Proceedings of *Tools for AI -90.*
Washington D.C., November 1990.

Generation of Feature Detectors for Texture Discrimination by Genetic Search.

J. W. Bala K. De Jong

Center for Artificial Intelligence
George Mason University
Fairfax, VA

Abstract

The paper presents a genetic algorithm based methodology for learning a set of feature detectors for texture discrimination. This methodology is implemented into a vision system that learns to classify noisy examples of different texture classes by evolving populations of simple and texture-specific local spatial feature detectors.

1 Introduction

Texture provides very useful information for performing automatic interpretation and recognition of images by computer. Textural features can be crucial for the segmentation of an image and can serve as the basis for classifying image parts. Usually texture models are simulated and studied under the categories of pixel-based and region-based models. The texture recognition problem combines two troublesome characteristics. First, texture classes have prototypes which correlate highly with prototypes of different texture classes. Second, the texture examples (to be classified) are randomly distorted and noisy. Texture classification has been the focus of interest for a long time. Various investigations in pattern recognition have attempted to classify texture primarily at the image level. A number of approaches to texture analysis and classification problem have been developed over the years. Two fundamentally different approaches to texture analysis are the statistical approach and the structural approach. The statistical approach generates parameters to characterize the stochastic properties of the spatial distribution of gray levels in an image. The structural approach analyzes visual scenes in terms of the organization and relationships among its substructures.

This paper describes a novel variation of the statistical approach. Our strategy for selecting useful texture features involves the use of an adaptive search technique, a genetic algorithm, to efficiently search the space of spatial feature detectors for a high performance feature set.

2 The Feature Selection Problem

Any object or pattern which can be recognized and classified possesses a number of discriminatory properties or features. The first step in any recognition process, performed either by a machine or by a human being, is to choose discriminatory features and to measure these features. Feature selection in pattern recognition concerns the extraction of characteristic features or attributes from received input data and the reduction of the dimensionality of pattern vectors. This is often referred to as the preprocessing and feature extraction problem.

It is evident that the number of features needed to successfully perform a given recognition task depends on the discriminatory qualities of the chosen features. However, the problem of feature selection is usually complicated by the fact that the most important features are either not easily measurable or, in many cases, their measurement is inhibited by economic considerations. The selection of an appropriate set of features, which support an acceptable performance in spite of difficulties encountered in the extraction and selection process, is one of the most difficult tasks in the design of pattern recognition system.

Feature selection may be accomplished by using two different approaches. The first one selects features independently of the performance of the classification scheme. Such an approach may be referred to as absolute feature selection. An alternative approach is performance-dependent feature selection, and its effectiveness is directly related to the performance of the classification system, usually in terms of the probability of correct recognition. This second approach of feature selection is addressed in this project.

The inclusion of the feature selection and extraction stage effectively partitions the pattern recognition problem into two subproblems, as illustrated in Figure 1.

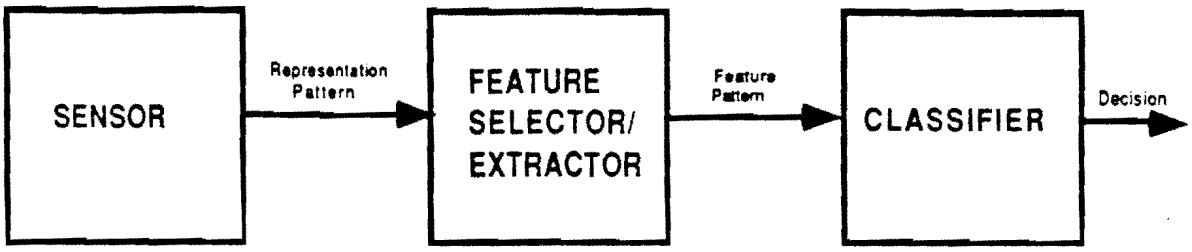

Figure 1. Partitioning of a pattern recognition problem by inclusion of a feature selection and extraction stage

3 Feature Generation by Genetic Search

A genetic algorithm [2] maintains a constant-sized population of candidate solutions, known as individuals. The initial seed population can be chosen randomly or on the basis of heuristics, if available for a given application. At each iteration, known as a generation, each individual is evaluated and recombined with others on the basis of its overall quality or fitness. The expected number of times an individual is selected for recombination is proportional to its fitness relative to the rest of population. The power of a genetic algorithm lies in its ability to exploit, in a highly efficient manner, information about a large number of individuals. By allocating more reproductive occurrences to above average individuals, the overall affect is an increase of the population's average fitness. New individuals are created using two main genetic recombination operators known as *crossover* and *mutation*. Crossover operates by selecting a random location in the genetic string of the parents (crossover point) and concatenating the initial segments of one parent with the final segment parent to create a new child. A second child is simultaneously generated using the remaining segments of the two parents. Mutation provides for occasional disturbance in the crossover operation by inverting one or more genetic elements during reproduction. This operation insures diversity in the genetic strings over long periods of time and prevents stagnation in the convergence of the optimization technique. The individuals in the population are typically represented using a binary notation to promote efficiency and application independence in the genetic operations.

We use genetic methods to generate the population of feature detectors. By using simple genetic operators (crossover, mutation, and selection), the population evolves and only the strongest elements (features) survive, thus contributing to the overall performance. Each detector contributes to textural class recognition and/or clustering. This contribution is used as an objective function to drive the machine learning procedures in its search for new, and useful feature detectors. Our method is illustrated in Figure 2.

