University of Alberta

Development of Feature Extraction Techniques with Immune Programming

by

Reid Orsten 

A thesis
submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree
of Master of Science

Department of Electrical and Computer Engineering
Edmonton, Alberta
Fall 2005

Your file Votre référence
ISBN: 0-494-09251-3
Our file Notre référence
ISBN: 0-494-09251-3

UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Reid Orsten
TITLE OF THESIS: Development of Feature Extraction Techniques with Immune Programming

DEGREE: Master of Science
YEAR THIS DEGREE GRANTED: 2005

(Signed) _____

Permanent Address:

Date: _____

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Development of Feature Extraction Techniques with Immune Programming** submitted by **Reid Orsten** in partial fulfillment of the requirements for the degree of Master of Science.

Petr Musilek (Supervisor)

Marek Reformat

John Whittaker

Date: _____

Abstract

An algorithm is presented which replaces the pixel-mapping operation in the Hough transform. Instead of performing the complete mapping, it estimates a range of parameter space and only performs the mapping in this range. It is somewhat able to adapt to image noise, and in high noise cases reverts to the full Hough transform with little additional overhead. The expression to estimate the range was generated with immune programming, and the performance of various clonal selection algorithms was compared and analysed on its components.

Contents

List of Figures

Figure

List of Tables

Table

Chapter 1

Introduction

Computer vision is an extremely exciting concept. The degree of environmental awareness that functional computer vision would allow would cause staggering advances in a huge number of fields, from transportation and autonomous navigation to industrial automation as well as a plethora of consumer applications and beyond.

Unfortunately, the barriers to achieving widely functional computer vision are staggering. The greater problem can be viewed as several smaller problems: feature extraction, object recognition, and environmental awareness. Feature extraction is the process of moving from raw pixels to features: knowing the location, size, etc of a square in an image, instead of knowing an image which contains a square. Object recognition is then the process of combining features into objects: knowing that there are letters on a page instead of knowing that there are shapes in a square. Finally, environmental awareness: knowing that there is a paper in front of you instead of knowing that there are letters on a page. None of these problems have been solved.

Several techniques for feature extraction exist, but none of them are exceptionally effective [1]. Some are reasonably fast, such as RANSAC [2], but begin are

extremely sensitive to noise. Others, such as ones based on the Hough [3] and Radon [4] transforms are much less sensitive to noise, but also much slower. Significant research has been done to improve the speed of these algorithms, for example by parallelisation [5, 6, 7].

Computational intelligence is a blanket term to describe any method by which the computer learns patterns or behaviours on its own. Collections of techniques have been developed which can be surprisingly powerful, based on the operation of the brain [8], the evolution of species [9, 10], and most recently the immune system [11, 12, 13, 14].

Techniques based on the immune system are called artificial immune system (AIS) techniques, and one of the more recent ones is Immune Programming [15]. Compared to genetic programming, it seems capable of solving complex computational problems very quickly, and there seems to be room for improvement.

This thesis presents a Hough-like feature extraction algorithm which relies on expressions generated by immune programming to reduce the amount of work required by choosing ranges of values to operate on. The ability of various types of immune programming to generate solutions is also examined in some detail.

Chapter 2

Background

Some knowledge of fuzzy logic, image processing, evolutionary computation and artificial immune system methods is required to understand the material to follow. This section attempts to provide an introduction to the material which should allow a reasonably knowledgeable reader to follow along.

2.1 Fuzzy Logic

Sometimes, applying boolean logic doesn't make sense. This happens frequently when working with imprecise data, most infamously with linguistic descriptions. Borel [16] coined the classic example:

> One seed does not constitute a pile nor two nor three [...] from the other side everyone will agree that 100 million seeds constitute a pile. What therefore is the appropriate limit? Can we say that 325 647 seeds don't constitute a pile but 325 648 do?

Image processing, specifically feature extraction, is another area where crisp techniques frequently fail. Noise in the image makes the nature and placement of

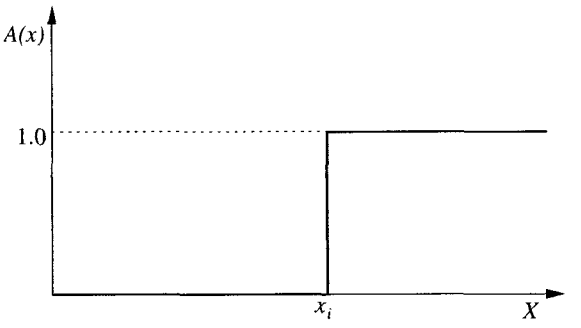

Figure 2.1: A crisp version of "is a pile"...

features more ambiguous. An example will be presented in Section 2.3.2.

Fuzzy sets, first introduced by Zadeh [17], provide an extremely intuitive method for dealing with uncertainty and descriptions without crisp boundaries. It does this by introducing the notion of partial membership. Consider a crisp set A inside a universe X: any element x_i would either be in A ($A(x_i) = 1$) or not ($A(x_i) = 0$). With a fuzzy set, $A(x_i)$ could take on any value from 0 to 1, representing a smooth continuum of partial memberships between "completely not in" and "completely in." Consider Figure 2.1. This could represent a crisp version of the seed problem. Here, x_i is the point that Borel mentioned, where with the addition of one seed, a pile appears. Figure 2.2 shows a fuzzy set equivalent, which has a smooth transition from "not a pile" to "a pile."

The fuzzy version is much more believable than the crisp version. In this case, x_i is the first number of seeds which would definitely be called "a pile." If one seed is removed, it is still mostly a pile, but slightly less so.

There is an important distinction to be made between the meaning of a partial membership and a probability, which is a tempting comparison to make. Bezdek [18] offers an example which clarifies the difference. A person is dying of thirst in the

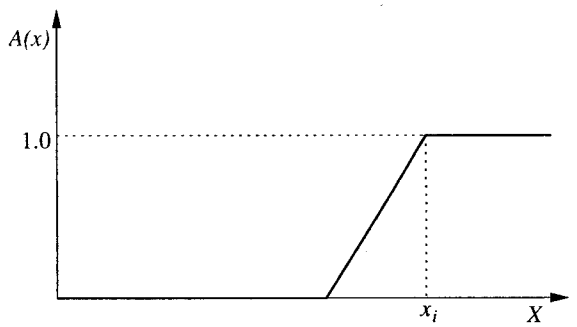

Figure 2.2: ...and its more believable fuzzy equivalent.

desert, and they are presented with two bottles: bottle A, labeled "$\mu_{potable} = 0.9$" (fuzzy membership in the set of potable fluids); and bottle B, labelled "$P_{potable} = 0.9$" (probability of being a potable fluid). Bezdek states that the person should choose bottle A, since it will contain something like swamp water, while bottle B has a ten percent chance of being deadly poison.

2.1.1 Operations on Fuzzy Sets

As with normal sets, fuzzy sets have operations for union, intersection, and negation. Just as crisp sets are a subset of fuzzy sets, so are crisp set operations subsets of fuzzy ones:

$$(A \cap B) = \min(A(x), B(x)) = A(x) \wedge B(x) \tag{2.1}$$

$$(A \cup B) = \max(A(x), B(x)) = A(x) \vee B(x) \tag{2.2}$$

$$\bar{A}(x) = 1 - A(x) \tag{2.3}$$

An entire class of functions frequently used with fuzzy sets are triangular norms (t-norms) and triangular co-norms (s-norms). Both are commutative, associative

S-Norm	T-Norm
$0\ \mathrm{s}\ x = 0$	$0\ \mathrm{t}\ x = x$
$1\ \mathrm{s}\ x = x$	$1\ \mathrm{t}\ x = 1$

Table 2.1: S- and T-Norm Boundary Conditions

and monotonic; they differ only in their boundary conditions, which are shown in table 2.1.1.

2.1.2 Comparing Fuzzy Sets

Since equality is a crisp measure, it is less meaningful when applied to fuzzy sets or fuzzy membership values. For example, consider the universe consisting of the set A, B, C, a fuzzy set A with membership values $0, 0.5, 1$ and another fuzzy set B with membership values $0, 0.49, 1$. The two are clearly not equal, but they are almost identical.

To account for this, we start with the following definition:

$$(A \equiv B)(x) = 0.5\{[A(x)\phi B(x)] \wedge [B(x)\phi A(x)] + [\bar{A}(x)\phi \bar{B}(x)] \wedge [\bar{B}(x)\phi \bar{A}(x)]\} \quad (2.4)$$

Where ϕ is residuation, defined as:

$$A(x)\phi B(x) = \sup_{c \in [0,1]} [A(x)\mathrm{t}c \leq B(x)] \quad (2.5)$$

With t defined as $\max(0, x + y - 1)$, this simplifies to equation 2.6, a measure of equality frequently referred to as resemblance.

$$(A \equiv B)(x) = 1 - |A(x) - B(x)| \quad (2.6)$$

-1	0	1
-2	0	2
-1	0	1

Figure 2.3: The Sobel mask for the horizontal derivative

2.2 Edge Detection

An edge detection algorithm is one which, given an input image, returns another image which has bright pixels on edges in the original image. This is usually done by convolving the input image with a mask which approximates a derivative of the image at the centre pixel.

One simple edge detection algorithm is the Sobel algorithm. It works with two masks, which calculate the numerical first derivative horizontally and vertically. These masks are shown in Figures 2.3 and 2.4 respectively

If G_h is the brightness of the centre pixel after convolution with the horizontal mask, and G_v is the brightness for the vertical mask, the brightness of that pixel in the Sobel edge image will be the value of G from equation 2.7.

$$G = |G_h| + |G_v| \tag{2.7}$$

Another common edge detection algorithm is the Laplacian, which convolves the image with a single mask which estimates the numerical second derivative. Since this algorithm uses a 5x5 mask instead of a 3x3 mask, it is far more sensitive to noise

-1	-2	-1
0	0	0
1	2	1

Figure 2.4: The Sobel mask for the vertical derivative

-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
-1	-1	24	-1	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1

Figure 2.5: The mask used in the Laplace edge detection algorithm

than the Sobel algorithm. Figure 2.5 shows the mask for the laplacian.

There are a large number of edge detection algorithms, and there are entire books on the subject.

2.3 The Hough Transform

The Hough Transform is an essential algorithm in computer vision. It is an algorithm to help find parametric features in an image.

Consider a straight line. The most common parameterisation for straight lines is shown in Figure 2.6: a vector from one corner of the image to the closest point on the line.

$$\rho = x\cos(\theta) + y\sin(\theta) \tag{2.8}$$

where ρ is the length and θ is the angle from the origin.

