LMAS: The Key to Enumerating 2-Spheres Over the Edit Metric

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

Abstract

This paper gives the exact size of edit spheres of radius 1 and 2 for any word over a finite alphabet. Structural information about the edit metric space, in particular a representation as a pyramid of hypercubes, will be given. The 1-spheres are easy to understand, being identical to 1-spheres over the Hamming metric. Edit metric 2-spheres are much more complicated. The size of a 2-sphere hinges on the structure of the word at its center. That is, the word's length, number of blocks, and most importantly (and troublesome) the number of locally maximal alternating substrings (LMAS) of each length. An alternating substring switches back and forth between two characters, e.g. 010101, and is maximal if it is contained in no other such substring. This variation in sphere size depending on center characteristics is what truly separates the algebraic character of codes over the edit metric from those over the Hamming metric.

1 Introduction

This paper will focus on the mathematics underlying error correcting codes over a natural biological metric space. It is hoped that a better theoretical understanding of this space and its code can be used to improve an evolutionary algorithm based search heuristic for these codes.

Often times when studying the genetics of a living organism, we wish to know what a particular gene does or which gene controls a certain characteristic. One problem in investigating these genes is that many genes are inactive during the life of the organism, only turned on by particular environmental conditions. Researchers must then provide the environmental conditions necessary to activate a particular gene. An example might be subjecting a plant to extreme cold, excess water, lack of sunlight, or treatment with a pesticide. Once the gene is activated, the RNA associated with the gene is isolated and mass produced by *E. coli* or a like organism to create a cDNA library. It is typically not cost effective to create a different cDNA library for each condition, so all of the samples are mixed together when making the cDNA library. The problem then becomes identifying which condition was being used for a particular sample.

To identify the source tissue, short stretches of DNA, called DNA barcodes, are embedded into the tissue 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.* An *(n,d)-code* is a collection of strings of length n with all pairs at distance d or more under some appropriate measure of distance. 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 using only those codewords that are sufficiently far apart ($d = 2k+1$) under the metric being used. 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 needed to transform one string into the other. Two codewords that are edit distance k from each other are called *k-edit neighbors.*

To find an error-correcting code we can use a greedy algorithm under evolutionary control. This greedy algorithm is described in the next section.

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 that always produces optimal results is Kruskal's algorithm for producing a minimum spanning tree of a connected weighted graph [4, 7].

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

Table 1: Comparison of binary Hamming code sizes found using Conway's Lexicode Algorithm and the greedy closure evolutionary algorithm for length n, minimum Hamming 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

varied, in a process similar to biological evolution, 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] gives an example of 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 1. 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.

A comparison of Conway's Lexicode Algorithm using the Hamming distance on binary words to the greedy closure evolutionary algorithm is given in Table 1. Notice that the greedy closure evolutionary algorithm performs better than Conway's Algorithm as the code length and minimum distance increase.

Notice Conway's algorithm can be re-specialized to the edit distance, so we can see the analog of Table 1 in Table 2. The edit metric version of this algorithm can be used to find a lower bound on the size of edit codes, see Table 3 for binary examples.

Table 2: Comparison of DNA edit metric code sizes found using Conway's Lexicode Algorithm and the greedy closure evolutionary algorithm for length n, minimum edit distance d codes. The figures in parenthesis are the fraction of times the best result was located.

n	d	Basic Lexicode	Evolutionary Algorithm
4	3	12	16 (18%)
5	3	36	41 (2%)
5	4	8	11 (1%)
6	3	96	106 (2%)
6	4	20	25 (11%)
6	5	4	9 (9%)
7	3	311	329 (2%)
7	4	57	63 (1%)
7	5	14	18 (12%)
7	6	4	7 (92%)

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 in [2] 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.

In order to improve the error correcting code produced by the greedy closure evolutionary algorithm, a thorough study of the edit metric space is needed. The following provides the beginnings of the theory of edit metric spaces.

2 Structure of the Edit Graph

The *edit distance* between two words 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. The edit distance is actually a metric, see [5].

A *block* of a word is a locally maximal substring comprised of exactly one character. The *block representation* of a word is a sequence of numbers representing how many times a character repeats. Notice the representation

Table 3: Lower bounds on the size of a maximal size binary edit code with length n and minimum distance d between words

	d							
n	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

does not specify a unique word. For example, over the binary alphabet, the block representation 1,2,2,1 represents the strings 011001 and 100110. The number of words with the same block representation depends on the size of the alphabet and the number of blocks.