4 Binary String Coding and Genetic Operators

Many possibilities exist for computing features to recognize textures. The best features will be those which define an attribute space which most easily lends itself to partitioning by the classification algorithm. We use neighboring gray-level values as the feature attributes. These attributes are the simplest attributes to characterize textures and the coding we use enables easy "manipulation" of this features by a genetic algorithm. We use an N*N array centered on each pixel to define the neighborhood of interest. Each of the N*N-1 elements of the array are either zero or one, indicating which of the neighboring pixels are to be used to construct a feature vector. Figure 4 is an example of a 5*5 array which extracts gray level values from 24 neighboring pixels. By selecting a variety of sample points in a texture area, a set of "typical" feature vectors can be extracted. For example, the following table illustrates feature vector values (10 gray levels resolution) extracted from seven

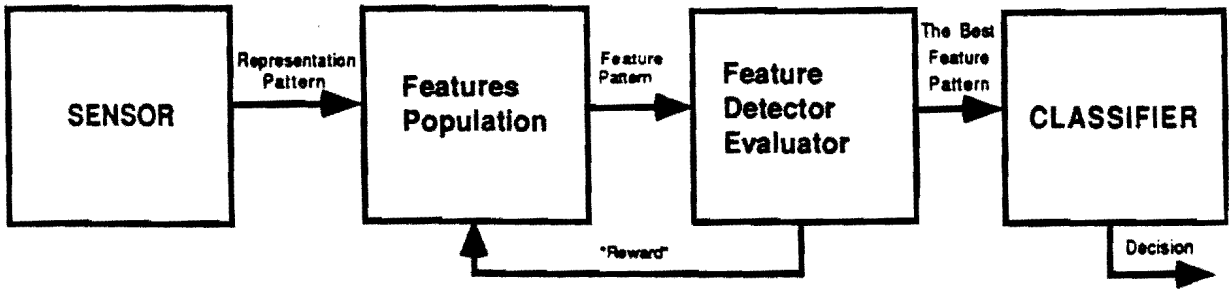

Figure 2. Genetic Algorithms approach to feature selection and extraction

different positions in a textural area using the above described 5 by 5 detector.

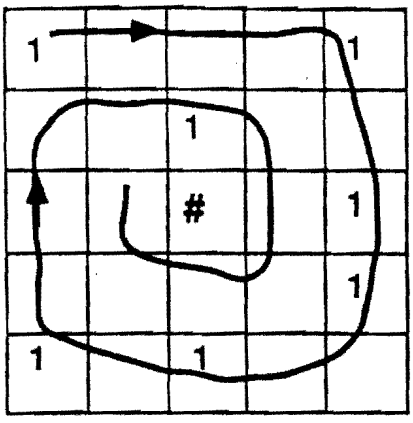

Figure 4. Feature coding for texture discrimination

Pixel positions

Sample Points

	x1	x2	x3	x4	x5	x6	x7	x8
1	1	4	8	5	8	4	4	8
2	1	8	6	7	4	6	3	8
3	4	7	4	5	6	6	3	2
4	3	6	5	5	4	3	2	8
5	6	6	5	7	4	9	6	5
6	5	4	3	6	8	4	3	5
7	9	7	5	0	5	9	4	4

Selecting useful feature extraction templates of the type shown in Figure 4 is a different task. In order to use the genetic algorithms to search this space, it must be represented as a binary string. The spiral inside the template in Figure 4 represents the way we traverse the template's position to obtain a binary string representation for use with the genetic algorithm, resulting in a binary string of

"10001011001010100100000".

Less significant bits of the extracted string (left side of the string) represent pixels located further from the centered pixel. More significant bits (right side of the string) represent close neighboring pixels to the center pixel. This coding method is very important in our algorithm. It reflects the way we describe textures, as statistical processes between neighboring pixels. The further the distances between the centered pixel and others in the window, the smaller the dependencies between pixels gray levels. By using this coding technique we are able to exchange important spatial discriminatory structures between different detectors in the population.

Figure 5 represents the idea of the crossover operation between two 5 by 5 detectors. The black dots in Figure 5 correspond to 1s in the encoded strings depicted beneath detectors. Because of the spiral coding for the elements of population the detector window is divided in to two regions according to the position of crossover point. The inner part (core) represents the coding of extraction information for the neighborhood of the central pixel. The other part is represented by the left hand side position in the encoded string. The crossover operation, when applied to two detectors produces offspring detectors. By dividing the detector's area into regions closer and further to the central pixel, the information about relevant extraction discriminatory patterns for given texture classes can be easily transferred into subsequent generations.

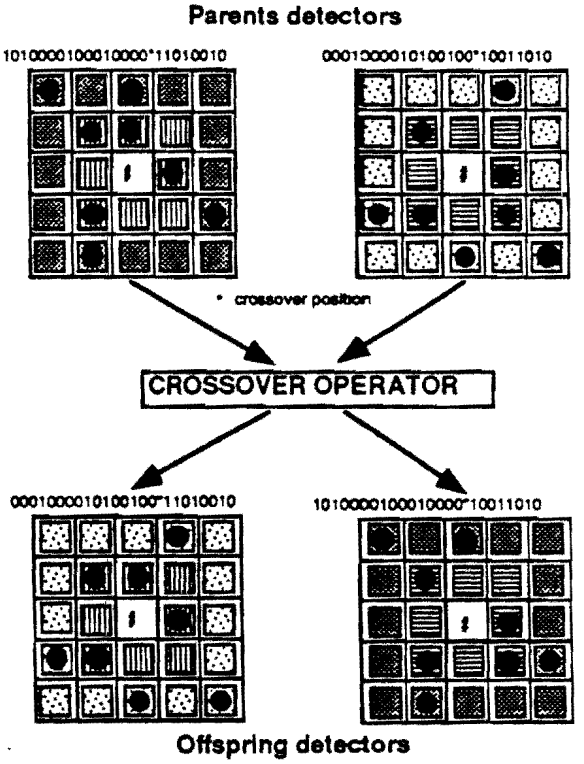

Figure 5. Crossover operation

The mutation operation in the initial phase of the system run is different than the standard one. Because our performance measure is the Euclidean distance (next section), the more 1s we have in the string, the more attributes the feature vector has, and the greater the distance between two vectors. This situation may cause instability in the system, where feature detectors with more attributes may dominate in the population. That is especially important during initial generations of the population. To aleviate this, the initial mutation operation (start-up mutation) is performed by shifting 1s in the feature string randomly to the left or to the right. This type of operation preserves the extracted vector length (the total number of positions with 1s in the string is the same). After a few initial generations, the

system is already tuned up to its input data (textures) so that the standard mutation can be introduced.

5 Feature Detector Performance Evaluation

The evaluation of the feature detector is based on its ability to "cluster" members of training examples. For each example of a given training set, the Euclidean distance to examples of other classes is computed. Pattern classification techniques based on distance functions are one of the earliest concepts in automatic pattern recognition.

