Genetic Algorithms for Soft Decision Decoding of Linear Block Codes

Harpal Maini
Syracuse University, hsmaini@top.cis.syr.edu

Kishan Mehrotra
Syracuse University, mehrotra@syr.edu

Chilukuri K. Mohan
Syracuse University, ckmohan@syr.edu

Sanjay Ranka
Syracuse University, ranka@top.cis.syr.edu

Genetic Algorithms for Soft Decision Decoding of Linear Block Codes

Harpal Maini
Kishan Mehrotra, Chilukuri Mohan, Sanjay Ranka

School of Computer and Information Science
4-116 Center for Science and Technology
Syracuse University
Syracuse, NY 13244-4100

email: hsmaini/kishan/mohan/ranka@top.cis.syr.edu
tel: (315) 443-2368

July 8, 1994

Abstract

Soft-decision decoding is an NP-hard problem of great interest to developers of communication systems. We show that this problem is equivalent to the problem of optimizing Walsh polynomials. We present genetic algorithms for soft-decision decoding of binary linear block codes and compare the performance with various other decoding algorithms including the currently developed A* algorithm. Simulation results show that our algorithms achieve bit-error-probabilities as low as 0.00183 for a $[104, 52]$ code with a low signal-to-noise ratio of 2.5 dB, exploring only $22,400$ codewords, whereas the search space contains 4.5×10^{15} codewords. We define a new crossover operator that exploits domain-specific information and compare it with uniform and two point crossover.

Keywords: genetic algorithms, soft-decision decoding, uniform crossover.

1 Introduction

Codes are used for the reliable transmission of data over communication channels susceptible to noise. Codes may be classified as either block codes or tree codes. An encoder for a block code accepts as input a k symbol message sequence (usually binary sequence) and maps it to an n $(> k)$ symbol sequence. Each n-symbol sequence is completely determined by a specific k-symbol message. Block codes may further be classified as linear or nonlinear. A linear code, is a vector space over a finite field. We restrict our attention to codes over the two-element field, Z_2. Figure 1 describes a typical communication system. As a result of noise, the received vector components are real numbers. Of the n codeword coordinates, exactly k are linearly independent. Let $\boldsymbol{i}$ be the information vector and $\mathbf{G} = (g_{jm})$ the generator matrix, a listing of the basis vectors of a code C; then the encoding operation yields $\boldsymbol{i}\mathbf{G} = \boldsymbol{c}$, and, consequently, $c_j = \sum_{m=1}^{k} i_m g_{jm}$ represents the j-th component of the codeword, $\boldsymbol{c}$. Let $\boldsymbol{r}$ be a received vector. "Hard decision" decoding involves quantizing each component of the received vector independently to the nearest value $\in \{0, 1\}$ and then moving to the code-vector nearest to the resulting sequence. "Soft-decision" decoding algorithms utilize received vector components, not just their quantized estimates (Taipale & Pursley, 1991). A maximum-likelihood decoder finds

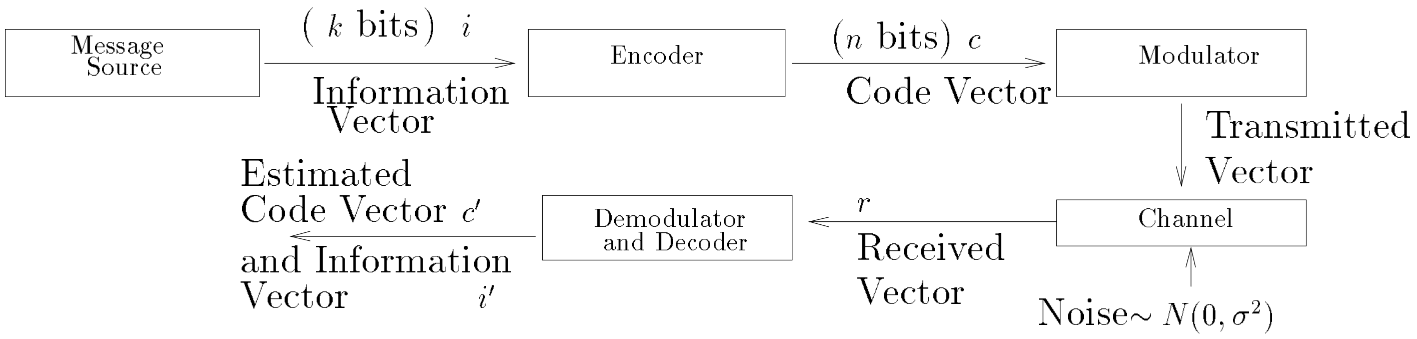

Figure 1: A Typical Communication System

a codeword $\boldsymbol{c}'$ that maximizes the conditional probability of receiving $\boldsymbol{r}$, i.e.

$$P(\boldsymbol{c}'|\boldsymbol{r}) = \max_{\boldsymbol{c}} P(\boldsymbol{c}|\boldsymbol{r}) = \max_{\boldsymbol{c}\in C} P(\boldsymbol{r}|\boldsymbol{c})P(\boldsymbol{c})/P(\boldsymbol{r})$$

The above equation holds since we assume that all codewords are equally likely to be transmitted. A maximum-likelihood decoder is optimal in this sense. If transmitted signals are binary antipodal over a discrete memoryless channel susceptible to additive white Gaussian noise, and the noise affects each symbol independently, then P$(\boldsymbol{r}|\boldsymbol{c}')$ is maximized when the squared Euclidean distance between vector $\boldsymbol{r}$ and $\boldsymbol{c}'$, $\sum_{j=1}^{n}(r_j - c'_j)^2$, is minimized (Clark & Cain, 1988; Farrell, Rudolph & Hartmann, 1983). Thus maximum-likelihood decoding reduces to nearest-neighbor decoding, with the Euclidean metric. More formally the soft-decision decoding problem

reduces to:
Given received real vector $\boldsymbol{r} = (r_1, ..., r_n)$, find a codeword $\boldsymbol{c} \in C$ that minimizes $\sum_{j=1}^{n}(r_j - c_j)^2$.

Most research in decoding algorithms has been focused on hard-decision decoding algorithms based on algebraic techniques. Soft-decision decoding has not been as extensively studied and until recently there were not many efficient decoding algorithms for linear block codes of large block length. An efficient algorithm is the recently developed A^*-based decoding algorithm (Han, 1993). Algorithm **GADEC** is a significant contribution to soft-decision decoding as shown by comparing it with an A* based approach that is currently the most successful algorithm for soft decision decoding.

The problem of decoding an error correcting code is known to be NP-hard. It is indeed desirable and often preferable to obtain suboptimal solutions to such a problem. In this paper we present a suboptimal decoding algorithm for linear block codes that is based on finding a near-global minimum for the function $\sum_{j=1}^{n}(r_j - c_j)^2$. In section 2, we describe the motivation for considering a genetic algorithm-based decoding scheme. In section 3, **GADEC**, our genetic algorithm for decoding, is described. In section 4, we present and discuss simulation results. In section 5, we analyze the algorithm. In Section 6, we give a comparison with other decoding algorithms.

1.1 An Example

Since code vectors are transmitted over the analog communication channel, the actual transmitted vector is typically a vector of the form $(a_i, -a_i, -a_i, a_i, -a_i, \ldots)$, where a_i is the amplitude of the transmitted signal, which is proportional to the square root of the signal energy. For our purposes, we assume $a_i = 1$, and that the modulator maps each 0 bit (component) of the code vector to +1, and each 1 bit of the code vector to -1. This is a one-to-one, onto mapping, and the transmitted vector is hence a vector consisting of positive and negative pulses.

Some channels are noisier than others, and the amount of corruption of the transmitted vector by the channel depends on the signal-to-noise ratio. Unbiased, normal error, noise models are often assumed (and observed). Components of the actual n-bit received vector are real numbers, whose values may differ from {1, -1}. The task of the decoder is to extract the transmitted vector from the received vector.

