AN ABSTRACT OF THE THESIS OF

Kenny Barrese for the degree of Honors Baccalaureate of Science of Mathematics and Honors Baccalaureate of Arts of Philosophy presented on June 4, 2008. Title: Decoding Methods for Linear Codes.

Abstract approved:

_____

Mary Flahive

It is necessary to encode data when transmitting over a noisy channel in order for errors to be detected and corrected. List decoding algorithms provide all code words within a specified distance of a received word in order to be sufficiently robust for cases when two or more code words are equidistant from a received word. This paper details a probabilistic method to obtain multiple close code words, motivated by list decoding methods, for linear codes over fields of two or three elements. It employs a variation on the LLL-Algorithm for lattice reduction that allows the LLL-Algorithm to determine small elements in a vector space. Although it fails to return all code words within a specified distance of a received word, the method is sufficiently robust to provide some information if a received word is equidistant to multiple code words.

Key Words: Coding Theory, List Decoding, Linear Codes, LLL-Algorithm

Corresponding e-mail address: barresek@gmail.com

Decoding Methods for Linear Codes

by

Kenny Barrese

A PROJECT

submitted to

Oregon State University

University Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Mathematics
Honors Baccalaureate of Arts in Philosophy

Presented June 4, 2008
Commencement June 2008

Honors Baccalaureate of Science in Mathematics and Honors Baccalaureate of Arts in Philosophy project of Kenny Barrese presented on June 4, 2008.

APPROVED:

Mentor, representing Mathematics

Committee Member, representing Mathematics

Committee Member, representing Electrical Engineering and Computer Science

---

Chair, Department of Mathematics

Dean, University Honors College

I understand that my project will become part of the permanent collection of Oregon State University, University Honors College. My signature below authorizes release of my project to any reader upon request.

Kenny Barrese, Author

TABLE OF CONTENTS

Decoding Methods for Linear Codes

1. Introduction

When transmitting information, there is often the possibility that some of the data will become corrupted, or changed, during the transmission. In light of this, it is often advantageous to encode messages so that errors in transmission can be detected and, ideally, corrected. Data so encoded includes additional information that is used to identify errors and sometimes correct the messages. One common method for encoding data utilizes linear transformations, represented by matrices. Here we develop an alternative method for reducing matrices in order to facilitate decoding linear codes. This reduction method employs the adaptation of the LLL-Algorithm given in [1]. The original LLL-Algorithm was designed by Lenstra, Lenstra, and Lovatz [4]. Although the intent was to develop a true method for list decoding general linear codes, this algorithm does not attain that level of generality. This algorithm is limited to codes with an alphabet of either $\mathbb{Z}_2$ or $\mathbb{Z}_3$. Additionally, unlike list decoding, this algorithm does not provide a list of all code words within a specified diameter. However, the algorithm presented here is sufficiently robust to handle instances when the received word lies close to two distinct code words. This thesis assumes the reader is familiar with the basics of Linear Algebra, as some general principles of linear algebra will be used without proof or explanation. For background information, we recommend *Fundamentals of Error-Correcting Codes* by Huffman and Pless [3].

1.1. **Codes.**

Definition 1.1. *A* m-n Code *over* $\mathbb{F}$ *is a subset of* $\mathbb{F}^n$ *with* $|C| = |\mathbb{F}^m|$, *where* $\mathbb{F}$ *denotes an arbitrary, fixed finite field and* $|S|$ *denotes the cardinality of set S.*

As said above, in order for the code to detect errors, some extra information must be added. This makes it necessary that $n > m$. Let us denote the code $\mathcal{C} \subseteq \mathbb{F}^n$. It is implied that the message to be encoded comes from $\mathbb{F}^m$, which is called the *message space*.

In coding theory the terms “vector” and “word” are used interchangeably. Thus, the original vector is termed the *message word*, the encoded version of our message word is a *code word*, and the vector we finally receive is the *received word*. By convention, let us denote the message word by **m**, the code word by **c**, and the received word by **r**. Because $m < n$, $\mathcal{C}$ is a proper subset of $\mathbb{F}^n$. If we receive a word $\mathbf{r} \in \mathbb{F}^n$, such that $\mathbf{r} \notin \mathcal{C}$, we know there was an error in transmission.

1.2. **Decoding.**

Definition 1.2. *The* Hamming Distance *between two words in $\mathcal{C}$ is the number of positions at which they contain different entries.*

Lemma 1.3. *For field $\mathbb{F}$ and $k \in \mathbb{N}$ the Hamming Distance is a metric on $\mathbb{F}^k$.*

Proof. Because the Hamming Distance between two words is the number of entries in which they differ, the Hamming Distance must be non-negative and, furthermore, can be zero if and only if the two words are identical. If v_1 differs from v_2 in j entries, then v_2 must differ from v_1 in j entries, thus the Hamming Distance is symmetric. Finally, if v_1 differs from v_2 in j entries, and v_2 differs from v_3 in k entries, then v_1 cannot possibly differ from v_1 in more than $j+k$ entries, so the Hamming Distance satisfies the triangle inequality. Thus, the Hamming Distance fulfills the three requirements of a metric.

□

Because $\mathcal{C}$ is a subset of $\mathbb{F}^n$, the Hamming Distance is a metric on $\mathcal{C}$. Let us denote the Hamming Distance between two words, $u, v \in \mathcal{C}$, by d(u,v). For example, let $\mathbb{F} = \mathbb{Z}_2$, $n = 3$, $u =< 1,0,1 >$ and $v =< 1,0,0 >$, then d(u,v) = 1 because u and v differ only in the third position. Note that d(u,v) is the number of non-zero entries in $u - v$.

Definition 1.4. *The* Hamming Weight *of a word is its Hamming Distance from the zero vector.*

Note that the weight of a word is the number of non-zero entries that it contains. Because the Hamming Distance is a metric on $\mathbb{F}^k$, the Hamming Weight is a norm. The most wide spread method of error correction is *Nearest Neighbor decoding*. Nearest Neighbor decoding works on the principle that the codeword at minimal Hamming Distance from the received word has the highest probability of being the true message word. For example, if the received word is a code word, no error correcting needs to be considered. In cases where two or more code words have the identical minimal Hamming Distance from the received word Nearest Neighbor decoding fails to correct the error.

Another method of decoding received words with errors is *list decoding*. List decoding produces a list of all code words within a fixed Hamming Distance d of a received word. This paper examines my algorithm for decoding arbitrary Linear Codes over the fields $\mathbb{Z}_2$ and $\mathbb{Z}_3$, which attempts to replicate some of the advantages of list decoding.

2. Linear Codes

Definition 2.1. *A* m-n Linear Code *is an m-n Code where $\mathcal{C}$ is an m-dimensional subspace of $\mathbb{F}^n$.*

Elements of codes are *n*-tuples that we sometimes regard as row vectors and sometimes as column vectors as necessary for matrix multiplication, determined by context.

