Brock University

Department of Computer Science

Bounds on Optimal Edit Metric Codes

S.K. Houghten, D. Ashlock and J. Lennarz
Technical Report # CS-05-07
July 2005

Brock University
Department of Computer Science
St. Catharines, Ontario
Canada L2S 3A1
www.cosc.brocku.ca

Bounds on Optimal Edit Metric Codes

Sheridan K. Houghten
Department of Computer Science,
Brock University,
St. Catharines, Ontario, Canada L2S 3A1,
houghten@brocku.ca

Dan Ashlock
Department of Mathematics and Statistics,
University of Guelph,
Guelph, Ontario, Canada, N1G 2R4,
dashlock@uoguelph.ca

Jessie Lenarz
Department of Mathematics and Computer Science,
Concordia College,
Moorhead, Minnesota, USA 56562,
lenarz@cord.edu

Abstract

The *edit distance* between two strings is the minimal number of substitutions, deletions, or insertions required to transform one string into another. An error correcting code over the edit metric includes features from deletion-correcting codes as well as the more traditional codes defined using Hamming distance. Applications of edit metric codes include the creation of robust tags over the DNA alphabet.
While codes over the edit metric are analogous to similar codes over the Hamming metric, little of the beautiful theory survives. The *block structure* of a word is its partition into maximal subwords composed of a single character. The size of a sphere about a word in the edit metric is heavily dependent on the block structure of the word, creating a substantial divergence from the theory for the Hamming metric.

This paper explores the theory underlying edit metric codes for small alphabets. An *optimal code* is one with the maximum possible number of codewords for its length and minimum distance. We provide tables of bounds on code sizes for edit codes with short length and small alphabets. We present several heuristics for constructing codes.

1 Introduction

Error-correcting codes are traditionally designed to correct substitution errors occurring as a result of noise during transmission. For binary codes, the substitution has the effect of changing the value of a single bit in a transmitted word, from 0 to 1 or vice-versa. These codes are defined using Hamming distance, which measures the number of bits that differ from one word to another. The definition of these codes is easily extended to different alphabets.

An $(n, M, d)_q$ code is a set of M codewords of length n in which each symbol is an element of an alphabet of q elements, and for which the minimum distance is d. The maximum number of codewords possible in an $(n, d)_q$ code is denoted by $A_q(n, d)$. A $(n, M, d)_q$ code is called *optimal* if $M = A_q(n, d)$. For a small number of parameter sets, the exact value of $A_q(n, d)$ is known, but for the remainder, we have only upper and lower bounds. For $q = 2$, 3 and 4, tables of values (or bounds) for $A_q(n, d)$ may be found at [4, 6, 5].

Deletion-correcting codes are designed to correct errors that occur over the deletion and insertion metric. Such errors occur as a result of synchronization errors. These codes have been studied since the sixties by Levenshtein; in [10], the problem is defined and asymptotic upper bounds for the number of codewords in optimal codes are provided. More recently, constructions for perfect deletion-correcting codes have been provided by several researchers (see, for example, [11, 2, 13]). A survey of codes capable of correcting a *single* insertion or deletion may be found in [16]; this paper also addresses the problem of finding optimal single-deletion-correcting codes.

As described in [12], many different types of errors can occur during transmission, and naturally, all combinations of these different types are possible.

Error-correcting codes over the edit metric are designed to correct not only substition errors, but also synchronization errors. Such codes are defined using *edit distance*, also known as *Levenshtein distance*. In [9], we find an example of a construction for codes capable of correcting a *single* synchronization or substitution error (but not both). A construction for edit codes with large length and specified rate may be found in [8]; a decoding algorithm is also provided. Edit-metric codes also have applications in bioinformatics. An example of such an application is the creation of robust tags over the DNA alphabet [15].

In this paper we explore the theory underlying edit codes for small alphabets. It is important to note that these codes allow codewords of different lengths. Hence when we consider the structure of the edit space, this is indeed the case. However, when considering actual codes, we restrict ourselves to the case in which all codewords must have the same length.

We consider the fundamental problem of finding optimal edit codes, and describe issues relating to exhaustive searches for such codes. We also present several heuristics for constructing codes, and discuss their relative merits. We provide tables of bounds on code sizes for edit codes with short length and small alphabets, analogous to those described above for codes defined using Hamming distance.

2 Structure of the Edit Space

Let A be a finite alphabet. The *edit graph* $E(\mathcal{A})$ is the graph with vertex set $\mathcal{A}^*$ and edge set $\{\{s,t\} : d_{edit}(s,t) = 1\}$. The vertices are called *words*. The edit graph for a given alphabet has *levels* composed of generalized hypercubes. The induced graph on words of a fixed length is a generalized hypercube, the Hamming distance one graph, on those words. The neighbors of a word of a given length are words the same length or differing in length by one character. The generalized hypercubes are thus joined by edges associated with character insertion and deletion. Picturing these generalized hypercubes as stacked with the smallest on top yields a view of the graph as an infinite tower with each "floor" being the generalized hypercube for words of a given length. This mental picture, termed the "Tower of Babel" is a convenient viewpoint from which to consider $E(\mathcal{A})$. The induced subgraph on $\mathcal{A}^n$ is called the nth level.

