Summer 2004

Algebraic Coding Theory

Ben Johnson

Algebraic Coding Theory

Ben Johnson, Class of 2006;   Nuh Aydin, Assistant Professor of Mathematics
Kenyon College Summer Science Scholars Project, Summer 2004

Abstract.

Almost since the first computers were built, protecting digital signals from errors due to naturally occurring noise has been a problem of immense practical importance. Adding redundancy to the message can increase the signal's resistance to noise. In this project, we first examined some existing **codes** -- procedures for adding redundancy. Then, with the aid of advanced theoretical algebraic tools, we devised some improvements to an existing computer program to search for new, "good" codes.

Background: Finite fields and linear codes

Certain assumptions and ideas are typical in coding theory. A **message** is composed of one or more **blocks** of k **symbols** from some **alphabet** of q possible symbols. All of the codes I worked with are **block codes**, which encode each block independently. The alphabet is usually a **field** – a set with operations called **addition** and **multiplication** where the "usual" rules of linear equations – the **field axioms** – are true.

A **systematic code** is a very general type of code. In a systematic code, the code constructs an **output block** to be sent over the noisy channel by appending **check symbols** to the original message, called **data symbols**. A **linear code** is a special type of systematic code. In a linear code, the check symbols are computed as a **weighted sum** of the data symbols. A weighted sum in a finite field looks like

$w_1*u_1+w_2*u_2+w_3*u_3+\ldots+w_k*u_k$ for weights w_j and inputs u_j (field elements)

Field axioms.

	Additive	Multiplicative
Commutative	$x+y = y+x$	$x*y = y*x$
Associative	$(x+y)+z = x+(y+z)$	$(x*y)*z = x*(y*z)$
Identity	$x+0 = x$	$x*1 = x$
Inverse	$x+(-x) = 0$	$x*(x^{-1}) = 1$
Distributive	$x*(y+z) = (x*y)+(x*z)$	

Finite fields commonly used in coding theory

Binary

+	0	1
0	0	1
1	1	0

*	0	1
0	0	0
1	0	1

Ternary

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

*	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

Quaternary

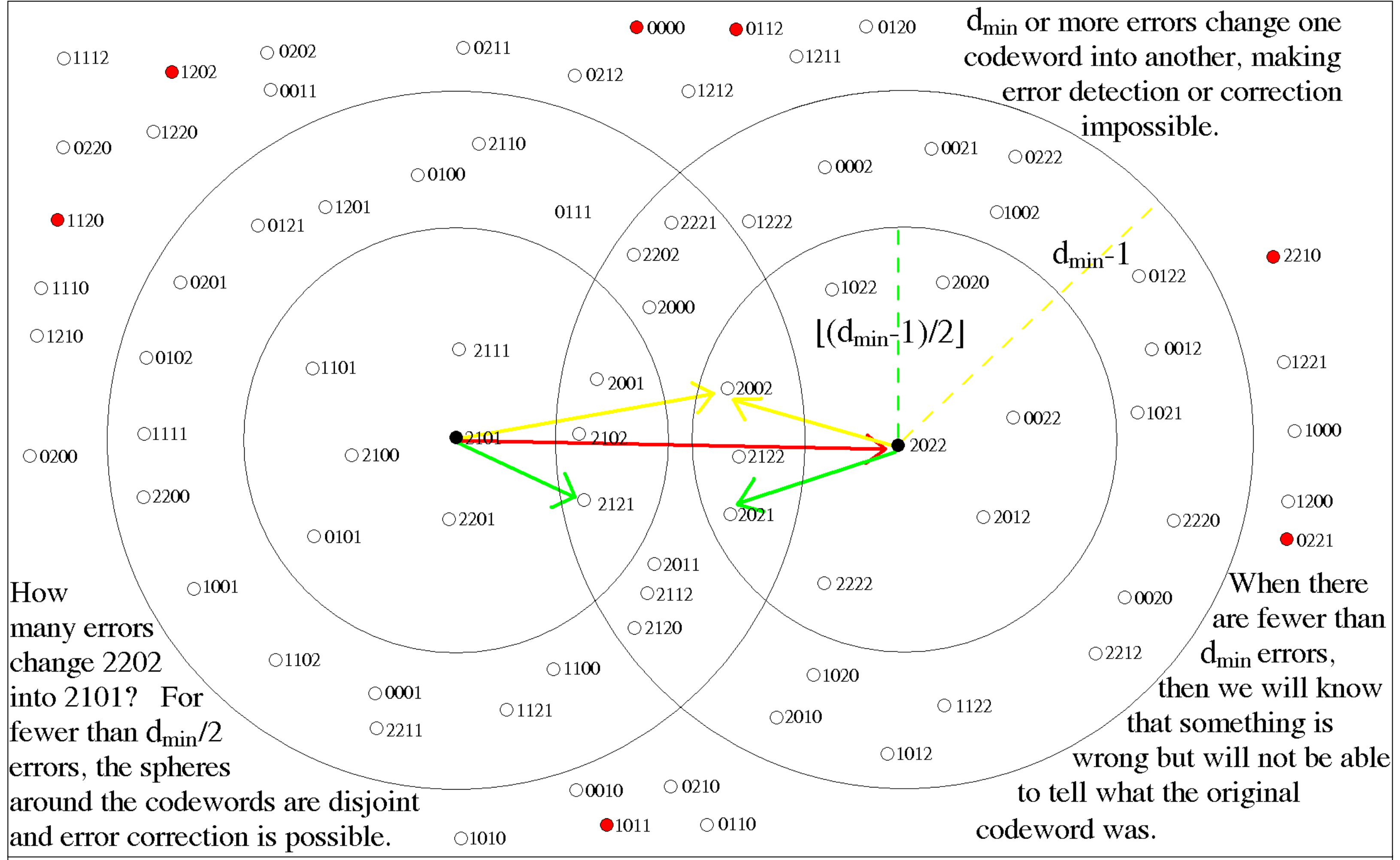
+	0	1	a	b
0	0	1	a	b
1	1	0	b	a
a	a	b	0	1
b	b	a	1	0

