CONSTRUCTING POLYNOMIALS OVER FINITE FIELDS

by

MARK STEVEN STAMP, B.S.

A THESIS

IN

MATHEMATICS

Submitted to the Graduate Faculty
of Texas Tech University in
Partial Fulfillment of
the Requirements for
the Degree of

MASTER OF SCIENCE

Approved

Accepted

December, 1988

73
1988
No. 175
Cop. 2

ACKNOWLEDGEMENTS

I would like to thank the members of my committee, William Gustafson, David Gilliam, and especially my advisor, Clyde F. Martin, who was more than generous with his time and support throughout my work on this thesis.

Much of the work presented in Chapter IV was motivated by a letter from Lajos Takacs of Case-Western Reserve University. I am very grateful to Dr. Takacs for his insights into a difficult problem. I would also like to thank John Lund at Montana State University for giving me the opportunity to present a talk based on this thesis at the Computation and Control conference held in Bozeman this past summer.

Finally, my lovely wife, 杜倩倩, deserves a very special thanks for her constant love and encouragement. It seems that our love for each other grows stronger with each passing day. More precisely, letting t equal time and $\ell(t)$ be our love for each other at time t , we have $\lim_{t \rightarrow \infty} \ell(t) = \infty$.

CONTENTS

ACKNOWLEDGEMENTS	ii
ABSTRACT	iv
LIST OF FIGURES	v
I. INTRODUCTION	1
II. THE ALGORITHM	3
III. CONSTRUCTING POLYNOMIAL MAPS ON V^n	7
3.1 Applying Algorithm 2.1	7
3.2 Polynomial Systems for a Linear Recurring Sequence	9
3.3 A Class of Interpolating Polynomials	16
3.4 The Lexicographic Map	19
IV. OUTPUT SEQUENCES IN $\mathbf{F}_2$	22
V. CONCLUSION	27
REFERENCES	28
APPENDIX	
FORTRAN AND C IMPLEMENTATIONS OF ALGORITHM 2.1 ...	29

ABSTRACT

In this paper we present an algorithm for constructing a polynomial system corresponding to a given directed labeled graph. Three specific classes of polynomial systems are considered in detail. Then we briefly consider some desirable properties for output sequences and we show how to construct a polynomial system that will produce a given output sequence of zeros and ones of length 2^n .

LIST OF FIGURES

1.1	A Cryptosystem	1
3.1	A Labeled Graph on V^3	8
3.2	Graph on V^2	10
3.3	Linear Shift Register	11
3.4	Graph of $P(\vec{x})$ on V^4	12
3.5	Graph of $T(\vec{x})$ on V^4	12
3.6	Graph of $h(\vec{x})$	12
3.7	Graph on V^3	17
3.8	Lexicographic Map on V^3	20
4.1	Lexicographic Map on V^4	25
4.2	Output Sequence on V^4	25

CHAPTER I

INTRODUCTION

A typical system used in cryptography is illustrated in Figure 1.1, [6]. The text which is to be transmitted is first converted into a sequence of binary digits using some type of algebraic encoding. This sequence is referred to as the plaintext. The process of encryption (or enciphering) is accomplished by adding to the plaintext (mod 2) the bit sequence produced by a pseudo-random number generator. After transmitting the encrypted text, the plaintext is recovered by adding (mod 2) the same pseudo-random sequence. This process of recovering the plaintext is usually called deciphering. Of course the entire procedure relies on the fact that for two binary digits a and b , addition modulo 2 (or equivalently, a binary exclusive or) has the property that $a + b + b = a$.

Figure 1.1: A Cryptosystem

Typically, the way such cryptosystems are broken is that a piece of the plaintext falls into the hands of an enemy cryptanalyst who then compares the plaintext with the encrypted text, thereby recovering a segment of the pseudo-random sequence. Using this information, the cryptanalyst attempts to recover more of the pseudo-random sequence, hence recovering more of the original plaintext.

The process just described is usually referred to as plaintext attack. The goal then in designing a cyptosystem is to construct a system which is as resistant as possible to plaintext attack. Obviously the security or insecurity of any cryptosystem of the type in Figure 1.1 depends primarily on the "randomness" of the pseudo-random number generator.

Our current work deals with the classification and construction of pseudo-random number generators for use in cryptography. In this paper we consider pseudo-random sequences in terms of directed labeled graphs and their corresponding polynomial systems. This approach is taken since in [1] it is proved that with every directed labeled graph satisfying two "observability" conditions, there is an associated class of equivalent observable polynomials. In other word, these labeled graphs give a complete invariant for polynomial systems of the type we will be considering.

Chapter II contains an algorithm for constructing a polynomial which corresponds to a given directed labeled graph. In Chapter III we construct several interesting polynomial mappings and consider their corresponding graphs. Finally, in Chapter IV we briefly consider the question of what constitutes a "good" pseudo-random sequence (for the purposes of cryptography) and we show how to construct a polynomial system, using some results from Chapter III, which will produce any desired output sequence.

CHAPTER II

THE ALGORITHM

Let $\mathbf{F}_2$ be the finite field $\{0, 1\}$ and let V^n be the n dimensional vector space over $\mathbf{F}_2$. In this chapter we present an algorithm, due to Martin, [1], for constructing a polynomial map from V^n to V^n that corresponds to a given directed labeled graph.

The following notation will simplify the derivation and statement of the algorithm. For $\vec{x} \in V^n$ define $|\vec{x}|$ to be the number of nonzero elements of $\vec{x}$. Use $\vec{x}$ to partition V^n by letting $P_k = \{\vec{x} \in V^n : |\vec{x}| = k\}$. The lexicographic ordering orders the elements of V^n "alphabetically," where the alphabet is taken to be 0 and 1. For example, the lexicographic ordering of V^3 is $(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1)$. Now we can use the lexicographic ordering to define a partial order on P_k , which we denote by $(P_k, \prec_k)$. Also, for $a, b \in V^n$ we say $a < b$ if

- 1) $a \in P_k, b \in P_m$ with $k < m$ and
- 2) $a_i = 1$ implies $b_i = 1$.

Considering the example of V^3 again, we have that $P_0 = \{(0, 0, 0)\}$, $P_1 = \{(0, 0, 1), (0, 1, 0), (1, 0, 0)\}$, $P_2 = \{(0, 1, 1), (1, 0, 1), (1, 1, 0)\}$, $P_3 = \{(1, 1, 1)\}$ and $(0, 0, 1) < (1, 0, 1)$ but $(1, 0, 0) \not< (0, 1, 1)$.

Observe that any function $T : V^n \rightarrow V^n$ has the form

$$T(\vec{x}) = (T_1(\vec{x}), T_2(\vec{x}), \dots, T_n(\vec{x})),$$

where $T_i : V^n \rightarrow \mathbf{F}_2$. Define $\vec{x}^\delta = x_1^{\delta_1} x_2^{\delta_2} \cdots x_n^{\delta_n}$ where $\delta = (\delta_1, \delta_2, \dots, \delta_n)$. Then letting a_δ^i be the coefficient of $\vec{x}^\delta$, T_i can be written as

$$T_i(\vec{x}) = \sum_{k=0}^n \sum_{\delta \in P_k} a_\delta^i \vec{x}^\delta.$$

Now if we let

$$H_k^i(\vec{x}) = \sum_{\delta \in P_k} a_\delta^i \vec{x}^\delta,$$

then

$$T_i(\vec{x}) = \sum_{k=0}^n H_k^i(\vec{x}).$$

The observation that $|\delta| \geq |\vec{x}|$ implies

$$\vec{x}^\delta = \begin{cases} 1 & \text{if } \vec{x} = \delta \\ 0 & \text{otherwise} \end{cases}$$

gives us:

Lemma 2.1 If $\vec{x} \in P_m$ then $T_i(\vec{x}) = \sum_{k=0}^{m-1} H_k^i(\vec{x}) + a_{\vec{x}}^i$.

Also, observe that if $\vec{x} \in P_m$ and $\delta \in P_k$ with $k < m$, then

$$\vec{x}^\delta = \begin{cases} 1 & \text{if } \delta < \vec{x} \\ 0 & \text{otherwise.} \end{cases}$$

This proves the following lemma.

Lemma 2.2 If $\vec{x} \in P_m$ and $k < m$ then $H_k^i(\vec{x}) = \sum_{\substack{\delta < \vec{x} \\ \delta \in P_k}} a_\delta^i$.

From these two lemmas we have

$$T_i(\vec{x}) = \sum_{\delta < \vec{x}} a_\delta^i + a_{\vec{x}}^i.$$

Now let $a_\delta = (a_\delta^1, a_\delta^2, \dots, a_\delta^n)$. Then for an arbitrary $T : V^n \rightarrow V^n$,

$$T(\vec{x}) = \sum_{\delta < \vec{x}} a_\delta + a_{\vec{x}}, \quad (2.1)$$

or

$$a_{\vec{x}} = \sum_{\delta < \vec{x}} a_\delta + T(\vec{x}). \quad (2.2)$$

Note that equation (2.2) gives us a recursive method for calculating the coefficients $a_{\vec{x}}$ in terms of the values of $T(\vec{x})$. Thus if we are given $T(\vec{x})$ for each $\vec{x} \in V^n$, an algorithm for constructing the corresponding polynomial map from V^n to V^n , [1], is:

Algorithm 2.1

0. Let $a_{\vec{0}} = T(\vec{0})$.
1. Using the ordering $\prec_1$, for each $\vec{x} \in P_1$ determine $a_{\vec{x}} = a_{\vec{0}} + T(\vec{x})$.
- $\vdots$
- k. Using the ordering $\prec_k$, for each $\vec{x} \in P_k$ determine $a_{\vec{x}} = \sum_{\delta < \vec{x}} a_{\delta} + T(\vec{x})$.
- $\vdots$
- n. Let $a_{\eta} = \sum_{\delta < \eta} a_{\delta} + T(\eta)$, where $\eta = (1, 1, \dots, 1)$.

