_{5.4} = 280$.

The results from testing HABclass on 107 sequences (that is, all labeled sequences except the common ones [22]) with a variety of vigilance levels are

reported in Table 5.1. Note that all two-way tests of the networks at vigilance levels adjacent to one another in the table indicate that the network which uses the higher vigilance level yields an error which is significantly smaller than that of the lower vigilance level (with p-value < 0.005). A more appropriate multi-way test for differences is under study. Note that the effect that raising the vigilance level has on classification error is somewhat predictable: the absolute minimum error can be obtained by forcing each sequence into its own individual category. This effect must be offset by other factors including a goal of keeping the number of singletons small and the number of categories significantly less than the number of sequences being categorized. The selection of an optimal number of singletons compared to the overall within cluster variation is a topic of on-going research.

CHAPTER 6
Comparisons of Algorithms

6.1 HABclass Networks and the Leader Algorithm

ART networks have been compared to the leader algorithm [6] [13]. Carpenter, *et. al.* point out that the major difference between the two algorithms lies in the learned expectations at the F2 layer [6]. In the leader algorithm, each category is represented by a particular sequence, the *leader* of that category, which has been classfied in that category [13]. In the ART2 algorithm, each category is represented by a template, which is a combination of all the sequences that have been classified in the category [3]. The template is created using equation 2.13 and can be recovered using equation 2.17.

Figure 6.1 shows the clustering of the 29 sequences from Brazil and Guatemala using the simple leader algorithm with a user-defined vector threshold distance of .94. Our tests suggest that there is little difference between using a vector distance metric and using a Euclidean distance metric. For example, compare Figure 6.1 with Figure 6.2, which is the clustering of these same sequences using a Euclidean distance threshold set to 4. The two differences between these clusterings are the clusterings of sequences GD90 and GM21. In Figure 6.1, sequence GM21 forms a singleton cluster while in Figure 6.2, it is placed into category **B0** along with the majority of the 29 sequences. In addition, sequence GD90 is placed in cluster **B1** using the vector distance while it is placed in the larger cluster **B0** using the Euclidean distance. As described in Chapter 5, se-

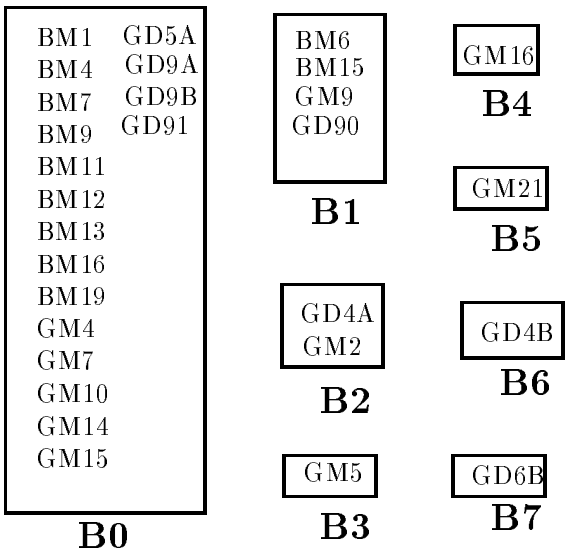

Figure 6.1: Simple Leader Algorithm with Vector Distance = .94

quence GD90 has more bases in common with the sequences in cluster **B1** than those in cluster **B0**. The clusters in Figure 6.1 show a slightly smaller within cluster variation, $M_{6.1} = 123$ than the clusters in Figure 6.2, where $M_{6.2} = 163$. This is due to the addition of a singleton cluster which adds nothing to a within cluster variation calculation and the fact that GD90 is closer to cluster **B1** than cluster **B0**.

Again, Figure 6.2 shows how the 29 sequences from Brazil and Guatemala have been clustered using the simple leader algorithm with a Euclidean distance threshold of 4. Most of the sequences in this test have been clustered in exactly the same way using a vigilance of .98 in HABclass as shown in Figure 5.1. A notable exception is sequence GD90 which in Figure 5.1 is clustered in category **B1** with a small number of other sequences while using the simple leader algo-

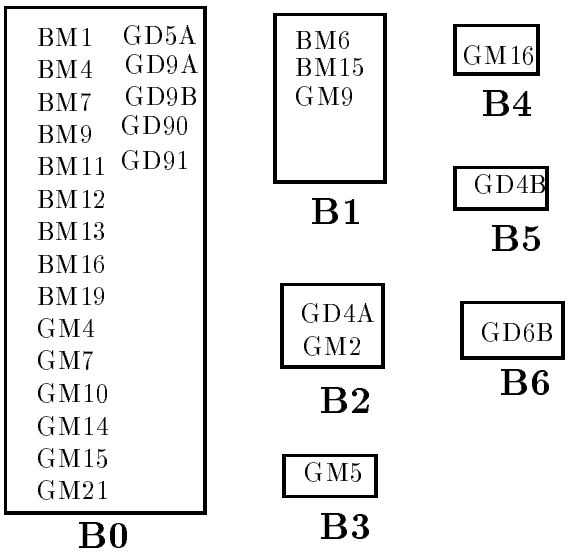

Figure 6.2: Leader Algorithm with Euclidean Distance Threshold = 4

rithm, this sequence is clustered in category **B0** with a large number of other sequences.

The simple leader algorithm clusters sequence GD90 in category **B0** because the algorithm puts a sequence into the first cluster whose leader is close enough to the sequence. The fact that the sequence is close enough to the leader of category **B0** causes that sequence to be placed into that cluster despite the fact there is another cluster whose leader is closer to the sequence. The leader of category **B0**, sequence BM1, differs from sequence GD90 by seven bases. Sequence GD90 is significantly closer to each of the sequences in category **B1** of Figure 6.2. That is, sequence GD90 differs from sequence BM6 by four bases, from sequence BM15 by six bases and from sequence GM9 by two bases. This example illustrates one of the main problems with the simple leader algorithm.

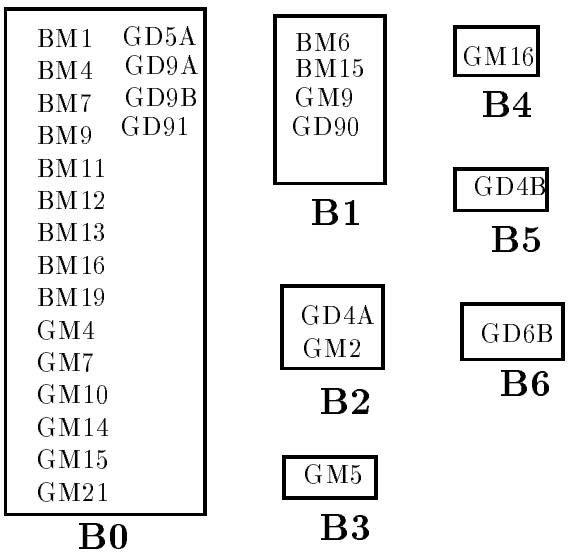

Figure 6.3: Modified Leader Algorithm with Threshold = 4

In this algorithm, the clusters that are created first are larger than clusters that are created later because the earlier clusters have a first look at any new sequence. This problem is solved with the modified leader algorithm in which a sequence is placed into the cluster whose leader is closest to the sequence at a given time.

Figure 6.3 shows the clustering of the 29 sequences from Guatemala and Brazil using the modified leader algorithm with a Euclidean distance threshold of 4. Notice that sequence GD90 is now in cluster **B1** just as it was in the HABclass net example shown in Figure 5.1. In fact, the only difference between Figure 5.1 and Figure 6.3 is the clustering of sequence GD6B. In Figure 5.1, sequence GD6B is part of category **B1** while in Figure 6.3, the sequence has formed a singleton category.

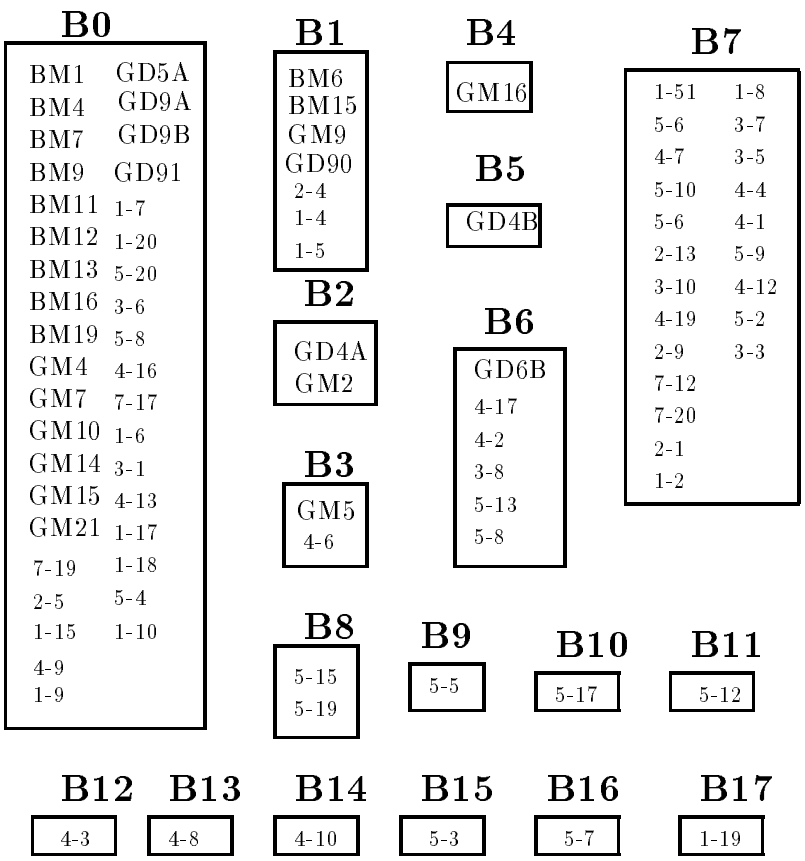

Figure 6.4: Modified Leader Classification of Unlabeled Data

Sequence GD6B differs from the leader of Figure 6.3 category **B1**, sequence BM6, by eleven bases and this is why the modified leader algorithm has created a singleton category for GD6B. Because the HABclass net creates a *template* for the category rather than relying on a particular sequence to represent the category, the sequence GD6B falls close enough to the template of Figure 5.4 category **B1** to be classified there. The difference in where sequence GD6B is clustered may not appear to have much of an effect but when using the resulting sets of clusters to classify new patterns, this seemingly small difference magnifies,

as can be seen when comparing the HABclass net shown in Figure 5.4 and the set of clusters shown in Figure 6.4. Figure 6.4 shows the set of clusters created when using the leaders developed in Figure 6.3 to classify the unlabeled *O. volvulus* sequences. The clusters shown in the HABclass network of Figure 5.4 result in a within cluster variation of $M_{5.4} = 280$ while the clusters shown in Figure 6.4 result in a within cluster variation of $M_{6.4} = 466$. Clearly, the HABclass network of Figure 5.4 creates a set of clusters that is better than the set created by the modified leader algorithm of figure

