Cxn You Xnderxtand xhe Titxe of txis Txlk?
(the basics of error-correcting codes)

Joe Fields

http://www.southernct.edu/~fields/coding.pdf

The experiment of deleting every third or fourth letter from an example of English prose – and finding that it remains quite readable – shows that English contains a fair amount of redundancy. Even if (say) a telegraphic system for transmitting information is quite unreliable, it will still, usually, be possible to determine what the original message was. This is error-correction in a nutshell; make sure that a message contains sufficient redundancy that *even if it gets mangled in transmission* we can retrieve the original meaning.

Error-correcting codes are in ubiquitous use these days: cellular telephony, CDs and DVDs, computer memory, and deep space communication (to name just a few key technologies) all make use of error-correcting codes. In the theoretical world we also find a multitude of uses for error-correcting codes – constructions for lattices in n-dimensional space, exceptional Lie algebras, sporadic simple groups (to name just another few).

Jupiter

a little humor

A Plan for the Improvement of English Spelling

For example, in Year 1 that useless letter "c" would be dropped to be replased either by "k" or "s", and likewise "x" would no longer be part of the alphabet.
The only kase in which "c" would be retained would be the "ch" formation, which will be dealt with later.
Year 2 might reform "w" spelling, so that "which" and "one" would take the same konsonant, wile Year 3 might well abolish "y" replasing it with "i" and iear 4 might fiks the "g/j" anomali wonse and for all.

humor continued

Jenerally, then, the improvement would kontinue iear bai iear with iear 5 doing awai with useless double konsonants, and iears 6-12 or so modifaiing vowlz and the rimeining voist and unvoist konsonants.

Bai iear 15 or sou, it wud fainali bi posibl tu meik ius ov thi ridandant letez "c", "y" and "x" – bai now jast a memori in the maindz ov ould doderez – tu riplais "ch", "sh", and "th" rispektivli.

Fainali, xen, aafte sam 20 iers ov orxogrefkl riform, wi wud hev a lojikl, kohirnt speling in ius xrewawt xe Ingliy-spiking werld.

channel properties

The example in the title of this talk is a so-called **erasure channel**

channel properties

The example in the title of this talk is a so-called **erasure channel** The **binary symmetric channel** is perhaps more typical.

channel properties

The example in the title of this talk is a so-called **erasure channel** The **binary symmetric channel** is perhaps more typical.

- messages are sequences of 0's and 1's

channel properties

The example in the title of this talk is a so-called **erasure channel** The **binary symmetric channel** is perhaps more typical.

- messages are sequences of 0's and 1's
- there is a transition probability that tells how likely it is that a 1 will be received as a 0 (or vice versa)

channel properties

The example in the title of this talk is a so-called **erasure channel** The **binary symmetric channel** is perhaps more typical.

- messages are sequences of 0's and 1's
- there is a transition probability that tells how likely it is that a 1 will be received as a 0 (or vice versa)

Other channels are also in wide use. For example in computer memory chips the probability that a 1 will decay to a 0, is much greater than the probability that a 0 will become a 1. This is an unsymmetric channel.

a generic channel

a generic channel

Errors may get introduced in the transmission channel.

a generic channel

Errors may get introduced in the transmission channel. In binary schemes we may think of the error as a separate “error vector” which the channel adds to the message.

a generic channel

Errors may get introduced in the transmission channel. In binary schemes we may think of the error as a separate "error vector" which the channel adds to the message.

- $\vec{m}$ is the message vector.

a generic channel

Errors may get introduced in the transmission channel. In binary schemes we may think of the error as a separate "error vector" which the channel adds to the message.

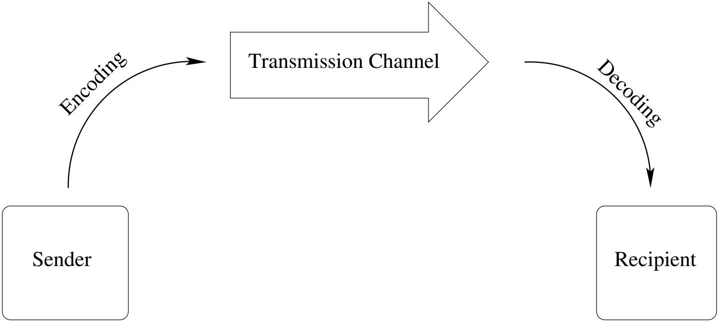
- $\vec{m}$ is the message vector.
- $\vec{e}$ is the error vector

a generic channel

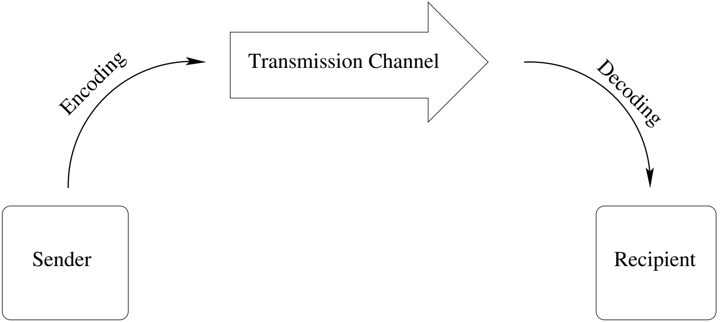
Errors may get introduced in the transmission channel. In binary schemes we may think of the error as a separate "error vector" which the channel adds to the message.

