The Design and Application of ICAT: Interactive Cluster Analysis Toolkit

A Thesis

Presented for the

Master of Science Degree

The University of Tennessee, Knoxville

Dax Marek Westerman

May 1998

Acknowledgments

I wish to thank Dr. Michael Berry for his patience, support, and guidance throughout the course of my thesis and my degree. I would also like to thank Dr. Jens Gregor and Dr. Brad Vander Zanden for serving on my committee and for their helpful advice during the course of my research.

I thank Karen S. Minser for her insight and assistance in aiding my comprehension of the original Hoshen-Koppelman implementation. Furthermore, I would like to thank my grandfather, Ted Allen Winter. From him I learned perseverance, which has served me well throughout my education. I would also like to thank the four most important women in my life – Janyce R. Westerman, Shannon Nichole Westerman, Whitney Susanne Westerman, and Kristin Ann Bruce. My mom and sisters, Janyce, Whitney, and Shannon, have supported me throughout my education with their uplifting words and wake-up calls. To my mother in particular, I thank her for teaching me about the breadth of the world before I attempted to plumb some of its depths. To Kristin, my most significant other, I offer thanks for her patience and support. In times of need, her strength become my strength. Finally, I wish to thank all my friends in the Department of Computer Science of the University of Tennessee. Their smiles and laughter made difficult times easier.

Finally, I wish to dedicate this thesis to the memory of one of my dearest friends, Amy Vaughan Weems. She was a mathematician of the highest caliber who always had a glint in her eye, a laugh on her lips, and hope in her heart.

Through her words, I learned to appreciate knowledge as a joy and learning as an adventure. Through her actions, I learned to savor life for the shades of grey, the good as well as the bad times. Foremost, though, it was her gentle, accepting friendship that captivated our worlds and stole our hearts. She was our Amy, she was well-loved, and she will be deeply missed.

Funding for this research has been provided by the Foundation for Australian Resources, an affiliate of the Department of Computer Science at the University of Tennessee.

Abstract

An individual afflicted with diabetes mellitus can develop a condition in which lipids and blood accumulate in the eye. Such deposits are indicative of the early stages of diabetes and potential blindness. In diabetic retinopathy, early detection of these anomalies can aid in diagnosis and prevention. Using cluster analysis on retinal images, computer-generated statistics of cluster size, radius of gyration, ellipsoidal eccentricity, and average texture density can be used to classify such anomalies. In lieu of an automated approach, this thesis promotes a general framework to assist a physician in the detection and classification of retinal abnormalities. The proposed Interactive Cluster Analysis Toolkit (or ICAT) utilizes the Enhanced Hoshen-Kopelman algorithm to provide a highly adaptable method for cluster analysis. Within the context of diabetic retinopathy, different neighborhood rules implemented within ICAT may provide better approaches for classifying retinal features such as neovascularization and exudates. The flexible design of ICAT allows new metrics for characterizing cluster geometry or new neighborhood rules for cluster identification to be easily incorporated.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

The disease *diabetes mellitus* is debilitating and potentially deadly. Over time it has become more manageable with new treatments and examinations available, but there is still no cure. Furthermore, while adults may live with diabetes, provided they follow a regimen of proper diet, medicine, and exercise, this malady is particularly fatal to children and young adults. With this in mind, any contribution to the struggle against this disease provides one more method by which those afflicted may be helped. Fortunately, there are indicators of the disease which act as mile-markers, informing a physician of the state of a patient. One such condition, known as *diabetic retinopathy* provides a very good assessment of a patient's current condition. Considered to be one of the leading causes of blindness in working-age Americans [OL93], this condition occurs as a result of elevated sugar levels in the blood stream. Such blood sugar imbalances are a direct result of diabetes mellitus, which causes improper storage of sugars. Elevated blood sugar levels can cause tremendous damage to the blood vessels in the eye, as well as throughout the body. With respect solely to diabetic retinopathy, the vessels in the eye become damaged, then begin to leak fluid (lipids or blood) into the vitreous gel of the eye. Yellowish deposits resulting from lipids, called *exudates*, may also form on the wall of the retina.

There are two forms of diabetic retinopathy: *nonproliferative* (background) retinopathy and *proliferative* retinopathy [Bre91] [OL93]. The former incarnation usually does not affect normal vision to any appreciable degree, though the presence of lipids and blood leakage may be evident. If noticed early, proper treatment may prevent the onset of the latter more destructive form of retinopathy. It is the stage of proliferative retinopathy where damage to the patient's vision begins. Specifically, the center 10% of the eye, referred to as the *macula*, can become filled with fluid thereby blurring normal vision. This condition is referred to as *macular edema*. Furthermore, when the inflow of blood is reduced as a result of vessel damage in the eye, fine outcroppings of tiny, hair-like vessels may develop as the circulatory system attempts to compensate. This process, known as *neo-vascularization*, can produce numerous blood vessels which are extremely fragile. Frequently, they burst open, releasing more blood and lipids into the vitreous gel.

As the fluid in the eye is normally clear, this can quickly blur normal vision quite badly. Should the damage to the retina become too extensive, the retina itself may detach from the back of the eye. Without proper treatment, this may lead to permanent blindness [OL93].

As mentioned previously, anything which assists physicians in the early diagnosis of diabetic retinopathy is of tremendous help. Other researchers have focused (no pun intended) on this problem, attempting to isolate and identify lipids, blood spots, and neovascularization. By developing software to classify these objects and make predictions accordingly, a physician may utilize such a decision-support environment when dealing with large volumes of images. One project in particular, the **STARE** (STructured Analysis of the REtina) system [GMT+96] has been designed as a fully automatic approach to diagnosis of diabetic retinopathy. **STARE** utilizes a framework for measuring the similarity of images on a layer of primitives for color, composition, texture, and structure [GMT+96]. To date, however, the determination of the best metric for measuring such primitives is an open question in computer vision and visual information retrieval [BW98].

The focus of this work does not involve an automated approach, but rather a more general, quantitative approach. As there is no one metric which can claim to best quantify any particular aspect of a human eye suffering from diabetic retinopathy, a more general framework of assessment was developed. An approach of *cluster analysis* [HK76] was chosen as the method by which anomalies were to be identified. Whereas STARE uses different components of an image to extract information, an approach based on cluster analysis requires a higher level of user interactivity. The user must specify a portion of the image for analysis, then an algorithm for cluster identification may be utilized to extract information about clusters (groups of neighboring pixels). While there are several candidate cluster identification algorithms, the algorithm chosen for this work is the Enhanced Hoshen-Kopelman (EHK) algorithm [HBM97]. This algorithm was chosen for two very important reasons: EHK can process an image in $O(n)$ time, and the algorithm is adaptable new metrics for cluster geometry and neighborhood rules for cluster identification. By utilizing such a flexible approach, an implementation of EHK can provide a more general-purpose approach. If a metric not discussed in this work is later found to more readily quantify neovascularization than current metrics, then the implementation may be augmented with little change to the current implementation. To further facilitate this, this implementation of EHK has been constructed using the language C++. By employing *object-oriented* techniques in the design of this implementation, this further strengthens the application of this work to not only diabetes retinopathy, but any field of image analysis.

1.2 Objectives

The ultimate goal of this research is to provide an interactive tool which may aid a physician with the diagnosis and early detection of diabetic retinopathy. The culmination of this work has resulted in the creation of the Interactive Cluster Analysis Toolkit (ICAT). ICAT is an environment which provides either a command-line or a graphical interface approach to cluster analysis. ICAT itself is composed of two components. The first is a C++ implementation of the Enhanced Hoshen-Kopelman algorithm, and the second is a Java-based *graphic user interface* (GUI). This provides two levels of interactivity. For an anomaly which has well-defined parameters, the C++ aspect of ICAT may be run in a batch-mode over several images, allowing a large quantity of images to be run for comparison. If, however, a physician wished to look closely at an image, the GUI created for ICAT would allow more control over the features provided by the C++ implementation. Ultimately, the main objective is to provide an environment which does not supplant the wisdom of the physician; rather, it could be used to aid in diagnosis. Furthermore, ICAT has inherited the adaptability of the EHK algorithm. Should better metrics in this field become available for quantifying anomalies, it is the goal of this research to provide a framework which could easily be adapted to new methodologies.

1.3 Overview of the Thesis

This thesis will be developed starting with the basic algorithm; in Chapter 2, the evolution of the Enhanced Hoshen-Kopelman algorithm from the original Hoshen-Kopelman algorithm [HK76] is presented. This will include a discussion of new metrics and associated data structures which have been created for use with EHK. Chapter 3 will deal primarily with implementation issues. These issues focus on the C++ development of ICAT as well as the Java-based graphic interface. The discussion of the C++ portion of ICAT will cover the implementation of EHK with different neighborhood rules, pre-processing, and methods for gathering cluster statistics. In considering the Java-based GUI, attention will be paid to features facilitate options implemented within ICAT. Chapter 4 comprises a performance analysis of ICAT under different situations to emphasize the preservation of the $O(n)$ performance of EHK within the implementation. Particular attention is be given to possible uses of the code which may enhance or hinder performance. Finally, a summary and a brief discussion of future work is provided in Chapter 5.

It should be noted that several terms used within this work are interchangeable with those from the field of image analysis. For the sake of clarification, these terms will be described here. References to *cluster identification* within this thesis relate exclusively the segmenting of an image into *connected components*. The process whereby this is accomplished is called *pixel labeling*. Next, the patterns use by EHK for cluster identification are given under specific names within the context of this thesis (e.g., the Four-Node von Neumann neighborhood rule). This differs from image analysis where such patterns are often referred to simply by the number of pixels in the pattern (e.g., four-neighbor template). Finally, two steps utilized during the pre-processing of an image, *scoping* and selection of *intensity levels* are equivalent to the *cropping* and *thresholding* of an image, respectively [Jai89].