A FORTRAN program and a C program based on this algorithm are given and analyzed in the Appendix. The next chapter contains several examples of polynomial maps constructed using the above algorithm.

If we consider the problem of computing the coefficients a_{δ}^i directly we have

$$T_i(\vec{x}) = \sum_{\delta \in V^n} a_{\delta}^i \vec{x}^{\delta}.$$

Evaluating T_i at each $\vec{x} \in V^n$ leads to a $2^n \times 2^n$ system, $A_n x = b$, where x is the vector of unknown a_{δ}^i 's and b is the vector of $T_i(\delta)$'s. But from equation (2.1) we have

$$T_i(\vec{x}) = \sum_{\delta \leq \vec{x}} a_{\delta}^i.$$

Therefore, when $n = 3$ the system becomes

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_{000}^i \\ a_{001}^i \\ a_{010}^i \\ a_{011}^i \\ a_{100}^i \\ a_{101}^i \\ a_{110}^i \\ a_{111}^i \end{pmatrix} = \begin{pmatrix} T_i(000) \\ T_i(001) \\ T_i(010) \\ T_i(011) \\ T_i(100) \\ T_i(101) \\ T_i(110) \\ T_i(111) \end{pmatrix}.$$

It is not difficult to see that in general (when x and b are arranged in lexicographic order as above) we have

$$A_n = \begin{pmatrix} A_{n-1} & 0 \\ A_{n-1} & A_{n-1} \end{pmatrix}, \text{ where } A_1 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}.$$

Since $A_n = A_n^{-1}$ we have $x = A_n b$ and hence

$$a_{\vec{x}}^i = \sum_{\delta \leq \vec{x}} T_i(\delta),$$

which implies

$$a_{\vec{x}} = \sum_{\delta \leq \vec{x}} T(\delta).$$

Thus we have an alternative method for computing the coefficients $a_{\vec{x}}$.

Finally, we note that the number of δ 's satisfying $\delta \leq \vec{x}$ is

$$\binom{|\vec{x}|}{0} + \binom{|\vec{x}|}{1} + \cdots + \binom{|\vec{x}|}{|\vec{x}|} = 2^{|\vec{x}|}.$$

This implies $a_{\vec{x}}^i$ is determined with $2^{|\vec{x}|}$ additions and hence computing $a_{\vec{x}}$ requires $n2^{|\vec{x}|}$ additions. Thus T is completely determined with

$$\sum_{\vec{x} \in V^n} n2^{|\vec{x}|} = \sum_{k=0}^n \sum_{\vec{x} \in P_k} n2^{|\vec{x}|} = \sum_{k=0}^n n \binom{n}{k} 2^k = n3^n$$

additions.

CHAPTER III

CONSTRUCTING POLYNOMIAL MAPS ON V^n

In this chapter we consider several polynomial maps on V^n which are of particular interest. First we give an example in Section 3.1 which merely demonstrates how to apply Algorithm 2.1 to construct the polynomial map associated with a given labeled graph. Then in Section 3.2 we construct a map which produces the same output as the polynomial system corresponding to a linear recurring sequence. However, the map we construct is highly nonlinear while the original system is linear. We go on to explore the connection between the original linear system and the constructed nonlinear system. The following section deals with a class of polynomials on V^n which are analogous to the Lagrange interpolating polynomials. This leads us to a consideration of two different bases for the space of polynomials on V^n and the mapping that takes us from basis to basis. The final section features two lemmas which completely characterize the mapping produced by Algorithm 2.1 when applied to the graph associated with the lexicographic ordering on V^n . For our purposes this is an extremely important map since it is periodic with period 2^n and hence can be used to generate pseudo-random sequences of maximal length.

3.1 Applying Algorithm 2.1

Here we consider the problem of applying Algorithm 2.1 to construct the polynomial mapping associated with a small labeled graph. This example is somewhat trivial, but it does demonstrate the actual mechanics of applying the algorithm.

Suppose that we desire to construct the polynomial mapping associated with the small labeled graph of Figure 3.1. First we associate the nodes of the graph with the elements of V^3 . This is accomplished by letting

Figure 3.1: A Labeled Graph on V^3

$$\begin{aligned} a &= (0, 0, 0), & b &= (0, 0, 1), \\ c &= (0, 1, 0), & d &= (0, 1, 1), \\ e &= (1, 0, 0), & f &= (1, 0, 1), \\ g &= (1, 1, 0), & h &= (1, 1, 1). \end{aligned}$$

Then the directed graph of Figure 3.1 implies

$$\begin{aligned} T(a) &= (0, 1, 0), & T(b) &= (0, 1, 0), \\ T(c) &= (1, 0, 0), & T(d) &= (0, 0, 1), \\ T(e) &= (0, 1, 1), & T(f) &= (1, 1, 0), \\ T(g) &= (1, 1, 1), & T(h) &= (1, 0, 1). \end{aligned}$$

Now we are in position to actually apply the algorithm. We begin by letting

$$a_{000} = T(a) = (0, 1, 0).$$

Then we calculate a_δ for each $\delta \in P_1$, taking the δ 's in lexicographic order. This gives us

$$\begin{aligned} a_{001} &= a_{000} + T(b) = (0, 0, 0), \\ a_{010} &= a_{000} - T(c) = (1, 1, 0), \\ a_{100} &= a_{000} - T(e) = (0, 0, 1). \end{aligned}$$

Proceeding similarly for P_2 , we see that

$$a_{011} = a_{000} + a_{001} + a_{010} + T(d) = (1, 0, 1),$$

$$a_{101} = a_{000} + a_{001} + a_{100} + T(f) = (1, 0, 1),$$

$$a_{110} = a_{000} + a_{010} + a_{100} + T(g) = (0, 1, 0).$$

Finally, we calculate

$$a_{111} = a_{000} + a_{001} + a_{010} + a_{100} + a_{011} + a_{101} + a_{110} + T(h) = (0, 1, 0).$$

Thus we have constructed the polynomial mapping

$$\begin{aligned} T(\vec{x}) = & (0, 1, 0) + (0, 0, 1)x_1 + (1, 1, 0)x_2 + (0, 1, 0)x_1x_2 + (1, 0, 1)x_1x_3 \\ & + (1, 0, 1)x_2x_3 + (0, 1, 0)x_1x_2x_3. \end{aligned}$$

It is straightforward to verify that this polynomial satisfies

$$T(a) = (0, 1, 0), \quad T(b) = (0, 1, 0),$$

$$T(c) = (1, 0, 0), \quad T(d) = (0, 0, 1),$$

$$T(e) = (0, 1, 1), \quad T(f) = (1, 1, 0),$$

$$T(g) = (1, 1, 1), \quad T(h) = (1, 0, 1).$$

Certainly Algorithm 2.1 may be applied to a large class of directed labeled graphs. It appears that the only requirement of a graph on V^n is that it cannot have more than one edge leaving from any given node. For example, we could not apply the algorithm to the graph in Figure 3.2, since $T(a)$ is not well defined. All of the directed graphs considered in this paper satisfy the condition that at most one edge leaves from any given node.

3.2 Polynomial Systems for a Linear Recurring Sequence

In this section we use Algorithm 2.1 to construct a polynomial map on V^4 which produces the same output sequence as a polynomial system associated with a linear recurring sequence (or m -sequence or pseudo-noise (PN) sequence). The map we construct turns out to be nonlinear, while the linear recurring sequence

Figure 3.2: Graph on V^2

is represented by a linear polynomial system. This leads us to consider the connection between the constructed mapping and the underlying linear system. Since we are primarily interested in the output sequences, these two different systems are in some sense equivalent.

Linear recurring sequences are commonly used to generate pseudo-random sequences of zeros and ones. These sequences are usually defined in terms of a primitive polynomial.

Definition 3.1 A primitive polynomial (mod 2) is an irreducible polynomial of degree n which is the minimal polynomial of a primitive root in $GF(2^n)$, [4].

Now a primitive polynomial of degree n defines a recurrence relation which, when seeded with any nonzero n -tuple, produces a sequence which has several interesting properties. For our purposes the most important property is that the output sequence is periodic with period $2^n - 1$ and it has no subperiods. [3].

As an example we consider a linear recurring sequence on V^4 . A primitive polynomial of degree four is $x^4 - x - 1$. [6], which defines the recurrence relation $x_0 = x_1 + x_4$, where x_0 is the next bit in the sequence, x_1 is the bit generated most recently and x_4 is the bit generated four steps earlier. While in general we do not yet know how to convert a polynomial system into a shift register, note that this particular recurrence relation is easily converted into the linear shift register in Figure 3.3.