- $\vec{m}$ is the message vector.
- $\vec{e}$ is the error vector
- $\vec{r}$ is the received message; $\vec{r} = \vec{m} + \vec{e}$.

a channel with encoding/decoding

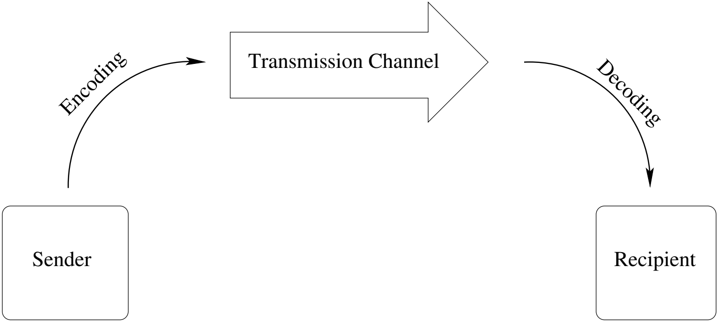


a channel with encoding/decoding



A simple encoding scheme: triplicate each symbol.

a channel with encoding/decoding



A simple encoding scheme: triplicate each symbol. Iiisss iiittt eeeaaasssyyy tttooo rrreeeaaaddd ttthhhiiisss???

notes on the triplication code

First note that the triplication code can deal with a lot of errors:

notes on the triplication code

First note that the triplication code can deal with a lot of errors:

Try decoding

hih eef lll qpp   mnm aba zyy   bbb eef   nnn fee baa rrr.

notes on the triplication code

First note that the triplication code can deal with a lot of errors:

Try decoding

hih eef lll qpp   mnm aba zyy   bbb eef   nnn fee baa rrr.

The same text with a burst of errors has a different meaning:

kkh eee lll ppp   mnm aaa yys   bbb ehe   nnn ege aaa rrs.

notes on the triplication code

First note that the triplication code can deal with a lot of errors:

Try decoding

hih eef lll qpp   mnm aba zyy   bbb eef   nnn fee baa rrr.

The same text with a burst of errors has a different meaning:

kkh eee lll ppp   mnm aaa yys   bbb ehe   nnn ege aaa rrs.

The triplication code handles some patterns of errors but not others.

terminology

- $\vec{m}$ is the message vector.

terminology

- $\vec{m}$ is the message vector.
- $\vec{c}$ is the encoded message.

terminology

- $\vec{m}$ is the message vector.
- $\vec{c}$ is the encoded message.
- $\vec{e}$ is the error vector

terminology

- $\vec{m}$ is the message vector.
- $\vec{c}$ is the encoded message.
- $\vec{e}$ is the error vector
- $\vec{r}$ is the received message; $\vec{r} = \vec{c} + \vec{e}$.

terminology

- $\vec{m}$ is the message vector.
- $\vec{c}$ is the encoded message.
- $\vec{e}$ is the error vector
- $\vec{r}$ is the received message; $\vec{r} = \vec{c} + \vec{e}$.
- $\vec{m'}$ is the decoded message;

terminology

- $\vec{m}$ is the message vector.
- $\vec{c}$ is the encoded message.
- $\vec{e}$ is the error vector
- $\vec{r}$ is the received message; $\vec{r} = \vec{c} + \vec{e}$.
- $\vec{m'}$ is the decoded message;
- Hopefully, $\vec{m} = \vec{m'}$.

ascii

- Perhaps you've heard the acronym ASCII?

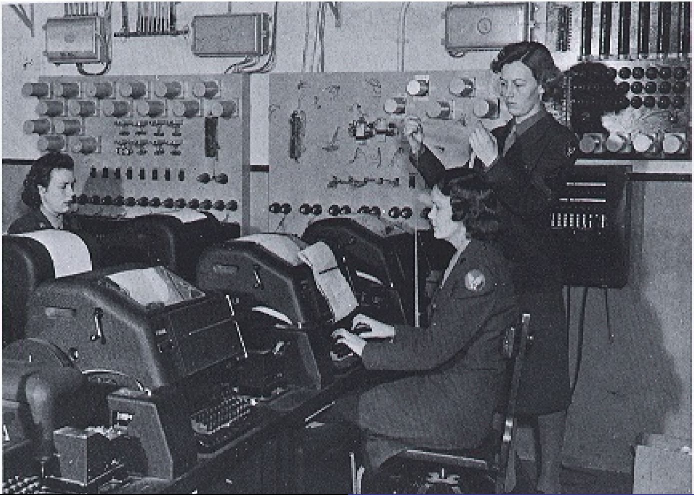
ascii

- Perhaps you've heard the acronym ASCII?
- American Standard Code for Information Interchange

ascii

- Perhaps you've heard the acronym ASCII?
- American Standard Code for Information Interchange
- Uses the numbers 0 to 127 – usually written in base-2 – to encode all the characters on the keyboard (plus a few others)

ascii is ancient



detecting an error

- In base-2 these ASCII characters all require 7 binary digits

detecting an error

- In base-2 these ASCII characters all require 7 binary digits
- Some bright person realized they could add an 8th bit called a **parity check bit** in such a way that single errors in transmission would get noticed.

detecting an error

- In base-2 these ASCII characters all require 7 binary digits
- Some bright person realized they could add an 8th bit called a **parity check bit** in such a way that single errors in transmission would get noticed.
- This is why, today, we have 8 bit bytes.

likelihood of missing an error

- We can recognize when a single error in one of the 8 bits of an ASCII message is made (when an error is detected we can ask for a retransmission).

likelihood of missing an error

- We can recognize when a single error in one of the 8 bits of an ASCII message is made (when an error is detected we can ask for a retransmission).
- Suppose that the transition probability is fairly large, say 5%. We say the channel is "noisy".

likelihood of missing an error

- We can recognize when a single error in one of the 8 bits of an ASCII message is made (when an error is detected we can ask for a retransmission).
- Suppose that the transition probability is fairly large, say 5%. We say the channel is "noisy".
- You can estimate the probability of having fault-free communication using the binomial distribution.

likelihood of missing an error

- We can recognize when a single error in one of the 8 bits of an ASCII message is made (when an error is detected we can ask for a retransmission).
- Suppose that the transition probability is fairly large, say 5%. We say the channel is "noisy".
- You can estimate the probability of having fault-free communication using the binomial distribution.
- $\binom{8}{0} \cdot (.95)^8(.05)^0 + \binom{8}{1} \cdot (.95)^7(.05)^1 \quad \approx \quad .94275$

likelihood of missing an error (cont.)