Define $G_{\mathcal{A}}$ to be the automorphism group of $E(\mathcal{A})$. A *block* in a word is a maximal run of a single character. Define $d_edit(x,y)$ to be the edit distance between words x and y.

Lemma 1 *The degree of a vertex v on level n with k blocks is*

$$degree(v) = 2n(q-1) + q + k, \tag{1}$$

where $q = |\mathcal{A}|$.

Proof:

Notice there are $n(q-1)$ neighbors of a word on the same level representing substitutions. There are k neighbors on the next level up as all deletions within a block are equivalent. Finally there are $n(q-1)+q$ neighbors on the next level down. To see this last, note that the only way to go from length n to length $n+1$ is to insert a character. This can be done in three ways. First, we can add on the left or right end of the word. There are $q-1$ possible characters to insert to create each of these end extensions without simply lengthening an existing block, so this gives us $2(q-1)$ neighbors. Second, we can lengthen an existing block. This can be done in k ways. Finally, we can split an existing block by inserting a distinct character into it. There are $n-1$ places to insert a character, but we can not insert the same number of characters in every place. There are $k-1$ block boundaries and we can only insert $q-2$ characters there, since the other two characters would result in lengthening one of the blocks. The remaining $(n-1)-(k-1) = n-k$ places to insert have $q-1$ possible characters, since the other character would result in lengthening the block. So there are $(k-1)(q-2)+(n-k)(q-1) = n(q-1)-k-q+2$ ways to obtain a new word by splitting blocks. This results in a total of $2(q-1)+k+n(q-1)-k-q+2 = n(q-1)+q$ neighbors on the next level down. Summing the neighbors above, below, and on the same level yields the stated total. □

Lemma 2 *$G_{\mathcal{A}}$ acts on the levels of $E(\mathcal{A})$.*

Proof:

The degree of the empty word λ is q by lemma 1. This same lemma tells us all other words in the graph have higher degree. This means that $G_{\mathcal{A}}$ fixes λ. A minimal edit path from λ to a word w consists of adding the characters of w to λ in any one of a number of orders. This means that $d_{edit}(\lambda, w) = |w|$. The levels are thus sets of words at fixed distances from λ. Since automorphisms preserve distance it follows that $G_{\mathcal{A}}$ must take a word to a word of the same length and so we have that $G_{\mathcal{A}}$ acts on the levels of $E(\mathcal{A})$. □

Let $\sigma \in Sym(\mathcal{A})$ be a bijection of $\mathcal{A}$. If we apply σ simultaneously to every character of every word in $\mathcal{A}^*$ then we obtain a permutation of $\mathcal{A}^*$ induced by σ. We will denote this induced permutation by σ^*.

Lemma 3 *(***existence of a symmetric subgroup***)*
The map

$$\tau : Sym(\mathcal{A}) \to Sym(\mathcal{A}^*)$$

given by $\sigma \mapsto \sigma^*$ *is an injective group homomorphism of* $Sym(\mathcal{A})$ *into* $G_{\mathcal{A}}$.

Proof:

The induced maps in the image of τ simultaneously change the identity of all the letters in each word. If we have a minimal edit path that is a witness to the distance between two words in the graph then that path is preserved by the induced maps with the appropriate changes to the identity of substituted or inserted characters. The induced maps thus preserve distance and hence adjacency in the graph. If follows that the induced maps are automorphisms. Furthermore, the map is clearly injective. □

We will denote the image of τ by $\Sigma(\mathcal{A})$.

Lemma 4 *(***existence of** γ*)*
The map γ *from* $\mathcal{A}^*$ *to itself that flips all words end-for-end is an element of* $G(\mathcal{A})$ *that is not in* $\Sigma(\mathcal{A})$ *if* $|\mathcal{A}| > 1$.

Proof:

Clearly γ is an involution in $Sym(\mathcal{A})$. When we apply γ it is equivalent to changing reading order from left-to-right to right-to-left. Changing the reading order of words does not change the edit distance. If follows that $\gamma \in G_{\mathcal{A}}$. To see that $\gamma \notin \Sigma(A)$ when the alphabet has two or more letters examine its action on the word 011. We see that the image of 011 under γ is 110. This transformation cannot be achieved by maps of the sort in $\Sigma(\mathcal{A})$ because the middle character does not change its identity while the first and third do. □

Lemma 5 *(***commutativity of** γ*)*
For all $\sigma \in \Sigma(\mathcal{A})$, $\sigma\gamma = \gamma\sigma$.