This method can yields practical and satisfactory results when the pattern classes tend to have clustering properties. We have decided to use it for two reasons. First, it is less computationally expensive than other methods which is important for a genetic search algorithm where performance measure for each detector in the population is computed in each cycle of the population. Second, even if we are unable to correctly classify examples from the training sets for a given feature extraction method, we can still compare the performance of the detectors and assign an evaluation value.

Our measure of performance is the total similarity between classes which is represented as the sum of the similarity measures between each two classes. The similarity measure between two classes is the sum of the similarity measures between all possible different pairs of examples (one from the first class and the other from the second class). If the distance is below a predefined threshold value, these two examples are considered to be "close" and statistics for each of the two classes are updated.

The measure we use for two examples is the square of Euclidean distance between these two examples. This measure for two vectors (examples) **x** and **y** of dimensionality D is denoted as:

$$\| x - y \| = \sum_{k=1}^{D} (x_k - y_k)^2$$

where x_k and y_k are k-th components of D-dimensional vectors **x** and **y**.

Let us suppose that we have N classes, M^i examples in each class (i=1,..,N). Let S be the number of detectors in the population (population size), and T be the threshold distance value. The performance measure p(v) of the feature detector v (v=1,..,S) is represented by the following formula:

$$p(v)=\sum_{a=1}^{M^i-1} \sum_{b=a+1}^{M^i} \sum_{i=1}^{N} \sum_{k=i+1}^{M^k} d(e_i^a, e_k^b)$$

e_i^a (e_k^b) is the a(b)-th example from the i(k)-th class

$$d(e_i^a, e_k^b) = \begin{cases} 1 & \text{if } \| e_i^a - e_k^b \| < T \\ 0 & \text{if } \| e_i^a - e_k^b \| \geq T \end{cases}$$

The performance measure of the whole population P is denoted as:

$$P=\sum_{v=1}^{S} p(v)/S ,$$

which is an average performance value of the population of feature detectors.

6 Experimental Framework

The Figure 6 depicts the 9 by 9 detector that has been used in experiments. The total length of a string is 81 bit positions.

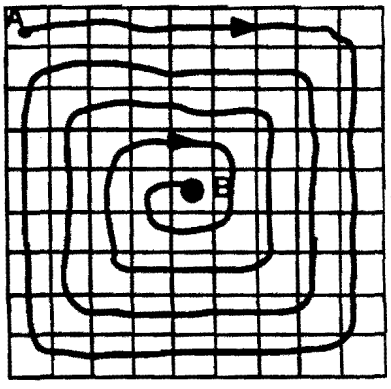

Figure 6. A large 9 by 9 window

By using the larger window, we are able to tune up our feature extraction process to different resolutions of textures (windows 5 by 5 and 3 by 3 are subset of the 9 by 9 window). The point A represents the left most position of the encoded string, and the point B represents the right most position. The black dot is the central pixel, for which the feature detector is applied. The 9 by 9 window is scanned throughout all positions of a given textural area, and in each position an example of the class description is extracted with the number of attributes equal to the number of 1s in the encoded string. Examples extracted from different pixel positions of the textural area represent its class description. The method was tested on 15 textural images from Brodatz

[1] album of textures. The extraction area (learning area) for each texture was 50 by 50 pixels. 500 locations were chosen randomly inside these areas to extract 500 feature vectors for each texture. The results of testing this methods on four specific textures are presented in the next section. These textures are depicted below in Figure 7.

7 Experimental Results

7.1 Evolving Feature Sets

Figure 8 shows the changes of similarity measure over 18 generations of the feature detectors population.This experiment was performed on the four textural images depicted in Figure 7. There were 35 individuals in the population. The total similarity measure (the sum of similarity measures of each class) drops from 1000 to 200, thus yielding better clustering of these two classes. A similarity measure of 200 in the 18th generation means that there are 200 pairs of extracted feature vectors from different classes (500 vectors in each class) at the distance below the threshold value. Similar results have been obtained for all different textural images.

Figure 9 shows the best detectors in the 1st, 6th and the 18th generation of population. These detectors show spatial arrangement of pixels positions (filled squares) that are important and relevant for discriminatory characteristics of these four textures. Although we use these detectors directly to extract attributes as the pixel gray levels values, it should be noticed that this higher level information obtained from these detectors (pixels structures) can be useful to "custom-design" standard extraction operators for specific needs (depending on input images). For example, we may choose a pecific mask operator from the collection of Law's mask operators. This aspect is the subject of further research.

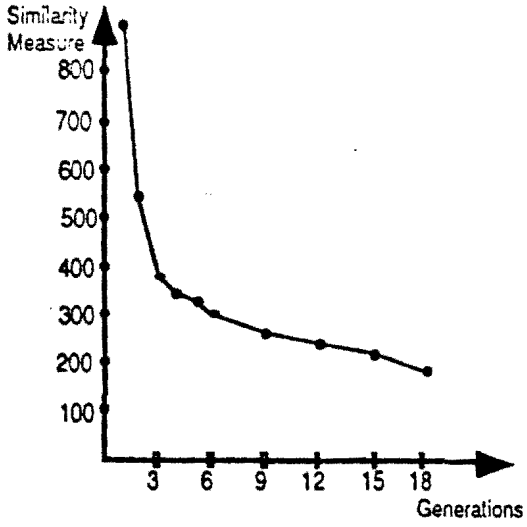

Figure 8. Total similarity measure for four textures

7.2 Testing Clustering Degree

We use the AQ learning module to generate symbolic rule-based descriptions of the texture classes from the feature sets selected by the genetic algorithm.The AQ15 program is based on the AQ algorithm [7], which generates decision rules from a set of examples. When building a decision rule, AQ performs a heuristic search through a space of logical expressions to determine those that account for all positive examples and no negative ones.