- In truth, things are even a little rosier since we can tolerate any odd number of errors:
$\binom{8}{0} \cdot (.95)^8(.05)^0 + \binom{8}{1} \cdot (.95)^7(.05)^1 + \binom{8}{3} \cdot (.95)^5(.05)^3 + \binom{8}{5} \cdot (.95)^3(.05)^5 + \binom{8}{7} \cdot (.95)^1(.05)^7 \approx .94819$

likelihood of missing an error (cont.)

- In truth, things are even a little rosier since we can tolerate any odd number of errors:
$\binom{8}{0} \cdot (.95)^8(.05)^0 + \binom{8}{1} \cdot (.95)^7(.05)^1 + \binom{8}{3} \cdot (.95)^5(.05)^3 + \binom{8}{5} \cdot (.95)^3(.05)^5 + \binom{8}{7} \cdot (.95)^1(.05)^7 \quad \approx \quad .94819$
- Notice how those multiple error patterns (3, 5 and 7 bit errors) make very little difference (they're quite rare!)

Fundamentally an error-correcting code is just some subset of the potential messages that can be sent. A practical consideration is that there should be some easy way to distinguish codewords from non-codewords.
Which of the following are codewords in the triplication code?

Fundamentally an error-correcting code is just some subset of the potential messages that can be sent. A practical consideration is that there should be some easy way to distinguish codewords from non-codewords.

Which of the following are codewords in the triplication code?

cccaaattt aaaaaa arrrdddvvvaaarrrkkk

Fundamentally an error-correcting code is just some subset of the potential messages that can be sent. A practical consideration is that there should be some easy way to distinguish codewords from non-codewords.

Which of the following are codewords in the triplication code?

cccaaattt aaaaaa arrrdddvvvaaarrrkkk

Which of the following are codewords in ASCII? (with parity checking)

Fundamentally an error-correcting code is just some subset of the potential messages that can be sent. A practical consideration is that there should be some easy way to distinguish codewords from non-codewords.

Which of the following are codewords in the triplication code?

cccaaattt aaaaaa arrrdddvvvaaarrrkkk

Which of the following are codewords in ASCII? (with parity checking)

11110000 11100001 11111110

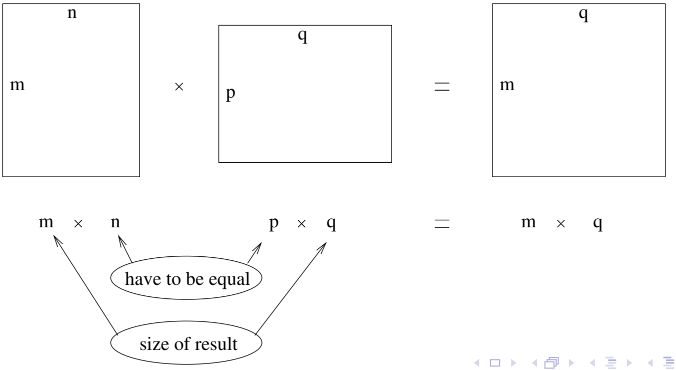
what is the matrix?

error detection as a linear algebraic operation

- Recall that (vector and) matrix multiplication require that the multiplicands be **conformable**

error detection as a linear algebraic operation

- Recall that (vector and) matrix multiplication require that the multiplicands be **conformable**



parity check matrices

We think of a received word as a vector $\vec{v}$. We carefully choose a matrix H so that whenever the matrix-vector product $H\vec{v}$ comes out zero we know that $\vec{v}$ is a codeword.
So, for instance, in the ASCII code (with parity check), the matrix is

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

H is known as a **parity check matrix**. The number of columns in H must be equal to the length of our codewords. The number of rows determines how big the vector $H\vec{v}$ will be.

syndromes

The vector one gets by multiplying a received word by H is called the **syndrome** of the received word. If the syndrome is a zero vector it means the received word was an unmodified codeword (in fact this is how one often defines what the set of codewords is). If the syndrome is something other than the zero vector we know we have an error. We *hope* the syndrome will allow us to figure-out where the error occurred.

a single-error, error-correcting code

In about 1950, Richard Hamming introduced the (7,4) single error correcting Hamming code. Probably Doctor Hamming was thinking about syndromes. What should the size of the syndrome be so that the non-zero syndromes will be able to distinguish the locations of error?

a single-error, error-correcting code

In about 1950, Richard Hamming introduced the (7,4) single error correcting Hamming code. Probably Doctor Hamming was thinking about syndromes. What should the size of the syndrome be so that the non-zero syndromes will be able to distinguish the locations of error?
If the syndromes are length 3 vectors then there are a total of 8 distinct possible syndromes. But $[0, 0, 0]^T$ must be excluded because that identifies codewords.

a single-error, error-correcting code

In about 1950, Richard Hamming introduced the (7,4) single error correcting Hamming code. Probably Doctor Hamming was thinking about syndromes. What should the size of the syndrome be so that the non-zero syndromes will be able to distinguish the locations of error?

If the syndromes are length 3 vectors then there are a total of 8 distinct possible syndromes. But $[0, 0, 0]^T$ must be excluded because that identifies codewords.