*	0	1	a	b
0	0	0	0	0
1	0	1	a	b
a	0	a	b	1
b	0	b	1	a

The Hamming metric

Now that we have discussed some things about the messages and codes, let us now focus on the errors. An **error** is defined to be the incorrect transmission of a single symbol. If we compare a corrupted message v to the originally intended message u, the number of errors is simply the number of positions where u and v differ. This is called the **Hamming distance** $d(u, v)$. The word "distance" will lead the mathematically inclined reader to (correctly) suspect a **metric space**.

How good is a code at detecting errors? Call the set of possible output blocks the set of **codewords**. If the receiver rejects any non-codeword transmission, then the only case in which the receiver can make a mistake is when the errors change one codeword into another. The number of errors necessary to do this is d_{min}, the minimum Hamming distance between two codewords. For error correction, we will be able to determine the original codeword if fewer than d_{min} / 2 errors occur.

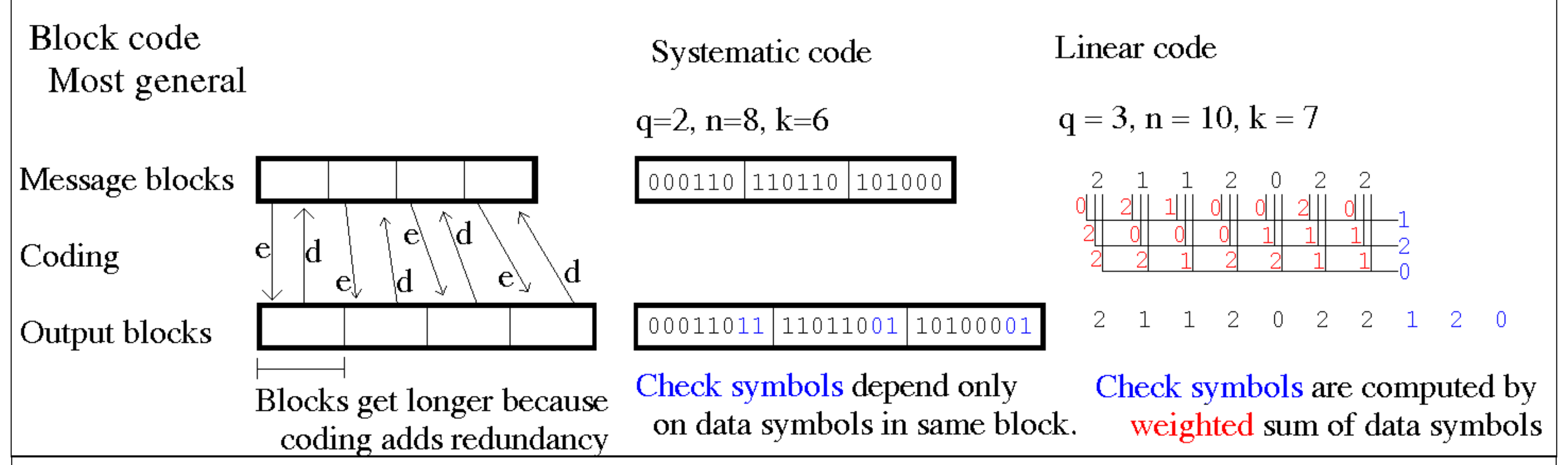


d_{min} or more errors change one codeword into another, making error detection or correction impossible.