Definition 2.2. *Two m-n Linear Codes, $\mathcal{C}_1$ and $\mathcal{C}_2$, are* equivalent *if and only if there exists an n-dimensional square matrix* **M** *such that* **M** *has exactly one non-zero entry in every row and column, and $\mathcal{C}_1 \cdot \mathbf{M} = \mathcal{C}_2$, which indicates that when every element in the set $\mathcal{C}_1$ is right multiplied by* **M** *the resulting set is $\mathcal{C}_2$.*

Because there are many matrices which fit the requirements given for matrix **M**, there are many codes that are equivalent to any given code. This definition provides little direct information about the consequences of two codes being equivalent, as is often the case with formal definitions. Since matrix **M** is square, we know that vectors in $\mathcal{C}_1$ and $\mathcal{C}_2$ all have the same number of entries. Also, since **M** can be obtained by permuting the rows of a diagonal matrix, the only operations that it can perform on elements in $\mathcal{C}_1$ are permutations and scaling of entries in the vectors of $\mathcal{C}_1$.

2.1. **The Parity Check and Generator Matrices.**

Definition 2.3. *The* Parity Check Matrix *of an m-n Linear Code is a matrix which, when right multiplied by a code word, yields the zero vector, but, when right multiplied by a non-code word, or a vector in $\mathbb{F}^n$ but not in $\mathcal{C}$, yields a vector with at least one non-zero entry.*

For any Linear Code $\mathcal{C}$, we know that $\mathcal{C}$ is the kernel of a linear transformation because $\mathcal{C}$ is a subspace of $\mathbb{F}^n$. Thus, a Parity Check Matrix must exist for any given $\mathcal{C}$. Because we right multiply the Parity Check Matrix by elements of $\mathbb{F}^n$, the Parity Check Matrix must have *n* columns. We can determine the number of rows necessary by remembering that we

desire the Parity Check Matrix to have an m-dimensional kernel. In order for the nullity to be at most m the Parity Check Matrix must have no fewer than $(n-m)$ rows; otherwise, $\mathcal{C}$ would not have the same dimension as the kernel. Even if we construct a Parity Check Matrix with more than $(n-m)$ rows, only $(n-m)$ can be linearly independent else the nullity will be less than m. We will assume that all Parity Check Matrices contain the minimum necessary rows, thus it is assumed that the Parity Check Matrices with which we deal are $(n-m)\times n$ matrices.

Definition 2.4. *The* syndrome *of a received word* $\mathbf{r}$ *is the product of left multiplying* $\mathbf{r}$ *by the Parity Check Matrix.*

If the syndrome is zero, then the received word is indeed a code word, and a non-zero syndrome indicates that an error has occurred.

Definition 2.5. *A* Generator Matrix *of a m-n Linear Code is any matrix representation of a linear transformation such that* $\mathbf{G}:\mathbb{F}^m \longrightarrow \mathcal{C}$.

Because both $\mathbb{F}^m$ and $\mathcal{C}$ are vector spaces over the field $\mathbb{F}$ with m basis vectors, any matrix which maps a basis of $\mathbb{F}^m$ onto a basis of $\mathcal{C}$ will be a Generator Matrix. Note that our message word is an m-tuple. Because $\mathbf{m}\in\mathbb{F}^m$, and code words are in $\mathbb{F}^n$, the Generator Matrix must be an $m\times n$ matrix. Because $\mathcal{C}$ and $\mathbb{F}^m$ are both vector spaces over $\mathbb{F}$ and have the same dimension, they must have the same number of elements. We want every message word in $\mathbb{F}^m$ to correspond to a single codeword in $\mathcal{C}$, so we can think of the relation between the message space and the code space as a function from $\mathbb{F}^m$ into $\mathcal{C}$. Additionally, code words must correspond to a unique message word if decoding is to be possible, so the function must be one-to-one. Since $\mathbb{F}^m$ and $\mathcal{C}$ have the same number of elements, a one-to-one function from $\mathbb{F}^m$ into $\mathcal{C}$ must be a bijection.

Definition 2.6. *Two Generator Matrices are* equivalent *if they correspond to equivalent m-n Linear Code. That is, they are functions* $\mathbf{G}_1:\mathbb{F}^m \longrightarrow \mathcal{C}_1$ *and* $\mathbf{G}_2:\mathbb{F}^m \longrightarrow \mathcal{C}_2$ *such that* $\mathcal{C}_1$ *is equivalent to* $\mathcal{C}_2$.

Definition 2.7. *When the Generator Matrix is written in the form:* $\left[\mathbf{I}_m \mid \mathbf{A}\right]$ *, where* $\mathbf{I}_m$ *is the* $m \times m$ *identity matrix and* $\mathbf{A}$ *is an arbitrary* $m \times (n-m)$ *matrix, it is said to be in* Standard Form.

Theorem 2.8. *Every Generator Matrix can be converted into an equivalent Generator Matrix in standard form through elementary row operations and permuting columns.*

Proof. Because codes are the image of a 1-1 function from the message space, the only element in the kernel of the Generator Matrix must be the zero vector. This implies that the rows of the Generator Matrix form a linearly independent set. Thus, when the Generator Matrix is reduced through Gauss-Jordan reduction, every row will contain a leading one in a pivot column, with zeroes above and below the 1 in that column. By permuting columns to collect the pivot columns on the left of the Generator Matrix, the standard form is obtained.

Performing an elementary row operation on a Generator Matrix produces an equivalent matrix. Any finite sequence of elementary row operations can be encoded into an elementary matrix $\mathbf{E}$ by performing the operations in the same order on $\mathbf{I}_m$. If $\mathbf{G}_2$ is the result of performing a sequence of elementary row operations collected in $\mathbf{E}$ on $\mathbf{G}_1$, then $\mathbf{G}_2 = \mathbf{E}\mathbf{G}_1$. For an arbitrary Generator Matrix $\mathbf{G}$, $\mathbf{G} : \mathbb{F}^m \longrightarrow \mathcal{C}$ by $\forall \mathbf{f} \in \mathbb{F}^m$, $\mathbf{G}(\mathbf{f}) = \mathbf{f}\mathbf{G}$. Thus $\mathbf{G}_1 : \mathbb{F}^m \longrightarrow \mathcal{C}_1$ and $\mathbf{G}_2 : \mathbb{F}^m \cdot \mathbf{E}^{-1} \longrightarrow \mathcal{C}_1$. Here the notation $\mathbb{F}^m \cdot \mathbf{E}^{-1}$ again refers to the set obtained by right multiplying every element in $\mathbb{F}^m$ by the matrix $\mathbf{E}^{-1}$. The matrix $\mathbf{E}$ must be invertible because its Gauss-Jordan reduced form is $\mathbf{I}_m$. Since $\mathbf{E}$ is an invertible, m-dimensional matrix, $\mathbb{F}^m \cdot \mathbf{E}^{-1} = \mathbb{F}^m$. Both $\mathbf{G}_1$ and $\mathbf{G}_2$ map $\mathbb{F}^m$ to $\mathcal{C}_1$, thus $\mathbf{G}_1$ and $\mathbf{G}_2$ are equivalent.