Let us consider $k = 2$, $n = 4$, and an encoder represented by the matrix

$$\mathbf{G} = \left(\begin{array}{cc|cc} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{array}\right)$$

The set of all possible messages (information vectors) is $\{00, 01, 10, 11\}$, and the corresponding set of code vectors is $\{0000, 0101, 1010, 1111\}$. The modulator transforms these into $\{(1,1,1,1), (-1,1,-1,1), (1,-1,1,-1), (-1,-1,-1,-1)\}$. A transmited vector such as $(-1,1,-1,1)$

may be corrupted by the communication channel into the received vector $(-0.7, 1.3, 0.2, 0.3)$. The decoder then needs to recover the transmitted vector from the received vector.

One obvious method is to map every negative received vector component to -1, and every non-negative component to 1, examining only the sign of the received vector components, and ignoring their magnitudes. This is called "quantization". In the above example, the result of this mapping is $(-1, 1, 1, 1)$ which is mapped by the demodulator to the bit-string 1000. Unfortunately, 1000 is not a code vector, and is equally distant from the code vectors 0000 and 1010.

Quantization results in the loss of information which can be gleaned from the magnitudes of received vector components. For instance, we can argue that a received vector component of -0.7 is closer than -0.1 to -1, hence we should have greater confidence in decoding the former to -1. The magnitude of a component is thus considered a measure of the reliability of its quantization to its sign. Soft-decision decoding attempts to take these magnitudes into account.

If the communication channel was malicious enough to invert the sign of every component (e.g., corrupting $(-1, 1, -1, 1)$ to $(0, 5, -0.5, 0, 5, -0.5)$, then there is no hope of recovering the original transmitted vector. In the absence of knowing what the transmitted vector was, the best that can be achieved by the decoder is to find a vector in $\{1, -1\}^n$ that corresponds to a code vector (in $\{0, 1\}^n$) that is at the least possible Euclidean distance from the received vector. Minimizing distance to the received vector maximizes the likelihood of recovering the transmitted vector. Soft-decision decoding is thus an optimization task.

For the example mentioned earlier , we can determine which of the vectors $\{(1, 1, 1, 1),$ $(1, -1, 1, -1), (-1, 1, -1, 1), (-1, -1, -1, -1)\}$ is closest to the received vector $(-0.9, 1.3, 0.2, 0.3)$, by computing all four distances. The result, $(-1, 1, -1, 1)$, is finally mapped back to the code vector 1010 and thence to the information vector 10.

For large codes, however, this exhaustive procedure is impractical since there are 2^k codewords, and k can be large. An efficient search procedure is needed to find the closest possible transmitted vector. We have accomplished this task using a genetic algorithm for $k = 52$, so far, requiring an order of magnitude less storage than the current best-known algorithm for this task (Han, 1993).

2 Walsh Polynomials and Nearest Neighbor Decoding

Walsh polynomials have been used as a benchmark for genetic algorithm performance by Tanese (Tanese, 1989). This section describes how the soft-decision decoding problem is equivalent to a Walsh polynomial optimization problem. A Walsh polynomial is a function of the form

$$f(\boldsymbol{x}) = \sum_{j \in B} w_j \psi_j(x)$$

where B is the set of l-bit strings and $\boldsymbol{x} \in B$. Note that each j can be uniquely identified with a vector of dimension l, $\boldsymbol{j}$. In the context of soft-decision decoding received vector components r_j play the role of w_j. Each r_j is real, and Walsh function $\psi_{\boldsymbol{j}}(\boldsymbol{x}) = (-1)^{\boldsymbol{j}.\boldsymbol{x}} = 1$ if $\boldsymbol{j}.\boldsymbol{x}$ has even parity and $\psi_{\boldsymbol{j}}(\boldsymbol{x}) = -1$ otherwise (Forrest & Mitchell, 1991). Consider the transformation:

$$(Z_2, +) \to (Z_2, *)$$

given by $a \mapsto (-1)^a$, where $a \in Z_2$. This isomorphism maps the additive binary group to the multiplicative binary group. Under this transformation, the c_j's defined earlier transform to $(-1)^{\boldsymbol{i}.\boldsymbol{g}_j} = 1$ if $\boldsymbol{i}.\boldsymbol{g}_j$ has even parity and -1 otherwise. This is the Walsh function corresponding to the j-th column of $\mathbf{G}$.

We need to minimize the following objective function for nearest-neighbor decoding:

$$\sum_{j=1}^{n}(r_j - c_j)^2 = \sum_{j=1}^{n}(r_j)^2 + n - 2\sum_{j=1}^{n} r_j c_j.$$

As the first and second terms are constants, this is equivalent to the task of maximizing: $\sum_{j=1}^{n} r_j c_j$. With $\boldsymbol{i}$ and the columns of $\mathbf{G}$ playing the role of $\boldsymbol{x}$ and $\boldsymbol{j}$ respectively, this is seen to be a problem of optimizing, $\sum_j r_j c_j$, a Walsh polynomial.

2.1 Decoding and Tanese Functions

Tanese reported (Tanese, 1989) that genetic algorithms performed rather poorly on optimizing a certain class of Walsh polynomials, genetic algorithms were in fact outperformed by hillclimbing [1]. An important investigation in this regard was undertaken by Forrest and Mitchell (Forrest & Mitchell, 1991) in which they remark that:

> in general, for a Tanese function of order n, no schema of order less than n will give any useful information; since for a given $\boldsymbol{j}$, half the instances of the schema will have even parity with respect to $\boldsymbol{j}$, and half will have odd parity with respect to $\boldsymbol{j}$. This is because each of the Walsh functions is of the same order. [Therefore] schema of lower order than n do not provide the GA with useful information.

Thus for problems equivalent to Tanese functions, crossover is not a useful tool for recombining building blocks. In the case of soft-decision decoding of linear block codes the $\boldsymbol{j}$'s correspond to the columns of the generator matrix, $\mathbf{G}$. Recall that $\mathbf{G}$ is constructed by listing the basis

[1]The Walsh polynomials generated and used by Tanese (Tanese, 1989) are hereafter referred to as Tanese functions. Tanese selected specific Walsh functions by randomly choosing 32 partition indices $\boldsymbol{j}$ all containing the same number of ones. The Walsh coefficient w_j for each of the 32 chosen partition indices was also chosen at random from the interval $(0.0, 0.5]$. The fitness function used in her experiments was a sum, $f(x) = \sum_j w_j \psi_j(x)$ of these 32 terms, other terms were effectively set to 0.

vectors of C as rows. Through elementary row operations $\mathbf{G}$ can be reduced to a systematic form, i.e:

$$\mathbf{G} = \left(\; I \mid P \; \right),$$

where I is a $k \times k$ identity matrix and P is an $n \times (n-k)$ matrix. Consider, for example the row reduced $\mathbf{G}$ matrix of the Hamming $[7,4]$ code:

$$\mathbf{G} = \left(\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array} \right)$$

The Walsh polynomials that arise as a consequence of soft-decision decoding consist of a sum of at most n terms, where n is the blocklength of the code. This is because there are n columns in the $\mathbf{G}$ matrix of an $[n,k]$ linear block code. Each transmission of a codeword gives rise to a potentially different Walsh polynomial as the components of $\boldsymbol{r}$ are likely to be different. We also note that not all Walsh functions are of the same order, in fact the first k terms are necessarily of order 1. This is because not all columns of the $\mathbf{G}$ matrix have the same Hamming weight, and the first k have Hamming weight (order) 1. Therefore this problem does not suffer from the lack of low-order schema processing characterizing Tanese functions and there is reason to believe that genetic algorithms would be a useful tool for the decoding problem. This motivates the development of the following genetic algorithm as a nearest-neighbor soft-decision decoding algorithm.

3 Algorithm GADEC

The following is an outline of an algorithm that performs a genetic search over the space of all codewords to search for the codeword nearest to received vector $\boldsymbol{r}$. For exhaustive experimentation, the algorithm first simulates the transmission of codewords over an additive white Gaussian noise channel. An important feature of this algorithm is the utilization of domain-specific "reliability" information for different components of the received vector.

Algorithm **GADEC**$(n, k, Y, p_m, p_{cross}, N)$.

Algorithm **GADEC** expects as input the blocklength, n, of the code, the dimension, k, of the linear code, the signal to noise ratio, Y, in decibels, the probability of mutating a single bit, p_m, the crossover probability, p_{cross}, and the size, N, of the population.

