IOWA STATE UNIVERSITY
Digital Repository

1-1-2003

Sphere packing bounds on edit metric codes

Jessie Katherine Campbell
Iowa State University

Sphere packing bounds on edit metric codes

by

Jessie Katherine Campbell

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Mathematics

Program of Study Committee:
Daniel Ashlock, Major Professor
Cliff Bergman
John Mayfield

Iowa State University

Ames, Iowa

2003

Graduate College
Iowa State University

This is to certify that the master's thesis of

Jessie Katherine Campbell

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. Introduction

This thesis will focus on the mathematics underlying a metric space being used in the development of an evolutionary algorithm. The algorithms being controlled by this evolutionary algorithm are greedy algorithms.

1.1 Greedy Algorithms

Mathematics often uses algorithms to solve problems. Many problems can be solved using *greedy algorithms*. A greedy algorithm is an algorithm in which a local measure of some sort chooses which option will yield the best immediate results. An example of a greedy algorithm is the following vertex coloring algorithm for combinatorial graphs.

Algorithm 1. *Input: A graph with an ordered vertex set, an ordered set of colors.*
Output: A coloring of the graph.
Algorithm: Examining the vertices in order, assign to the vertex the smallest color that is not already assigned to one of its neighbors.

The vertex coloring algorithm given here often yields sub-optimal results. The number of colors used by the greedy algorithm is not typically the chromatic number of the graph. An example of a greedy algorithm that produces optimal results is Kruskal's algorithm for producing a minimum spanning tree of a connected weighted graph (5; 8).

Algorithm 2. Kruskal's minimum spanning tree algorithm
Input: A weighted connected graph G.

Output: A minimum cost spanning tree H.

Algorithm: Examining the edges in order of nondecreasing weight, add an edge of G to E(H)

if it does not create a cycle.

1.2 Evolutionary Algorithms

So the natural question becomes, how can one modify a greedy algorithm to produce better, possibly optimal results? Before we attempt to answer that question, we need to define *evolutionary algorithm.* An evolutionary algorithm creates a population and then evaluates its fitness by some measure. The members with high fitness are copied and the copies are slightly varied, in a process similar to biological recombination, creating a new population. The process is then repeated. The process may or may not terminate, depending on the context of the problem. For example, if an evolutionary algorithm is used to find a maximum value, it will terminate, but if an evolutionary algorithm is used to find a strategy for playing a game like tic-tac-toe, it will not terminate (1). (6) shows that an evolutionary algorithm may be used to control a greedy algorithm. Using Conway's Lexicode algorithm as an example, (2) evolves the order in which words are considered.

Algorithm 3. Conway's Lexicode algorithm

Input: A minimum distance d under a specified metric 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 distance d or more from each word placed in C so far.

We will place this algorithm under evolutionary control in the following fashion. A set of words called a *seed* is initially chosen and Conway's algorithm extends the seed to complete a code. The fitness of a seed is the size of the code it creates. The evolutionary algorithm evolves the seeds to find more fit ones.

We can show that an evolutionary algorithm can be used to control Algorithm 1 to produce

optimal results. First we need a few definitions. A coloring is *optimal* if its vertices are colored by a set of colors with minimum cardinality. If a set of colors is ordered, then a *packed* coloring is one in which for each vertex V with color c, all colors $d < c$ appear on neighbors of V.

Theorem 1. *Every graph has an optimal packed coloring.*

Proof. Suppose we have an optimal coloring of a graph with a finite number of vertices and an ordered set of colors $C = \{1, 2, \ldots, n\}$. Without loss of generality, we may assume each color is used in the optimal coloring. If the coloring is packed we are done. Suppose the coloring is not packed. Choose a vertex that is a witness to the coloring being unpacked (i.e. such that every color smaller than it does not occur on a neighbor). Replace its color with the smallest color in C not appearing on a neighbor. This replacement leaves the coloring proper. The vertex is no longer a witness to the coloring being unpacked. If the coloring is now packed, we are done. If it is not, continue the process.

Suppose the process does not terminate. We would perform an infinite number of color changes on a finite number of vertices and the colors are positive integers in decreasing order. However, by the Well Ordering Principle (7), a set of positive integers (colors) has a lower bound. So the process not terminating on each vertex is a contradiction of the Well Ordering Principle. Therefore, since there are a finite number of vertices, the process must terminate. When the process terminates, the graph has a packed coloring. Since the number of colors cannot have increased, the coloring is still optimal. ∎

Because every graph has an optimal packed coloring, the evolutionary algorithm will eventually find the correct seed (initial set of vertices in this case) to arrive at an optimal packed coloring and hence yield optimal results. "Eventually" is quite a long time, so this technique is not practical for most graphs. A comparison of Conway's Lexicode Algorithm using the Hamming distance on binary words to the greedy evolutionary algorithm is given in Table 1.1.

An application of greedy evolutionary algorithms that is practical is to DNA barcodes. When DNA libraries are formed, a number of different tissues from an organism are used. To identify the source tissue, short stretches of DNA, called DNA barcodes, are embedded into the tissue

Table 1.1   Comparison of using Conway's Lexicode Algorithm or the greedy
fitness evolutionary algorithm for length n, distance d codes

n	d	Basic Lexicode	Evolutionary Algorithm
16	7	32	32
18	7	128	128
18	9	8	20
19	9	16	40
19	11	4	6

libraries. Tissue identifying is then simply a matter of finding the barcode and matching it to the correct source tissue. A potential problem with this method lies in the sequencing of the DNA. Sequencers often miscall, ignore or duplicate bases. To correct this problem, one can employ an error correcting code.

A code of length n is simply a collection of strings, each n characters long, called *codewords*. A *k-error correcting code* is a code in which up to k errors in a codeword can be corrected (3). This is done by choosing an appropriate measure of distance between codewords and using only those codewords that are sufficiently far apart ($d = 2k + 1$) under this metric. The natural choice to measure distance for DNA barcodes is the edit distance over the set of bases {C, G, A, T}. The *edit distance* between two strings is the least number of single character insertions, deletions, and substitutions to transform one string into the other. Two codewords that are edit distance k from each other are called *k-edit neighbors*. Notice Conway's algorithm can be re-specialized to the edit distance. The edit metric version of this algorithm can be used to find a lower bound on the size of edit codes, see Table 1.2 for binary examples.

An examination of the structure of the edit metric space suggests that an improvement to the greedy closure evolutionary algorithm for finding a k-error correcting code of length n can be made by starting with codewords with the smallest number of k-edit neighbors and continuing from there. This amounts to replacing the lexical order in Conway's algorithm with a potentially more efficient ordering. This will hopefully yield a larger number of words in the code.

Table 1.2   Lower bounds on the size of a maximal size binary edit code with length n and minimum distance d between words

n	d 3	4	5	6	7	8	9	10
4	2	2	-	-	-	-	-	-
5	4	2	2	-	-	-	-	-
6	5	4	2	2	-	-	-	-
7	10	5	2	2	2	-	-	-
8	15	9	4	2	2	2	-	-
9	28	10	4	4	2	2	2	-
10	46	19	5	4	2	2	2	2
11	84	26	8	5	4	2	2	2
12	150	43	12	7	4	4	2	2
13	268	71	19	10	5	4	2	2
14	478	117	29	13	7	5	4	2

CHAPTER 2.   Edit Metric on the Binary Alphabet

2.1   Edit Metric versus Hamming Metric

We begin by considering a binary alphabet. The *edit distance* between two words over this alphabet is the minimum number of edit operations needed to change one word into the other, where an edit operation is a substitution, insertion, or deletion of a single character. To show that edit distance is actually a metric, we prove a general result.