6.2 HABclass Networks and the K-means Algorithm

The K-means algorithm requires that the user specify the number of clusters into which the data will be placed. This number can be difficult to obtain. As a result, in the following comparisons, the number of clusters for the K-means algorithm will be determined by looking at the number of clusters created by the algorithm against which the comparison is to occur. For example, for a comparison of the K-means algorithm and the HABclass net, using Figure 5.1, the number of clusters for the K-means algorithm would be six. Of course, in a real world situation, there may not be an algorithm against which one can compare in order to determine the number of clusters to be used in the K-means algorithm. If such an alternative algorithm were available, one might simply use that algorithm for clustering rather than the K-means algorithm. In addition, the tests below use a version of the K-means algorithm implemented with a Euclidean distance metric since, the difference between using a Euclidean distance metric and using a vector distance metric appears to be minimal.

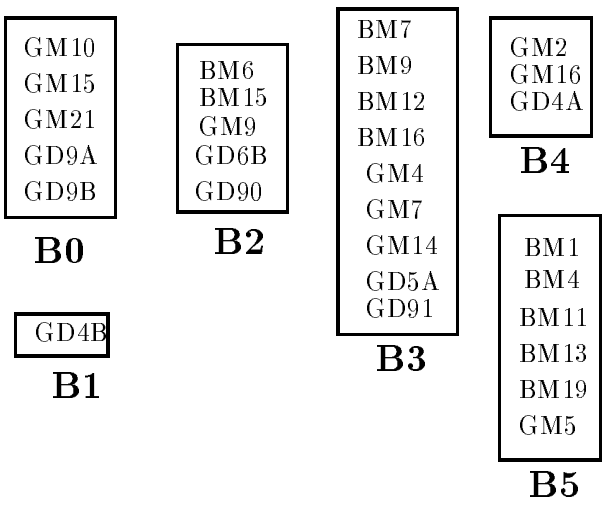

Figure 6.5: K-means Clustering of Brazil and Guatemala Data - 6 Clusters

The K-means algorithm, executed for grouping the 29 Brazil and Guatemala sequences into 6 clusters, results in the set of clusters shown in Figure 6.5. This set of clusters is very different from the clusters created by the one-layer HABclass net shown in Figure 5.1, which also results in six clusters. The within cluster variation for the K-means algorithm with six clusters is $M_{6.5} = 415$. For the one-layer HABclass net shown in Figure 5.1 with six clusters, the within cluster variation is $M_{5.1} = 202$. Clearly, the one-layer HABclass net in this case has created a better set of clusters than the set created by the K-means algorithm. The within cluster variation for the HABclass net classification can be further reduced by running a second layer of clustering for some of the categories shown in Figure 5.1. An example of this is shown in Figure 5.3, where nine clusters are created. The within cluster variation for the HABclass net of Figure 5.3 is $M_{5.3} = 53$.

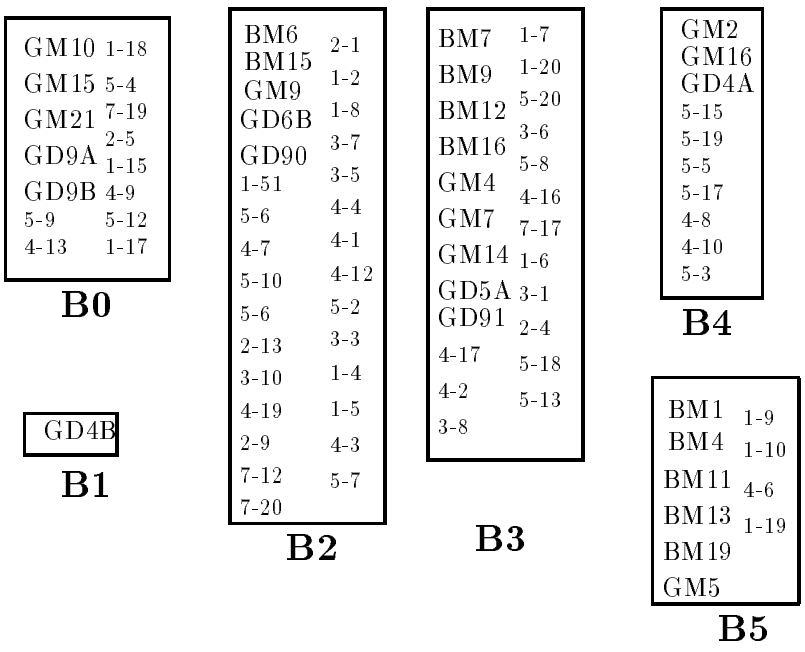

Figure 6.6: K-Means Classification of Unlabeled Data - 6 Clusters

One of the primary advantages of HABclass nets is their ability to be saved and used for later classification of new sequences without retraining on the old sequences. Figure 5.4 shows a sample run of a two-layer HABclass network. First, the 29 Brazil and Guatemala sequences were clustered using the two-layer algorithm as described in Section 5.2 and shown in Figure 5.3. Then, using the nets created in that example, the unlabeled *O. volvulus* sequences were classified. The within cluster variation for the twenty-two clusters created by this HABclass net is $M_{5.4} = 280$. As a comparison, Figure 6.6 shows the use of the clusters created by the K-means clustering algorithm as shown in Figure 6.5 for classification of the 61 unlabeled *O. volvulus* sequences. The within cluster variation for this set of clusters is $M_{6.6} = 4236$. Six clusters are used for

the K-means classification since this is the number of clusters that was initially used for Figure 6.5. In this comparison, it is clear that as the K-means clusters are used for later classification of new sequences, the within cluster variation gets significantly worse since the K-means classification algorithm has no ability to dynamically create new clusters in the classification phase. Each sequence must simply be placed into the cluster to whose means it is closest.

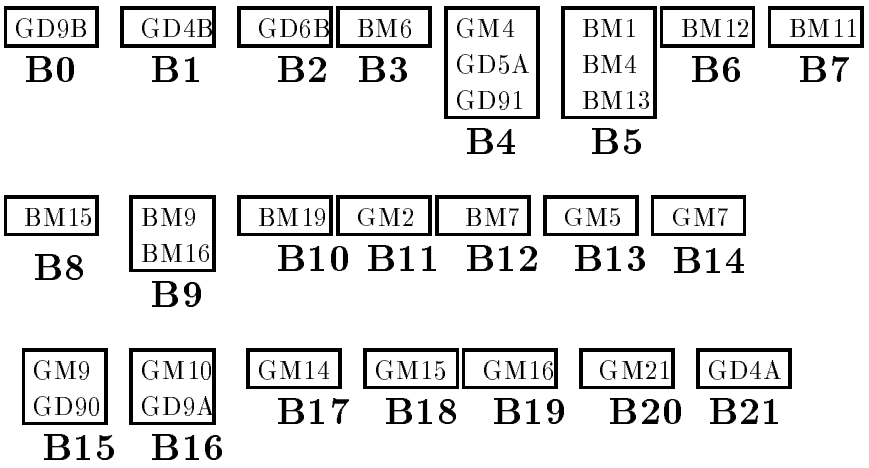

Figure 6.7: K-Means Clustering of Brazil and Guatemala Data - 22 Clusters

Perhaps it is unfair to compare the six clusters created by the K-means algorithm shown in Figure 6.6 with the 22 clusters created by the HABclass net shown in Figure 5.4. Perhaps a fairer comparison would be to set the number of clusters for K-means to 22. Figure 6.7 shows the 29 sequences from Brazil and Guatemala placed into 22 categories. The within cluster variation for this set of clusters is very low due to the high number of singleton clusters. In fact, the

