

**Enumeration and symmetry of edit metric spaces**

by

Jessie Katherine Campbell

A dissertation submitted to the graduate faculty  
in partial fulfillment of the requirements for the degree of  
DOCTOR OF PHILOSOPHY

Major: Mathematics

Program of Study Committee:  
Daniel Ashlock, Major Professor  
Patrick Schnable  
Cliff Bergman  
Ryan Martin  
John Mayfield

Iowa State University

Ames, Iowa

2005

Copyright © Jessie Katherine Campbell, 2005. All rights reserved.

Graduate College  
Iowa State University

This is to certify that the doctoral dissertation of  
Jessie Katherine Campbell  
has met the dissertation requirements of Iowa State University

---

Major Professor

---

For the Major Program

## DEDICATION

I would like to dedicate this thesis to my parents, Ken and Dorothy. They have always believed that I would accomplish my goals, even when I didn't think I would. Thank you for believing in me and encouraging me to go on.

## TABLE OF CONTENTS

<b>LIST OF TABLES</b> . . . . .	vi
<b>LIST OF FIGURES</b> . . . . .	vii
<b>CHAPTER 1. Introduction</b> . . . . .	1
1.1 Greedy Algorithms . . . . .	2
1.2 Evolutionary Algorithms . . . . .	3
<b>CHAPTER 2. Edit Metric on the Binary Alphabet</b> . . . . .	7
2.1 Edit Metric versus Hamming Metric . . . . .	7
2.2 Structure of Edit Graph . . . . .	8
2.3 Spheres of Radius 1 . . . . .	8
2.4 Spheres of Radius 2 . . . . .	11
<b>CHAPTER 3. Generalizing to <math>q</math>-ary Strings</b> . . . . .	25
3.1 Spheres of Radius 1 . . . . .	25
3.2 Spheres of Radius 2 . . . . .	27
<b>CHAPTER 4. Symmetries of the Edit Graph</b> . . . . .	35
<b>CHAPTER 5. Counting Formulas for Repetitive Strings</b> . . . . .	40
5.1 Monotone Strings . . . . .	40
5.2 Alternating Strings . . . . .	41
<b>CHAPTER 6. Unanswered Questions</b> . . . . .	52
<b>CHAPTER 7. Possible Applications</b> . . . . .	53
<b>APPENDIX A. Code to calculate edit neighbors of repetitive strings</b> . . . .	55

<b>APPENDIX B. Code to compute all neighbors of a given string . . . . .</b>	<b>58</b>
<b>BIBLIOGRAPHY . . . . .</b>	<b>62</b>
<b>ACKNOWLEDGEMENTS . . . . .</b>	<b>63</b>

## LIST OF TABLES

1.1	Conway’s algorithm vs. greedy fitness evolutionary algorithm for binary codes under Hamming metric . . . . .	5
1.2	Conway’s algorithm vs. greedy closure evolutionary algorithm for DNA codes under edit metric . . . . .	6
1.3	Lower bounds on maximal binary edit code . . . . .	6
2.1	Sphere packing bounds for 1-error correcting edit codes with code words of length $n$ . . . . .	11
3.1	Sphere packing bounds for 1-error correcting codes over the DNA alphabet with code words of length $n$ . . . . .	27
5.1	Output of code in Appendix A using generator $g = 0$ . . . . .	41
5.2	Edit distance $d$ -neighbors of 01 and 010 . . . . .	45
5.3	Edit distance $d$ -neighbors of 0101 . . . . .	46
5.4	Output of code in Appendix A using generator $g = 01$ . . . . .	48
5.5	Patterns in Factored Regression Polynomials . . . . .	49

**LIST OF FIGURES**

2.1	Top levels of edit graph . . . . .	9
2.2	Examples of del-in events and corresponding del-ins . . . . .	13

## CHAPTER 1. Introduction

This thesis 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.

## 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 suboptimal 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 always produces optimal results is Kruskal's algorithm for producing a minimum spanning tree of a connected weighted graph (6; 9).

**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 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). (7) 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 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 *proper* coloring is a coloring in which adjacent vertices are assigned distinct colors. For the remainder of the chapter, all colorings

are proper. 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  $C = \{1, 2, \dots, n\}$ . 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 (8), 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 was never increased, the coloring is still optimal. ■

**Lemma 1.** *Suppose that  $P(v)$  is a coloring function that gives a packed coloring. If we give Algorithm 1 the vertices of a packed coloring in an order  $v_1, v_2, v_3, \dots, v_n$  so that  $P(v_i) \leq P(v_{i+1})$  it will return that coloring.*

*Proof.* Notice that the ordering will place the vertices in some order so that the smallest color is assigned to a collection of vertices, then the next smallest color is assigned, and so on. So any vertex  $v_i$  will be colored with some color  $c$  only after all of its neighbors that are colored with colors  $d < c$  have been colored. When Algorithm 1 terminates, the coloring is the packed coloring we began with. ■

Table 1.1 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

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 closure evolutionary algorithm is given in Table 1.1. 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 1.1 in Table 1.2. The edit metric version of this algorithm can be used to find a lower bound on the size of edit codes, see Table 1.3 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.

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 chapters provide the beginnings of the theory of edit metric spaces.

Table 1.2 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%)

Table 1.3 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

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. 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  $u_1, u_2, \dots, u_m$ . 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  $\{(x, u_i(x)) : x \in S \text{ and } i = 1, 2, \dots, m\}$ . The minimum number of operations required 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 (9). ■

**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, as can be seen by using the operation ‘flip the  $i$ th bit’ in Theorem 2. 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 2.** *If the size of the alphabet is  $q$  and the block representation has  $k$  blocks then there are  $q(q - 1)^{k-1}$  words with that block representation.*

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

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 complex structure. For any given word length  $n$ , 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 construct the edit graph, we let  $n = 0, 1, 2, \dots$  and stack the hypercubes with the empty string  $\lambda$  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. We refer to the edit graph over the alphabet  $\mathcal{A}$  as  $E(\mathcal{A})$ .

## 2.3 Spheres of Radius 1

Using the greedy closure 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

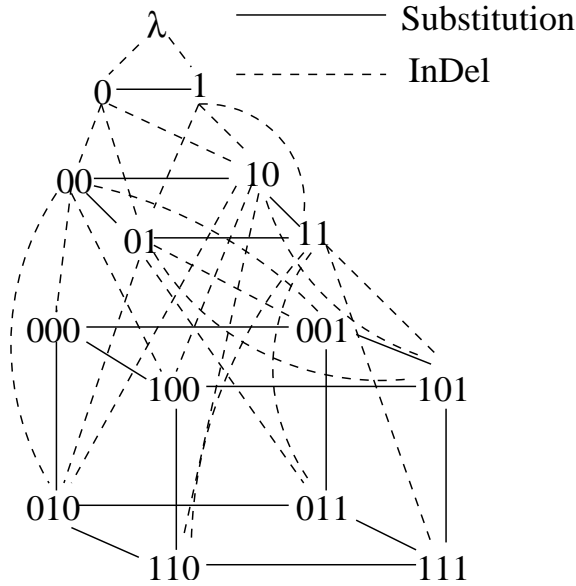


Figure 2.1 The top “levels” of the edit metric on the binary alphabet shown as a graph with distance-one edges. Recall that  $\lambda$  denotes the empty string.