Proof:

If we flip a word end-for-end and then change the identity of some subset of the characters in the word we obtain the same word as if we had performed the identity change and then flipped the word end-for-end. Thus $\sigma\gamma = \gamma\sigma$ for all $\sigma \in \Sigma(\mathcal{A})$. □

Lemma 6 *(**block number preservation**)*
Automorphisms of $E(\mathcal{A})$ preserve the number of blocks in a word.

Proof:

From Equation 1 we see that within a level degree is determined by the number of blocks in a word in a manner so that words with different numbers of blocks have different degrees. Since automorphisms preserve degree, it follows that they preserve the number of blocks. □

Definition 1 *For a word w and a character $a \in \mathcal{A}$ we denote by $|w|_a$ the number of occurrences of a in w.*

Theorem 1 $G_{\mathcal{A}} \cong \langle \gamma \rangle \times \Sigma(\mathcal{A})$.

Proof:

Recall that $G_{\mathcal{A}}$ acts on the levels of $E(\mathcal{A})$. A *monotone* word is a word composed of a single character. In other words, monotone words are those that have a single block. In a given level Equation 1 implies that the monotone words have the minimal degree within that level and are unique in having that degree within the level. As automorphisms preserve degree it follows that $G_{\mathcal{A}}$ acts on the monotone words within a level. Consider the representation f of $G_{\mathcal{A}}$ on the monotone words. The image of $\Sigma(\mathcal{A})$ under this representation is the full symmetric group on the monotone words. Since flipping a monotone word end-for-end does not change the word, it follows that $\gamma \in ker(f)$. Let $H = ker(f)$.

Claim: members of H preserve $|w|_a$. Since $G_{\mathcal{A}}$ acts on levels, it follows that when restricted to any given level, any member of $G_{\mathcal{A}}$ acts as an automorphism of the generalized hypercube forming the level. This means that it preserves distances within the generalized hypercube. Suppose we are on level n. Within level nm the distance from a monotone word made of the character a to any word w on level n is $n - |w_a|$. Since the monotone words are fixed points of members of H it follows that members of H preserve these distances and so preserve $|w|_a$. We have the claim.

From the claim we can deduce that H acts on the neighbors of any monotone word. Fix a monotone word w and denote by w_i^c the word that differers from it only in position i with c being the value of the character in the position where the word differs from w. Then we can see that members of H act on w_i^c for any fixed c by preservation of $|w|_c$. Fix c and study this action.

Claim: H acts on $\{w_k^c, w_{n+1-k}^c\}$. For $k = 1$ this is true by preservation of the number of blocks as these two words have two blocks while all other w_i^c have three. Assume the claim is true for all $l < k$ and for all levels of $E(\mathcal{A})$ above n. Examine w_k^c. If there exists an m such that w_k^c is taken to w_m^c by some member of H in a fashion that contradicts the claim then $k < m < n + 1 - k$ by examining the actions on the w_i^c already known for $l < k$. Let $\overline{w_i^c}$ be w_i^c with its first character deleted. The H acts on $\{\overline{w_k^c}, \overline{w_{n-k}^c}\}$ by the assumption. Notice that w_k^c and $\overline{w_k^c}$ are neighbors. By the assumption we know that if a member of H moves $\overline{w_k^c}$ that it must go to $\overline{w_{n-k}^c}$. The only member of w_i^c that

is a neighbor of $\overline{w^c_{n-k}}$ is w^c_{n+1-k}. It follows that no w^c_m of the sort envisioned exists and the claim follows by induction.

Examine the action of H on the set of w^c_l, $l = 1 \ldots n$ in level n of $E(\mathcal{A})$ for fixed c. By considering how the action of H can move 4-cycles made of the monotone word w, w^c_i, w^c_j, $i \neq j$, and the unique neighbor x of w^c_i and w^c_j for which $|x|_c = 2$, we see that the action of H on the w^c_l is exactly that of $\langle\gamma\rangle$. From what is known of the structure of the automorphism group of the generalized hypercube [3] we see that any members of H must act by simultaneously permuting the positions of characters within all words. The action of H on the w^c_l permits the only such action to be that of $\langle\gamma\rangle$. If follows that $H = \langle\gamma\rangle$. Given that γ commutes with $\Sigma(\mathcal{A}$ we can see that $G_{\mathcal{A}} \cong \langle\gamma\rangle \times \Sigma(\mathcal{A})$. $\square$

Corollary 1 *Automorphisms of $E(\mathcal{A})$ preserve the number of blocks and block sizes present in words.*

Proof:

Applying a member of $\Sigma(\mathcal{A})$ changes the identities of the characters in the blocks without changing the block structure. γ reverses the order of blocks without changing their number of size. The corollary thus follows from the theorem.