Theorem 2. *Let S be a set with a finite number of unary operations $u_1, u_2, \ldots, u_m$. Then the minimal number of operations to turn x into y is a metric on S.*

Proof. Construct a graph with vertex set S and edge set $\{(x, u_i(x)) : x \in S \text{ and } i = 1, 2, \ldots, m\}$. The minimum number of operations to turn x into y is simply the minimum number of edges needed for a path from x to y. But the minimal path distance is a metric, see (8). ■

Corollary 1. *The edit distance on a finite alphabet is a metric.*

Proof. This is a special case of Theorem 2, where S is the set of all words comprised of characters in the given alphabet and the unary operations are substitution of a single character, deletion of a single character, or insertion of a single character. ■

Another important distance measure for strings is the *Hamming distance*. The Hamming distance between two words is the number of substitutions needed to transform one word into the other. The binary alphabet with the Hamming distance also forms a metric space. We now elucidate the structure of the edit metric space over the binary alphabet.

First some definitions are needed. A *block* of a word is a locally maximal substring comprised of exactly one character. For 10011100 there are 4 blocks: 1, 00, 111, and 00. The *block*

representation of a word is a sequence of numbers representing how many times a character repeats. For example, $1, 2, 3, 2$ denotes either 10011100 or 01100011. Notice the representation does not represent a unique word. The number of words with the same block representation depends on the size of the alphabet and the number of blocks.

Lemma 1. *If the size of the alphabet is A and the block representation has k blocks then there are $A(A-1)^{k-1}$ words with that block representation.*

Proof. There are A choices for the first block and $A-1$ choices for each block after that. Since there are k blocks, we have $A(A-1)^{k-1}$. ∎

For binary strings, each block representation corresponds to $2(1)^{k-1} = 2$ words. The length of the word is the sum of the numbers of the block representation. For our example the length of the word represented by $1, 2, 3, 2$ is $1 + 2 + 3 + 2 = 8$.

2.2   Structure of Edit Graph

The edit graph has a complex structure. For any given n, the subgraph that considers only substitutions is the Hamming graph. The structure of this subgraph is known to be an n-hypercube, where the vertices are words of length n and edges connect words that differ in exactly one position. To recover the edit graph, we let $n = 0, 1, 2, \ldots$ and stack the hypercubes with the empty string λ at the top, and continue down in increasing length n. Then connect the hypercubes by connecting vertices that differ by a deletion or insertion. Notice that each level of the stack of hypercubes is only connected to the level directly above it and the level directly below it. We end up with a pyramid of hypercubes with an extensive network of edges between hypercubes adjacent to each other in the stack. Figure 2.1 shows the top portion of the graph.

Figure 2.1   The top "levels" of the edit metric on the binary alphabet shown as a graph with distance-one edges. Recall that λ denotes the empty string.
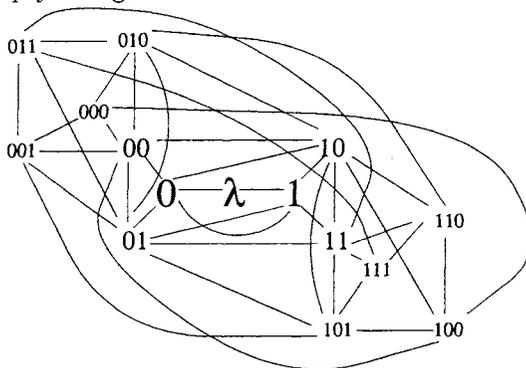
2.3   Spheres of Radius 1

Using the greedy evolutionary algorithm, the size of the code we end up with depends on the order in which words are considered. If we begin with a word that has many k-edit neighbors, we eliminate all of those words from our code. It makes sense to consider words with the fewest number of k-edit neighbors first to pack the initial portion of the code more efficiently. To do this we consider sphere sizes on the edit metric over the binary alphabet. For a sphere of radius one, this is not difficult.

Before we begin, we need a few definitions. We say we *lengthen* a block if we insert a character in a block that changes only the length of the block and does not add any new blocks. For example, if $W = 01100$, $W' = 011100$ is found by lengthening the second block of W. We say we *split* a block if we insert a character within a block that changes the number of blocks in the word. For example, if $W = 00111$, $W' = 010111$ is found by splitting the first block of W. Lastly, we say we add *end extensions* if we create a new first or last block of size one.

Theorem 3. *Let W be a word of length n with k blocks in its block representation. W has k 1-edit neighbors of length $n - 1$, n 1-edit neighbors of length n, and $n + 2$ 1-edit neighbors of length $n + 1$.*

Proof. The only way to go from length n to length $n - 1$ in one edit operation is to delete a

character. Deletions at different positions within a block result in the same word (for example, if W=110001 we can delete the third, fourth or fifth character to arrive at 11001). Therefore, the number of blocks determines all possible 1-edit neighbors of length $n - 1$.

The only way to stay length n with one edit operation is to substitute one character for another. There are n possible ways to do this for a word of length n.

The only way to go from length n to length $n + 1$ in one edit operation is to insert a character. This can be done in three ways. First, we can add an end extension on the left or on the right (for example, we can insert a 1 on the left end of 001100011 to get 1001100011, changing the block representation from 2,2,3,2 to 1,2,2,3,2). This gives us two 1-edit neighbors. Second, we can lengthen an existing block (example: we can insert a 1 after the third, fourth or fifth position of 001110 to get 0011110). This can be done k ways. Lastly, we can split an existing block (example: we can insert 0 after the third position of 011100 to get 0110100 changing the block representation from $1, 3, 2$ to $1, 2, 1, 1, 2$). There are $n - 1$ places to insert a character, but we will not insert a character at a block boundary (between two blocks) because this yields the same result as lengthening a block. There are $k - 1$ block boundaries, so there are $(n - 1) - (k - 1) = n - k$ ways to obtain a new word by splitting blocks. This results in a total of $2 + k + n - k = n + 2$ 1-edit neighbors of length $n + 1$. ∎

Corollary 2. *The number of 1-edit neighbors of a word of length n with k blocks is $2n + k + 2$.*

Proof. From Theorem 3, we see that the number of 1-edit neighbors of a word of length n with k blocks in its block representation is $k + n + n + 2 = 2n + k + 2$. ∎

This gives us our next result.

Corollary 3. *Let W be a word of length n. The lower bound on the number of 1-edit neighbors of W is $2n + 3$ and the upper bound on the number of 1-edit neighbors of W is $3n + 2$.*

Proof. The number of 1-edit neighbors is $2n + k + 2$ by Corollary 2. Since we always have at

Table 2.1   Sphere packing bounds for 1-error correcting edit codes with code words of length n

n	Code Size
1	1
2	1
3	2
4	3
5	5
6	9
7	16
8	28
9	51
10	93

least one block and never more than n blocks, we have

$$1 \leq k \leq n$$

$$2n + 1 \leq 2n + k \leq 3n$$

$$2n + 3 \leq 2n + k + 2 \leq 3n + 2. \quad \blacksquare$$

The total number of 1-neighbors is useful in providing intuition about how to write the evolutionary algorithm. In practice, for 1-error correcting codes, we are actually only concerned with 1-neighbors that have the same length as the original word. To find the sphere packing bound for a 1-error correcting code where we allow the codewords to have length n only, we find the classic sphere packing bound given in (4)

$$|M| \leq \frac{2^n}{\sum_{i=0}^{1} \binom{n}{i} (2 - 1)^i} = \frac{2^n}{n + 1}$$

where $|M|$ is the maximum size of the code. Table 2.1 gives the sphere packing bounds for 1-error correcting codes with words of length $n \leq 10$.