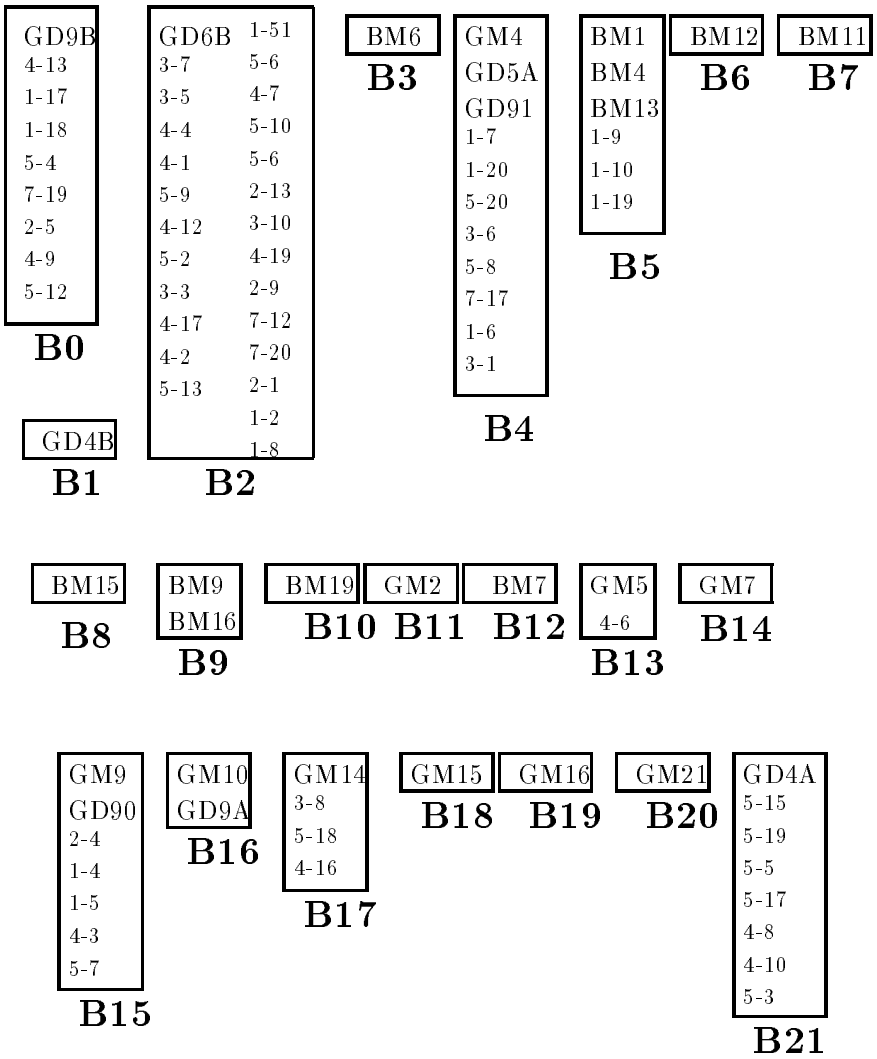

Figure 6.8: K-Means Classification of Unlabeled Data - 22 Clusters

within cluster variation is only $M_{6.7} = 4$. Recall that a singleton cluster has a within cluster variation of 0 and thus, adds nothing to the total within cluster variation for the set of clusters.

Using the set of clusters shown in Figure 6.7 to classify the 61 unlabeled *O. volvulus* sequences results in the set of clusters shown in Figure 6.8. The within cluster variation for this set of 22 clusters is $M_{6.8} = 1680$. Clearly, the within cluster variation, $M_{5.4} = 280$, for the HABclass net shown in Figure 5.4 is significantly smaller than that for Figure 6.8.

The K-means algorithm does a very good job at creating clusters with a minimum of within cluster variation. It falls short of the HABclass net technique, however, when the task requires the later classification of a set of new sequences. This is, in fact, one of the main advantages of HABclass nets and is also one of the main requirements for an adequate solution to the problem of tracing the evolutionary history of the human parasite *O. volvulus*.

6.3 Comparison Conclusions

The HABclass network tool creates clusters that compare favorably to the clusters created by the simple leader algorithm, the modified leader algorithm and the K-means algorithm. In each of the examples cited above, the HABclass network clusterings resulted in a significantly smaller within cluster variation than comparable clusterings.

The simple leader algorithm suffers from a number of problems. The most significant of these problems is probably that this algorithm is very susceptible to changes in presentation order. For example, the first sequence presented

always becomes a cluster leader. In addition, the clusters that are created first will grow larger than those clusters created later since a sequence is placed in the *first* cluster whose leader is closer than the user-defined threshold rather than in the cluster whose leader is *closest*. The result is that the tightness of the clusters is suboptimal.

The modified leader algorithm takes care of the simple leader algorithm's naive placement of a sequence into the first cluster whose leader is close enough. In other words, in the modified leader algorithm, a sequence will be placed into the existing cluster whose leader is closest. The modified leader algorithm does still suffer from the presentation order problem in that, again, the first sequence presented will always be a cluster leader. Again, the within cluster variation for the modified leader algorithm is suboptimal.

The K-means algorithm creates clusters that are better than those created by either the simple or the modified leader algorithms. It, however, suffers from other problems. The first problem is that the user must determine *a priori* how many clusters are needed by this data. Finding an appropriate value for K can be extremely difficult. Once a K is chosen, the K-means algorithm does reasonably well with the initial task of clustering. The algorithm breaks down, however, when the task changes to classification of new sequences using previously determined clusters. The within cluster variation for these classification tasks is very large.

CHAPTER 7
Future Work

The work presented here has proven useful to biologists working to trace the evolutionary history of the human parasite *Onchocerca volvulus*. There are a number of areas where work is on-going.

The issue of how to select categories for further refinement remains an open issue. Currently, human intervention is required for such selection. Biologists use a set of heuristics to determine which categories are interesting in the first layer of the HABclass net and then send those categories through a second layer net with vigilance turned up a bit. The automation of such selection in the form of an expert system appears to be within immediate reach.

The choice of an appropriate vigilance level at each layer of the HABclass net is also a subject of on-going investigation. Currently, a user of the system must hand tune the vigilance at each layer by trying out various vigilance levels. The choice of an appropriate vigilance level appears to be data dependent. Thus, a solution to this problem will most likely involve a set of heuristics or tests through which the data can be run for determining, from the data, an appropriate vigilance level.

Adaptive resonance networks are presentation order dependent, that is, given a different presentation order, a different set of clusters will be created. Many clustering algorithms, including both the simple and modified leader algorithms, have this dependence. Table 7.1 shows the effect of presentation order on the

creation of clusters using the HABclass net technique. In this table, our data was taken from four isolates, labeled Forest, Savannah, Brazil, and Guatemala. The table shows the effect of presenting the isolates in five different orders. Tests of significance indicated that Presentation orders 1 and 5 yielded classification networks which had significantly lower error than Presentation orders 2, 3, or 4.

Getting significant differences in such a test indicates that presentation order does need to be taken into account, and that choosing the best network among those resulting from several runs of this technique with differing presentation orders of the data is a good first step. Determining an optimal presentation order is, thus, also an area of on-going research. A solution to this problem will also likely involve a set of heuristics. One suggestion might be to choose the first sequences to be presented to the net by examining the data for the two sequences which are farthest apart using some distance metric, eliminate the next n closest sequences and repeat the processes until there are no sequences left.

Singleton clusters still present a problem in determining how good a particular clustering is. Many singleton clusters will cause the within cluster variation to be small so that the best solution appears to be when each sequence is placed in its own, individual cluster. This solution tends not to be very useful, however. What to do with singleton clusters in the evaluation process remains an open issue. Currently, a user makes a decision about where to draw the line between a large number of singleton clusters and a small within cluster variation. Automation of this process will probably involve some sort of penalty function for

Table 7.1: Effect of presentation order (vigilance = 0.98)

	Presentation Order	Number of Categories	Number of Singletons	Error	Degrees of Freedom (Est.)	Error / df
1	Forest Savannah Brazil Guatemala	19	10	786	12114	0.06488
2	Savannah Brazil Guatemala Forest	18	11	1475	12538	0.11764
3	Guatemala Forest Brazil Savannah	16	10	1682	12796	0.13145
4	Savannah Brazil Forest Guatemala	18	9	1469	12471	0.11779
5	Brazil Savannah Guatemala Forest	20	11	763	12028	0.06344

large numbers of singleton clusters or will include the among cluster variation in the calculation of error.

The work that remains to be done on this project does not preclude the use of HABclass nets currently for determining a hierarchical clustering of a data set. Solutions to the problems described above, however, will make the tools developed to this point even more useful.

CHAPTER 8
Conclusions

As human beings, we are very good at classifying objects. In fact, our ability to classify objects allows us to use natural languages since without such ability, we would have no common conception of "chair" or "to run" or any other noun or verb. Our ability to speak is part of what is human intelligence. Therefore, one might say that the ability to distinguish one object from another is part of what makes us intelligent.

Much of our classification ability relies on context-dependent information. We are able to observe that an apple is a different kind of object than an orange, despite the fact that they are both fruits. If we are given the task of going to the grocery store to buy some fruit, we would classify apples and oranges together and allow both to be among the possibilities for purchase. If we are told to buy apples, on the other hand, we will not consider oranges among the possibilities but will consider macintosh, red delicious and granny smith apples.

We are also able to recognize novel objects even if we may not have a name for them. For example, if, as adults, we see an object that is completely unlike any we have seen before, we will probably not conclude that this new object is an example of X, where X is some class of objects with which we are familiar. We will, in other words, recognize the new object for what it is: a new object.

There are, of course, limitations in human classification ability. If, for example, we were shown a group of upwards of 1000 people and were asked to

determine whether this group had an even number or odd number of people in it, most of us could only complete this task through a tedious count of the people. Even then, we would likely make mistakes in the counting process. Classification tasks that require many computations or simply involve a large number of objects or features of objects are difficult for most humans to complete. Such tasks are perhaps best performed by electronic devices such as computers, which are better suited than humans to the largeness of the task.

Humans, then, are adept at classification tasks that involve a few objects or a few features at a time, context-dependent information, hierarchical classifications and potentially novel objects. The goal for an automated classification tool is to combine the parts of the human classification ability that work well with the computational ability of computers. Such a classification tool would be useful in situations where the number of objects or the number of features to be used in the classification is very large, such as in the problem of tracing the evolutionary history of the *O. volvulus* parasite. In this problem, as described in this thesis, there are more than 200 DNA sequence fragments, each comprised of 106 bases (or features). Humans have a difficult time assigning this number of long fragments to some sort of clustering. But the human abilities to use context-dependent information, to create hierarchical classifications and to recognize truly novel sequences are important to finding good solutions to this problem.

The artificial neural networks created using adaptive resonance theory attempt to model human classification ability. These networks are very good at classifying objects in an unsupervised manner, that is, determining what the

underlying, common structure of the data is (given an appropriate data representation, of course). The networks are able to use features of the objects in a context-dependent manner. In one situation, a particular feature may be a key feature for distinguishing one object from another, while in another situation, that same feature may be noise and thus, be ignored by the network. In addition, unlike most classification algorithms, an ART network is able to recognize that an object represents a truly unique event and report that fact by placing the object in a newly created category. Most other classification algorithms will place an object never before seen by the algorithm into one of the existing categories no matter how closely (or distantly) related to this category the object actually is.