Figure 3.3: Linear Shift Register

This recurrence relation can be expressed as the linear system

$$\begin{aligned}\vec{x}_{k+1} &= P(\vec{x}_k) \\ y_k &= p(\vec{x}_k),\end{aligned}\tag{3.1}$$

where

$$\begin{aligned}P(\vec{x}) &= (x_1 + x_4, x_1, x_2, x_3) \\ &= (1, 1, 0, 0)x_1 + (0, 0, 1, 0)x_2 + (0, 0, 0, 1)x_3.\end{aligned}$$

and $p(\vec{x}) = x_1 + x_4$. Observe that the output sequence $y_0, y_1, y_2, \dots$ is the linear recurring sequence defined by $x^4 + x + 1$, provided the initial vector $\vec{x}_0 \neq \vec{0}$. For example, if $\vec{x}_0 = (0, 0, 0, 1)$, then the system in (3.1) produces the sequence

$$1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, \dots\tag{3.2}$$

which is periodic with period fifteen as claimed. The labeled graph corresponding to $P(\vec{x})$ is given in Figure 3.4.

We now construct a map using Algorithm 2.1 which has the same underlying graph as $P(\vec{x})$ and produces the same output sequence. Proceeding as in Section 3.1, in Figure 3.5 we have arbitrarily associated the elements of V^4 with the nodes of our given graph. In this case we also have a desired output sequence which must be taken into account. This is accomplished by simply including a label at each node which corresponds to the desired output at that particular element. In Figure 3.6 we have labeled the nodes of the graph with the elements of the linear recurring sequence in (3.2).

Now let $T : V^4 \rightarrow V^4$ be the function depicted in Figure 3.5 and let $h : V^4 \rightarrow \mathbb{F}_2$ be the function depicted in Figure 3.6. Next, we define $T'(\vec{x}) : V^4 \rightarrow V^5$

Figure 3.4: Graph of $P(\vec{x})$ on V^4 Figure 3.5: Graph of $T(\vec{x})$ on V^4 Figure 3.6: Graph of $h(\vec{x})$

by $T'(\vec{x}) = (T(\vec{x}), h(\vec{x}))$. Then it makes sense to apply Algorithm 2.1 to $T'(\vec{x})$. Doing so we find that the coefficients of $T'(\vec{x})$ are given by

$$\begin{aligned} a_{0000} &= (0, 0, 0, 0, 0), & a_{0001} &= (0, 0, 1, 0, 1), \\ a_{0010} &= (0, 1, 0, 0, 1), & a_{0100} &= (0, 1, 1, 0, 1), \\ a_{1000} &= (1, 0, 1, 0, 0), & a_{0011} &= (0, 1, 1, 1, 0), \\ a_{0101} &= (0, 1, 1, 1, 0), & a_{0110} &= (1, 0, 1, 0, 1), \\ a_{1001} &= (1, 1, 1, 1, 0), & a_{1010} &= (0, 0, 1, 0, 0), \\ a_{1100} &= (0, 0, 1, 0, 1), & a_{0111} &= (1, 1, 1, 1, 0), \\ a_{1011} &= (1, 1, 1, 1, 0), & a_{1101} &= (1, 1, 1, 1, 1), \\ a_{1110} &= (1, 1, 0, 1, 1), & a_{1111} &= (0, 0, 0, 0, 0). \end{aligned}$$

Note that it takes $5(3^4)$ additions to compute the coefficients of T' above. In general, $(n+1)3^n$ additions are required to compute the coefficients of a map from V^n to V^{n+1} using Algorithm 2.1.

Now

$$\begin{aligned} T'(\vec{x}) &= (1, 0, 1, 0, 0)x_1 + (0, 1, 1, 0, 1)x_2 + (0, 1, 0, 0, 1)x_3 + (0, 0, 1, 0, 1)x_4 \\ &\quad + (0, 0, 1, 0, 1)x_1x_2 + (0, 0, 1, 0, 0)x_1x_3 + (1, 1, 1, 1, 0)x_1x_4 \\ &\quad + (1, 0, 1, 0, 1)x_2x_3 + (0, 1, 1, 1, 0)x_2x_4 + (0, 1, 1, 1, 0)x_3x_4 \\ &\quad + (1, 1, 0, 1, 1)x_1x_2x_3 + (1, 1, 1, 1, 1)x_1x_2x_4 + (1, 1, 1, 1, 0)x_2x_3x_4. \end{aligned}$$

and hence

$$\begin{aligned} T(\vec{x}) &= (1, 0, 1, 0)x_1 + (0, 1, 1, 0)x_2 + (0, 1, 0, 0)x_3 + (0, 0, 1, 0)x_4 \\ &\quad + (0, 0, 1, 0)x_1x_2 + (0, 0, 1, 0)x_1x_3 + (1, 1, 1, 1)x_1x_4 \\ &\quad + (1, 0, 1, 0)x_2x_3 + (0, 1, 1, 1)x_2x_4 + (0, 1, 1, 1)x_3x_4 \\ &\quad + (1, 1, 0, 1)x_1x_2x_3 + (1, 1, 1, 1)x_1x_2x_4 + (1, 1, 1, 1)x_2x_3x_4. \end{aligned}$$

and

$$h(\vec{x}) = x_2 + x_3 + x_4 + x_1x_2 + x_2x_3 + x_1x_2x_3 + x_1x_2x_4.$$

Thus the nonlinear system with output given by

$$\begin{aligned}\vec{x}_{k+1} &= T(\vec{x}_k) \\ y_k &= h(\vec{x}_k),\end{aligned}\tag{3.3}$$

produces the same output sequence (namely, the linear recurring sequence in (3.2)) as the system in (3.1), provided, of course, that we begin with the same initial vector $\vec{x}_0$.

Since the systems in (3.1) and (3.3) give the same output sequences, we consider the relationship between these two systems. Let us denote by the ordered pair (F, g) a dynamical system with output of the form

$$\begin{aligned}\vec{x}_{k+1} &= F(\vec{x}_k) \\ y_k &= g(\vec{x}_k).\end{aligned}$$

Definition 3.2 Two systems, (F_1, g_1) and (F_2, g_2) , are said to be equivalent if there exists a one-to-one polynomial mapping H such that $H(F_1(\vec{x})) = F_2(H(\vec{x}))$ and $g_2(H(\vec{x})) = g_1(\vec{x})$, [2].

The motivation behind this definition is that if such an H exists then the two systems (F_1, g_1) and (F_2, g_2) produce the same set of output sequences and hence they are in effect the same system.

Observe that if H satisfies

$$\begin{aligned}H(0, 0, 0, 0) &= (0, 0, 0, 0), & H(0, 0, 0, 1) &= (0, 0, 0, 1), \\ H(0, 0, 1, 0) &= (1, 0, 0, 0), & H(0, 0, 1, 1) &= (0, 0, 1, 0), \\ H(0, 1, 0, 0) &= (1, 1, 0, 0), & H(0, 1, 0, 1) &= (0, 1, 0, 0), \\ H(0, 1, 1, 0) &= (1, 1, 1, 0), & H(0, 1, 1, 1) &= (1, 0, 0, 1), \\ H(1, 0, 0, 0) &= (1, 1, 1, 1), & H(1, 0, 0, 1) &= (0, 0, 1, 1), \\ H(1, 0, 1, 0) &= (0, 1, 1, 1), & H(1, 0, 1, 1) &= (0, 1, 1, 0), \\ H(1, 1, 0, 0) &= (1, 0, 1, 1), & H(1, 1, 0, 1) &= (1, 1, 0, 1), \\ H(1, 1, 1, 0) &= (0, 1, 0, 1), & H(1, 1, 1, 1) &= (1, 0, 1, 0).\end{aligned}$$

then $H(T(\vec{x})) = P(H(\vec{x}))$ and $h(\vec{x}) = p(H(\vec{x}))$. Also, H is clearly one-to-one and onto. Therefore, (P, p) and (T, h) are, according to our definition, equivalent

systems on V^4 . This is not surprising since the system (T, h) was explicitly constructed so as to produce the same output sequence as the linear system (P, p) .

The coefficients b_δ of H are easily computed using Algorithm 2.1. We have

$$\begin{aligned} b_{0000} &= (0, 0, 0, 0), & b_{0001} &= (0, 0, 0, 1), \\ b_{0010} &= (1, 0, 0, 0), & b_{0100} &= (1, 1, 0, 0), \\ b_{1000} &= (1, 1, 1, 1), & b_{0011} &= (1, 0, 1, 1), \\ b_{0101} &= (1, 0, 0, 1), & b_{0110} &= (1, 0, 1, 0), \\ b_{1001} &= (1, 1, 0, 1), & b_{1010} &= (0, 0, 0, 0), \\ b_{1100} &= (1, 0, 0, 0), & b_{0111} &= (0, 1, 0, 0), \\ b_{1011} &= (0, 1, 1, 0), & b_{1101} &= (0, 0, 1, 1), \\ b_{1110} &= (1, 1, 0, 0), & b_{1111} &= (0, 0, 0, 0), \end{aligned}$$

and hence

$$\begin{aligned} H(\vec{x}) &= (1, 1, 1, 1)x_1 + (1, 1, 0, 0)x_2 - (1, 0, 0, 0)x_3 + (0, 0, 0, 1)x_4 \\ &\quad + (1, 0, 0, 0)x_1x_2 + (1, 1, 0, 1)x_1x_4 + (1, 0, 1, 0)x_2x_3 - (1, 0, 0, 1)x_2x_4 \\ &\quad + (1, 0, 1, 1)x_3x_4 + (1, 1, 0, 0)x_1x_2x_3 + (0, 0, 1, 1)x_1x_2x_4 \\ &\quad + (0, 1, 1, 0)x_1x_3x_4 + (0, 1, 0, 0)x_2x_3x_4. \end{aligned}$$

An interesting consequence of the work presented in this section is that unless we know the underlying graph and the precise ordering of the elements corresponding to the nodes, Algorithm 2.1 may produce a system which is very different from what we had in mind. The obvious drawback here is that our algorithm may not produce the “simplest” possible polynomial system. This is precisely what happened when we constructed $T(\vec{x})$ above. Certainly Algorithm 2.1 has advantages in that it is extremely efficient and reasonably easy to implement, but from this example we see it also has a disadvantage in that there is no guarantee that it will return a linear system even when there exists an equivalent linear polynomial system. Work will continue in trying to develop a reasonably efficient algorithm which produces the simplest possible polynomial system corresponding to a specified graph with output.