Consider a pixel (x_i, y_j) in an image. The solutions to equation 2.9 represent all of the possible lines which pass through (x_i, y_j), If one takes a second point (x_k, y_l), the point where the curves for the two points intersect will be the parameterisation of the line connecting those two points, as illustrated by Figure 2.6.

$$\rho = x_i\cos(\theta) + y_j\sin(\theta) \tag{2.9}$$

The Hough transform starts with an edge image and a second, empty image

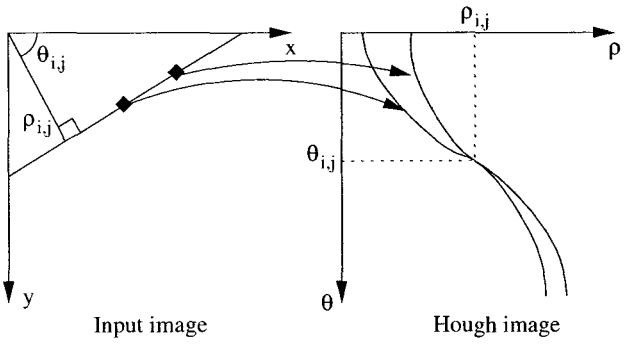

Figure 2.6: Co-linear pixels and their parameter-space mappings

representing the parameter space for features in the image. For every pixel (x_i, x_j) in the image, the brightness of the pixel in the edge image is added to every pixel in the parameter image which lies under the curve. Once this is complete, the brightest points will represent the parameterisations of the lines crossing over the largest number of edge pixels and therefore will represent the most prominent lines in the image.

The size of the parameter image must be chosen somewhat carefully. It must be large enough to distinguish the smallest feature required anywhere on the image. For example, two parallel lines that are very close to each other might appear to be one line if the parmeter space image doesn't have sufficient resolution to distinguish them. On the other hand, if the parameter space image is too large, somewhat noisy lines will appear as a large number of almost colinear lines. Additionally, the nature of the parameterisation means that the corner closest to the origin will have a higher resolution in parameter space than the corner opposite it [19]. Therefore, in some cases it is possible for both problems to occur in opposite corners of the image. This can be countered by having a non-linear parameter space image, at the cost of increased computational complexity, or using an alternate parameterisation.

In general, the size of the parameter image must grow with the size of the input image, or increasing the image size would not improve accuracy. This brings up one of drawbacks of the Hough transform, which is its speed. Assume that the length of the Hough image on the ρ-axis grows proportionally to the diagonal length of the original image. Also, assume that the input image has a fixed aspect ratio. The ρ-axis length therefore grows with the root of the number of pixels in the input image. Since the length of the θ-axis should grow in similar proportion, the amount of work required to plot each pixel's curve in Hough space also increases, since it involves, in the fastest case, $\sqrt{n}$ calls to Bresenham's line algorithm [20], or a more expensive anti-aliased line algorithm. Since the $\sqrt{n}$ calls will result in $O(\sqrt{n})$ pixels being drawn, they are treated as being constant time, meaning that the Hough transform's speed is slightly worse than $O(n^{\frac{3}{2}})$. Since the atomic units are trigonometric functions, the algorithm can become very slow.

Although the Hough transform will never fail to find an answer, problems arise when dealing with noisy data, as shown in [21]. If edge pixels aren't precisely colinear, the Hough transform might consider a noisy line to be several lines, and the maxima in parameter space might not correspond to the best shape in the input space.

Distributed voting is one method for dealing with this noise: instead of casting a vote for a specific curve in the parameter space, each point casts votes for a set of curves nearby, with vote strength decreasing with distance. The most computationally efficient method would be to convolve the parameter space with a mask before searching for maxima. Han et. al. [21] demonstrated that it sufficient to perform a 1-dimensional convolution along the ρ-axis when searching for lines and circles.

2.4 Evolutionary Computing

Evolutionary computing is a collection of computational intelligence techniques that imitate natural evolutionary processes to develop solutions to a problem. They all follow a similar form.

The basic requirements for any evolutionary algorithm are a method of encoding solutions and a method of evaluating their fitness. For example, if one was using an evolutionary algorithm to fit some data to a polynomial of the form $y = ax^3 + bx^2 + cx + d$, one could encode solutions as a vector of the form $\langle a, b, c, d \rangle$. An appropriate measure of fitness in this case would be the average error magnitude over a non-contiguous subset of the data, with 0 being a perfect fit.

An evolutionary algorithm starts by generating a population of random individuals. The fitness of these individuals is calculated, and another population is generated from the individuals with the best fitness. The process is reapeated until a maximum number of generations has been reached or the desired goal has been achieved.

The variation in evolutionary approaches centres around how new populations are generated from old ones. Traditionally, generating a new population primarily involved crossover, with some amount of mutation. Crossover in this sense is inspired by crossover between chromosomes that occurs in sexual reproduction: the two individuals are "snipped" and the resulting segments are mixed. Along with mutation at rates comparable to biological systems, the overall effect is an iterative improvement in fitness similar to that witnessed in natural selection. Eventually, and with some luck, the population will generate the desired solution.

2.4.1 Clonal Selection

More recently, another evolutionary-style algorithm has emerged as part of another set of techniques which imitate the human immune system, collectively called Artificial Immune System [14]. This technique, called clonal selection, mimics the evolution of T lymphocytes in the immune system. T lymphocytes have antigen receptors, which latch onto and immobilise antigens, so they can be destroyed by B lymphocytes. Instead of using crossover, clonal selection uses cloning and hypermutation. Hypermutation varies from mutation as it appears in normal evolutionary algorithms, since instead of being a probability that an individual will contain a mutation, it is the probability that an element within an individual will be mutated.

Artificial clonal selection uses a repertoire of individuals, just as genetic algorithms use a population. The individuals in the repertoire are indistinguishable from those used in genetic algorithms. The measure of an individual's quality in clonal selection is called an affinity measure.

One method of generating a new repertoire via clonal selection first involves sorting individuals by affinity. Then, the population is partitioned into three sections. The top section, consisting of the top performers, is cloned into the new population. The second section is hypermutated into the new repertoire, and the bottom section is replaced.

Another technique, developed by Musilek et al, [15], uses a more complex probabilistic approach. It doesn't require that the population be sorted by affinity, but it does require that the population affinities be normalized. It requires three parameters: the probability of replacement (P_r), the probability of cloning (P_c) and the probability of hypermutation (P_m). This technique first generates a random number

in the range of $[0, 1]$ and compares it to P_r. If the random number is less, it generates a random individual and adds it to the new repertoire. Otherwise, it selects a member of the current repertoire (it chooses them sequentially) and generates another random number. If the random number is less than the individual's normalized affinity (so the best member of the population, if it is selected, will always pass this test), then it is a candidate to go on to the next repertoire. A random number is generated, and if it is less than P_c, the member is cloned into the new repertoire; otherwise, another random number is generated and compared to P_m to determine if a hypermutated version will be added to the population.

The probabilistic approach has several advantages over the partition method for some classes of problems. It maintains a more diverse population and is therefore less prone to getting stuck at local minima. On most problems it also tends to converge more quickly than partition-based algorithms.

2.4.2 Genetic Programming

Another class of evolutionary algorithms fall into the category of genetic programming. Here, instead of having individuals representing parameters, the individuals are programs. Otherwise, there is no difference in the process, except that individuals must be executed in order to evaluate their performance.

Generally, genetic programs consist of a string of characters, representing operations and data variables. Also, genetic programs tend to work with very simple architectures, usually tree- or stack-based virtual machines.

In the most basic form of genetic programming, several new difficulties can arise. For example, individuals can actually be invalid, if they represent code which cannot actually execute. This can prevent the algorithm from converging, since it

effectively reduces the population size. It also makes it less likely that mutations will be meaningful improvements if they can invalidate the individual.

One way around the problem of program validity is the use of K-Expressions, as presented by Ferreira [10]. A K-Expression is a way of organising a genes that ensures validity and completeness. It breaks each gene into two parts, a head and a tail. The head can contain operations and data variables. The tail's length is determined by the length of the head, and can contain only data variables. To execute a K-Expression, the head and tail are concatenated and converted into a tree. Since the tail cannot contain operations, all leaves in the tree are guaranteed to be data variables. The tree is then evaluated with a post-order traversal. Since the tree cannot have any cycles, K-Expressions cannot get stuck in an infinite loop.

2.5 Convergence in Evolutionary Algorithms

Due to the complexity of evolutionary algorithms and the problems they deal with, it is very difficult to predict – or even understand – their performance on any given problem. In a broader sense, however, some theory and terminology have been developed.

No Free Lunch theorems [22] explain the inconsistency in performance. In short, these state that the performance of all search algorithms, averaged over all cost functions, is equal. In other words, for every family of cost functions where genetic programming performs well, there is another where it performs dismally. The average performance of genetic programming – or any other search algorithm, for that matter – is identical to the average performance of a random search.

In order to properly explain No Free Lunch theorems, the concepts of explo-

ration and exploitation must be presented. An exploration-based search will not rely on existing knowledge. Random and sequential searches are entirely based on exploration: there's no assumption that the $(n+1)$th element will be any more or less a valid solution than the nth. Exploitation, on the other hand, does make use of existing knowledge. Binary search is exploitative: it assumes that the solution space has order. Grabbing the $(n+1)$th element of an array must give a larger result than grabbing the nth, or the search will fail.

The speed advantage of exploitative algorithms is derived from the knowledge implicitly present in the algorithm, and that advantage will quickly deteriorate if the knowledge is less meaningful. For example, in a backpropagation based neural network, the value of the learning rate has a huge impact on the network's performance. In essence, it is a parametric assumption about the smoothness of the solution space. If this assumption isn't valid, the network will perform poorly.

For evolutionary algorithms, the fitness function is the source of implicit knowledge. Their power lies in that the exploitative element isn't deterministic, but rather is used to guide a random search. Therefore, the two factors affecting the performance of an evolutionary algorithm are the degree to which the knowledge in the fitness function is exploited in choosing the next generation and the degree to which the fitness function actually reflects the quality of an individual.

This brings up an important issue with genetic programming, or any evolutionary algorithm which determines structure. In most cases, numerical performance is the only computationally feasible fitness function, but, especially in structure problems, numerical performance can be poorly correlated with actual performance.

Chapter 3

Algorithm

The intent of this thesis is to design a more computationally efficient version of the mapping of a pixel from an image into the parameter space. The existing version, which – for straight lines – involves plotting the curve $\rho = x_i \cos(\theta) + y_j \sin(\theta)$ into the parameter space image, has a computational complexity of $O(\sqrt{n})$. However, in most cases, there is at most one line passing through each pixel, so most of this work is redundant.

By designing a constant time calculation to approximately determine the angle of any line passing through a given pixel, it would be possible to plot only a small section of the curve for each pixel, or – if the algorithm is accurate enough – to plot a computationally inexpensive approximation.

The algorithm is broken into two key sections, which operate on 5x5 subimages. The first calculates two values: the weighting to apply to the vote of the subimage's centre pixel – equivalent to calculating it's edge strength – and a signal-to-noise measure η.

If the vote strength is above a certain minimum threshold, the second section

of the algorithm will estimate the angle θ of a line passing through the centre of the 5x5 subimage. This involves scaling the pixel values in the subimage to the range of pixel values $[0, 1]$, η as calculated above can be used as the input to an empirically determined profile $n(x)$ of how the algorithm's performance degrades in the presence of additive gaussian noise. This provides a range $[\theta - n(\eta), \theta + n(\eta)]$ that should contain the actual angle of a line passing through the image.