- STEP 1: Simulate message transmission.
  Randomly generate k binary information bits. Encode these information bits using the $\mathbf{G}$ matrix of the code, to yield an n-bit vector. Add simulated Gaussian noise after transforming each 0 to a 1 and each 1 to a -1 to obtain a received vector, $\boldsymbol{r} \in R^n$.

- STEP 2: Permute the coordinates of received vector $\boldsymbol{r}$ so that the first k positions are the most reliable linearly independent positions of $\boldsymbol{r}$.
  By assumption data is transmitted over an additive white Gaussian noise channel. Hence, r_i is considered to be more reliable than r_j if $|r_i| > |r_j|$. Permute the vector $\boldsymbol{r}$ in such a way that $|r_i| > |r_{i+1}|$, for $1 \leq i \leq n$. Further, permute the coordinates of $\boldsymbol{r}$ to ensure that the first k positions of $\boldsymbol{r}$ are its most reliable linearly independent positions. Call this vector $\boldsymbol{r}^{'}$. Modify the generator matrix of the code by applying the same transformation to the columns of $\mathbf{G}$ that produces $\boldsymbol{r}^{'}$ from $\boldsymbol{r}$, to get $\mathbf{G}^{'}$. Store permutation in vector $PERM$.

- STEP 3: Randomly generate a population of possible message vectors.
  Quantize the first k bits of $\boldsymbol{r}^{'}$ to obtain vector $\boldsymbol{h}$, which is used to seed the initial population. In addition, uniformly randomly generate $(N-1)$ number of vectors $\in \{-1, 1\}^k$. Let $best$ be the member of the population that is closest to $\boldsymbol{r}^{'}$.

- STEP 4: while(generation_counter < Total-Number-Generations) do
  - STEP 4.1: Compute the fitness of each individual in the population.
    An individual represents an information vector of length k bits which is encoded, $\boldsymbol{c} = \boldsymbol{i}\ \mathbf{G}^{'}$, to an n-bit codevector. The fitness function is the negated squared Euclidean distance $-\sum_{i=1}^{n}(r_i^{'} - c_i)^2$ between the received word and the encoded individual.
  - STEP 4.2: Sort population in increasing order of fitness.

```
    – STEP 4.3: Allot ranks to individuals in population and allocate reproductive trials to
      them.
    – STEP 4.4: while(population-size-counter < N ) do
        ∗ STEP 4.4.1: Randomly select two individuals $\boldsymbol{a}$ and $\boldsymbol{b}$ for reproduction.
        ∗ If(random_num < $p_{cross}$)
            · STEP 4.4.2: $Crossover(\boldsymbol{a}, \boldsymbol{b})$ with probability $p_{cross}$ to produce offspring $\boldsymbol{c}$.
              If $\boldsymbol{a} = (a_1, ..., a_k)$ and $\boldsymbol{b} = (b_1, ..., b_k)$ then:
              for each bit $i$ compute
              $P(d_i = 1) = \begin{cases} 0 & \text{if } a_i = b_i = -1 \\ 1 & \text{if } a_i = b_i = 1 \\ \frac{1}{1+\exp\frac{-2r_i^{'}}{\sigma^2}} & \text{if } a_i \neq b_i, \text{ where } \sigma^2 \text{ is the variance of the noise sample.} \end{cases}$
              Offspring $\boldsymbol{d}$ inherits 1 with probability $P(d_i = 1)$ and -1 with probability
              $1 - P(d_i = 1)$.
            · STEP 4.4.3: Mutate($\boldsymbol{d}$).
              Flip each bit of $\boldsymbol{d}$ with probability $p_{mut}$.
            · STEP 4.4.4: Introduce $\boldsymbol{d}$ as an individual into new population.
        ∗ else
            · STEP 4.4.5: Introduce either $\boldsymbol{a}$ or $\boldsymbol{b}$ into new population with equal probability.
        ∗ STEP 4.5: Let $currbest$ = fittest member of the new population. If fitness($best$) <
          fitness($currbest$), then $best = currbest$.
    – end while
      {loop invariant: Among all the vectors examined so far, $best$ is the closest to $\boldsymbol{r}^{'}$ }
• end while
• STEP 5: To $best$ apply the inverse of the permutation applied in STEP 2 of the algorithm.
  Return $\boldsymbol{c}^{'} = PERM^{-1}(best)$ as the decoded result.
```

3.1 Description of Algorithm GADEC

3.1.1 Codeword Transmission

This step is not a part of the decoder(**GADEC**) but is necessary in order to simulate a message source and noisy transmission channel. In STEP 2, the j-th component of the transmitted codeword $\boldsymbol{c}$ and the received vector $\boldsymbol{r}$ are $c_j = (-1)^{c_j}\sqrt{E}$ and $r_j = (-1)^{c_j}\sqrt{E} + e_j$ respectively, where E is the signal energy per channel bit and e_j is a noise sample of a Gaussian process with single-sided noise power per hertz N_0. The mean of e_j is zero and the variance, σ^2, is $N_0/2$. The signal-to-noise ratio for the channel is $Y = E/N_0$. In order to account for the redundancy in codes of different rates, the signal-to-noise ratio per transmitted bit, i.e., $Y_b = Yn/k$ is used. For simulation purposes, E is set to 1 and the mean and variance of e_i computed accordingly.

3.1.2 Chromosome Representation

In STEP 3, solutions to the optimization problem are represented as k-dimensional vectors. The k bits represent the information bits of a code-vector. Hence, crossover and mutation operate only on the information bits, which means that all the individuals in the population are always feasible solutions. An alternative strategy is to represent individuals as n bit codewords. Reproduction could then create an offspring that is not necessarily a feasible solution. Feasibility could be restored by following this up with a hard-decision decoding step to find the codeword closest in hamming distance to the n bit offspring.

3.1.3 Population Initialization

The initial population is generated uniformly randomly, so that every schema of a given length is equiprobably represented in the intial population. The initial population also contains a binary vector, $\boldsymbol{h}$, consisting of components $h_i = sgn(r_i^{'})$. [2] It is possible to seed the population with several good estimates of $\boldsymbol{i}$, the transmitted information vector, by using minor perturbations of $\boldsymbol{h}$, or perhaps choosing different information sets.

3.1.4 Fitness Evaluation

In STEP 4.1, to evaluate the fitness of an individual, it is necessary to first encode it by multiplying it with matrix $\mathbf{G}^{'}$. The squared Euclidean distance between $\boldsymbol{r}^{'}$ and this encoded vector is then computed. An individual is fitter than another if it is closer in squared Euclidean distance to $\boldsymbol{r}^{'}$.

3.1.5 Selection

In STEP 4.2 & 4.3, the selection strategy used is "Linear Ranking Selection". Individuals in the population are sorted by non-decreasing order of squared Euclidean distance and each individual is assigned a rank which determines the number of reproductive trials for that individual. This

[2] $\boldsymbol{r}$ is reordered so that the most reliable k linearly independent bits occupy the first k positions of $\boldsymbol{r}^{'}$.

approach was first proposed by Baker (Baker, 1985), as a means of slowing convergence. It has been reported to also result in more accurate optimization (Whitley, 1989).

3.1.6 RUX: Reliability Based Uniform Crossover

STEP 4.4.2 is the key component of genetic search. We have developed a unique crossover operator for this application, a variant of uniform crossover. In uniform crossover, a choice between inheriting a bit from either of two parents is made at each component of an offspring.

In population-elitist selection, every individual mates every generation without regard to its fitness (Schaffer, Eschelman, & Offut, 1991), after which offspring are pooled with all the parents and the best 50% are selected to yield the next generation. Algorithm **GADEC** implicitly uses elitism in that only one of two possible offspring is preserved; the one that is more likely to contain good schema. This algorithm explicitly enforces the survival of the best individual in the current generation into the next generation. These conservative selection policies and uniform crossover coupled with a technique for exploiting problem specific knowledge lead to very good performance. We discuss the performance of RUX and provide a comparison between RUX, uniform crossover and two point crossover in Section 4, Table 2.

RUX, the reliability based uniform crossover operator used in algorithm **GADEC**, repeatedly exploits the reliability of the received vector. Incorporating problem specific information is useful in any search problem; in genetic search, Grefenstette has exploited this for the TSP problem (Grefenstette, 1987).