Because of the two features described above, ART networks represent a first step toward modeling the manner in which humans classify objects. In addition, these networks classify very quickly so that a large number of objects with many features can be classfied in a relatively short period of time. ART networks are unable, however, to make classifications which place an object into a particular category and then further classify the object into a subcategory. For example, a network based on adaptive resonance theory is able to distinguish an apple from an orange but, after creating the apple category, is unable to distinguish macintosh apples from red delicious apples. That is, standard ART networks cannot classify objects hierarchically. Hierarchical ART-Based Classification Networks represent the next step in the modeling of human classification ability. Using several layers of adaptive resonance networks, the HABclass algorithm is able to further refine selected categories to provide hierarchical classification of

objects. Thus, a HABclass network provides the ability to classify objects using some of the most desireable features of the human classfication ability while at the same time providing the additional ability to classify large numbers of objects very quickly.

Although HABclass networks possess some of the best features of human classification ability, there are numerous steps in the HABclass process where human intervention is still required. One step is the determination of interesting categories to be further refined. Although one can imagine this step being taken over by a computer-based expert system, such a system would necessarily be application dependent, with a new expert system for each new application. Before the algorithm can even begin, however, the issue of data representation must be considered. A human classifier somehow learns not only which of the features possessed by the object are critical to the object's membership in a particular category, but also what is an appropriate representation for those features. HABclass nets currently must have some of this information given to them by the human user. For example, a human must determine which are the features of an object that will be presented to the HABclass network and what representation must be used for those features. As long as the human user chooses an adequate set of features and represents them in an appropriate manner, the network can determine which of the features are critical for membership in a particular category.

The exploration of ART networks and HABclass networks provides a step toward automation of the classification task in a human-like manner. Although such networks are not as proficient as humans at creating such classifications

when the classification task involves a relatively small number of objects or features, these networks are more amenable to real-world, human-like classification tasks than many other classification algorithms. In addition, as the number of objects or features in the classification task increases, humans are less able to provide classifications in a timely manner and tools which can provide human-like classifications become more crucial.

Appendix A

O. volvulus DNA Sequences

LM1 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATGTAATTTCAAAAAA
CGGGT CGTACCTTCAAACTGGGTCCTAAGAAGAAAATTCGACCGAATTTTGG
LM2 AGTTTGCAAAATTAATTAACAACTGATGACCTATAACCCCTAATTTCAAGAAG
CGGATATGTACCGTCAAATTGAGTGCGAAAAAAAATATTCGACTATATTTTGG
LM3 AGTTTGCAAAATTAATCAATAACTGATGACCTATGATCCCTAATTTCAAGAAG
CGGATATGTACCGTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTGG
LM4 AGTTTGCAAAATTAATCAATAACTGATTACCTATGATTCCTAATTTCAAAAAA
CGGGTACGTACCTTCAAATTGAGTCCTAAGAAAAATATTCGACTATATTTCAA
LM5 AGTTTGCAAAATTAATTAATAACTGATAACCTATGACCCCTAGTATCAAAAAA
CGCTAACGTCCCTTCAAATTGAGTCCTAAAAAAAGTTTTCGCCTATATTTTTT
LM6 AGTTTTCAAAATTGCCTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA
CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG
LM7 AGTTTGCAAAATTAATCAATAACTGATGACCTATGATCCCTAATTTCAAGAAG
CGGATATGTACCGTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTAT
LM9 AGTTTGCAAAATTAATCAATAACTGATGACCTATGATCCCTAATTTCAAGAAG
CGGATATGTACCGTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTTT
LM10 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA
CGGGTACGTACCTTCAAATTGAGTCCTAAGAAAAATATTCGACTATATTTTGG

LM11 AATTTGCAAAATTAATCAATAACTGATTACCTATGATTCCTAATTTCAAGAAG

CGGATATGTACCGTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTGG

LM12 AGTTTGCAAAATTAATCAATAACTGATGAGCTGTGATTCCTAATTTCAAGAAA

CGGGTACATACCTTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTGG

LM13 AGTTTGCAAAATTAATCAATAACTGATGACCTATGATTCCTAATTTCAAGAAG

CGGATATGTACCGTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTGG

LM14 AGTTTGCAAAATTAACCAATAACTGATTACCTATGATTCCTAATTTCAAGAAG

CGGATATGTACCGTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTGG

LM15 AGTTTGCAAAATTAATCAATAACTGATTACCTATGATTCCTAATTTCAAGAAG

CGGATATGTACCGTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTGG

LM16 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAATTGAGTCCTAAGAAAAATATTCGACTATATTTTTT

LM17 AGTTTGCAAAATTAATCAATAACTGATGAGCTGTGATTCCTAATTTCAAGAAG

CGGGTACGTACCTTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTGG

LM18 AGTTTGCAAAATTAACTAATAACTGATGATCTATGACCCCTAATTTCAAAAAA

GGCTAACGTGCCTTCAAATTGAGTCCTAAAGAAAATATTCGTCTATATTTTGG

LM20 AGTTTGCAAAATTAATTAACAACTGATGACCTATGATCCCTAATTTCAAGAAA

CGGGTACATACCTTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTGG

LM21 AGTTTGCAAAATTAATTAACAACTGATTACCTATGATCCCTAATTTCAAGAAA

CGGGTACATACCTTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTGG

LM22 AGTTTGCAAAATTAATCAATAACTGATGACCTATGATCCCTAATTTCAAGAAG

CGGATATGTACCGTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTGG

LM23 AGTTTGCAAAATTAATCAATAACTGATGACCTATGATTCCTAATTTCAAGAAG

CGGATATGTACCGTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTGG

LM24 AGTTTGCAAAATTAATCAATAACTGATTACCTATGATTCCTAATTTCAAGTTG

CGGATATGTACCGTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTATGG

LM25 AGTTTGCAAAATTAATTAACAACTGATGACCTATGATCCCTAATTTCAAGAAA

CGGGTACATACCTTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTGG

LM26 AGTTTGCAAAATTAATTAACAACTGATGACCTATGATCCCTAATTTCAAGAAA

CGGGTACATACCTTCAAATTCAGTGCTAAAAAAAATATTCGACTATATTTTGG

LM27 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

LM28 AGTTTGCAAAATTAATCAATAACTGATTACCTATGATTCCTAATTTCAAGAAG

CGGATATGTACCGTCAAATTGAGTGCTAAAAAAAATATTCGACTATCTTTTGG

LM29 AGTTTGCAAAATTAACTAATAACTGATGATCTATGACCCCTAATTTCAAAAAA

CGGGTAAGTACCTTCAAATTGAGTCCTAAAAAAAGTTTTCGCCTATATTTTTT

LM30 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAAGATTCGACCATATTTTGG

LM31 AGTTTGCAAAATTAATCAATAACTGATGACCTATGATCCCTAATTTCAAGAAG

CGGATATGTACCGTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTGG

LM32 AGTTTGCAAAATTAATCAATAACTGATGACCTATGATCCCTAATTTCAAGAAG

CGGATATGTACCGTCAAATTGAGTGCTAAAAAAAATATTCGTCTATATTTTGG

LM34 AGTTTGCAAAATTAATCAATAACTGATTACCTATGATCCCTAATTTCAAGAAG

CGGATATGTACCGTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTGG

LM36 AGTTTGCAAAATTAATTAACAACTGATGACCTATGATCCCTAATTTCAAGAAA

CGGGTACATACCTTCAAATTGAGTGCTAAAAAAA TGTTCGACTATATTTTAC

LM37 AGTTTGCAAAATTAATCAATAACTGATGAGCTGTGATTCCTAATTTCAAGAAG

CGGATATGTACCGTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTGG

LM39 AGTTTTCAAAATTGCCTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

LM40 AGTTTGCAAAATTAATCAATAACTGATGACCTATGGTCCCTAATTTCAAGAAG

CGGATATGTACCGTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTGG

LM41 AGTTTTCAAAATTGCCTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

LM42 AGCTTGCAAAATTAACTAATAACTGATGATCTATGACCCCTAATTGCAAAAGA

CGCTAACGTGCCTTCAAACTGAGTCCTAAAAAAAGTTTTCGACTATATTTTCT

LM43 AGTTTGCAAAATTAATCAATAACTGATGACCTATGATCCCTAATTTCAAGAAG

CGGATATGTACCGTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTGG

LM44 AGTTTGCAAAATTAATCAATAACTGATTACCTATGATTCCTAATTTCAAGCCG

CGGATATGTACCGTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTGG

MM1 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAAGATTCGACCATATTCTGG

MM2 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAATTGAGTCCTAAGAAAAATATTCGACTATATTTTTT

MM3 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAAGTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTC

MM4 AGTTTTCAAAATTGCCTAATAAC TGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTTAGAAAAATATTCGACCATATTTTGG

MM5 AGTTTTCAAAATTGCCTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACTATATTTTGG

MM6 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCCAAGAAAAATATTC

MM7 ATTTTGCAAAATTGATTATTAACAGATGACCTATCTGATATAATTTCAAAAAA

CGGGTACGTACCTTCAAATTGAGTCCAAGAAAAATATTCGAA

MM36 AGTTTGCAAAATTAACTAATAACTGATGATCTATGACCCCTAATTTCAAAAAA

CGGGTAAGTACCTTCAAATTGAGTCCTAAAAAAAGTTTTCGCCTATATTTTTT

MM37 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

MM38 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATGTTCGACTATATTTTTT

MM39 AGTTTTCAAAATTGCCTAATAACTGATGTCCTATGT CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

MM40 ATTTTGCAAAATTGATTATTAACGGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAATTGAGTCCTAAGAAAAATATTCGACTATATTTTTT

EB1 AATTTTCAAAATTGCCTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTAGG

EB2 ATTTTGCAAAATTGATTATTAACAGATGACCTACGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

EB3 AGTTTTCAAAATTGCTTAATAACTGATGACCTGTGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

EB4 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

EB5 AATTTTCAAAATTGCCTAATAACTGATGACCTATGA CCCTATTCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

EB6 AGTTTTCAAAATTGCCTAATAATTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