Permuting columns of a Generator Matrix $\mathbf{G}_1$ results in a Generator Matrix $\mathbf{G}_2$ for an equivalent code. Because the dot product of the message word $\mathbf{m}$ and the i-th column of the Generator Matrix produces the i-th position in the code word, permuting columns of $\mathbf{G}$

results in a permutation of the entries of vectors in $\mathcal{C}$. Thus the codes produced by $\mathbf{G}_1$ and $\mathbf{G}_2$ are equivalent. Multiplying a column of a Generator Matrix by a non-zero element of $\mathbb{F}$ will produce a Generator Matrix for an equivalent code by similar reasoning.

Because elementary row operations and interchanging columns preserve code equivalency, given an arbitrary Generator Matrix for an m-n Linear Code, we can always obtain a Generator Matrix for an equivalent m-n Linear Code in Standard form.

□

3. List Decoding

Definition 3.1. *Given an arbitrary received word* $\mathbf{r} \in \mathbb{F}^n$*, an algorithm that determines if* $\mathbf{r} \in \mathcal{C}$ *is a method of* Error Detection.

Definition 3.2. *Given a received word* $\mathbf{r}$ *which we know not to be a code word, an algorithm that provides the codeword(s) most likely to yield* $\mathbf{r}$ *due to an error in transmission is a method of* Error Correction.

Traditionally, decoding is performed according to the nearest neighbor principle. This principle says that the code word at closest Hamming Distance to a given received word is most likely to be the actual code word that was sent. Thus, if a code word is received, since it is at distance zero from a code word, itself, it "corrects" to itself.

In order to be assured of correcting d errors when decoding using the nearest neighbor principle, it is necessary that no vector in $\mathbb{F}^n$ be within distance d of two elements of $\mathcal{C}$. Thus the minimum Hamming Distance between two code words, or the *designed diameter* must be at least $2d+1$. This makes creating codes analogous to a sphere packing problem, wherein we are attempting to fit the most spheres, each around to a code word, into the limited space provided by $\mathbb{F}^n$. However, if we allow spheres to overlap just a little, we can obtain a significant increase in the number of errors we can correct, while admitting only a few vectors that lie within the overlapping spheres of two or more code words ([2], page 6).

List decoding is an attempt to correct for more errors by increasing d until $2d$ is larger than the designed diameter, which is analogous to enlarging the spheres about the code words to allow overlap. List decoding provides a unique code word when possible and still can handle the worst-case scenario, when a received word lies within two spheres. List decoding provides a list of all code words within a specified distance, usually d, of a given received word. This is the closed ball of Hamming radius d around the received word $\mathbf{r}$, $\mathbf{B}_{\mathbf{r}}(d)$. Because small increases in d will result in few overlaps between spheres, most often

this list will contain one element. Unlike nearest neighbor decoding however, list decoding enables an algorithm to handle the unlikely worst case scenarios.

Decoding m-n Linear Codes makes use of some of the properties of the Parity Check Matrix. Let $\mathbf{S}$ denote a Parity Check Matrix for the code $\mathcal{C}$. Because $\mathcal{C}$ is the kernel of $\mathbf{S}$, if $\mathbf{r} \in \mathcal{C}$ then $\mathbf{S} \cdot \mathbf{r} = \mathbf{0}$, otherwise the product will have at least one non-zero entry. It is apparent that $\mathbf{S}$ provides a simple method of error detection. However, a method of error correction is desirable. To this end, we note that the syndrome contains further useful information. We can think of our received word $\mathbf{r}$ as the sum of our code word, $\mathbf{c}$, and an error word, $\mathbf{e}$:

$$\mathbf{r} = \mathbf{c} + \mathbf{e}$$

Since multiplying by a matrix is a linear operation and $\mathbf{c} \in \mathcal{C}$:

$$\mathbf{S} \cdot \mathbf{r} = \mathbf{S} \cdot \mathbf{c} + \mathbf{S} \cdot \mathbf{e} = 0 + \mathbf{S} \cdot \mathbf{e} = \mathbf{S} \cdot \mathbf{e}$$

Thus any vector $\mathbf{e} \in \mathbb{F}^n$ such that $\mathbf{S} \cdot \mathbf{e} = \mathbf{S} \cdot \mathbf{r}$ allows us to generate a code word , by $\mathbf{r} - \mathbf{e} = \mathbf{c}$. As noted above, when using nearest neighbor decoding, the goal is to acquire the error word with the lowest Hamming Weight possible. When using a list decoding method however, it is desirable to find all error words with Hamming Weight less than d, in order to account for the possibility of a received word being close to multiple code words.

4. LLL-ALGORITHM

The LLL-Algorithm was originally put forward by A. K. Lenstra, H. W. Lenstra Jr., and L. Lovasz to factor polynomials with rational coefficients [4]. Although the LLL-Algorithm is probabilistic in nature, it functions well in all but a few, rare, aberrant conditions. Because it efficiently returns a basis for a given lattice which is both nearly orthogonal and minimized with respect to the standard Euclidean norm, the LLL-Algorithm is implemented in a wide variety of applications. Here orthogonal vectors refer to vectors where the standard inner product, or dot product, of any distinct pair is zero. The lattice generated by a given basis $< \mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_k >$ is every element that can be expressed as an integral combination of basis elements, $a_1\mathbf{b}_1 + a_2\mathbf{b}_2 + \ldots + a_k\mathbf{b}_k$. A lattice contains only integral valued coordinates when the basis vectors of a lattice contain only integral values.

One useful application of the LLL-Algorithm, developed in [1], is computing the minimum distance, the *designed diameter*, between two code words in a linear code over $\mathbb{Z}_2$ or $\mathbb{Z}_3$. Because linear codes are closed under addition, the difference between any two code words will itself be a code word. Thus, finding the minimum distance of a linear code corresponds to finding a code word with minimal Hamming Weight.

Because the LLL-Algorithm probabilistically produces shortest, or minimal, lattice basis with respect to the standard Euclidean norm, we must limit our consideration to linear codes where $\mathbb{F}$ is either $\mathbb{Z}_2$ or $\mathbb{Z}_3$. Both fields can be expressed in terms of $\{0, 1, -1\}$, with $1 = -1$ in $\mathbb{Z}_2$, thus the Hamming Weight of vector $\mathbf{v} =< v_1, v_2, \ldots, v_n >$ is $\sum_{i=1}^{n} |v_i|$, which is the number of non-zero entries in a vector, while the Euclidean norm is $\sqrt{\sum_{i=1}^{n} v_i^2}$. Because $\forall i,\ v_i \in \{0, 1, -1\}$, $|v_i| = v_i^2$, the Hamming Weight of $\mathbf{v}$ is the square of its Euclidean norm. Therefore, vectors that are short in the Euclidean norm will also be short in the Hamming norm.