3 Optimal Edit Codes

We define $E_q(n, d)$ to be the maximum number of codewords in a q-ary code of length n with minimum edit distance d; as such an $(n, M, d)_q$ edit code is *optimal* if $M = E_q(n, d)$. We have several simple observations to make about certain values of $E_q(n, d)$.

Observation 1 *(***Hamming distance***)*
Since edit distance encompasses Hamming distance, an optimal code defined using Hamming distance must have at least as many codewords as an optimal code defined using edit distance. Therefore $E_q(n, d) \leq A_q(n, d)$.

Observation 2 *(***Trivial codes***)*
If $d > n$ then $E_q(n, d) = 1$.

Observation 3 *(***Codes with** $n = d$*)*
If $n = d$ then $E_q(n, d) = q$.

Observation 4 *(***Codes with** $d = 1$*)*
If $d = 1$ then no insertions or deletions are possible, as we consider only codes for which all codewords are required to have the same length. The set of all q-ary vectors of length n forms a code with minimum distance $d = 1$. Therefore $E_q(n, d) = q^n$ if $d = 1$.

The following theorem relates upper bounds for different parameter sets and is analogous to a similar theorem for codes defined using Hamming distance.

Theorem 2 $E_q(n,d) \leq q \cdot E_q(n-1,d)$.

Proof:

Suppose instead that $E_q(n,d) > q \cdot E_q(n-1,d)$. Then there must exist some $(n, E_q(n,d), d)$ code C. Partition the vectors of C according to the symbol contained in the first column, forming $C_0, \ldots C_{q-1}$. Suppose that C_i is the largest of these codes. Then the number of codewords in C_i must be at least $\lceil E_q(n,d)/q \rceil > \lceil q \cdot E_q(n-1,d)/q \rceil = E_q(n-1,d)$. Every codeword in C_i is at distance at least d from every other codeword in the same code; furthermore, if we remove the first column from C_i, then the resulting truncated code also has this same property. Therefore we have an $(n-1, M, d)_q$ code with $M > E_q(n-1,d)$, which is a contradiction. Therefore $E_q(n,d) \leq q \cdot E_q(n-1,d)$. □

An important note relating to the above proof is that if we partition C according to the symbol contained in any column other than the first or the last, and then remove that column, then the resulting code may have a different minimum edit distance. This is a feature of edit codes that does not exist in codes defined using Hamming distance.

Codes with $d = 2$ are well-defined as they allow only 2 substitutions, or a single insertion/deletion combination. We have the following theorem.

Theorem 3 *(***Codes with** $d = 2$*)*
If $d = 2$ *then* $E_q(n,d) = q^{n-1}$.

Proof:

From Theorem 2, we already have $E_q(n,2) \leq q \cdot E_q(n-1,2)$. Also, observe that we can create an $(n,2)_q$ code with q^{n-1} codewords by listing all codewords of length $n-1$ and appending a parity check that is the sum of all previous entries mod q. Notice that this parity check code has minimum edit distance 2 because a single deletion or insertion changes the length of the code word and so an insertion and a deletion are both required to move between words of the same length. Thus we hit the upper bound. □

The following theorem is useful when performing an exhaustive search for codes with a fixed number of codewords. For any code C, define C_i to be the subcode of C obtained by taking all codewords of C starting with the symbol i.

Theorem 4 *Any* $(n, M, d)_q$ *code* C *is equivalent to some code* C' *with* $|C'_0| \geq |C'_1| \geq \ldots \geq |C'_{q-1}|$.

Proof:

Suppose that $|C_i| > |C_j|$ for some $i < j$. From Theorem 1, we can exchange the symbols i and j in all codewords simultaneously to produce an equivalent code C' with $|C'_i| < |C'_j|$. By performing a similar process for any other symbols out of order, we obtain a code C' with $|C'_0| \geq |C'_1| \geq \ldots \geq |C'_{q-1}|$. □

4 Exhaustive Searches and Heuristic Algorithms

Our goal is to determine bounds on the value $E_q(n,d)$ for small values of n, d and q. To determine the exact value of $E_q(n,d)$, or to decrease upper bounds on

this value, we use exhaustive computer searches. We use heuristic algorithms to set lower bounds once the exhaustive searches become too time-consuming.

4.1 Exhaustive Searches

From section 3, we already know exact values for $E_q(n, d)$ when $d = 1$ or 2, and when $n \leq d$. Combined with Theorem 2, these give very loose upper bounds for all remaining parameter sets.

It is possible to decrease upper bounds by employing a method analogous to that described in [14] for codes defined using Hamming distance. Suppose that we have $E_q(n, d) \leq M$. Then from Theorem 4, any code C with M codewords must be equivalent to one in which at least M/q codewords start with 0. Furthermore, if we remove the first column of C_0, the truncated result is an $(n-1, d)_q$ code.