Chapter 2

Algorithms and Associated Data Structures

A *cluster* is a collection of pixels formed within some image by utilizing an algorithm for cluster identification. Data about a cluster may be accumulated, providing metrics to be used to further characterize a cluster. While there exist several algorithms which could be implemented for these tasks, the Hoshen-Kopelman algorithm (HK) [HK76] was chosen for its adaptability and speed. The recent development of an *Enhanced* Hoshen-Kopelman algorithm (EHK) [HBM97] demonstrates HK's versatility. In the creation of the EHK algorithm new *neighborhood rules* (see Section 2.2) were added to the original HK algorithm along with new data structures. These neighborhood rules are the patterns by which clusters are formed. The addition of a new data structure now allows the accumulation of cluster statistics; as this data is collected during cluster identification, it does not alter the $O(n)$ performance established by the original HK algorithm. The statistics generated by the EHK algorithm can be used to define metrics for the characterization of cluster geometry.

2.1 Pre-Processing the Images

In order to apply the Enhanced Hoshen-Kopelman algorithm (EHK) to an image, the image must be transferred to a format which it can interpret. There are two steps involved in this pre-processing; the image is first converted into an *intensity* representation, then an array is initialized to represent a portion of the image (see Section 2.3) for use by EHK. An intensity representation can be considered to be a gray-scale image with 256 levels of gray ranging from 0 (black) to 255 (white). It can be obtained by converting a RGB (*red, green,* and *blue*) image to a HSI (*hue, saturation,* and *intensity*) format and extracting the intensity component. Figure 2.1 provides an example of what an intensity image looks like. A one-dimensional array of *unsigned chars* is allocated with length equal to the $width \times height$ of the image. The gray-level (intensity) of each pixel in the image can be stored using 8 bits to encode values from 0 to 255. This array will hold the intensity information of an image until a particular intensity or group of intensities is chosen.

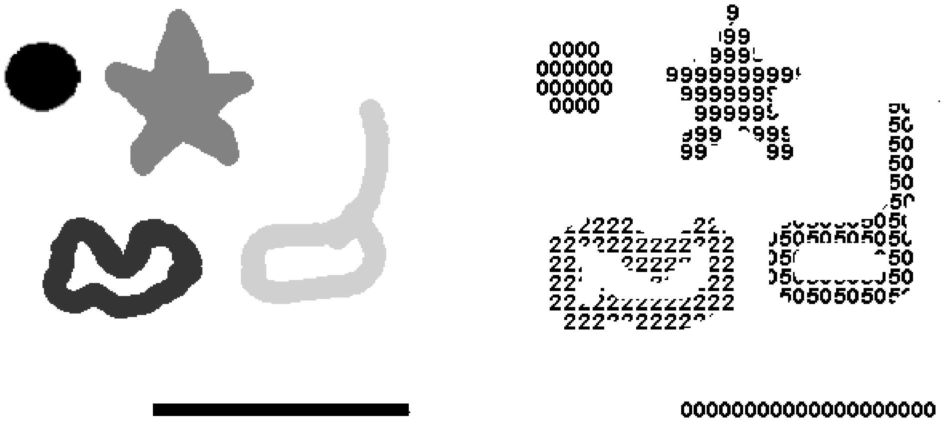

Figure 2.1: Gray-scale (intensity) image of objects of varying size, shape, and levels of gray (intensity). The figure on the right gives the intensity values of each shape on the left.

After the intensity information of an image is stored, the second step of preprocessing may begin. First, an array of integers is allocated with all its entries initialized to zero. Initially, its purpose is to hold data extracted from the intensity information; it plays a different role when used by EHK (see Section 1.4). A value or group of values between 0 and 255 is selected as being of interest. The intensity image currently in memory is traversed, searching for this value or set of values. When a match is found, a -1 is placed in the corresponding index in this second array. When the traversal of the intensity image is finished, this second array will contain -1's and 0's; the -1's represent the intensity levels of interest from the image. This process was performed on the leftmost image in Figure 2.1, where the color black (intensity level 0) was chosen as being of interest. The second array was then initialized to represent any incidences of black within the original image. Figure 2.2 represents this array as an image. The black objects in this image represent where -1's exist in this array.

2.2 The Neighborhood Rules

Before continuing with a discussion about the application of the Enhanced Hoshen-Kopelman algorithm (EHK), the basis for cluster identification should be introduced. In the process of forming a cluster within an image, a template or pattern is used to determine its shape. These templates are often referred to as *neighborhood rules* (see Figure 2.3). A pixel belongs to a cluster if it is incident upon that cluster, based upon the pattern of the current neighborhood rule.

This method of grouping pixels has found several applications. *Symmetric* neighborhood rules, such as the Four-Node von Neumann neighborhood rule [PW85] have been used in landscape ecology [BCM94] and diabetic retinopathy [BW98]. Clusters found in a 2D grid representing a section of land typically represent a habitat type or vegetation cover in the study of landscape heterogeneity [BCM94]. In diabetic retinopathy, such rules are used to help isolate anomalies which may lead to blindness [BW98]. *Asymmetric* rules have also been used in the field of landscape ecology to simulate fire starts under windy conditions [PG93].

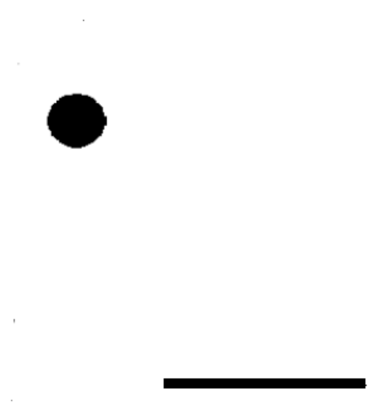

Figure 2.2: Gray-scale representation of Figure 2.1 after pre-processing and representation of working array.

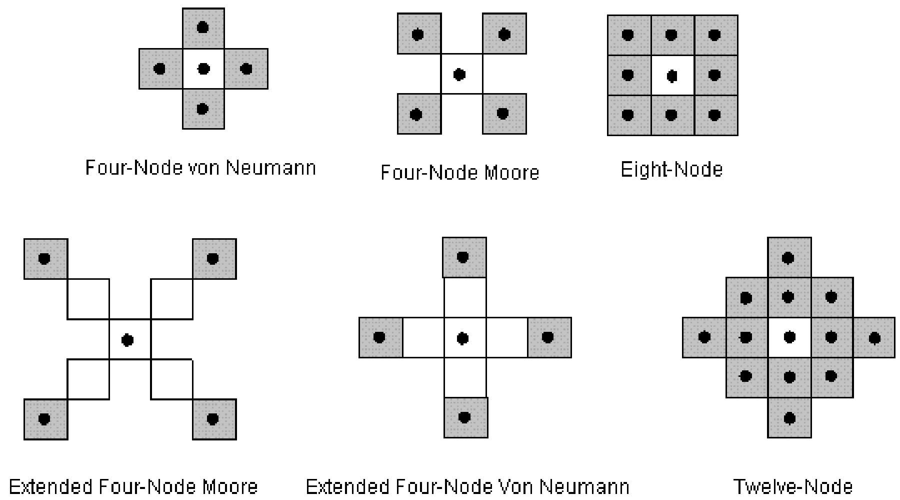

Figure 2.3: Neighborhood rules used with the Enhanced Hoshen-Kopelman Algorithm.

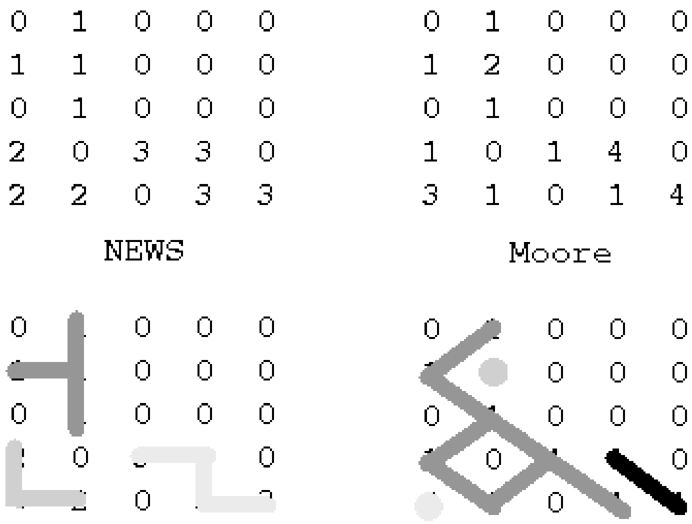

Figure 2.4: Clustering by NEWS and Moore neighborhood rules for a 5x5 grid.

As they do not preserve symmetry, they are more sensitive to starting position than symmetric rules. There are six neighborhood rules of interest within the context of this research: Four-Node von Neumann [PW85], Four-Node Moore [PW85], extended NEWS, extended Four-Node Moore, 8-node and 12-node neighborhood rules (see Figure 2.3). As an aside, the Four-Node von Neumann neighborhood rule is often referred to by an acronym keyed to its shape, NEWS (North, South, East and West) [BCM94].

Figure 2.4 is an example of how the use of different neighborhood rules can affect cluster formation. The 11 occupied cells of the 5×5 grid form either 3 or 4 clusters depending on which rule is used. With respect to images, one rule may provide more desirable results over another depending on the context of the application. As discussed in [BW98], lipids, microaneurysms, and neovascularization are two anomalies of interest in diabetic retinopathy. Exudates are typically compact, while neovascularization, the proliferation of new fragile blood vessels in the eye, form a network of fine web-like lines. In these two examples, compact rules such as NEWS or the 8-node neighborhood rule may provide more desirable results for exudates while a Moore neighborhood rule may be more suitable for detecting neovascularization (see Figure 2.5).

2.3 The Enhanced Hoshen-Kopelman Algorithm

As previously stated, the Enhanced Hoshen-Kopelman algorithm (EHK) is based upon the original Hoshen-Kopelman algorithm (HK) [HK76]. The additions to HK which allowed the development of EHK include the addition of neighborhood rules beyond NEWS and changes in a data structure to facilitate the accumulation of cluster statistics. Additional implementation changes are discussed in Chapter 3. Beyond these changes, however, HK and EHK perform the same basic algorithm, a one-pass approach to cluster identification which runs in $O(n)$ time.