So, is it possible to have the seven non-zero syndromes of length 3 correspond to error positions?

the Hamming code via parity check matrix

Well, yes! Provided there are no more than seven possible errors!

the Hamming code via parity check matrix

Well, yes! Provided there are no more than seven possible errors!

$$H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

more on the Hamming code

Since matrix multiplication is linear,

$$H \cdot \vec{r} \quad = \quad H \cdot (\vec{c} + \vec{e}) \quad = \quad H \cdot \vec{c} + H \cdot \vec{e} \quad = \quad \vec{0} + H \cdot \vec{e}$$

more on the Hamming code

Since matrix multiplication is linear,

$$H \cdot \vec{r} \quad = \quad H \cdot (\vec{c} + \vec{e}) \quad = \quad H \cdot \vec{c} + H \cdot \vec{e} \quad = \quad \vec{0} + H \cdot \vec{e}$$

If $\vec{e}$ represents a bit error in just one of the seven positions, then $\vec{e}$ is what is often called a **standard basis vector**; it is all zeros except for one position where it has a 1.

more on the Hamming code

Since matrix multiplication is linear,

$$H \cdot \vec{r} \quad = \quad H \cdot (\vec{c} + \vec{e}) \quad = \quad H \cdot \vec{c} + H \cdot \vec{e} \quad = \quad \vec{0} + H \cdot \vec{e}$$

If $\vec{e}$ represents a bit error in just one of the seven positions, then $\vec{e}$ is what is often called a **standard basis vector**; it is all zeros except for one position where it has a 1.
In this case the matrix-vector product $H\vec{e}$ just serves to "pick out" one column of H.

more more on the Hamming code

Because of the clever way that H was constructed, the syndrome of a received vector is either $\vec{0}$ (the received message is correct) or it is a non-zero length 3 vector that can be read as a number between 1 and 7 in binary (the received word has an error in the corresponding position).

null spaces

We've defined a linear code to be the set

$$\{\vec{v} \mid H \cdot \vec{v} = \vec{0}\}.$$

null spaces

We've defined a linear code to be the set

$$\{\vec{v} \mid H \cdot \vec{v} = \vec{0}\}.$$

This is more generally known as the **null space** of the matrix H (or, more properly, the linear transformation associated to H).

generators instead

It seems preferable to many that we have an explicit way to construct the codewords (as opposed to a way of verifying that a vector is a codeword).

generators instead

It seems preferable to many that we have an explicit way to construct the codewords (as opposed to a way of verifying that a vector is a codeword).
Finding a basis for the space of all codewords would do the trick.

generators instead

It seems preferable to many that we have an explicit way to construct the codewords (as opposed to a way of verifying that a vector is a codeword).
Finding a basis for the space of all codewords would do the trick.
A **generator matrix** for a code C is a matrix whose rows are a basis for C.

generator matrix of the (7,4) Hamming code

$$G = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

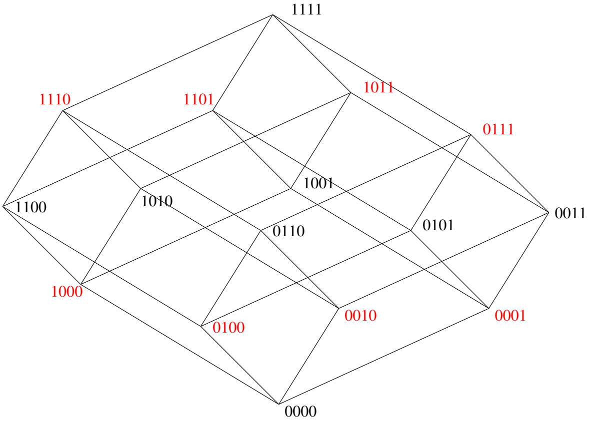
generator matrix of the (7,4) Hamming code

$$G = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

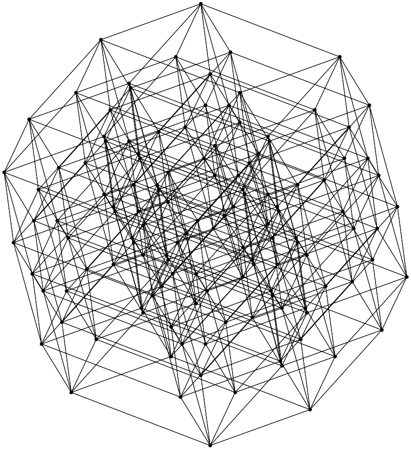
or

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

a hypercube


1111
1110
1101
1011
0111
1100
1010
1001
0110
0101
0011
1000
0100
0010
0001
0000

a bigger hypercube



weights and weight enumerators

The **Hamming distance** between two vectors in a hypercube is the number of positions in which they differ.

weights and weight enumerators

The **Hamming distance** between two vectors in a hypercube is the number of positions in which they differ.
The **weight** of a vector is its distance from $\vec{0}$. In other words, the number of 1's in it. We usually write $w(\vec{v})$ for the Hamming weight.

weights and weight enumerators

The **Hamming distance** between two vectors in a hypercube is the number of positions in which they differ.
The **weight** of a vector is its distance from $\vec{0}$. In other words, the number of 1's in it. We usually write $w(\vec{v})$ for the Hamming weight.
We use a formal polynomial called the **weight enumerator** of a code C to keep track of how many vectors of which particular weights it has.

example: the (7,4) Hamming code again

Let's call the four vectors in the rows of the generator matrix of the Hamming code c_1, c_2, c_3 and c_4 respectively.
The elements (codewords in this code are:

$\emptyset$	0000000
c_1	1000011
c_2	0100101
c_3	0010110
c_4	0001111
$c_1 + c_2$	1100110
$c_1 + c_3$	1010101
$c_1 + c_4$	1001100

$c_2 + c_3$	0110011
$c_2 + c_4$	0101010
$c_3 + c_4$	0011001
$c_1 + c_2 + c_3$	1110000
$c_1 + c_2 + c_4$	1101001
$c_1 + c_3 + c_4$	1011010
$c_2 + c_3 + c_4$	0111100
$c_1 + c_2 + c_3 + c_4$	1111111