The edit graph has complex structure. For words of length n, the induced subgraph that considers only substitutions is the Hamming graph. When $q = 2$, 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 construct 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 1 shows the top portion of the graph.

3 Edit Metric Over the Binary Alphabet

To begin, we consider the edit metric over a binary alphabet. Given a binary string w with length n and k blocks, we would like to know the number of neighbors it has at edit distance d. Due to the complexity of the edit metric, we restrict ourselves to $d = 1, 2$. We will later generalize these

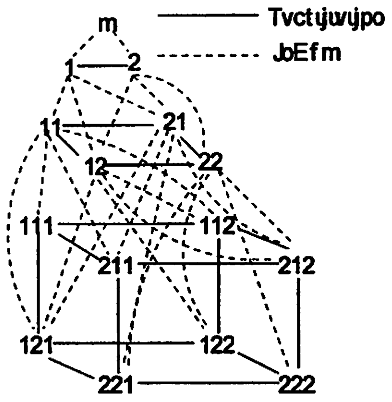

Figure 1: The top "levels" of the edit metric on the binary alphabet shown as a graph with distance one edges. Recall that λ denotes the empty string.

results to a q-ary alphabet.

3.1 Spheres of Radius 1

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. We say we *split* a block if we insert a character within a block that changes the number of blocks in the word. Lastly, we say we add *end extensions* if we create a new first or last block of size one.

Theorem 1. *Suppose that w is a binary word of length n with k blocks. Then the number of 1-edit neighbors of w in is $2n + 2 + k$.*

Proof. The word w has three sorts of neighbors, those obtained by single character substitution, insertion, and deletion. The deletion of a character from any position in a block of w yields the same result as a deletion in any other position of that block. The number of neighbors resulting from deletion is thus the number of blocks k. Substitutions require that a position be picked and a new character placed there which can happen in n ways. Insertions are slightly more complex. There are three possible types of insertion. We can insert a matching character into a block, lengthening it. There is one way to do this per block for a total of k such insertions. We may insert at a position in the interior of the string that is not a boundary between blocks, splitting the block. There are $(n - 1) - (k - 1) = (n - k)$ positions where this can happen and 1 possible character for each such insertion. Finally we may add a character to the beginning or end of the string in a manner that does not lengthen an existing block in 2 positions with 1 available character. Summing up,

$$k + n + k + (n - k) + 2 = 2n + 2 + k.$$

∎

3.2 Spheres of Radius 2

We 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 2. *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. Obvious. ∎

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. A 2-edit neighbor found by a deletion followed by an insertion is called 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. 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 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 1. *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 the proof of Theorem 1 we know there are k ways to delete a character from a word. Now we have a word of length $n - 1$. Also by the proof of Theorem 1, 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. 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.*

Lemma 2. *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. The proof, found in [5], is a tedious but simple examination of cases. ■

Theorem 3. *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 2, 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 $2 \leq i \leq n$.*

Let n_i be the length of the ith block. We say a block is non-trivial if it has length exceeding 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.*

For example, the L-B decomposition of 11011 is 11,101,11. Notice that there is overlap between the elements.

Theorem 4. *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. The proof, found in [5], is another tedious but simple examination of cases. ∎

Theorem 5. *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 1. But by Theorem 4, 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 a LMAS of length i, so we arrive at

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

∎

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 3. *Given a word w, every word w' found by substituting one character in w can also be found by a del-in event.*

Proof. Obvious. ■

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 both a del-in event and by substituting two characters.

Theorem 6. *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 $|i-j| = 2$, $1 \le i, j \le k$.
2. delete from block i and split block j, where $|i-j| = 1$, $1 \le i, j \le k$.
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

$$\begin{array}{ll} i < j & \text{we let } i = 1, \ldots, k-2;\ j = 3, \ldots, k \\ i > j & \text{we let } i = 3, \ldots, k;\ j = 1, \ldots, k-2 \end{array}$$

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

$$\begin{array}{ll} i<j & \text{we let } i=1,\ldots,k-1;\ j=2,\ldots,k \\ i>j & \text{we let } i=2,\ldots,k;\ j=1,\ldots,k-1 \end{array}$$

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

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

But this overcounts insertions made at the block boundaries. 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 $k-1$.
<u>Case 3</u>: 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 7. *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 5, Lemma 3 and Theorem 6, we have

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

■

Finding bounds on the 2 spheres is not very satisfying.