$k$ -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, we can delete the third, fourth or fifth character of 110001 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). The number of ways to split a block can be found by noticing the following. 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  $n > 0$ . The minimum number of 1-edit neighbors of  $w$  is  $2n + 3$  and the maximum 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

$$\begin{aligned} 1 &\leq k \leq n \\ 2n + 1 &\leq 2n + k \leq 3n \\ 2n + 3 &\leq 2n + k + 2 \leq 3n + 2. \quad \blacksquare \end{aligned}$$

The total number of 1-neighbors is useful in providing intuition about how to write the evolutionary algorithm. For 1-error correcting codes, we will only concern ourselves 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  $w = x_1x_2\dots x_n$  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

$$w'_1 = x_1x_2\dots x_{i-1}x_{i+1}\dots x_n.$$

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

$$w''_1 = x_1x_2\dots x_{i-1}x_{i+1}\dots x_jzx_{j+1}\dots x_n.$$

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

$$w'_2 = x_1x_2\dots x_jzx_{j+1}\dots x_n.$$

Now delete character  $x_i$  to arrive at

$$w''_2 = x_1x_2\dots x_{i-1}x_{i+1}\dots x_jzx_{j+1}\dots x_n.$$

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*

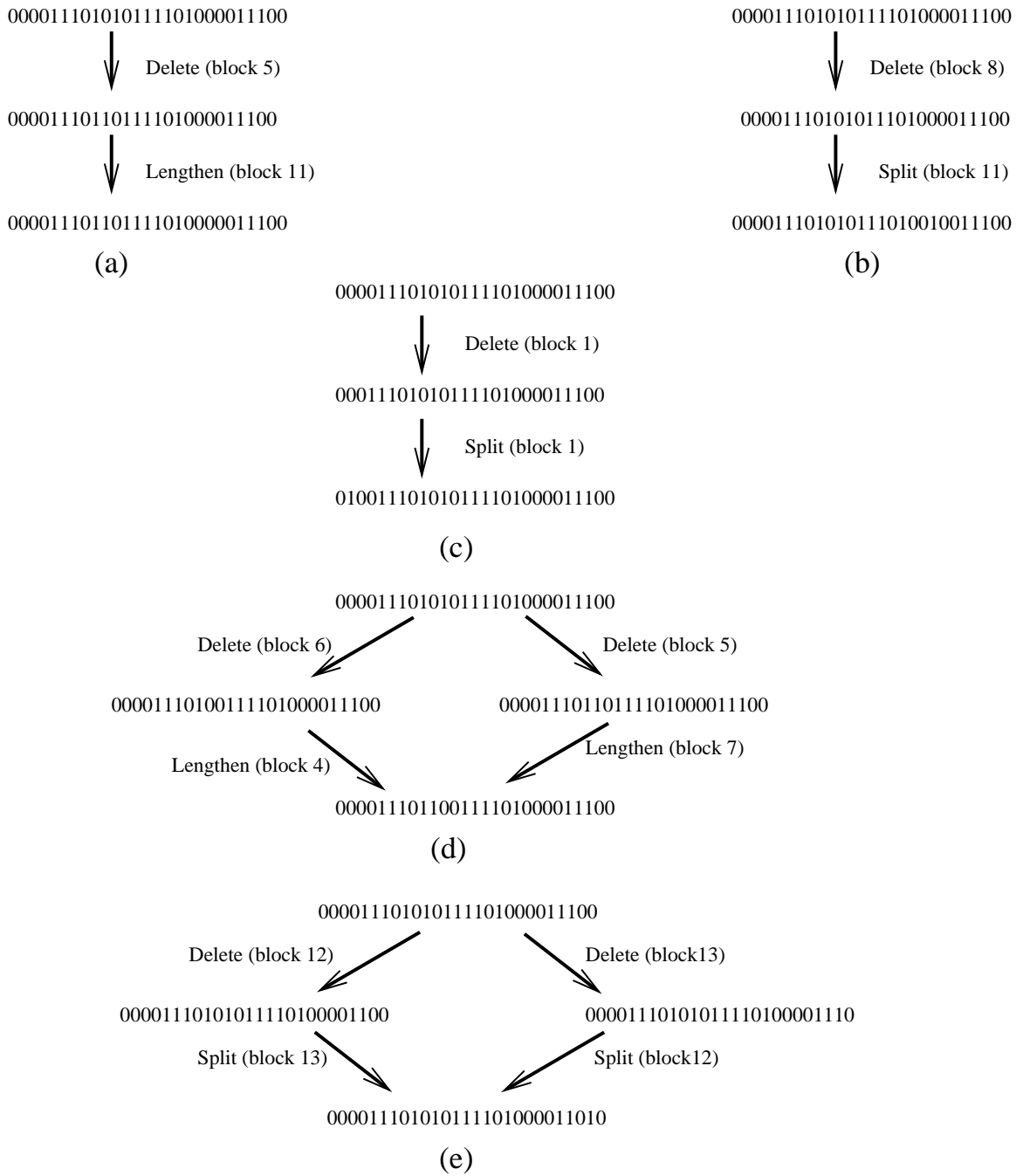


Figure 2.2 Some examples of del-in events and the corresponding del-ins. (a), (b) and (c) show del-ins that occur in only one way. (d) and (e) each show two del-in events that lead to the same del-in. Note that the insertion block refers to the block number in the original string.

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 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 3.** *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. 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 4.** *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  $i = 2, \dots, n-1$  and  $j = 1, \dots, n$  with  $|i - j| \geq 2$ .
2.  $w'$  is found from  $w$  by deleting block  $i = 1$  or  $i = n$  and lengthening block  $j$  where  $j = 1, \dots, n$  with  $|i - j| \geq 2$ .
3.  $w'$  is found from  $w$  by deleting block  $i$  and end extending where  $i = 2, \dots, n-1$ .
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  $i = 2, \dots, n-1$  and lengthening block  $j$  where  $|i - j| \geq 2$ . If  $i < j$ , we have  $w' = x_1 \dots x_{i-1} x_{i+1} \dots x_{j-1} x_j y x_{j+1} \dots x_n$  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  $p < q$ ,  $w' = x_1 \dots x_{p-1} x_{p+1} \dots x_{q-1} x_q z x_{q+1} \dots x_n$ , 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  $w' = x_1 \dots x_{q-1} x_q z x_{q+1} \dots x_{p-1} x_{p+1} \dots x_n$ . Let  $w' = u_1 \dots u_n = v_1 \dots v_n$  where

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

and

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

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  $i = 2, \dots, n - 1$  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.

Case 2: 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  $w' = x_2 \dots x_{j-1} x_j z x_{j+1} \dots x_n$ , where  $z = x_j$ . Since  $w = x_1 \dots x_n$  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  $w' = x_0 x_1 \dots x_{p-1} x_{p+1} \dots x_n$  where  $x_0 = x_2$ . Let  $w' = u_1 \dots u_n = v_1 \dots v_n$  where

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

and

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

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.

Case 3: Suppose  $w'$  is formed from  $w$  by deleting block  $i = 2, \dots, n - 1$  and end extending. If we end extend on the left, then  $w' = x_0 x_1 \dots x_{i-1} x_{i+1} \dots x_n$  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  $w' = x_2 \dots x_{q-1} x_q z x_{q+1} \dots x_n$  where  $z = x_q$ . Let  $w' = u_1 \dots u_n = v_1 \dots v_n$  where

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

and

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

If  $i < q - 1$ , then  $u_{i+1} = v_{i+1} = x_{i+2} \neq x_{i+1}$ . 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  $w' = x_2 \dots x_n x_{n+1}$  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  $i = 2, \dots, n - 1$ , the deletion occurs in block  $i = 1$ , or the deletion occurs in block  $i = n$ . If the deletion occurs in block  $i = 2, \dots, n - 1$ , 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 exactly  $\binom{n}{2}$ .*

*Proof.* By Lemma 4, 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  $i$ th block. 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.*

**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 4, 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 LMASs of  $w$  of length  $i$ .

*Proof.* The number of ways to delete and then insert is  $nk$  by Lemma 3. 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  $\sum_{i=2}^n a_i$  LMASs 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

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

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 5.** *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.  $\blacksquare$

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.

**Lemma 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  $|i - j| = 2, 1 \leq i, j \leq k$ .
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).

Case 1: To delete from block  $i$  and lengthen block  $j$  with

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

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

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.