There are: a single vector of weight 0, seven vectors each of weight 3 and weight 4, and a single vector of weight 7. This makes the weight enumerator

$$w_C(x) = 1 + 7x^3 + 7x^4 + x^7.$$

Often, we make this polynomial into a homogeneous one. You can interpret this as x's power counts the number of 1's, y's power counts the number of 0's. Naturally, the sum of these will be the length of the code.

$$w_C(x, y) = x^0y^7 + 7x^3y^4 + 7x^4y^3 + x^7y^0.$$

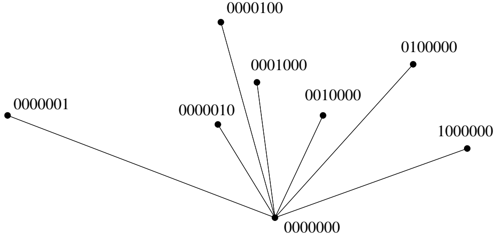
the geometric viewpoint: sphere packing

We want to have codewords that are mutually at large Hamming distance. This can be interpreted geometrically — we want each codeword to have a large "sphere" around it that doesn't contain other codewords. Indeed, we want the spheres around codewords to be disjoint!

more geometry

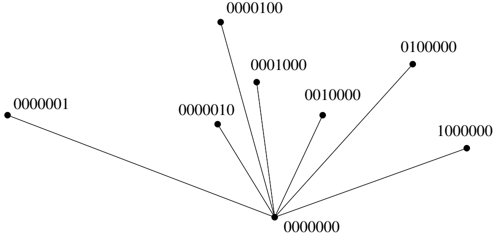
more geometry

This is the sphere around the zero vector in Q_7.



more geometry

This is the sphere around the zero vector in Q_7.



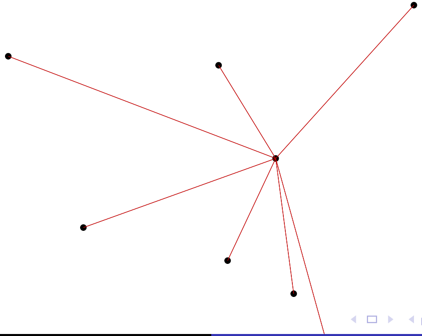
Other spheres look "rounder" than this...

more on sphere packing

Because most of the codes we study are actually vector spaces, the "spheres around codewords" all look the same. You get them by translating the sphere around zero.

more on sphere packing

Because most of the codes we study are actually vector spaces, the "spheres around codewords" all look the same. You get them by translating the sphere around zero. This is the sphere around another vector in Q_7.



definition of a dual code

Given a length n linear code C having parity check matrix H and generator matrix G, there is another code denoted $C^{\perp}$ that has the roles of generator and parity check matrix reversed.
We know that $\text{Null}(H) = \text{Rank}(G)$, so, the rank-nullity theorem from linear algebra tells us that these codes have dimensions that are complementary (in the sense that they sum to n).

self-dual codes

We've seen that a binary vector can be perpendicular to itself. Would you believe that an entire vector space can consist of vectors that are all mutually perpendicular (including to themselves)?

self-dual codes

We've seen that a binary vector can be perpendicular to itself. Would you believe that an entire vector space can consist of vectors that are all mutually perpendicular (including to themselves)?

Binary is kind of funny…

self-dual codes

We've seen that a binary vector can be perpendicular to itself. Would you believe that an entire vector space can consist of vectors that are all mutually perpendicular (including to themselves)?

Binary is kind of funny...

Certain codes known as **self dual** codes are actually equal to their own duals. A self-dual code will have dimension $n/2$ and a generator matrix for it will also be a parity check matrix for it (and vice versa).

example: the Golay code

There is a particularly interesting code of length 24 and dimension 12 that has minimum weight 8. (Thus it can correct 3 and detect 4 errors.)
The **Golay code** has a generator matrix of the form

$$\left[\; I \; \middle| \; J - A \; \right].$$

where I represents a 12×12 identity matrix, J is the 12×12 matrix having all entries 1, and A is the adjacency matrix of a regular icosahedron.

the MacWilliams identity

There is a remarkable connection between the weight enumerator of a code C and the weight enumerator of its dual code $C^{\perp}$.

Jesse MacWilliams and Vera Pless (two prominent female coding theorists) independently discovered this connection – although in slightly different forms. The "MacWilliams identities" and the "Pless power moments" are equivalent and both can be encapsulated by the following incredible expression.

the MacWilliams identity

There is a remarkable connection between the weight enumerator of a code C and the weight enumerator of its dual code $C^{\perp}$.

Jesse MacWilliams and Vera Pless (two prominent female coding theorists) independently discovered this connection – although in slightly different forms. The "MacWilliams identities" and the "Pless power moments" are equivalent and both can be encapsulated by the following incredible expression.

$$w_{C^{\perp}}(x, y) = \frac{1}{|C|} w_C(y + x, y - x).$$

homogeneity humor

Now you know why people like homogeneous polynomials for their weight enumerators...

back to the Golay code

Because its minimum weight is so large, and it is self-dual, its weight distribution is so thoroughly constrained by the MacWilliams identity that it is (in fact) uniquely determined. To save space I'll write it in inhomogeneous form:

$$1 + 759x^8 + 2576x^{12} + 759x^{16} + x^{24}.$$

permutation groups

A one-to-one and onto map from $\{1, 2, 3, \ldots, n\}$ to itself is a **permutation**. The collection of all such maps forms the **symmetric group** S_n.

More generally, a **group** is a set of objects endowed with an associative binary operation with two simple provisos:

permutation groups

A one-to-one and onto map from $\{1, 2, 3, \ldots, n\}$ to itself is a **permutation**. The collection of all such maps forms the **symmetric group** S_n.