RUX is a variant of uniform crossover where a choice is made between inheriting a bit from either parent based on a suitable probability function. For example:

$$P(c_i = a_i) = \begin{cases} 1 & \text{if } a_i = b_i \\ \frac{1}{1+\exp\frac{-2r_i^{'}}{\sigma^2}} & \text{if } a_i = 1 \neq b_i \\ 1 - \frac{1}{1+\exp\frac{-2r_i^{'}}{\sigma^2}} & \text{if } a_i = -1 \neq b_i \end{cases}$$

In case $a_i = 1 \neq b_i$, $P(c_i = a_i)$ approaches 1 as $r_i^{'}$ approaches ∞ and 0 as $r_i^{'}$ approaches $-\infty$. Similarly if $a_i = -1 \neq b_i$ then $P(c_i = a_i)$ approaches 1 as $r_i^{'}$ approaches $-\infty$ and 0 as r_i' approaches $-\infty$.

$P(c_i = a_i)$ depends on r_i in such a way that the probability of c_i inheriting either a_i or b_i increases, depending upon which one has the same sign as r_i. The crossover operator thus exploits reliability information provided by vector $\boldsymbol{r}^{'}$.

3.1.7 Mutation

STEP 4.4.3 is the mutation step. Mutation is done bit-wise on offspring with probability p_{mut}. Mutation rate for each bit is kept low, since the probability that a vector is perturbed is $1 - (1 - p_{mut})^k) \approx kp_{mut}$, and k is as high as 52 in some of the problems we have experimented with.

4 Simulation Results and Discussion

We present simulation results at various signal-to-noise ratios for the $[104, 52]$ extended binary quadratic residue code. This is a large code, with a search space of size 2^{52}. We present graphs and data showing the excellent performance of our approach. We illustrate the relation between bit error probability and the number of genetic algorithm generations. Simulation parameters for the results presented in Figures 2, 3, and Table 1 are: $n = 104, k = 52, p_{mut}$ =3%, $p_{\text{cross}} = 70\%$, N = population size = 300.

Figures 2 and 3 indicate the evolution of bit-error-probability with genetic algorithm generations. 1000 codewords were transmitted and the fraction of information bits in error calculated each time. Bit error probability is the average, over 1000 simulations, of the fraction of information bits in error. Notice that the bit-error-probability decreases with increasing number of generations, reflecting the fact that it is possible to balance solution quality with computational efficiency. Also notice in figure 3, the rapid initial evolution which settles down to a steady rate of improvement in bit-error-probability.

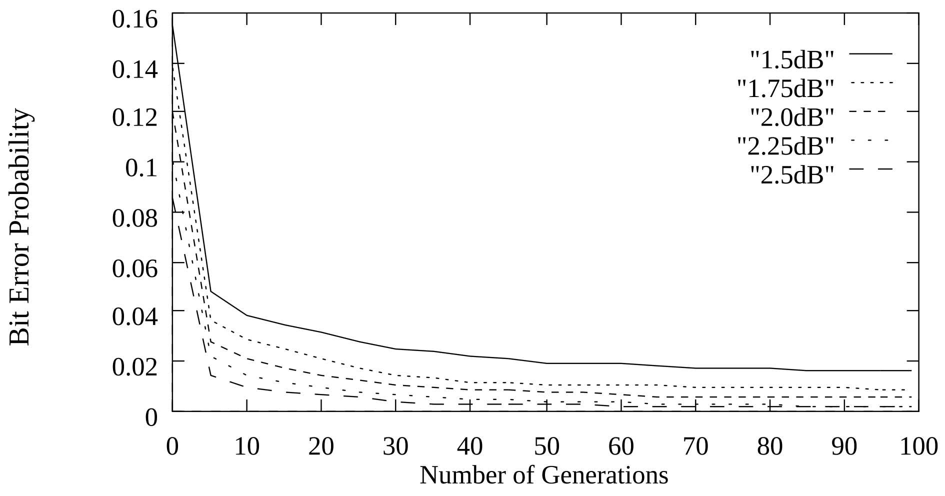

Figure 2: Bit Error Probability vs Number of Generations

An important question about this algorithm and, more generally, any genetic algorithm-based optimization technique, is about the stopping criterion. When should one stop iterating STEP 4 of algorithm **GADEC**? Since **GADEC** is iterative, one has the luxury of balancing

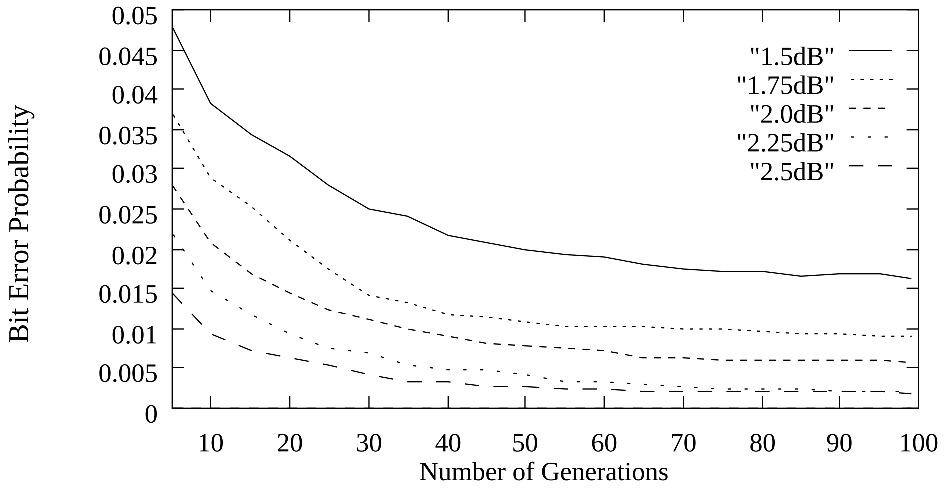

Figure 3: Bit Error Probability in the Later Stages of Evolution

performance and computational effort. Table 1 shows that there is a steady decrease in bit error probability with increase in the number of generations, with about 30 to 40% reduction when the number of generations is increased from 50 to 100. When iterated still further to 1000 generations, the bit error probability drops from .016 to .011 indicating that it is possible to obtain an improvement in performance at the cost of more computation.

For a given $[n, k]$ code, it is possible to perform a regression between the bit-error rate and numbers of genetic algorithm generations, based on simulation data. This will give a relation between bit error probability and an upper bound on the number of codewords evaluated (= number of generations$\times$ population size).

We present in Table 1, bit error probability and related statistics after 50 and 100 genetic algorithm generations. Figures 4 and 5 exhibit the relation between bit-error probability and the signal-to-noise ratio, after 50 and 100 generations, respectively. Some of these errors would necessarily be made by any maximum likelihood decoder (MLD) as well and reflect cases where a codeword other than the transmitted codeword was found to be closer to the received vector, $\boldsymbol{r}'$. This "lower bound" is also presented. The difference between the two curves, given in Figures 4 and 5, is often used to gauge the performance of a suboptimal decoding algorithm.

The results obtained using **GADEC** are excellent: the ratio of the bit error probability to the maximum likelihood decoding lower bound, is as low as 1.65 to 2.2 after 100 generations. As shown by the simulation results, it is possible to obtain a lower bit error probability and P_b/MLD ratio, at the expense of more computation.

For a fixed bit error probability, it is possible to compute the difference in SNR between the lower bound maximum-likelihood decoding curve and the curve obtained from algorithm **GADEC**. This difference is at most 0.55 dB after 50 generations of genetic search and reduces to at most 0.35 dB after 100 generations.