Looking at it from the opposite direction, consider the set of all all $(n-1, M', d)_q$ codes with $M' \geq M/q$. For each such code D, insert a column of zeroes at the start of D, and then extend it by performing an exhaustive search for the maximum-size set of codewords at edit distance at least d from each other, and from D itself. If the resulting code has M codewords, then we have $E_q(n, d) = M$. If none of these searches resulted in a code with M codewords, then we must have $E_q(n, d) \leq M - 1$.

To further improve the search, we may assume that while extending a code D, there are at least as many codewords starting with the symbol i as there are codewords starting with the symbol j, for any $1 \leq i < j \leq q - 1$.

This argument may also be applied in a backtrack search for optimal $(n, d)_q$ edit codes when we are not aiming at any predetermined upper bound. By keeping track of the number of codewords M in the best $(n, d)_q$ code seen so far, we can ensure that in any subsequently-generated codes C with M codewords, we always have $|C_0| \geq M/q$ and $|C_0| \geq |C_1| \geq \ldots \geq |C_{q-1}|$.

4.2 Heuristic Algorithms

Conway's lexicode algorithm [7], given as Algorithm 1, is a simple greedy algorithm that permits immediate construction of edit metric codes of modest length. Two search heuristics that incorporate versions Conway's algorithm are also presented. The first is a *greedy closure evolutionary algorithm* [1] that uses small initial segments of codes to modify the trajectory of Conway's algorithm. This heuristic has the best performance of those presented but is quite slow. The second heuristic, a far faster evolutionary algorithm for locating edit metric codes, permits search for codes of greater length and is presented here for the first time. In this latter heuristic, Conway's algorithm is used as a means of blending two existing codes together with a random word as part of a search for a larger code.

Algorithm 1 Conway's Lexicode Algorithm

Create an initial population at random.
Repeat 40,000 times:
  Select seven chromosomes at random.
  Copy the two best as "parents".
  Apply variation operators to the copies.
  Place resulting new code(s) in place of the worst.
End Repeat;
Report largest code size found.

Figure 1: The basic loop of the evolutionary heuristics.

Input: A minimum distance d and a word length n.
Output: An $(n, d) - code$.
Algorithm:

Place the binary words of length n in lexicographical order. Initialize an empty set C of words. Scanning the ordered collection of binary words, select a word and place it in C if it is at Hamming distance d or more from each word placed in C so far.

Evolutionary algorithms are computational analogies to the theory of evolution. They are quite simple to write and, typically, are most useful when very little is known about the problem being addressed. As more knowledge is gained, domain specific heuristics and theorems take the place of the initial search algorithms. The basic loop of the evolutionary heuristics used is given in Figure 1. Evolutionary algorithms operate on a population of putative solutions called *chromosomes*. The *variation operators* are used to create new structures that are variations of existing ones. Binary variation operators that combine material from two parents are analogous to biological sex while unary variation operators are analogous to biological mutation. The process of *selection* picks chromosomes with a quality bias to be reproduced and varied. The evolutionary heuristics here were run for a fixed amount of time and the size of the resulting codes then recorded. For each set of code parameters 100 distinct evolutionary simulations, with different random number seeds, were performed.

The chromosomes for the first evolutionary heuristic are lists of three code words that satisfy the minimum distance requirements of the desired code. These chromosomes are called *code seeds*. An initial population is generated at random, rejecting words that violate the minimum distance criterion, until three acceptable words are found. The quality of a code seed is assessed by modifying Conway's algorithm to start with the code seed rather than an empty set C of words. The size of the resulting code is the quality measure for the code seed. One binary and one unary variation operator were used for this

heuristic. The binary operator creates two new code seeds placing one copy of any common words into both the seeds and randomly distributing the others to obtain two seeds with three members. If this creates a seed whose members violate the minimum distance requirement of the desired code the quality of the seed is defined to be zero. This heuristic was applied to a population of 120 code seeds in each evolutionary run.

The chromosomes for the second heuristic directly stores edit metric codes as lists of strings. The initial population is generated by applying Conway's algorithm to 400 randomly generated words in the order in which they were generated. The quality of a code is simply its size. A binary variation operator is used to produce a single new code from two others. The two parent codes are shuffled, together with a single new random word, into random order. Conway's algorithm is then applied to "polish" the resulting list of words to obtain a code obeying the minimum distance requirements.

The former heuristic lists all words in the code space each time it evaluated a code seed. This means that evaluation of one population member is exponential in the length of the code word. Codes resulting from this heuristic are constructively maximal. The second heuristic operates directly on codes, using a modification of Conway's algorithm as a search operator. The creation and evaluation of a new individual are combined with an algorithm time roughly proportional to the square of the sum of the sizes of the codes, Z, times the square of the length of the code words L, $O(Z^2L^2)$. This algorithm time derives from the need to compare all words in the list of words being polished to those that make it into the code over the edit metric which has time word-length squared. Since words are rejected the first time they fail to meet the distance bound, and since the $O(L)$ Hamming distance bounds the edit distance, the algorithm can be made fairly fast relative to its $(ZL)^2$ time complexity. Rejecting words based on hamming distance often permits the algorithm to avoid an edit distance computation, thereby saving a significant amount of time. In short the second heuristic is far faster even though it does not produce constructively maximal codes. Codes produced by the second heuristic can be used as (extra large) code seeds and "finished" with Conway's algorithm.