Although [1] presents their adaptation of the LLL-Algorithm as a method for determining the minimum distance of a linear code, it solves the equivalent problem of finding elements in a vector space with minimal Hamming Weight, which is the purpose for which we desire to implement their method. For a psuedo-code implementation of the LLL-Algorithm from Algorithm 7.6.4 in [1], please refer to the first Appendix.

4.1. **Computing Small Elements in a Vector Space.** Although the convention is to perform LLL reduction upon the columns of a matrix, in [1] and here it is more intuitive to think of performing the LLL reduction upon the rows, which is what we shall do.

The matrix upon which we will implement the LLL-Algorithm is as follows:

$$\mathbf{B} = \begin{bmatrix} N \cdot \mathbf{G} & \mathbf{I}_m \\ \\ N \cdot q\mathbf{I}_n & \mathbf{0} \end{bmatrix}$$

where $\mathbf{G}$ is an $m \times n$ matrix whose rows are a basis for our vector space over finite field $\mathbb{F}$, N is a large integer, q is the number of elements in our field, $\mathbf{I}_m$ and $\mathbf{I}_n$ are identity matrices of the indicated dimension, and $\mathbf{0}$ is the $n \times m$ zero matrix. Thus matrix $\mathbf{B}$ is an $(n+m)$ square matrix.

Now we shall explain why matrix $\mathbf{B}$ must be constructed as it is. Although we wish to find the shortest basis for span($\mathbf{G}$) in order to determine the designed diameter of the corresponding m-n Linear Code, we cannot simply implement the LLL-Algorithm upon the rows of $\mathbf{G}$. This is because the LLL-Algorithm is designed to minimize the size of basis elements of lattices in $\mathbb{Z}^n$, whereas we are attempting to minimize the basis elements of spaces over $\mathbb{Z}_2$ or $\mathbb{Z}_3$. In order to force the LLL-Algorithm to perform the reduction mod q, we place $q\mathbf{I}_n$ beneath $\mathbf{G}$. When we think about the lattice points that this adds, we see they are elements of the form $\mathbf{v} =< v_1q, v_2q, \ldots, v_nq >$. Returning to our problem, this allows

the algorithm to reduce any vector $\mathbf{g} =< g_1, g_2, \ldots, g_n > \in$ span($\mathbf{G}$) to an equivalent vector $\mathbf{g}' =< g'_1, g'_2, \ldots, g'_n >$ with each $|g'_i| \leq (q/2)$ and integral by subtracting the projection of $\mathbf{g}$ onto the dimensions spanned by $a\mathbf{I}_n$ from $\mathbf{g}$. Because we have subtracted off a multiple of q, $g'_i \equiv g_i \mod q$, which mimics modular arithmetic.

The LLL-Algorithm requires the basis of a lattice as its input; thus, all the rows in our input matrix must be linearly independent. Once we include $q\mathbf{I}_n$ beneath $\mathbf{G}$, it is sufficient to append $\mathbf{I}_m$ to the right of $\mathbf{G}$, with the zero matrix alongside the right of $\mathbf{I}_n$, which makes the rows a basis of an $(m+n)$-dimensional lattice. However, the inclusion of the new m columns creates the possibility that the LLL-Algorithm might output a basis with larger than necessary values in the first n columns in order to avoid creating a large value in the last m columns, because the algorithm minimizes the length of the entire row. This is undesirable because, ultimately, the basis for $\mathbf{G}$ will occur in the first n columns, and the data contained in the last m columns will be extraneous. In order to prevent the LLL-Algorithm from returning lattice points with unnecessarily large entries in the first n columns, we scale these columns by large integer N to force minimal entries in these rows. This completes the rationale for the construction of matrix $\mathbf{B}$, upon which we perform the LLL-Algorithm.

As mentioned above, once the algorithm terminates, we may discard the last m columns. We are left with an $(m+n) \times n$ matrix. This matrix will contain m rows devoid of non-zero entries. This occurs because every non-zero entry in the first n columns of $\mathbf{B}$ is a multiple of N, which is large. Matrix $\mathbf{B}$ contains n columns that are scaled by N, corresponding to n-dimensions of the span of $\mathbf{B}$. Thus the reduced basis must contain n linearly independent rows containing at least one non-zero element in the first n columns, otherwise the reduced basis would not span the same lattice as the rows of matrix $\mathbf{B}$. For the remaining m rows to contain a non-zero multiple of N is unnecessary, because the dimensions represented by the first n columns can be spanned by n rows. In fact, the m rows must not contain a

non-zero multiple of N, because the Euclidean norm of a row containing a non-zero multiple of N is significantly larger than one that does not, and the LLL-Algorithm produces a basis minimized with respect to the Euclidean norm. The rows corresponding to the zero vector provide no solutions, so they are discarded. The remaining n rows are the vectors in $N \cdot \text{span}(\mathbf{G})$ over $\mathbb{Z}$ that have minimal Euclidean norm, thus we rescale by $1/N$ in order to obtain vectors in span($\mathbf{G}$) over $\mathbb{Z}_2$ or $\mathbb{Z}_3$ with minimal Hamming Weight.

5. List Decoding of Linear Codes

As we have seen, when we consider Linear Codes, decoding a received word $\mathbf{r}$ is equivalent to finding a vector $\mathbf{F} \in \mathbb{F}^n$ such that $\mathbf{S} \cdot \mathbf{e} = \mathbf{S} \cdot \mathbf{r}$, where $\mathbf{S}$ is the Parity Check Matrix, because $\mathbf{r} - \mathbf{e} \in \mathcal{C}$. Let $\mathbf{p} = \mathbf{S} \cdot \mathbf{r}$ denote the syndrome, then possible error vectors will be solutions to the following augmented matrix:

$$\left[\mathbf{S} \quad | \quad \mathbf{p}\right]$$

Because $\mathbf{S}$ has n rows and only $n-m$ columns, this is an underdetermined system, which will provide multiple possible error vectors. However, the traditional method for solving such linear systems, Gauss-Jordan elimination, will present the solution set in an undesirably disorganized format, lacking useful algebraic structure, because $\mathbf{p}$ is not the zero vector. Thus we employ an alternative algorithm to find the possible error vectors as elements of a vector space, a more structured construction. Since our system of linear equations is non-homogeneous, the solutions will not form a vector space, however, we can find solutions to most of the rows as a vector space. Because we utilize the LLL-Algorithm, we must constrain $\mathbb{F}$ to either $\mathbb{Z}_2$ or $\mathbb{Z}_3$. We begin with the Parity Check Matrix $\mathbf{S}$ in the following format:

$$\mathbf{S} = \begin{bmatrix} s_{1,1} & s_{1,2} & \dots & s_{1,n} \\ s_{2,1} & s_{2,2} & \dots & s_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n-m,1} & s_{n-m,2} & \dots & s_{n-m,n} \end{bmatrix}$$