Signal-to-Noise Ratio, dB	1.5	1.75	2.0	2.25	2.5
Bit-Error Probability (Uncoded Data)	.0462	.0418	.0375	.0334	.0296
No. of Codewords Evaluated	30000	30000	30000	30000	30000
Number of Generations	100	100	100	100	100
P_b, Bit-Error Probability (Coded Data)	0.0165	0.00873	0.00563	0.00217	0.00183
MLD Lower Bound	0.00904	0.00404	0.00267	0.00135	0.00083
Ratio, P_b/MLD	1.82	2.16	2.10	1.65	2.2
No. of Codewords Evaluated	15000	15000	15000	15000	15000
Number of Generations	50	50	50	50	50
P_b, Bit-Error Probability (Coded Data)	0.019730	0.010838	0.007769	0.004200	0.002569
MLD Lower Bound	0.008307	0.004039	0.002676	0.001336	0.000810
Ratio, P_b/MLD	2.37	2.28	2.90	3.14	3.17

Table 1: Simulation Results for the $[104, 52]$ Code

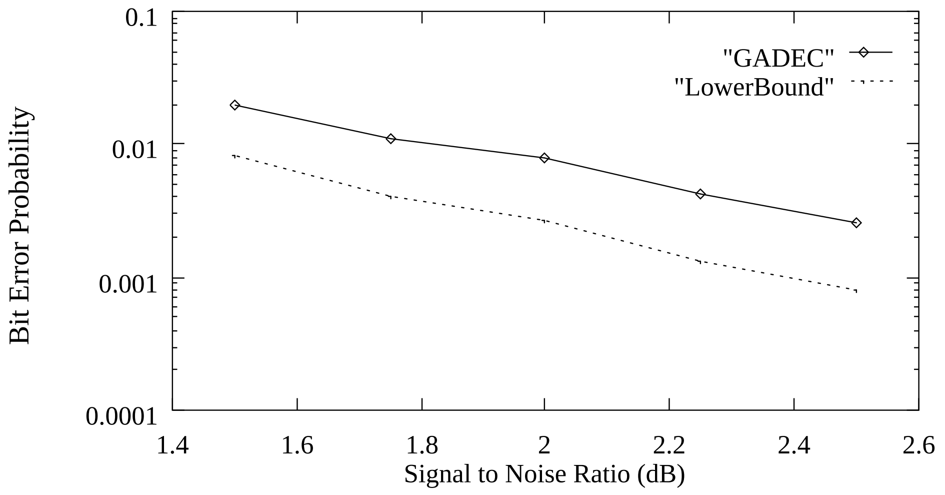

Figure 4: Bit Error Probability, (BEP) vs Signal-to-noise Ratio, 50 generations: [104,52] Code

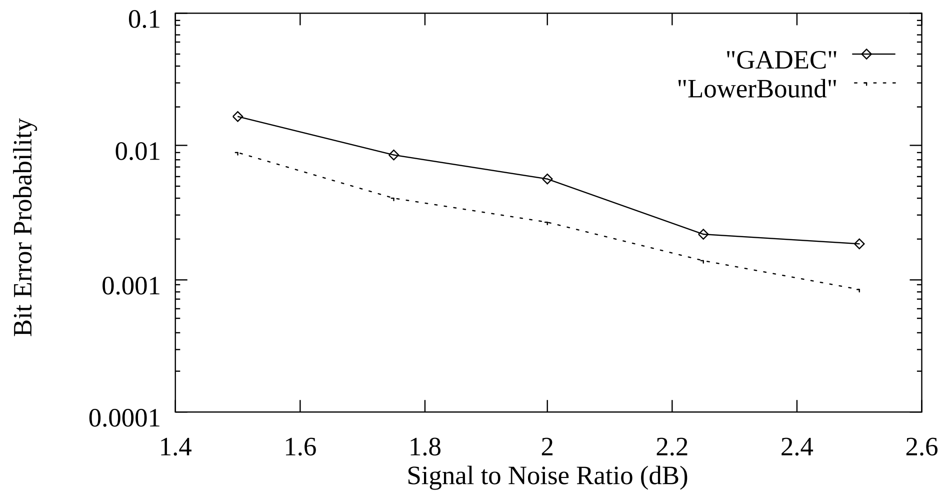

Figure 5: Bit Error Probability, (BEP) vs Signal-to-noise Ratio, 100 generations: [104,52] Code

Signal-to-Noise Ratio, dB	1.5	2.0	2.5
No. of Codewords Evaluated	30000	30000	30000
Number of Generations	100	100	100
P_b, Bit-Error Probability, RUX (Coded Data)	0.0165	0.00563	0.00183
P_b, Bit-Error Probability, UX (Coded Data)	0.151	0.116	0.0851
P_b, Bit-Error Probability, 2PTX (Coded Data)	0.130	0.098	0.065

Table 2: Comparison Between Different Crossover Operators in Algorithm **GADEC**

In Table 2 we present a comparison between results obtained using RUX, uniform crossover (UX), and two point crossover (2PTX) in algorithm **GADEC**. In the case of two point crossover the algorithm was modified to produce two offspring.

Simulation results show that RUX is superior to UX and 2PTX by at least an order of magnitude.

5 Analysis of Algorithm GADEC

In this section we show analyze the time and memory complexity of algorithm **GADEC**. We present a probabilistic analysis of the algorithm which includes an analysis of the role played by crossover.

5.1 Complexity Analysis

We show that algorithm **GADEC** has polynomial time complexity per generation.
Given that:

n = blocklength of code,
k = dimension of code = length of individuals in population,
N = population size = total number of individuals in population,

At any given stage, we maintain a few sets of $N \times k$ arrays, therefore the memory complexity of this algorithm is $O(Nk)$.

STEP 3 has time complexity of $O(k^2 n)$ (Han, 1993). STEPS 4.1 to 4.3 have a computational complexity of $O(knN) + O(N \log N) + O(k)$.
The complexity of STEP 3 depends on the random number generator in use; but the cost is negligible compared to that of STEP 4

STEPS 4.4.1 to 5 have an average case complexity of $O(1) + p_{cross}[O(k) + O(k) + O(k)] + (1 - p_{cross})[O(1) + O(k)] = O(1) + p_{cross}O(k) + (1 - p_{cross})O(k)$. This reduces to $O(k)$, which is also the worst case complexity. Hence each iteration of the genetic algorithm part of **GADEC** has a total time complexity of $O(knN + N \log N)$ per generation.

Algorithm **GADEC** has a time complexity of $O(knN + N \log N)$ per generation + an initialization complexity of $O(k^2 n)$.

5.2 Probabilistic Analysis of Algorithm GADEC

Next, we proceed to give a mathematical and empirical analysis of the algorithm. Such an analysis must necessarily be probabilistic because genetic algorithms are essentially stochastic processes. We start this analysis by computing the effect of STEP 2 on the decoding effort. We obtain the probability that the vector produced as a result of STEP 3, is the nearest neighbor code vector to the received vector $\boldsymbol{r}$. We know that vector $\boldsymbol{r} = (r_1, ..., r_n)$ has components r_i that are Gaussian random variables with mean $+1$ or -1 and variance $N_0/2$, as described in STEP 2 of algorithm **GADEC**.

If n independent and identically distributed random variables with probability density function $f(x)$ are rearranged in order of decreasing magnitude, denoted by $x_{(1)} \geq x_{(2)} \geq \cdots \geq x_{(n)}$, then the probability density function of $x_{(i)}$ is given by

$$p_i(x) = n \times \binom{n-1}{i-1} \cdot f(x) \cdot F^{n-i}(x) G_1^{i-1}(x)$$
$$\text{where } \; F(x) = \int_{-\infty}^{x} f(t)dt \; \text{ and } \; G(x) = 1 - F(x).$$

We are interested in computing the probability that STEP 3 of algorithm **GADEC** produces the nearest-neighbor codeword. Let $Y_i = |r_i|$; then $P(Y_i \leq x) = P(-x < r_i < x)$.