3.3 A Class of Interpolating Polynomials

In this section we consider the interpolating polynomials $L_{\vec{k}} : V^n \rightarrow \mathbb{F}_2$ where

$$L_{\vec{k}}(\vec{x}) = \begin{cases} 1 & \text{if } \vec{x} = \vec{k} \\ 0 & \text{otherwise.} \end{cases} \quad (3.4)$$

After proving a convenient formula for the $L_{\vec{k}}$, we give an example which shows the clear analogy between these polynomials and the Lagrange interpolating polynomials. Taking this analogy a step further, we then see that the set $\{L_{\vec{k}} : \vec{k} \in V^n\}$ is a basis for the space of polynomials on V^n . Since $\{\vec{x}^\delta : \delta \in V^n\}$ also forms a basis, we proceed to consider the map from one basis to the other.

Let $a_{\delta}^{\vec{k}}$, for $\delta \in V^n$, be the coefficients of $L_{\vec{k}}$. Then it is straightforward to apply Algorithm 2.1 to compute the coefficients $a_{\delta}^{\vec{k}}$, when, for example, $n = 3$ and $\vec{k} = (0, 1, 1)$. We have

$$\begin{aligned} a_{000}^{\vec{k}} &= 0, & a_{001}^{\vec{k}} &= 0, \\ a_{010}^{\vec{k}} &= 0, & a_{011}^{\vec{k}} &= 1, \\ a_{100}^{\vec{k}} &= 0, & a_{101}^{\vec{k}} &= 0, \\ a_{110}^{\vec{k}} &= 0, & a_{111}^{\vec{k}} &= 1, \end{aligned}$$

and hence $L_{011}(\vec{x}) = x_2x_3 + x_1x_2x_3$. The following lemma gives us a useful formula for $L_{\vec{k}}(\vec{x})$ in general.

Lemma 3.1 Let $L_{\vec{k}}(\vec{x})$ be defined as in equation (3.4). Then

$$L_{\vec{k}}(\vec{x}) = \sum_{\delta \geq \vec{k}} \vec{x}^\delta.$$

Proof: If $\vec{x} = \vec{k}$ then

$$\sum_{\delta \geq \vec{k}} \vec{x}^\delta = \vec{x}^{\vec{k}} + \sum_{\delta > \vec{k}} \vec{x}^\delta = 1$$

since $|\delta| > |\vec{x}|$ implies $\vec{x}^\delta = 0$.

If $\vec{x} \neq \vec{k}$ then we consider the following three cases:

Case I: $|\vec{x}| < |\vec{k}|$. Then $\delta \geq \vec{k}$ implies $|\delta| > |\vec{x}|$ and hence $\sum_{\delta \geq \vec{k}} \vec{x}^\delta = 0$.

Case II: $|\vec{x}| = |\vec{k}|$. Then since $\vec{x} \neq \vec{k}$ there exists an index $j \in \{1, 2, \dots, n\}$ such that $x_j = 0$ and $k_j = 1$. Hence $\vec{x}^{\vec{k}} = 0$ and

$$\sum_{\delta \geq \vec{k}} \vec{x}^\delta = \vec{x}^{\vec{k}} + \sum_{\delta > \vec{k}} \vec{x}^\delta = 0$$

Case III: $|\vec{x}| > |\vec{k}|$. If there exists an index $j \in \{1, 2, \dots, n\}$ such that $x_j = 0$ and $k_j = 1$ then $\sum_{\delta \geq \vec{k}} \vec{x}^\delta = 0$ since $\delta_j = 1$ for all $\delta \geq \vec{k}$. If no such index j exists then $\vec{k} < \vec{x}$ and we simply count the number of δ satisfying $\vec{k} \leq \delta \leq \vec{x}$. There are

$$\binom{|\vec{x}| - |\vec{k}|}{0} + \binom{|\vec{x}| - |\vec{k}|}{1} + \dots + \binom{|\vec{x}| - |\vec{k}|}{|\vec{x}| - |\vec{k}|} = 2^{|\vec{x}| - |\vec{k}|}$$

such δ and furthermore $\vec{x}^\delta = 1$ for each of these since $\delta \leq \vec{x}$. Since $2^{|\vec{x}| - |\vec{k}|}$ is even, we have $\sum_{\delta \geq \vec{k}} \vec{x}^\delta = 0$. This completes the proof of the lemma.

Let $F : V^3 \rightarrow V^3$ be the function represented by the graph in Figure 3.7.

Figure 3.7: Graph on V^3

In a clear analogy with the Lagrange interpolating polynomials, the function F can be written as

$$\begin{aligned} F(\vec{x}) = & (0, 1, 0)L_{000}(\vec{x}) + (0, 1, 1)L_{001}(\vec{x}) + (0, 1, 1)L_{010}(\vec{x}) \\ & + (1, 0, 1)L_{100}(\vec{x}) + (1, 0, 0)L_{011}(\vec{x}) + (1, 1, 0)L_{101}(\vec{x}) \\ & + (1, 1, 1)L_{110}(\vec{x}) + (0, 1, 0)L_{111}(\vec{x}). \end{aligned} \quad (3.5)$$

If instead we apply Algorithm 2.1 to the graph in Figure 3.7, we find

$$a_{000} = (0, 1, 0), \quad a_{001} = (0, 0, 1).$$

$$\begin{aligned}
a_{010} &= (0, 0, 1), & a_{100} &= (1, 1, 1), \\
a_{011} &= (1, 1, 0), & a_{101} &= (0, 1, 0), \\
a_{110} &= (0, 1, 1), & a_{111} &= (0, 0, 0),
\end{aligned}$$

and hence

$$\begin{aligned}
T(\vec{x}) &= (0, 1, 0) + (1, 1, 1)x_1 + (0, 0, 1)x_2 + (0, 0, 1)x_3 + (0, 1, 1)x_1x_2 \\
&\quad + (0, 1, 0)x_1x_3 + (1, 1, 0)x_2x_3.
\end{aligned}$$

Substituting $\sum_{\delta \geq \vec{k}} \vec{x}^\delta$ for $L_{\vec{k}}(\vec{x})$ in equation (3.5), we see that $F(\vec{x}) = T(\vec{x})$. Therefore, we have written the polynomial represented by the graph in Figure 3.7 with respect to the two bases $\{L_{\vec{k}} : \vec{k} \in V^n\}$ and $\{\vec{x}^\delta : \delta \in V^n\}$. As noted above, the polynomials $L_{\vec{k}}$ are clearly analogous to the Lagrange interpolating polynomials. Also observe that the procedure used in Algorithm 2.1 is somewhat similar to the construction of the Newton interpolating polynomial. Therefore one reasonable interpretation of Algorithm 2.1 is that it constructs a type of interpolating polynomial (with respect to the basis $\{\vec{x}^\delta : \delta \in V^n\}$) on V^n .

In matrix form, Lemma 3.1 implies that on V^3 we have

$$\begin{pmatrix} L_{000} \\ L_{001} \\ L_{010} \\ L_{011} \\ L_{100} \\ L_{101} \\ L_{110} \\ L_{111} \end{pmatrix} (\vec{x}) = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ x_3 \\ x_2 \\ x_2x_3 \\ x_1 \\ x_1x_3 \\ x_1x_2 \\ x_1x_2x_3 \end{pmatrix}.$$

Now define

$$\mathcal{L}(\vec{x}) = \begin{pmatrix} L_{\delta^1} \\ L_{\delta^2} \\ \vdots \\ L_{\delta^{2^n}} \end{pmatrix} (\vec{x}), \quad \text{and} \quad X = \begin{pmatrix} \vec{x}^{\delta^1} \\ \vec{x}^{\delta^2} \\ \vdots \\ \vec{x}^{\delta^{2^n}} \end{pmatrix},$$

where $\delta^1, \delta^2, \dots, \delta^{2^n}$ are the elements of V^n arranged in lexicographic order. Then

it is not difficult to see that in general we have

$$A_n = \begin{pmatrix} A_{n-1} & A_{n-1} \\ 0 & A_{n-1} \end{pmatrix}, \quad \text{and} \quad A_1 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

Since $A_n = A_n^{-1}$ we also have $X = A_n \mathcal{L}(\vec{x})$ and hence $\vec{x}^\delta = \sum_{\vec{k} \geq \delta} L_{\vec{k}}(\vec{x})$. We have thus given a characterization of the mapping from the basis $\{L_{\vec{k}} : \vec{k} \in V^n\}$ to the basis $\{\vec{x}^\delta : \delta \in V^n\}$ and vice versa.