EB7 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGAGTCCTAAGAAAAATATTCGACTATATTTTTT

EB8 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

EB9 AGTTTTCAAAATTGCTTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CATACA TTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

EB10 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATGTAATTTCAAAAAA

CGGGTACGTACCTTCAAATTGAGTCCCAAAAAATATATTCGACTATATTTTCG

EB11 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAGA TTCGACCATATTTTGG

1F01 AATTTTCAAAATTGCCTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTAGG

1F02 ATTTTGCAAAATTGATTATTAACAGATGACCTATGTCATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAAGATTCGACCATATTTTGG

1F03 AGTTTGCAAAATTAATCAATAACTGATTACCTATGATTCCTAATTTCAAGAAG

CGGATATGTACCGTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTAT

1F04 ATTTTGCAAAATTGATTATTAACAGATGACCTATGGCATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAAGATTCGACCATATTTTGG

1F05 AATTTTCAAAATTGCCTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

1F06 AGTTTTCAAAATTGCCTAATAATTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

1F07 AGTTTTCAAAATTGCCTAATAATTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGATCCTAAGAAAAATATTCGACCATATTTTGG

1F08 AGTTTGCAAAATTAATCAATAACTGATTACCTATGATTCCTAATTTTAAGAAG

CGGATATGTACCGTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTAT

1F09 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

1F10 AATTTTCAAAATTGCCTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CGAGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

1F11 AGTATGCAAAATTAATTAACAACTGATGACCTATGATCCCTAATTTCAAGAAA

CGGGTACATACCTTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTGG

1F12 AGTTTGCAAAATTAATCAATAACTGATTACCTATGATTCCTAATTTCAAGAAG

CGGATATGTACCGTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTAT

1F15 ATTTTGCAAAATTGATTATTAACAGATGACCTATGTCATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACCATATTTAGG

1F17 AGTTTGCAAAATTAATTAACAACTGATGACCTATGATCCCTAATTTCAAGAAA

CGGATGCATACCTTCAAATTGAATGCTAAAAAAAATGTTCGACTATATTTTAC

1F18 AATTTTCAAAATTGCCTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

GCCCTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTAGG

1F20 AGTTTGCAAAATTAATTAACAACTGACGACCTATGATCCCTAATTTCAAGAAA

CGGGTACATACCTTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTGG

1F22 AATTTTCAAAATTGCCTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAACCGGGTCCTAAGAAAAATATTCGACCATATTTAGG

1F24 AGTTTGCAAAATTAATCAATAACTGATTACCTATGATTCCTAATTTCAAGAAG

CGGATATGTACCGTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTAT

1F25 ATTTTGCAAAATTGATTATTAACAGATGACCTATGTCATATAATTTCAAAAAA

CGGGTACATACCTTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTGG

1F26 AGTTTTCAAAATTGGTTAATAACTGATGACCTGTGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCTATTTTTGG

1F28 TGAAAGCAAAATTAATTAACAACTGATGACCTATGATCCCTAATTTCAAGAAA

CGGGTACATACCTTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTAT

ZM4 ATTTTGTAAAATTGATTATTAACAGATGACCTATGCCATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAAGATTCGACCATATTTTGG

ZM5 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGTAAAAGATTCGACCATATTTTGG

ZM6 AGTTTTCAAAATTGCTTAATAACTGATGACCTATGACATATAATTTCAAAAAA

CAGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATGTTCGACTATATTTTTT

ZM7 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGAGTCCTAAGAAAAATATTCGACTATATTTTTT

ZM8 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAATTGAGTCCTAAGAAAAATATTCGACTATATTTTTT

ZM11 AGTTTGCAATACTAATTAATAACTGATGACCTTTGACCCCTAATTTCAAAAAA

CGGGTACGTACCTTCAAATTGAGTCCTAAAATAAATATTCGACTATATCTTGG

ZM12 AGTTTGCAAAATTAATTAATAACTGGTGACATTTGACCCCTGATTTCACGAAA

GGGGTACGTGCTTTCAAATTGAGTCCTAAAAAAAATATTCGACTATATTTGGT

ZM14 AGTTTGCATAATTAATTAATAACTGATGACCTATGATCCCTAATTTCCAAAAA

CGGGAACGTACCCTCAAATTTAGTCCTGAAAAAAATATTCGATTATATTTTAA

ZM15 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

ZM16 AGTTTTCAAAATTGCCTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

ZM17 AGTTTTCAAAATTGCCTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCGTATTTTGG

ZM18 ATTTTGCAAAATTGATTATTAACAGACGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTGCACTATATTTTTT

ZM19 AGTTTTCAAAATTGCCTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTTAGAAAAATATTCGACCATATTTTGG

ZM20 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

ZM21 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

ZM22 AGTTTGCATAATTAATTAATAACTGATGACCTATGATCTCTAATATCAAGAAA

CGGGTACATACCCTCAAATTGAGTGCTAAAAAAAATGCTCGACTATATTTTGG

BM1 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGAGTCCTAAAAAAAGTTTTCGACTATATTTTTT

BM2 AATTTTCAAAATTGCCTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

BM3 AATTTTCAAAATTGCCTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

BM4 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGAGTCCTAAAAAAAGTTTTCGACTATATTTTTT

BM5 AATTTTCAAAATTGCCTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACGTACCTTCAAACAGGGTCCTAAGAAAAATATTCGACTATATTTTTT

BM6 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAGAAGATTCGACCATATTTTAA

BM7 ATTTTGCAAAATTGATTATTAACAGATGACCTACGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

BM9 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

BM11 GTTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGAGTCCTAAAAAAAGTTTTCGACTATATTTTTT

BM12 GTTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

BM13 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGAGTCCTAAAAAAAGTTTTCGACTATATTTTTT

BM14 AATTTTCAAAATTGCCTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTAGG

BM15 ATTTTGCAAAATTGATTATTAACAGATGACCATAGACATGTAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAGAAGATTCGACCATATTTTGG

BM16 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

BM17 GTTTTC AAAATTGCCTAATAATTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

BM19 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGAGTCCTAAAAAAAGTTTTCGACTATATTTTTT

BM20 AATTTTCAAAATTGCCTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

BM22 AATTTTCAAAATTGCCTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTAGG

BM23 AGTTTTCAAAATTGCTTAATAACTGATGACCTGTGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

BM24 AATTTTCAAAATTGCCTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

GM1 AGTTTTCAAAATTGCCTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTTAGAAAAATATTCGACCATATTTTGG

GM2 AGTTTGCAAAATTAATTAACAACTGATGACCTATAACCCCTAATTTCAACAAA

CGGGTACATACCTTCAAATTGAGTGCTAAAAAAAATGTTCGACTATATTTTCT

GM3 AGTTTTCAAAATTGCCTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTTG

GM4 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

GM5 AGTTTGCAAAATTGACTAATAACTGATGAGCTATGACCCCTAATTTCAAAAAA

CGCTAACGTACCTTCAAACTGAGTCCTAAAAAAAGTTTTCGACTATATTTTCT

GM7 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

GM9 ATTTTGCAAAATTGATTATTAACAGATCACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAAGATTCGACCATATTTTGG

GM10 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAATTGAGTCCTAAGAAAAATATTCGACTATATTTTTT

GM12 AGTTTTCAAAATTGCTTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

GM13 AGTTTTCAAAATTGCCTAATAACTGATGACCTACGA TCTCAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

GM14 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATGTTCGACTATATTTTTT

GM15 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAATTGAGTCCTAAGAAAAATATTCGACTATATTTTTT

GM16 AGTTTCGAAAATTAATCAATAACTGATTACCTATGATTCCTAATTTCAAGAAG

CGGATATGTACCGTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTGG

GM17 AGTTTTCAAAATTCGTTAATAACTGATGACCTATGA CCCTAATCTTAAAAAA

C ACACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

GM19 AGTTTTCAAAATTGCCTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTTAGAAAAATATTCGACCATATTTTGG

GM20 AGTTTTCAAAATTGCTTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

C ATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

GM21 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAATTAGATCCTAAGAAAAATATTCGACTATATTTTTT

GM22 AGTTTTCAAAATTGCCTAATAACTGATGACTTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

GM24 AGTTTTCAAAATTGCCTAATAATTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

GD1 AGTTTTCAAA TTGCCTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

GD2 AGTTTTCAAAATTGCCTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

GD4A AATTTGCAAAATTAATTAACAACTGATGACCTATAACCCCTAATTTCAAGAAA

CGGGTACATACCTTCAAATTGAGTGCTAAAAAAAATATTCGACTATATTTTGG

GD4B AGTTTGCAAAATTAATTAACAACTGATGACCTATGATCCCTGATTTCAAGAAA

CGGGTACATACCTTCAAATTGAGTGCTAAAAAAATGTTCTACTATATTTTATT

GD5A ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

GD5B AGTTTTCAAAATTGCCTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAGATATTCGACCATATTTTGG

GD6A AGTTTTCAAAATTGCTTAATAACTGATGACCTATGA CCCTAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

GD6B AGTTTTCAAAATTGCCTAATAACTGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAAGATTCGACCATATTTTGG

GD9A ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAATTGAGTCCTAAGAAAAATATTCGACTATATTTTTT

GD9B ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAATTGAGTCCTAAGAAAAATATTCGACTATATTTTTT

GD90 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

GD91 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

1-51 AGTTTTCAAAATTGCCTAATAACTGATGACCTATGACCC TAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

5-6 AGTTTTCAAAATTGCCTAATAACTGATGACCTATGACCC TAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

4-7 AGTTTTCAAAATTGCCTAATAACTGATGACCTATGACCC TAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

5-10 AGTTTTCAAAATTGCCTAATAACTGATGACCTATGACCC TAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

5-6 AGTTTTCAAAATTGCCTAATAACTGATGACCTATGACCC TAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

2-13 AGTTTTCAAAATTGCCTAATAACTGATGACCTATGACCC TAATCTCAAGAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

3-10 AATTTTCAAAATTGCCTAATAACTGATGACCTATGACCC TAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

4-19 AGTTTTCAAAATTGCCTAATAACTGATGACCTATGACCC TAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTTAGAAAAATATTCGACCATATTTTGG