5 Best Known Results

In this section we present tables of bounds on the number of codewords in an optimal code for $q = 2$, 3 and 4. For binary and ternary codes, we provide results for $n \leq 12$ and $d \leq 12$; for quaternary codes, we restrict ourselves to $n \leq 10$ and $d \leq 10$. To the best of our knowledge, these are the best results for these parameter sets.

5.1 Optimal Binary Edit Codes

Table 1 shows the value of $E_2(n, d)$ for $n \leq 12$ and $d \leq 12$. For the cases for which the exact value of $E_2(n, d)$ is unknown, upper and lower bounds are given.

n\d	1	2	3	4	5	6	7	8	9	10	11	12
1	2	1	1	1	1	1	1	1	1	1	1	1
2	4	2	1	1	1	1	1	1	1	1	1	1
3	8	4	2	1	1	1	1	1	1	1	1	1
4	16	8	2	2	1	1	1	1	1	1	1	1
5	32	16	4	2	2	1	1	1	1	1	1	1
6	64	32	7*	4	2	2	1	1	1	1	1	1
7	128	64	12	5	2	2	2	1	1	1	1	1
8	256	128	19*	9	4	2	2	2	1	1	1	1
9	512	256	31^g 36^u	13	5	4	2	2	2	1	1	1
10	1024	512	54^g 72	21^g 24^u	8*	4	2	2	2	2	1	1
11	2048	1024	90^g 144	30^g 48	11	6	4	2	2	2	2	1
12	4096	2048	160^g 256^h	50^g 96	15^g 22	9	4	4	2	2	2	2

Table 1: Table of Upper and Lower Bounds for $E_2(n, d)$

Unless otherwise specified, upper bounds are determined by applying Theorem 2.

This table shows several interesting results. First, entries denoted by an asterisk are cases for which no optimal code containing the all-zero vector exists. This demonstrates a departure from codes defined using Hamming distance, for which one can always assume the existence of the all-zero vector.

Entries denoted by the superscript h are cases for which the upper bound for $E_2(n, d)$ is taken from the known upper bound for $A_2(n, d)$ given in [4]. In other words, the upper bound is obtained by looking at codes with the same parameters defined using Hamming distance alone.

Entries denoted by the superscript u are cases for which the upper bound for $E_2(n, d)$ was decreased by the method described in section 4.1. It is interesting to note that as a result of this process, we now have $E_2(9, 3) \leq 36$; this in turn implies $E_2(10, 3) \leq 72$ and $E_2(11, 3) \leq 144$, each of which are equal to the known values given in [4] for codes defined using Hamming distance.

Entries denoted by the superscript g are cases for which the best known code was found by the seed-based genetic algorithm.

5.2 Optimal Ternary Edit Codes

Table 2 shows the value of $E_3(n, d)$ for $n \leq 12$ and $d \leq 12$. For the cases for which the exact value of $E_3(n, d)$ is unknown, upper and lower bounds are given. Unless otherwise specified, upper bounds are determined by applying Theorem 2. Entries denoted by the superscript u are cases for which the upper bound for

n\d	1	2	3	4	5	6	7	8	9	10	11	12
1	3	1	1	1	1	1	1	1	1	1	1	1
2	9	3	1	1	1	1	1	1	1	1	1	1
3	27	9	3	1	1	1	1	1	1	1	1	1
4	81	27	7^{*}	3	1	1	1	1	1	1	1	1
5	243	81	15	5	3	1	1	1	1	1	1	1
6	729	243	33 38^{h}	13^{*}	4	3	1	1	1	1	1	1
7	2187	729	71 111^{h}	21 33^{h}	8	3	3	1	1	1	1	1
8	6561	2187	179 333^{h}	41 99^{h}	15 24	7	3	3	1	1	1	1
9	19683	6561	438^{c} 937^{h}	88 297^{h}	25 72	12 18	6^{*}	3	3	1	1	1
10	59049	19683	1133^{c} 2811^{h}	202^{c} 891^{h}	49 216	18 54	9 14^{h}	5	3	3	1	1
11	177147	59049	2972^{c} 7029^{h}	471^{c} 2561^{h}	98 648	29 162	11 36^{h}	7 12^{h}	4^{h}	3^{h}	3	1
12	531441	177147	7819^{c} 19683^{h}	1129^{c} 6839^{h}	208^{c} 1557^{h}	56 486	20 108^{h}	10 36^{h}	7 9^{h}	4^{h}	3^{h}	3

Table 2: Table of Upper and Lower Bounds for $E_3(n, d)$

$E_3(n, d)$ was decreased by the method described in section 4.1.