The neighborhood rules used by EHK have already been discussed, but the changes to how EHK accumulates data have not been. There are two data structures needed to implement the enhanced and original versions of the Hoshen-

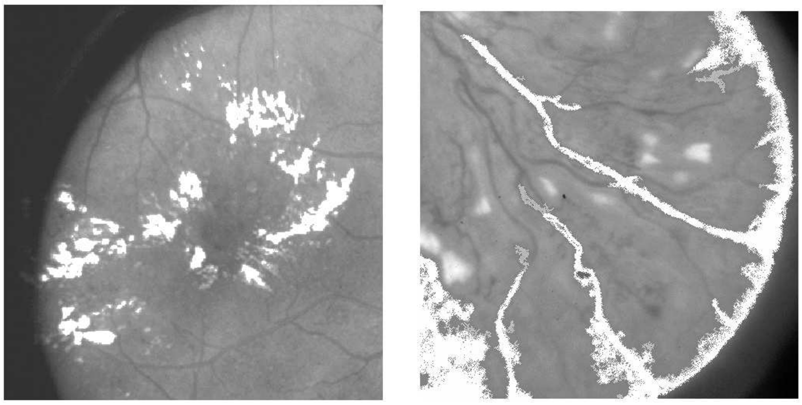

Figure 2.5: (left to right) The first image shows the identification of lipids floating within the vitreous fluid of the eye (NEWS rule), while the second image illustrates the identification of blood vessels and blood spots (Moore rule).

Kopelman algorithm, **matrix** and **csize**. In the original version of the HK algorithm [HK76], these data structures were formulated as a two-dimensional and a one-dimensional integer array, respectively, and were used to keep track of the temporary labeling of clusters. While the **matrix** array has kept its form in the enhanced version of HK, the **csize** array has undergone a moderate transformation to accommodate new features. **Csize** is now an array of *structs* containing not only an integer field for holding the temporary *size* of a cluster or pointer (index) to another cluster but also other fields for values calculated during run-time. The remaining fields within the **csize** struct are used for cluster geometry calculation rather than cluster identification (see Section 2.4). The use of this new **csize** data structure greatly increases the efficiency by which new cluster statistics may be gathered. **Matrix** is a two-dimensional integer array of size $(n+1) \times (m+1)$, where n is the height (rows) and m is the width (columns) of the image in pixels. The dimensions of n and m are increased by 1 to accommodate a box of -9's around the image which EHK uses as a boundary for the image. Pointers are used both to index and traverse the **matrix** array and to minimize the overhead of base-plus-offset array indexing. The **matrix** array requires integer storage (32 bits/pixel), for it serves two purposes. It holds the post-processed map data, and it serves as a working array for the pixels' temporary cluster label.

As mentioned earlier, the original design of the Hoshen-Kopelman algorithm used an integer array (**csize**) to store either the running total of a cluster size or an index leading to another cluster. The principle is the same for the EHK algorithm when considering the size field of the **csize** struct. If the value in an entry is positive, it represents the number of pixels currently included in that cluster. In Figure 2.6, the value 12 in position 1 of the **csize** array indicates that temporary cluster 1 has a current size of 12. If an entry is negative, the absolute value of that entry would point to the cluster (index) in the **csize** array which has merged with this cluster. Negative values may follow a non-circular, recursive path for a finite number of steps. Figure 2.6 contains such an example, starting at temporary cluster 2. Here a 5-step pointer path ($2 \rightarrow 6 \rightarrow 9 \rightarrow 3 \rightarrow 7$) leads to temporary cluster 7 which contains 21 pixels. The existence of the path denotes that the temporary cluster at the start of the path, position 2 in the array, has

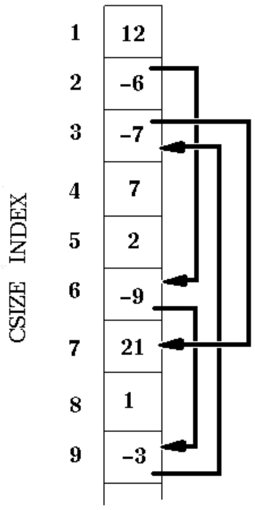

Figure 2.6: Example of the **csize** array (size field) used in the enhanced Hoshen-Kopelman algorithm.

been merged into the temporary cluster at the end of the path, indexed at position 7. Using this method rather than an exhaustive search for cluster labels during run-time is significant to the overall efficiency of the Enhance Hoshen-Kopelman algorithm [CBZ97][HK76].

Similar to the discussion of the implementation of the original Hoshen-Kopelman algorithm in [BW98], EHK will be described using the NEWS neighborhood rule. The other rules discussed in this work are implemented in a similar fashion. As each row of an image is traversed in the EHK algorithm, the value of the pixel is compared to the value of the previous pixel in that row (the west neighbor) and the pixel value in the same column of the previous row (the north neighbor). If all the pixels are in the same cluster, the smallest label is assigned to all three pixels, thus reducing memory requirements of the **csize** array (maximum size is a function of pixel density and clustering patterns of a specific neighborhood rule). Figure 2.7 shows the results of the working array **csize** after merging the temporary cluster labeled 3 from the top row into the temporary cluster labeled 1.

Before the merge, temporary cluster 1 contained 7 pixels; temporary cluster 3 contained 2 pixels. After adding the current pixel (1 pixel) and the pixels from temporary cluster 3 (2 pixels), temporary cluster 1 now has 10 pixels $(7 + 2 + 1)$, and temporary cluster 3 shows a pointer to temporary cluster 1 in the **csize** array. If only the north or west neighbor of the current pixel holds a value, then that label is assigned to the current pixel.

Any changes in the values of the west or north pixels are recorded in the working arrays. Upon completion, the **csize** array holds one of three values: a zero, the number of pixels in a cluster, or a pointer to another index in the array. If the value in the *size* field of the **csize** array is positive, this indicates an actual size, while a negative value represents the aforementioned pointer. De-referencing the pointer, which may follow a non-circular, multi-step path, is accomplished by indexing into the **csize** array using the absolute value of the size field of **csize**. Upon completion of the HK algorithm, the **matrix** array alone may not reflect a correctly labeled map, for the cluster information is distributed between

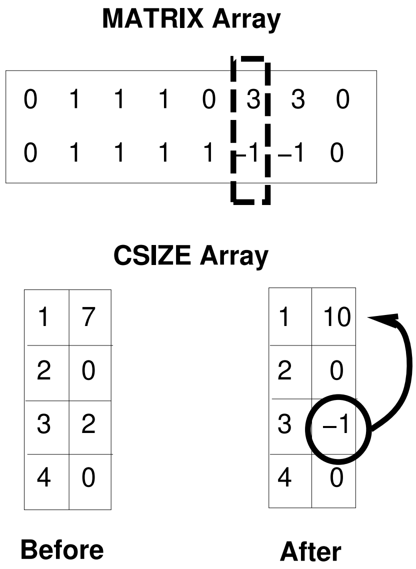

Figure 2.7: The effect of merging clusters on the **csize** and **matrix** arrays

matrix and **csize**. However, a second pass through the working arrays changing the cluster labels in **matrix** by following all merges within the **csize** array will correct this in $O(n)$ time, thus preserving the overall $O(n)$ time for the algorithm. The result is a correctly labeled map of the identified clusters, held in **matrix**.

If the entire **matrix** array cannot be stored in memory, a divide-and-conquer approach can be used [CBZ97]. Cluster identification is accomplished by performing the analysis on subsections of the image and merging the results. Each partition is of size $k \times m$ where k is the number of rows in each partition. This approach may be implemented for HK and consequently EHK as a result of the nature of this process [CBZ97], since at any given time the algorithm only requires access to the array **csize**, the current pixel, and enough rows from the image to accommodate the current neighborhood rule. In the case of NEWS, for example, one would need at least two rows to use EHK: the row containing the current pixel and the row immediately preceding. This preceding row would contain information about possible neighbors to the north of the pixel.

2.4 Cluster Statistics and Shape

Upon completion of the Enhanced Hoshen-Kopelman (EHK) algorithm, information about cluster *statistics* and *shape* may be obtained from the **csize** array. The statistics or *metrics* currently associated with EHK are *size*, *radius of gyration*, the *eccentricity* of a best fit ellipse and the *average texture density* of a cluster. It is with respect to the last three metrics that the new incarnation of the **csize** array gains value; they will be the main focus of this section.

As discussed in [BW98], the squared *radius of gyration*, R_s, is one shape metric [CBZ97][SA93] commonly used in cluster classification. Define

$$R_s^2 = \frac{1}{2s^2} \sum_i \sum_j |\mathbf{r_i} - \mathbf{r_j}|^2, \tag{2.1}$$

where s is the size of the cluster, r_i and r_j denote the positions of sites of pixels within the image at positions i and j, respectively.

The squared distance $|\mathbf{r_i} - \mathbf{r_j}|^2$ in Equation (2.1) may be rewritten [HBM97] as

$$|\mathbf{r_i} - \mathbf{r_j}|^2 = \sum_{\mu=1}^{2} (x_{i,\mu} - x_{j,\mu})^2, \tag{2.2}$$

where $x_{i,\mu}$ and $x_{j,\mu}$ denote the μ^{th} coordinates of pixels i and j in 2 dimensions.