2.4   Spheres of Radius 2

We first wish to count the 2-edit neighbors of a given word W that have the same length as W. We need to consider words we can obtain from W by two substitutions, or an insertion followed by a deletion, or a deletion followed by an insertion.

Theorem 4. *The set of 2-edit neighbors of a word W found by first deleting a character and then inserting a character is the same as that found by first inserting a character and then deleting a character.*

Proof. Let $W = x_1 x_2 \ldots x_n$ be a word of length n where $x_i \in \{0, 1\}$. We wish to delete at position i and insert to the left of position j. If $i = j$ we insert a character and then delete it, arriving back at W or we delete a character and then insert a character arriving back at W or at a word that differs from W in only one position. In either case, the word we arrive at is edit distance 0 or edit distance 1 and hence not a 2-edit neighbor of W. So it suffices to consider $i \neq j$.

By considering reflections, without loss of generality, $i < j$. First we will consider deletion followed by insertion. Let W_1' be the word obtained from W by deleting at position i. Then

$$W_1' = x_1 x_2 \ldots x_{i-1} x_{i+1} \ldots x_n.$$

Now we insert a character z to the right of character x_j to arrive at

$$W_1'' = x_1 x_2 \ldots x_{i-1} x_{i+1} \ldots x_j z x_{j+1} \ldots x_n.$$

Now we consider insertion followed by deletion. Let W_2' be the word obtained from W by

inserting z to the right of character x_j. Then

$$W_2' = x_1 x_2 \ldots x_j z x_{j+1} \ldots x_n.$$

Now delete character x_i to arrive at

$$W_2'' = x_1 x_2 \ldots x_{i-1} x_{i+1} \ldots x_j z x_{j+1} \ldots x_n.$$

Since $W_1'' = W_2''$ we see that the order in which the insertion and deletion are performed does not matter. ∎

This result means we only need consider the 2-edit neighbors of a word found by two substitutions or a deletion followed by an insertion, which we will call a *del-in*. A *del-in event* is the ordered pair that gives the specific deletion and insertion used to arrive at a del-in. We make this distinction because distinct del-in events can lead to the same del-in. Figure 2.2 shows examples of del-ins events. Notice that all of the action occurs between the point of insertion and the point of deletion. The initial and terminal segments of the string remain anchored, which severely limits the amount of "frame shifting" allowed.

Observe that the number of words found by substituting d characters in a word W of length n is $\binom{n}{d}$ since there are d positions we want to change and n positions to choose from. So the number of 2-edit neighbors found by two substitutions is $\binom{n}{2}$.

Finding the number of del-ins is a bit more difficult.

Lemma 2. *Let W be a word of length n with k blocks. The number of ways to delete a character and then insert a character to arrive at a word $W' \neq W$ is nk.*

Proof. From Theorem 3 we know there are k ways to delete a character from a word. Now we have a word of length $n - 1$. Also by Theorem 3, we now have $(n - 1) + 2 = n + 1$ ways to insert a character in the shortened word to arrive back at a word of length n. However, one of these words will be the original word, so there are n ways to insert and arrive at a new word.

Figure 2.2 Some examples of del-in events and the corresponding del-ins.
(a), (b) and (c) show del-ins that occur in only one way. (d)
and (e) each show two del-in events that lead to the same del-in.
Note that the insertion block refers to the block number in the
original string.

00001110101011110100001 1100

↓ Delete (block 5)

00001110110111101000011100

↓ Lengthen (block 11)

00001110110111101000001 1100

(a)

00001110101011110100001 1100

↓ Delete (block 8)

00001110101011101000011100

↓ Split (block 11)

0000111010101110100100011100

(b)

00001110101011110100001 1100

↓ Delete (block 1)

0001110101011110100001 1100

↓ Split (block 1)

0100111010101110100001 1100

(c)

00001110101011110100001 1100

Delete (block 6) ╱   ╲ Delete (block 5)

000011101001111010000l 1100   00001110110111101000011100

Lengthen (block 4) ╲   ╱ Lengthen (block 7)

00001110110011110100001 1100

(d)

00001110101011110100001 1100

Delete (block 12) ╱   ╲ Delete (block13)

0000111010101111010l00001100   00001110101011110100001110

Split (block 13) ╲   ╱ Split (block12)

00001110101011110100001 1010

(e)

Since there are k ways to delete and n ways to insert, there are kn ways to delete and then insert and not arrive back at the original word. ∎

This formula overcounts the number of del-ins because there are some words that can be obtained by two distinct del-in events. To find the amount of overcounting, we first need to consider certain special strings.

Definition 1. *An* alternating string *is a string comprised of exactly two distinct characters in which each character is different from the characters adjacent to it.*

Note that in the binary case, an alternating string is simply a word of length n such that $k = n$.

Lemma 3. *If W is an alternating string of length n, then any word $W' \neq W$ obtained from W by a del-in event can be found by a del-in event in exactly two ways if the deletion and insertion occur in non-adjacent blocks and exactly one way if the deletion and insertion occur in adjacent blocks.*

Proof. Let W be an alternating string of length n. First, let W' be found from W by a del-in event in non-adjacent blocks. Then there are four possibilities:

1. W' is found from W by deleting block i and lengthening block j where $i = 2, \ldots, n-1$ and $j = 1, \ldots, n$ with $|i - j| \geq 2$.

2. W' is found from W by deleting block $i = 1$ or $i = n$ and lengthening block j where $j = 1, \ldots, n$ with $|i - j| \geq 2$.

3. W' is found from W by deleting block i and end extending where $i = 2, \ldots, n-1$.

4. W' is found from W by deleting block $i = 1$ (respectively $i = n$) and end extending on the right (resp. left).

Case 1: Suppose W' is found from W by deleting block $i = 2, \ldots, n-1$ and lengthening block j where $|i - j| \geq 2$. If $i < j$, we have $W' = x_1 \ldots x_{i-1} x_{i+1} \ldots x_{j-1} x_j y x_{j+1} \ldots x_n$ where $y = x_j$. Now suppose W' can also be found from W by deleting block p and lengthening block q where

$|p - q| \geq 2$.

If $p < q$, $W' = x_1 \ldots x_{p-1} x_{p+1} \ldots x_{q-1} x_q z x_{q+1} \ldots x_n$, where $z = x_q$. If $p - 1 > i - 1$, we have $x_{i-1} \neq x_{i+1}$, which contradicts the fact that W is an alternating string. If $p - 1 < i - 1$, we have $x_{p-1} \neq x_{p+1}$, which also contradicts the fact that W is an alternating string. So $p - 1 = i - 1$ and we have $p = i$. If $q < j$, we have $x_q \neq z$, which is a contradiction. If $q > j$, we have $x_j \neq y$, which is also a contradiction. So $q = j$. Therefore, if $p < q$, $p = i$ and $q = j$.