How many errors change 2202 into 2101? For fewer than $d_{min}/2$ errors, the spheres around the codewords are disjoint and error correction is possible.

When there are fewer than d_{min} errors, then we will know that something is wrong but will not be able to tell what the original codeword was.

Types of codes



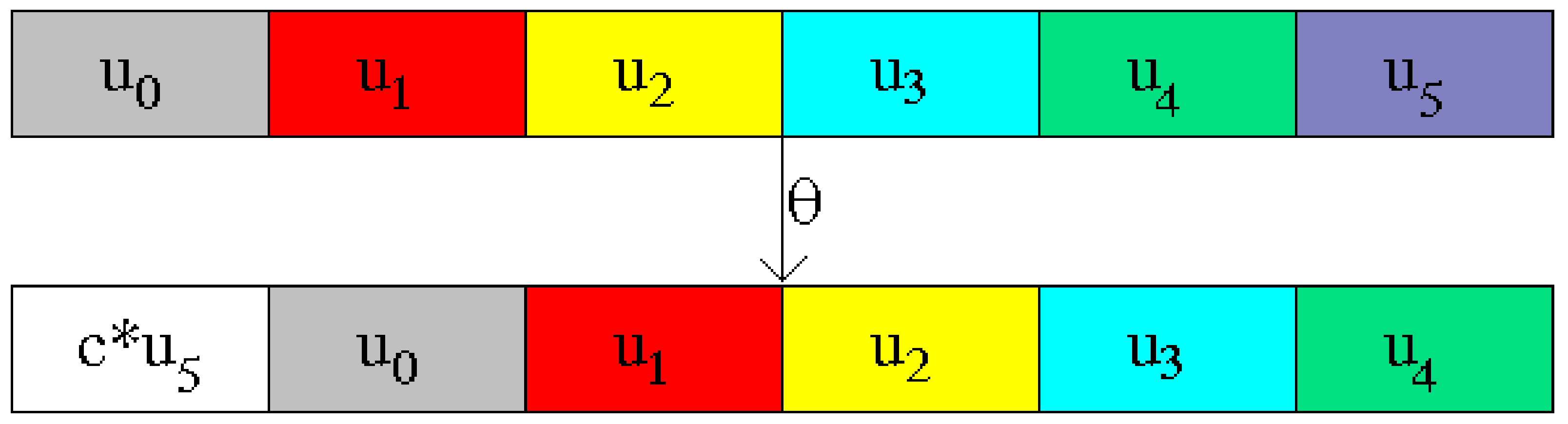
Blocks get longer because coding adds redundancy

Check symbols depend only on data symbols in same block.

Check symbols are computed by weighted sum of data symbols

Constacyclic codes

The focus of my project is how to efficiently compute the minimum Hamming distance for a specific type of linear code. First of all, $d(u, v) = d(u-v, \mathbf{0})$. For linear codes, u-v is also a codeword. So instead of finding the minimum distance between pairs of codewords, it suffices to find the minimum number of nonzero components of a codeword. (The number of nonzero components of u is called the Hamming weight, $w(u)$.)



This diagram shows an operation called **constacyclic shift**, denoted with Greek letter θ (theta). It just shifts the vector one position to the right; the symbol that "falls off" the right edge is multiplied by a nonzero **constacyclic constant**, c. A **constacyclic code** is a linear code where every constacyclic shift of a codeword is also a codeword.

An **orbit** of a codeword u under θ is just all of the codewords you can get from successive constacyclic shifts of u. A well-known result from group theory states that every codeword falls into exactly one orbit; the orbits **partition** the code.