Since each dimension μ from Equation (2.2) may be treated independently, the double summation in Equation (2.1) can be reduced [HBM97] for each μ to

$$\sum_i \sum_j (x_{i,\mu} - x_{j,\mu})^2 = 2s \sum_i x_{i,\mu}^2 - 2\left(\sum_i x_{i,\mu}\right)^2 = 2\left[sX_\mu^{(2)} - (X_\mu^{(1)})^2\right] \tag{2.3}$$

where

$$X_\mu^{(1)} = \sum_i x_{i,\mu} \text{ and } X_\mu^{(2)} = \sum_i x_{i,\mu}^2$$

are referred to as the first and second spatial moments [Pra91], respectively, for the given cluster and the μ^{th} coordinate. It follows from Equations (2.1-2.3) that

$$R_s^2 = \frac{1}{s^2} \sum_{\mu=1}^{2} \left[sX_\mu^{(2)} - \left(X_\mu^{(1)}\right)^2\right]. \tag{2.4}$$

It is this form that permits the algorithm to retain its $O(n)$ operating time. Since $X_\mu^{(1)}$ and $X_\mu^{(2)}$ can be gathered during the use of the algorithm, R_s^2 can be calculated without additional traversals of the image. The eccentricity of a best-fit ellipse over a given cluster may be derived from concepts used in mechanics. In this case, the *moment of inertia* provides the basis for this metric. To begin, one must assume that each pixel reflects a point unity mass. This assumption implies that the *mass* of a cluster is equivalent to its size s or number of pixels within the cluster. One may also assume that any pixel which is a nearest neighbor is 1 unit distance away, giving a mass density (mass per unit area) of 1. It can be shown that the moment of inertia, I, perpendicular to the X-Y plane which passes through the cluster's center of mass is given by

$$I = sR_s^2.$$

Now, suppose an ellipse with major and minor axes labeled a and b respectively is superimposed upon a cluster with the center of the ellipse directly over the center of mass of the cluster. If the ellipse has the same area (mass) and moment of inertia as the cluster, then the ellipse's area can be represented by $s = \pi ab$. The moment of inertia through the center of the ellipse, perpendicular to the X-Y plane, is given by

$$I = (s/4)(a^2 + b^2),$$

where $a = (\xi + \phi)/2$, $b = (\xi - \phi)/2$, $\xi = \sqrt{4I/s + 2s/\pi}$, and $\phi = \sqrt{4I/s - 2s/\pi}$. The effective eccentricity of an ellipse is then defined by

$$e = (\sqrt{a^2 - b^2})/a. \tag{2.5}$$

The eccentricity e may be used to aid in the characterization of the shape of the ellipse (cluster). If e is near 0 in value, this indicates a more circular cluster while a value of e nearer to 1 reflects more elongated clusters. Examples of these extremes are illustrated in Figure 2.8 involving the circular and an elongated cluster from Figure 2.2, both having size $s = 2025$. The circular cluster has values $R_s^2 = 322.277$ and $e = 0$ while the longer cluster has values $R_s^2 = 2834.007$ and $e = 0.998$.

The final metric utilized within this implementation of EHK is the *average texture density*. As described in [KD96], the average texture density, $T(c)$, provides the means to determine how uniform a cluster is with respect to intensity levels. Define

$$T(c) = \frac{1}{A_c} \sum_i \sum_{j \in N(i)} \frac{|f_i - f_j|}{8}, \tag{2.6}$$

where f_i is the pixel value of the ith pixel, $N(i)$ is the set of 8 pixels which form the 8-node neighborhood rule around the ith pixel, and A_c is the area (size) of cluster c. The effect of implementing this equation is to average the absolute value of the difference between a pixel and its neighbors, then average these values over the size of the cluster. The average cluster density can be used to determine how uniform a cluster is with respect to the intensities in the cluster.

The enhanced version [HBM97] of the Hoshen-Kopelman algorithm [BCM94] used for cluster identification can make use of the augmented **csize** array in a one-

Figure 2.8: Examples of extreme cases for the eccentricity of a best-fit ellipse

pass approach to gather data needed for both the squared radius of gyration R_s^2 (Equation 2.1) and the eccentricity of the best-fit ellipse e (Equation 2.5). Since the required data can be gathered in $O(n)$ time, these metrics become valuable tools in the characterization and comparison of clusters.

Chapter 3

Software Environment

As discussed in Chapter 2, the original implementation of the Hoshen-Kopelman algorithm (HK) involves a one-pass approach using one neighborhood rule: NEWS [HK76]. When considering only this case, implementation in an *imperative* language such as C is a logical step. However, the Enhanced Hoshen-Kopelman Algorithm (EHK) [HBM97] has been implemented with additional functionality. New neighborhood rules, data structures, and metrics suggest a more modular approach. Even though such code could be written completely in an imperative language, an *object-oriented* design greatly improves its usefulness. By implementing EHK in C++, the addition of new rules, new cluster metrics, and general code maintenance becomes dramatically simplified. Furthermore, pre-processing, setup, and implementation of EHK can be grouped into *classes* which allows greater flexibility in design.

Another aspect of this software environment is the *graphical user interface* (GUI) which has been developed in Java. This interface has been dubbed the **Interactive Cluster Analysis Toolkit** or **ICAT**. The Java-based GUI interfaces with the C++ code, allowing a user to exploit features built into this implementation of EHK (see Figure 3.1).

This C++ implementation of EHK utilizes three classes:

- Image,
- Setup, and
- Cluster

Each of these classes deals with a particular aspect of the implementation. The **Image** class handles image interpretation (pre-processing) and color table creation for use in distinguishing clusters according to their size. It also contains methods for providing a visual record of detected clusters.

The **Setup** class provides the transition from intensity image to the working array **matrix** (see Chapter 2) and an optional pre-processing step. Finally, the class **Cluster** encapsulates the EHK algorithm, the neighborhood rules used, and the implementation of new metrics. Table 3.1 illustrates the general class hierarchy and the main methods within each class. There are many other methods encapsulated within each class; those listed in Table 3.1 are either unique in their use or interfaces for other protected methods.

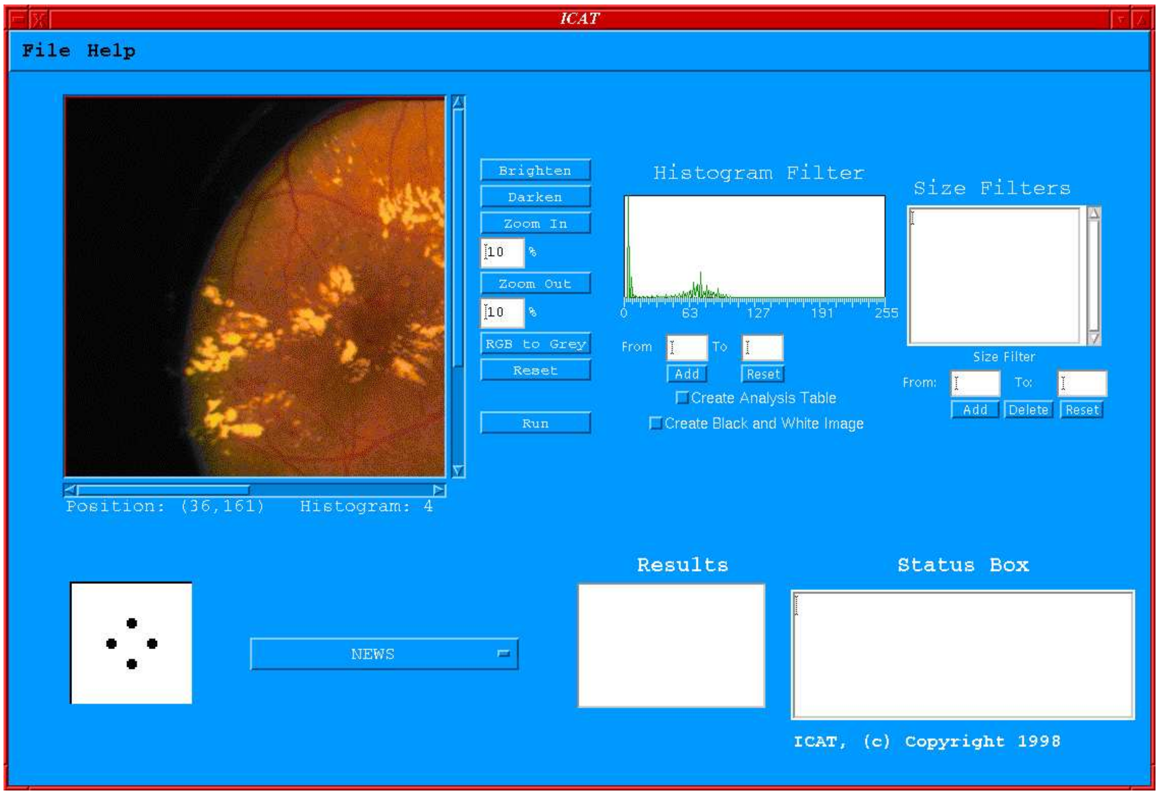

Figure 3.1: The Java-based graphic user interface (GUI) for ICAT

Table 3.1: The three classes implemented for EHK and the main methods they encapsulate.

Image	Setup	Cluster
create_shades	LoadMask	Label_first_row
ReadImage	set_borders	EHK_interface
Write_BW_Image	movmap	reLabel
WriteOverlayImage		Radius
CovertToIntensity		AvgTextureofPixel
		Create_overlay

3.1 The Image Class

An image must first undergo the pre-processing steps described in Chapter 2 before use by EHK. It falls to the **Image** class to perform these actions. This class is also responsible for the creation of *color tables* and *overlay images*. Color tables generated within an object of this class are used to distinguish clusters of different size categories within an image. Overlay images come in two flavors: *cluster* overlays, generated by the method **WriteOverlayImage** and *black-and-white* overlays. A final option is encapsulated within this class: *scoping*. This is the process of choosing a smaller rectangular portion of the image for use by EHK. In this manner, all operations dealing specifically with images have been encapsulated within the **Image** class.

3.1.1 Image Interpretation

The Enhanced Hoshen-Kopelman Algorithm (EHK) utilizes an integer array representing all or part of an image. Most images generally are not encoded in such a format. This specification requires an implementation which can read different image formats and translate them for use with EHK. Current formats used by the method **ReadImage** are *XPM* (X11 pixmap) and *PGM* (portable gray-map), which allows for color and gray-scale images. Other images can certainly be utilized by this implementation of EHK if a method for translation is implemented (see Chapter 4).