5.1. **Row Reduction.** The first task is to perform row reduction on the augmented matrix. Gauss-Jordan elimination proceeds from the leftmost column to the right, establishing pivot ones in each row, then using them to induce zeroes in the remaining places of that column. In contrast, this algorithm starts on the far right column. Use the three types of elementary

row operations on this column, which is **p** written as a column vector, to obtain a one as the first entry of the column, with only zeroes beneath it, this is possible because **p** is not the zero vector. At this point, the new augmented matrix, $\mathbf{S}'$ will be of the form:

$$\mathbf{S}' = \left[\begin{array}{cccc|c} s'_{1,1} & s'_{1,2} & \cdots & s'_{1,n} & 1 \\ s'_{2,1} & s'_{2,2} & \cdots & s'_{2,n} & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ s'_{n-m,1} & s'_{n-m,2} & \cdots & s'_{n-m,n} & 0 \end{array}\right]$$

Here $s'_{i,j}$ indicate the new entries of matrix $\mathbf{S}'$ after the elementary row operations.

5.2. **Forming a Homogeneous System.** At this point it is useful to note that, ignoring the first row, the matrix $\mathbf{S}'$ corresponds to a homogeneous system of linear equations. This allows us to ignore the first row and solve the remaining homogeneous system of linear equations, which we shall denote by $\mathbf{S}''$. The solutions will form a subspace of $\mathbb{F}^n$ because they are exactly the kernel of $\mathbf{S}''$.

$$\mathbf{S}'' = \left[\begin{array}{cccc|c} s''_{2,1} & s''_{2,2} & \cdots & s''_{2,n} & 0 \\ s''_{3,1} & s''_{3,2} & \cdots & s''_{3,n} & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ s''_{n-m,1} & s''_{n-m,2} & \cdots & s''_{n-m,n} & 0 \end{array}\right]$$

5.3. **Gauss-Jordan Reduction.** Use Gauss-Jordan reduction on $\mathbf{S}''$ to obtain the row-reduced form. Permute columns to collect the pivot columns at the far left, forming a $(n-m-1)$-dimensional Identity matrix on the left side of matrix $\mathbf{S}''$. Keep track of these permutations in an elementary column operation matrix **E** obtained by performing the exact same permutation on a n-dimensional Identity matrix. We will denote the row-reduced form of $\mathbf{S}''$ by **T**.

$$\mathbf{T} = \left[\begin{array}{cccc|cccc|c} 1 & 0 & \dots & 0 & t_{1,n-m} & t_{1,n-m+1} & \dots & t_{1,n} & 0 \\ 0 & 1 & \dots & 0 & t_{2,n-m} & t_{2,n-m+1} & \dots & t_{2,n} & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & t_{n-m-1,n-m} & t_{n-m-1,n-m+1} & \dots & t_{n-m-1,n} & 0 \end{array}\right]$$

5.4. **Determine a Basis.** Now we can easily find a basis for the kernel of **T**. The nullity of **T** is $m+1$, which is to be expected because we have n columns and only $n-m-1$ rows, all of which are linearly independent. Let us note the structure of **T**, $\mathbf{T} = \left[\begin{array}{c|c} \mathbf{I}_{n-m-1} \ : \ \mathbf{T}' & \mathbf{0} \end{array}\right]$, this implies that the first $n-m-1$ entries of a solution to the system of equations represented by $\mathbf{Tx} = \mathbf{0}$ will be dependent upon the last $m+1$ entries. Construct a new matrix **U** by taking the transpose of the part of matrix **T** denoted by $\mathbf{T}'$ above, then appending an $m+1$-dimensional Identity matrix to the right of this. The rows of matrix $\mathbf{U} = \left[\mathbf{T}'^{T} \ : \ \mathbf{I}_{m+1}\right]$, shown below, form a basis for the solution space.

$$\mathbf{U} = \begin{bmatrix} -t_{1,n-m} & -t_{2,n-m} & \dots & -t_{n-m-1,n-m} & 1 & 0 & \dots & 0 \\ -t_{1,n-m+1} & -t_{2,n-m+1} & \dots & -t_{n-m-1,n-m+1} & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ -t_{1,n} & -t_{2,n} & \dots & -t_{n-m-1,n} & 0 & 0 & \dots & 1 \end{bmatrix}$$

Note that **U** has $m+1$ rows, corresponding to the nullity of **T**.

The vector space spanned by the rows of **U** is the kernel of $\mathbf{S}''$, which implies that any vector in span(**U**) will satisfy all the equations, except the first, in the system of linear equations represented by $\mathbf{S}'$. This means that any solution to the original system of linear equations $\left[\mathbf{S} \ | \ \mathbf{p}\right]$, multiplied by **E** from step 3, to take into account necessary column permutations, is an element of span(**U**). In fact, necessary and sufficient conditions for $v \in \mathbb{F}^n$ to be a solution to **S** are as follows:

(1) $$v \in span(\mathbf{U})$$

$$v \cdot S_1 = 1, \tag{2}$$

where S_1 denotes the first row of matrix $\mathbf{S}'$. This must be the case, because condition (1) is equivalent to being a solution for every row of $\mathbf{S}'$ except the first row and any vector where $v \cdot S_1 = 1$ is a solution to the first row of $\mathbf{S}'$. Thus, vectors that meet both conditions are in the intersection between the set containing the solution to the first row, and the set containing the solutions all of the other rows simultaneously.

Additionally, any vector that meets the first condition can be easily made to meet the second condition, provided that $v \cdot S_1 \neq 0$. Suppose that $v \cdot S_1 = k \in \mathbb{F}$ such that $k \neq 0$. In this case, $k^{-1}v \cdot S_1 = 1$ because the dot product is a bilinear operator. We also know that $k^{-1}v \in span(\mathbf{U})$ because span(**U**) is a vector space and $k^{-1}v$ is a scalar multiple of an element we know to be in span(**U**). Thus, any vector that meets condition (1) and is not orthogonal to S_1 will be a solution to the system of linear equations. However, we search not just for a solution to the system of linear equations, but a solution that also has a low Hamming Weight. To that end we employ the LLL-Algorithm.