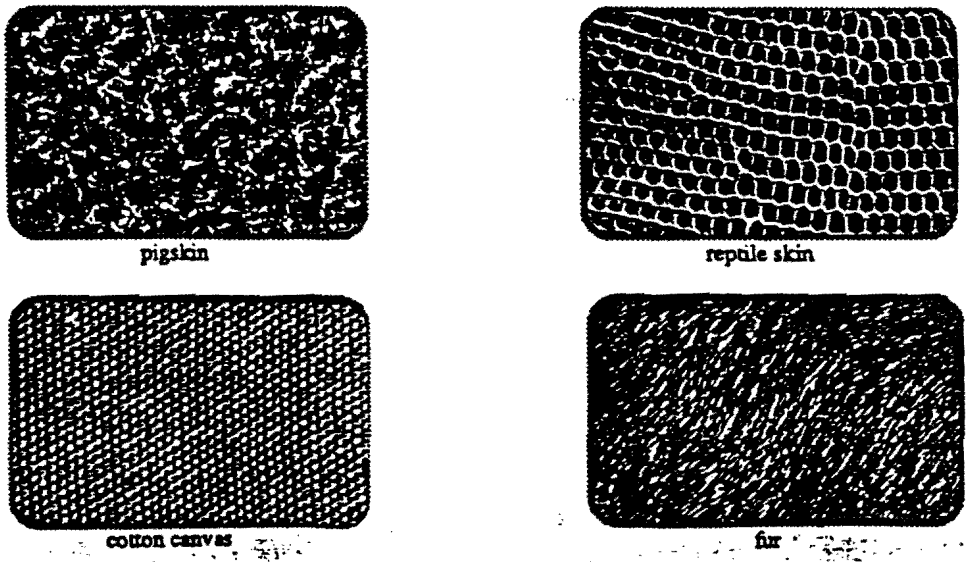

Figure 7. Textural images used in experiments

Learning examples are given in the form of events, which are vectors of attribute values. Events from a The given class are considered its positive examples, and all other events are considered its negative examples. For each class a decision rule is produced that covers all positive examples and no negative ones.

A decision rule is a disjunction of complexes describing all positive examples and none of the negative examples of the concept. A complex is a conjunction of selectors. A cover is formed for each decision class separately. It defines the condition part of a corresponding decision rule. The following is an example of decision rule (with 4 complexes):

1. **[x1=5..9] [x3=6..9] [x9=0..15]**
   (total:10, unique 6)

2. **[x4=6..9] [x7=6..9] [x9=0..16]**
   (total:8, unique 5)

3. **[x2=4..9] [x4=4..9] [x6=0..5] [x9=0..15]**
   (total:8, unique 5)

4. **[x3=5..9] [x5=0..6] [x6=0..6] [x9=0..16]**
   (total:2, unique 1)

The following example with nine attributes

<1 6 0 4 6 4 5 6 9>

is covered by the third complex of the above decision rule. We can notice that x2=6 is satisfied by the first selector of this complex, x4=4 by the second selector, x6=4 by the third, and x9=9 by the last selector. Since there are no selectors for other attribute values (x1, x3, x5, x7, x8), they are satisfied by this complex.

For each complex the total number of examples covered by this complex and the number of unique examples covered only by this complex are presented. Feature sets which produce better clusters will result in simpler symbolic descriptions of the feature classes in the sense that fewer complexes are in the cover (more examples are covered by the first complexes of the class description). The following is the decision class with three complexes:

1. **[x2=2..4] [x3=10..13] [x5=7..9]**
   (total:17, unique 10)

2. **[x1=5..9] [x2=6] [x7=0..11]**
   (total:12, unique 7)

3. **[x2=4..9] [x4=4..9] [x6=0..5]**
   (total:5, unique 4)

It has less complexes than the previous description and each complex covers more examples. The third complex of this decision covers the same example (*<1 6 0 4 6 4 5 6 9>*).

During the second, the sixth, and the final generations of population, we select the best feature detector in the population to extract a set of feature vectors as to the AQ module for classification. We observe a significant improvement in the class description generated by AQ from one generation to the next. The number of complexes in the class cover drops by 12%. Also the number of total and unique vectors covered by the first complex of a given class cover is about 8-10% higher than that for the first cover generated during the initial generation of the population. Different experiments performed on numerous images clearly show that successive generations of the population produce a better degree of clustering for a given class description.

8 Conclusion

Preliminary work with an application of genetic algorithms to a texture feature selection has been described. The results obtained confirm the utility of the genetic algorithm in solving problems within the specified domain. The next step would be to test usefulness of this genetic approach to more complex texture discrimination and general computer vision operators. More complex operators should be tested.

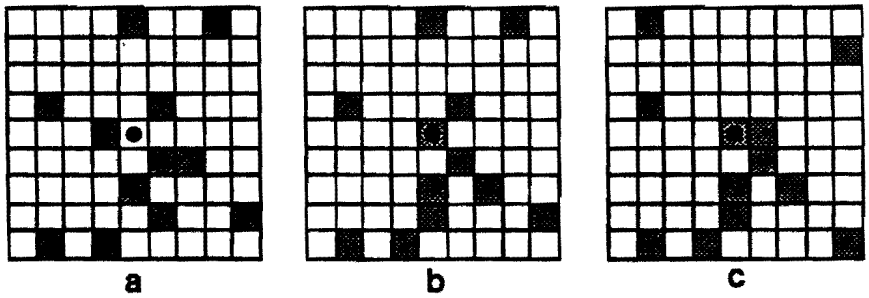

Figure 9. An example of the best detectors in (a) the 1st, (b) 6th, and (c) 18th generation of population

For example, we may introduce weights to our previous operator. Three bits are reserved for each pixel position to code eight levels of weighting information. The total bit length for the string is 9x9x3=243. The string coding for such an operator is depicted in Figure 10.

"010•000•101•000•000•000•000•000•001...."

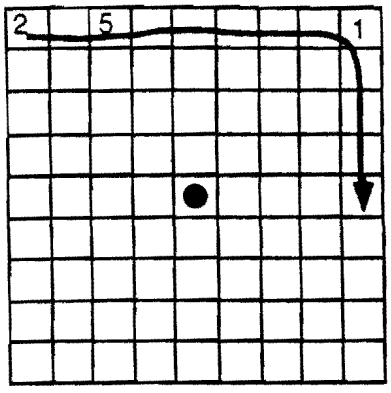

Figure 10. Coding with weights

There is another aspect of the genetic approach which can be exploited. Namely, this approach enables the exploration of different feature vectors. The final class representation can be generated by using a few of the best operators of the population. These operators can be used to generate several descriptions of the same class. This "multi-view" class representation encodes a more complete description of the class, that is of significant importance in the recognition phase of our system yielding better results than a single view representation of the class. This characteristic of the genetic approach to feature extraction is the subject of further research.