3.4 The Lexicographic Map

Before defining the lexicographic map we need the following definition.

Definition 3.3 For $\vec{x} \in V^n$ let

$$\|\vec{x}\| = \sum_{i=1}^n x_i 2^{n-i}.$$

Thus $\|\vec{x}\|$ gives us the decimal number represented by $\vec{x}$ if we consider $\vec{x}$ as a binary number.

Consider $T : V^n \rightarrow V^n$, where T has the property that $\|T(\vec{x})\| = \|\vec{x}\| + 1 \pmod{2^n}$. For lack of a better term we will refer to this function as the lexicographic map. Now $T(\vec{x}_1) = T(\vec{x}_2)$ implies $\|T(\vec{x}_1)\| = \|T(\vec{x}_2)\|$, which in turn implies $\|\vec{x}_1\| = \|\vec{x}_2\|$ and hence $\vec{x}_1 = \vec{x}_2$. Therefore $T(\vec{x})$ is one-to-one. This map also has the important property that the sequence $T^k(\vec{x}_0)$ is periodic with period 2^n for any $\vec{x}_0 \in V^n$. We say that any system with a period of length 2^n (and no subperiods) is fully periodic.

Since the lexicographic map is fully periodic and since the elements of V^n must be ordered in a specific manner, there is only one possible graph. For example, when $n = 3$ the graph must have the form given in Figure 3.8. The following lemma shows that the polynomial map $T : V^n \rightarrow V^n$ which produces the lexicographic ordering has a particularly convenient form.

Lemma 3.2 Let

$$\eta_k = (\underbrace{0, \dots, 0}_k, \underbrace{1, \dots, 1}_{n-k}).$$

Figure 3.8: Lexicographic Map on V^3

Then for $P(\vec{x})$ given by

$$\begin{aligned} P(\vec{x}) &= (x_1 + \vec{x}^{\eta_1}, x_2 + \vec{x}^{\eta_2}, \dots, x_n + \vec{x}^{\eta_n}) \\ &= (x_1 + x_2 x_3 \cdots x_n, x_2 + x_3 x_4 \cdots x_n, \dots, x_{n-1} + x_n, x_n + 1) \end{aligned} \quad (3.6)$$

we have $\|P(\vec{x})\| = \|\vec{x}\| + 1 \pmod{2^n}$.

Proof: If $\vec{x} = \eta_0 = (1, 1, \dots, 1)$ then we have $P(\vec{x}) = \vec{0}$. Let $\vec{x} \in V^n$ with $\vec{x} \neq \eta_0$ and let $m = \max\{i : x_i = 0\}$. Then $x_j = 1$ for $j = m+1, m+2, \dots, n$ which implies $P_j(\vec{x}) = 0$ for $j = m+1, m+2, \dots, n$. Also, $P_m(\vec{x}) = 1$ and $P_i(\vec{x}) = x_i$ for $i = 1, 2, \dots, m-1$.

Now

$$\|\vec{x}\| = \sum_{j=1}^n x_j 2^{n-j} = \sum_{j=1}^{m-1} x_j 2^{n-j} + \sum_{j=m+1}^n 2^{n-j}$$

and

$$\|P(\vec{x})\| = 2^{n-m} + \sum_{j=1}^{m-1} x_j 2^{n-j}.$$

Since

$$\sum_{j=m+1}^n 2^{n-j} + 1 = 2^{n-m-1} + 2^{n-m-2} + \dots + 2^0 + 1 = 2^{n-m}$$

we have the result.

It follows from this lemma that the graph in Figure 3.8 can be written as

$$\begin{aligned} T(\vec{x}) &= (x_1 + x_2 x_3, x_2 + x_3, x_3 + 1) \\ &= (0, 0, 1) + (1, 0, 0)x_1 + (0, 1, 0)x_2 + (0, 1, 1)x_3 + (1, 0, 0)x_2 x_3. \end{aligned}$$

Since the graph of the lexicographic map is uniquely determined we would expect that the polynomial $T(\vec{x})$ produced by Algorithm 2.1 would agree with the function $P(\vec{x})$ in equation (3.6). The following lemma shows that this is indeed the case.

Lemma 3.3 Let η_k be as in Lemma 3.2. Then applying Algorithm 2.1 to any graph satisfying $\|T(\vec{x})\| = \|\vec{x}\| + 1 \pmod{2^n}$ produces

$$T(\vec{x}) = (x_1 + \vec{x}^{\eta_1}, x_2 + \vec{x}^{\eta_2}, \dots, x_n + \vec{x}^{\eta_n}).$$

Proof: Let $P(\vec{x})$ be as in equation (3.6) and let $T(\vec{x})$ be the map produced by Algorithm 2.1 when $\|T(\vec{x})\| = \|\vec{x}\| + 1 \pmod{2^n}$. Then by the construction of T we have $T(\vec{x}) = P(\vec{x})$ for all $\vec{x} \in V^n$. Let $H(\vec{x}) = T(\vec{x}) - P(\vec{x})$. Then $H(\vec{x}) = \vec{0}$ for all $\vec{x} \in V^n$. We must show that the coefficients a_δ of H satisfy $a_\delta = \vec{0}$ for all $\delta \in V^n$.

Now $H(\vec{0}) = a_{\vec{0}} = \vec{0}$. Suppose that we have $a_\delta = \vec{0}$ for all $\delta \in P_k$, for all $k = 0, 1, \dots, i-1$. Let $\gamma \in P_i$ be given. Then $H(\gamma) = a_\gamma = \vec{0}$ and hence by induction we have shown that $a_\delta = \vec{0}$ for all $\delta \in V^n$. Therefore, Algorithm 2.1 gives us $T(\vec{x}) = P(\vec{x}) = (x_1 + \vec{x}^{\eta_1}, x_2 + \vec{x}^{\eta_2}, \dots, x_n + \vec{x}^{\eta_n})$.

The two preceding lemmas completely characterize the polynomial produced by Algorithm 2.1 when applied to any system satisfying $\|T(\vec{x})\| = \|\vec{x}\| + 1 \pmod{2^n}$. This particular map is of great importance because it is fully periodic, which makes it useful in generating pseudo-random sequences of period 2^n . In the next chapter we consider some desirable output sequences of length 2^n and we show how to construct systems which generate these using the lexicographic map and Algorithm 2.1.

CHAPTER IV

OUTPUT SEQUENCES IN $\mathbb{F}_2$

Suppose we have the dynamical system with output

$$\begin{aligned}\bar{x}_{k+1} &= T(\bar{x}_k) \\ y_k &= h(\bar{x}_k),\end{aligned}\tag{4.1}$$

where $T : V^n \rightarrow V^n$ and $h : V^n \rightarrow \mathbb{F}_2$. For an output sequence $y_1, y_2, \dots, y_{2m}$, where $m \leq 2^{n-1}$, we would ideally like for any polynomial in the preceding j elements, $y_{k-j}, y_{k-j+1}, \dots, y_{k-1}$ to have a probability of one half of producing the correct value of y_k . Below we show that this is not possible even for a much simplified case. The question then becomes: Using the above criterion, what is the “best” pseudo-random sequence we can produce? Here we consider this problem only up to one-term linear polynomials and then only for very short output sequences. We conclude the chapter by showing how to construct a system using Algorithm 2.1 and the lexicographic map of Section 3.4 which produces a “good” output sequence.

Let $y_1, y_2, \dots, y_{2m}$ be an output sequence consisting of m zeros and m ones. Observe that there are $\binom{2m}{m}$ such sequences and that for each of these, simply guessing zero for each y_k will give the correct value precisely half of the time. The same can be said of guessing one for each y_k . In other words, any constant polynomial will give the correct value of y_k precisely half of the time. Next we will require that for m of the indices $k = 1, 2, \dots, 2m$, the output sequence satisfies $y_{k-1} = y_k$ (where the subtraction is mod $2m$). Then guessing the value of y_{k-1} for y_k will give the correct value of y_k fifty percent of the time. Using the same reasoning, we would like to have $y_{k-2} = y_k$ for m of the indices $k = 1, 2, \dots, 2m$, $y_{k-3} = y_k$ for m of the indices $k = 1, 2, \dots, 2m$ and so on, up to $y_{k-m} = y_k$, if possible. The following notation is extremely useful, [7].

Given an output sequence $y_1, y_2, \dots, y_{2m}$, let $\mu_k(2m)$ be the number of subscripts $i = 1, 2, \dots, 2m$ such that $y_{i+k} = y_i$. For example, given the sequence

0, 0, 1, 1, 1, 0, 1, 0, we have $m = 4$, $\mu_1(2m) = 4$, $\mu_2(2m) = 4$, $\mu_3(2m) = 2$ and $\mu_4(2m) = 4$. Also, observe that in general $\mu_{2m-k}(2m) = \mu_k(2m)$ for $k = 1, 2, \dots, 2m - 1$.