2-9 AGTTTTCAAAATTGCCTAATAACTGATGACCTATGACCC TAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCGTATTTTGG

7-12 A TTTTCAAAATTGCCTAATAACTGATGACCTATGACCC TAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

7-20 AGTTTTCAAAATTGCCTAATAACTGATGACCTTAGACCC TAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

2-1 AGTTTTCAAAATTGCCTAATAACTGATGACCTATGACCC TAATCTCAAAAAA

CGGGTACATACATTCGAACTGGGTCCTAAGAAAAAGATTCGACCATATTTTGG

1-2 A TTTTCAAAATTGCCTAATAATTGATGACCTATGACCC TAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

1-8 A TTTTCAAAATTGCCTAATAACTGATGACCTATGACCC TAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATAATTTGG

3-7 AATTTTCAAAATTGCCTAATAACTGATGACCTATGACCC TAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTAGG

3-5 AGTTTTCAAAATTGCCTAATAACTGATGACTTATGACCC TAATCTCAAAAAA

CGGGTACATACATCCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTAGG

4-4 TGTTTTCAAAATGGCTTAATAACTGATGACCTATGACCC TAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

4-1 AGTTTTCAAAATTGCCTAATAACTGATGACCTATGACCC TAATCTCAAAAAA

CGAATACATACATTCGAATTGGGTCCTTAGAAAAATATTCGACCATATTTTGG

5-9 AGTTTTCAAAATTGCCTAATAACTGATGACCTATGACCC TAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

4-12 AGTTTTCAAAATTGCTTAATAACTGATGACCTATGACCC TAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTGGACCATAGTTTGG

5-2 AATTTTCAAAATTGCCTAATAACTGTAGACCTATGACCC TAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAACTATTCGACCATATTTTGG

3-3 AGTTTTCAAAATTGCTTAATAAATGATGACCTATGACCC TAATCTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTGGACCATAGTTTGG

4-17 AGTTTTCAAAATTGCTTAATAACTGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

4-2 AGTTTTCAAAATTGCTTAATAACTGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

3-8 AGTTTTCAAAATTGCTTAATAACTGATGACCTATGACATATAATTTCAAAAAA

CAGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATGTTCGACTATATTTTTT

5-13 AGTTTTCAAAATTGCTTAATAACTGATGAGCTATGACATATAATTTCAAAAAA

CAGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

5-18 AGTTTTCAAAATTGCTTAATAACTGATGACCTATGACATATAATTTCAAAAAA

CAGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATGTTCGACTATATTTTTT

1-7 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

1-20 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

5-20 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

3-6 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

5-8 AGTTTTCAAAATTGATTATTAACAGATGACCTATGACCTATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAAATAAATATTCGACTATATTTTTT

4-16 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATGTTCGACTATATTTTTT

7-17 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGATTATATTTTTT

1-6 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGCACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

3-1 ATTTTGCAAAATTGATTATTAACAGATGACCTATGTCATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTTT

4-13 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAATTGAGTCCTAAGAAAAATATTCGACTATATTTTTT

1-17 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAATTGAGTCCTAAGAAAAATATTCGACTATATTTTTT

1-18 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAATTGAGTCCTAAGAAAAATATTCGACTATATTTTTT

5-4 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAATTGAGTCCTAAGAAAAATATTCGACTATATTTTTT

7-19 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAATTGAGTCCTAAGAAAAATATTCGACTATATTTTTT

2-5 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAATTGAGTCCTAAGAAAAATATTCGACTATATTTTTT

2-4 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAAAATATTCGACTATATTTTGG

1-15 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAATTGAGTCCTAAGAAAAATATTCGACTATATTTTTT

4-9 ATTTTGCAAAATTGATTATTAACAGATGACCTACGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAATTGAGTCCTAAGAAAAATATTCGACTATATTTTTT

1-4 ATTTTGCAAAATTGATTATTAACAGATGGCCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGGTCCTAAGAAGAAGATTCGACCATATTTTGG

1-5 ATTTTGCAAAATTGATTATTAACAGATGGCCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGGG CCTAAGAAGAAGATTCGACCATATTTTGG

1-9 ATTTTGCAAAATTGATTATTAACAGATGACCTATGAC TATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGAGTCCTAAAAAAAGTTTTCGACTATATTTTGG

1-10 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGAGTCCTAAAAAAAGTTTTCGACTATATTTTCT

5-15 AGTTTGCAAAATTATTTAATAACTGATGTCCTATGATCCCTAATTTCAAGAAA

CGGGTACGTTCCTTCAAATTGGGTCCTAAAATAAATATTCGACTCTATTTTGG

5-19 AGTTTGCATAATTAATTAATAACTGGTGACCTATGATCTCTAATTTCAAGAAA

CCGGTACGTTCCTTCAAATTGGGTGCTAAAATAAATATTCGACTCTATTTTGG

5-5 AGTTTACAAAATTAATTAATAACTGATCACCTATGACCCCTAACTGCAAAAAA

CGGGTACGTACCTTCAAATTGAGTCCAAAAGAAAATATTGGACTATATTTTGG

5-17 AGTTTGCATAGTTAATTAATATCTGATGACCTATGACCCCTAACTTCAAAAAA

CGGATACGCACCTTCAAATTGATTCCAAAAGAAAATATTGGACTATATTTTGG

5-12 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAATTGAGTCCAAAAGAAAATATTGGACTATATTTTGG

4-3 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACATACATTCGAATTGGGTCCTAAGAAAAATATTCGACCATATTTTGG

4-8 AGTTTGCATAATTAATTAATAACTGGTGACCTATAATCTCTAATTTCAAGAAA

CGGGTACATACACTCAAATTGAGTGCTAAAAAAAATATTCAACTATATTTTGG

4-10 AGTTTGCATAATTAATTAATAACTGGTGACATATGACCCCTAATTTCACGAAA

GGGGTACGTGCTTTCAAATTGAGTCCTAAAAAGAATATTCGACCATATTTTGG

5-3 AGTTTGCAAAATTAATTAATAACTGATGACCTATGACCCCTAATTTCAAAAAA

CGGTTACGTACCTTCAAATTGGGTCCTAGGAAAAATATTCGAAAATATTTTGG

4-6 AGTTTTCAAAA AACTAATAACTGATGATCTATGACCCCTAATTTCAAAAAA

CGCTAACGTACCTTCAAACTGAGTCCTAAAAAAAGTTTTCAACTATATTTTGG

5-7 ATTTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGATCGATCCTTCAAACTGGGTCCTAAGAAAAATATTCGACATATTTTTGG

1-19 AATTTGCAAAATTGATTATTAACAGATGACCTATGACATATAATTTCAAAAAA

CGGGTACGTACCTTCAAACTGAGTCCTAGAAAAATATTCGACTATATTTTTTT

Appendix B
Description of the *O. volvulus* Data

The data listed in the previous appendix is the set of DNA fragments taken from the human parasite, *O. volvulus*. Each fragment is labelled with a descriptor in the first three or four positions for the line.

For the first 150 sequences, the descriptor indicates the geographic location from which this fragment was sampled. Individuals beginning with `Z, M,` or `E` are from the Savannah region of Africa, those beginning with `B` are from Brazil, those beginning with `L or 1` are from the Forest region of Africa, and those beginning with `G` are from Guatemala.

For the remaining sequences, the descriptor indicates the date on which the fragment was sampled. For example, the sequence labelled 1-15 was sampled on January 15 but the descriptor does not give an indication of the geographic region from which the sequence originated.

Each sequence fragment consists of 106 bases taken from the intergene region *O. volvulus*. The sequences have been aligned in some cases. Such alignment is relatively easy since most of the sequences have very few skips and none of the sequences have any insertions.

The DNA sequences have been deposited in GENBANK under accession number U02590 to U02594 and U02731 to U02875 [22].

Appendix C
Changes Made to the ART2 Software

The ART2 software was obtained via anonymous ftp from park.bu.edu. Several changes were required to allow the use of the software in this problem domain. The changes are described below.

C.1 Changes to art2.c

Throughout the program, I removed all `fprintf(flog,...)` statements because such statements slowed execution and used a large amount of storage space.

C.1.1 Changes to function do_cycle

1. Moved the following code from outside the first i loop to immediately inside the i loop. The result of this change is that a reset of an F2 node will only last for the current input pattern rather than for the entire learning trial.

```
for (j=0; j<N-M; j++) /* Initially, no mismatches yet! */
    mis[j]=0;
n_mis=0;
```

2. Added the following code in the second i loop of the function in order to write information about where patterns are coded to the output file.

```
if (i == 0)
```

```
        {
          fprintf(fout,"\nrho = %lf\n",rho);
          fprintf(fout,"theta = %lf\n",theta);
        }
fprintf(fout,"Pattern %s is coded at F2 node %d\n",mem[c_p[i]],
IL[c_p[i]]);
```

3. Removed the code at the end of the function to print weights during learning to output file primarily because of the slow down in execution time this caused.