The angle approximation part of the algorithm is discovered using clonal selection.

3.1 Strength and Certainty Estimation

The first value calculated for the strength estimation is the edginess of the pixel at the centre of the subimage, S. This is equivalent to convolving the centre pixel and its neighbours with a 3x3 Sobel mask, as described earlier.

If S is less than some threshold, no further calculation is performed for the subimage. Otherwise, the pixel values of the subimage are scaled to the range $[0, 1]$ and the average magnitude of the second derivative is calculated in four directions: horizontally, vertically, and diagonally up and down (from left to right).

$$d_h = \frac{1}{30} \sum_{i=0}^{4} \sum_{j=0}^{2} |p_{i,j} - 2p_{i,j+1} + p_{i,j+2}| \tag{3.1}$$

$$d_v = \frac{1}{30} \sum_{i=0}^{4} \sum_{j=0}^{2} |p_{j,i} - 2p_{j,i+1} + p_{j,i+2}| \tag{3.2}$$

$$d_d = \frac{1}{18} \sum_{i=0}^{2} \sum_{j=0}^{2} |p_{i,j} - 2p_{i+1,j+1} + p_{i+2,j+2}| \tag{3.3}$$

$$d_u = \frac{1}{18}\sum_{i=2}^{4}\sum_{j=0}^{2}|p_{j,i} - 2p_{j-1,i+1} + p_{j-2,i+2}| \tag{3.4}$$

If the image is noise-free with a strong line passing through the centre, at least one of these directions will be roughly colinear with it, so the average magnitude of the second derivative should be small. When noise is added, the smallest of these values will tend to be larger. Therefore η, as defined by equation 3.5, where $d_{max} = \max(d_h, d_v, d_u, d_d)$, will tend to be very small in clean images and significantly larger in noisy ones.

$$\eta = \frac{d_h d_v d_u d_d}{(d_{max})^4} \tag{3.5}$$

To generate the noise profile presented in the analysis, a 5x5 array of 0-mean random numbers was generated and added with saturation to the pixels of the subimage. Figure 3.1 shows a scatter plot of η versus the average magnitude of this array. As the diagram shows, small values of η are strongly correlated to clean images.

The certainty measure involves fitting a correlation between η and a profile of how the angle approximation's performance degrades with noise. The details of this calculation appear in the analysis section.

3.2 Angle Approximation

The angle approximation operates on the resemblance of pixels diametrically opposite the centre pixel in the image. Before calculating resemblances, the pixel values are to the range $[0, 1]$, which can be treated as a membership value in the set "white." Initially, the angle approximation was intended to be a single expression which would operate over the entire range from $[0, \pi]$, but in initial experiments its performance was dismal. Instead, it was broken into two separate expressions: one

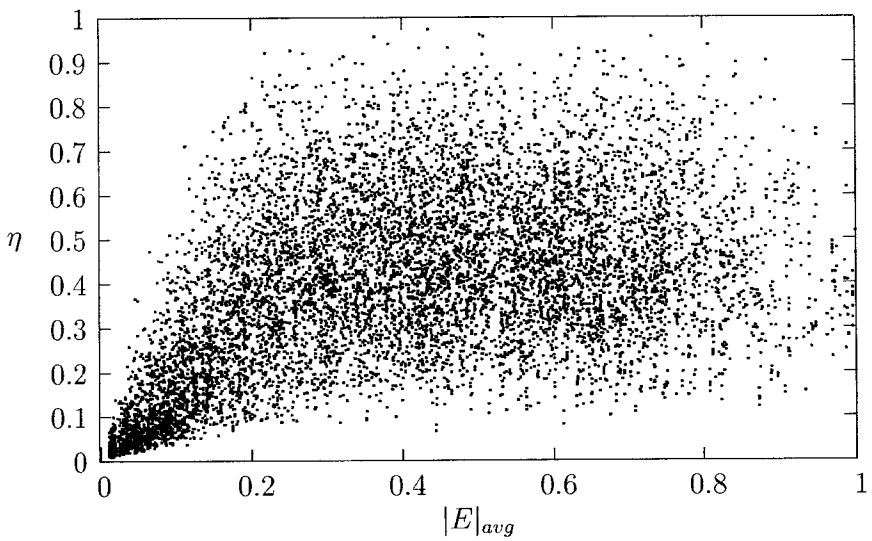

Figure 3.1: η versus average noise magnitude.

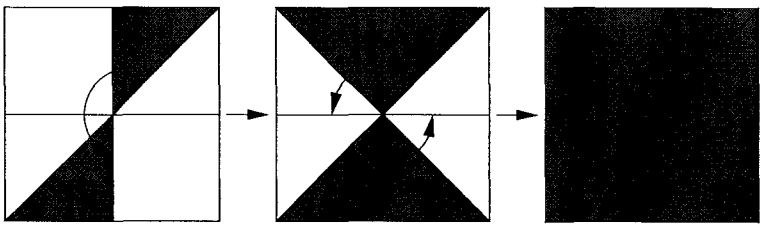

Figure 3.2: Solving for the remaining quadrants.

which approximates the angle of lines between $[0, \pi/4]$ (θ_1), and one which operates over the entire range of $[0, \pi]$ but only identifies if a line falls in the range of $[0, \pi/4]$ (q_1). When combined, these two expressions will identify and estimate angles in one quadrant of the problem space. The two expressions in the angle approximation have been discovered using clonal selection. Flipping the subimage vertically will yield the expressions for $[3\pi/4, \pi]$ (θ_4 and p_4), and by flipping the subimage across the diagonal $y = x$, the expressions for the remaining two quadrants are discovered. These steps are clarified in Figure 3.2.

Since resemblance is commutative and the image as presented as the resemblance of pixels on opposite sides of the centre pixel:

$$r_{i,j} = p_{i,j} \,\mathrm{R}\, p_{4-i,4-j} = r_{4-i,4-j} \tag{3.6}$$

Therefore, flipping the image vertically is equivalent to flipping the resemblances horizontally, so $r_{i,j}$ becomes $r_{4-i,j}$.

For a given subimage I, let q_{max} be number of the quadrant identifier which returns the largest value; for example, q_{max} will be 1 if $q_1(I)$ is the largest. The final output of the angle approximation will then be:

$$A(I) = \begin{cases} \theta_1(I) & q_{max} = 1 \\ \frac{\pi}{2} - \theta_2(I) & q_{max} = 2 \\ \frac{\pi}{2} + \theta_3(I) & q_{max} = 3 \\ \pi - \theta_4(I) & q_{max} = 4 \end{cases} \tag{3.7}$$

3.3 Mapping Into Parameter Space

Given the strength and angle range from the strength estimator and the angle estimator, mapping the pixel into parameter space can be done more efficiently.

In higher noise cases, where the angle range is larger, the mapping process would remain the same, except that the curve would only need to be plotted for a section of the parameter space.

In cases where the range is smaller it is sufficient to replace equation 2.9 with a first or second order taylor series expansion centred around the estimated angle. The small error induced by this can be countered with the fuzzy distributed voting method described in [21].

3.4 The Complete Algorithm

The complete algorithm is as follows:

(1) Examine the first pixel where it is possible to select a 5x5 subimage with that pixel at the centre.

(2) Calculate the Sobel edge strength. If less than some minimum threshold, go on to the next pixel.

(3) Scale the pixels in the subimage to the range $[0, 1]$

(4) Calculate η.

(5) Evaluate the four quadrant classifier expressions.

(6) Evaluate the angle approximator expression corresponding to the quadrant classifier with the highest output.

(7) Calculate the range of angles to plot based on the estimated angle and η. This step depends on the specific expressions, and is done later on.

(8) Add the edge strength to every parameter image pixel under the relevant portion of equation 2.9.

(9) Repeat steps 2 through 8 for all pixels which can be the centre of a 5x5 subimage.

Chapter 4

Experimental Design

The angle approximation for $[0, \pi/4]$ and the expression to identify the first quadrant in the range $[0, \pi]$, were grown by clonal selection. This section explains the data that was used to generate the expressions, as well as the K-Expression grammar composition and clonal selection algorithms used.

4.1 Training Data

The training data for both expressions was a set of resemblances for 5x5 images with lines passing through them, in the form shown in Figure 4.1. This form of data was used to ensure that the expression would remain accurate even if the line were not passing directly through the centre of the image. For both expressions, the range of displacements (ρ) was $[0, 1/\sqrt{2}]$ pixels, which is the longest unique length. Anything larger than that will overlap with an adjacent pixel. For the angle approximation, the angle (θ) varied from $[0, \pi/4]$, and for the quadrant classifier, $[0, \pi]$.

Due to the symmetry of the resemblance data, as well as due to the commu-

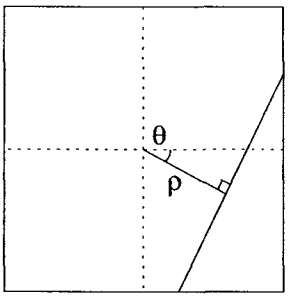

Figure 4.1: Training data parameterisation.

tativity of fuzzy resemblance operations, the data for images (ρ, θ) and $(-\rho, \theta)$ is identical. Therefore, the training data covers the entire range of images that the algorithm could be presented with.

4.2 AIS Configuration

For training both expressions, the K-Expression grammar contained a basic set of fuzzy set operations, one contrast enhancing operation (sin), the data variables, and a handful of constants. In earlier experiments for the quadrant classifier expression, drastic sum and product were added to the grammar, since they allow abrupt changes in value. They were removed since they lead to extremely noise-sensitive expressions, and the addition of noise to prevent sensitivity harmed convergence. See table 4.2.

One of the distinct features of this grammar is the fact that elements are weighted. The rationale for this decision lies in the number of operations compared to the number of data variables. In most experiments presented in the literature, there are between one and three data variables. In this experiment, there are seventeen, including constants, compared to 7 operations in the angle approximation. If all members of the grammar were evenly weighted, this would make it needlessly difficult

Symbol	Meaning	Arity	$f(\bar{x})$	Weight
R	Resemblance	2	$1 - \|x_1 - x_2\|$	5
$\cup$	Bounded Sum	2	$x_1 + x_2 - x_1 x_2$	5
$\cap$	Bounded Product	2	$x_1 x_2$	5
$\vee$	Max	2	$\max(x_1, x_2)$	5
$\wedge$	Min	2	$\min(x_1, x_2)$	5
$\sqcup$	Drastic Sum	2		3 (0)
$\sqcup$	Drastic Product	2		3 (0)
$\neg$	Negation	1	$1 - x_1$	5
sin	Sine	1	$\sin(\pi x_1)$	2
$r_{1,1}$	Data variable	0	$p_{1,1}\,\mathrm{R}\,p_{5,5}$	1
$r_{2,1}$	Data variable	0	$p_{2,1}\,\mathrm{R}\,p_{4,5}$	1
$r_{3,1}$	Data variable	0	$p_{3,1}\,\mathrm{R}\,p_{3,5}$	1
$r_{4,1}$	Data variable	0	$p_{4,1}\,\mathrm{R}\,p_{2,5}$	1
$r_{5,1}$	Data variable	0	$p_{5,1}\,\mathrm{R}\,p_{1,5}$	1
$r_{1,2}$	Data variable	0	$p_{1,2}\,\mathrm{R}\,p_{5,4}$	1
$r_{2,2}$	Data variable	0	$p_{2,2}\,\mathrm{R}\,p_{4,4}$	1
$r_{3,2}$	Data variable	0	$p_{3,2}\,\mathrm{R}\,p_{3,4}$	1
$r_{4,2}$	Data variable	0	$p_{4,2}\,\mathrm{R}\,p_{2,4}$	1
$r_{5,2}$	Data variable	0	$p_{5,2}\,\mathrm{R}\,p_{1,4}$	1
$r_{1,3}$	Data variable	0	$p_{1,3}\,\mathrm{R}\,p_{4,3}$	1
$r_{2,3}$	Data variable	0	$p_{2,3}\,\mathrm{R}\,p_{5,3}$	1
1/4	Constant	0	1/4	1
1/3	Constant	0	1/3	1
1/2	Constant	0	1/2	1
2/3	Constant	0	2/3	1
3/4	Constant	0	3/4	1

Table 4.1: Grammar composition

to generate longer K-Expressions: more than 2/3 of randomly generated expressions would not contain an operation.