5.5. **LLL-Algorithm.** The LLL-Algorithm takes a set of linearly independent vectors as an input, then outputs the shortest, under the standard Euclidean norm, nearly orthogonal basis for the lattice for which the input elements form a basis. Using the LLL-Algorithm to find vectors of low Hamming Weight requires that we restrict $\mathbb{F}$ to either $\mathbb{Z}_2$ or $\mathbb{Z}_3$. Since these two fields only have 0, 1, and possibly -1 as elements, minimizing lattice points with regard to the standard Euclidean norm, which is the purpose of the LLL-Algorithm, is equivalent to minimizing the Hamming Weight of the corresponding vector. Applying the LLL-Algorithm as modified in section 4.1 to the rows of **U** yields good candidates for the smallest vectors, with respect to the Hamming Weight, in span(**U**). We will then check the second condition for these vectors, knowing that they must fulfill the first condition because they are in span(**U**).

Finally, once vectors of low Hamming Weight which satisfy both conditions are obtained, they must be right multiplied by the elementary matrix **E** that we obtained during step three, Gauss-Jordan Reduction. This is done in order to correct any permutations that occurred during the algorithm's third step, wherein columns were permuted in order to collect the identity matrix on the left side of matrix **T**. Afterward we have a list of short error vectors from which possible code words can be obtained, by subtracting error vectors from the received word **r**

6. First Example

For this example:

$$\mathbb{F} = \mathbb{Z}_3, \qquad \mathbf{S} = \begin{bmatrix} 1 & 0 & 0 & 1 & -1 & -1 & 1 \\ 1 & -1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & -1 & 0 & -1 & -1 & 0 \\ 1 & 0 & 1 & -1 & 1 & 1 & 0 \end{bmatrix} \quad \text{and} \quad \mathbf{p} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \end{bmatrix}$$

6.1. **Row Reduction.** When we augment **S** with **p** we get:

$$\left[\begin{array}{ccccccc|c} 1 & 0 & 0 & 1 & -1 & -1 & 1 & 1 \\ 1 & -1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & -1 & 0 & -1 & -1 & 0 & 0 \\ 1 & 0 & 1 & -1 & 1 & 1 & 0 & -1 \end{array}\right]$$

We add the first row to the fourth row in order to induce zeros in all positions of the syndrome column except the first, thus obtaining:

$$\mathbf{S}' = \left[\begin{array}{ccccccc|c} 1 & 0 & 0 & 1 & -1 & -1 & 1 & 1 \\ 1 & -1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & -1 & 0 & -1 & -1 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{array}\right]$$

6.2. **Forming a Homogeneous System.** Temporarily removing the first row to generate a Homogeneous system of linear equations we obtain:

$$\mathbf{S}'' = \left[\begin{array}{ccccccc|c} 1 & -1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & -1 & 0 & -1 & -1 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{array}\right]$$

6.3. **Gauss-Jordan Elimination.** Performing Guass-Jordan elimination upon matrix $\mathbf{S}''$ yields:

$$\mathbf{T} = \left[\begin{array}{ccccccc|c} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & -1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & -1 & 0 \end{array}\right]$$

without requiring any permutation of columns, thus $\mathbf{E} = \mathbf{I}_7$.

6.4. **Determine a Basis.** If matrix $\mathbf{U}$ begins with $-\mathbf{T}'^T$, where $\mathbf{T} = \left[\begin{array}{c|c} \mathbf{I} : \mathbf{T}' & \mathbf{0} \end{array}\right]$, followed by a four-dimensional Identity matrix, each row of matrix $\mathbf{U}$ will be a unique solution to the system of linear equations given by matrix $\mathbf{T}$. Thus we construct the matrix $\mathbf{U}$ as follows:

$$\mathbf{U} = \begin{bmatrix} -1 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

6.5. **LLL-Algorithm.** Here we set large integer $N = 10$, remember that $q = 3$. Thus

$$\mathbf{B} = \begin{bmatrix} -10 & 0 & -10 & 10 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 10 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 & 10 & 0 & 0 & 0 & 1 & 0 \\ 0 & -10 & 10 & 0 & 0 & 0 & 10 & 0 & 0 & 0 & 1 \\ 30 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 30 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 30 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 30 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 30 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 30 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 30 & 0 & 0 & 0 & 0 \end{bmatrix}$$

We use Maple's LLL function to perform the LLL-Algorithm on **B**, this code is included in the second Appendix, and obtain the following matrix:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3 \\ -10 & 0 & -10 & 10 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 10 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 & 10 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 10 & 0 & 10 & 0 & 10 & 0 & 1 & 0 & 1 \\ 0 & 10 & 10 & 0 & 0 & -10 & 10 & 0 & 0 & -1 & 1 \\ -10 & 0 & 0 & 10 & 10 & 0 & 10 & 1 & 1 & 0 & 1 \\ 20 & 10 & 10 & 10 & 0 & 0 & -10 & 1 & 0 & 0 & -1 \end{bmatrix}$$

Discarding the last four columns, which are extraneous, and the first four rows, which do not correspond to possible non-zero solutions, and factoring out $N = 10$ yields:

$$\begin{bmatrix} -1 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & -1 & 1 \\ -1 & 0 & 0 & 1 & 1 & 0 & 1 \\ -1 & 1 & 1 & 1 & 0 & 0 & -1 \end{bmatrix}$$

where rows two and three correspond to possible solutions with Hamming Weight 2, rows one and four correspond to solutions of weight 3, rows five and six correspond to solutions of weight 4, and row seven is a possible solution of weight 5. We check condition (2) when our possible error vector $\mathbf{e}'$ equals row two:

$$\mathbf{e}' \cdot S_1 = < 0,1,0,0,1,0,0 > \cdot < 1,0,0,1,-1,-1,1 > = -1$$

Since the dot product is -1 and -1 = -1^{-1}, we multiply $\mathbf{e}'$ by -1 to obtain an error vector:

$$\mathbf{e} = < 0,-1,0,0,-1,0,0 >$$

To finish with this vector, we would multiply **e** by **E** in order to correct the column switching from the third step, except in this case $\mathbf{E} = \mathbf{I}_7$. Because it has the same weight, we proceed to check the possible error vector represented by the third row:

$$\mathbf{e}' \cdot S_1 = < 0,1,0,0,0,1,0 > \cdot < 1,0,0,1,-1,-1,1 > = -1$$

Again, we must multiply the vector by -1 to obtain an error vector:

$$\mathbf{e} = < 0,-1,0,0,0,-1,0 >$$

Our algorithm terminates and returns two vectors with minimum Hamming Weight, $\{< 0,-1,0,0,-1,0,0 >, < 0,-1,0,0,0,-1,0 >\}$.