Lemma 4. *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 7 we see the number of 2-edit neighbors is

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

This shows that the obvious sphere packing bound for spheres of radius 2 is no better than that for codes over the Hamming metric: the smallest 2-edit sphere is the same size as a 2-Hamming sphere at each length. The number of words for which equality in Lemma 4 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.

3.3 Generalizing to q-ary Strings

We would like to generalize our results about the edit metric on binary strings to q-ary strings. This will have biotech applications for $q = 4$ (DNA) and $q = 20$ (protein). We will use block representation in the same manner.

3.3.1 Spheres of Radius 1

Theorem 8. *Let w be a word over a q-ary alphabet of length n with k blocks in its block representation. w has k 1-edit neighbors of length $n-1$, $n(q-1)$ 1-edit neighbors of length n, and $n(q-1)+q$ 1-edit neighbors of length $n+1$.*

Proof. The proof is the same as the proof of Theorem 1, except there are more characters available for substituting, splitting and end extending, resulting in different formulas for the number of 1-neighbors. ■

As with binary words, for 1-error correcting codes, we have restricted ourselves to 1-edit neighbors that have the same length as the original word, due to the nature of the motivating application of DNA barcodes.

3.3.2 Spheres of Radius 2

Let w be a word of length n created from a q-ary 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 2 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 $(q-1)^d\binom{n}{d}$ since there are d positions we want to change, n positions to choose from, and $q-1$ possible characters to use. So the number of 2-edit neighbors found by two substitutions is $(q-1)^2\binom{n}{2}$. Now we need to compute the number of del-ins. We begin by counting the number of del-in events.

Lemma 5. *Let w be a q-ary 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 $nk(q-1)$.*

Proof. From the proof of Theorem 8 we know there are k ways to delete a character from a word. Now we have a word of length $n-1$. By the proof of Theorem 8, we now have $(q-1)(n-1)+q = n(q-1)+1$ 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 $n(q-1)$ ways to insert and arrive at a new word. Since there are k ways to delete and $n(q-1)$ ways to insert, there are $nk(q-1)$ 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 3.2, we begin by considering the special case of alternating strings.

Lemma 6. *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 neither X nor Y.*

Proof. The first part of the lemma follows directly from Lemma 2 with X taking on the role of 0 and Y taking on the role of 1. The second part also follows from Lemma 2 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. ■

Notice that Theorem 3 still holds.

Now consider any word over a q-ary alphabet. We can segment the word into locally maximal alternating substrings (LMASs) and blocks. Notice that we may have a character appear in two adjacent LMASs. For example $CGCGAGAGAG$ is comprised of two LMASs: $CGCG$ and $GAGAGAG$. Notice that the boundaries of these LMASs do not provide any further problems because they are comprised of three characters, not just two. Hence we only worry about the block boundaries and LMASs to find the number of ways to arrive at a del-in as shown in the following:

Theorem 9. *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. *the deletion and insertion occur 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. This is a straightforward generalization of Theorem 4 ■

Using the results of Lemma 1, Lemma 6, Theorem 3, and Theorem 9, we arrive at the following result.

Theorem 10. *If w is a q-ary word of length n with k blocks then the number of del-ins of w of length n is*

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

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

Proof. The number of ways to delete and then insert is $nk(q-1)$ by Lemma 5. But by Theorem 9, 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

$$\begin{aligned} nk(q-1)-(k-1)-\sum_{i=2}^{n} a_i\binom{i}{2}+\sum_{i=2}^{n} a_i(i-1) \\ = nk(q-1)-k+1-\sum_{i=2}^{n} a_i\left(\binom{i}{2}-(i-1)\right). \end{aligned}$$ ∎

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 Lemma 3 still holds. Since the number of words of length n found by substituting one character is $\binom{n}{1}=n$ and there are $q-1$ possible characters to substitute, there are $n(q-1)$ 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 11. *The number of 2-edit neighbors of a q-ary word that can be found by both a del-in event and by substituting two characters is $(q-1)(2n-n_1-n_k)-k+1$, where n_i is the length of the ith block.*