ACKNOWLEDGMENTS

The authors wish to thank Dr Ryszard Michalski, Dr Gheorghe Tecuci, and Dr Harry Wechsler for valuable comments and discussion, and Janet Holmes for editing suggestions.

This research was done in the Artificial Intelligence Center of George Mason University. Research activities of the Center are sponsored in part by the Defense Advanced Research Projects Agency under grant, administrated by the Office of Naval Research, No. N00014-87-K-0874, in part by the Office of Naval Research under grant No. N00014-K0226, and in part by the Office of Naval Research under grant No. N00014-88-K0397.

REFERENCES

[1] Brodatz, P., : *A Photographic Album for Arts and Design*, Toronto, Dover Publishing Co., 1966.

[2] De Jong, K., : *Learning with Genetic Algorithms: An Overview*, Machine Learning 3; 123-138, Kluwer Academic Publishers, 1988.

[3] Fitzpatrick, J.M. and J.J. Grefenstette, : *Genetic Algorithms in Noisy Environments*, Machine Learning 3 pp. 101-12-(1988).

[5] Fitzpatrick, J.M., J.J. Grefenstette, and D. Van Gucht, : *Image Registration by Genetic Search, Proceedings* of IEEE Southeastern Conference, pp. 460-464 (1984).

[6] Gillies, A., : *Machine Learning Procedures for Generating Image Domain Feature Detectors*, Ph.D. dissertation, Computer and Communication Sciences in The University of Michigan, 1985.

[7] Michalski, R.S., Mozetic I., Hong J.R., Lavrac N., :*The AQ Inductive Learning System*, Report No. UIUCDCS-R-86-1260, Department of Computer Science, University of Illinois at Urbana-Champagne, July 1986.

[8] Stadnyk, I., : *Schemata Recombination in Pattern Recognition Problems. Genetic algorithm and their applications*: Proceedings of the Second International Conference on Genetic Algorithms, pp. 27-35.

[9] Tou, J., Gonzalez, R. : *Pattern Recognition Principle*, Addinson-Wesley Publishing Company, 1974.

90-35

Discovery of Maximal Distance Codes Using Genetic Algorithms

Kejitan Dontas and Kenneth De Jong
Department of Computer Science
George Mason University
Fairfax, VA 22030

Abstract

This paper describes an application of genetic algorithms to the problem of discovering communication codes with properties useful for error corrections. Search spaces for these codes are very large so as to rule out any exhaustive search strategy. Coding theory provides a rich and interesting domain for genetic algorithms. There are some coding problems about which a lot is known and good codes can be generated systematically. On the other hand, there are problem areas where little can be said about the characteristics of the codes in advance. Genetic algorithms have been advocated for these kinds of problems where domain knowledge is either limited or hard to represent and formalize. This paper describes some initial experiments performed in using genetic algorithms to discover maximal distance codes, and discusses the potential advantage of genetic algorithms in this problem domain.

1. Introduction

In artificial intelligence and other areas there are several problems for which computationally tractable solutions either have not been found or have shown to be nonexistent [6]. Even the polynomial time algorithms may take a large amount of time to be of practical use for realistically sized problems. Solutions obtained by such methods are often static and not capable of adaptation to environmental changes. Genetic algorithms, neural networks, and simulated annealing have been proposed or revived in recent years in an attempt to obtain approximate, or quasi-optimal solutions to combinatorial optimization problems [4, 5]. Genetic algorithms (GAs) provide a powerful domain-independent search mechanism, useful in domains which do not yield to systematic analysis. These advantages make GAs a good candidate for learning and discovery in relatively unexplored areas. Due to their adaptive nature, GAs are also valuable in exploring potential improvement to candidate solutions obtained by rules of thumb which may have no solid theoretical basis.

The introduction of genetic algorithms as a general purpose problem solving mechanism was inspired by the evolutionary phenomena in biological systems, where survival of organisms is at stake in an environment with finite resources. Organisms must evolve to compete and cooperate with others in order to ensure perpetuation of their kind. It would be difficult to incorporate all the complexities of natural evolving systems into genetic algorithms; however, simplified simulations have been shown to be useful in solving a variety of practical problems. Genetic algorithms typically start as a randomly generated collection of individuals representing an initial sample of the space to be searched. Each individual is evaluated as to its "fitness" with respect to the goals of the application domain. This cumulative information is exploited to bias subsequent search into promising sub-spaces [2, 7, 11] by selecting parents from the current population on the basis of fitness to produce "offspring" which correspond to new samples of the space. The survivability of offspring depends on their success in inheriting good traits from their parents and discarding those that are detrimental to the overall performance. As the number of simulated generations increases, the offspring produced become increasingly "fit" with respect to the specified goal, ultimately resulting in (near) optimal solutions.

2. Maximal Distance Coding Problem

Coding theory is relatively new and was inspired by Shannon's seminal paper on information theory in 1948 [14]. In general we wish to transmit a message through a noisy channel as quickly and reliably as possible (Figure 1). Speed in transmitting a message is achieved by eliminating redundancy in the message. One way to do this is to assign shorter length vectors to those parts of the message which occur frequently (e.g. Huffman codes, [12]). In order to achieve noise immunity, one can add redundancy to the message in a manner that allows detection of errors. Addition of a parity bit to a block of message allows detection of a single (or odd number of) error/s in the block, which can be transmitted again for error recovery. It is possible to locate the bit positions in error and correct them by designing appropriate encoding and decoding schemes. A composite encoder first compresses the code and then encodes it using error detection/ error correction codes. The receiver performs these operations in reverse order. It first decodes the received message through a matched error correcting decoder and then decompresses the resultant code to obtain the transmitted message.