Case 2: To delete from block  $i$  and split block  $j$  with

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

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

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

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

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

**Theorem 8.** *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 LMASs 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, Lemma 5 and Lemma 6, we have

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

Finding bounds on the 2 spheres is not very satisfying. The following lemma shows that the best scenarios occur when  $k$  is small.

**Lemma 7.** *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  $i = 2, \dots, n$ . Using Theorem 8 we see the number of 2-edit neighbors is

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

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 7 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 $q$ -ary Strings

We would like to generalize our results about the edit metric on binary strings to  $q$ -ary 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 use block representation in the same manner. The block representation of a  $q$ -ary string is a sequence of numbers representing how many times a character (from a  $q$ -ary alphabet) 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 2, we see there are  $q(q-1)^{k-1}$  strings with a given block representation, where  $k$  is the number of blocks in the block representation.

#### 3.1 Spheres of Radius 1

**Theorem 9.** *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 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. 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 places to do this for a word of length  $n$  and  $q-1$  characters to substitute at each place. So the total number of ways to substitute is  $n(q-1)$ .

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. There are  $q - 1$  possible characters to insert to create each of these end extensions, so this gives us  $2(q - 1)$  1-edit neighbors. Second, we can lengthen an existing block. This can be done  $k$  ways. Lastly, we can split an existing block. There are  $n - 1$  places to insert a character, but we can not insert the same number of characters in every place. There are  $k - 1$  block boundaries and we can only insert  $q - 2$  characters there, since the other two characters would result in lengthening one of the blocks. The remaining  $(n - 1) - (k - 1) = n - k$  places to insert can have  $q - 1$  characters inserted, since the other character would result in lengthening the block. So there are  $(k - 1)(q - 2) + (n - k)(q - 1) = n(q - 1) - k - q + 2$  ways to obtain a new word by splitting blocks. This results in a total of  $2(q - 1) + k + n(q - 1) - k - q + 2 = n(q - 1) + q$  1-edit neighbors of length  $n + 1$ . ■

**Corollary 4.** *The number of 1-edit neighbors of a  $q$ -ary word of length  $n$  with  $k$  blocks is  $2n(q - 1) + q + k$ .*

*Proof.* From Theorem 9, we see that the number of 1-edit neighbors of a  $q$ -ary word of length  $n$  with  $k$  blocks in its block representation is  $k + n(q - 1) + n(q - 1) + q = 2n(q - 1) + q + k$ . ■

**Corollary 5.** *Let  $w$  be a  $q$ -ary word of length  $n > 0$ . The minimum number of 1-edit neighbors of  $w$  is  $2nq - 2n + q + 1$  and the maximum number of 1-edit neighbors of  $w$  is  $2nq - n + q$ .*

*Proof.* The number of 1-edit neighbors is  $2n(q - 1) + q + k$  by Corollary 4. Since we always have at least one block and we never have more than  $n$  blocks, we have

$$\begin{aligned} 1 &\leq k \leq n \\ q + 1 &\leq q + k \leq q + n \\ 2n(q - 1) + q + 1 &\leq 2n(q - 1) + q + k \leq 2n(q - 1) + q + n. \end{aligned}$$

Thus, the result is proved. ■

As with binary words, for 1-error correcting codes, we are only concerned with 1-edit neighbors that have the same length as the original word. To find the sphere packing bound

Table 3.1 Sphere packing bounds for 1-error correcting codes over the DNA alphabet 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

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),

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

where  $|M|$  is the maximum size of the code. Table 3.1 gives the sphere packing bounds for 1-error correcting codes over the DNA alphabet ( $q = 4$ ) with words of length  $n \leq 10$ .

### 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 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  $(q-1)^d \binom{n}{d}$  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 8.** *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 Theorem 9 we know there are  $k$  ways to delete a character from a word. Now we have a word of length  $n-1$ . By Theorem 9, 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 Chapter 2, we begin by considering the special case of alternating strings.

**Lemma 9.** *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 4 with  $X$  taking on the role of 0 and  $Y$  taking on the role of 1. The second part also follows from Lemma 4 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 10.** *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 exactly  $\binom{n}{2}$ .*

*Proof.* By Lemma 9, 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 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  $CGCGGAGAGAGAG$  is comprised of two LMASs:  $CGCGCG$  and  $GAGAGAGAG$ . 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 11.** *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 nontrivial 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 9, 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 occur 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 3, Lemma 9, Theorem 10, and Theorem 11, we arrive at the following result.

**Theorem 12.** *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 LMASs of  $w$  of length  $i$ .

*Proof.* The number of ways to delete and then insert is  $nk(q-1)$  by Lemma 8. But by Theorem 11, 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  $\sum_{i=2}^n a_i$  LMASs 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

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

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 Lemma 5 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.

**Lemma 10.** *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  $i$ th 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  $j$ th 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  $j = 2, \dots, k-1$ , 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

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

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

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

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

**Theorem 13.** *If  $w$  is a  $q$ -ary word of length  $n$  with  $k$  blocks then the number of 2-edit neighbors of  $w$  is*

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

where  $a_i$  is the number of LMASs 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 12, Lemma 5, and Lemma

10, we have

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

■

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

**Lemma 11.** *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  $i = 2, \dots, n$ . Using Theorem 13 we see the number of 2-edit neighbors is

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

■

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 11 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 4. Symmetries of the Edit Graph

We will now show that the edit graph,  $E(\mathcal{A})$ , has very little symmetry. Define  $G_{\mathcal{A}}$  to be the automorphism group of  $E(\mathcal{A})$ .

Recall from Corollary 4 that the degree of a vertex  $v$  with  $k$  blocks is

$$\deg(v) = 2n(q - 1) + q + k, \quad (4.1)$$

where  $q = |\mathcal{A}|$ . Formally we will call the  $k$ th level of  $E(\mathcal{A})$  the induced subgraph on the words in  $\mathcal{A}^k$ . We denote the symmetric group of  $\mathcal{A}$  by  $Sym(\mathcal{A})$ .

**Lemma 12.**  $G_{\mathcal{A}}$  acts on the levels of  $E(\mathcal{A})$ .

*Proof.* The degree of the empty word  $\lambda$  is  $q$  by Equation 4.1. This same equation tells us all other words in the graph have higher degree. Since  $G_{\mathcal{A}}$  must preserve degree, we know that  $G_{\mathcal{A}}$  fixes  $\lambda$ . A minimal edit path from  $\lambda$  to a word  $w$  consists of adding the characters of  $w$  to  $\lambda$  in any one of a number of orders. This means that the edit distance between  $\lambda$  and  $w$  is simply the length of  $w$ . The levels are thus sets of words at fixed distances from  $\lambda$ . Since automorphisms preserve distance it follows that  $G_{\mathcal{A}}$  must take a word to a word of the same length and so we have that  $G_{\mathcal{A}}$  acts on the levels of  $E(\mathcal{A})$ . ■

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

We now show that a symmetric subgroup of  $G_{\mathcal{A}}$  does in fact exist.

**Lemma 13.** *The map*

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

given by  $\sigma \mapsto \sigma^*$  is an injective group homomorphism of  $\text{Sym}(\mathcal{A})$  into  $G_{\mathcal{A}}$ . We will denote the image of  $\text{Sym}(\mathcal{A})$  under  $\tau$  as  $\Sigma(\mathcal{A})$ . Note that  $\Sigma(\mathcal{A}) \cong \text{Sym}(\mathcal{A})$

*Proof.* The induced maps in the image of  $\tau$  simultaneously change the identity of all the letters in each word. If we have a minimal edit path that is a witness to the distance between two words in the graph then that path is preserved by the induced maps with the appropriate changes to the identity of substituted or inserted characters. The induced maps thus preserve distance and hence adjacency in the graph. It follows that the induced maps are automorphisms. ■

It can also be shown that a map  $\gamma$  that flips words end-for-end belongs to  $G_{\mathcal{A}}$ . For example,  $\gamma(010111) = 111010$ . The map  $\gamma$  simply reads the characters from right to left, rather than left to right.