7. Second Example

Let us now consider an example of this algorithm over a finite field that is not $\mathbb{Z}_2$ or $\mathbb{Z}_3$. We will start with the Parity Check Matrix and the syndrome and continue until it is time to employ the LLL-Algorithm, which cannot be utilized in this example because minimizing the Euclidean norm no longer minimizes the Hamming Distance. For this example:

$$\mathbb{F} = \mathbb{Z}_{11}, \qquad \mathbf{S} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 4 & 9 & 5 & 3 & 3 & 5 \\ 1 & 8 & 5 & 9 & 4 & 7 & 2 \end{bmatrix} \quad \text{and} \quad \mathbf{p} = \begin{bmatrix} 2 \\ 6 \\ 3 \\ 10 \end{bmatrix}$$

7.1. **Row Reduction.** When we augment $\mathbf{S}$ with $\mathbf{p}$ to obtain $\mathbf{S}'$ we get:

$$\left[\begin{array}{ccccccc|c} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 6 \\ 1 & 4 & 9 & 5 & 3 & 3 & 5 & 3 \\ 1 & 8 & 5 & 9 & 4 & 7 & 2 & 10 \end{array}\right]$$

We multiply the first row by 6, which happens to be 2^{-1}, to obtain a 1 in the first row position of the syndrome. Using this 1 as a pivot, we zero out the lower positions in the syndrome to obtain:

$$\mathbf{S}' = \left[\begin{array}{ccccccc|c} 6 & 6 & 6 & 6 & 6 & 6 & 6 & 1 \\ 9 & 10 & 0 & 1 & 2 & 3 & 4 & 0 \\ 5 & 8 & 2 & 9 & 7 & 7 & 9 & 0 \\ 7 & 3 & 0 & 4 & 10 & 2 & 8 & 0 \end{array}\right]$$

7.2. **Forming a Homogeneous System.** Temporarily removing the first row to generate a Homogeneous system of linear equations we obtain:

$$\mathbf{S}'' = \left[\begin{array}{ccccccc|c} 9 & 10 & 0 & 1 & 2 & 3 & 4 & 0 \\ 5 & 8 & 2 & 9 & 7 & 7 & 9 & 0 \\ 7 & 3 & 0 & 4 & 10 & 2 & 8 & 0 \end{array}\right]$$

7.3. **Gauss-Jordan Elimination.** Performing Gauss-Jordan reduction on matrix $\mathbf{S}''$ yields:

$$\mathbf{T} = \left[\begin{array}{ccccccc|c} 1 & 0 & 0 & 7 & 4 & 9 & 9 & 0 \\ 0 & 1 & 0 & 7 & 10 & 10 & 0 & 0 \\ 0 & 0 & 1 & 3 & 6 & 10 & 4 & 0 \end{array}\right]$$

Again, no column permutation is required to isolate the leading 1's, so $\mathbf{E} = \mathbf{I}_7$.

7.4. **Determine a Basis.** Constructing matrix **U** as before, we obtain:

$$\mathbf{U} = \begin{bmatrix} 4 & 4 & 8 & 1 & 0 & 0 & 0 \\ 7 & 1 & 5 & 0 & 1 & 0 & 0 \\ 2 & 1 & 1 & 0 & 0 & 1 & 0 \\ 2 & 0 & 7 & 0 & 0 & 0 & 1 \end{bmatrix}$$

7.5. **LLL-Algorithm Replacement.** The rows of this matrix suggest three possible errors with Hamming Weight 4, rows one, two, and three, and one with Hamming Weight 3, row four. Having obtained matrix **U** we would apply the LLL-Algorithm to the rows of **U** to ascertain if errors with less weight existed, corresponding to rows in span(**U**) with fewer non-zero entries, if **U** were over $\mathbb{Z}_2$ or $\mathbb{Z}_3$. This is not possible because the Euclidean norm is not equivalent to the Hamming norm when elements of absolute value greater than 1 are included. However, there is an alternative method to consider. When we examine **U** we notice that $\mathbf{U} = \begin{bmatrix} \mathbf{A} & \mathbf{I}_4 \end{bmatrix}$, since the right side is an Identity matrix, adding a multiple of one row to another will invariably introduce another non-zero entry to that row, which is undesirable since we are attempting to minimize the Hamming Weight, or number of non-zero entries, of our error vectors. Thus, when we perform elementary row operations on the vectors, the addition of non-zero elements in the last four columns must be offset by

gains made in the first three columns. Therefore, we are looking for a linear combination of two rows that yields new zeros in two or three of the first three columns. If we make the first entry in each row 1, we obtain the following matrix:

$$\begin{bmatrix} 1 & 1 & 2 & 3 & 0 & 0 & 0 \\ 1 & 8 & 7 & 0 & 8 & 0 & 0 \\ 1 & 6 & 6 & 0 & 0 & 1 & 0 \\ 1 & 0 & 9 & 0 & 0 & 0 & 1 \end{bmatrix},$$

which indicates that we cannot induce a zero the first and second or third entry simultaneously. In order to test all possibilities, we set the second entry in each of the first three rows to one, to determine if any combination of rows will produce zeros the second two entries. We obtain the matrix:

$$\begin{bmatrix} 1 & 1 & 2 & 3 & 0 & 0 & 0 \\ 7 & 1 & 5 & 0 & 1 & 0 & 0 \\ 2 & 1 & 1 & 0 & 0 & 2 & 0 \\ 1 & 0 & 9 & 0 & 0 & 0 & 1 \end{bmatrix},$$

showing that there is no combination of two rows which will zero the second and third entry. The fourth row has Hamming Weight 3, which can see is the best we can do. Let us examine the possible error vector $\mathbf{e}'$ obtained by the fourth row. In this case, $\mathbf{e}' =< 1,0,9,0,0,0,1 >$. We know that $\mathbf{e}' = \mathbf{e}$ if and only if $\mathbf{e}' \cdot S_1 = 1$.

$$\begin{aligned} \mathbf{e}' \cdot S_1 &=< 1,0,9,0,0,0,1 > \cdot < 6,6,6,6,6,6,6 > \\ &= (1+9+1) \times 6 \\ &= 0 \times 6 \\ &= 0 \end{aligned}$$

Since $\mathbf{e}' \cdot \mathbf{e} = 0$, we cannot obtain an error vector from $\mathbf{e}'$. For another possible error vector, we must accept a vector of weight 4, let us simply use the first row:

$$< 1,1,2,3,0,0,0 > \cdot S_1 = 9$$

so, since 5 = 9^{-1}, $5 \cdot < 1,1,2,3,0,0,0 > = < 5,5,10,4,0,0,0 >$ corresponds to a possible error vector of weight 4. Since we performed no column switches, $\mathbf{E} = \mathbf{I}_7$. Thus, this is the vector we would subtract from our received word in order to obtain a code word that may correspond to our received word. Note however, that since the sum of the elements in rows two and three does not equal zero, they too will produce possible error vectors of weight 4.