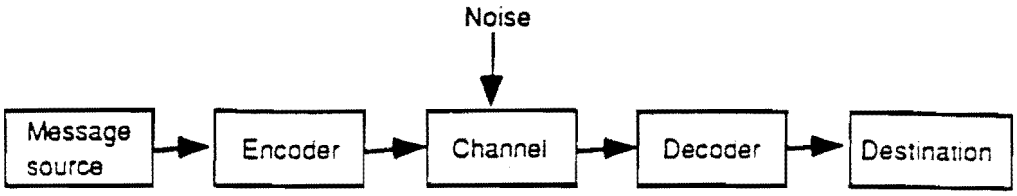

Figure 1: A typical message transmission path from source to destination

A variety of codes such as linear block codes, cyclic codes, burst error correcting codes, and convolutional codes are used for error correction [13]. In this paper we restrict our attention to block error correction codes which are characterized by the following parameters:

1) The *Arity* of the code (q). This represents the number of distinct symbols in the alphabet used in the code. Binary codes, one of the most common, have arity 2 and use the alphabet {0, 1}.

2) The *Length* of a code (l). This is the number of letters (from the alphabet) needed to make a word (vector) in the code, where a code denotes a set of words (vectors) which are used to transmit a message.

3) The *Size* of a code (M). The size of a code is the number of distinct words (vectors) in the code used to compose the messages to be sent.

4) *Distance* (d) of a code. The distance of a code is the minimum Hamming distance between pairs of words (vectors) belonging to the code. The Hamming distance between two vectors x and y, denoted by d(x, y) is the number of positions in which the vectors x and y differ. In other words the Hamming distance between x and y denotes the minimum number of substitution operations that must be performed to convert the vector x to the vector y. A code C can correct up to t errors if $d(C) > (2t+1)$, where, $d(C) = \min\{d(x, y) \mid x, y \in C, x \neq y\}$.

A *repetition* code of length n is a simple example of a block error correction code. It is defined for any arity; for a binary repetition code, each bit is repeated n times. An n bit code tolerates $\text{floor}(\frac{n-1}{2})$ bits of errors in each block of n bits. The information content in each block of n bits is 1 bit.

An (n,M,d) code is a code of length n, containing M code words and having minimum distance d. A good (n,M,d) code has a small n (reflecting smaller redundancy and faster transmission), a large M (denoting a larger vocabulary) and large d (indicating greater tolerance to noise and error). Figure 2 lists theoretical maximum values of M (in some cases, estimation of lower and upper bounds of M) for some combinations of n and d (Source: [10, 15]).

n	d = 3	d = 5	d = 7
5	4	2	-
6	8	2	-
7	16	2	2
8	20	4	2
9	40	6	2
10	72-79	12	2
11	144-158	24	4
12	256	32	4
13	512	64	8
14	1024	128	16
15	2048	256	32
16	2560-3276	256-340	36-37

Figure 2 : Theoretical limit on the size for codes of various lengths and distances

Although such bounds provide some insight into the best we can hope to achieve, there is frequently no guidance available as to how to construct an actual code with (near) maximal distance. As the code size (n) increases, the search space of possible codes grows exponentially, ruling out any possibility of systematic or random search. For (7,16,3) codes, the search space is at least 10^{20}.

This paper describes experiments and results in using genetic algorithms as an efficient adaptive search strategy for "discovering" maximal distance codes. Figure 3 shows an example of a (7,16,3) maximal distance code solution. Solutions are in general not unique, but are sparse in the space to be searched.

3. Representation Issues

In order to apply genetic algorithms as an adaptive search strategy, the problem of interest must be mapped into a representation suitable for genetic search. The simplest approach is to represent points in the space to be searched as fixed-length binary strings to which the standard genetic operators of crossover and mutation can be applied when producing new offspring [2]. For the coding domain, concatenation of the M code words, each of length n into one long binary string, represents a suitable mapping. Using this approach, a (7, 16, 3) code would need a total of 112 bits to represent 16 words (vectors) each of which

0000000
0001011
0010110
0011101
0100111
0101100
0110001
0111010
1000101
1001110
1010011
1011000
1100010
1101001
1110100
1111111

Figure 3: A sample maximal distance code of length 7.

is 7 bits in length. Since we wish to maximize the minimum mutual distance among the words, it is clear that the performance of a code is dependent only on the collective composition of the words and is independent of the order in which the words in the code are represented. Sorting the code in the decreasing order of the words reduces all equivalent codes to a single representation. In the case of a (7,16,3) code, this heuristic reduces *both* the search space and the number of possible solutions by a factor of 16!. Initial experiments showed that it is more profitable to discover a good code in the reduced space. One possible reasons for this behavior is that it dramatically reduces competition between potential solutions having high degree of bitwise dissimilarity, but identical evaluation functions. Consequently we adopted the sorted representation for the experiments reported here.

4. Evaluation Function

A good evaluation function is vital to the successful application of GAs to a problem. Evaluation functions provide goal-directed feedback about the fitness of individuals in the population. The adaptive behavior of the GAs depends on this feedback to drive the population towards better overall performance. In this respect the objective function does not always constitute a good evaluation function.

Let $\mathfrak{C}(L, n)$ denote the set of all possible codes of length L and size n. Let d(x, y) denote Hamming distance between strings x and y of length L. Let $d(C) = \min\{d(x, y) \mid x, y \in C, x \neq y\}$. Our goal is to discover a code $C \in \mathfrak{C}(L, n)$, such that $d(C) = \max\{d(C) \mid C \in \mathfrak{C}(L,n)\}$ the minimum distance between any pair of words is highest possible. Our objective function d(C), which we wish to maximize maps all $\binom{2^L}{n}$ codes onto L+1 possible distances (0..L). Clearly it lacks ability to discriminate between promising and non-promising codes. It is possible to remedy the situation by choosing an evaluation function f(C) such that it has many values in the co-domain, and if d(C) denotes max(d(C)) then f(C) denotes max(f(C)) over the space of all possible codes, $\mathfrak{C}(L, n)$. The objective function d(C) is very sensitive to small changes in the codes. An evaluation function f(C) may be chosen to provide a smoother gradient over bit changes in the codes. We used the following evaluation function initially: $f_1(C) = \dfrac{1}{\sum_{i=1}^{L} \min d_{ij}\ j=1..16; j\neq i}$ where d_{ij} represents Hamming distance between words i and j in the code C.