**Lemma 14.** *The map  $\gamma$  from  $\mathcal{A}^*$  to itself that flips all words end-for-end is an element of  $G_{\mathcal{A}}$  that is not in  $\Sigma(\mathcal{A})$  if  $|\mathcal{A}| > 1$ .*

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

It is also not hard to see that  $\sigma$  and  $\gamma$  commute.

**Lemma 15.** *For all  $\sigma \in \Sigma(\mathcal{A})$ ,  $\sigma\gamma = \gamma\sigma$ .*

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

We can also see that any automorphism of  $E(\mathcal{A})$  does not change the number of blocks in a word.

**Lemma 16.** *Automorphisms of  $E(\mathcal{A})$  preserve the number of blocks in a word.*

*Proof.* From Equation 4.1 we see that within a level, degree is determined by the number of blocks in a word in such a way that words with different numbers of blocks have different degrees. Since automorphisms preserve degree, it follows that they preserve the number of blocks. ■

**Definition 4.** *For a word  $w$  and a character  $a \in \mathcal{A}$  we denote by  $|w|_a$  the number of occurrences of  $a$  in  $w$ .*

Recall from Lemma 12 that  $G_{\mathcal{A}}$  acts on the levels of  $E(\mathcal{A})$ . A *monotone* word is a word composed of a single character. In other words, monotone words are those that have a single block. In a given level, Equation 4.1 implies that the monotone words have the minimal degree within that level and are unique in having that degree within the level. As automorphisms preserve degree, it follows that  $G_{\mathcal{A}}$  acts on the monotone words within a level. Consider the representation  $f$  of  $G_{\mathcal{A}}$  on the monotone words. The image of  $\Sigma(\mathcal{A})$  under this representation is the full symmetric group on the monotone words. Since flipping a monotone word end-for-end does not change the word, it follows that  $\gamma \in \ker(f)$ . Let  $H = \ker(f)$ .

**Lemma 17.** *Members of  $H$  preserve  $|w|_a$ .*

*Proof.* Since  $G_{\mathcal{A}}$  acts on levels, it follows that any member of  $G_{\mathcal{A}}$  acts as an automorphism of the Hamming graph forming the level when restricted to a level. This means that it preserves distances within the Hamming graph. Suppose we are on level  $n$ . The distance from a monotone word made of the character  $a$  to a word  $w$  on level  $n$  within the level is  $n - |w|_a$ . Since the monotone words are fixed points by members of  $H$ , it follows that members of  $H$  preserve these distances and so preserve  $|w|_a$ . ■

From Lemma 17 we can deduce that  $H$  acts on the neighbors of any monotone word. Fix a monotone word  $w$  and denote by  $w_i^c$  the word that differs from it only in position  $i$  with  $c$  being the value of the character in the position where the word differs from  $w$ . Then we can see that members of  $H$  act on  $w_i^c$  for any fixed  $c$  by preservation of  $|w|_c$ . We fix  $c$  and study this action.

**Lemma 18.**  $H$  acts on  $\{w_k^c, w_{n+1-k}^c\}$ .

*Proof.* For  $k = 1$  this is true by preservation of the number of blocks as these two words have two blocks while all other  $w_i^c$  have three. Assume the lemma is true for all  $l < k$  and for all levels of  $E(\mathcal{A})$  above  $n$ . Examine  $w_k^c$ . If there exists an  $m$  such that  $w_k^c$  is taken to  $w_m^c$  by some member of  $H$  in a fashion that contradicts the lemma then  $k < m < n + 1 - k$  by examining the actions on the  $w_i^c$  already known for  $l < k$ . Let  $\overline{w_i^c}$  be  $w_i^c$  with its first character deleted. Then  $H$  acts on  $\{\overline{w_k^c}, \overline{w_{n-k}^c}\}$  by the assumption. Notice that  $w_k^c$  and  $\overline{w_k^c}$  are neighbors. By the assumption we know that if a member of  $H$  moves  $\overline{w_k^c}$  that it must go to  $\overline{w_{n-k}^c}$ . The only member of  $w_i^c$  that is a neighbor of  $\overline{w_{n-k}^c}$  is  $w_{n+1-k}^c$ . It follows that no  $w_m^c$  of the sort envisioned exists and the lemma follows by induction. ■

The following lemma bridges the gap between the results in (5) about the Hamming graph and the desired result for  $E(\mathcal{A})$ .

**Lemma 19.** *Permuting the positions of a collection of strings or permuting the characters of an alphabet at any fixed position in a collection of strings preserves the Hamming metric.*

*Proof.* Suppose we permute the positions in a collection of strings by the permutation  $\sigma$ . If  $w$  and  $w'$  are two strings that differ at position  $i$ , then  $w$  and  $w'$  will differ in position  $\sigma(i)$ . If  $w$  and  $w'$  are the same in position  $i$ , they will be the same in position  $\sigma(i)$ . So the number of bits that disagree (i.e. the Hamming distance) remains the same.

Now suppose we permute the characters of the alphabet at some fixed position  $i$  in a collection of strings by the permutation  $\pi$ . Let  $w_i$  denote the  $i$ th character of the string  $w$ . If  $w$  and  $w'$  are two strings that differ in position  $i$ , then  $\pi(w_i) \neq \pi(w'_i)$ . (If not, we would violate the condition that  $\pi$  is a permutation.) So  $\pi(w)$  and  $\pi(w')$  still differ in position  $i$ . If  $w$  and  $w'$  are the same in position  $i$ , then  $\pi(w_i) = \pi(w'_i)$ . So  $\pi(w)$  and  $\pi(w')$  are the same in position  $i$ . Thus the number of bits that disagree (i.e the Hamming distance) remains the same. ■

Putting these results together, we arrive at the following.

**Theorem 14.**  $G_{\mathcal{A}} \cong \mathbb{Z}_2 \times \text{Sym}(\mathcal{A})$ .

*Proof.* Examine the action of  $H$  on the set of  $w_l^c$ ,  $l = 1, \dots, n$  in level  $l$  of  $E(\mathcal{A})$  for fixed  $c$ . By considering how the action of  $H$  can move 4-cycles made of the monotone word  $w$ ,  $w_i^c$ ,  $w_j^c$ , for  $i \neq j$ , and the unique neighbor  $x$  of  $w_i^c$  and  $w_j^c$  for which  $|x|_c = 2$ , we see that the action of  $H$  on the  $w_l^c$  is exactly that of  $\langle \gamma \rangle$ . From what is known of the structure of the automorphism group of the Hamming graph (5) and Lemma 19, we see that any members of  $H$  must act by simultaneously permuting the positions of characters within all words. The action of  $H$  on the  $w_l^c$  permits the only such action to be that of  $\langle \gamma \rangle$ . It follows that  $H = \langle \gamma \rangle$ . Given that  $\gamma$  commutes with  $\Sigma(\mathcal{A})$  we can see that  $G_{\mathcal{A}} \cong \langle \gamma \rangle \times \Sigma(\mathcal{A})$ . Notice that  $\langle \gamma \rangle \cong \mathbb{Z}_2$  since is a cyclic group of order 2, and all cyclic groups of order 2 are isomorphic to the group  $\mathbb{Z}_2 = (\{0, 1\}, +, 0)$ , where  $+$  is addition modulo 2. We already noted in Lemma 13 that  $\Sigma(\mathcal{A}) \cong \text{Sym}(\mathcal{A})$ , so we have our result. ■

**Corollary 6.** *Automorphisms of  $E(\mathcal{A})$  preserve the number of blocks and block sizes present in words.*

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

In the theory of codes over the Hamming metric, the beautiful, symmetric space in which the codes lie is of substantial utility. The result just proved shows that the space of the binary edit metric has 4 total automorphisms. The relative rigidity of the space contributes to the relative difficulty in the theory of edit metric codes.