8. Conclusion

8.1. **Overview.** In this paper we developed an algorithm for providing code words in a linear codes over $\mathbb{Z}_2$ or $\mathbb{Z}_3$ that are likely to correspond to a given received word with error, an algorithm which essentially produces multiple "best guess" possible error vectors with minimal Hamming Weight. Although this algorithm does not produce a true list decoding method, as it will not provide all possible code words within a specified distance of a given received word, because it is designed to return multiple error vectors the algorithm does not necessarily fail when the spheres around code words begin to overlap.

The method by which these error vectors are obtained involves an alternative reduction method for underdetermined, non-Homogeneous systems of linear equations. This reduction method allows us to consider solutions to the equations in the system, other than the first, as a subspace of $\mathbb{F}^n$. This allows us to take advantage of the lattice-like structure of the subspace in order to employ the LLL-Algorithm.

The LLL-Algorithm works recursively on a lattice basis in order to determine the basis with shortest possible basis vectors, with respect to the standard Euclidean norm. In $\mathbb{Z}_2$ and $\mathbb{Z}_3$ these will be the vectors with lowest Hamming Weight, because a vector's Hamming Weight is the square of its Euclidean length as shown in [1]. Once short candidates for solutions to the system of linear equations are obtained, a dot product is sufficient to test whether they satisfy the first equation, and thus whether they are possible error vectors.

8.2. **Further Consideration.** Using the LLL-Algorithm limits $\mathbb{F}$ to either $\mathbb{Z}_2$ or $\mathbb{Z}_3$. It would be desirable to invent some different lattice basis which will allow the LLL-Algorithm to find vectors of low Hamming Weight over general finite fields. Lacking that, further investigation of matrix reduction may develop a method to generate short error vectors directly from matrix **U**, which would eliminate the need to employ the LLL-Algorithm. A method such as I envision would involve a rigorous generalization of the alternative to the

LLL-Algorithm employed in the second example.

When considering solutions to the underdetermined, non-homogeneous system of equations, solutions to the rows excepting the top row, the system represented by $\mathbf{S}'$, form a vector space, for which we can easily determine a basis. Because solutions to the first row require a non-zero dot product with the first row, they are, in a sense, the vectors of the subspace that are not orthogonal to the first row. It seems that there should be a method to use this geometric structure to impose structure upon the solutions to the entire system of linear equations. Unfortunately, my investigation into these questions is limited by time, which grows all too short. Thus I must leave this enterprise as is, for the moment. The process has been illuminating and thought provoking, but must come to an end here.

REFERENCES

[1] Betten, Anton; Braun, Michael; Fripertinger, Harald; Kerber, Adalbert; Kohnert, Axel; and Wassermann, Alfred. *Error-Correcting Linear Codes*. New York: Springer. 2006.

[2] Guruswami, Venkateasan. *Algorithmic Results in List Decoding*. Boston: Now Publishers Inc. 2007.

[3] Huffman, W. Cary and Vera Pless. *Fundamentals of Error-Correcting Codes 2nd ed*. New York: Cambridge University Press. 2003.

[4] Lenstra, A. K.; Lenstra, H. W.; and Lovasz, L. "*Factoring Polynomials with Rational Coefficients*." Mathematische Annalen 261. (1982). 515-534.

E-mail address: barresek@onid.orst.edu

Appendices

Appendix I

In the following algorithm we are reducing the basis $(b^{(0)}, b^{(1)}, \ldots, b^{(m-1)})$. The Gram-Schmidt orthogonal basis obtained from $(b^{(0)}, b^{(1)}, \ldots, b^{(m-1)})$ is $(\hat{b}^{(0)}, \hat{b}^{(1)}, \ldots, \hat{b}^{(m-1)})$. The Gram-Schmidt coefficents are μ_{kj}, where $\mu_{kj} = \frac{<b^{(k)}, b^{(j)}>}{<\hat{b}^{(j)}, \hat{b}^{(j)}>}$. In the sixth line, $\pi_k(b^{(k)})$ denotes $\hat{b}^{(k)}$ and $\pi_k(b^{(k+1)})$ is $\hat{b}^{(k+1)} + \mu_{k+1,k}\hat{b}^{(k)}$. Thus the sixth line is comparing the Euclidean length of the k-th basis vector projected upon the space perpendicular to the first k basis vectors, $(b^{(0)}, b^{(1)}, \ldots, b^{(k-1)})^{\perp}$, with the Euclidean length of the next basis vector projected upon the same space.

(1) Let $\delta \in \mathbb{R}$ with $1/4 < y < 1$.

(2) **Set** $k := 0$.

(3) **do**

(4) 1.**for** $j = 0, \ldots, k-1$

(5) **replace** $b^{(k)}$ **by** $b^{(k)} - \lfloor \mu_{kj} \rceil b^{(j)}$

(6) 2.**if** $\delta \parallel \pi_k(b^{(k)}) \parallel^2 > \parallel \pi_k(b^{(k+1)}) \parallel^2$ **then**

(7) **interchange** $b^{(k+1)}$ and $b^{(k)}$

(8) **update** $\hat{b}^{(k+1)}$, $\hat{b}^{(k)}$ and μ

(9) **set** $k := \max(k-1, 0)$

(10) **else**

(11) **set** $k := k+1$

(12) **until** $k = m-1$

Appendix II

The Maple code for the LLL-Algorithm performed on page 16:

```
> with(IntegerRelations):
> LLL([[-10,0,-10,10,0,0,0,1,0,0,0],
> [0,10,0,0,10,0,0,0,1,0,0],
> [0,10,0,0,0,10,0,0,0,1,0],
> [0,-10,10,0,0,0,10,0,0,0,1],
> [30,0,0,0,0,0,0,0,0,0,0],
> [0,30,0,0,0,0,0,0,0,0,0],
> [0,0,30,0,0,0,0,0,0,0,0],
> [0,0,0,30,0,0,0,0,0,0,0],
> [0,0,0,0,30,0,0,0,0,0,0],
> [0,0,0,0,0,30,0,0,0,0,0],
> [0,0,0,0,0,0,30,0,0,0,0]], 'integer');
```

$$[[0, 0, 0, 0, 0, 0, 0, -3, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, -3, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, -3, 0],$$
$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -3], [-10, 0, -10, 10, 0, 0, 0, 1, 0, 0, 0],$$
$$[0, 10, 0, 0, 10, 0, 0, 0, 1, 0, 0], [0, 10, 0, 0, 0, 10, 0, 0, 0, 1, 0],$$
$$[0, 0, 10, 0, 10, 0, 10, 0, 1, 0, 1], [0, 10, 10, 0, 0, -10, 10, 0, 0, -1, 1],$$
$$[20, 0, 0, 10, 10, 0, 10, 1, 1, 0, 1], [-10, 10, 10, 10, 0, 0, -10, 1, 0, 0, -1]]$$