More generally, a **group** is a set of objects endowed with an associative binary operation with two simple provisos:

- There is an element in the group that is an identity for the operation.

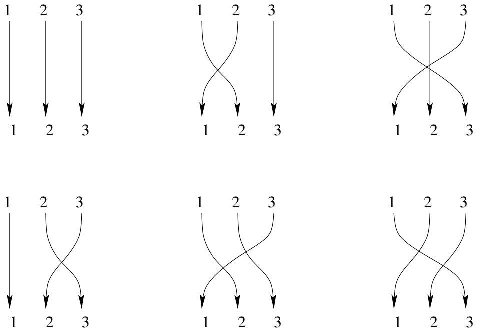
permutation groups

A one-to-one and onto map from $\{1, 2, 3, \ldots, n\}$ to itself is a **permutation**. The collection of all such maps forms the **symmetric group** S_n.

More generally, a **group** is a set of objects endowed with an associative binary operation with two simple provisos:

- There is an element in the group that is an identity for the operation.
- The group is closed with respect to the operation.

S_3



other kinds of groups

- Abstract groups given in terms of generators and relations.

other kinds of groups

- Abstract groups given in terms of generators and relations.
- The set of symmetries of a geometric object. (E.g. the regular dodecahedron.)

other kinds of groups

- Abstract groups given in terms of generators and relations.
- The set of symmetries of a geometric object. (E.g. the regular dodecahedron.)
- Matrix groups.

a little terminology

The word “order” gets used in two different senses. The context is supposed to make it clear what is meant.

- The order of an element g is the smallest number such that $g^n = 1$.

a little terminology

The word "order" gets used in two different senses. The context is supposed to make it clear what is meant.

- The order of an element g is the smallest number such that $g^n = 1$.
- The order of a group G is its size as a set.

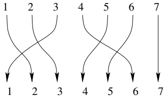
group of a code

group of a code

For any particular code C, we can view permutations of length n as acting on the coordinates (i.e. positions) of C.

group of a code

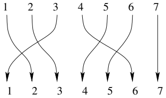
For any particular code C, we can view permutations of length n as acting on the coordinates (i.e. positions) of C.

If we apply the permutation 

to the codeword 0111100 we will get 1011010. Now this happens to be another codeword (trust me).

group of a code

For any particular code C, we can view permutations of length n as acting on the coordinates (i.e. positions) of C.

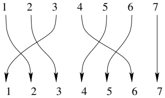
If we apply the permutation 

to the codeword 0111100 we will get 1011010. Now this happens to be another codeword (trust me).

The group Aut(C) is the group of all permutations that act in such a way that they send any codeword to some other (or even the same) codeword.

group of a code

For any particular code C, we can view permutations of length n as acting on the coordinates (i.e. positions) of C.

If we apply the permutation 

to the codeword 0111100 we will get 1011010. Now this happens to be another codeword (trust me).

The group Aut(C) is the group of all permutations that act in such a way that they send any codeword to some other (or even the same) codeword.

In other words, the group Aut(C) fixes C as a set.

direct products

Consider the code whose generator matrix is

$$G = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

This code only has four codewords: 00000000, 11100000, 00011111, and 11111111.

direct products

Consider the code whose generator matrix is

$$G = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

This code only has four codewords: 00000000, 11100000, 00011111, and 11111111.

A permutation in $\mathrm{Aut}(C)$ will clearly have no problem with 00000000 or with 11111111 – these will automagically get sent to themselves.

No permutation can interchange the other two codewords because they are of different weights.

more about direct products

We can permute the first 3 coordinates as we wish and likewise we can permute the last 5 at will.

more about direct products

We can permute the first 3 coordinates as we wish and likewise we can permute the last 5 at will.
$\mathrm{Aut}(C) = S_3 \otimes S_5$.

more about direct products

We can permute the first 3 coordinates as we wish and likewise we can permute the last 5 at will.
$\mathrm{Aut}(C) = S_3 \otimes S_5$.
(This is a direct product of groups.)

simple groups

A fairly broad oversimplification:

simple groups

A fairly broad oversimplification:
One can use the direct product of groups to form larger groups out of smaller ones.

simple groups

A fairly broad oversimplification:
One can use the direct product of groups to form larger groups out of smaller ones.
Simple groups are groups that *can't* be realized in this fashion.

simple groups

A fairly broad oversimplification:
One can use the direct product of groups to form larger groups out of smaller ones.
Simple groups are groups that *can't* be realized in this fashion.
So simple groups have somewhat the same relationship to the set of all finite groups as prime numbers have to the integers.

some famous simple groups

So far we've looked fairly closely at two binary error-correcting codes: the (7,4)-Hamming code and the (24,12)-Golay code.

some famous simple groups

So far we've looked fairly closely at two binary error-correcting codes: the (7,4)-Hamming code and the (24,12)-Golay code. The automorphism group of the Hamming code is isomorphic to $PSL(2,7)$ — at order 168, it is the second smallest non-abelian simple group. (The smallest is A_5 of order 60.)

some famous simple groups

So far we've looked fairly closely at two binary error-correcting codes: the (7,4)-Hamming code and the (24,12)-Golay code. The automorphism group of the Hamming code is isomorphic to $PSL(2,7)$ — at order 168, it is the second smallest non-abelian simple group. (The smallest is A_5 of order 60.) The automorphism group of the Golay code is the Mathieu group M_{24}. It (and some of its important subgroups) form the earliest known family of sporadic simple groups. The order of M_{24} is $2^{10} \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 23$

some famous simple groups

So far we've looked fairly closely at two binary error-correcting codes: the (7,4)-Hamming code and the (24,12)-Golay code. The automorphism group of the Hamming code is isomorphic to $PSL(2,7)$ — at order 168, it is the second smallest non-abelian simple group. (The smallest is A_5 of order 60.) The automorphism group of the Golay code is the Mathieu group M_{24}. It (and some of its important subgroups) form the earliest known family of sporadic simple groups. The order of M_{24} is $2^{10} \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 23$
That's about 244 million elements!