## CHAPTER 5. Counting Formulas for Repetitive Strings

A string is *repetitive* if it can be formed by taking a finite substring, called a *generator*, and concatenating it repeatedly. If only a finite number of concatenations are performed, the last concatenation need only concatenate a portion of the generator. Let  $S_d^{(g)_q^*}(n)$  be the number of edit distance  $d$  neighbors of length  $n$  of a repetitive string with generator  $g$  over a  $q$ -ary alphabet. We begin by finding  $S_d^{(g)_q^*}(n)$  for two simple generators, since these two generators are examples of extreme cases for all strings.

### 5.1 Monotone Strings

A repetitive string is *monotone* if its generator consists of a single character. Monotone strings have relatively few neighbors under the edit metric. We will show that monotone strings locally resemble strings under the Hamming metric. Let  $w$  be a monotone string and  $w'$  an edit distance  $d$  neighbor of  $w$  of length  $n$ .

**Lemma 20.**  $w'$  is found from  $w$  by performing  $d$  substitutions.

*Proof.* Suppose that a deletion and insertion occur in a sequence of edit operations transforming  $w$  to  $w'$ . By the same reasoning as Theorem 4, we know that we can perform all deletions first followed by all insertions and all substitutions. Suppose we have  $i$  deletions,  $i$  insertions, and  $d - 2i$  substitutions. All deletions from a monotone string are equivalent, since it only has one character. Inserting the same character that the string already consists of is redundant, so  $i$  characters (that are not the same as the generator) are then inserted in the string. Then we perform our  $d - 2i$  substitutions. However, the deletions and insertions could be replaced by  $i$  substitutions, which would mean that  $w'$  is a  $d - i$  edit neighbor. Since it is known that  $w'$  is a  $d$  edit neighbor, we have that  $i = 0$  and no deletions or insertions occur. ■

Table 5.1 Output of code in Appendix A using generator  $g = 0$ . Notice this is just Pascal's triangle.

$n$	$d = 0$	$d = 1$	$d = 2$	$d = 3$	$d = 4$	$d = 5$	$d = 6$	$d = 7$	$d = 8$
2	1	2	1						
3	1	3	3	1					
4	1	4	6	4	1				
5	1	5	10	10	5	1			
6	1	6	15	20	15	6	1		
7	1	7	21	35	35	21	7	1	
8	1	8	28	56	70	56	28	8	1
9	1	9	36	84	126	126	84	36	9
10	1	10	45	120	210	252	210	120	45
11	1	11	55	165	330	462	462	330	165
12	1	12	66	220	495	792	924	792	495
13	1	13	78	286	715	1287	1716	1716	1287
14	1	14	91	364	1001	2002	3003	3432	3003
15	1	15	105	455	1365	3003	5005	6435	6435

We noted in Chapter 3 that the number of words over a  $q$ -ary alphabet 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 and  $n$  positions to choose from and  $q - 1$  characters we can substitute for. Thus we have

$$S_d^{(0)_q^*}(n) = (q - 1)^d \binom{n}{d}$$

A code to calculate the edit distance  $d$  neighbors of a repetitive string over the binary alphabet for every nontrivial value of  $d$  (i.e.  $S_d^{(g)_2^*}(n)$ ) is given in Appendix A. If the generator  $g = 0$  is used, the data in Table 5.1 are generated. Notice that the entries are exactly  $\binom{n}{d}$ . If we wanted to know  $S_d^{(g)_q^*}(n)$ , we would simply multiply each entry by  $(q - 1)^d$ .

## 5.2 Alternating Strings

For the remainder we use  $q = 2$ . Notice an alternating string is a repetitive string with generator  $g = 01$  or  $g = 10$ . Without loss of generality, we will use  $g = 01$ . We now find a formula for  $S_d^{(01)_2^*}(n)$ .

**Theorem 15.** For  $d \geq 3$  and  $n \geq 4$ ,

$$S_d^{(01)_2^*}(n+1) = S_d^{(01)_2^*}(n) + 2S_{d-1}^{(01)_2^*}(n-1)$$

*Proof.* Let  $w$  be the alternating string of length  $n+1$  and let  $w'$  be a  $d$ -edit neighbor of  $w$ . There are three possible cases:

1.  $w$  and  $w'$  end in the same character.
2.  $w$  and  $w'$  agree in the  $n$ th position, but not the  $(n+1)$ th position.
3.  $w$  and  $w'$  do not agree in the  $n$ th and  $(n+1)$ th positions.

Case 1: Since  $w$  and  $w'$  end in the same character, we can assume without loss of generality that no edit operations occur in the last position. Hence, we can ignore the last character. Let  $w_1$  and  $w'_1$  be the strings that result in deleting the last character from  $w$  and  $w'$ , respectively. Notice that  $w_1$  is the alternating string of length  $n$  and  $w'_1$  will be a  $d$ -edit neighbor of  $w_1$ . This case contributes  $S_d^{(01)_2^*}(n)$  to  $S_d^{(01)_2^*}(n+1)$ .

Case 2: Without loss of generality, we assume that  $w'$  is found from  $w$  by a substitution in position  $n$  and no edit operations in the  $(n-1)$ th position. Let  $w_2$  and  $w'_2$  be the strings that result in deleting the last two characters from  $w$  and  $w'$ . Notice that  $w_2$  is the alternating string of length  $n-1$  and  $w'_2$  will be a  $d-1$  edit neighbor of  $w_2$ , since one of the edits occurs in the last position. This case contributes  $S_{d-1}^{(01)_2^*}(n-1)$  to  $S_d^{(01)_2^*}(n+1)$ .

Case 3: Notice to transform  $w$  into  $w'$  we need to flip the last two characters. This can be accomplished by two substitutions or by a deletion of the last character of  $w$  and an insertion event elsewhere in  $w$ . However, the other characters in the string effect whether or not it can be two substitutions. (For example, to transform  $w = 010101$  into its 3-edit neighbor  $w' = 011110$ , we must do a deletion and insertion. It is not possible to go from  $w$  to  $w'$  in only three edits if we do not perform a deletion and insertion.) So without loss of generality we will consider that all neighbors in Case 3 were found by a deletion and insertion, possibly followed by other edit operations.

Suppose we delete the last character from  $w$ . Without loss of generality, we consider insertions only to the left of position  $n-1$ . If we insert a character to the right of the  $n$ th

position, we either arrive back at  $w$  (which is frivolous) or we arrive at a string that differs from  $w$  in the  $(n + 1)$ th position and agrees in the  $n$ th position. Since we already considered strings of this form in case 2, we can ignore this possible insertion site. If we insert a character between position  $n - 1$  and position  $n$ , we either insert the same character as position  $n$  or the same character as position  $n - 1$ . If we insert the same character as position  $n$ , then we arrive at a string that differs from  $w$  in the  $(n + 1)$ th position and agrees in the  $n$ th position. Since we already considered strings of this form in case 2, we can ignore this possible insertion. If we insert the same character as position  $n - 1$ , we arrive at a string we can find by inserting between position  $n - 2$  and  $n - 1$ . So it suffices to only consider insertions occurring to the left of position  $n - 1$ .

We will insert a phantom character into  $w$ , which is somewhere to the left of position  $n - 1$  to show the insertion site in  $w$  (which is a deletion site in  $w'$ ) and into  $w'$  to the right of position  $n + 1$  to show the deletion site in  $w$  (which is an insertion site in  $w'$ ). Let  $\bar{w}$  and  $\bar{w}'$  denote these two strings with the phantom characters. If we examine the last two characters of each string, we will see that only one edit operation occurs in this section. Let  $w_3$  and  $w'_3$  be the strings that result in deleting the last two characters from  $\bar{w}$  and  $\bar{w}'$ . Notice that  $w_3$  is the alternating string of length  $n - 1$  with a phantom character inserted and  $w'_3$  will be a  $(d - 1)$ -edit neighbor of  $w_3$ , since we have only accounted for one of the edits from  $w$  to  $w'$ . This case contributes  $S_{d-1}^{(01)_2^*}(n - 1)$  to  $S_d^{(01)_2^*}(n + 1)$ .