If $q < p$, then $W' = x_1 \ldots x_{q-1} x_q z x_{q+1} \ldots x_{p-1} x_{p+1} \ldots x_n$. Let $W' = u_1 \ldots u_n = v_1 \ldots v_n$ where

$$u_m = \begin{cases} x_m & m = 1, \ldots, i-1 \\ x_{m+1} & m = i, \ldots, j-1 \\ x_m & m = j, \ldots, n \end{cases}$$

and

$$v_m = \begin{cases} x_m & m = 1, \ldots, q \\ x_{m-1} & m = q+1, \ldots, p \\ x_m & m = p+1, \ldots, n \end{cases}$$

If $q < i - 1$, then $x_q = v_{q+1} = u_{q+1} = x_{q+1}$. But $x_{q+1} \neq x_q$, so we have a contradiction. If $i - 1 < q < j - 1$, then $x_q = v_q = u_q = x_{q+1}$. But $x_q \neq x_{q+1}$, which gives a contradiction. If $j - 1 < q$, then $x_q = v_{q+1} = u_{q+1} = x_{q+1}$. But $x_q \neq x_{q+1}$, so we have another contradiction. If $q = j - 1$, then $x_{j-1} = x_q = v_q = u_q = u_{j-1} = x_j$. But $x_j \neq x_{j-1}$, and we arrive at yet another contradiction. This gives us $q = i - 1$. If $p < i$, then $x_p = u_p = v_p = x_{p-1}$. But $x_p \neq x_{p-1}$, so we have a contradiction. If $i < p < j - 1$, then $x_{p+2} = u_{p+1} = v_{p+1} = x_{p+1}$. But $x_{p+2} \neq x_{p+1}$, which gives a contradiction. If $p > j - 1$, then $x_p = u_p = v_p = x_{p-1}$. But $x_{p-1} \neq x_p$, so we have a contradiction. If $p = i$, then we will have $p = q + 1$ since $q = i - 1$. But this will give us $|p - q| = 1 < 2$, which contradicts the fact that $|p - q| \geq 2$. This gives us $p = j - 1$. Therefore if $q < p$, then $p = j - 1$ and $q = i - 1$. In other words, if W' is found from W by deleting block $i = 2, \ldots, n - 1$ and lengthening block j, then it is also found by deleting block $j - 1$ and lengthening block $i - 1$.

A symmetric argument shows that if $j < i$ then either $p = i$ and $q = j$ or $p = j + 1$ and

$q = i + 1$. So in this case, W' can be found from W by two distinct del-in events.

<u>Case 2</u>: Suppose W' is found from W by deleting block $i = 1$ or $i = n$ and lengthening block j where $|i - j| \geq 2$. If $i = 1$, then $W' = x_2 \ldots x_{j-1} x_j z x_{j+1} \ldots x_n$. Since $W = x_1 \ldots x_n$ and $x_1 \neq x_2$, the only other possible way to get from W to W' is to end extend on the left and delete at some block p. So we also have $W' = x_0 x_1 \ldots x_{p-1} x_{p+1} \ldots x_n$ where $x_0 = x_2$. Let $W' = u_1 \ldots u_n = v_1 \ldots v_n$ where

$$u_m = \begin{cases} x_{m+1} & m = 1, \ldots, j-1 \\ x_m & m = j, \ldots, n \end{cases}$$

and

$$v_m = \begin{cases} x_{m-1} & m = 1, \ldots, p \\ x_m & m = p+1, \ldots, n \end{cases}$$

If $p + 1 < j$, then $v_{p+1} = u_{p+1} = x_{p+2} \neq x_{p+1}$. But $v_{p+1} = x_{p+1}$, so we have a contradiction. If $p + 1 > j$, then $u_j = v_j = x_{j-1} \neq x_j$. But $u_j = x_j$, which gives a contradiction. So $p = j - 1$. A symmetric argument shows if $i = n$, then W' can also be found from W by deleting block $p = j + 1$ and end extending on the right.

<u>Case 3</u>: Suppose W' is formed from W by deleting block $i = 2, \ldots, n-1$ and end extending. If we end extend on the left, then $W' = x_0 x_1 \ldots x_{i-1} x_{i+1} \ldots x_n$ where $x_0 = x_2$. The only other possible way to get from W to W' is to delete the first character and lengthen block q. So we also have $W' = x_2 \ldots x_{p-1} x_p z x_{p+1} \ldots x_n$ where $z = x_p$. Let $W' = u_1 \ldots u_n = v_1 \ldots v_n$ where

$$u_m = \begin{cases} x_{m-1} & m = 1, \ldots, i \\ x_m & m = i+1, \ldots, n \end{cases}$$

and

$$v_m = \begin{cases} x_{m+1} & m = 1, \ldots, q-1 \\ x_m & m = q, \ldots, n \end{cases}$$

If $i < q - 1$, then $u_{i+1} = v_{i+1} = x_{i+2} \neq x_{x+1}$. But $u_{i+1} = x_{i+1}$, so we have a contradiction. If $i > q - 1$, then $v_q = u_q = x_{q-1} \neq x_q$. But $v_q = x_q$, which is a contradiction. So $q = i + 1$.

A symmetric argument shows if we end extend on the right, then deleting block n and lengthening block $q = i - 1$ is the same.

Case 4: Suppose W' is found from W by deleting block $i = 1$ and end extending on the right. Then $W' = x_2 \ldots x_n x_{n+1}$ where $x_{n+1} = x_{n-1}$. The only other way to get from W to W' by a del-in event is to end extend on the left and delete block n. A symmetric argument applies for W' found from W by deleting block $i = n$ and end extending on the left.

In all four cases, if W' is found from W by deleting block i and lengthening block j or end extending, W' can also be found by deleting block $p \neq i$ and lengthening block $q \neq j$ or end extending. Moreover, p and q or the end extension are completely determined by the choice of i and j or the end extension. So each of the four cases leads to two distinct del-in events. Notice that in these four cases W' has one of three forms: it is either an alternating string with two blocks of length two inserted, an alternating string with one block of length two inserted or an alternating string.

Now suppose W' is found from W by a del-in event in adjacent blocks. There are three possibilities: the deletion occurs in block $i = 2, \ldots, n - 1$, the deletion occurs in block $i = 1$, or the deletion occurs in block $i = n$. If the deletion occurs in block $i = 2, \ldots, n - 1$, we fuse the two adjacent blocks into one block, so there is no distinction made between inserting in block $i - 1$ or $i + 1$ since they actually are now one block. This gives us an alternating string with a block of length three inserted. Any other possible del-in would have to occur in non-adjacent blocks, but we already noted that if we have a del-in event in non-adjacent blocks we will not get a block of length three, only one or two blocks of length two. If the deletion occurs in $i = 1$, the lengthening must be in block 2, so we arrive at a string with a block of length two that is comprised of characters different from the first character of W, followed by an alternating string. There is no other way to perform a del-in event that has a block of length two at the beginning that is comprised of characters different from the first character of W. A symmetric argument applies if we delete from n and lengthen $n - 1$. So in any of the three possibilities, we have only one del-in event that transforms W into W'. ∎

Theorem 5. *Let W be an alternating string of length n. If we count all possible ways to delete*

and then insert, we overcount the number of del-ins by $\binom{n}{2}$.