3.1.2 Color Tables

Another aspect of the Image class, the creation of a color table, is implemented via the method **create_shades**. The user may specify one or more ranges for cluster size; each range must be between 0 and the image size (in pixels). These ranges determine which clusters are displayed in the overlay images. Furthermore, cluster statistics will only be generated for clusters within these ranges. It can become visually confusing when several ranges are specified; therefore, a color table can be generated to provide each range with a specific color (see Figure 3.2). This color table ramps from yellow to orange to red, assigning colors based upon the order in which the original ranges were specified. This helps to discriminate between clusters of different sizes within the same overlay image.

3.1.3 Black and White Overlay Images

Often, visual inspection of clusters can be difficult due to noise or cluster position within an image. One solution is to create a black-and-white overlay of the results using the method **Write_BW_Image**. Once all the clusters have been identified, they are represented as white on a black background. Using a black-and-white image provides a method for quickly observing basic shape information as well (see Figure 3.3).

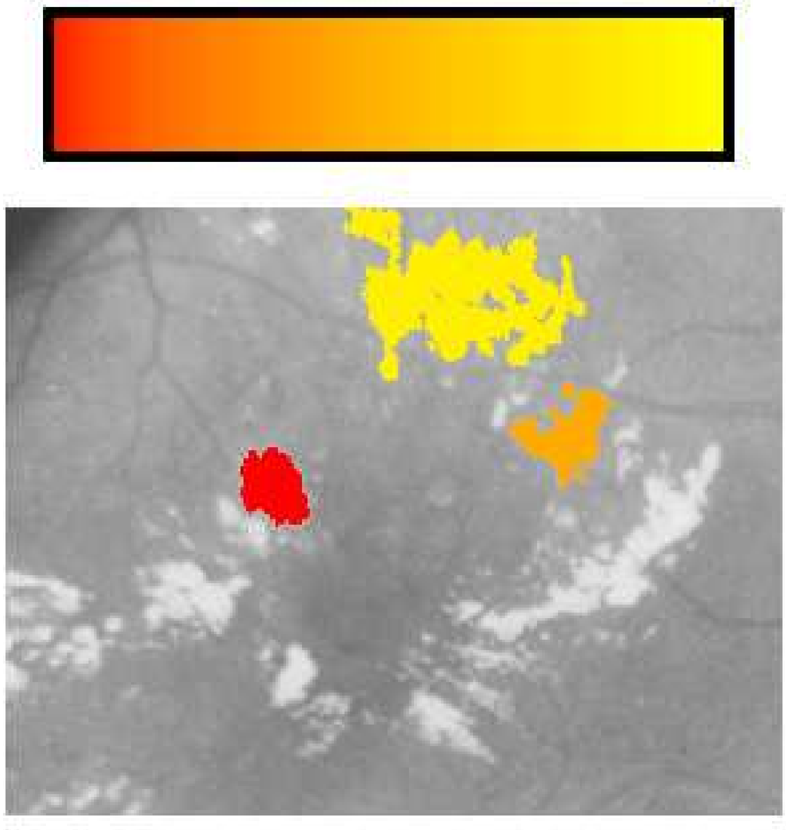

Figure 3.2: An example of the color table used with ICAT. By using a color table, clusters can be quickly identified by the order in which their size ranges were specified. In this case, colors from the table are chosen in the order in which each size range was selected.

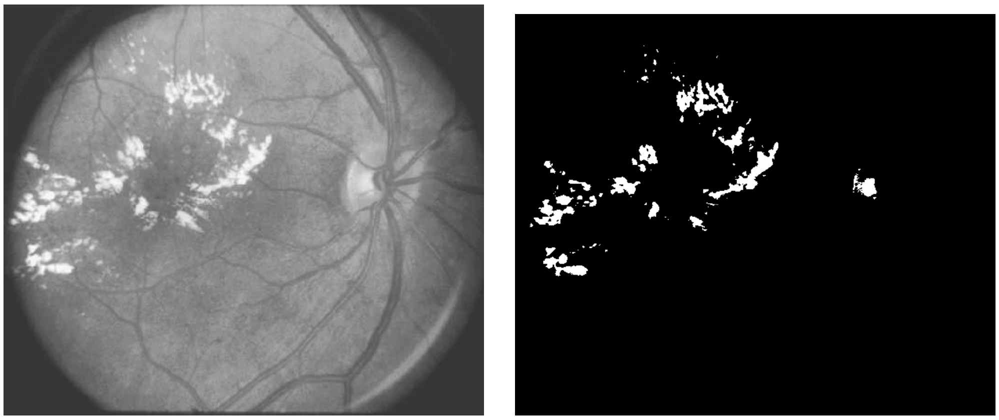

Figure 3.3: Black-and-white overlay of clusters found between intensities 130 to 200, inclusive.

3.1.4 Scoping the Image

Another utility of this implementation is the scoping feature, which allows the user to specify a smaller portion of an image for use by EHK. While basically uncomplicated, this feature does allow significant gains in overall speed, especially if the image to undergo analysis is very large. A rectangular sub-section of the image is copied from the original image (see Figure 3.4). This portion of the image is then translated to an intensity representation. Changes are made accordingly to account for the smaller image with respect to size variables.

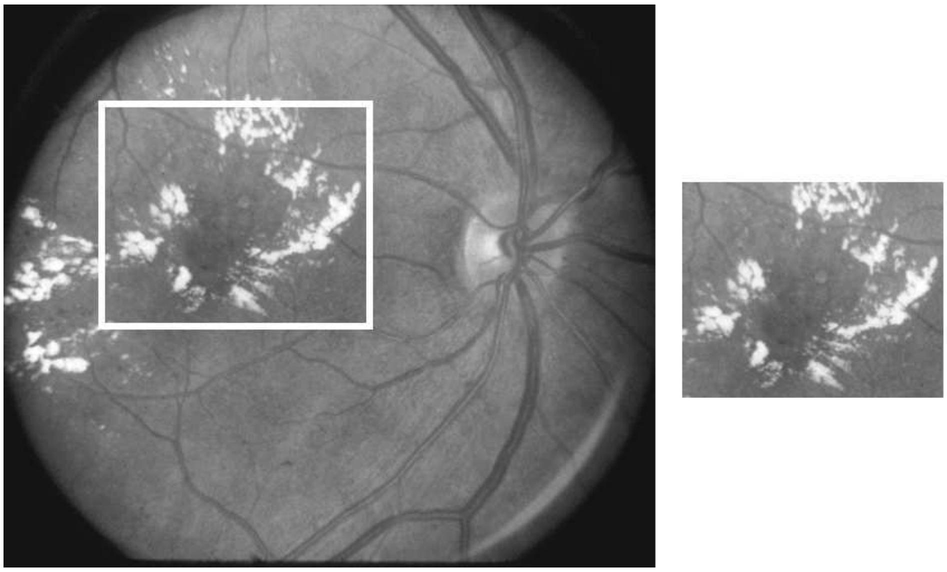

Figure 3.4: The operation of *scoping* or choosing a sub-portion of an image.

3.2 The Setup Class

Of the three classes which compose this implementation of EHK, the **Setup** class is the most succinct. Despite its size, it is a very necessary part of this implementation. **Setup** encapsulates the translation between the intensity representation of an image and the array **matrix** (see Chapter 2). It falls to this class to allocate **matrix** to the correct size and initialize it, using the specified intensity ranges. This class also implements a method akin to scoping, called *masking*.

3.2.1 Selection of Intensity Ranges

As described in Chapter 2, intensity ranges are chosen for analysis during the preprocessing step. Let h_k represent an intensity range of one or more intensities, $0 \leq k \leq 255$ such that $h_k \in \{0, 1, 2, ...255\}$ with the following conditions:

- $inf(h_k) \geq 0$ and $sup(h_k) \leq 255$,
- $h_k \bigcap h_j = \{\emptyset\}, 1 \leq k, j \leq 255$.

An *amalgam* of ranges may be specified as well. An amalgam can be informally described as using several intensity ranges simultaneously for an analysis, rather than one set per use of EHK. Let an amalgam of ranges, H, be defined such that $H = \bigcup_k h_k, 0 \leq k \leq 255$. EHK will be applied for each specified intensity range, and a combination of distinct ranges and an amalgam of ranges may be used at the same time.

Another method which decreases CPU-time and focuses the area of interest is the masking option. This option allows the user to employ an *integer* array of size equal to the intensity image for specifying which areas of the image are of interest. The array holds two values: 0 or 1. A value of 1 for an array element indicates a pixel of interest. The logical AND is then taken between the intensity image and masking arrays with the result placed into the array **matrix**. Although this method may be used with the scoping option, the masking option creates a rectangular bounding-box around the region of interest so that no appreciable gain in time is observed.

3.2.2 Translating the Intensity Image to the Matrix array

Based upon the discussion in Chapter 2, the intensity array is translated into the **matrix** array utilizing the intensity ranges chosen utilizing the method **movmap**. If none are specified, then the default intensity range is set to be 0 to 255, inclusive. (This is not recommended unless the masking option is used as well, for the result will be a cluster of size equal to the original image.) The code accomplishes this task by checking each pixel in the intensity image to see if its value falls into a specified range. If so, then **matrix** is assigned the value -1 at the corresponding position. If not, that **matrix** element is defined to be 0.

3.3 The Cluster Class

The **Cluster** class encapsulates

- the use of EHK,
- the ranking of clusters identified by EHK, and
- the collection of data for the generation of statistics.

The idea of using one class to encapsulate all the possible rules and metrics associated with this implementation of EHK fosters versatility. If future implementations of the code require changes in pre-processing steps, the performance of the **Cluster** class will not be significantly affected.