If we sum up the contribution of each case, we have our result. ■

**Examples:** Let  $n = 5$  and  $d = 3$ .

Case 1:

$$w = 010101 \quad w_1 = 01010$$

$$w' = 000011 \quad w'_1 = 00001$$

Notice  $w_1$  and  $w'_1$  are 3 edit neighbors.

Case 2:

$$w = 010101 \quad w_2 = 0101$$

$$w' = 000000 \quad w'_2 = 0000$$

Notice  $w_2$  and  $w'_2$  are 2 edit neighbors.

Case 3:

$$w = 010101 \quad \bar{w} = 010\Box 101 \quad w_3 = 010\Box 1$$

$$w' = 000010 \quad \bar{w}' = 000010\Box \quad w'_3 = 00001$$

Notice  $w_3$  and  $w'_3$  are 2 edit neighbors.

It is obvious that  $S_0^{(g)*}(n) = 1$ , since the only 0-edit neighbor of a string is itself. It is also clear that  $S_1^{(g)*}(n) = (q-1)n$ , since the only way to have a one edit neighbor of the same length is to perform a single substitution and there are  $(q-1)\binom{n}{1} = (q-1)n$  of these.

We also know from Theorem 8 that the number of two edit neighbors of an alternating string over the binary alphabet is

$$\begin{aligned} & \binom{n}{2} + n(k-3) + n_1 + n_k - \sum_{i=2}^n a_i \left( \binom{i}{2} - i + 1 \right) \\ &= \binom{n}{2} + n(n-3) + 1 + 1 - \left( \binom{n}{2} - n + 1 \right) \\ &= \binom{n}{2} + n^2 - 3n + 2 - \binom{n}{2} + n - 1 \\ &= n^2 - 2n + 1 \\ &= (n-1)^2 \end{aligned} \tag{5.1}$$

We can now find a formula for  $S_d^{(01)_2^*}(n)$  for  $d \geq 3$  and  $n \geq 4$ .

**Theorem 16.** For  $3 \leq d < n$  and  $n \geq 4$ ,

$$S_d^{(01)_2^*}(n) = \frac{2^{d-2}(2n-3d+4)(n-d+1)!}{d!(n-2d+2)!}$$

Note that we also have

$$S_d^{(01)_2^*}(n) = \frac{2^{d-2}(2n-3d+4)}{(n-2d+2)!} \binom{n-d+1}{d}$$

*Proof.* Fix  $d$ . We will prove the result by induction on  $n$ . Our base case is  $n = 4$ . Using the code in Appendix B, we notice that

$$S_d^{(01)_2^*}(2) = \begin{cases} 1 & \text{if } d = 0 \\ 2 & \text{if } d = 1 \\ 1 & \text{if } d = 2 \\ 0 & \text{if } d \geq 3 \end{cases}$$

Table 5.2 Edit distance  $d$ -neighbors of 01 and 010

$d$ -neighbors of 01			$d$ -neighbors of 010		
$d = 0$	$d = 1$	$d = 2$	$d = 0$	$d = 1$	$d = 2$
01	00	11	010	000	001
	10			011	100
				110	101
					111

$$S_d^{(01)_2^*}(3) = \begin{cases} 1 & \text{if } d = 0 \\ 3 & \text{if } d = 1 \\ 4 & \text{if } d = 2 \\ 0 & \text{if } d \geq 3 \end{cases}$$

See Table 5.2 for verification.

By Theorem 15, we know that

$$\begin{aligned} S_d^{(01)_2^*}(4) &= \begin{cases} S_3^{(01)_2^*}(3) + 2S_2^{(01)_2^*}(2) & \text{if } d = 3 \\ S_d^{(01)_2^*}(3) + 2S_{d-1}^{(01)_2^*}(2) & \text{if } d \geq 3 \end{cases} \\ &= \begin{cases} 0 + 2(1) & \text{if } d = 3 \\ 0 + 2(0) & \text{if } d \geq 3 \end{cases} \\ &= \begin{cases} 2 & \text{if } d = 3 \\ 0 & \text{if } d \geq 3 \end{cases} \end{aligned}$$

Notice that this means there are two 3-edit neighbors of 0101 and no 4-edit neighbors. This is easily verified (by code in Appendix B), see Table 5.3.

We now verify that the formula holds:

$$\begin{aligned} S_d^{(01)_2^*}(4) &= \frac{2^{d-2}(2(4) - 3d + 4)((4) - d + 1)!}{d!((4) - 2d + 2)!} \\ &= \frac{2^{d-2}(12 - 3d)(5 - d)!}{d!(6 - 2d)!} \end{aligned}$$

Table 5.3 Edit distance  $d$ -neighbors of 0101

$d = 0$	$d = 1$	$d = 2$	$d = 3$
0101	0001	0000	1000
	0100	0010	1110
	0111	0011	
	1101	0110	
		1001	
		1010	
		1011	
		1100	
		1111	

So we have that

$$\begin{aligned}
S_3^{(01)_2^*}(4) &= \frac{2^{3-2}(12 - 3(3))(5 - 3)!}{3!(6 - 2(3))!} \\
&= \frac{(2)(3)(2!)}{3!0!} \\
&= 2
\end{aligned}$$

Thus our base case is proved. Now suppose that for  $i \leq n$ ,

$$S_d^{(01)_2^*}(i) = \frac{2^{d-2}(2i - 3d + 4)(i - d + 1)!}{d!(i - 2d + 2)!}.$$

Notice that

$$\begin{aligned}
2S_{d-1}^{(01)_2^*}(n-1) &= 2 \left( \frac{2^{(d-1)-2}[2(n-1) - 3(d-1) + 4][(n-1) - (d-1) + 1]!}{(d-1)![(n-1) - 2(d-1) + 2]!} \right) \\
&= \frac{2^{d-2}(2n - 3d + 5)(n - d + 1)!}{(d-1)!(n - 2d + 3)!}
\end{aligned}$$

By our recursion, we know that

$$\begin{aligned}
S_d^{(01)_2^*}(n+1) &= S_d^{(01)_2^*}(n) + 2S_{d-1}^{(01)_2^*}(n-1) \\
&= \frac{2^{d-2}(2n-3d+4)(n-d+1)!}{d!(n-2d+2)!} + \frac{2^{d-2}(2n-3d+5)(n-d+1)!}{(d-1)!(n-2d+3)!} \\
&= \frac{2^{d-2}(n-d+1)!}{(d-1)!(n-2d+2)!} \left[ \frac{2n-3d+4}{d} + \frac{2n-3d+5}{n-2d+3} \right] \\
&= \frac{2^{d-2}(n-d+1)!}{(d-1)!(n-2d+2)!} \left[ \frac{(2n-3d+4)(n-2d+3) + d(2n-3d+5)}{d(n-2d+3)} \right] \\
&= \frac{2^{d-2}(n-d+1)!}{d!(n-2d+3)!} [(2n-3d+4)(n-2d+3) + d(2n-3d+5)] \\
&= \frac{2^{d-2}(n-d+1)!}{d!(n-2d+3)!} [2n^2 - 5nd + 10n + 3d^2 - 12d + 12] \\
&= \frac{2^{d-2}(n-d+1)!}{d!(n-2d+3)!} [(2n-3d+6)(n-d+2)] \\
&= \frac{2^{d-2}(2n-3d+6)(n-d+2)!}{d!(n-2d+3)!} \\
&= \frac{2^{d-2} [2(n+1) - 3d + 4] [(n+1) - d + 1]!}{d! [(n+1) - 2d + 2]!}
\end{aligned}$$