Lemma 4.1 For an output sequence $y_1, y_2, \dots, y_{2m}$ with m zeros and m ones, and for $\mu_k(2m)$ defined as above, we have

$$\sum_{k=1}^{2m-1} \mu_k(2m) = 2m(m-1).$$

Proof: For any element y_i in the output sequence, there are $m-1$ indices k_j^i , $j = 1, 2, \dots, m-1$, such that $y_i = y_{i+k_j^i}$ (the addition is mod $2m$). Thus we have

$$\begin{aligned} y_1 &= y_{1+k_1^1} = y_{1+k_2^1} = \dots = y_{1+k_{m-1}^1} \\ y_2 &= y_{2+k_1^2} = y_{2+k_2^2} = \dots = y_{2+k_{m-1}^2} \\ &\vdots \\ y_{2m} &= y_{2m+k_1^{2m}} = y_{2m+k_2^{2m}} = \dots = y_{2m+k_{m-1}^{2m}} \end{aligned}$$

Now $\mu_1(2m)$ is equal to the number of indices k_j^i such that $k_j^i = 1$, $\mu_2(2m)$ is equal to the number of indices k_j^i such that $k_j^i = 2$ and so on. Therefore the sum of the $\mu_k(2m)$, $k = 1, 2, \dots, 2m-1$, must equal the total number of indices k_j^i , $i = 1, 2, \dots, 2m$, $j = 1, 2, \dots, m-1$, i.e.,

$$\sum_{k=1}^{2m-1} \mu_k(2m) = 2m(m-1).$$

Now our problem can be restated as follows: Are there any output sequences $y_1, y_2, \dots, y_{2m}$ with m zeros and m ones such that $\mu_k(2m) = m$ for $k = 1, 2, \dots, m$? The answer is “no,” since for any such sequence we would have

$$\sum_{k=1}^{2m-1} \mu_k(2m) = m(2m-1) \neq 2m(m-1),$$

contradicting Lemma 4.1. So instead we ask: Are there any output sequences satisfying $\mu_k(2m) = m$ for $k = 1, 2, \dots, m-1$? When $m = 2$, we have a trivial example in 1, 1, 0, 0 which satisfies $\mu_1(2m) = 2$. But for $m = 3, 4, \dots, 8$ and

$m = 16$ no such sequences exist, [7]. For larger values of m we do not know if there are any such sequences, but it seems unlikely.

There are, however, sequences satisfying $\mu_k(2m) = m$ for $k = 1, 2, \dots, m-2$. For example, when $m = 4$ the sequence

$$1, 1, 1, 0, 1, 0, 0, 0 \quad (4.2)$$

satisfies $\mu_1(2m) = \mu_2(2m) = 4$, $\mu_3(2m) = 2$ and $\mu_4(2m) = 4$. It is interesting to note that for $m = 4$ all sequences satisfying $\mu_1(2m) = \mu_2(2m) = 4$ can be obtained from the sequence (4.2) by cyclic permutations and reflections. For $m = 8$ the sequence

$$1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0 \quad (4.3)$$

satisfies $\mu_i(2m) = 8$ for $i = 1, 2, \dots, 6$ and $\mu_7(2m) = 6$ and $\mu_8(2m) = 4$. In this case all of the other favorable sequences can be obtained from the sequence in (4.3) by cyclic permutations, reflections and by taking the bitwise complement. For $m = 16$ we have verified (using several hours of computer time) that there are no sequences satisfying $\mu_k(2m) = 16$ for $k = 1, 2, \dots, j$ when $j \geq 12$. But the sequence

$$1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0$$

satisfies $\mu_k(2m) = 16$ for $k = 1, 2, \dots, 11$ and $\mu_{12} = 18$, $\mu_{13} = 12$, $\mu_{14} = 16$, $\mu_{15} = \mu_{16} = 12$.

Now consider the problem of constructing a system on V^4 of the same form as the system in (4.1) but where the output sequence $y_1, y_2, \dots, y_{2m}$ is given by the sequence in (4.3). Since our desired output sequence is fully periodic it is reasonable to choose the lexicographic map of Section 3.4 for the function $T(\vec{x})$. In terms of a labeled graph, we have $T(\vec{x})$ given by Figure 4.1 with the output sequence in (4.3) (and hence $h(\vec{x})$) given in Figure 4.2.

Since Section 3.4 contains an explicit formula for $T(\vec{x})$, we need only compute the coefficients of $h(\vec{x})$. Observe that

Figure 4.1: Lexicographic Map on V^4 Figure 4.2: Output Sequence on V^4

$$\begin{aligned}
 h(0000) &= 1, & h(0001) &= 1, \\
 h(0010) &= 0, & h(0011) &= 1, \\
 h(0100) &= 1, & h(0101) &= 1, \\
 h(0110) &= 0, & h(0111) &= 0, \\
 h(1000) &= 0, & h(1001) &= 0, \\
 h(1010) &= 1, & h(1011) &= 0, \\
 h(1100) &= 0, & h(1101) &= 0, \\
 h(1110) &= 0, & h(1111) &= 1.
 \end{aligned}$$

Applying Algorithm 2.1 to compute the coefficients of h , we have

$$\begin{aligned}
 a_{0000} &= 1, & a_{0001} &= 0, \\
 a_{0010} &= 1, & a_{0100} &= 0, \\
 a_{1000} &= 1, & a_{0011} &= 1, \\
 a_{0101} &= 0, & a_{0110} &= 0, \\
 a_{1001} &= 0, & a_{1010} &= 0, \\
 a_{1100} &= 0, & a_{0111} &= 0, \\
 a_{1011} &= 0, & a_{1101} &= 0, \\
 a_{1110} &= 1, & a_{1111} &= 0,
 \end{aligned}$$

and hence

$$h(\vec{x}) = 1 + x_1 + x_3 + x_3x_4 - x_1x_2x_3. \quad (4.4)$$

Therefore the system

$$\begin{aligned}
 \vec{x}_{k+1} &= T(\vec{x}_k) \\
 y_k &= h(\vec{x}_k)
 \end{aligned}$$

with

$$T(\vec{x}) = (x_1 + \vec{x}^{\eta_1}, x_2 + \vec{x}^{\eta_2}, \dots, x_n + \vec{x}^{\eta_n})$$

and $h(\vec{x})$ as in equation (4.4) produces the output sequence (4.3), provided the initial vector $\vec{x}_1 = (0, 0, 0, 0)$. Any other choice for $\vec{x}_1$ will generate a cyclic permutation of the sequence in (4.3).

The important thing to notice here is that once we decided what constituted a “good” output sequence, it was not difficult to construct a system which gave us the desired sequence. Since the same type of construction would work for any specified output sequence, we see that the lexicographic map gives us a powerful tool for constructing polynomial systems on V^n . However, a very difficult problem remains in determining what constitutes a “good” (for the purposes of cryptography) pseudo-random sequence.

CHAPTER V

CONCLUSION

In this paper we presented an algorithm for constructing polynomial mappings associated with certain directed labeled graphs. Then we considered several important classes of polynomials and their corresponding graphs. The most important of these polynomials, the so-called lexicographic map, was analyzed in detail in Section 3.4. Finally, in Chapter IV we considered the question of what constitutes a “good” output sequence and we showed how to construct a system which would produce a desired output sequence using the lexicographic map.

However, many interesting questions remain unanswered. For example, we would like to know: Can we classify the labeled graphs which correspond to linear polynomials? Given a one-to-one polynomial map on V^n , can we determine the invariant subsets of its labeled graph (and hence the subset with the largest period)? Is there an efficient algorithm which will always produce the “simplest” polynomial system corresponding to a given labeled graph? We must also determine what is a “good” output sequence and whether the method discussed in Chapter IV is applicable for longer output sequences or for higher levels of complexity. Finding the most efficient possible implementation of Algorithm 2.1 is also vital (see the Appendix). Perhaps most importantly we must eventually be able to convert an arbitrary polynomial system on V^n (or equivalently, an arbitrary labeled graph) into some type of shift register device.

REFERENCES

- [1] Martin, Clyde F. *A Note on Linearization and Construction of Polynomial Mappings*. Preprint, Department of Mathematics, Texas Tech University, Lubbock, Texas.
- [2] Martin, Clyde F. *Local and Global Observation of Polynomial Systems Over a Finite Field*. Preprint, Department of Mathematics, Texas Tech University, Lubbock, Texas.
- [3] Lidl, Rudolf and Harold Niederreiter. *Finite Fields*. Vol. 20 of *Encyclopedia of Mathematics and Its Applications*. Reading, MA: Addison-Wesley Publishing Company, 1983.
- [4] McEliece, Robert J. *Finite Fields for Computer Scientists and Engineers*. Norwell, MA: Kluwer Academic Publishers, 1987.
- [5] Knuth, Donald E. *Seminumerical Algorithms*. 2nd ed. Vol. 2 of *The Art of Computer Programming*. Reading, MA: Addison-Wesley Publishing Company, 1981.
- [6] Press, William H., Brian P. Flannery, Saul A. Teukolsky and William T. Vetterling, *Numerical Recipes: The Art of Scientific Computing*. New York: Cambridge University Press, 1986.
- [7] Takacs, Lajos F. Letter to Clyde F. Martin. 19 May 1988.