Proof. By Lemma 3, all del-ins are counted either once or twice. The del-ins found by deleting block i and lengthening block j (consider an end extension as a lengthening of block 0 or block $n+1$) where $|i-j| \geq 2$ are counted twice. Consider if $i < j$, we can also find the same del-in by deleting block $j-1$ and lengthening block $i-1$. Now, if $i > j$, we can also find the same del-in by deleting block $j+1$ and lengthening block $i+1$. But if $i > j$, the other del-in event that deletes block $j+1$ and lengthens block $i+1$ has already been considered because $j+1 < i+1$. So to find the overcounting we only need to count the pairs of pairs $(i,j), (j-1, i-1)$, which amounts to counting pairs of the first entries $(i, j-1)$ where $i > j$. Notice this is equivalent to counting unordered pairs (i,j) which we see totals $\binom{n}{2}$. ∎

Definition 2. *Let W be a word of length n. A* locally maximal alternating substring (LMAS) *is a substring of W that is an alternating string contained in no other alternating string that is a substring of W. Define a_i to be the number of locally maximal alternating substrings of W of length $2 \leq i \leq n$.*

We say a block is non-trivial if it is a block of length $n_i > 1$.

Definition 3. *The L-B decomposition of a string is a segmentation into adjacent substrings that are either locally maximal alternating substrings or locally maximal non-trivial blocks. These are called the* elements *of the L-B decomposition.*

It is easy to see that every string has a unique L-B decomposition.

Theorem 6. *If $W' \neq W$ is obtained from W by a del-in event, then this may be done in only one way unless the deletion and insertion occur within a single element that is a locally maximal alternating substring (LMAS) or the deletion occurs on the boundary between two adjacent blocks and the insertion occurs one character into one of the two blocks, in which case there are exactly two ways.*

Proof. Suppose the deletion occurs in a LMAS but the insertion does not, and the insertion does not occur on the boundary between two adjacent blocks (see Figure 2.2(a)). Then the LMAS becomes one character shorter with a block of length two somewhere within it. To arrive at that same pattern in a different way, we would need to insert in the LMAS to get a block of length two, but then it would be one character longer and hence not the same as if we deleted from it. So there is only one way to arrive at a del-in found by deleting in a LMAS and inserting elsewhere, where neither the deletion nor insertion occur on a boundary between elements.

If the deletion occurs in a non-trivial block but the insertion does not, and neither the insertion nor deletion occur on the boundary between two adjacent blocks, the block becomes one character shorter and there is no other way to accomplish this (see Figure 2.2(b)). So there is only one way to arrive at a del-in found by deleting in a non-trivial block and inserting elsewhere, where neither the deletion nor insertion occur on the boundary between two adjacent blocks.

By Lemma 3, if the deletion and insertion occur within a LMAS, there are exactly two del-in events that yield the same del-in. An example of this can be seen in Figure 2.2(d).

Suppose the deletion and insertion occur within a non-trivial block (see Figure 2.2(c)). There is only one way to accomplish this del-in since all the deletions within the block are equivalent and all the insertions yield distinct del-ins.

Suppose the deletion and insertion occur on the boundary between two adjacent blocks (see Figure 2.2(e)). We can delete from the left block and split the right block after its first character if it is of length two or more. If the right block is of length one, we lengthen the block to its right. This yields the same del-in we get if we delete from the right block and split the left block between its last two characters if it is of length two or more or lengthen the block to the left of it if it is of length one. So there are exactly two del-in events that arrive at the same del-in. Notice that a deletion and insertion on the boundary of two adjacent blocks occurs either within a LMAS or on the boundary of two adjacent elements.

So all del-in paths lead to distinct del-ins unless they occur within a LMAS or on the boundary

of two adjacent blocks.   ■

Theorem 7. *If W is a word of length n with k blocks then the number of del-ins of W of length n is*

$$nk - k + 1 - \sum_{i=2}^{n} a_i \left(\binom{i}{2} - (i-1) \right),$$

where a_i is the number of LMAS of W of length i.

Proof. The number of ways to delete and then insert is nk by Lemma 2. But by Theorem 6, this counts del-ins that can be found by deleting and inserting within a LMAS or on a boundary between two adjacent blocks twice. Since there are $k-1$ block boundaries, we need to subtract $k-1$ from our total number of ways to delete and then insert. There are $\sum_{i=2}^{n} a_i$ LMAS's and the amount of overcounting that occurs in each LMAS is $\binom{i}{2}$. So the total amount of overcounting due to a del-in event within a LMAS is $\sum_{i=2}^{n} a_i \binom{i}{2}$. But block boundaries occur within a LMAS as well as between adjacent elements. Since we have now subtracted them twice, we need to add them back in. There are $i-1$ block boundaries in an LMAS of length i, so we arrive at

$$nk - (k-1) - \sum_{i=2}^{n} a_i \binom{i}{2} + \sum_{i=2}^{n} a_i(i-1) = nk - k + 1 - \sum_{i=2}^{n} a_i \left(\binom{i}{2} - (i-1) \right). \quad ■$$

In order to finish counting the number of 2-edit neighbors, we also need to exclude any del-in events that can be accomplished by a one character substitution, since these are 1-edit neighbors and are therefore, by definition, not 2-edit neighbors.

Lemma 4. *Given a word W, every word W′ found by substituting one character in W can also be found by a del-in event.*

Proof. Suppose we substitute the character in position j. We could also delete the character at position j and insert the other character in that position.   ■

Notice that a word found from W by one substitution can also be found from W by a del-in event in exactly one way. Since the number of words of length n found by substituting one character is $\binom{n}{1} = n$, there are n del-in events that are edit distance 1 (and hence not edit distance 2).

We also need to be concerned if a word can be found by making two substitutions and by a del-in event. Let n_i be the length of the ith block.

Theorem 8. *The number of 2-edit neighbors of a word that can be found by both a del-in event and by substituting two characters is $2n - n_1 - n_k - k + 1$.*

Proof. Let W be a word of length n and W' found from W by a deletion followed by an insertion. Notice if we delete from block i and insert in block j, we can say something about the possible Hamming distance between W and W'. Deleting in block i causes all of the blocks between block i and j to shift to the left or right one character. All of the block boundaries between block i and j will contribute one to the Hamming distance between W and W'. If the insertion in block j is a lengthening, it will not contribute any more to the Hamming distance. However, if the insertion in block j is a split, it will contribute one more to the Hamming distance. So the Hamming distance between W and W' is either $|i - j|$ if the insertion is a lengthening or $|i - j| + 1$ if the insertion is a split.

There are three types of del-in events that produce words that are also found by two substitutions:

1. delete from block i and lengthen block j where $|i - j| = 2$, $1 \leq i, j \leq k$.

2. delete from block i and split block j, where $|i - j| = 1$, $1 \leq i, j \leq k$.

3. delete from block 2 (respectively $k - 1$) and add an end extension on the left (resp. right).

<u>Case 1</u>: To delete from block i and lengthen block j with

$$i < j \quad \text{we let } i = 1, \ldots, k-2; \; j = 3, \ldots, k$$
$$i > j \quad \text{we let } i = 3, \ldots, k; \; j = 1, \ldots, k-2$$

There is only one way to lengthen block j so we have

$$([(k-2)-1]+1) + [(k-3)+1] = 2k-4$$

ways to do this.