As for the binary case, entries denoted by an asterisk are cases for which no optimal code containing the all-zero vector exists. Also, entries denoted by the superscript h are cases for which the upper bound for $E_3(n, d)$ is taken from the known upper bound for $A_3(n, d)$ given in [6].

Entries denoted by the superscript c are cases for which the best known code was found by Conway's Lexicode algorithm, while entries denoted by the superscript g are cases for which the best known code was found by a genetic algorithm. It is interesting to note that for the more difficult cases, the lexicode algorithm easily outperformed the fast genetic algorithm; this is most likely simply a case in which the genetic algorithm can be improved by tuning of parameters. At present the sole advantage of the fast genetic algorithm is its substantially greater speed, a factor only important in applications.

5.3 Optimal Quaternary Edit Codes

Table 3 shows the value of $E_4(n, d)$ for $n \leq 10$ and $d \leq 10$. For the cases for which the exact value of $E_4(n, d)$ is unknown, upper and lower bounds are given. Unless otherwise specified, upper bounds are determined by applying Theorem 2. Entries denoted by the superscript u are cases for which the upper bound for $E_4(n, d)$ was decreased by the method described in section 4.1.

n\d	1	2	3	4	5	6	7	8	9	10
1	4	1	1	1	1	1	1	1	1	1
2	16	4	1	1	1	1	1	1	1	1
3	64	16	4	1	1	1	1	1	1	1
4	256	64	16^*	4	1	1	1	1	1	1
5	1024	256	41 64	11^*	4	1	1	1	1	1
6	4096	1024	106^g 179^h	28 40^u	8	4	1	1	1	1
7	16384	4096	329^g 614^h	63^g 160	18 32	8^*	4	1	1	1
8	65536	16384	1025^c 2340^h	164^c 611^h	38 128	14 32	5^h	4	1	1
9	262144	65536	3451^c 9360^h	481^c 2314^h	90^c 512	26 128	11 20^h	4 5^h	4	1
10	1048576	262144	11743^c 30427^h	1463^c 8951^h	242^c 2045^h	57^c 496^h	19 80^h	10 16^h	4 5^h	4

Table 3: Table of Upper and Lower Bounds for $E_4(n, d)$

As before, entries denoted by an asterisk are cases for which no optimal code containing the all-zero vector exists. Also, entries denoted by the superscript h are cases for which the upper bound for $E_4(n, d)$ is taken from the known upper bound for $A_4(n, d)$ given in [5].

Entries denoted by the superscript g are cases for which the best known code was found by a genetic algorithm. The quaternary genetic algorithm results were all obtained with the greedy closure genetic algorithm.

6 Applications

An application of edit metric lexicodes, over the alphabet **C, G, A, T** is to provide embeddable markers for complementary DNA libraries. These libraries are used to find genes by taking transcribed RNA and re-transcribing it back to DNA, after which it is embedded in an *e-coli* artificial chromosome. This practice permits the researcher to use the information, intrinsic in an organism, of which small fraction of its DNA actually encode genes. These gene-capture constructs are called *expressed sequence tags* or ESTs.

Most genes in an organism are not being transcribed all the time. This means that a researcher will typically treat an organism in many ways to induce transcription of genes involved in disease response, heat shock protection, or other circumstances. Since cDNA libraries are pooled for sequencing (in order to contain cost) embedded markers can be used to preserve information about the treatment(s) that induced transcription of a given gene. The sequencing

of ESTs is an error-prone process and so the ability to correct errors in the embedded tags is valuable.

Because these markers are to be embedded in constructs there are a number of constraints on the sequence that may be used that are driven by biology beyond the need for error correction. In making the DNA constructs various restriction enzymes are used which cut a DNA strand at a particular pattern. Creating additional instances of this pattern either within our markers or as a side effect of embedding our markers into the construct. Substrings of the form TT or AAA will interfere with use of the construct because of a long string of T's near the point were the marker is embedded. It turns out that modifying the heuristics to deal with such constraints is not difficult. Code that checks the acceptability of each word is used whenever a net code word is produced. Embedded error-correcting markers obeying the constraints given below have been synthesized into cDNA libraries for zea Mays (corn)[15]. The constraints shrink the size of the resulting codes substantially and the codes located were about three-quarters the size of the unrestricted codes.

- The code word must not end in T.
- The strings TT and AAA must not appear.
- When the code word replaces the NNNNNN in the nucleotide string: AACTGGAAGAATTCGCGGCCGCNNNNNNTTTTTTTTTTTTTTT-TTT no new instances of the following restriction enzyme sites must be created: GCGGCCGC, GAATTC, or CAGCTG.