For an affinity measure, numerical performance over a subset of 256 elements was used. This is larger than usual for an affinity measure, but anything less would not be large enough to reasonably represent the problem space.

For the angle approximation, the average ($E_{i,avg}$) and maximum ($E_{i,max}$) error values were recorded. The affinity for the individual was then defined in terms of the Hamming length of a vector composed of these values, as well as the performance of other members of the population, as in equation 4.2. The error vector is used as a basis for the affinity function because the expression must not just have a low average error, it must consistently perform well over the entire range of input. The additional weighting to the average error is included since, in experiments, it tended to be about 1/3 the magnitude of the maximum error, and because once the maximum error is reasonably low, improvements in the average error should take for forefront.

$$E_i = \sqrt{\|3E_{i,avg}\| + \|E_{i,max}\|} \tag{4.1}$$

$$A = 1 - \frac{E_i - E_{best}}{E_{worst} - E_{best}} \tag{4.2}$$

For the quadrant classifier, a misclassification rate was recorded, and this was used as E_i in 4.2, and A was cubed to decrease the average affinity of the population, which tended to be quite high.

4.3 Clonal Selection Algorithms

Four different selection algorithms were used, and their relative performances compared. The partition and probabilistic methods as described in the background

were used, as well as one variant of the partition method and two variants of the probabilistic method.

The modified partition method clones the top partition, and mutates the top and middle partitions, instead of only the middle. It is believed that this algorithm will achieve noticeably higher performance, since cloned individuals do not move in the problem space and therefore do not help the population to converge. By only cloning the top performers, the original algorithm removed the top performers in the population from the search.

The modified version of the probabilistic selection algorithm varies only in that it clones the top performers before selecting the remainder of the population according to the original algorithm. The benefit of this approach is that smaller improvements are guaranteed to be maintained. In the original version, these small improvements can be lost if the affinity difference is small. Also, situations can arise where the probabilistic algorithm will suddenly perform worse: if the proportion of individuals with a high affinity is small, it is possible to lose all of the best performing members entirely by chance. In fact, this was observed in several experiments.

For all experiments, the partition methods used partitions at 10% and 50% of the total population sizes. The values of P_r, P_c, and P_m for the probabilistic methods tended to be around 0.20, 0.08, and 0.16 for the original probabilistic algorithm for the angle estimator; 0.40, 0.05, and 0.25 for the variant on the estimator and 0.4, 0.02 and 0.35 for the variant on the quadrant classifier.

Chapter 5

Results

Tables 5.1 and 5.2 show performance results from the graphed runs. These results are fairly normal for these problems. It should be noted, however, that the parameters of the probabilistic algorithms were not fine-tuned, so their optimal performance could be considerably higher.

For the angle approximation, the maximum and average error was recorded for the fittest individual every generation. Figures 5.1 to 5.4 show typical results for the four clonal selection algorithms, displayed every fifth generation for the sake of neatness.

In the probabilistic algorithm, notice the occasional increases in error. This is

Selection Method	Partition	Probabilistic	Modified Part.	Modified Prob.
Head Length	32	32	32	32
Population Size	500	500	500	500
Hypermutation Rate	0.06	0.06	0.06	0.06
Gen. to $E_{avg} < 0.05$	145	125	10	10
Gen. to $E_{avg} < 0.03$	260	310	20	80
E_{avg} at G=500	0.0248	0.0280	0.00911	0.0170

Table 5.1: Clonal selection performance, angle approximation

Selection Method	Partition	Probabilistic	Modified Part.	Modified Prob.
Head Length	64	64	64	64
Population Size	1000	1000	1000	1000
Hypermutation Rate	0.04	0.04	0.04	0.04
Gen. to $E_{avg} < 0.05$	45	N/A	20	700
E_{avg} at G=1000	0.046	0.070	0.043	0.043

Table 5.2: Clonal selection performance, quadrant identifier

the a side effect of the selection algorithm, which will be discussed in more detail in the analysis chapter.

For the quadrant classifier, the misclassification rate was recorded for the fittest individual, also every generation. There is no graph for the probabilistic algorithm, because its behaviour was unpredictable and it consistently failed to converge within the allotted 500 generations. This will be discussed in more detail in the analysis section.

Equation 5.1 shows the expression for the angle approximator, and equation 5.2 shows the expression for the quadrant classifier.

$$P_1 = sin((\neg((r_{0,1} \cup r_{1,2}) \vee r_{1,2})\mathrm{R}(1/4 \vee (r_{1,1} \cap r_{1,0}))))\mathrm{R}((1/2 \cup (1/4 \cap r_{1,3})) \cap (\neg r_{0,2}\mathrm{R}r_{1,3}))$$

$$A = ((P_1 \cup (r_{0,4} \cup (r_{0,2} \vee r_{0,4}))) \cap (r_{2,1}\mathrm{R}sin(r_{0,3}))) \cup r_{1,1} \tag{5.1}$$

$$Q_1 = (r_{1,3}\mathrm{R}sin((r_{0,1} \cup ((1/3 \wedge sin(r_{1,4})) \cap (r_{0,3}\mathrm{R}r_{0,0}))))) \vee r_{1,4}$$

$$Q_2 = (3/4 \vee r_{0,0})\mathrm{R}(r_{1,0} \cup r_{1,0})$$

$$Q = sin(Q_1 \cap Q_2) \tag{5.2}$$

The output of the angle approximator over the range of angles $[0, \pi]$ and displacements $[0, \frac{1}{\sqrt{2}}]$ is shown in Figure 5.8. Since 0 and π are congruent, the approxi-

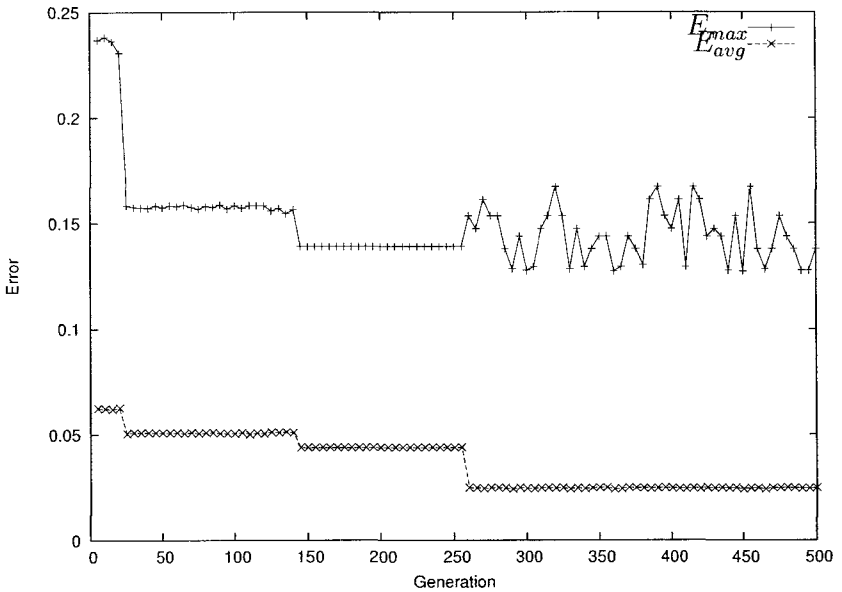

Figure 5.1: Highest affinity individual, partition algorithm, angle approximation

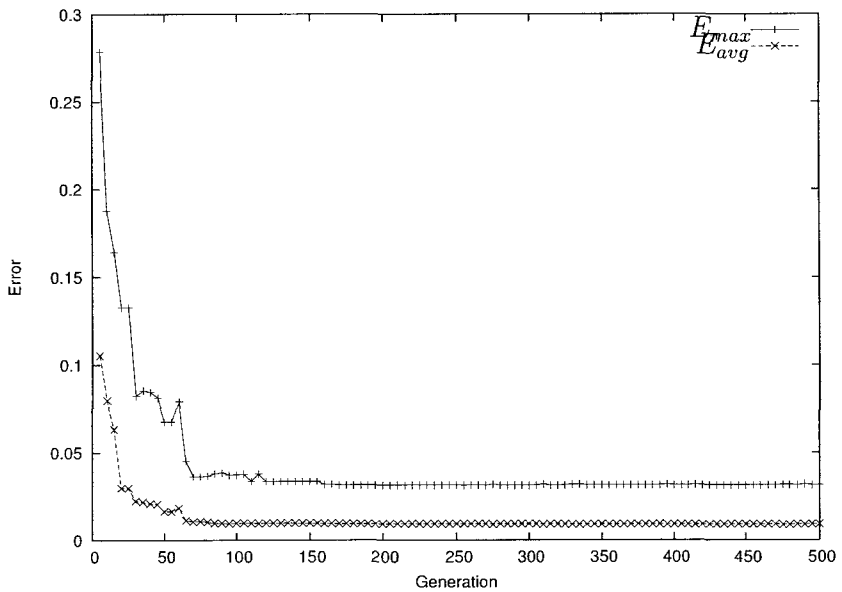

Figure 5.2: Highest affinity individual, modified partition algorithm, angle approximation