<u>Case 2</u>: To delete from block i and split block j with

$$i < j \quad \text{we let } i = 1, \ldots, k-1; \; j = 2, \ldots, k$$
$$i > j \quad \text{we let } i = 2, \ldots, k; \; j = 1, \ldots, k-1$$

The number of ways to split block j is $n_j - 1$, so the number of ways to delete from a block and split a block 1 block away is

$$
\begin{aligned}
\sum_{j=2}^{k}(n_j - 1) + \sum_{j=1}^{k-1}(n_j - 1) &= \left(2\sum_{j=1}^{k}(n_j - 1)\right) - (n_1 - 1) - (n_k - 1) \\
&= 2\sum_{j=1}^{k}n_j - 2\sum_{j=1}^{k}1 - n_1 + 1 - n_k + 1 \\
&= 2n - 2k - n_1 - n_k + 2.
\end{aligned}
$$

But this overcounts insertions made at the block boundaries. For example, if we consider $W = 000111001$, we can delete from block 2 and split block 1 OR delete from block 1 and split block 2 to arrive at $W' = 001011001$. Every delete-split event at a block boundary is counted twice, so the number of unique del-ins found by a delete-split is

$$2n - 2k - n_1 - n_k + 2 - (k-1) = 2n - 3k - n_1 - n_k + 3$$

since the number of block boundaries is $k - 1$.

Case 3: There are only two possibilities here.

So the total number of 2-edit neighbors found by both a del-in event and two substitutions is

$$2k - 4 + 2n - 3k - n_1 - n_k + 3 + 2 = 2n - n_1 - n_k - k + 1. \qquad \blacksquare$$

Putting all these results together, we arrive at a formula for the number of 2-edit neighbors.

Theorem 9. *If W is a word of length n with k blocks, then the number of 2-edit neighbors of W is*

$$\binom{n}{2} + n(k - 3) + n_1 + n_k - \sum_{i=2}^{n} a_i \left(\binom{i}{2} - i + 1 \right),$$

where a_i is the number of LMAS in W of length i.

Proof. The number of 2-edit neighbors of a word is the number of words found by two substitutions together with the del-ins that are not also 1-edit neighbors. But some del-ins are also found by substitution, so we need to subtract them. By Theorem 7, Theorem 4 and Theorem 8, we have

$$\binom{n}{2} + \left[nk - k + 1 - \sum_{i=2}^{n} a_i \left(\binom{i}{2} - (i - 1) \right) - n \right] - (2n - n_1 - n_k - k + 1)$$

$$= \binom{n}{2} + nk - k + 1 - \sum_{i=2}^{n} a_i \left(\binom{i}{2} - i + 1 \right) - n - 2n + n_1 + n_k + k - 1$$

$$= \binom{n}{2} + nk - 3n - \sum_{i=2}^{n} a_i \left(\binom{i}{2} - i + 1 \right) + n_1 + n_k$$

$$= \binom{n}{2} + n(k - 3) + n_1 + n_k - \sum_{i=2}^{n} a_i \left(\binom{i}{2} - i + 1 \right). \qquad \blacksquare$$

Finding bounds on the 2 spheres is not very satisfying.

Lemma 5. *Let W be a word of length n. Let $|S_2|$ be the number of 2-edit neighbors $W' \neq W$ of length n. Then*

$$\binom{n}{2} \leq |S_2|$$

and equality holds if $k = 1$ where k is the number of blocks.

Proof. Every word at Hamming distance 2 from W is also a 2-edit neighbor of W (if it was not a 2-edit neighbor, it would have to be a 1-edit neighbor, but since it must be of length n, that would make it Hamming distance 1 which contradicts being Hamming distance 2). If $k = 1$, notice that $n_1 = n_k = n$ and that $a_i = 0$ for $i = 2, \ldots, n$. Using Theorem 9 we see the number of 2-edit neighbors is

$$
\begin{aligned}
\binom{n}{2} + n(k - 3) &+ n_1 + n_k - \sum_{i=2}^{n} a_i \left(\binom{i}{2} - i + 1 \right) \\
&= \binom{n}{2} + n(1 - 3) + n + n - \sum_{i=2}^{n} (0) \left(\binom{i}{2} - i + 1 \right) \\
&= \binom{n}{2} + n(-2) + 2n \\
&= \binom{n}{2}.
\end{aligned}
$$
∎

This shows that the obvious sphere packing bound for spheres of radius 2 is no better than that of the Hamming code. The number of words for which equality in Lemma 5 holds should be small, so it would be a good next step to examine all possible cases where equality holds and exclude them to see if we can find a better bound for that subset of the edit code.

CHAPTER 3.   Generalizing to {A,C,G,T}

We would like to generalize our results about the edit metric on binary strings to quaternary strings, more specifically to strings of characters from the alphabet $\{A, C, G, T\}$ because of potential biotech applications. Some of the results follow immediately, while others must take into consideration the increased number of characters available for operations.

We will still use block representation in the same manner. The block representation of a string of $\{A, C, G, T\}$ is a sequence of numbers representing how many times a character repeats. So the block representation of $AAACCCCCCTAGG$ is $3, 6, 1, 1, 2$. Notice there are many words corresponding to each block representation. Another possible string with block representation $3, 6, 1, 1, 2$ is $CCCGGGGGGCTAA$. By Lemma 1, we see there are $4(3^{k-1})$ strings with a given block representation, where k is the number of blocks in the block representation.

3.1   Spheres of Radius 1

Theorem 10. *Let W be a word of length n with k blocks in its block representation. W has k 1-edit neighbors of length $n - 1$, $3n$ 1-edit neighbors of length n, and $3n + 4$ 1-edit neighbors of length $n + 1$.*

Proof. The number of ways to delete remains the same as in the binary case because deleting characters does not depend on the size of the alphabet. The number of ways to substitute is multiplied by three because the number of sites of substitution is the same, but we now have three characters to choose from to substitute.

The number of ways to insert is similar. We can add a new block of length one on the end. There are three characters that will create a block of length one on each end, so there are six

possibilities here. If we want to lengthen an existing block, there are still only k ways to do this. If we wish to split a block, there are $(n-1)$ places we can insert a character. But inserting a character on a block boundary that is the same as one of the two blocks that creates the boundary is the same as lengthening one of the blocks. So there are only two insertions allowed on block boundaries. So we have three possible characters to insert at $(n-1)-(k-1) = n-k$ positions and two possible characters to insert at $k-1$ positions, since there are $k-1$ block boundaries. This gives a total of

$$6 + k + 3(n-k) + 2(k-1) = 3n + 4. \qquad \blacksquare$$

Corollary 4. *The number of 1-edit neighbors of a word of length n with k blocks is $6n + k + 4$.*

Proof. From Theorem 10, we see that the number of 1-edit neighbors of a word of length n with k blocks in its block representation is $k + 3n + 3n + 4 = 6n + k + 4$. $\qquad \blacksquare$

Corollary 5. *Let W be a word of length n. The lower bound on the number of 1-edit neighbors of W is $6n + 5$ and the upper bound on the number of 1-edit neighbors of W is $7n + 4$.*

Proof. The number of 1-edit neighbors is $6n + k + 4$ by Corollary 4. Since we always have at least one block and we never have more than n blocks, we have

$$
\begin{array}{ccccc}
1 & \leq & k & \leq & n \\
6n + 1 & \leq & 6n + k & \leq & 7n \\
6n + 5 & \leq & 6n + k + 4 & \leq & 7n + 4. \quad \blacksquare
\end{array}
$$

As with binary words, for 1-error correcting codes, we are actually only concerned with 1-neighbors that have the same length as the original word. To find the sphere packing bound for a 1-error correcting code where we allow the codewords to have length n only, we find the classic sphere packing bound given in (4)

$$|M| \leq \frac{4^n}{\displaystyle\sum_{i=0}^{1} \binom{n}{i} (4-1)^i} = \frac{4^n}{3n + 1}$$

Table 3.1 Sphere packing bounds for 1-error correcting codes with code words of length n

n	Code Size
1	1
2	2
3	6
4	20
5	64
6	216
7	745
8	2621
9	9362
10	33825

where $|M|$ is the maximum size of the code. Table 3.1 gives the sphere packing bounds for 1-error correcting codes with words of length $n \leq 10$.