If r_i is received with mean $\mu = 1$, then

$$P(Y_i < x) = \int_{-x}^{x} \frac{1}{\sqrt{2\pi}\sigma} \, e^{-\frac{1}{2}\left(\frac{y-1}{\sigma}\right)^2} dy = \Phi\left(\frac{x-1}{\sigma}\right) + \Phi\left(\frac{x-1}{\sigma}\right) - 1 \tag{1}$$

$$\text{where } \; \Phi(u) = \int_{-\infty}^{u} \frac{1}{\sqrt{2\pi}} \, e^{-t^2/2} dt.$$

Likewise, if r_i is received with mean $\mu = -1$, then

$$P(Y_i < x) = \int_{-x}^{x} \frac{1}{\sqrt{2\pi}\sigma} \, e^{-\frac{1}{2}\left(\frac{t+1}{\sigma}\right)^2} dt = \Phi\left(\frac{x+1}{\sigma}\right) + \Phi\left(\frac{x-1}{\sigma}\right) - 1$$

In other words, the random variables $Y_1, \ldots, Y_n$ are distributed independently and identically, irrespective of the mean of the received component r_i for $i = 1, \ldots, n$ and the common distribution function of the Y_i's is given by equation (1) with associated density function

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \, e^{-\frac{1}{2}\left(\frac{x+1}{\sigma}\right)^2} + \frac{1}{\sqrt{2\pi}\sigma} \, e^{-\frac{1}{2}\left(\frac{x-1}{\sigma}\right)^2}; \quad \text{for } x > 0. \tag{2}$$

We will approximate the probability of STEP 3, producing the nearest-neighbor codeword by neglecting the effect of interchanges required to restore linear independence to the first k bit positions. These interchanges should not affect the probability very much (Battail & Fang, 1983), as there are very few of them; they are hence neglected in the analysis.

The probability of quantization yielding the transmitted bit is given by

$$\int_{0}^{\infty} \frac{1}{\sqrt{2\pi}} \, e^{-\frac{1}{2}\left(\frac{t-1}{\sigma}\right)^2} dt = \Phi\left(\frac{1}{\sigma}\right)$$

if 1 was transmitted, and

$$\int_{-\infty}^{0} \frac{1}{\sqrt{2\pi}} \, e^{-\frac{1}{2}\left(\frac{t+1}{\sigma}\right)^2} dt = \Phi\left(\frac{1}{\sigma}\right)$$

if -1 was transmitted. In other words, this probability of quantization yielding the transmitted bit is also independent of the transmitted bit.

The following expression for $A(n, j)$ approximates the probability that the j-th received vector component is quantized to its associated transmitted bit.

$$A(n,j) = \frac{n!}{(n-j)!(j-1)!} \int_{0}^{\infty} F(x)^{n-j} \cdot G(x)^{j-1} f(x) \, dx$$

where $f(x)$ is defined in Equation 2, $F(x)$ is the associated distribution function and, $G(x) = 1 - F(x)$.

The product $\prod_{j=1}^{k} A(n,j)$, approximates the probability that all of $r_{(1)}, \ldots, r_{(k)}$ are quantized to their associated transmitted bits. Since the first k positions uniquely determine a codeword, product $\prod_{j=1}^{k} A(n,j)$ represents the probability that STEP 3 yields the transmitted codeword. The results presented in Table 3 arise from simulating several codeword transmissions to observe

SNR (dB)	σ	Theoretical Estimate	Observed by Simulation
1.5	0.837	0.415	.398
1.75	0.813	0.482	.462
2.0	0.7905	0.546	.533
2.25	0.768	0.611	.592
2.5	0.746	0.673	.656
2.75	0.725	0.729	.719

Table 3: Estimates of $\prod_{j=1}^{k} A(n,j)$ for the $[104,52]$ Code

the frequency with which STEP 3 produced the transmitted codeword correctly. The theoretical estimate of probability given by $\prod_{j=1}^{k} A(n,j)$ computed using *Mathematica*, agrees very closely with simulation results for the $[104,52]$ code at six different signal-to-noise ratios.

Since we are interested in computing the probability that STEP 3 yields the nearest-neighbor codeword, $\prod_{j=1}^{k} A(n,j)$ does not account for the case where the transmitted codeword is not the nearest neighbor codeword. In this sense, too, $\prod_{j=1}^{k} A(n,j)$ is an approximation, albeit a reasonable one.

The above has provided us with a handle on the first step of algorithm **GADEC** and indicates that at least one member of the initial population is chosen reasonably close to the transmitted vector. In (Maini, Mehrotra, Mohan, & Ranka, 1993) we present a schema theorem for RUX showing that high reliability schema constitute the building blocks for the evolutionary process.

6 Comparison of Algorithm GADEC with Other Decoding Algorithms

We provide a comparison of algorithm **GADEC** with pure random search, a systematic exhaustive search method and an A* based algorithm. The comparison puts in perspective the excellent results obtained which are comparable in performance from the A* based algorithm, an order of magnitude better than the systematic exhaustive search method and several orders of magnitude better than pure random search.

6.1 Systematic Exhaustive Search

To further bolster the claim that the genetic algorithm is driven by the forces of reproduction and intelligent search space sampling, we give a comparison of the performance of algorithm **GADEC** with a systematic, exhaustive search procedure. The systematic exhaustive search procedure described below fixes some of the most reliable positions in a code vector and allow the others vary over all possibilities. This search exhaustively samples that schema of the search space that is most likely to contain a candidate nearest neighbor codeword.

Algorithm SystematicExhaustiveSearch(n, k, Y, dim)
n is the of the code, k is the dimension of the code, Y is the signal to noise ratio, in decibels, and dim the dimension of the subspace to be exhaustively searched.

- STEP 1: Same as STEP 1 of algorithm **GADEC**.
- STEP 2: Same as STEP 2 of algorithm **GADEC**.
- STEP 3: Compute squared Euclidean distance between vector $\boldsymbol{r}$ and transmitted codeword $\boldsymbol{c}$, call it $dist$. Let $\boldsymbol{h}$ be the quantized estimate obtained from vector $\boldsymbol{r}'$.
- STEP 4: while ($counter < 2^{dim}$) do
  - STEP 4.1: Store binary equivalent of integer $counter$, in vector $\boldsymbol{b}$.
  - STEP 4.2: Set $\boldsymbol{i} = (h_1, h_2, ..., h_{k-dim}, b_1, ..., b_{dim})$.
  - STEP 4.3: Encode $\boldsymbol{i}$, i.e. $\boldsymbol{c}' = \boldsymbol{i}\ \mathbf{G}'$.
  - STEP 4.4: Compute the squared Euclidean distance, $dist'$, between vector $\boldsymbol{c}'$ and vector $\boldsymbol{r}'$. Keep a running count of the minimum $dist'$ encountered so far. Find the number of its information bits in error.

Steps 1 through 3 of this algorithm are essentially the same as those of algorithm **GADEC**. Once vector $\boldsymbol{h}$ has been obtained, its most reliable $n-dim$ bits determines the subspace (schema) that is exhaustively searched for the codeword nearest in Euclidean distance to the received vector $\boldsymbol{r}$.

The following simulations were done on the $[104, 52]$ code with $dim = 15$, and the results are shown in Table 4. Bit error probability was computed in the same way as for algorithm **GADEC**, i.e., taking the average fraction of information bits in error over all simulated codeword transmissions. A total of $2^{15} = 32767$ codewords were evaluated for each simulated transmission. This is to provide a fair comparison with algorithm **GADEC** simulated to perform

SNR (dB)	SystematicExhaustiveSearch (BEP)	**GADEC** (BEP)
1.5	0.067	0.0165
2.0	0.045	0.00563
2.5	0.028	0.00183

Table 4: Systematic Exhaustive Search vs Algorithm **GADEC**

$30,000$ codeword evaluations. Simulation results obtained from this approach demonstrate that **GADEC** is superior by an order of magnitude showing that selection, crossover and mutation do play important roles in nearest-neighbor decoding.

6.2 Pure Random Search

To further support the claim that crossover plays the major role role in algorithm **GADEC**, simulation results with $p_{\text{cross}} = 0$ (crossover turned off) and a bit mutation rate of 3% show that the bit error probability rose from .0165 to .120 at a signal-to-noise ratio of $1.5dB$, which is a significant deterioration in performance.

We now analyze the behavior of a random search over the space of all codewords for the one closest in Euclidean distance to the received vector, r. This is done to show the failure of blind random search for this problem. We use the **extreme value theory** of random variables to approximate this behavior.

In a random sample of size n drawn from a population with cumulative distribution function $P(x)$, the asymptotic distribution of the largest/smallest element, $x_{(n)}$, may possess a limiting distribution. If it does, then the limiting distribution must be one of three possible types whose forms are found in (David, 1970). If $P(x)$ denotes the distribution of the standard normal random variable then the limiting cumulative distribution of $x_{(n)}$, the largest observation in a sample of n, normalized as $\sqrt{2\ln n}[x_{(n)} - \sqrt{2\ln n}]$, is $\exp(-e^{-x})$.