However, the results of our initial experiments were discouraging and were partially attributed to an inadequate choice of the evaluation function. The function $f_1(C)$ is better than d(C), but still is not very helpful for GAs in terms of providing a good feedback. Consider two hypothetical codes C_1 and C_2, such that C_1 has the minimum mutual distance of 2.0, and the average mutual distance of 2.5, and C_2 has the minimum mutual distance of 2.0, and the average mutual distance of 3.0. The code C_2 seems superior in terms of its proximity to optimal solution and therefore deserves a better fitness value. The evaluation function $f_1(C)$ is unable to discriminate between the two codes C_1 and C_2 and it is not effective in driving the population towards more desired code configurations. Note that a function which returns average distance between the pairs of vectors in the code is not a good fitness function. It is easy to construct codes that maximize this criteria, but have a minimum distance of 0. For example, a code consisting of 8 copies of all 1 vectors and 8 copies of all 0 vectors, has the average distance of 3.5, which is the maximum for a 7 bit code of 16 size, but has a minimum mutual distance of 0.

One way to think of the problem of finding a (7, 16, d_{max}) code is to visualize placing 16 individuals on the corners of seven dimensional space so as to maximize the mutual distance between them. The situation is analogous to particles of equal charge trying to position themselves in the minimum energy configuration in a bounded space. The evaluation function

$$f_2(C) = \sum_{i=1}^{L} \sum_{j=1; j\neq i}^{L} \frac{1}{d_{ij}^2}$$

captures this idea and satisfies all the desirable requirements such as being independent of the length or size of the code, and providing a smoother gradient with respect to the evaluation function. It resulted in significantly better GA performance and was used for all the experiments reported in this paper.

5. Experimental Results

In all of the experiments reported here we used GENESIS (1.0 used), a version of GENESIS shell originally developed by John Grefenstette [8] and later enhanced at

While they represent a significant improvement over random search, none of the experiments converged to an optimal code in 1,000,000 cycles. The on-line and off-line performances were relatively stagnant for at least 600,000 cycles in each case, which indicated that convergence in the near future was unlikely.

5.2 Changing the Alphabet Size

There were a number of possible directions one could take at this point. We were curious as to whether this might be an application in which treating individual code words as indivisible units (with respect to genetic operators) might improve things as is occasionally suggested (Antonisse, 1989) in spite of the theoretical results to the contrary (Holland, 1975). We defined a modified crossover operator in which crossover points could occur only at code word boundaries while the crossover rate remained at 0.6. A code word was selected for mutation with probability of 0.01. Once a word is selected, exactly one of the bits in the word was mutated at random.

With these changes, a set of experiments was run using the identical seeds and other parameters as the previous set of experiments. The modified GA discovered an optimal code in about 350,000 cycles on the average (four experiments with different seeds). Figure 6 shows codes discovered with each seed, each of the words in the code has a Hamming distance of at least 3 from its nearest neighbors. This results are surprising and contrary to the established belief that GA representations using smaller sized alphabets are preferred over larger ones. Further analysis is required to understand these results more fully.

Even though the evaluation function is multi-modal, in a given experiment the algorithm drives the population to one of the peaks, rather than dividing the population of different peaks. This suggests another direction for future experiments: to study the effects of larger populations and the use of techniques to permit the evolution of "sub-species" around competing optimal peaks.

5.3 Increasing the Number of Crossover Points.

We were also curious if in this domain increasing the number of crossover points would have a significant effect on the ability to find optimal codes. The crossover operator was modified to perform 4 (rather than 2) point crossover. A set of four experiments was run in which the random number seeds, as well as all the parameters were kept identical to the previous experiments. Figure 7 summarizes the results. No significant changes were observed. Three of the populations converged to a desirable solution within 1,000,000 cycles and the other did not. Time did not permit a more extensive analysis of the effects of increasing the number of crossover points.

5.4 Scaling Up to Larger Coding Problems

Another important issue is how well the performance of GAs scales up to larger problems. We were able to compare results on an (8,16,4) problem which required a 128 bit representation (an increase of a factor of $2^{16} = 65000$ in the size of the search space).

Experiment	1	2	3	4
Cycle	430,000	270,000	390,000	310,000
Evaluation function f(C)	19.6525	19.6525	19.6525	19.6525
Code words and Hamming distance from their nearest neighbors	1111011 3	1111101 3	1111110 3	1111010 3
	1110101 3	1110011 3	1111001 3	1110001 3
	1101110 3	1101000 3	1100100 3	1101101 3
	1100000 3	1100110 3	1100011 3	1100110 3
	1011000 3	1011010 3	1010101 3	1011100 3
	1010110 3	1010100 3	1010010 3	1010111 3
	1001101 3	1001111 3	1001111 3	1001011 3
	1000011 3	1000001 3	1001000 3	1000000 3
	0111100 3	0111110 3	0110111 3	0111111 3
	0110010 3	0110000 3	0110000 3	0110100 3
	0101001 3	0101011 3	0101101 3	0101000 3
	0100111 3	0100101 3	0101010 3	0100011 3
	0011111 3	0011001 3	0011100 3	0011001 3
	0010001 3	0010111 3	0011011 3	0010010 3
	0001010 3	0001100 3	0000110 3	0001110 3
	0000100 3	0000010 3	0000001 3	0000101 3

Figure 6: Four optimal codes discovered in four experiments.

Experiment	1		2		3		4	
Cycle	240.000		210.000		290.000		600.000	
Evaluation function f2(C)	19.6525		20.450894		19.6525		19.6525	
Code words and Hamming distance from their nearest neighbors	1111100	3	1111111	2	1111000	3	1111000	3
	1110010	3	1110001	3	1110110	3	1110111	3
	1101001	3	1101000	3	1101111	3	1101011	3
	1100111	3	1100110	2	1100001	3	1100100	3
	1011011	3	1011100	3	1011101	3	1011101	3
	1010101	3	1011011	2	1010011	3	1010010	3
	1001110	3	1000101	3	1001010	3	1001110	3
	1000000	3	1000010	2	1000100	3	1000001	3
	0111111	3	0111010	3	0111011	3	0111110	3
	0110001	3	0110100	2	0110101	3	0110001	3
	0101010	3	0101101	2	0101100	3	0101101	3
	0100100	3	0100011	3	0100010	3	0100010	3
	0011000	3	0010111	3	0011110	3	0011011	3
	0010110	3	0010000	2	0010000	3	0010100	3
	0001101	3	0001110	3	0001001	3	0001000	3
	0000011	3	0001001	2	0000111	3	0000111	3

Figure 7: Examples of best codes found using four point crossover.