3.3.1 Implementation of New Neighborhood Rules

The original design of the Hoshen-Kopelman algorithm (HK) was based on the four-node Von Neumann or NEWS neighborhood rule (see Figure 2.3). This neighborhood rule's pattern dictates how HK is implemented [HK76]. In creating software for new neighborhood rules, it becomes necessary to alter the original algorithm slightly. This accommodates the necessarily individual approaches ensuing from each rule. For example, the original implementation of HK required three levels of functionality:

1. Labeling the first row of **matrix**, using the method **Label_first_row**.
2. Performing cluster identification, using **EHK_interface**.
3. Relabeling **matrix** to provide a correct mapping of clusters, using **reLabel**.

While EHK preserves these three actions, changes within the actions themselves are necessitated by new neighborhood rules. In the original algorithm the labeling of the first line involves moving to the right, one pixel at a time. Then, the immediate west and north neighbors of this pixel are checked for cluster membership. After this procedure, the algorithm can progress normally [HK76]. Implementing the Extended Von Neumann neighborhood rule (see Figure 2.3) within

EHK requires looking not to the immediate neighbors of the current pixel but two pixels before the current one as the initial labeling step. A check to the north is not necessary, as no neighbors exist to the north of the first line (see Figure 3.5). Once this task is accomplished, the algorithm must be adapted to allow the code to look two cells to the west and two cells to the north in lieu of one in each case. All the symmetrical rules are treated in a similar fashion. Since no changes are made to the **csize** and **matrix** arrays, the method reLabel does not require modification.

3.3.2 Collection of Cluster Data and Generation of Cluster Statistics

It is within the context of generating of cluster statistics that the new **csize** data structure earns its keep. The idea of using an $O(n)$ time algorithm for cluster identification is to provide a quick analysis; in keeping with this philosophy, it follows that data collected *on the fly* would be more desirable than performing a second pass of the EHK algorithm. To this end, **csize**, as defined for the original Hoshen-Kopelman, algorithm [HK76] has been replaced by a *struct* named **c_info** with the following fields:

- size,
- rank (discussed in Chapter 4),
- the first spatial moment, $X_{\mu}^{(1)}$,

Matrix

0	0	−1	−1	−1	0	−1	−1	−1	−1
0	−1	−1	−1	−1	−1	−1	−1	−1	0
0	−1	−1	−1	−1	0	0	0	0	−1

NEWS

0	0	1	1	1	0	1	1	1	1
0	1	1	1	1	1	1	1	1	0
0	0	1	1	−1	0	0	0	0	−1

Extended NEWS

0	0	1	2	1	0	1	3	1	3
0	4	5	4	5	4	5	4	5	0
0	0	1	2	−1	0	0	0	0	−1

Figure 3.5: At the top of the figure are the first few lines of an initialized **matrix** array. Following the matrix array are examples of the differences between the implementation of the NEWS neighborhood rule and the Extended NEWS neighborhood rule in ICAT.

- the second spatial moment, $X_\mu^{(2)}$,
- the radius of gyration, R_s,
- the squared radius of gyration R_s^2, and
- the average texture density of a cluster, $T(c)$.

As each pixel is visited in the EHK algorithm, values for $X_\mu^{(1)}$, $X_\mu^{(2)}$, and $T(c)$ are continuously updated for each cluster in the **csize** array. After the second pass is made through the **matrix** array to correctly label the clusters, the method **Radius** is called to calculate values for R_s and R_s^2.

At first glance, the design of **csize** seems to require a tremendous amount of memory. However, speed is the more important issue for this implementation. By caching this information, subsequent calculations of the eccentricity of a best-fit ellipse and the average texture density take less time (see Chapter 2).

3.4 The Interactive Cluster Analysis Toolkit

The C++ implementation of the Enhanced Hoshen-Kopelman algorithm (EHK) may be somewhat difficult for a non-technical individual to use. Choosing intensity ranges and scoped areas of interest is non-intuitive at best. To address this problem, a graphical user interface (GUI) was designed, using the Java language. The Java interface along with the C++ implementation of EHK, referred to as the Interactive Cluster Analysis Toolkit (ICAT), provides a smoother interface between the user and the EHK algorithm.

3.4.1 Communication between Java and C++ in ICAT

The idea of developing a graphical user interface (GUI) for this implementation of EHK becomes more and more significant as work with EHK progresses. An approach was needed, however, which would allow the C++ code to be usable with or without the GUI. As a result, a design using network sockets was implemented. Through this method, the interface creates the appropriate command-line parameters from user interaction within the GUI, and the GUI forwards the parameters to the EHK implementation.

3.4.2 Implementation of EHK features in ICAT

As discussed previously, there are several choices in how to use this implementation of the EHK algorithm which relate to pre-processing, choice of neighborhood rules used, and other salient features of the C++ implementation. Since most of these options are non-intuitive in their use, various parts of ICAT have been adapted to provide graphical representations. Figure 3.6 shows the scrolling canvas used to view the image. Within this canvas, a user may click and drag the mouse cursor over the image to form the boundaries of a box, as shown in the right side of the image. Position of the cursor and the intensity level at this point are

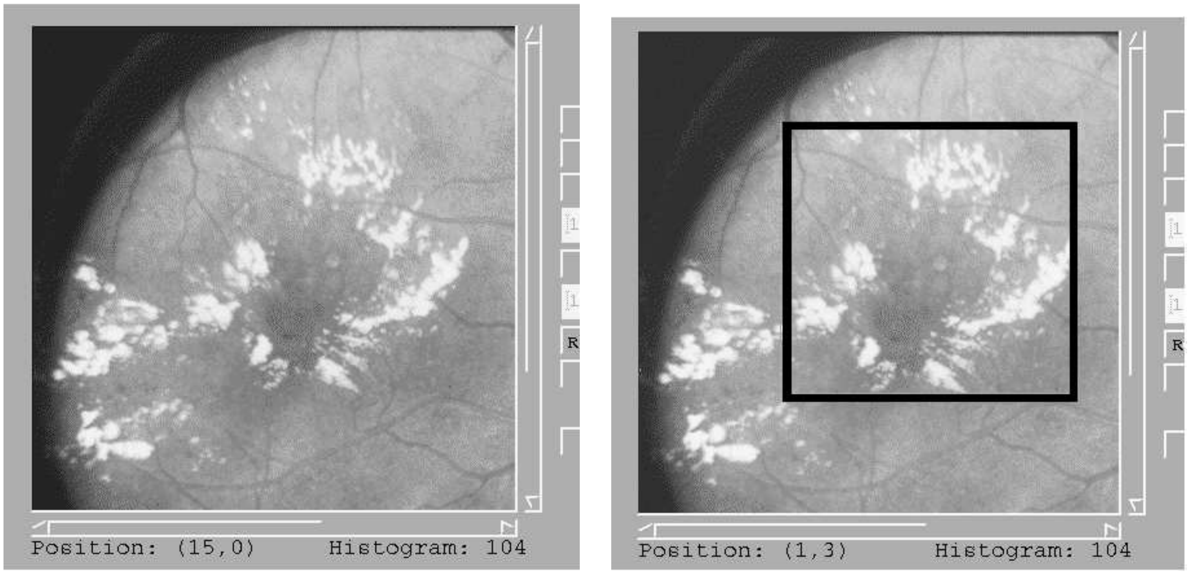

Figure 3.6: The scrolling image canvas (left) and an example of scoped portion of the image (right).

displayed under the image. Since the user can interact with the image in this fashion, the masking feature is not implemented in ICAT. Figure 3.7 depicts the panel used to give the user choices about what intensity ranges should be used for analysis. The graph represents a *histogram* of intensities from the image. This graph is normalized by dividing by the number of pixels associated with the most dominant intensity level in the image. The user may type in an upper-bound and lower-bound for an intensity range or may simply click and drag across the graph, thus shading in a region.

By using the mechanism depicted in Figure 3.8, a user may also specify what

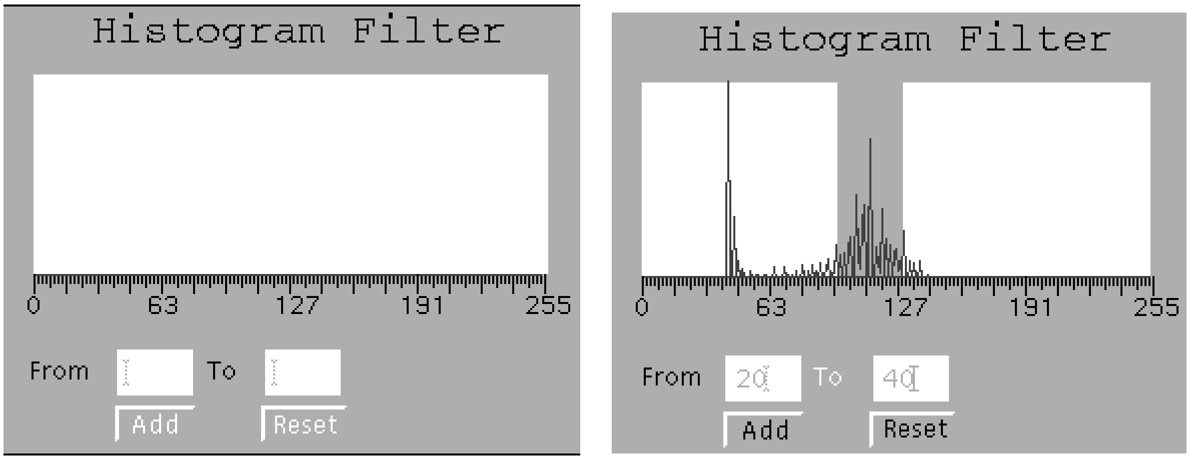

Figure 3.7: Intensity selection panel (left) and an example of selecting intensity ranges (right).