APPENDIX
FORTRAN AND C IMPLEMENTATIONS
OF ALGORITHM 2.1

This appendix contains a FORTRAN and a C program to compute the coefficients of a polynomial mapping on V^n by the procedure given in Algorithm 2.1. Below is a listing of the FORTRAN program followed by a brief explanation of the method used to find the $\delta < \bar{x}$. Then we show how to run the program on a simple example. Next we give a listing of the C program, which is essentially the same as the FORTRAN program, and finally we consider the performance of these programs.

```

1      PROGRAM POLY
2      IMPLICIT INTEGER (A-Z)
3
4      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
5      C                                                    N
6      C  A(i) MUST BE DIMENSIONED FROM 0 TO AT LEAST 2 -1
7      C
8      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
9
10     INTEGER A(0:66000),C(0:32,0:32),E(0:100),B(0:100)
11     INTEGER X(0:100),Z(0:100)
12
13     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
14     C                                                    N
15     C  SET N TO DIM OF VECTOR SPACE V.
16     C
17     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
18
19     N=16
20
21     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
22
23     DO 33 I=0,N-1
24         E(I)=2**I
25 33    CONTINUE

```

```

26
27 C
28 C 'TFILE.DAT' CONTAINS VALUES OF T(X)
29 C   STORED AS BASE 10 INTEGERS.
30 C   (adjustment to RECL may be required)
31
32       OPEN(1,FILE='TFILE.DAT',ACCESS='DIRECT',RECL=1,
33       +     FORM='UNFORMATTED',STATUS='OLD')
34
35 C   PARTITION ZERO.
36
37       READ(1,REC=1) A_0
38       A(0)=A_0
39
40 C
41 C   PARTITION NUMBER ONE.
42 C
43 C   X(i) IS THE VECTOR X. NORMX IS THE 'NORM' OF THE VECTOR X,
44 C   I. E. THE BASE TEN NUMBER REPRESENTED BY X.
45 C
46
47       DO 7 J=0,N-1
48           NORMX=E(J)
49           READ(1,REC=NORMX+1) T
50           A(NORMX)=IEOR(A_0,T)
51 7       CONTINUE
52
53 C   PARTITION NUMBER TWO.
54
55       X(0)=0
56       X(1)=1
57       DO 77 J=1,C(N,2)
58           NORMX=E(X(0))+E(X(1))

```

```

59          SUM=A_0
60          SUM=IEOR(SUM,A(E(X(0))))
61          SUM=IEOR(SUM,A(E(X(1))))
62
63          READ(1,REC=NORMX+1) T
64          A(NORMX)=IEOR(SUM,T)
65
66          X(0)=X(0)+1
67          IF(X(0).LT.X(1)) GOTO 77
68          X(0)=0
69          X(1)=X(1)+1
70 77      CONTINUE
71
72  C
73  C  PARTITIONS 3 THRU N-1.
74  C
75
76          DO 10 I=3,N-1
77
78  C  GET FIRST ELEMENT OF Ith PARTITION
79
80          DO 15 J=0,I-1
81              X(J)=J
82 15      CONTINUE
83
84  C  CALCULATE X AND BEGIN SUM OVER DELTA < X
85
86          DO 20 J=1,C(N,I)
87              SUM=A_0
88              NORMX=0
89              DO 11 K=0,I-1
90                  B(K)=E(X(K))
91                  SUM=IEOR(SUM,A(B(K)))

```

```

92          NORMX=NORMX+B(K)
93  11          CONTINUE
94
95  C
96  C  SUM OVER REMAINING DELTA < X
97  C
98
99          DO 50 K=2,I-1
100             DO 55 L=0,K-1
101                Z(L)=L
102  55          CONTINUE
103             DO 60 L=1,C(I,K)
104                DELTA=0
105                DO 96 J1=0,K-1
106                   DELTA=DELTA+B(Z(J1))
107  96          CONTINUE
108                SUM=IEOR(SUM,A(DELTA))
109                Z(0)=Z(0)+1
110                DO 97 J2=1,K-1
111                   IF(Z(J2-1).LT.Z(J2)) GOTO 60
112                   Z(J2-1)=J2-1
113                   Z(J2)=Z(J2)+1
114  97          CONTINUE
115  60          CONTINUE
116  50          CONTINUE
117
118
119  C  ADD T(X).
120
121          READ(1,REC=NORMX+1) T
122          A(NORMX)=IEOR(SUM,T)
123
124  C  GET NEXT ELEMENT OF Ith PARTITION

```

```

125
126             X(0)=X(0)+1
127             DO 14 K=1,I-1
128                 IF(X(K-1).LT.X(K)) GOTO 20
129                 X(K-1)=K-1
130                 X(K)=X(K)+1
131 14          CONTINUE
132 20          CONTINUE
133 10          CONTINUE
134
135  C
136  C  PARTITION N.
137  C
138
139          SUM=A_0
140          NN=2**N
141          DO 200 I=1,NN-2
142              SUM=IEOR(SUM,A(I))
143 200        CONTINUE
144          READ(1,REC=NN) T
145          A(NN-1)=IEOR(SUM,T)
146
147          CLOSE(1)
148
149  C
150  C  WRITE A          THRU A          TO 'AFILE.DAT'
151  C          (0,0...0)          (1,1...1)
152  C
153  C  IN LEXICOGRAPHICAL ORDER. NOTE THAT
154  C
155  C  THE VECTORS A  ARE WRITTEN AS BASE 10 INTEGERS.
156  C          X
157

```

```

158      OPEN(2,FILE='AFILE.DAT',ACCESS='DIRECT',RECL=1,
159      +      FORM='UNFORMATTED',STATUS='NEW')
160      DO 100 I=0,NN-1
161          WRITE(2,REC=I+1) A(I)
162 100    CONTINUE
163      CLOSE(2)
164
165  C
166  C  BINOMIAL COEFFICIENTS. THE PROGRAM USES THE
167  C  BIN. COEFFICIENTS (3 choose k) FOR k=0,1,2,3
168  C  UP TO (N choose k) FOR k=0,1,2,...,N.
169  C
170
171      DATA ((C(I,J),J=0,I),I=3,20)/1,3,3,1,1,4,6,4,1,
172      + 1,5,10,10,5,1,1,6,15,20,15,6,1,1,7,21,35,35,
173      + 21,7,1,1,8,28,56,70,56,28,8,1,1,9,36,84,126,
174      + 126,84,36,9,1,1,10,45,120,210,252,210,120,
175      + 45,10,1,1,11,55,165,330,462,462,330,165,55,11,
176      + 1,1,12,66,220,495,792,924,792,495,220,66,12,1,
177      + 1,13,78,286,715,1287,1716,1716,1287,715,286,
178      + 78,13,1,1,14,91,364,1001,2002,3003,3432,3003,
179      + 2002,1001,364,91,14,1,1,15,105,455,1365,3003,
180      + 5005,6435,6435,5005,3003,1365,455,105,15,1,1,
181      + 16,120,560,1820,4368,8008,11440,12870,11440,
182      + 8008,4368,1820,560,120,16,1,1,17,136,680,2380,
183      + 6188,12376,19448,24310,24310,19448,12376,6188,
184      + 2380,680,136,17,1,1,18,153,816,3060,8568,18564,
185      + 31824,43758,48620,43758,31824,18564,8568,3060,
186      + 816,143,18,1,1,19,171,969,3876,11628,27132,50388,
187      + 75582,92378,92378,75582,50388,27132,11628,3876,
188      + 969,171,19,1,1,20,190,1140,4845,15504,38760,77520,
189      + 125970,167960,184756,167960,125970,77520,
190      + 38760,15504,4845,1140,190,20,1/

```


191

192

END

Be sure to observe that in line 10 the array A must be dimensioned from 0 to at least $2^n - 1$ and that N (see line 19) must be set to the dimension of V^n . Also, note that addition modulo 2 is accomplished using the FORTRAN exclusive or, IEOR. In some versions of FORTRAN the intrinsic function is XOR, not IEOR.

Most of the program follows directly from Algorithm 2.1. One possible area of confusion is in the method of finding the $\delta < \bar{x}$, or in terms of the variables used in the program, finding the $\text{DELTA} < X$. The procedure used in the program (see lines 99 through 116) is as follows. We use the vector Z to find the elements of the partitions $P_2, P_3, \dots, P_I$ of V^I , where I equals the number of nonzero elements of X . Then we use the nonzero elements of Z to determine which bits of DELTA are to be "on." For example, if $N = 8$ and $X = (1, 0, 1, 1, 0, 1, 0, 1)$ (and hence $I = 5$) then when $Z = (0, 0, 0, 1, 1)$, we have $\text{DELTA} = (0, 0, 0, 0, 0, 1, 0, 1)$. When $Z = (0, 0, 1, 0, 1)$ we have $\text{DELTA} = (0, 0, 0, 1, 0, 0, 0, 1)$, when $Z = (1, 1, 0, 1, 1)$ then $\text{DELTA} = (1, 0, 1, 0, 0, 1, 0, 1)$ and so on. The trick is in the double subscript. $B(Z(J1))$ at line 106.