Thus, the equation holds for  $n+1$  and our proof is complete. ■

The formula in Theorem 16 is not intuitive. We went through many steps to arrive at the formula and proved the recursion in Theorem 15 from the formula. We began by running the code in Appendix A with generator  $g = 01$  and got the results in Table 5.4. We then fixed  $d$  and looked at each column of the table as a function of  $n$ . Running polynomial regressions on

Table 5.4 Output of code in Appendix A using generator  $g = 01$ 

$n$	$d = 0$	$d = 1$	$d = 2$	$d = 3$	$d = 4$	$d = 5$	$d = 6$	$d = 7$	$d = 8$
2	1	2	1						
3	1	3	4	0					
4	1	4	9	2	0				
5	1	5	16	10	0	0			
6	1	6	25	28	4	0	0		
7	1	7	36	60	24	0	0	0	
8	1	8	49	110	80	8	0	0	0
9	1	9	64	182	200	56	0	0	0
10	1	10	81	280	420	216	16	0	0
11	1	11	100	408	784	616	128	0	0
12	1	12	121	570	1344	1456	560	32	0
13	1	13	144	770	2160	3024	1792	288	0
14	1	14	169	1012	3300	5712	4704	1408	64
15	1	15	196	1300	4840	10032	10752	4992	640

the data for columns corresponding to  $d = 0, 1, 2, 3, 4, 5, 6$  yielded the following formulas with correlation coefficient  $R^2 = 1$ :

$$S_0^{(01)_2^*}(n) = 1$$

$$S_1^{(01)_2^*}(n) = n$$

$$S_2^{(01)_2^*}(n) = n^2 - 2n + 1$$

$$S_3^{(01)_2^*}(n) = \frac{1}{3}(2n^3 - 15n^2 + 37n - 30)$$

$$S_4^{(01)_2^*}(n) = \frac{1}{3}(n^4 - 16n^3 + 95n^2 - 248n + 240)$$

$$S_5^{(01)_2^*}(n) = \frac{1}{45}(6n^5 - 165n^4 + 1800n^3 - 9735n^2 + 26094n - 27720)$$

$$S_6^{(01)_2^*}(n) = \frac{2}{45}(n^6 - 42n^5 + 730n^4 - 6720n^3 + 34549n^2 - 94038n + 105840)$$

After factoring the formulas we have

$$\begin{aligned}
S_0^{(01)_2^*}(n) &= 1 \\
S_1^{(01)_2^*}(n) &= n \\
S_2^{(01)_2^*}(n) &= (n-1)^2 \\
S_3^{(01)_2^*}(n) &= \frac{1}{3}(n-2)(2n-5)(n-3) \\
S_4^{(01)_2^*}(n) &= \frac{1}{3}(n-3)(n-4)^2(n-5) \\
S_5^{(01)_2^*}(n) &= \frac{1}{45}(n-4)(n-5)(2n-11)(n-6)(n-7) \\
S_6^{(01)_2^*}(n) &= \frac{2}{45}(n-5)(n-6)(n-7)^2(n-8)(n-9)
\end{aligned}$$

We noticed a pattern in the polynomials shown in Table 5.5. The first term of the product is

Table 5.5 Patterns in Factored Regression Polynomials

$d$	$S_d^{(01)_2^*}(n)$			
2	$\binom{n-1}{1}$	$\times \frac{n-1}{1}$	$=$	$1! \binom{n-1}{1} \times \frac{2n-2}{2}$
3	$2 \binom{n-2}{2}$	$\times \frac{2n-5}{3}$	$=$	$2! \binom{n-2}{2} \times \frac{2n-5}{3}$
4	$6 \binom{n-3}{3}$	$\times \frac{n-4}{3}$	$=$	$3! \binom{n-3}{3} \times \frac{2n-8}{6}$
5	$24 \binom{n-4}{4}$	$\times \frac{2n-11}{15}$	$=$	$4! \binom{n-4}{4} \times \frac{2n-11}{15}$
6	$120 \binom{n-5}{5}$	$\times \frac{2(n-7)}{45}$	$=$	$5! \binom{n-5}{5} \times \frac{2n-14}{45}$

clearly  $(d-1)! \binom{n-d+1}{d-1}$ . The second term of the product can be found by observing that the numerator is the arithmetic sequence  $2n-3d+4$  and the denominator  $a_d$  can be found by the recursion  $a_d = \frac{d}{2}a_{d-1}$ . This recursion has closed form  $a_d = \frac{d!}{2^{d-2}}$ . Therefore the second term in our product is

$$\frac{2n-3d+4}{d!/2^{d-2}} = \frac{2^{d-2}(2n-3d+4)}{d!}$$

This gives us our formula

$$S_d^{(01)_2^*}(n) = \frac{2^{d-2}(2n-3d+4)(n-d+1)!}{d!(n-2d+2)!}$$

This formula did not give us any intuition about why it would be true, so we pressed on to find a recursion by taking the difference:

$$\begin{aligned} \Delta S_d^{(01)_2^*}(n) &= S_d^{(01)_2^*}(n+1) - S_d^{(01)_2^*}(n) \\ &= \frac{2^{d-2}[2(n+1)-3d+4][(n+1)-d+1]!}{d![(n+1)-2d+2]!} - \frac{2^{d-2}(2n-3d+4)(n-d+1)!}{d!(n-2d+2)!} \\ &= \frac{2^{d-2}(2n-3d+6)(n-d+2)!}{d!(n-2d+3)!} - \frac{2^{d-2}(2n-3d+4)(n-d+1)!}{d!(n-2d+2)!} \\ &= \frac{2^{d-2}(n-d+1)!}{d!(n-2d+2)!} \left[ \frac{(2n-3d+6)(n-d+2)}{n-2d+3} - (2n-3d+4) \right] \\ &= \frac{2^{d-2}(n-d+1)!}{d!(n-2d+2)!} \left[ \frac{(2n-3d+6)(n-d+2)}{n-2d+3} - \frac{(2n-3d+4)(n-2d+3)}{n-2d+3} \right] \\ &= \frac{2^{d-2}(n-d+1)!}{d!(n-2d+3)!} [(2n-3d+6)(n-d+2) - (2n-3d+4)(n-2d+3)] \\ &= \frac{2^{d-2}(n-d+1)!}{d!(n-2d+3)!} [2nd+5d-3d^2] \\ &= \frac{2^{d-2}(n-d+1)!}{d!(n-2d+3)!} [d(2n+5-3d)] \\ &= \frac{2^{d-2}(2n+5-3d)(n-d+1)!}{(d-1)!(n-2d+3)!} \\ &= 2 \left( \frac{2^{d-3}(2n-3d+5)(n-d+1)!}{(d-1)!(n-2d+3)!} \right) \\ &= 2 \left( \frac{2^{(d-1)-2}[2(n-1)-3(d-1)+4][(n-1)-(d-1)+1]!}{(d-1)![(n-1)-2(d-1)+2]!} \right) \\ &= 2S_{d-1}^{(01)_2^*}(n-1) \end{aligned}$$

This leads immediately to the recursion

$$S_d^{(01)_2^*}(n+1) = S_d^{(01)_2^*}(n) + 2S_{d-1}^{(01)_2^*}(n-1)$$

## CHAPTER 6. Unanswered Questions

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

First, there is no generalization of the formula given in Theorem 16, which gives the number of edit distance  $d$  neighbors of a strictly alternating string, to a  $q$ -ary alphabet.

Secondly, Theorem 16 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 thesis 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 thesis. The material in Chapter 4 suggests that such constructions are likely not to be as elegant as those for the Hamming metric. Nevertheless, this is an essentially untouched area.