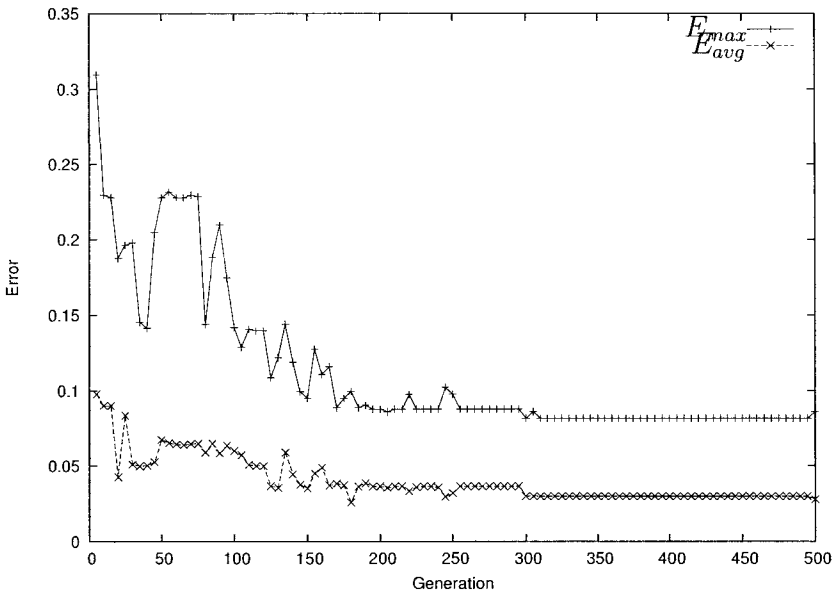

Figure 5.3: Highest affinity individual, probabilistic algorithm, angle approximation

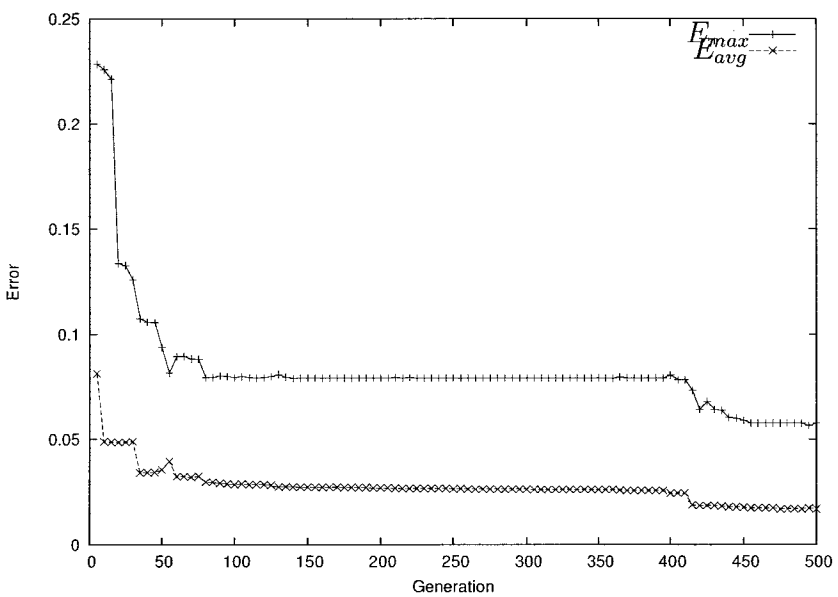

Figure 5.4: Highest affinity individual, modified probabilistic algorithm, angle approximation

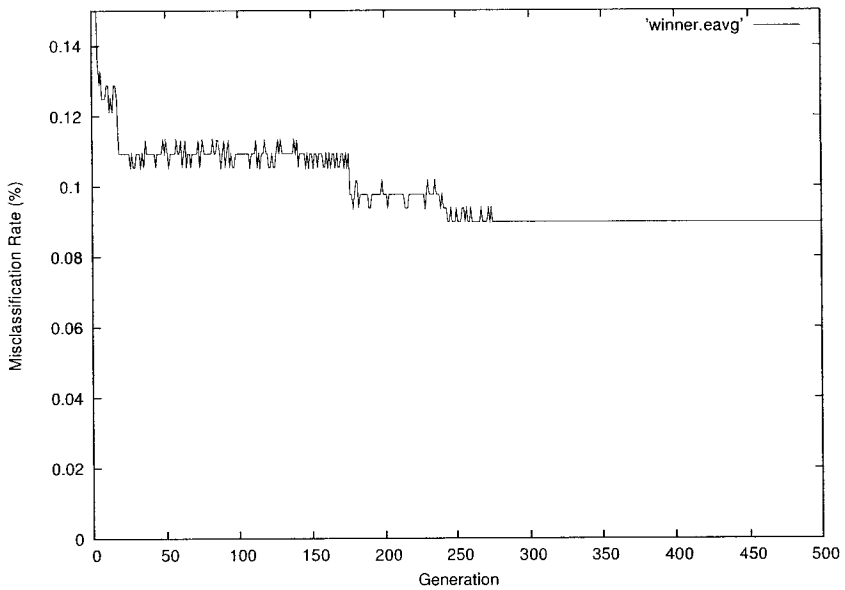

Figure 5.5: Highest affinity individual, partition algorithm, quadrant classifier

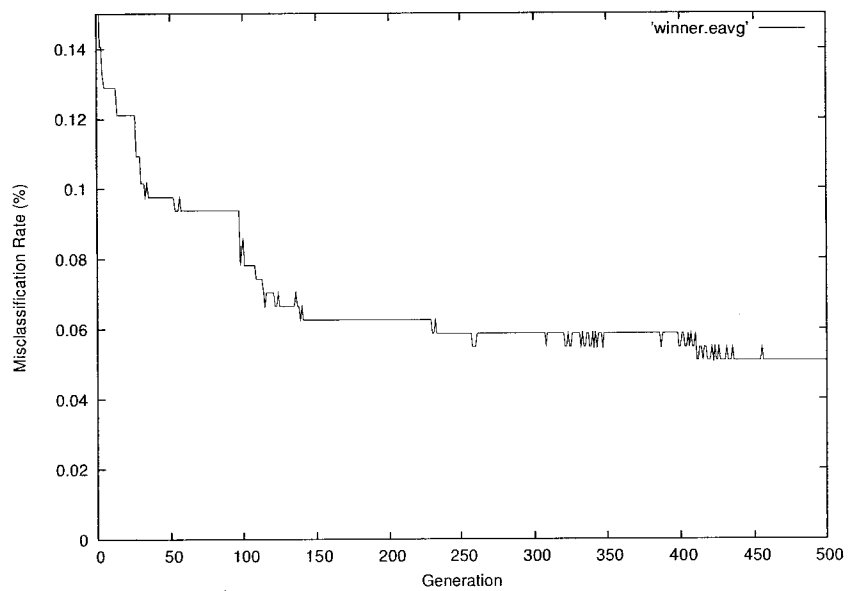

Figure 5.6: Highest affinity individual, modified partition algorithm, quadrant classifier

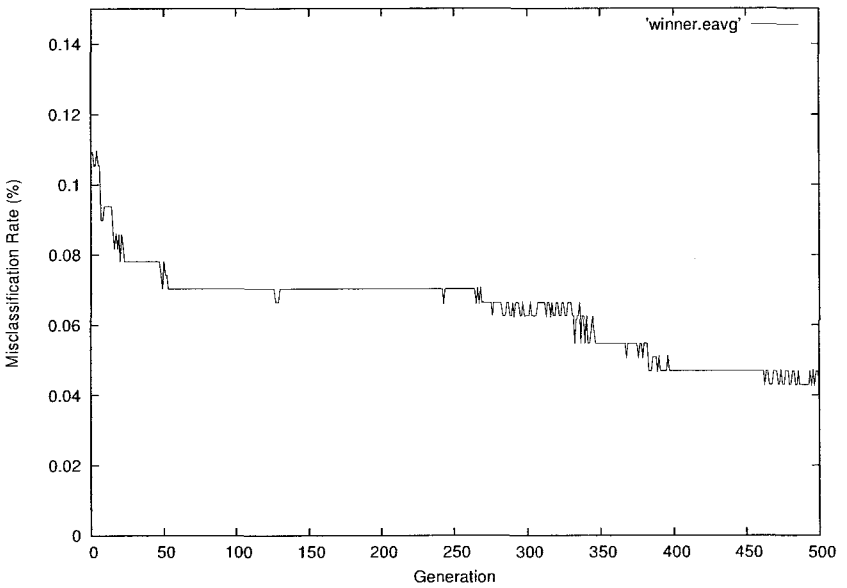

Figure 5.7: Highest affinity individual, modified probabilistic algorithm, quadrant classifier

mator performs extremely well in the noiseless case. Figure 5.9 shows magnitude of approximator's error in the noiseless case.

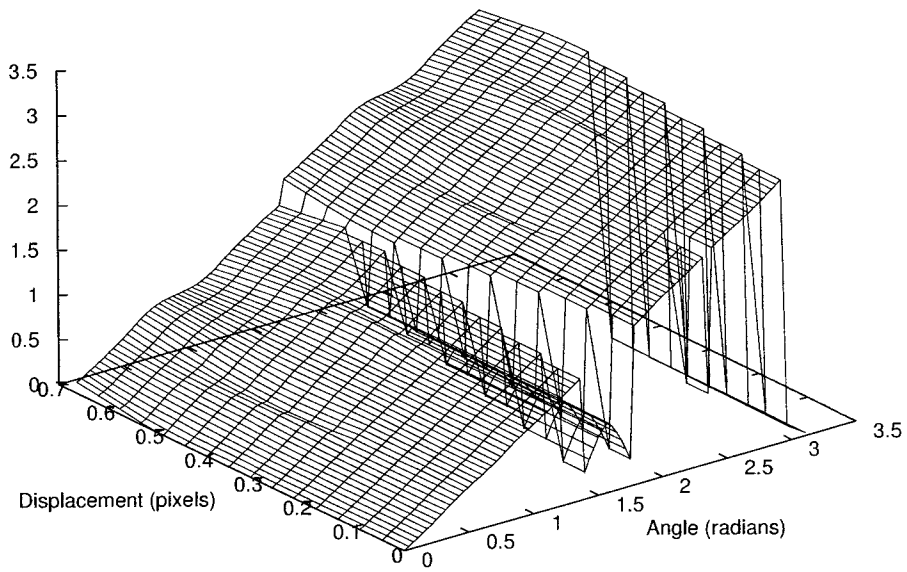

Figure 5.8: Angle approximator performance, clean images

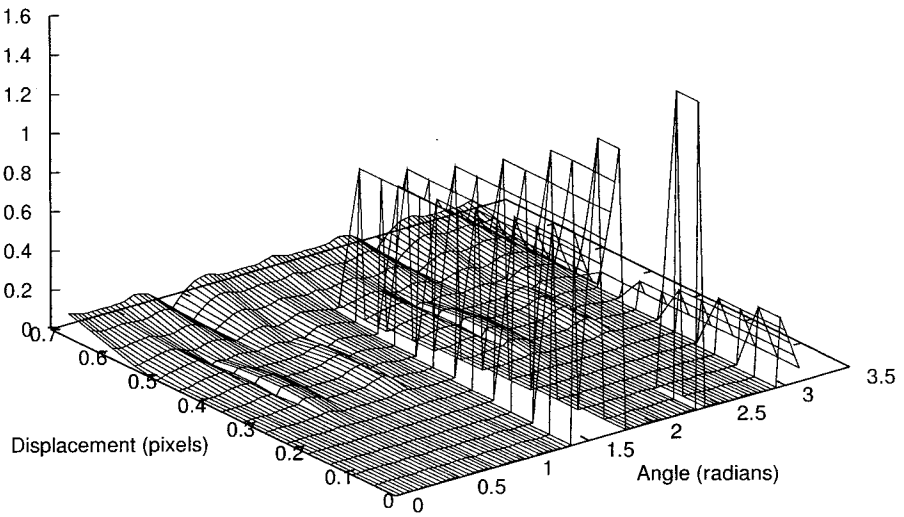

Figure 5.9: Angle approximator error, clean images

Chapter 6

Analysis

The analysis of the results comprise two distinct parts. First, and most obvious, is the performance of the Hough transform approximation, using the expressions developed by clonal selection. This is covered in the first part of the chapter.