In this case we knew that distance 4 optimal codes existed from a theorem in coding theory which states that if M is the largest value for a (n, M, d) code where d is odd, then there exists a (n+1, M, d+1) code, such that M is also the largest value for a (n+1, M, d+1) code. Since we already have optimal (7, 16, 3) codes in hand, this asserts the existence of a (8, 16, 4) optimal code.

It is easy to construct such a code starting from a (7, 16, 3). An additional bit is appended at the end of every 7 bit vector from (7, 16, 3) code such that the odd (or even) parity is preserved. This construction suggests that the number of solutions for (8, 16, 4) code are about 16 times the number of solutions for (7, 16, 3) code. (For each solution in (7, 16, 3) space we can generate two solutions by choosing either even or odd parity for the eighth bit. And additional factor of eight is explained by the freedom to insert the eighth bit on either side of one of the 7 bits.) The important observation here is that since, as noted above, the size of the search space has increased by a factor of 2^{16}, the density of optimal solutions has decreased.

Figure 8 presents the results of two experiments on the (8,16,4) problem. Nouce that no combinatorial increase in convergence time is observed as the search space size increases combinatorially. This is a surprising result which needs to be further tested on larger coding problems.

6. Conclusions and Future Research

The initial experiments presented here provide encouraging results regarding the application of GAs to the discovery of communication codes. Clearly additional work needs to be done to clarify several important issues: the apparent advantage of a larger alphabet size in the GA representation; the potential advantage of a speciation mechanism; and effects on performance when scaling up to larger problems with no known solutions.

Experiment	1		2	
Cycle	67.000		60.000	
Evaluation function	14.179638		14.179638	
Code words and Hamming distance from their nearest neighbors	11111010	4	11111111	4
	11100100	4	11101000	4
	11010001	4	11010010	4
	11001111	4	11000101	4
	10110111	4	10110001	4
	10101001	4	10100110	4
	10011100	4	10011100	4
	10000010	4	10001011	4
	01111101	4	01110100	4
	01100011	4	01100011	4
	01010110	4	01011001	4
	01001000	4	01001110	4
	00110000	4	00111010	4
	00101110	4	00101101	4
	00011011	4	00010111	4
	00000101	4	00000000	4

Figure 8: Solutions for the (8, 16, 4) code.

This approach can also be extended to investigate the usefulness of GAs for solving problems involving variable length source codes and cryptographic codes.

cyclic codes, and non-block codes with estimated search spaces 2^{1000} large.

Another future direction would be to apply GAs at a higher level of concept and/or theory formation. In the process of evaluating a large number of candidate codes, humans generally infer some general concepts, properties, or knowledge about the domain which can in turn be utilized to solve other similar problems more efficiently. For example, in coding theory there are numerous theorems which state (without having to perform uncertain search) whether a solution is possible to some problem. It would be highly desirable to be able to use GAs to discover, represent and then use this operational knowledge in order to build practical problem-solving procedures by these kinds of bootstrapping procedures.

Acknowledgement

This research was done in the Artificial Intelligence Research Center of George Mason University. Research activities of the Center are supported in part by the Office of Naval Research under grant No. N00014-88-K-0397, in part by the Office of Naval Research under grant No. N00014-88-K-0226, and in part by the Defense Advanced Research Projects Agency under grant, administered by the Office of Naval Research, No. N00014-87-K-0874.

References

[1] Antonisse, J. (1989), A New Interpretation of Schema Notation that Overturns the Binary Encoding Constraint, In *Proceedings of the Third International Conference on Genetic Algorithms* , pp. 86-91.

[2] De Jong, K. A.(1988), Learning with Genetic Algorithms: An Overview, *Machine Learning* Vol 3, pp. 121-138.

[2a] De Jong, K. A.(1975), An Analysis of the Behavior of a Class of Genetic Adaptive Systems. (Doctoral Dissertation, University of Michigan). Dissertation Abstracts International 36(10), 5140B. (University Microfilms No. 76-9381).

[3] De Jong, K. A.(1990): Genetic Algorithms Based Learning, Book Chapter in *Machine Learning An Artificial Intelligence Approach, Volume 3*, eds. Y. Kodratoff, R. S. Michalski.

[4] De Jong, K. A., and Spears, W. M. (1989), Using Genetic Algorithms to Solve NP-Complete Problems, In *Proceedings of the Third International Conference on Genetic Algorithms* , pp. 124-132

[5] Gamal, A. A., Hemachandra, L. A., Shperling, I., Wei, V.K. (1987), Using Simulated Annealing to Design Good Codes, *IEEE Transactions on Information Theory*, IT-33, No 1, January 1987

[6] Garey, M. R. and Johnson, D. S.(1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, San Fransisco, CA.

[7] Goldberg, D. E. (1989), *Genetic Algorithms in Search Optimization and Machine learning*, Addison Wesley Publishing Company, Inc.

[8] Grefenstette, J. (1984). GENESIS: A System for Using Genetic Search Procedures, In *Proceedings of the 1984 Conference of Intelligent Systems and Machines*, pp. 161-165.

[9] Grefenstette, J. (1986). Optimization of Control Parameters for Genetic Algorithms, *IEEE Transactions on Systems, Man, and Cybernetics*, 16, pp. 122-128.

[10] Hill, R. (1986), *A First Course in Coding Theory*, Oxford University Press, New York.

[11] Holland, J. H.(1975), *Adaptation in Natural and Artificial Systems*, The University of Michigan Press.

[12] Huffman, D. A. (1952), A Method for the Construction of Minimum Redundancy Codes, In *Proc. IRE*, vol 40, pp 1098-1101.

[13] Lin, S. and Costello, D. J., (1983), *Error Control Coding: Fundamentals and Applications*, Prenuce Hall NJ.

[14] Shannon, C. E. (1948), A Mathematical Theory of Communication, *Bell System Tech. J.*, vol 27, (pt I), pp 379-423 (pt II), pp 623-656.

[15] Sloane, N. J. A (1981), Recent Bounds for Codes, Sphere Packing and Related Problems Obtained by Linear Programming and other Methods, *Contemporary Mathematics* 9, pp. 153-185.

[16] Viterbi, A. J. and Omura, J. K. (1979), *Principles of Digital Communication and Coding*, McGraw Hill