the classification

The classification of all the finite simple groups was completed in the previous millennium (right near the end). The proof of this “theorem” consists of several hundred separate journal articles published between 1955 and about 1983.

the classification

The classification of all the finite simple groups was completed in the previous millennium (right near the end). The proof of this “theorem” consists of several hundred separate journal articles published between 1955 and about 1983.
Daniel Gorenstein was working on a revised proof up until his death in 1992. Others have continued that work, and a revised proof amounting to about 5,000 pages should be ready soon.

the classification

The classification of all the finite simple groups was completed in the previous millennium (right near the end). The proof of this “theorem” consists of several hundred separate journal articles published between 1955 and about 1983.

Daniel Gorenstein was working on a revised proof up until his death in 1992. Others have continued that work, and a revised proof amounting to about 5,000 pages should be ready soon. (It’s in several volumes.)

what the “enormous theorem” says

Every finite simple group is either

- A cyclic group of prime order.

what the “enormous theorem” says

Every finite simple group is either

- A cyclic group of prime order.
- An alternating group of degree ≥ 5.

what the "enormous theorem" says

Every finite simple group is either

- A cyclic group of prime order.
- An alternating group of degree ≥ 5.
- A group of Lie type.

what the “enormous theorem” says

Every finite simple group is either

- A cyclic group of prime order.
- An alternating group of degree ≥ 5.
- A group of Lie type.
- One of 26 sporadic simple groups.

more on automorphism groups

Let's return to a very simple example. Consider the code whose generator matrix is

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

more on automorphism groups

Let's return to a very simple example. Consider the code whose generator matrix is

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

This code contains 00000000, 11111111, 11110000, and 00001111.

We can apply any permutation in S_8 that permutes the first 4 coordinates amongst themselves and the last 4 coordinates amongst *themselves*.

more on automorphism groups

Let's return to a very simple example. Consider the code whose generator matrix is

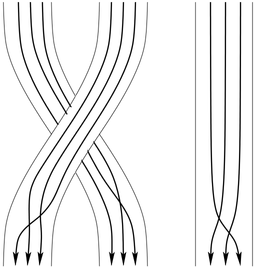
$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

This code contains 00000000, 11111111, 11110000, and 00001111.

We can apply any permutation in S_8 that permutes the first 4 coordinates amongst themselves and the last 4 coordinates amongst *themselves*.

BUT! We can also interchange these two blocks of coordinates.

wreath products

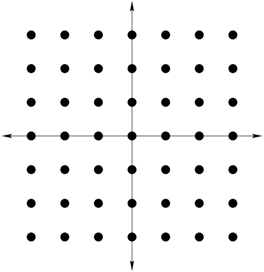


2-d

In 2-dimensional space there are essentially just two lattices.

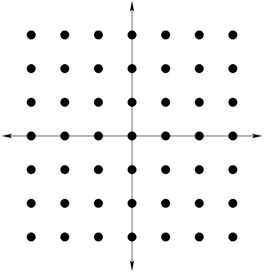
2-d

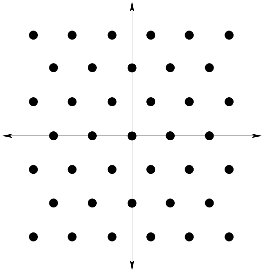
In 2-dimensional space there are essentially just two lattices.



2-d

In 2-dimensional space there are essentially just two lattices.





3-d

In 3-d we have several lattices. They have been studied extensively by Crystallographers.

- Z^3 The integer lattice in 3 dimensions.
- fcc The face-centered cubic lattice.
- bcc The body-centered cubic lattice.
- tet The tetrahedral lattice. Technically this is not a lattice.

4-d ?

We'll just "look" at one example, D_4. This lattice is simply the subset of $\mathbb{Z}^4$ where the sum of the coordinates is even.

definitions

A **lattice** in $\mathbb{R}^m$ is a discrete set L of points that forms an abelian subgroup of $\mathbb{R}^m$ (the operation is the addition of vectors)

The zero vector $\vec{0}$ must be in L

If $\vec{u}$ and $\vec{v}$ are elements of L, then $\vec{u} + \vec{v}$ and $\vec{u} - \vec{v}$ must also be in L.

A set of lattice points $\{\vec{v}_1, \vec{v}_2, \vec{v}_3, \dots \vec{v}_n\}$ such that every element of L can be expressed as a sum

$$\sum k_i \cdot \vec{v}_i$$

(where the coefficients k_i are integers) is said to be a **basis** for the lattice.

more definitions

Given a basis $\{\vec{v}_1, \vec{v}_2, \vec{v}_3, \ldots \vec{v}_n\}$ for a lattice L, the **fundamental polytope** of L is the set

$$\left\{ r_1\vec{v}_1 + r_2\vec{v}_2 + r_3\vec{v}_3 + \ldots + r_n\vec{v}_n \,\middle|\, 0 \leq r_i < 1 \text{ for all } i \right\}$$

There may be many choices of a basis for a lattice, and many different fundamental polytopes, but the volumes of all such fundamental polytopes must be equal.

The square of the volume of a fundamental polytope is called the **determinant** of L.

Let M be the $n \times m$ matrix whose rows are the coordinates of the basis vectors. The matrix $A = MM^T$ is called the **Gram matrix** of L, its determinant is synonymous with the

further definitions

The **Voronoi cells** are the sets of points that are closer to a given lattice point than they are to any other. Voronoi cells and fundamental polytopes and any other regions that contain one lattice point and whose translates tile $\mathbb{R}^n$ are called **fundamental regions**

The **packing radius** is the largest real number so that spheres of this radius, centered at the lattice points will not intersect.