The second part deals with analysing the performance of the clonal selection algorithms used to generate the expressions.

6.1 Algorithm Performance

Figure 6.1 shows a scatter plot of the error of the angle estimation as a function of the estimated angle, on data with added gaussian noise and $\sigma \in [0, 0.5]$. Notice the relatively low error values for all but the first quadrant: this is a result of the selection of q_{max}. Figure 6.2 shows the response of the quadrant identifier for the first quadrant as a function of σ. At higher values of sigma, the estimated angle consistently settles to 0.4. The mirrored expressions will also settle out to 0.4, so at higher noise levels q_{max} will depend more on the order of comparisons in the implementation than on

the values themselves. As it is currently implemented, the first quadrant is usually selected in high noise environments.

Conveniently, this allows almost all of the error incurred by misclassification to be ignored by treating angles in the first quadrant as completely uncertain: the entire range of $[0, \pi]$ will be plotted.

Figure 6.3 shows the error as a function of $\sqrt{\eta}$, but only when the estimated angle is greater than $\frac{\pi}{4}$. It also shows the curve $\epsilon = \sqrt{\eta} + 0.1$, which fits almost all of the points comfortably.

The final expression for the range of angles to be plotted, $\bar{\theta}$, given an estimated angle θ_e and η, is shown in equation 6.1.

$$\bar{\theta} = \begin{cases} [0, \pi] & \theta_e < \frac{\pi}{4} \\ [\theta_e - \sqrt{\eta} - 0.1, \theta_e + \sqrt{\eta} + 0.1] & otherwise \end{cases} \tag{6.1}$$

6.2 Clonal Selection Performance

For both partition methods, the affinity of cloned, mutated, and replaced individuals was recorded. For the original probabilistic method, clone and mutant affinities were recorded, along with the average affinity.

Figures 6.4 and 6.5 show a plot of the average error for an angle approximator experiment using the modified partition method, as well as the recorded affinity data. Since the affinity function is normalised, it doesn't significantly reflect the improvement in the population, except in the short downward spikes in the affinity of the cloned individuals.

Then, Figures 6.6 and 6.7 show the same results for the original partition method. The same downward spikes in the affinity of cloned individuals are ob-

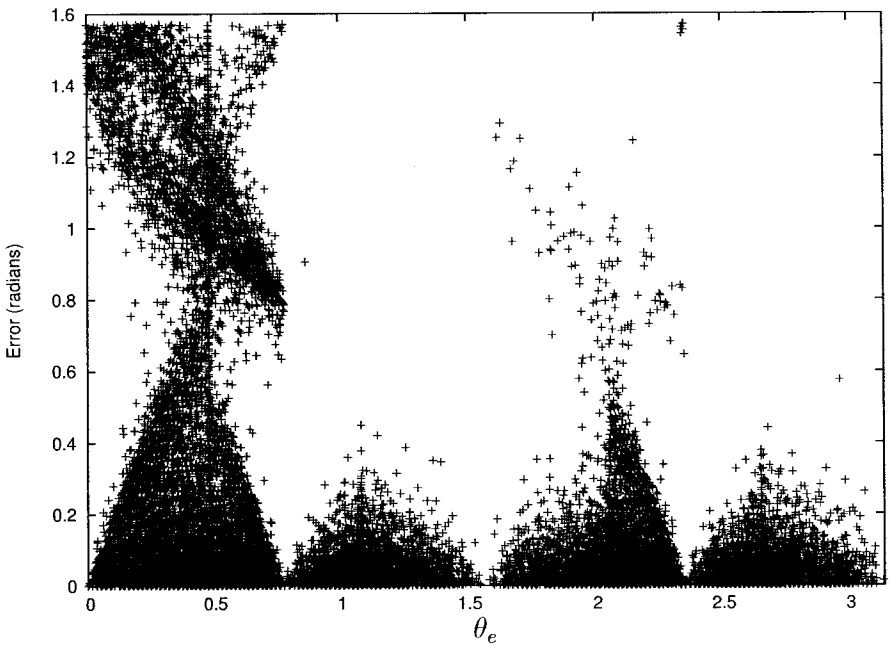

Figure 6.1: Error as a function of estimated angle

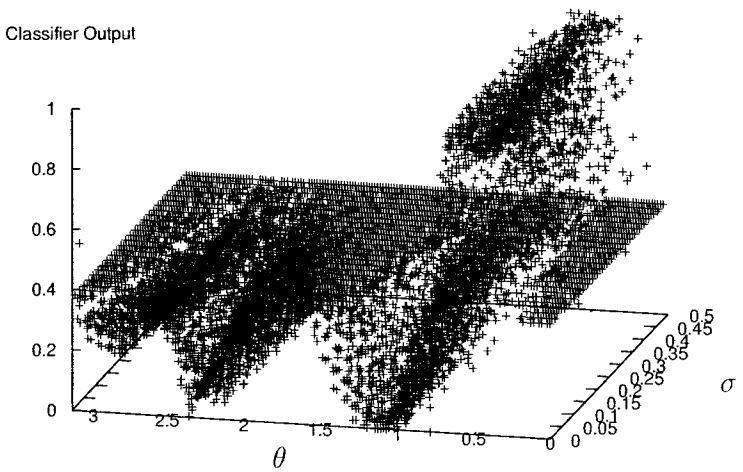

Figure 6.2: Quadrant identifier response with increasing σ

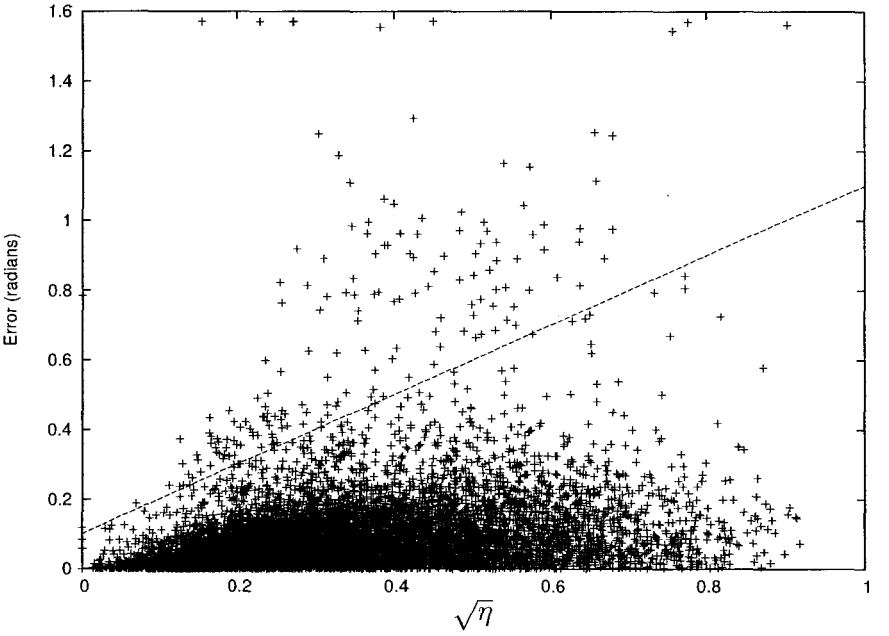

Figure 6.3: Error versus the root of η, with fitting curve

served, but the recovery time is significantly longer. This is related to the handling of the top partition, as is discussed below.

Figures 6.8 and 6.9 show the errors and the clone and mutant affinities as well as the average affinity of the population for an experiment using the probabilistic method, altered to use the two random number clone selection method described below. The average affinity is used because there is no direct correspondence between randomly generated individuals and individuals in the existing population.

No figures are included for the modified probabilistic method. In these experiments, the clone affinity resembles the clone affinity in the modified partition method, albeit with slightly more noise due to the inclusion of a small number of generally lower affinity individuals. The mutant and average affinities resemble those from the original partition method. They offer no additional insight into the convergence process.

Also, there are no figures for the quadrant identifier experiments, because they convey considerably less information than the graphs for the angle approximation. In the quadrant identifier experiments, the average affinity value tended to be very high. This occurred since any expression which evaluated to zero for any input value had a 75% rate of successful classification, but any expression which evaluated to one had a 25% classification rate. As a result of this skew in the population affinity, the experiments converge more slowly. Also, since the improvement in performance over the entire experiment seems less significant ($\tilde{8}8\%$ to $\tilde{9}6\%$) and the improvement occurs less frequently and in smaller increments, the affects of breakthroughs on the population were not clearly visible.