7 Conclusions and Future Directions

This paper identifies the fundamental problem of determining $E_q(n, d)$, the maximum possible number of codewords in an edit code for given parameters, n, d and q. We have presented tables of best-known values for $E_q(n, d)$, for $q = 2$, 3 and 4, and for small values of n and d. In many cases, we do not have an exact value but rather a set of upper and lower bounds; in several cases those bounds, especially upper bounds, are somewhat weak. Therefore an obvious next step is to improve these bounds where possible. These tables could also be expanded to include a larger range of parameter sets.

Three heuristics were used to obtain lower bounds; Conway's algorithm, the greedy closure genetic algorithm, and the novel genetic algorithm that uses Conway's algorithm as a binary variation operator. There was an absolute correlation of quality and runtime. The slowest algorithm, the greedy closure genetic algorithm, provided the best results whenever there was sufficient computer power to permit it to finish running. The novel genetic algorithm, designed for speed, was bested by the unmodified version of Conway's algorithm. For long codes, such as length 30 binary or length 20 quaternary, with relatively modest minimum distance such as 5, the basic Conway's algorithm will take hours to terminate where the rapid heuristic will generate some sort of code in a few

seconds and then begin improving it. The rapid algorithm is thus more useful in the bioinformatics applications than in searching for examples germane to the problem of bounds.

It is also of interest to further examine the structure of the edit space. We have shown that the size of a sphere about a word in the edit metric depends upon the block structure of that word; this is substantially different from the Hamming metric, in which all words behave in an equivalent manner. It would be of interest to further consider the cases in which no optimal code exists containing the all-zero vector.

Finally, we have considered edit codes with the restriction that all codewords must have the same length. One reason for doing this was because of the type of application. Nonetheless it is still of great interest to examine the closely-related problem in which the lengths of codewords may differ.

8 Acknowledgments

This work was supported in part by the National Sciences and Engineering Research Council of Canada. This research was also partially supported by a competitive grant from the National Science Foundation Plant Genome Program (award number: DBI-9975868) to Patrick S. Schnable, Daniel Ashlock and others.

The first author would like to thank Wolfgang Haas and Justin Kalicharan for their valuable comments and assistance.

References

[1] Dan Ashlock, Ling Guo, and Fang Qiu. Greedy closure genetic algorithms. In *Proceedings of the 2002 Congress on Evolutionary Computation*, pages 1296–1301, Piscataway, NJ, 2002. IEEE Press.

[2] Patrick A.H. Bours. On the construction of perfect deletion-correcting codes using design theory. *Designs, Codes and Cryptography*, 6:5–20, 1995.

[3] A. E. Brouwer, A. M. Cohen, and A. Neumaier. *Distance Regular Graphs.* Springer-Verlag, Berlin, 1989.

[4] Andries E. Brouwer. Small binary codes: Table of general binary codes, website at `http://www.win.tue.nl/~aeb/codes/binary-1.html`.

[5] Andries E. Brouwer. Small quaternary codes: Table of general quaternary codes, website at `http://www.win.tue.nl/~aeb/codes/quaternary-1.html`.

[6] Andries E. Brouwer. Small ternary codes: Table of general ternary codes, website at `http://www.win.tue.nl/~aeb/codes/ternary-1.html`.

[7] Richard A. Brualdi and Vera Pless. Greedy codes. *Journal of Combinatorial Theory(A)*, 64:10–30, 1993.

[8] Matthew C. Davey and David J.C. MacKay. Reliable communication over channels with insertions, deletions and substitutions. *IEEE Trans. Information Theory*, 47:687–698, 2001.

[9] L.Calabi and William E. Hartnett. A family of codes for the correction of substitution and synchronization errors. *IEEE Transactions on Information Theory*, 15:102–106, 1969.

[10] V.I. Levenshtein. Binary codes capable of correcting spurious insertions and deletions of ones. *Problems of Information Transmission*, 1:8–17, 1965.

[11] V.I. Levenshtein. On perfect codes in deletion and insertion metric. *Discrete Math. Appl.*, 2:241–258, 1992.

[12] Vladimir I. Levenshtein. Efficient reconstruction of sequences. *IEEE Transactions on Information Theory*, 47:2–22, 2001.

[13] A. Mahmoodi. Existence of perfect 3-deletion-correcting codes. *Designs, Codes and Cryptography*, 14:81–87, 1998.

[14] Patric R.J. Ostergard, Tsonka Baicheva, and Emil Kolev. Optimal binary error correcting codes of length 10 have 72 codewords. *IEEE Transactions on Information Theory*, 45:1229–1231, 1999.

[15] F. Qiu, T.J. Wen, D.A. Ashlock, and P.S. Schnable. Dna sequence-based bar-codes for tracking the origins of ests from a maize cdna library constructed using multiple mrna sources. *Plant Physiology*, 133:475–481, 2003.

[16] N.J.A. Sloane. On single-deletion-correcting codes, manuscript available online at `http://www.research.att.com/ njas/doc/dijen.pdf`.