## CHAPTER 7. Possible Applications

There are numerous possible applications of the work previously shown. We mentioned the first, barcoding ESTs to track their origin, in Chapter 1. Three other possible applications are listed here.

The second possible application is sequencing genomes with many repetitive elements. Corn is an excellent example of one such genome. Suppose that an organism, like corn, with many repeated elements, needs to be sequenced. The fragment-and-assemble strategy used in humans impractical. The repeated sequences make the assembly problem too hard. Instead of fragmenting, the DNA is broken into big chunks that are put in bacteria as bacterial artificial chromosomes (BACs). These are then fragmented and assembled and then the resulting assemblies are assembled. This is a divide-and-conquer strategy for getting around the challenge of the repeated sequences. However, this gives one sequencing project per BAC with all the management overhead. Sequencing pooled BAC fragments would be cheaper just as sequencing pooled ESTs was cheaper. One solution to this problem is to barcode the BACs.

A third possible application is tagging or watermarking intellectual property. When a new or engineered gene is novel, it is its own marker. As the engineering of DNA goes on, there will be more cases where two very similar genes are owned by different people or when one public-sector gene is modified in many different ways by different groups. Associating an error tolerant tag with that gene that encodes a serial number may be valuable. If the gene is reproduced *in vivo* then multiple sequences with high error correction value may be needed. To watermark an organism the sequences would be inserted not with the engineered construct, but somewhere else. That way, tests for the watermark barcode and the gene would have different amplifying sequences. This permits the verification of an organism that has a

modification without giving away the amplification sequences for the modification itself.

The fourth possible application is performing mutational studies. An entire distance  $k$  code must, because it is a code, incorporate many sequences. Inserting an entire code into *E. coli*, a different one for each barcode, would permit study of the mutation rate of many, many sequences at the insertion point. In this case the code would be a form of experimental design for presenting many sequences.

**APPENDIX A. Code to calculate edit neighbors of repetitive strings**

```
#include <iostream>
#include <cstring>

using namespace std;

#define Lmax 16

int ED(int a[Lmax], int b[Lmax]);
int n2[Lmax+1];
int c[Lmax+1];
int L;

main(int argc, char **argv){

    if(argc<2){
        cout << "a.out string" << endl;
        return(0);
    }

    int q[Lmax];
    int r[Lmax];
    int i,j,k;
    n2[0]=1;
```

```

for(i=0;i<Lmax;i++)
    n2[i+1]=n2[i]+n2[i];
for(i=0;i<Lmax;i++)
    q[i]=argv[1][i%strlen(argv[1])]-'0';;
for(L=2;L<Lmax;L++){
    for(i=0;i<=L;i++)
        c[i]=0;
    for(i=0;i<n2[L];i++){
        for(j=0;j<L;j++){
            if(n2[j]&i)
                r[j]=1;
            else
                r[j]=0;
        }
        c[ED(q,r)]++;
    }
    cout << endl;
    for(i=0;i<=L;i++)
        cout << c[i] << " ";
    cout << endl;
}
}

int M[Lmax+1][Lmax+1];

int ED(int a[Lmax],int b[Lmax]){ //computes edit distance

int i,j;

```

```
int q,r,s;

for(i=0;i<=L;i++){
    M[i][0]=-i;
    M[0][i]=-i;
}
for(i=1;i<=L;i++){
    for(j=1;j<=L;j++){
        q=M[i-1][j-1];
        if(a[i-1]!=b[j-1])
            q--;
        r=M[i-1][j]-1;
        s=M[i][j-1]-1;
        if(s>q)
            q=s;
        if(r>q)
            q=r;
        M[i][j]=q;
    }
}
return(-M[L][L]);
}
```

**APPENDIX B. Code to compute all neighbors of a given string**

```
#include <cstring>
#include <iostream>
#include <math>

using namespace std;

int editd(char[],char[], int);
char increment(char[]);

int main(){
    char x[20];
    char y[20];
    int dist[20];
    int k;
    int l;
    int s;
    int d;
    int t;
    cout << "Enter a binary string: " << endl;
    cin.getline(x,20,'\n');
    l=strlen(x);
    strcpy(y,x);
```

```

for (k=0;k<1;k++)
    y[k]='0';
cout << "The edit distance between " << x << " and " << y << " is ";
d=editd(x,y,1);
for (k=0;k<=1;k++)
    dist[k]=0;
for (t=0;t<=1;t++){
    if (t==d)
        dist[t]+=1;
}
for (s=0;s<(pow(2,1)-1);s++){
    increment(y);
    cout << "The edit distance between " << x << " and " << y << " is ";
    d=editd(x,y,1);
    for (t=0;t<=1;t++){
        if (t==d)
            dist[t]+=1;
    }
}
for (t=0;t<=1;t++){
    cout << "The number of d=" << t << " neighbors is " << dist[t] << endl;
}
}

int editd(char x[], char y[], int l){ //computes edit distance
    int m=1;                // initialize m to length of x
    int n=1;                // initialize n to length of y
    int a[20][20];         // initialize array a to store dynamic values

```

```

int i;                // initialize counters i and j
int j;
int p;                // initialize p the alignment cost
int q;                // initialize q to keep track of max
int d;                // initialize d the edit distance
for (i=0;i<=m;i++){  // fills in the first row of the matrix
    a[i][0]=-i;
}
for (j=0;j<=n;j++){  // fills in the first column of the matrix
    a[0][j]=-j;
}
for (i=1;i<=m;i++){  // nested loops to fill in entries of matrix
    for (j=1;j<=n;j++){
        if (x[i-1]==y[j-1])
            p=0;
        else
            p=-1;
        q=max(a[i-1][j]-1, a[i][j-1]-1);
        a[i][j]=max(a[i-1][j-1]+p,q); //gives best possible value
    }
}
d=-a[m][n];          // d=edit distance between x and y
cout << d << endl;
return d;
}

char increment(char y[]){
    int a=strlen(y)-1;

```

```
while (a>0 && y[a]=='1'){
    y[a]='0';
    a--;
}
if (a>=0)
    y[a]='1';
return 0;
}
```

**BIBLIOGRAPHY**

- [1] D. ASHLOCK, *Optimization and Modeling with Evolutionary Computation*, Springer-Verlag, New York, 2005.
- [2] D. ASHLOCK, L. GUO, AND F. QIU, *Greedy closure evolutionary algorithms*, in Proceedings of the 2002 Congress on Evolutionary Computation, IEEE Neural Networks Council, IEEE, 2002, p. 1296.
- [3] N. L. BIGGS, *Discrete Mathematics*, Clarendon Press, Oxford, 1985.
- [4] A. E. BROUWER, *Bounds on the size of linear codes*, in Handbook of Coding Theory Vol. 1, V. Pless and W. Huffman, eds., Elsevier Science BV, New York, 1998, p. 297.
- [5] A. E. BROUWER, A. M. COHEN, AND A. NEUMAIER, *Distance-Regular Graphs*, A Series of Modern Surveys in Mathematics, Springer-Verlag, Berlin, 1989.
- [6] R. A. BRUALDI, *Introductory Combinatorics*, Prentice Hall, Edgewood Cliffs, NJ, second ed., 1992.
- [7] I. K. EVANS, *Evolutionary Algorithms for Vertex Cover*, vol. 1447 of Lecture Notes in Computer Science, Springer-Verlag, New York, 1998.
- [8] T. W. HUNGERFORD, *Algebra*, Graduate Texts in Mathematics, Springer-Verlag, New York, 1974.
- [9] D. B. WEST, *Introduction to Graph Theory*, Prentice Hall, Upper Saddle River, NJ, second ed., 2001.

## 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 and patience throughout the writing of this thesis. His insights and encouragements kept me motivated and helped me complete my graduate education.

I would also like to thank Doug Ray for his inspiration in finding the formula in Theorem 16. He was also invaluable as a proofreader of this document. Thanks are also due to Gordan Royal for the reference (5).