Now to see how to run the program, let us consider the example given in Section 3.1. Recall that we had

$$\begin{aligned} T(0, 0, 0) &= (0, 1, 0), & T(0, 0, 1) &= (0, 1, 0), \\ T(0, 1, 0) &= (1, 0, 0), & T(0, 1, 1) &= (0, 0, 1), \\ T(1, 0, 0) &= (0, 1, 1), & T(1, 0, 1) &= (1, 1, 0), \\ T(1, 1, 0) &= (1, 1, 1), & T(1, 1, 1) &= (1, 0, 1). \end{aligned}$$

Our input file must contain the base ten numbers represented by the $T(\bar{x})$ above. So we create an unformatted direct access file called TFILE.DAT which has as its elements

$$2, \quad 2, \quad 4, \quad 1, \quad 3, \quad 6, \quad 7, \quad 5.$$

Next, we set $N = 3$ in line 19 and we make sure that the array A is dimensioned from 0 to at least 7. After running the program, the output written to the

sequential file AFILE.DAT will be

2, 0, 6, 5, 1, 5, 2, 2,

which means that

$$\begin{aligned}
 a_{000} &= (0, 1, 0) \quad (\text{i.e. } \|a_{000}\| = 2), \\
 a_{001} &= (0, 0, 0), \\
 a_{010} &= (1, 1, 0), \\
 a_{011} &= (1, 0, 1), \\
 a_{100} &= (0, 0, 1), \\
 a_{101} &= (1, 0, 1), \\
 a_{110} &= (0, 1, 0), \\
 a_{111} &= (0, 1, 0).
 \end{aligned}$$

Now we give a C program which is essentially the same as the FORTRAN program listed above.

```

1
2  /* Set N to dimension of V^N and dimension A to 2^N.
3
4      The array C contains the binomial coefficients.      */
5
6  unsigned short N=16;
7  unsigned long A[65536];
8
9  unsigned long C[17][17]={0}, {0}, {0}, {1,3,3,1}, {1,4,6,4,1},
10 {1,5,10,10,5,1}, {1,6,15,20,15,6,1}, {1,7,21,35,35,21,7,1},
11 {1,8,28,56,70,56,28,8,1}, {1,9,36,84,126,126,84,36,9,1},
12 {1,10,45,120,210,252,210,120,45,10,1}, {1,11,55,165,330,
13 462,462,330,165,55,11,1}, {1,12,66,220,495,792,924,792,495,
14 220,66,12,1}, {1,13,78,286,715,1287,1716,1716,1287,715,286,
15 78,13,1}, {1,14,91,364,1001,2002,3003,3432,3003,2002,1001,

```

```

16 364,91,14,1},{1,15,105,455,1365,3003,5005,6435,6435,5005,
17 3003,1365,455,105,15,1},{1,16,120,560,1820,4368,8008,11440,
18 12870,11440,8008,4368,1820,560,120,16,1}}};
19
20 #include<stdio.h>
21 FILE *tfile, *afile;
22
23 /* ***** */
24
25 main()
26 {
27     unsigned long E[100], B[100], DELTA, SUM;
28     unsigned short X[100], Z[100];
29     unsigned long I, J, K, J1, J2, L, NORMX, A_0, NN, T;
30     for(I=0; I<N; ++I)
31         E[I]=1<<I;
32
33     /*      Input file containing values of T(x) is TFILE.      */
34
35     tfile=fopen("TFILE","r");
36
37     /*      Partition zero      */
38
39     A_0=getw(tfile);
40     A[0]=A_0;
41
42     /*      Partition one      */
43
44     for(J=0; J<N; ++J)
45     {
46         NORMX=E[J];
47         fseek(tfile,NORMX*4,0);
48         A[NORMX]=A_0+getw(tfile);

```

```

49     }
50
51     /*      Partition two          */
52
53     X[0]=0;
54     X[1]=1;
55     for(J=1; J<=C[N][2]; ++J)
56     {
57         NORMX=E[X[0]]+E[X[1]];
58         fseek(tfile,NORMX*4,0);
59         A[NORMX]=A_0^A[E[X[0]]]^A[E[X[1]]]^getw(tfile);
60         ++X[0];
61         if(X[0] == X[1])
62         {
63             X[0]=0;
64             ++X[1];
65         }
66     }
67
68     /*      Partitions 3 thru N-1      */
69
70     for(I=3; I<N; ++I)
71     {
72         for(J=0; J<I; ++J)
73             X[J]=J;
74         for(J=1; J<=C[N][I]; ++J)
75         {
76             SUM=A_0;
77             NORMX=0;
78             for(K=0; K<I; ++K)
79             {
80                 B[K]=E[X[K]];
81                 SUM=SUM^A[B[K]];

```

```

82             NORMX=NORMX+B[K];
83         }
84         for(K=2; K<I; ++K)
85         {
86             for(L=0; L<K; ++L)
87                 Z[L]=L;
88
89         /*      SUM over A[DELTA] where DELTA < X          */
90
91             for(L=1; L<=C[I][K]; ++L)
92             {
93                 DELTA=0;
94                 for(J1=0; J1<K; ++J1)
95                     DELTA=DELTA+B[Z[J1]];
96                 SUM=SUM^A[DELTA];
97                 ++Z[0];
98                 for(J2=1; J2<K; ++J2)
99                 {
100                     if(Z[J2-1] < Z[J2])
101                         break;
102                     Z[J2-1]=J2-1;
103                     ++Z[J2];
104                 }
105             }
106         }
107
108     /*      Add T(X) and get next X in partition I.      */
109
110         fseek(tfile,NORMX*4,0);
111         A[NORMX]=SUM^getw(tfile);
112         ++X[0];
113         for(K=1; K<I; ++K)
114         {

```

```

115             if(X[K-1] < X[K])
116                 break;
117             X[K-1]=K-1;
118             ++X[K];
119         }
120     }
121 }
122
123 /*      Partition N                      */
124
125     NN=1<<N;
126     SUM=A_0;
127     for(I=1; I<(NN-1); ++I)
128         SUM=SUM^A[I];
129     fseek(tfile,(NN-1)*4,0);
130     A[NN-1]=SUM^getw(tfile);
131     fclose(tfile);
132
133 /*      Write coefficients A(x) to AFILE.      */
134
135     afile=fopen("AFILE","w");
136     for(I=0; I<=(NN-1); ++I)
137         putw(A[I],afile);
138     fclose(afile);
139 }
140

```

As a comparison of the efficiency of these two programs we computed the coefficients of the polynomial maps from V^n to V^n , for $n = 12$, $n = 15$ and $n = 16$, where $\|T(\vec{x})\| = 6$ for all $\vec{x} \in V^n$ (i.e., every element of the input file TFILE.DAT is 6). The programs will return $\|a_{\vec{0}}\| = 6$ and $a_{\vec{x}} = \vec{0}$ for all $\vec{x} \in V^n$ such that $\vec{x} \neq \vec{0}$. In other words, the polynomial $T(\vec{x})$ given by the algorithm is the constant polynomial, as expected. Using this example, we ran both programs

on an IBM PC AT (using a Microsoft FORTRAN Optimizing Compiler with the emulator option and a Microsoft C Compiler), on Texas Tech University's VAX 8650 and on the Texas Tech Mathematics Department's Sun 3/260 computer. The results, with approximate times in minutes and seconds, are summarized in the following tables. Recall that $n3^n$ additions (binary exclusive or's) are required to compute the 2^n coefficients a_δ .

$N = 12$		Coefficients: 4096		Additions: 6.377.292	
Programming language	VAX 8650		Sun 3/260		IBM PC AT
	CPU time	run time	CPU time	run time	run time
FORTRAN	0 : 05	0 : 09	0 : 08	0 : 27	1 : 48
C	0 : 05	0 : 11	0 : 12	0 : 13	1 : 53

$N = 15$		Coefficients: 32768		Additions: 215.233.605	
Programming language	VAX 8650		Sun 3/260		IBM PC AT
	CPU time	run time	CPU time	run time	run time
FORTRAN	2 : 02	2 : 45	3 : 37	6 : 16	52 : 41
C	2 : 08	3 : 05	6 : 13	6 : 23	—

$N = 16$		Coefficients: 65536		Additions: 688.747.536	
Programming language	VAX 8650		Sun 3/260		IBM PC AT
	CPU time	run time	CPU time	run time	run time
FORTRAN	6 : 13	7 : 46	10 : 57	16 : 50	2 : 39 : 19
C	6 : 43	9 : 29	19 : 19	19 : 41	—

It appeared likely that a great deal of time was being spent in finding the $\delta < \bar{x}$. To confirm this we commented out lines 100 through 102, 105 through 107 and 109 through 113 of the FORTRAN program listed earlier. These lines of code are used solely to calculate the $\delta < \bar{x}$. Then running the program should give us an indication of the amount of time involved in finding the $\delta < \bar{x}$. With $N = 16$ we found that on the VAX 8650 it took only 49 seconds of CPU time and 2:19

to run as compared with 6:13 and 7:46 respectively, (see the table above) for the working program. This experiment shows that any substantial improvement in the performance of this program will most likely result from finding a more efficient method of finding the vectors δ such that $\delta < \bar{x}$.

PERMISSION TO COPY

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Texas Tech University, I agree that the Library and my major department shall make it freely available for research purposes. Permission to copy this thesis for scholarly purposes may be granted by the Director of the Library or my major professor. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my further written permission and that any user may be liable for copyright infringement.

Disagree (Permission not granted)

Agree (Permission granted)

Student's signature

M. L. E. Jr.

Student's signature

Date

11/3/88

Date