3.2 Spheres of Radius 2

Let W be a word of length n created from the DNA alphabet. We need to consider words of length n that we can obtain from W by two substitutions, an insertion followed by a deletion, or a deletion followed by an insertion. Theorem 4 still holds because the size of the alphabet was irrelevant in the proof. So we now need only consider words obtained from W by two substitutions or by a deletion followed by an insertion. Observe that the number of words found by substituting d characters in a word W of length n is $3^d \binom{n}{d}$ since there are d positions we want to change, n positions to choose from and three possible characters to use. So the number of 2-edit neighbors found by two substitutions is $9 \binom{n}{2}$. Now we need to compute the number of del-ins.

Lemma 6. *Let W be a word of length n with k blocks. The number of ways to delete a character and then insert a character to arrive at a word $W' \neq W$ is $3nk$.*

Proof. From Theorem 10 we know there are k ways to delete a character from a word. Now we have a word of length $n - 1$. By Theorem 10, we now have $3(n - 1) + 4 = 3n + 1$ ways to insert a character in the shortened word to arrive back at a word of length n. However, one of these words will be the original word, so there are $3n$ ways to insert and arrive at a new word. Since there are k ways to delete and $3n$ ways to insert, there are $3nk$ ways to delete and then insert and not arrive back at the original word. ∎

This result overcounts the number of del-ins because some del-ins can be reached by two distinct del-in events. As in Chapter 2, we begin by considering the special case of alternating strings.

Lemma 7. *If W is an alternating string of length n comprised of characters X and Y, then any word $W' \neq W$ obtained from W by a del-in event can be found by a del-in event*

1. *in exactly two ways if the deletion and insertion occur in non-adjacent blocks and the character inserted is either X or Y, or*

2. *exactly one way if the deletion and insertion occur in adjacent blocks or the character inserted is not X or Y.*

Proof. The first part of the lemma follows directly from Lemma 3 with X taking on the role of 0 and Y taking on the role of 1. The second part also follows from Lemma 3 and noticing that if you insert a character that is not X or Y, there is no other possible way to arrive at that word. ∎

Theorem 11. *Let W be an alternating string of length n. If we count all possible ways to delete and then insert, we overcount the number of del-ins by $\binom{n}{2}$.*

Proof. By Lemma 7, all del-ins are counted either once or twice. The del-ins found by deleting block i and lengthening block j with the correct character (consider an end extension as a lengthening of block 0 or block $n + 1$) where $|i - j| \geq 2$ are counted twice. Consider if $i < j$, we can also find the same del-in by deleting block $j - 1$ and lengthening block $i - 1$ with the correct character. Now, if $i > j$, we can also find the same del-in by deleting block $j + 1$ and

lengthening block $i+1$ with the correct character. But if $i > j$, the other del-in event that deletes block $j+1$ and lengthens block $i+1$ has already been considered because $j+1 < i+1$. So to find the overcounting we only need to count the pairs of pairs $(i,j), (j-1, i-1)$, which is the same as in Theorem 5, so we see that we overcount the del-ins by $\binom{n}{2}$. ∎

Now we can use this result to find the number of ways to arrive at a del-in of any word W.

Theorem 12. *If $W' \neq W$ is obtained from W by a del-in event, then this may be done in only one way unless*

1. *the deletion and insertion occur within a single element that is a locally maximal alternating substring (LMAS) of characters X and Y and the character inserted is either X or Y, or*

2. *on the boundary between two adjacent blocks composed of characters X and Y and the character inserted is either X or Y,*

in which case there are exactly two ways.

Proof. Suppose the deletion occurs in a LMAS but the insertion does not, and the insertion does not occur on the boundary between two adjacent blocks. This is the same as in Theorem 6.

If the deletion occurs in a non-trivial block but the insertion does not, and neither the insertion nor deletion occur on the boundary between two adjacent blocks, we have the identical case as in Theorem 6.

By Lemma 7, if the deletion and insertion occur within a LMAS composed of characters X and Y and the character inserted is either X or Y, there are exactly two del-in events that yield the same del-in. If the character inserted is not X or Y, we will only be able to accomplish this in one way.

Suppose the deletion and insertion occur within a non-trivial block. There is only one way to accomplish this del-in since all the deletions within the block are equivalent and all the

insertions yield different del-ins.

Suppose the deletion and insertion occur on the boundary between two adjacent blocks composed of characters X and Y. We can delete X from the left block and split the right block after its first character by inserting a Y if it is of length two or more. If the right block is of length one, we lengthen the block to its right by inserting a Y. This yields the same del-in we get if we delete a Y from the right block and split the left block between its last two characters with an X if it is of length two or more or lengthen the block to the left of it by inserting an X if it is of length one. So there are exactly two del-in events that arrive at the same del-in. Notice that a deletion and insertion on the boundary of two adjacent blocks occurs either within a LMAS or on the boundary of two adjacent elements. If we insert a character that is not X or Y, we create a new block of size one and there is no other way to arrive at the same del-in. The theorem follows. ∎

Using the results of Lemma 2, Lemma 7, Theorem 11, and Theorem 12, we arrive at the following result.

Theorem 13. *If W is a word of length n with k blocks then the number of del-ins of W of length n is*

$$3nk - k + 1 - \sum_{i=2}^{n} a_i \left(\binom{i}{2} - (i-1) \right),$$

where a_i is the number of LMAS of W of length i.

Proof. The number of ways to delete and then insert is $3nk$ by Lemma 6. But by Theorem 12, this counts del-ins that can be found by deleting and inserting (the correct character) within a LMAS or on a boundary between two adjacent blocks twice. Since there are $k - 1$ block boundaries, we need to subtract $k - 1$ from our total number of ways to delete and then insert. There are $\sum_{i=2}^{n} a_i$ LMAS's and the amount of overcounting that occurs in each LMAS is $\binom{i}{2}$. So the total amount of overcounting due to a del-in event within a LMAS is $\sum_{i=2}^{n} a_i \binom{i}{2}$. But block boundaries occur within a LMAS as well as between adjacent elements. Since we have

now subtracted them twice, we need to add them back in. There are $i - 1$ block boundaries in a LMAS of length i, so we arrive at

$$3nk - (k - 1) - \sum_{i=2}^{n} a_i \binom{i}{2} + \sum_{i=2}^{n} a_i(i - 1) = 3nk - k + 1 - \sum_{i=2}^{n} a_i \left(\binom{i}{2} - (i - 1) \right). \quad \blacksquare$$

We now know how many words can be obtained from W by two substitutions and how many are del-ins of W. But this overcounts the number of 2-edit neighbors in two ways. First, notice that Theorem 4 still holds. Since the number of words of length n found by substituting one character is $\binom{n}{1} = n$ and there are three possible characters to substitute, there are $3n$ del-in events that are edit distance 1 (and hence not edit distance 2). We also want to find the number of words that can be found from W by both a del-in event and by two substitutions.