C.1.2 Changes to function BU_reset

1. Added the following code in order to be able to load the initial bottom-up weights from a file.

Declarations:

```
char response[1];
char buname[25];
char filename[80];
FILE *ifp;
```

First thing in function:

```
printf("\n Load bottom up weights from a file (y or n)? ");
scanf("%s",response);
fflush(stdin);
if (strcmp(response,"y") == 0)
{
```

```
    printf("\n Name of the file ---> ");
    scanf("%s",buname);
    fflush(stdin);
    strcpy(filename,PATH);
    strcat(filename,buname);
    ifp = fopen(filename,"r");
  }
```

In the j loop, before the weight assignment statement:

```
    {
       if (strcmp(response,"y") == 0)
       {
         printf(".");
         fscanf(ifp,"%lf",&Zij[i][j]);
       }
       else
```

Immediately after the weight assignment statement:

```
    }
  if (strcmp(response,"y") == 0)
    fclose(ifp);
```

C.1.3 Changes to function TD_reset

1. Added the following code in order to be able to load initial top-down weights from a file.

Declarations:

```
char response[1];
char tdname[25];
char filename[80];
FILE *ifp;
int temp;
```

First thing in function:

```
printf("\n Load top down weights from a file (y or n)? ");
scanf("%s",response);
fflush(stdin);
if (strcmp(response,"y") == 0)
{
  printf("\n Name of the file ---> ");
  scanf("%s",tdname);
  fflush(stdin);
  strcpy(filename,PATH);
  strcat(filename,tdname);
  ifp = fopen(filename,"r");
  fscanf(ifp,"%d",&temp);
  fscanf(ifp,"%d",&temp);
}
```

In the i loop, immediately before the weight assignment statement:

```
    {
       if (strcmp(response,"y") == 0)
       {
         printf(".");
         fscanf(ifp,"%lf",&Zji[j][i]);
       }
       else
```

Immediately after the weight assignment statement:

```
    }
  if (strcmp(response,"y") == 0)
    fclose(ifp);
```

C.1.4 Changes to function update_Zs

1. Took the advice of the main header comment and used the following two lines in the i loop:

```
zt1=d*(1.0-d);
zt2=U[i]/(1.0-d);
```

rather than these lines:

```
zt1=d;
zt2=P[i];
```

The use of the first two lines will avoid the appearance of "dips". See the main header comment of ART2.c.

C.1.5 Changes to function print_weights

I changed the purpose of this function. In the original ART2 software, its purpose was to write the weights to the .dat file in order to follow the progress of the algorithm. The new purpose of this function is to save a set of weights that has been learned using a set of patterns.

1. Added the following code.

Declarations:

```
char bufile[25];
char tdfile[25];
char filename[80];
FILE *ofp1;
FILE *ofp2;
```

2. First thing in function, I changed

```
fprintf(fout,"\t\tBottom up weights \n\n");
```

To:

```
printf("Name of bottom up weight file ---> ");
scanf("%s",bufile);
fflush(stdin);
printf("Name of top down weight file ---> ");
scanf("%s",tdfile);
fflush(stdin);
strcpy(filename,PATH);
```

```
    strcat(filename,bufile);
    ofp1 = fopen(filename,"w");
    strcpy(filename,PATH);
    strcat(filename,tdfile);
    ofp2 = fopen(filename,"w");
    fprintf(ofp2,"%d\n%d\n",M,N-M);
```

3. In the first j loop, I changed

```
          fprintf(fout,"%5.4lf ",Zij[i][j]);
fprintf(fout,"\n");
```

To:

```
          fprintf(ofp1,"%5.4lf ",Zij[i][j]);
        fprintf(ofp1,"\n");
```

4. In the second i loop, I changed

```
  fprintf(fout,"%5.4lf ",Zji[j][i]);
fprintf(fout,"\n");
```

To:

```
    fprintf(ofp2,"%5.4lf ",Zji[j][i]);
fprintf(ofp2,"\n");
```

5. At the end of the function, I added:

```
fclose(ofp1);
fclose(ofp2);
```

C.2 Changes to in_vec.c

In the global declarations, I added the following code in order to allow the reading in of the pattern (member) names.

```
char **mem;
```

C.2.1 Changes to function get_inputs

1. After the if statement for I[i] but before the if statement for p_mag, I added the following code to read in member name.

```
    if ((mem=(char **) calloc(p_num,sizeof(char *))) == NULL)
    {
fprintf(stderr,"Error: could not allocate
               %d %d-D patterns!\n",p_num,p_dim);
exit();
    }


    for (i=0; i<p_num; i++)
      if ((mem[i]=(char *) calloc(5,sizeof(char))) == NULL)
      {
  fprintf(stderr,"Error: could not allocate
                %d %d-D patterns!\n",p_num,p_dim);
  exit();
      }
```

2. In the i loop to get the patterns, immediately following the for statement, I added the following code.

```
if (fscanf(datain,"%s",mem[i]) == EOF)
{
   fprintf(stderr,"Error: Unexpected End-of-File
           in %s\n",in_name);
   exit(1);
}
```

C.2.2 Changes to function select_cycle

1. Before the loop to individually select patterns for cycling, I added the following code to allow the selection of all the loaded patterns automatically.

```
if (ppc == p_num)
   {
      for (i=0; i<ppc; i++)
         c_p[i] = i;
   }
else
   {
```

C.3 Changes to in_vec.h

1. Added the following declarations to allow reading in of the member name, the isolate name and the cluster name.

```
extern char **mem, **isol, **clus;
```

Appendix D

Other Software used in the *O. volvulus* Application

The additional software needed for the tests described in this thesis is listed below. The software is available at `cs.fsu.edu` in the subdirectory `leblanc/pub/`.

D.1 project.c

This is the main piece of supporting software for this project. It is a menu-driven program whose main menu looks as follows:

```
Input file has 210 patterns
WORM DATA SYSTEM
1. Display array of raw data
2. Compare two strings
3. Create file of only certain categories
4. Preprocess data for input into ART2
5. Create file for new vigilance level run of ART2
6. Run the art2 program
7. Create an evaluation file from art2 output
            for Katholi's software
8. Calculate within class variation using evaluation file
9. Run the clustering algorithms on processed data
10. Exit
```

```
Enter your choice ---->
```

The purposes of the options are:

1. After loading the initial raw sequences, that is, in nucleotide format, display a particular sequence. The output will be displayed on the screen.

2. Compare two sequences to determine how many differences there are between them. The comparison may be done in either raw form or processed form. The output will be displayed on the screen and may optionally be stored in a file.

3. Create a file that contains only a subset of the total number of sequences. For example, one may want to work only with the labeled sequences from Brazil. The sequences may be retrieved in either raw form or processed form and will be stored in a file.

4. Change a file containing sequences in raw form into a set of sequences in processed (0s and 1s) form and store the processed sequences in a file.

5. Create a file containing only a certain cluster from an execution of ART2. For example, one may determine that after the first run of ART2, cluster 0 should be further refined. This option will create a file with only the sequences that fell into that initial cluster 0 to be sent through a second-layer ART2 network.

6. Execute the art2.c program to run the art2 program (with modifications as described in Appendix C.

7. This option uses a program stored in evalfile.c. Use the output file from a run of one of the clustering algorithms to create a file that can be used to evaluate how good the clustering is. The evaluation file can be sent through the oneway.for program created by Dr. Charles Katholi (if attempting to find within class variation, among class variation and degrees of freedom) or can be

sent through option 8 of this program, which will calculate only the within class variation. The tests in this thesis used the within class variation calculated through option 8 of this program.

8. This option uses a program stored in calcM.c. Use the evaluation file created in option 7 to calculate the within class variation for each cluster and for the overall set of clusters. The results will be displayed to the screen and will be placed in an output file.

9. Run the alternative clustering algorithms. This option displays an additional menu shown and described below.

```
Clustering Algorithms
1. Run the leader algorithm for clustering
2. Run the Kmeans algorithm for clustering
3. Run the leader algorithm for classification
4. Run the Kmeans algorithm for classification
10. Return to main menu
Enter your choice ---->
```

The purposes of these options are:

1. Run the leader algorithm to create a set of clusters of whatever data was last preprocessed using option 4 of the main menu. The user must specify a threshold value, whether the simple leader or the modified leader algorithm should be used, and whether Euclidean or vector distance should be used. The resulting clusters are displayed on the screen as well as stored in a file. Optionally, the leaders of each cluster can be stored in a file for later use in classification of new sequences.

2. Run the Kmeans algorithm to create a set of clusters of whatever data was last preprocessed using option 4 of the main menu. The user must specify the number of clusters into which the sequences should be placed. Euclidean distances are used in this function. The resulting clusters are displayed on the screen as well as stored in a file. Optionally, the means of each cluster can be stored in a file for later use in classification of new sequences.

3. Use a file of leaders created in option 1 of this menu to classify into the existing clusters whatever data was last preprocessed using option 4 of the main menu. The resulting clusters are displayed on the screen as well as stored in a file.

4. Use a file of means created in option 2 of this menu to classify into the existing clusters whatever data was last preprocessed using option 4 of the main menu. The resulting clusters are displayed on the screen as well as stored in a file.

Appendix E
Instructions for using Additional Software

To recreate the tests described in this thesis, use the following steps. All programs and data are available in the subdirectory *leblanc/pub* at *cs.fsu.edu*.

The data originally appears in raw form, that is, as a label followed by a sequence of nucleotides. The file containing all the data described in this thesis is contained in *old+new.inp*.

E.1 Necessary Software Files

The following source code files are necessary to recreate all of the tests described in this thesis:

1. art2.c
2. art2.h
3. calcM.c
4. evalfile.c
5. in_vec.c
6. in_vec.h
7. project.c
8. projstr.h
9. rk4d.c
10. rk4d.h
11. std_fun.c

12. std_fun.h

A make file exists to compile all of the above except calcM.c, evalfile.c, and project.c. These three programs should be compiled using an ANSII C compiler.

E.2 Clustering Using HABclass Networks

1. Begin execution of project.c executable file. The program will ask for the name of an input file. Any file can be used that contains raw input data. An example of such a file is the old+new.inp file described above.

2. If only a subset of the total number of sequences in the input file chosen in step 1 are desired for additional processing, choose option 3 from the main menu. Pattern names or isolate and cluster names can be used to select sequences. Pattern names should be used if sequences are to be selected one at a time. This option will create two files: filename.out and filename.out2 where filename is supplied by the user. The file filename.out contains the actual sequences chosen while filename.out2 contains the length of the sequences followed by the number of sequences included in filename.out.

3. To be used in any of the clustering algorithms, the sequences must be processed into a set of 0s and 1s. Choose option 4 from the main menu to accomplish this. The function will look for a file called filename.ext, supplied by the user, which should contain the raw sequences to be processed as well as filename.ext2 which should contain the length of the sequences followed by the number of sequences included in filename.ext. The function creates a file called filename2.inp, where filename2 is supplied by the user. The file filename2.inp

will contain the length of the raw sequences followed by the number of sequences in the file. These two numbers are followed by the raw sequences themselves, each of which will contain a label specifying the name by which the sequence is known.