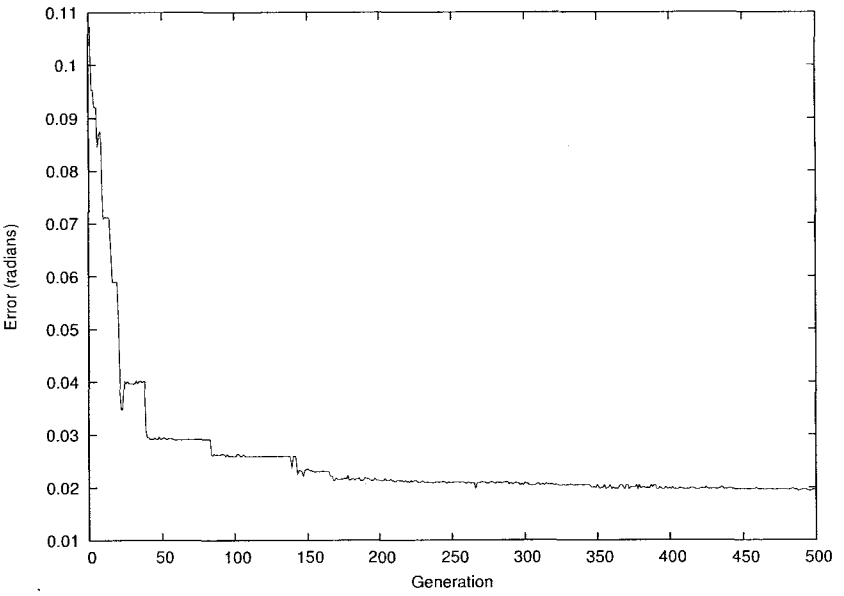

Figure 6.4: Modified partition method average error

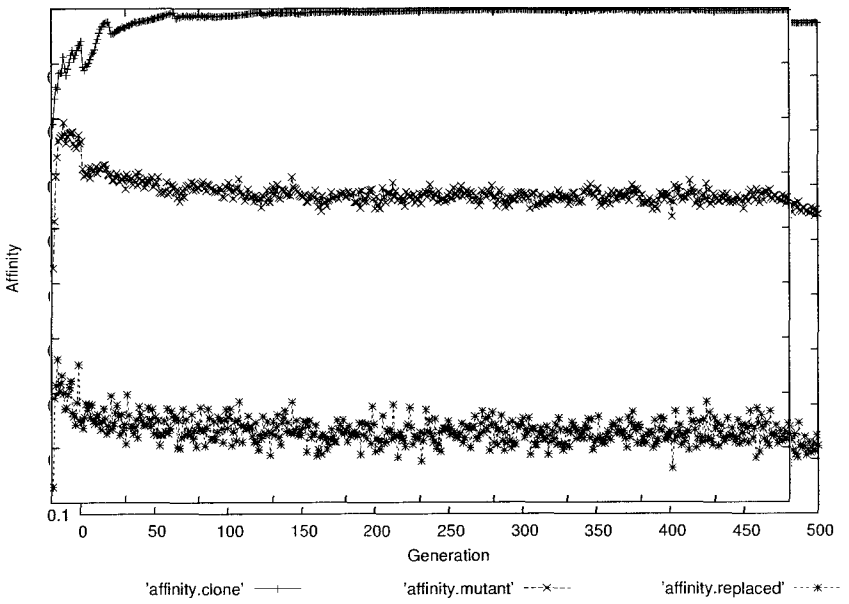

Figure 6.5: Modified partition method affinity values

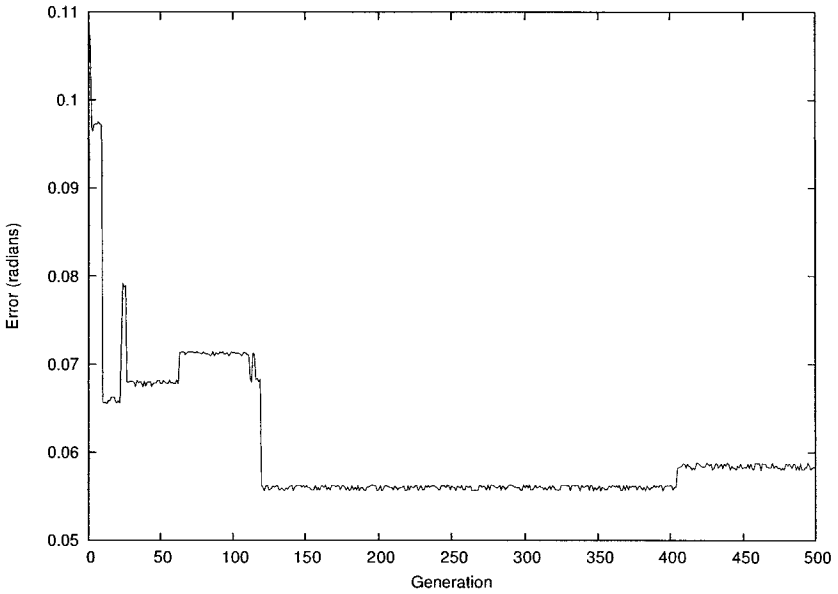

Figure 6.6: Partition method average error

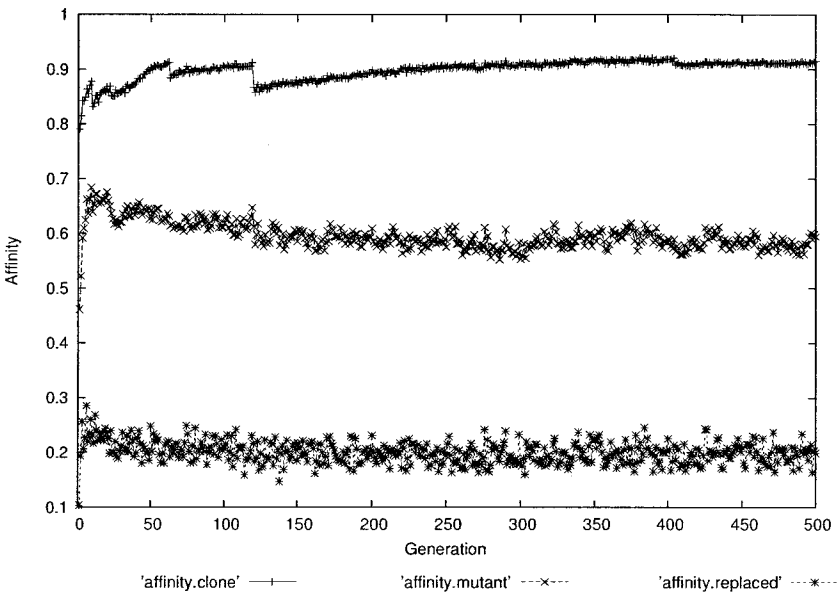

Figure 6.7: Partition method affinity values

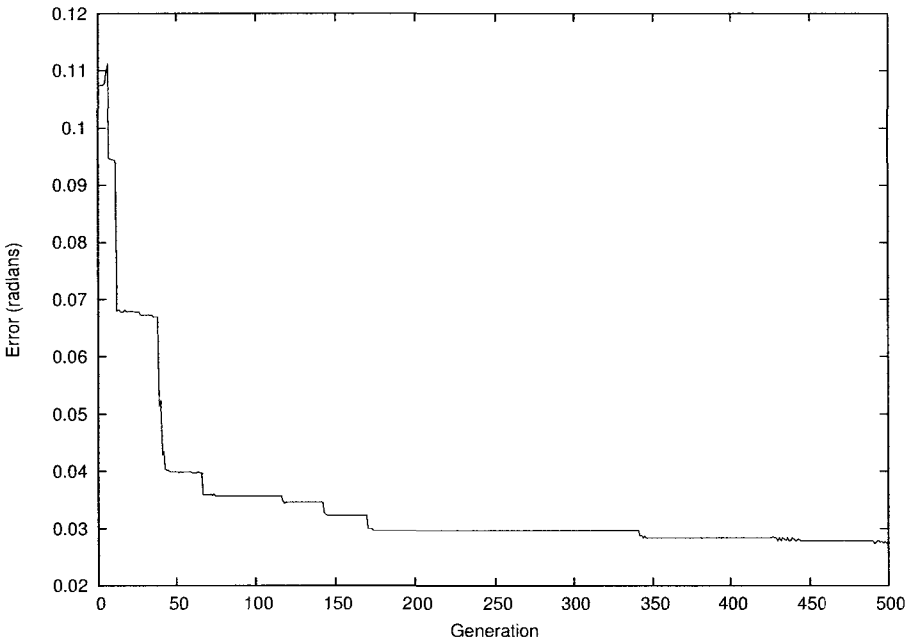

Figure 6.8: Probabilistic method average error

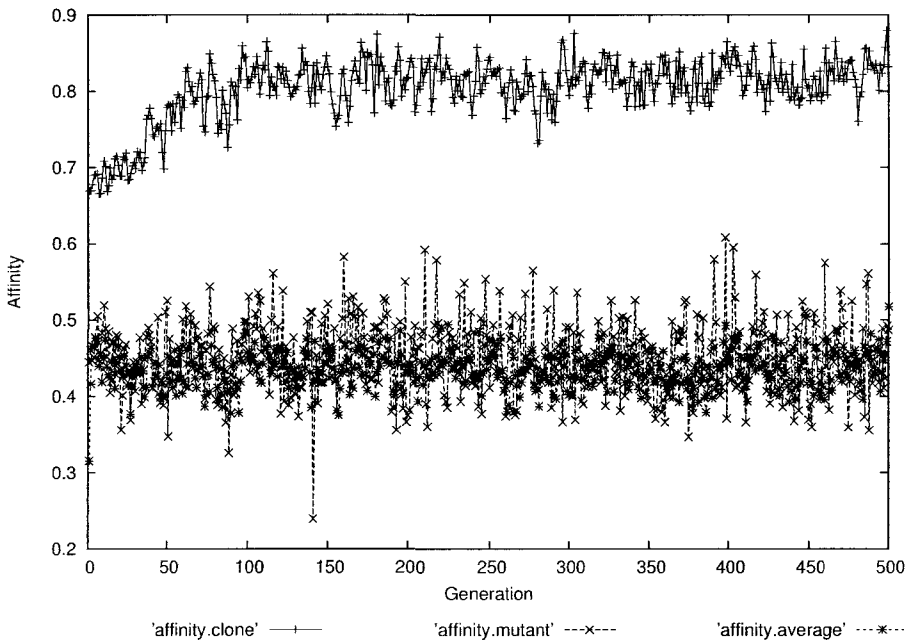

Figure 6.9: Probabilistic method affinity values

6.2.1 Probabilistic Method Analysis

The probabilistic method generally failed to converge on the quadrant classifier problem. This is because of extinction: the fitter individuals might not be included in the next repertoire, by chance.

Equation 6.2 shows the probability that the individual under consideration will be cloned into the next generation's repertoire using the original probabilistic method. In the experiments run in [15], P_r and P_c were set at approximately 0.5 and 0.1 respectively. This results in a probability of 0.05 that the fittest indivudal in the repertoire will be cloned, per iteration through the repertoire.

On the other hand, the probability that an individual will be added to the next repertoire when a given individual i is being considered is given by 6.3. With the same values for P_r and P_c, and a value of 0.30 for P_m, this probability evaluates to $0.5 + 0.185A_i$. Regardless of the distribution of affinity values, this will usually fill the next repertoire in no more than three passes through the current repertoire. This puts the average probability of the single fittest member being cloned into the next repertoire at least once at less than 0.15.

$$P_{G+1} = \bar{P}_r \cap A_i P_c \tag{6.2}$$

$$P_{new,i} = P_r \cup (A_i \cap (P_c \cup P_m)) \tag{6.3}$$

There are several ways of decreasing the likelihood of extinctions in this algorithm: one is to reduce P_r, which will increase the number of passes through the current repertoire required to create the next one, and the other is to increase P_c, which will increase the number of cloned individuals at the expense of a more stagnant population.

Some modifications to the algorithm can also be made: when deciding if an individual will be cloned, another random number can be generated and compared to the affinity. This will decrease the probability that individuals with a lower fitness will be cloned. Combined with a higher value for P_c, this creates a number of clones comparable to the original algorithm, but will have a higher average clone affinity. The downside is that the highest affinity individuals tend more towards cloning and less towards mutation, which slightly increases stagnation in the population. With this adjustment, the probabilistic algorithm performed well for the angle approximation (as the results of Figure 6.8), but was still unable to converge for the quadrant identifier.

In the previous section, it was explained that the quadrant identifier experiments had relatively high average affinities. This means that the fittest individuals, even with a normalized affinity measure, are not significantly above the average in terms of numerical performance. Not only does this reduce the advantage enoyed by fitter individuals, it also reduces the number of passes through the population. This was countered by raising the affinity value to the sixth power, which lowered the average affinity without affecting the fittest individuals. This aided convergence in the modified probabilistic algorithm, but was insufficient to help the original version.