Theorem 14. *The number of 2-edit neighbors of a word than can be found by both a del-in event and by substituting two characters is $6n - k - 3n_1 - 3n_k + 1$, where n_i is the length of the ith block.*

Proof. Just as in Theorem 8, we have three cases. Case 1 is identical. In Case 2, the number of ways to split a block becomes $3(n_j - 1) + 2$. To see where the extra 2 comes from observe that inserting a different character in the last position of the block is not always the same as lengthening the next block because we have two extra characters. So for each $j = 2, \ldots, k - 1$ we can insert a character in block j that does not lengthen block j in $3(n_j - 1) + 2$ ways. For $j = 1$ and $j = k$, we only have $3(n_j - 1)$ ways to split because the extra two are now end

extensions. This causes our result for Case 2 to become

$$\sum_{j=2}^{k-1}[3(n_j - 1) + 2] + 3(n_k - 1) + \sum_{j=2}^{k-1}[3(n_j - 1) + 2] + 3(n_1 - 1)$$

$$= \sum_{j=2}^{k} 3(n_j - 1) + \sum_{j=2}^{k-1} 2 + \sum_{j=1}^{k-1} 3(n_j - 1) + \sum_{j=2}^{k-1} 2$$

$$= 3\left(\sum_{j=2}^{k}(n_j - 1) + \sum_{j=1}^{k-1}(n_j - 1)\right) + 4\sum_{j=2}^{k-1} 1$$

$$= 3(2n - 2k - n_1 - n_k + 2) + 4(k - 2)$$

$$= 6n - 2k - 3n_1 - 3n_k - 2.$$

This counts delete-splits occurring at block boundaries twice. So we need to subtract the number of block boundaries, which is $k - 1$. This give us

$$6n - 2k - 3n_1 - 3n_k - 2 - (k - 1) = 6n - 3k - 3n_1 - 3n_k - 1.$$

Case 3 now has six possibilities, three for each end. The overall result is

$$2k - 4 + 6n - 3k - 3n_1 - 3n_k - 1 + 6 = 6n - k - 3n_1 - 3n_k + 1. \qquad \blacksquare$$

Putting all these results together, we find the number of 2-edit neighbors.

Theorem 15. *If W is a word of length n with k blocks then the number of 2-edit neighbors of W is*

$$9\binom{n}{2} + 3n(k - 3) + 3n_1 + 3n_k - \sum_{i=2}^{n} a_i\left(\binom{i}{2} - i + 1\right),$$

where a_i is the number of LMAS of W of length i.

Proof. The number of 2-edit neighbors of a word is the number of words found by two substitutions together with the del-ins that are not also 1-edit neighbors. But some del-ins are also found by substitution, so we need to subtract them. By Theorem 13, Theorem 4 and Theorem

14, we have

$$9\binom{n}{2} + \left[3nk - k + 1 - \sum_{i=2}^{n} a_i \left(\binom{i}{2} - (i-1) \right) - 3n \right] - (6n - k - 3n_1 - 3n_k + 1)$$

$$= 9\binom{n}{2} + 3nk - k + 1 - \sum_{i=2}^{n} a_i \left(\binom{i}{2} - i + 1 \right) - 3n - 6n + k + 3n_1 + 3n_k - 1$$

$$= 9\binom{n}{2} + 3nk - 9n - \sum_{i=2}^{n} a_i \left(\binom{i}{2} - i + 1 \right) + 3n_1 + 3n_k$$

$$= 9\binom{n}{2} + 3n(k-3) + 3n_1 + 3n_k - \sum_{i=2}^{n} a_i \left(\binom{i}{2} - i + 1 \right). \qquad \blacksquare$$

Lemma 8. *Let W be a word of length n over a four letter alphabet. Let $|S_2|$ be the number of 2-edit neighbors $W' \neq W$ of length n. Then*

$$9\binom{n}{2} \leq |S_2|$$

and equality holds if $k = 1$ where k is the number of blocks.

Proof. Every word at Hamming distance 2 from W is also a 2-edit neighbor of W (if it was not a 2-edit neighbor, it would have to be a 1-edit neighbor, but since it must be of length n, that would make it Hamming distance 1 which contradicts being Hamming distance 2). If $k = 1$, notice that $n_1 = n_k = n$ and that $a_i = 0$ for $i = 2, \ldots, n$. Using Theorem 15 we see the number of 2-edit neighbors is

$$9\binom{n}{2} + 3n(k-3) + 3n_1 + 3n_k - \sum_{i=2}^{n} a_i \left(\binom{i}{2} - i + 1 \right)$$

$$= 9\binom{n}{2} + 3n(1-3) + 3n + 3n - \sum_{i=2}^{n} (0) \left(\binom{i}{2} - i + 1 \right)$$

$$= 9\binom{n}{2} + 3n(-2) + 6n$$

$$= 9\binom{n}{2}. \qquad \blacksquare$$

This shows that the sphere packing bound for spheres of radius 2 is once again no better

than that of the Hamming code. Again, the number of words for which equality in Lemma 8 holds should be small, so it would be a good next step to examine all possible cases where equality holds and exclude them to see if we can find a better bound for that subset of the edit code.

BIBLIOGRAPHY

[1] Ashlock, Dan. *Optimization and Modeling with Evolutionary Computation.* Unpublished lecture notes.

[2] Ashlock, Dan, Ling Guo, Fang Qiu. Greedy Closure Evolutionary Algorithms *Proceedings of the 2002 Congress on Evolutionary Computation.* 2001, pp 1296-1301.

[3] Biggs, Norman L. *Discrete Mathematics.* Oxford: Claredon Press, 1985. pp. 375-396.

[4] Brouwer, A.E. Bounds on the Size of Linear Codes *Handbook of Coding Theory Vol. 1.* Eds. V.S. Pless and W.C. Huffman. New York: Elsevier Science BV, 1998. pp. 297-298.

[5] Brualdi, Richard A. *Introductory Combinatorics*, 2nd Edition. Edgewood Cliffs, NJ: Prentice Hall, 1992. pp 466-467.

[6] Evans, Isaac K. Evolutionary Algorithms for Vertex Cover *Evolutionary Programming VII, Vol. 1447*, Lecture Notes in Computer Science. Ed. V.W. Porto. New York: Springer-Verlag, 1998. pp 377-386.

[7] Hungerford, Thomas W. *Algebra*, Graduate Texts in Mathematics. New York: Springer-Verlag, 1974. p. 10

[8] West, Douglas B. *Introduction to Graph Theory*, 2nd Edition. Upper Saddle River, NJ: Prentice Hall, 2001. pp 95-96.

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, Dr. Daniel Ashlock for his guidance, patience and support throughout this research and the writing of this thesis. I could not have done it without him. I would also like to thank Doug Ray and Amy Wangsness for providing assistance in proofreading this thesis and making insightful suggestions on improving this document.