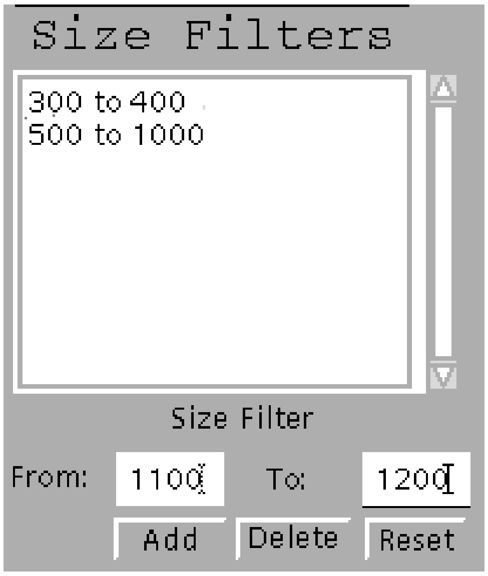

Figure 3.8: The size range selection panel.

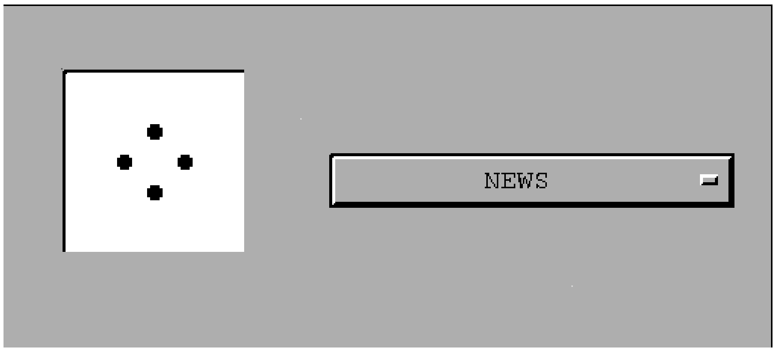

Figure 3.9: The Neighborhood Rule selection panel.

size ranges should be employed. Its use is similar to the intensity selection panel in that an upper- and lower-bound may be entered in the text boxes. The neighborhood rule may be chosen from a list; the current rule is reflected by an image depicting the template as shown in Figure 3.9. Other features, such as a black-and-white image or textual output, have been implemented as radio-buttons.

3.4.3 Extra Features

Since the primary goal of ICAT is to aid a user in utilizing the C++ implementation of EHK, other features have been added. For example, users may zoom in or out from the picture, enabling them to gain a better perspective of objects in the image. If the image is difficult to see as a result of contrast or brightness problems, minor adjustments may be made to produce a clearer image. Such changes do not affect EHK's interpretation of the image. In addition, if the image is brightened or darkened, the histogram will reflect this shift in intensities. Furthermore, if the image is enlarged or shrunk, a scoped portion of the image will be resized relative to the current aspect ratio. While these options add nothing to the results, they do provide a more comfortable environment for use of EHK.

Chapter 4

Computational Performance

This chapter presents the performance results of a series of analyses. They were obtained on a SPARCstation 20 with 256 megabytes RAM, utilizing SunOS 5.5.1 (Solaris 2.5). These results demonstrate the performance of the C++ implementation of the Enhanced Hoshen-Kopelman algorithm (EHK). This performance is based upon the relative impact of dominant methods within each class. Within this chapter, performance of nine different methods will be examined for each example. Each of the nine methods listed below is followed by the name of its *class* and, if applicable, the *method* by which it was invoked (see Table 3.1):

- ReadXPM (class **Image**, called by **ReadImage**),
- XPMtoIntensityImage (class **Image**, called by **ConvertToIntensity**),
- movmap (class **Setup**),

- HKid4_NEWS (class **Cluster**, called by **EHK_interface**),
- reLabel (class **Cluster**),
- AvgTextureofPixel (class **Cluster**),
- Radius (class **Cluster**),
- CreateOverlay (class **Cluster**), and
- WriteOutputPPM (class **Image**, called by **WriteOverlayImage**).

Each of these methods demonstrate necessary functionalities of this implementation. **ReadXPM** and **XPMtoIntensity** provide the pre-processing step described in Section 2.1. **ReadXPM** feeds a translation of an *XPM* (X11 pixmap) image into **XPMtoIntensity**, which in turn converts the image into an intensity representation. The method **movmap** then utilizes the intensity ranges defined by the user to create the array **matrix**, also described in Section 2.1. **HKid4_NEWS** then applies the Enhanced Hoshen-Kopelman (EHK) algorithm (Section 2.3) with the Four-Node von Neumann or NEWS neighborhood rule (Section 2.2). Upon completion of the EHK algorithm, the method **reLabel** is used to correctly label the **matrix** array (Section 2.3) using information from the **csize** array. For this implementation of EHK, the method **AvgTextureofPixel** is used on each pixel within each cluster identified by the algorithm. This method gathers data for use in calculating the *average texture density* (see Section 2.4).

The method **Radius** is invoked after EHK is complete; this method calculates the *radius of gyration* for each cluster listed in the **csize** array (see Section 2.4). **CreateOverlay** and **WriteOutputPPM** are used to create a resulting image. **CreateOverlay** creates copies of the clusters on the intensity map in memory. **WriteOutputPPM** then takes this modified intensity image and writes the results to a *PPM* (portable pixmap) file.

4.1 Discussion of Results

Within the context of the results, there are three main effects on the efficiency of the Interactive Cluster Analysis Toolkit (ICAT). The first is *image size*. This factor directly affects the methods **ReadXPM**, **XPMtoIntensity**, **movmap**, **CreateOverlay**, and **WriteOutputPPM**. This is not surprising, for all of these methods are directly involved with reading, manipulating, or re-creating the original image, though **CreateOverlay** and **ReadXPM** are somewhat of an exception. **CreateOverlay** is dependent on the total number of pixels selected for analysis. Each pixel in the intensity image must be checked to see if it belongs to a cluster or not before it is labeled. **ReadXPM** incurs extra cost in cpu-time since it must first create a color map specific to each *XPM* file.

The second trend evidenced within these tables is a dependency on the amount of information designated for analysis. When ranges of intensity are chosen, a sub-section of the image is flagged for use by the EHK algorithm (see Section 2.1). As a rule of thumb, the larger the range of intensities set aside for analysis, the more information will be processed by EHK. As demonstrated in the examples provided, this affects the methods **HKid4_NEWS**, **CreateOverlay**, and **AvgTextureofPixel**. (It should be noted that implementations of neighborhood rules other than NEWS are certainly affected in a similar manner.) The ratio between the number of pixels selected for use with EHK and the size of the image becomes very important, as it directly affects the time requirements of certain methods. For example, **HKid4_NEWS** maintains an average percentage use of about 4.5% of the total elapsed CPU time in the second and third examples. In these examples, the ratio of the number of pixels used for analysis versus the number of pixels in the image in each case were relatively close (4% versus 9% respectively). This is consistent with the assertion that EHK can be implemented in $O(n)$ time [HBM97]. Even when the ratio of pixels used to total pixels in the image is slightly changed, the percentage CPU-time use is nearly the same. Other situations are also presented in the other examples which the percentage CPU-time consumed by **HKid4_NEWS** changes according to the number of pixels analyzed. **CreateOverlay** is affected by the number of comparisons it must make when creating an overlay image (see Chapter 3). Each pixel must be queried to determine if it is a part of a cluster or not. Finally, since **AvgTextureofPixel** checks the eight neighbors of each pixel in a cluster (excluding boundaries), it too is dependent upon the number of pixels selected for analysis by the user.

The final trend demonstrated within the following examples depends on the number of clusters located by EHK. The only method listed which relates to this case is **Radius**. Within this method each cluster in the **csize** array is examined for computing the radius of gyration, R_s, (see Chapter 2) from cached values. Even though this method's effect on the total CPU-time is nominal in the following examples, it has been included to emphasize this trend.

4.2 Comparison of Images

The image used for performance comparisons is a 768×512 image in *XPM* (X11 pixmap) format. This image depicts an eye heavily affected by exudates (lipids) resulting from diabetes (see Figure 4.1). Lipids show up as bright, globular anomalies which tend to exist within a well-defined region of intensities; generally, they are easily identifiable [BW98]. The series of tests which have been run upon this image (and selected sub-portions of this image) demonstrate the impact of the following situations:

- identification of a cluster within a scoped portion of an image,
- identification of all clusters within an intensity range,
- identification of all clusters within an intensity range on a larger image, and
- utilization of the entire range of intensities for an analysis.

Figure 4.1: Image depicting exudates (lipids) floating within the vitreous fluid of a diabetic eye. These exudates are the high-intensity objects to the left of center in the image.

The results of these tests have been tabulated so that each method and its corresponding percentage of time can be compared. Such data demonstrates the effects of file I/O, the EHK algorithm, and statistical calculation upon the efficiency of ICAT.

It is important to note that within the context of the following examples, the method **HKid4_NEWS** specifies the implementation of EHK using the NEWS neighborhood rule (see Section 2.2). Results and observations from utilizing other neighborhood rules with the basic method (EHK) are quite similar to those obtained for **HKid4_NEWS** and are not presented.

4.2.1 Scoping an Image

This example illustrates the scoping feature (see Section 3.1.4) for processing a smaller portion of an image. This demonstrates how a user may optimize CPU-time usage by selecting a smaller area of interest within an image for analysis. The scoped portion of Figure 4.1 shown in Figure 4.2 is of dimension 148×129. Intensities ranging from 120 to 255, inclusive, from Figure 4.2 were analyzed in order to capture information about lipids. Table 4.1 shows the performance results.

Within this example, the scoped portion of the image is significantly smaller than the original image (about 5% of the original in size). As can be seen in the table, those methods which depend upon the intensity image for information (**HKid4_NEWS**, **movmap**, and **reLabel**) do not require as much time as those which operate on the original image (i.e., **ReadXPM** and **XPMtoIntensityImage**). As a result, a specific region may be processed more efficiently. An example using the entire image follows in the next section.

4.2.2 All Objects

This example illustrates the performance of ICAT when used on an entire image within a certain intensity range. The intensity range 120 to 255, inclusive, was again chosen in order to isolate the exudates (lipids) within the image (see Fig-

Figure 4.2: Scoped portion of Figure 4.1.

Table 4.1: Performance results from isolating one lipid (cluster) via scoping.