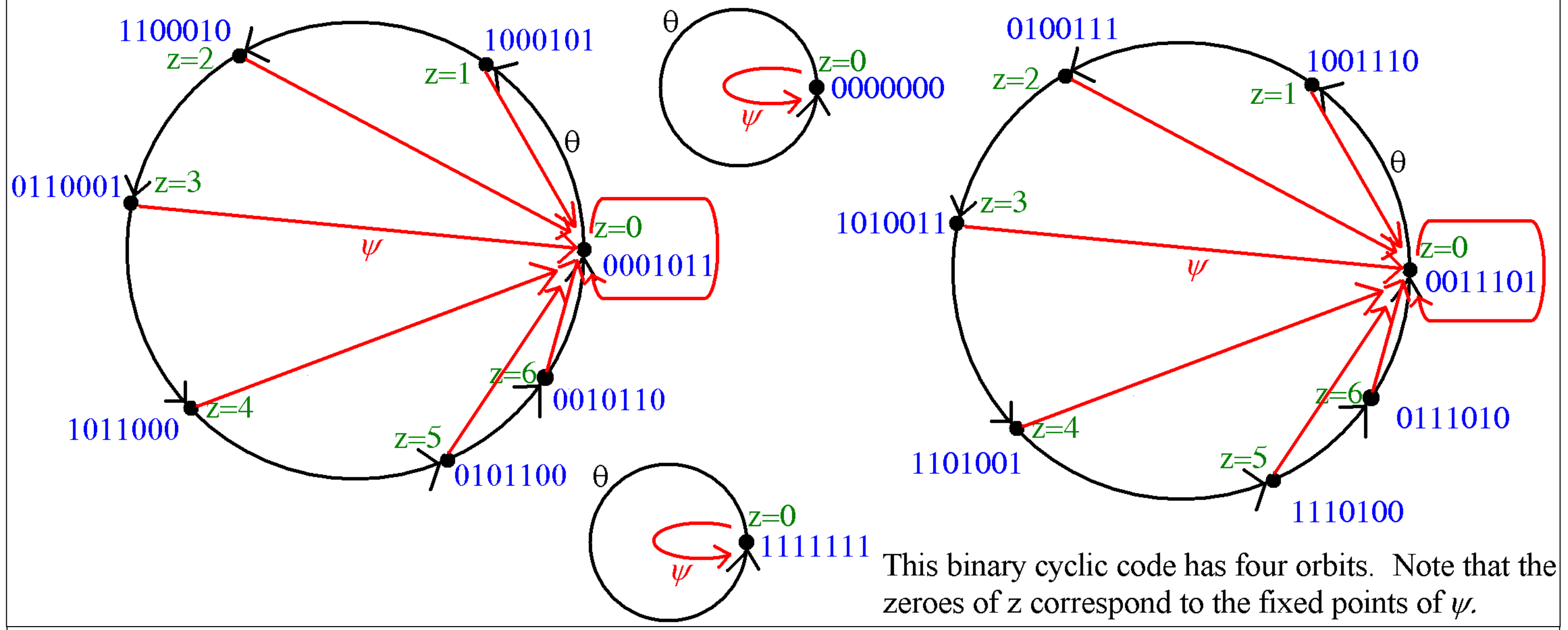
The constacyclic shift preserves the Hamming weight; therefore the codewords within any given orbit all have the same weight. Thus, we can find the minimum Hamming weight more efficiently if we only consider one codeword from each orbit.

Finding representatives: Zeroes of t

We can think of the orbits as points on a circle and θ as a rotation operation. Let us call our representative ψ. Going in the direction of θ, give each point u a label $t(u)$. Let $t(\psi) = 0$ and $t(\theta u) = t(u)+1$. Clearly, then, $u = \theta^{t(u)}\ \psi$. Now solve for $\psi : \psi = u\ \theta^{-t(u)}$. Intuitively, ψ takes any element from the orbit and maps it to a unique representative of that orbit.

Can we construct some function t such that $t(\theta u) = t(u)+1$? I did it, but the function is very difficult to describe. It uses the advanced algebraic tools like the theory of **polynomial rings**, multiple applications of a result known as the **Chinese Remainder Theorem**, and it may require extremely computationally intensive **discrete logarithms** to implement.

Fortunately, though, we don't have to actually compute t for any specific value of u. Rather, we are interested in the reverse: we are interested in the **fixed points** of ψ, which correspond to the **zeroes** of t; that is, there is exactly one point u in each orbit such that $t(u) = 0$. So we need to solve $t(u) = 0$ for u. This turns out to be efficiently doable.



This binary cyclic code has four orbits. Note that the zeroes of z correspond to the fixed points of ψ.

Future work

I am presently working on a computer implementation of my procedure for a more general class of codes, the **quasi-twisted codes**, which are closed under θ^m for some m. I plan to complete and run this program at the Ohio Supercomputing Center at Ohio State University sometime this semester. I also plan to publish a paper detailing my findings and any new, good codes found by my computer search.

Acknowledgements and bibliography.

I would like to thank the Kenyon College Summer Science Scholars program for providing funding for this research; the Ohio Supercomputing Center at Ohio State University, for providing a grant of supercomputer time; Robert Lewis of Fordham University for writing and maintaining the free computer algebra software Fermat; and my faculty advisor, Noah Aydin, for giving this work encouragement, support, and direction.

Aydin, Nuh. *The Structure of 1-Generator Quasi-Twisted Codes and New Linear Codes*. Designs, Codes and Cryptography, 24, 313-326, 2001.

G.-M. Greuel, G. Pfister, and H. Schonemann. *Singular 2.0. A Computer Algebra System for Polynomial Computations*. Centre for Computer Algebra, University of Kaiserslautern (2001). http://www.singular.uni-kl.de

Lewis, Robert. *Fermat*. Computer algebra software. http://www.bway.net/~lewis

Pless, Vera. *Introduction to the Theory of Error-Correcting Codes*. 3rd Ed.