4. The ART2 software may be used on any file containing processed data (as described in step 3 above). An input file name (of the file containing the processed sequences whose extension is .inp) and an output file name (which will get a file extension of .dat) are requested. The user is then asked whether bottom-up weights are to be loaded from a file. If the answer is yes, the user is prompted for the name of this file. Note that if a weight file from a previous ART2 run is to be used, at the bottom of the input file (with the processed data), there should be two numbers specifying the size of the network for which the weights were created. These numbers should be, first, the number of input nodes for the network and, second, the number of input nodes plus the number of category nodes for the network. If these numbers are not in the input file, the ART2 software uses as a default the length of the sequences and the length of the sequences plus the number of sequences.

The user is then asked whether the top-down weights are to be loaded from a file. If the answer is yes to this question, the same note as for the bottom-up file applies.

The ART2 menu is then displayed. The user must select some patterns to be included in the learning run of ART2. Option 2 allows this. The software will request the number of patterns to be selected. If the number selected is different than the number of sequences in the input file, the user must specify

which of the patterns are to be included by choosing them one by one.

If the user wishes to modify the vigilance parameter (or any of the other parameters described in the thesis), option 5 should be chosen.

Once all patterns are selected and all parameters are set as desired, the user should choose option 4 to begin the creation of clusters for the data. The program will display a series of messages to the screen as it is creating the clusters. After all clusters are created, the names of the patterns along with the number of the cluster it was placed into will be displayed to the screen and written to the output file specified by the user at the beginning of execution.

The input data should probably be cycled through the algorithm at least one more time so that if the earlier sequences (in the presentation order) are to jump to some of the later created clusters, they have the opportunity to do so.

After learning has occurred on the selected patterns, the bottom-up and top-down weights may be saved by selecting option 6 (initialize weights). This will display a menu of options to be executed on the weights. To save the weights to a file, choose option 7 (print weights).

ART2 will create a file containing the nodes at which each pattern has been coded. The file will be named filename.dat (where filename was specified by the user). This file will contain a list of patterns for each separate run of ART2. Therefore, if this file is to be used in later options, the .dat file should be editted to contain only one run of the software.

5. Any .dat file which contains only one run of a clustering algorithm can be used to create a file for evaluation of the within class variation. Choose option 7 from the main project.c program menu to create such a file. To use

this option, the user must know how many clusters were created as well as the number of patterns in the file. This option will create a file called filename.eva where filename is specified by the user.

6. To calculate the within class variation, choose option 8 from the main project.c program menu. The user must specify the name of the evaluation file created in option 7 as well as the name of the processed data file. This option will create a file called filename.cal where filename is specified by the user. This file will contain the within class variation for each cluster as well as the overall within class variation for the entire set of clusters.

E.3 Clustering Using Alternative Algorithms

1. To use other clustering algorithms, choose option 9 from the main project.c program menu. To use this option, the user must first, during this run of the project.c program, have run the preprocessing option (option 4). This option will display a second menu, which lists the various clustering and classification options available.

2. To run either the simple or modified leader algorithm for clustering, choose option 1 from the clustering algorithm menu. The user will be asked a series of questions, after which, the patterns in the previously preprocessed file will be clustered. A file called filename.ldr.dat will be created, where filename is specified by the user. This .dat file may be processed in a variety of ways, as described in points 5 and 6 of Section E.2 above.

3. After running the leader algorithm for clustering, the user also has the option to create a file containing only the leaders of the clusters. This leader

file can later be used for classification of new patterns. The leaders will be put into a file called filename.leaders where filename is specified by the user. If the user would like to use these leaders for classification, option 3 from the clustering algorithms menu should be chosen. This option will create a file called filename.class.ldr.dat where filename is specified by the user. Again, this .dat file can be processed as described in points 5 and 6 of Section E.2.

4. Rather than using the leader algorithms for clustering, the user may use the K-means algorithm. Option 2 from the clustering algorithms menu should be chosen to execute this algorithm. The user must specify a number of clusters and a file called filename.kme.dat, where filename is specified by the user will be created. The file will contain the clustering of the specified patterns and can be processed as described in points 5 and 6 of Section E.2 above.

5. The user may choose to save the means for each of the clusters created using the K-means algorithm for clustering. These files will be saved in a file called filename.means where filename is specified by the user. If the user would like to use these means to later classify new patterns, option 4 from the clustering algorithms menu should be chosen. A file called filename.class.kme.dat will be created where filename is specified by the user. This file will contain the clustering for the new patterns using the means previously determined and can be processed as described in points 5 and 6 of Section E.2 above.

Bibliography

[1] Gail A. Carpenter and Stephen Grossberg. (1988). The ART of adaptive pattern recognition by a self-organizing neural network. *IEEE Computer*, **3**(3), 77–88.

[2] Gail A. Carpenter and Stephen Grossberg. (1990). ART3: Hierarchical search using chemical transmitters in self-organizing pattern recognition architectures. *Neural Networks*, **3**,129–152.

[3] Gail A. Carpenter and Stephen Grossberg. (1991). ART2: Self-organization of stable category recognition codes for analog input patterns. In Gail A. Carpenter and Stephen Grossberg, (Eds.), *Pattern Recognition by Self-organizing Neural Networks*, (pp. 397–423). Cambridge, Massachusetts: MIT Press.

[4] Gail A. Carpenter and Stephen Grossberg. (1991). A massively parallel architecture for a self-organizing neural pattern recognition machine. In Gail A. Carpenter and Stephen Grossberg, editors, *Pattern Recognition by Self-organizing Neural Networks*, (pp. 313–382). Cambridge, Massachusetts: MIT Press.

[5] Gail A. Carpenter and Stephen Grossberg. (1992). *Neural Networks for Vision and Image Processing*. Cambridge, Massachusetts: MIT Press..

[6] Gail A. Carpenter, Stephen Grossberg, and David B. Rosen. (1991). ART2-A: An adaptive resonance algorithm for rapid category learning and recognition. *Neural Networks*, **4**, 493–504.

[7] Richard O. Duda and Peter E. Hart. (1973). *Pattern Classification and Scene Analysis*. New York: John Wiley and Sons.

[8] B.S. Everitt. (1979). Unresolved problems in cluster analysis. *Biometrics*, **35**, 169–181.

[9] Edgardo Ferran, Pascual Ferrara, and Bernard Pflugfelder. (1993). Protein classification using neural networks. *Proceedings of the First International Conference on Intelligent Systems for Molecular Biology*, (pp. 127–135), Menlo Park, CA: AAAI Press.

[10] Stephen Grossberg. (1982). *Studies of Mind and Brain: Neural Principles of Learning, Perception, Development, Cognition and Motor Control.* Boston: Kluwer.

[11] Stephen Grossberg. (1987). Competitive learning: From interactive activation to adaptive resonance. *Cognitive Science*, **11**, 23–63.

[12] Nomi Harris, Lawrence Hunter, and David J. States. (1992). Mega-classification: Discovering motifs in massive datastreams. *Proceedings of the Tenth National Conference on Artificial Intelligence*, (pp. 837–842), Menlo Park, CA: AAAI Press.

[13] John A. Hartigan. (1975). *Clustering Algorithms.* New York: John Wiley and Sons.

[14] J. Hertz, A. Krogh, and R.G. Palmer. (1991). *Introduction to the Theory of Neural Computation.* New York: Addison-Wesley.

[15] Cathie LeBlanc, Charles R. Katholi, Thomas R. Unnasch, and Susan I. Hruska. (1994). DNA sequence analysis using hierarchical ART-based classification networks. *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology*, (pp. 253–260), Menlo Park, CA: AAAI Press.

[16] Daniel S. Levine. (1989). Selective vigilance and ambiguity detection in adaptive resonance networks. In W. Webster, (Ed.), *Simulation and AI, 1989*, (pp. 1–7). San Diego: Simulations Councils, Inc.

[17] Daniel S. Levine. (1991). *Introduction to Neural and Cognitive Modelling.* Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

[18] Albert Nigrin. (1993). *Neural Networks for Pattern Recognition.* Cambridge, Massachusetts: MIT Press.

[19] Robert J. Schalkoff. (1992). *Pattern Recognition: Statistical, Structural and Neural Approaches.* New York: John Wiley and Sons.

[20] Robert R. Sokal. (1974). Classification: Purposes, principles, progress, prospects. *Science*, **185**, 1115–1123.

[21] B. Thylefors. (1978). Ocular Onchocerciasis. *Bulletin of the World Health Organization*, **56**, 63–72.

[22] Peter Zimmerman, Charles R. Katholi, Michael C. Wooten, Naomi Lang-Unnasch, and Thomas Unnasch. (1994). Recent evolutionary history of American Onchocerca volvulus based on analysis of a tandemly repeated DNA sequence family. *Molecular Biology and Evolution*, **11**, 384–392.

Vita

Catherine LeBlanc was born in Goffstown, NH in 1963. After graduating from Goffstown High School in 1981, she attended Dartmouth College, graduating *cum laude* in 1985 with a BA in computer science. She worked for the Geary Corporation in Pittsfield, MA from 1985-1990 and for Necessities/Necesidades in Northampton, MA from 1990-1992.

She returned to graduate school at Florida State University in 1992 where she has worked as a teaching assistant in the Department of Computer Science. She received a Departmental Teaching Award in both 1993 and 1994 as well as a University Teaching Award in 1994. She has also served as a Teaching Associate with the Program for Instructional Excellence from 1993-1995.

Ms. LeBlanc was chosen in 1993 as one of 100 female graduate students to attend the Windows of Opportunity Conference sponsored by the National Science Foundation. She presented a paper entitled "DNA Sequence Analysis using Hierarchical ART-based Classification Networks" at the Second Conference on Intelligent Systems for Molecular Biology at Stanford University in 1994. She has served on the steering committee of Florida State University's Preparing Future Faculty Program and has given a number of presentations to groups such as the Council of Graduate Deans and the Graduate Students Union.