In pure random search we denote the squared Euclidean distance by random variable x_i, i.e., $x_i = \sum(r_j - c_{ij})^2$, where $\boldsymbol{r} = (r_1, \ldots, r_n)$ is the fixed received vector and c_{ij}'s are randomly generated binary bits. By the central limit theorem, x_i is normally distributed for large n. In the context of nearest-neighbor decoding we are interested in the distribution of the minimum x_i, and $\sqrt{2\ln n}[x_{(n)} - \sqrt{2\ln n}]$ describes the distribution of the maximum, consequently, we will consider the statistical behavior of $-\max(-x_i)$.

In case the c_{ij}'s are randomly selected, each satisfying the Bernoulli distribution with probabilty $p = 1/2$, each x_i has expected value $\sum_{j=1}^{n} r_j^2 + n$ and standard deviation $2\sqrt{\sum_{j=1}^{n} r_j^2}$. Hence, $-x_i$ has mean $\mu = -\sum_{j=1}^{n} r_j^2 - n$, standard deviation $\sigma = 2\sqrt{\sum r_j^2}$, and

$$(\frac{-x_i + \sum r_j^2 + n}{2\sqrt{\sum r_j^2}})$$

represents a standard normal random variable. Consider a search algorithm that randomly samples L codewords from a code of size 2^k. Then, by the extreme value theorem, the minimum distance among L codewords is

$$P\left[\frac{-x_{(L)} + \sum r_j^2 + n}{2\sqrt{\sum r_j^2}} - \sqrt{2\ln L} \leq \frac{y}{\sqrt{2\ln L}}\right] = e^{-e^{-y}}.$$

Let $x^* = -2\sqrt{\sum r_j^2}\left\{\frac{y}{\sqrt{2\ln L}} + \sqrt{2\ln L}\right\} + \sum r_j^2 + n$. For a given value of $e^{-e^{-y}} = c$ and a specified threshold x^*, we can estimate the number, L, of samples that would be required to discover a codeword at squared Euclidean distance less than x^*, with probability $1 - c$. This can be done by solving x^* as a quadratic equation in L.

The results presented in Table 5, give an estimate of L for threshold x^* set to some typical values of distance obtained after STEP 3 of algorithm **GADEC**. We use values of unconditional mean, $E(\sum_{j=1}^{j=n} r_j^2 + n) = \mu_{uc} = n(2+\sigma^2)$ and unconditional variance, $\sigma_{uc}^2 = 4n(1+\sigma^2)$, where $\sigma^2 = \frac{N_0}{2}$ (see description of **GADEC**) in place of conditional mean and variance, μ and σ^2 respectively. It is observed that L is a significant fraction (between $\frac{1}{2}$ and $\frac{2}{3}$) of the search space

SNR dB	1.5	1.75	2.0	2.25	2.5	2.75
Unconditional Mean, μ_{uc}	280	276.64	272.8	268.32	265.82	262.6
Unconditional St.Dev., σ_{uc}	26.5	26.27	25.99	25.63	25.44	25.18
A Typical x^*	116.01	183.82	93.55	109.97	92.32	91.65
Estimated Number of Codewords, L	10^8	10^8	10^{10}	10^8	10^9	10^9

Table 5: Pure Random Search vs Algorithm **GADEC**

that has size $2^{52} = 4.5 \times 10^{15}$, for the $[104, 52]$ code. Hence, if a random search procedure is going to produce any improvement over that obtained in STEP 3 of **GADEC**, it would probably take a very long time to do so.

6.3 Algorithm A* vs GADEC

In the A* based algorithm, a linear code is represented as a trellis wherein each path represents a codeword (Han, 1993). The suboptimal version of algorithm A* restricts the list of nodes to be expanded for exploration based on a limit on memory size and prunes search paths which are estimated to contain the required solution with a probability less than an A* specific threshold δ.

The bit error probability values obtained for the $[104, 52]$ code using a suboptimal version of algorithm A* ($delta = 0$) whose performance is stated in (Han, 1993) as being practically optimum are comparable to those of algorithm **GADEC** after 19,200 codeword evaluations.

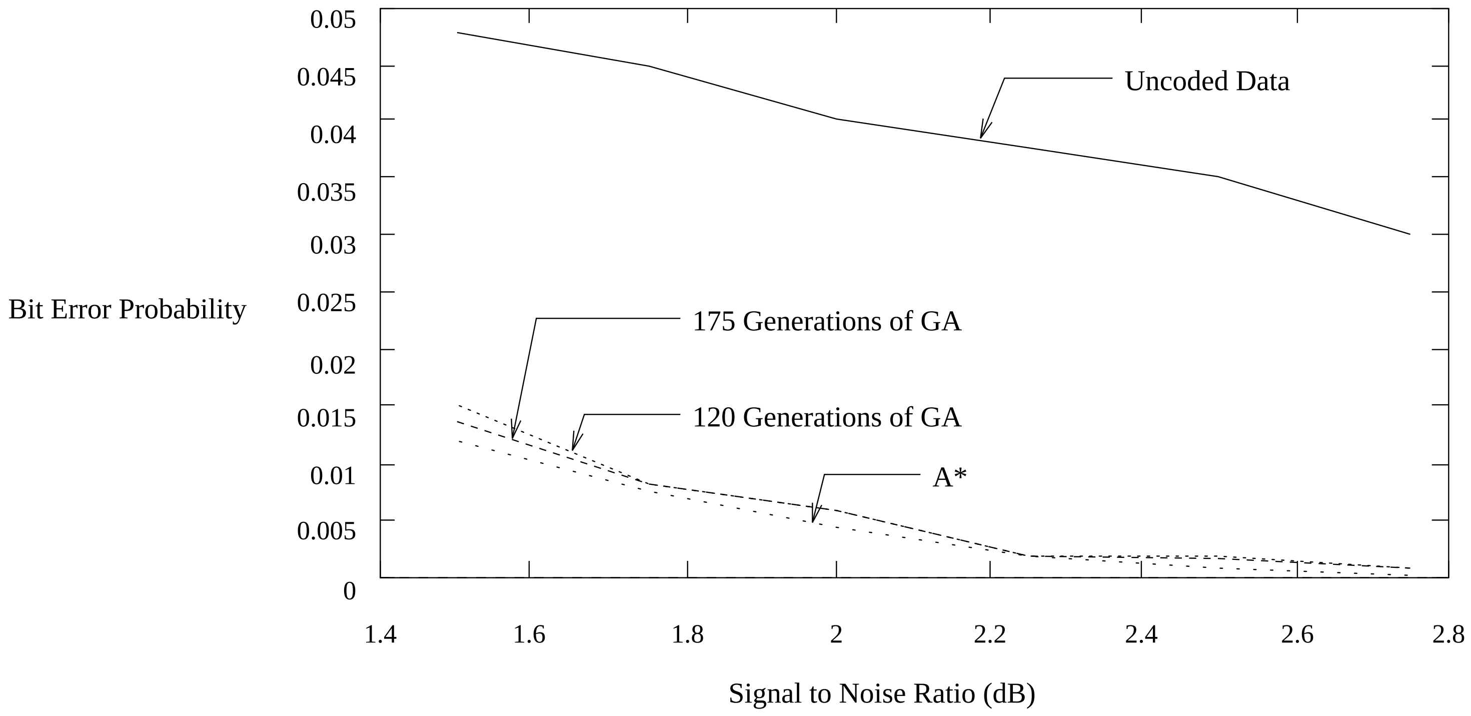

Figure 6: A Comparison of Bit Error Probabilites in Algorithm **GADEC** and A*

Table 6 and Figure 6 show the results of decoding using a distributed population version of our algorithm. This consists of 16 subpopulations each of size 10 configured as a mesh; subpopulation chromosomes are made available to neighboring subpopulations at intervals of 10 generations. Further, each subpopulation is reinitialized with genetic material upon convergence. We find that bit error probabilities and P_b/MLD ratios are comparable with those obtained via algorithm A* (see Figure 6 and Table 6). The P_b/MLD ratio indicates the fraction of errors that were actually committed by the decoder since MLD percentage of errors would actually have been committed by any optimal decoder. This ratio is what is used as a benchmark of performance, and we find that our algorithm yields P_b/MLD ratios in the 1 to 2.3 range, which is as good as that given by the A* algorithm.