Method	**% time**
ReadXPM	22.34
XPMtoIntensityImage	26.71
movmap	2.39
HKid4_NEWS	1.98
reLabel	2.20
AvgTextureofPixel	1.33
Radius	0.01
CreateOverlay	7.22
WriteOutputPPM	4.26
Total Time (in seconds)	
CPU:	0.19
USER:	0.80
Wall-clock:	1.07

Table 4.2: Performance results from gathering information for all lipids (cluster s) in a single image.

Method	**% time**
ReadXPM	3.24
XPMtoIntensityImage	19.23
movmap	6.75
HKid4_NEWS	4.22
reLabel	5.73
AvgTextureofPixel	0.76
Radius	0.01
CreateOverlay	20.68
WriteOutputPPM	12.40
Total Time (in seconds)	
CPU:	1.62
USER:	4.34
Wall-clock:	6.54

ure 4.1). In contrast to the example in the previous section, not only is this image larger, but also it contains more information which must be processed by ICAT's implementation of EHK (see Table 4.2).

Since the entire image has been utilized, the consumption of time among the methods has been redistributed. Methods used in the creation of resulting images (**CreateOverlay** or **WriteOutputPPM** for example) now operate across the entire image, reducing **ReadXPM**'s contribution. **XPMtoIntensityImage** is not affected as much, since the entire image is converted to an intensity image for later use in creating overlay images. **HKid4_NEWS** uses a larger percentage of the time than in the previous example, but this correlates to a larger data set

Table 4.3: The performance from the analysis of a sub-section of an enlarged sub- section of Figure 4.1.

Method	**% time**
ReadXPM	3.56
XPMtoIntensityImage	21.20
movmap	7.44
HKid4_NEWS	4.64
reLabel	6.32
AvgTextureofPixel	0.83
Radius	0.01
CreateOverlay	22.78
WriteOutputPPM	13.67
Total Time (in seconds)	
CPU:	4.31
USER:	12.19
Wall-clock:	17.14

being used.

4.2.3 Scalability

A subsection of Figure 4.1 has been enlarged to 1152×900 in size to provide an example of scalability (see Figure 4.3). This example also utilized a choice of intensity range from 120 to 255, inclusive. Despite the fact that this image is roughly 260% larger than the image in Figure 4.1, the percentages of time associated with EHK (i.e. **HKid4_NEWS**) remain comparable to that of the previous example. The performance results are shown in Table 4.3.

A comparison between Table 4.3 and Table 4.2 demonstrates the $O(n)$ opera-

Figure 4.3: A portion of Figure 4.1 enlarged to 1152×900.

tion time of this implementation of EHK. For example, even though this image is much larger than the image used in the previous example, the percentage of time used by **HKid4_NEWS** differs only by 0.42 seconds of wall-clock time. This results from the fact that EHK is dependent on the amount of data involved relative to the original image and not just the image itself. As the performance for each field deviates little between these tables, this further demonstrates the robustness of this implementation.

4.2.4 Entire Image

The final assessment of the division of labor for the C++ implementation of ICAT is obtained by analyzing all intensities of Figure 4.1. The result of this approach is the identification of one cluster equal in size to the original image. Such an analysis

Table 4.4: The analysis of the entire image utilizing all intensities.

Method	**% time**
ReadXPM	1.52
XPMtoIntensityImage	9.02
movmap	4.76
HKid4_NEWS	10.84
reLabel	4.12
AvgTextureofPixel	19.14
Radius	0.01
CreateOverlay	12.89
WriteOutputPPM	6.56
Total Time (in seconds)	
CPU:	1.63
USER:	9.87
Wall-clock:	11.97

can be used to assure the robustness and efficiency of methods such as **reLabel**, **AvgTextureofPixel**, and any implementation of different neighborhood rules within EHK. The performance results are shown in Table 4.4.

As might be expected, methods associated with EHK dominate when the entire image is processed. Whereas methods such as **ReadXPM** consume an amount of CPU-time relative to the size of the whole image, methods such as **HKid4_NEWS** and **AvgTextureofPixel** depend on the number of pixels selected for analysis. As this example shows, the larger the data set under analysis, the larger the cost in such functions.

4.3 Options Which Reduce Efficiency

There are two options provided by ICAT which can degrade performance: selection of cluster size ranges and utilizing multiple intensity ranges. The selection of cluster size ranges (Chapter 3) can require additional CPU-time due to an this increase in the number of comparisons which must be made. For example, the method **CreateOverlay** utilizes a list of cluster size ranges to determine if clusters identified within the EHK implementation should be displayed. Thus, every cluster in the **csize** array must be checked against this list. This effect is only noticeable, however, if several such ranges are specified. The other situation is defining more than one intensity range. This increases the overall wall-clock time, for each intensity range selected causes ICAT to run a separate analysis. Despite the increase in wall-clock time, however, these options may help increase productivity by allowing the definition of multiple scenarios.

To qualify the above claim, the example in Section 4.1.2 was run using two intensity ranges (120 to 200 and 201 to 255, inclusive) instead of one (120 to 255, inclusive). This guarantees that the same area will be covered as in the original example. The original analysis required 6.54 wall-clock seconds to run, whereas this second example required 11.04 seconds. This can be attributed to the fact that EHK was performed upon two different intensity maps resulting from the two different intensity ranges. The same example from Section 4.1.2 was run using multiple cluster size ranges. Using just one range (50 to 1000) did little to affect the overall wall-clock time (6.63 seconds). Extra ranges were used in subsequent analyses; two ranges (50 to 1000 and 1001 to 2000) required 7.77 seconds and three ranges (50 to 1000, 1001 to 2000 and 2001 to 3000) required 8.19 seconds.

Chapter 5

Conclusions and Future Work

The primary task of this thesis was to develop a general cluster analysis environment, specifically for use in the analysis of images from diabetic retinopathy. This task became two-fold: develop an implementation of the Enhanced Hoshen-Kopelman (EHK) which was adaptable to new patterns of cluster identification and new metrics for quantification of clusters and implement a graphic user interface (GUI) to facilitate the use of features. To this end, a C++ implementation of EHK was developed around six different neighborhood rules. Within each of these implementations, new data structures were used to allow the accumulation of data without hindering the speed of the algorithm. When combined with increased user control over choices during pre-processing (i.e., scoping and choice of intensity ranges), the end result was an relatively efficient and robust implementation. Combined with the development of a GUI, this provided an interactive medium for either processing an image singularly and with fine control over detail or processing several images in series. Finally, this implementation maintained flexibility to facilitate the addition of new metrics and neighborhood rules

Some additions could be made in the course of future work which would expand the usefulness of the ICAT. With respect to the C++ implementation, some reduction in memory usage is possible (at a possible reduction in speed, however). Furthermore, additional interpreters for different image types (i.e., **GIF** or **JPEG**) would extend ICAT to more platforms. With respect to the GUI, other image enhancement options could be introduced to further aid a clinician in observing fine detail within an image.

Bibliography

Bibliography

[BCM94] M. Berry, E. Comiskey, and K. Minser. Parallel Analysis of Clusters in Landscape Ecology. *IEEE Computational Science and Engineering*, 1(2):24–38, 1994.

[Bre91] G.H. Bresnick. Diabetic retinopathy. In J.R. Heckenlively and G.B. Arden, editors, *Principles and Practive of Clinical Electro-physiology of Vision*. Mosby Year Book, St. Louis, 1991.

[BW98] M. Berry and D. Westerman. Cluster Analysis for Diabetic Retinopathy. *Medical Modeling in Medical and Health Sciences, Vanderbilt Press*, 1998. In Press.

[CBZ97] J.M. Constantin, M.W. Berry, and B.T. Vander Zanden. Parallelization of the Hoshen-Koppelman Algorithm Using a Finite State Machine. *J. of Super. Applic. and High Perf. Comput.*, 11(1):31–45, 1997.

[GMT+96] A. Gupta, S. Moezzi, A. Taylor, S. Chatterjee, E. Hunter, and R. Jai n. Content-Based Retrieval of Opthamalogical Images. *IEEE International*

Conference on Image Processing, ICIP-96, 1996.

[HBM97] J. Hoshen, M.W. Berry, and K.S. Minser. Percolation and Cluster Structure Parameters. II. The Enhanced Hoshen-Kopelman Algorithm. *Physical Review E*, 56(2):1455–1460, 1997.

[HK76] J. Hoshen and R. Kopelman. Percolation and Cluster Distribution. I. Cluster Multiple Labeling Technique and Critical Concentration Algorithm. *Physical Review*, B(14):3438–3445, 1976.

[Jai89] Anil K. Jain. *Fundamentals of Digital Image Processing.* Prentice-Hall International, Inc., Englewood Cliffs, 1989.

[KD96] V.P. Kumar and U.B. Desai. Image Interpretation Using Bayesian Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(1):74–77, 1996.

[OL93] R.W. Olk and C.M. Lee. *Diabetic Retinopathy: Practical Management.* J.B. Lippincott Company, Philadelphia, 1993.

[PG93] R.E. Plotnick and R.H. Gardner. Lattices and Landscapes. *American Mathematical Society Series: Lectures on Mathematics in the Life Sciences*, 23:129–17, 1993.

[Pra91] W.K. Pratt. *Digital Image Processing.* John Wiley, New York, second edition, 1991.

[PW85] N.H. Packard and S. Wolfram. Two-Dimensional Cellular Automata. *J. Statistical Physics*, 38:901–946, 1985.

[SA93] D. Stuaffer and A. Aharony. *Introduction to Percolation Theory.* Taylor & Francis Ltd., London, 1993.

Vita

Dax Marek Westerman was born in Havre De Grace, Maryland on February 13, 1973. He graduated from Greeneville High School (Greeneville, Tennessee) in 1991, and received a Bachelor of Science in Mathematics from the University of Tennessee, Knoxville in 1996. In August of 1996, he entered in the Computer Science graduate program at the University of Tennessee, Knoxville, and received a Master of Science degree in Computer Science in May 1998.