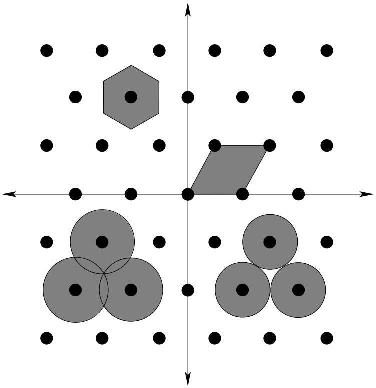
The **covering radius** is the smallest real number so that if spheres of this radius are centered at the lattice points, every point in $\mathbb{R}^n$ will be in a sphere.

more further definitions

Deep holes are the last guys to get covered as we approach the covering radius.
The **kissing number** is the number of packing spheres that touch a particular packing sphere.
The **density** of a lattice is the proportion of $\mathbb{R}^n$ that is inside the packing spheres. Alternatively, it it the fraction

$$\frac{\text{volume of a packing sphere}}{\text{volume of a fundamental region}}.$$

a picture



laminated lattices in dimensions 1–3

- $\mathbb{Z}$ is (by definition) the laminated lattice in dimension 1.

laminated lattices in dimensions 1–3

- $\mathbb{Z}$ is (by definition) the laminated lattice in dimension 1.
- The hexagonal packing is the laminated lattice in dimension 2.

laminated lattices in dimensions 1–3

- $\mathbb{Z}$ is (by definition) the laminated lattice in dimension 1.
- The hexagonal packing is the laminated lattice in dimension 2.
- Stacking up copies of the hexagonal packing will give you the laminated lattice in dimension 3.

laminated lattices in dimensions 1–3

- $\mathbb{Z}$ is (by definition) the laminated lattice in dimension 1.
- The hexagonal packing is the laminated lattice in dimension 2.
- Stacking up copies of the hexagonal packing will give you the laminated lattice in dimension 3.
- And so on. . .

connecting lattices to codes

Perhaps it's clear that a lattice in $\mathbb{R}^n$ is geometrically similar to an error-correcting code in Q_n. Both consist of a set of "centers" such that spheres centered at these centers are disjoint.
High density for a lattice is analogous to good error-correcting properties for a code.

construction A

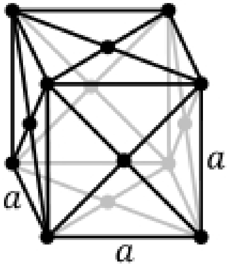
Construction A is an explicit connection. It allows you to construct a lattice in $\mathbb{R}^n$ from a code of length n.
A vector in $\mathbb{Z}^n$ will be a lattice point iff it is in C after reduction mod 2.

fcc

The face-centered cubic lattice is the lattice we get in $\mathbb{R}^3$ by using construction A with the "even" code whose generator matrix is

$$G = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}.$$

fcc picture



(stolen from Wikipedia)

H_8

If we add an overall parity check bit to the (7,4) Hamming code we get a code known as the (8,4) extended Hamming code. Its generator matrix is

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}.$$

E_8

Applying construction A to the (8,4) extended Hamming code yields the E_8 lattice (sometimes called the Gossett lattice).
E_8 is the root lattice of the exceptional Lie algebra that also goes by the name E_8.
This is a really good lattice. It solves the lattice sphere-packing problem and the kissing number problem in dimension 8.

the sphere-packing problem

What is the densest arrangement of non-overlapping balls in $\mathbb{R}^n$?

the sphere-packing problem

What is the densest arrangement of non-overlapping balls in $\mathbb{R}^n$?
In $\mathbb{R}^2$ this is obvious.

the sphere-packing problem

What is the densest arrangement of non-overlapping balls in $\mathbb{R}^n$?

In $\mathbb{R}^2$ this is obvious. In $\mathbb{R}^3$ it *seems* obvious (the fcc lattice ought to do the trick).

the sphere-packing problem

What is the densest arrangement of non-overlapping balls in $\mathbb{R}^n$?
In $\mathbb{R}^2$ this is obvious. In $\mathbb{R}^3$ it *seems* obvious (the fcc lattice ought to do the trick). The Kepler conjecture dates back to 1611.

the sphere-packing problem

What is the densest arrangement of non-overlapping balls in $\mathbb{R}^n$?

In $\mathbb{R}^2$ this is obvious. In $\mathbb{R}^3$ it *seems* obvious (the fcc lattice ought to do the trick). The Kepler conjecture dates back to 1611. Thomas Hales at University of Pittsburgh has written a proof, involving some massive computer assistance, that the referees are 99% sure is right...

Weird but true

We know the densest lattice packings in dimensions 1 through 8.

We also know the densest lattice packing in dimension 24 because of the remarkable Leech lattice Λ_{24}.

Weird but true

We know the densest lattice packings in dimensions 1 through 8.
We also know the densest lattice packing in dimension 24 because of the remarkable Leech lattice Λ_{24}.
The Leech lattice can be constructed in dozens of ways, but one of them is a slight generalization of construction A applied to the binary Golay code.

good sources

- Coding theory
  - "Introduction to the theory of error-correcting codes" by Vera Pless
  - "Fundamentals of error-correcting codes" by W. Cary Huffman and Vera Pless
  - "The theory of error-correcting codes" by F. J. MacWilliams and N. J. A. Sloane
- Groups
  - "Contemporary Abstract Algebra" by Joe Gallian
  - "Symmetry and the monster" by Mark Ronan

more literature

- Lattices
  - "Sphere packings, lattices and groups"
    by J. H. Conway and N. J. A. Sloane