An important advantage of **GADEC** is its low memory complexity of, which is $O(kN)$. This is contrast to algorithm A* which, in the worst case has a memory complexity that is exponential in the dimension of the code, $O(n2^k)$. In the suboptimal version of algorithm A*, whose results are presented in Table 6, the size of the open list used was restricted to 3000; this is still an order of magnitude larger than the memory required by the genetic algorithm.

Perhaps the most significant advantage of **GADEC** over the A* based approach is the fact that genetic algorithms are scalably parallel, suitable for implementation on a wide range of parallel architectures including massively parallel ones (Cohoon, Martin, & Richards, 1991; Muhlenbein, 1989; Manderick & Spiessens, 1989). The distributed population simulation results mentioned in Table 6 were conducted on a Sun workstation with a 33 Mhz CPU; for a single transmission, each genetic algorithm generation required approximately 2.0 seconds. (Several other algorithmic optimizations are also possible). This is in comparison with the A* algorithm which in the worst case takes time exponential in the dimension of the code and on the average takes over 8 minutes to decode a single transmission on a 33 Mhz Sun workstation. For the genetic algorithm, it is therefore reasonable to expect decoding times of approximately 18 seconds on a 16 processor parallel machine. On the other hand, an A* based algorithm is limited in speedup because it is necessary to compute the maximum node value at each level in the trellis before proceeding to the next one.

Additionally, the A* algorithm uses knowledge about the weight distribution of the code (as in the case of the $[104, 52]$ code) in computing the heuristic function (h) to improve its computation speed. The weight distribution of a code (the Hamming weights that its code vectors can have) is not always known and is a significant research problem in coding theory. The genetic algorithm approach makes no assumptions about the weight distribution of the code and is applicable to a superset of the codes to which the A* approach is applicable.

Therefore, we see that the genetic algorithm approach offers three advantages; an order of magnitude reduction in memory complexity, faster computation time and ease of parallelization, wide applicability to linear block codes irrespective of whether their weight distributions are known or not.

These advantages make the genetic algorithm using RUX an attractive decoding strategy in comparison with Algorithm A*.

7 Conclusions

We have investigated a novel and realistic application of genetic algorithms. We have presented an efficient genetic algorithm for soft-decision decoding, analyzed this algorithm, implemented it on a large code, of size 2^{52} and demonstrated the superiority of this method over other existing soft-decision decoding methods. We have presented a new crossover operator whose performance is superior by atleast an order of magnitude to conventional crossover operators. This is just the first step in what will emerge as a very important technique in soft-decision decoding of linear block codes with issues such as population representation, population size, distributed population models and crossover and mutation rates constituting important research topics in soft-decision decoding. The relationship between soft-decision decoding and optimizing Walsh polynomials, which has been observed for the first time, provides an interesting analogy between

Signal-to-Noise Ratio, dB	1.5	1.75	2.0	2.25	2.5	2.75
ALGORITHM **GADEC**						
No. of Codewords Evaluated	19200	19200	19200	19200	19200	19200
P_b, Bit-Error Probability MLD Lower Bound	.015 0.0086	.01 0.0046	.006 0.0036	.002 0.001	.002 .001	.00083 0.00036
Ratio, P_b/MLD	1.74	2.17	1.6	2.0	2.0	2.3
No. of Codewords Evaluated	22400	22400	22400	22400	22400	22400
P_b, Bit-Error Probability MLD Lower Bound	.015 0.0086	.008 0.0046	.006 0.0036	.002 0.001	.018 .001	.00083 0.00036
Ratio, P_b/MLD	1.74	1.73	1.6	2.0	1.8	2.3
No. of Codewords Evaluated	28000	28000	28000	28000	28000	28000
P_b, Bit-Error Probability MLD Lower Bound	.014 .0086	.008 .0046	.006 .0036	.002 .001	.017 .001	.00083 .00036
Ratio, P_b/MLD	1.62	1.73	1.6	2.0	1.7	2.3
No. of Codewords Evaluated	32000	32000	32000	32000	32000	32000
P_b, Bit-Error Probability MLD Lower Bound	.014 0.0086	.008 0.0046	.005 0.0036	.002 0.001	.017 .001	.00083 0.00036
Ratio, P_b/MLD	1.62	1.73	1.38	2.0	1.7	2.3
ALGORITHM A*						
P_b, Bit-Error Probability MLD Lower Bound	.01 .008	.0075 .005	.0045 .003	.002 .0015	.01 .0005	.00036 .00015
Ratio, P_b/MLD	1.25	1.5	1.5	1.3	2.0	2.3

Table 6: Comparison Between Algorithm A* and A Distributed Population Version of Algorithm **GADEC**: Decoding The $[104, 52]$ Code.

fundamental problems in genetic algorithms and information theory.

From the perspective of soft-decision decoding, results of a P_b/MLD ratio between 1.65 and 2.2 represent excellent performance in the presence of very high noise, i.e., between 1.5 dB and 2.75 dB. These results suggest that **GADEC** is a viable and good soft-decision decoding algorithm with a very low memory complexity, near-optimal performance and wide applicability, that can be iterated to balance computation with performance. The crossover operator developed for soft-decision decoding has been generalized and successfully applied to other optimization problems such as the graph partitioning and the TSP.

Acknowledgements

We would like to acknowledge the willing help and cooperation of Prof. C. R. P . Hartmann and Dr. Y. S. Han. We also thank Prof. H. F Mattson, Jr., for helpful comments.

References

Baker, J. E. (1985). Reducing bias and inefficiency in the selection algorithm. *Proc. of the 3rd ICGA*, 14-22.

Battail, G., & Fang, J. (1983). Décodage pondere optimal des codes lineaires en blocs. *Extrait Annales Des Telecommuinications, tome 38*, 11–12.

Cohoon, J. P., Martin, W. N., & Richards, D. S. (1991). A multi-population genetic algorithm for solving the k-partition problem on hypercubes. *Proc. of the 4th ICGA*, 244–248.

Clark, G. C., & Cain, J. B. (1988). *Error Correcting Coding for Digital Communications.* Plenum Press.

David, H.A. (1970). *Order Statistics.* NY: John Wiley & Sons.

Farrell, K. H., Rudolph, L. D., & Hartmann, C. R. P. (1983). Decoding by Local Optimization. *IEEE TIT, IT-29* (5).

Forrest, S., & Mitchell, M. (1991). The Performance of Genetic Algorithms on Walsh Polynomials: Some Anomalous Results and their Explanation. *Proc. of the 4th ICGA*, 182–189.

Grefenstette, J. J. (1987). Incorporating problem specific knowledge into genetic algorithms. In

L. Davis and M. Kaufmann (Eds.) *Genetic Algorithms and Simulated Annealing*, 420–460.

Han, Y. S. (1993). Efficient Soft Decision Algorithms for Linear Block Codes Using Algorithm A*. *Doctoral dissertation*, Syracuse University, (Technical Report SU-CIS-93-29).

Maini, H. S., Mehrotra, K. G., Mohan, C. K., Ranka, S. (1993). Genetic algorithms for soft-decision decoding of linear block codes. (Technical Report SU-CIS-93-25.) School of Computer and Information Science, Syracuse University: Syracuse, N.Y.

Manderick, B., & Spiessens, P. (1989). Fine-grained parallel genetic algorithms. *Proc. of the 3rd ICGA*, 428–434.

Muhlenbein, H. (1989). Parallel genetic algorithms, population genetics and combinatorial optimization. *Proc. of the 3rd ICGA*, 416–422.

Schaffer, J. D., Eschelman, L. J., & Offut, D. (1991). Spurious Correlations and Premature Convergence in Genetic Algorithms. *Foundations of Genetic Algorithms.* Morgan Kaufmann, 102–115.

Taipale, D. J., & Pursley, M. B. (1991). An Improvement to Generalized Minimum Distance Decoding. *IEEE TIT, 37* (1).

Tanese, R. (1989). Distributed Genetic Algorithms. *Proc. of the 3rd ICGA*, 434–439.

Whitley, D. (1989). The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. *Proc. of the 3rd ICGA*, 116–121.