Also, saying that the probabilistic algorithm fails to converge at all is incorrect: it fails to converge within the length of the experiment. Figure 6.10 shows the data from a quadrant identifier experiment that was run for 15,000 generations, with parameters that seemed not to converge. Although there were frequent extinctions, in the later generations the population tended towards lower misclassification rates. Since an individual is considerably more likely to be mutated than cloned, even when extinctions occur, mutants of the higher performance individuals will remain in the

population. Therefore, as long as P_m and P_c are not so small that even the mutants will die out, the repertoire will contain a growing number of individuals that are close to higher performance ones. In other words, even if the actual performance of the repertoire does not seem to improve, its potential for improvement continues to grow.

6.2.2 Partition Method Analysis

In the angle approximation problem, every other algorithm outperformed the original partition method. In the quadrant identifier problem, it outperformed the original probabilistic algorithm, but performed dismally compared to the modified versions. In both problems, the original partition method tended to stagnate considerably.

The explanation for this lies in its treatment of the top partition, which is cloned. Cloned individuals do not add to the convergence of the algorithm: they only maintain current performance. To put it in evolutionary terms, the fittest survive, but they do not breed.

This is visible in Figure 6.7, where breakthroughs cause noticeable drops in the affinity of the cloned section of the population. Unlike the other algorithms, the individual responsible for the breakthrough is never mutated, so the recovery of the average clone affinity comes solely from the discovery of fitter individuals as a result of mutations from the considerably lower affinity second partition. Also, as the overall performance of the population improves, it becomes less likely that an individual from the second partition will be mutated into one which will outperform members of the cloned partition. This is reflected in the slower recovery later in the experiment as well as the dismal overall performance of the algorithm.

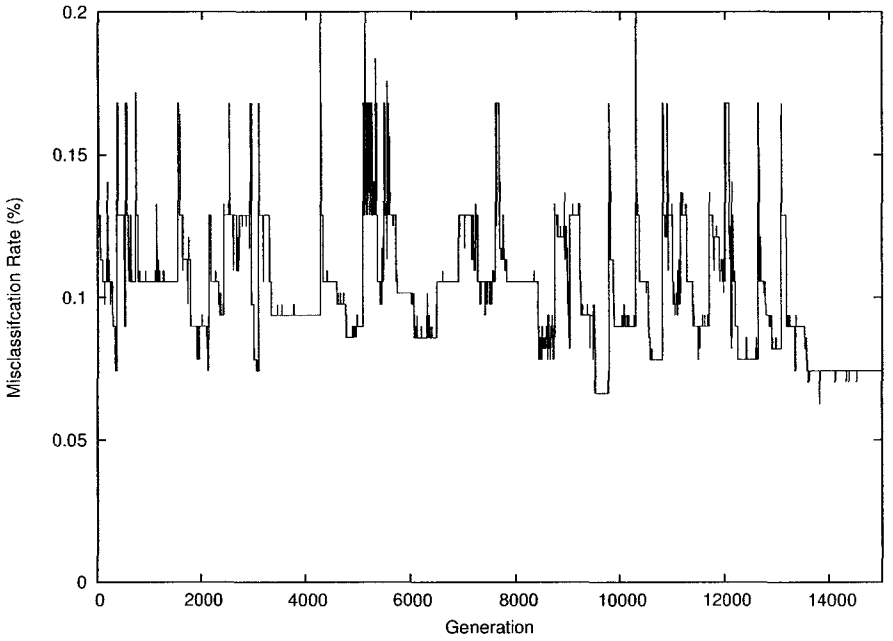

Figure 6.10: A longer experiment showing gradual improvement.

6.2.3 Relative Merits of Selection Algorithms

For speed and simplicity of implementation, the modified partition algorithm is a clear winner. It converged consistently and quickly in most experiments. However, the high average clone and mutant affinities suggest that it generates a fairly homogenous repertoire fairly quickly, so it is expected to perform poorly on problems where the correlation between affinity and correctness is low, and to deal somewhat poorly with time-varying affinity functions.

The probabilistic method and its variant both seem to maintain more diverse populations, as is indicated by the lower clone and mutant affinities, and are therefore more likely to retain their performance in problems where the modified partition algorithm might not. The modified probabilistic algorithm would generally be preferable since its performance is more consistent from generation to generation.

The original partition algorithm has no advantages that are not outdone by the other algorithms.

Chapter 7

Conclusion

Feature extraction is one of the fundamental problems of computer vision, and one major category, based on the Hough and Radon transforms, seems to provide the best performance in the presence of noise. However, they are both slow.

This class of algorithms discovers parametric features by mapping pixels in the input (presumably an edge image) to curves in another image which represents the parameter space of the feature. Since each pixel in the input image maps to a curve, performing the mapping is computationally intensive.

This thesis presents an algorithm to decrease the computational complexity of the mapping for parameterised lines by estimating a range of the parameter space to plot. In low noise images with decent contrast, the algorithm can narrow down the range of parameters to about 0.2 radians, about fifteen times smaller than the full $[0, \pi]$ range. As image noise increases, the range is increased until, in high noise images, the entire parameter space is plotted, rendering the algorithm identical to the Hough transform.

The centrepiece of the algorithm is a pair of expressions to estimate the angle of

any line which might be passing through the image. These expressions were discovered using Immune Programming, a new computational intelligence technique which is a hybrid of genetic programming and clonal selection, an artificial immune system algorithm.

Several different clonal selection methods were implemented and tried on the problems, and their performance and function were analysed, providing some amount of insight into the relative merits of the different algorithms.

Two of the presented algorithms were new variants of existing methods. The modified partition method compensates for low exploitation of high-performance individuals in the original partition method and achieves excellent convergence speeds at the cost of developing a very specialised repertoire. The other was a modified version of Musilek's probabilistic clonal selection method which guarantees that the highest affinity members of the population will survive, which compensates for a class of problems where the original version converges extremely slowly, although in most cases it does not necessarily improve convergence speed.

Immune programming and other immune-related techniques are still in their infancy, and although their initial results are impressive, there is room for improvement, not only with clonal selection alone, but also by combination of various immune techniques.

Although the feature extraction algorithm presented here is, in most cases, considerably faster than the normal Hough transform, the feature extraction problem is still unsolved. The difficulty lies in the fact that the parameter image generated by Hough-like line-finding algorithms does not determine the endpoints of features, only the direction of features. Object recognition based on these algorithms therefore usually involves segmenting the image into many images containing a single region of

data that might be a feature (for example, by region growing techniques) followed by several Hough transforms. Since segmentation algorithms tend to be extremely noise-prone, this is far from ideal. More effective holistic techniques might be possible, and could achieve considerably higher performance. Computational intelligence techniques could yield a path to this.

Bibliography

[1] C. F. Olson, "A general method for geometric feature matching and model extraction," Int. J. Comput. Vision, vol. 45, no. 1, pp. 39–54, 2001.

[2] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," Commun. ACM, vol. 24, no. 6, pp. 381–395, 1981.

[3] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," Commun. ACM, vol. 15, no. 1, pp. 11–15, 1972.

[4] S. Hao, "Numerical solution of radon's problem in a two dimensional space," J. Comput. Math., vol. 4, no. 3, pp. 249–254, 1986.

[5] L. Jin and L. Yang, "Parallel solution of hough transform and convolution problems – a novel multimodal approach," in SAC '92: Proceedings of the 1992 ACM/SIGAPP symposium on Applied computing, (New York, NY, USA), pp. 775–781, ACM Press, 1992.

[6] L. Chen, H. Chen, Y. Pan, and Y. Chen, "A fast efficient parallel hough transform algorithm on larpbs," J. Supercomput., vol. 29, no. 2, pp. 185–195, 2004.

[7] D. Krishnaswamy and P. Banerjeer, "Exploiting task and data parallelism in parallel hough and radon transforms," in ICPP '97: Proceedings of the international Conference on Parallel Processing, (Washington, DC, USA), p. 441, IEEE Computer Society, 1997.

[8] B. Ruf and M. Schmitt, "Learning temporally encoded patterns in networks of spiking neurons," Neural Process. Lett., vol. 5, no. 1, pp. 9–18, 1997.

[9] J. R. Koza, Genetic programming: on the programming of computers by means of natural selection. Cambridge, MA, USA: MIT Press, 1992.

[10] C. Ferreira, "Gene expression programming: A new adaptive algorithm for solving problems," Complex Systems, vol. 13, no. 2, pp. 87–129, 2001.

[11] D. DasGupta, Artficial Immune Systems and Their Applications. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1998.

[12] R. L. King, S. H. Russ, A. B. Lambert, and D. S. Reese, "An artificial immune system model for intelligent agents," Future Gener. Comput. Syst., vol. 17, no. 4, pp. 335–343, 2001.

[13] S. T. Wierzchon, "Multimodal optimization with artificial immune systems," in Proceedings of the International Symposium on "Intelligent Information Systems X", pp. 167–178, Physica-Verlag, 2001.

[14] L. R. de Castro and J. Timmis, Artificial Immune Systems: A New Computational Intelligence Paradigm. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2002.

[15] P. Musilek, A. Lau, M. Reformat, and L. Wyard-Scott, "Immune programming," Information Sciences, 2005.

[16] E. Borel, Probabilite et certitude. Paris: Press Université de France, 1950.

[17] L. A. Zadeh, "Fuzzy sets," Information and Control, vol. 8, no. 3, pp. 338–53, 1965.

[18] J. C. Bezdek, "A review of probabilistic, fuzzy, and neural models for pattern recognition," Journal of Intelligent and Fuzzy Systems, vol. 1, no. 1, pp. 1–25, 1993.

[19] I. D. Svalbe, "Natural representations for straight lines and the hough transform on discrete arrays," IEEE Trans. Pattern Anal. Mach. Intell., vol. 11, no. 9, pp. 941–950, 1989.

[20] E. Angel and D. Morrison, "Speeding up bresenham's algorithm," IEEE Comput. Graph. Appl., vol. 11, no. 6, pp. 16–17, 1991.

[21] J. H. Han, L. T. Kóczy, and T. Poston, "Fuzzy hough transform," Pattern Recognition Letters, vol. 15, pp. 649–658, 1994.

[22] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," IEEE Transactions on Evolutionary Computation, vol. 1, pp. 67–82, April 1997.

[23] W. Pedrycz and F. Gomide, An Introduction to Fuzzy Sets: Analysis and Design. The MIT Press, 1998.

[24] R. T. Chin and C. R. Dyer, "Model-based recognition in robot vision," ACM Comput. Surv., vol. 18, no. 1, pp. 67–108, 1986.

[25] "International ground vehicle competition." Last accessed May 27, 2005.

[26] J. Kim and P. Bentley, "Immune memory and gene library evolution in the dynamic clonal selection algorithm," Genetic Programming and Evolvable Machines, vol. 5, no. 4, pp. 361–391, 2004.

[27] H. Du, L. Jiao, and R. Liu, "Adaptive polyclonal programming algorithm with applications," in ICCIMA '03: Proceedings of the 5th International Conference on Computational Intelligence and Multimedia Applications, (Washington, DC, USA), p. 350, IEEE Computer Society, 2003.