Proof. Just as in Theorem 6, we have three cases. Case 1 is identical. In Case 2, the number of ways to split the jth block becomes $(q-1)(n_j-1)+(q-2)$. To see where the extra $q-2$ 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 $q-2$ 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 $(q-1)(n_j-1)+(q-2)$ ways. For $j=1$ and $j=k$, we only have $(q-1)(n_j-1)$ ways to split because the extra two are now end extensions. This causes our result for Case 2 to become

$$\begin{aligned}\sum_{j=2}^{k-1}&[(q-1)(n_j-1)+(q-2)]+(q-1)(n_k-1)\\ &+\sum_{j=2}^{k-1}[(q-1)(n_j-1)+(q-2)]+(q-1)(n_1-1)\\ &=(q-1)(2n-n_1-n_k)-2q-2k+6.\end{aligned}$$

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

$$(q-1)(2n-n_1-n_k)-2q-2k+6-(k-1)=(q-1)(2n-n_1-n_k)-2q-3k+7.$$

Case 3 now has $2(q-1)$ possibilities, $q-1$ for each end. The overall result is

$$2k-4+(q-1)(2n-n_1-n_k)-2q-3k+7+2(q-1)=(q-1)(2n-n_1-n_k)-k+1.$$

■

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

Theorem 12. *If w is a q-ary word of length n with k blocks then the number of 2-edit neighbors of w is*

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

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 10, Lemma 3, and Theorem 11, we have

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

∎

Now we have a sphere packing bound for 2-edit neighbors, but again, it is not very satisfying.

Lemma 7. *Let w be a word of length n over a q-ary alphabet. Let $|S_2|$ be the number of 2-edit neighbors $w' \neq w$ of length n. Then*

$$(q-1)^2\binom{n}{2} \leq |S_2|$$

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 12 we see the number of 2-edit neighbors is

$$\begin{aligned}
&(q-1)^2\binom{n}{2} + (q-1)\left[n(1-3) + n + n\right] - \sum_{i=2}^{n}(0)\left(\binom{i}{2} - i + 1\right) \\
&\qquad = (q-1)^2\binom{n}{2}.
\end{aligned}$$

∎

4 Discussion

While significant progress has been made in enumerating the edit space and determining the symmetry of the edit space [5], there are still a number of unanswered questions.

First, while there has been a formula found to compute the number of edit distance d neighbors of a strictly alternating string [5], there is no generalization to a q-ary alphabet.

Secondly, the above mentioned formula needs to be generalized for arbitrary strings. At present, formulae are only given for monotone and strictly alternating strings. Computer enumeration demonstrates that the formula for other types of strings is not a polynomial as the formulae for monotone and strictly alternating strings are.

Third, because of the motivating biotech application, the theory has been developed for codes made of strings of uniform length. Other applications may require formulae for all neighbors, rather than neighbors of the same length.

This paper illuminates some features of the edit metric space and develops tools toward finding an upper bound on the size of edit metric codes. The codes in actual use were created by various computer heuristics. Constructions for edit metric codes are a potentially interesting area not treated in this paper. The material in [5] suggests that such constructions are likely not to be as elegant as those for the Hamming metric. Nevertheless, this is an essentially untouched area.

References

[1] Daniel Ashlock. *Optimization and Modeling with Evolutionary Computation.* Springer-Verlag, New York, 2005.

[2] Daniel Ashlock, Ling Guo, and Fang Qiu. Greedy closure evolutionary algorithms. In *Proceedings of the 2002 Congress on Evolutionary Computation*, page 1296. IEEE Neural Networks Council, IEEE, 2002.

[3] Norman L. Biggs. *Discrete Mathematics.* Claredon Press, Oxford, 1985.

[4] Richard A. Brualdi. *Introductory Combinatorics.* Prentice Hall, Edgewood Cliffs, NJ, second edition, 1992.

[5] Jessie Campbell. *Enumeration and Symmetry of Edit Metric Spaces.* PhD thesis, Iowa State University, 2005.

[6] Isaac K. Evans. *Evolutionary Algorithms for Vertex Cover*, volume 1447 of *Lecture Notes in Computer Science.* Springer-Verlag, New York, 1998.

[7] Douglas B. West. *Introduction to Graph Theory.* Prentice Hall, Upper Saddle River, NJ, second edition, 2001.