A progress report on GUAVA

a free and open-source coding theory package

David Joyner

GAP conference, Braunschweig, Sep. 2007

GUAVA homepage:
http://sage.math.washington.edu/home/wdj/guava/

GUAVA homepage:
http://sage.math.washington.edu/home/wdj/guava/

Recent contributors: David Joyner, Cen Tjhal, Robert Miller, Tom Boothby (Joe Fields, U Conn. Prof., plans to help as well)

Figure: Robert Miller, Univ Wash, grad student

Figure: Cen Tjhal ("CJ"), Univ Plymouth, grad student

Figure: Tom Boothby, Univ Wash, undergrad

Outline

Basic notation and terms

There are two areas where group theory impacts most seriously coding theory:

- automorphism groups of codes and associated $\mathbb{F}[G]$-modules,
- invariance properties of wt enumerator polys of f.s.d. codes,
- some improved decoding algorithms.

These will be discussed.

(Also, work of R. Liebler, K.-H. Zimmermann, A. Kerber, A. Kohnert, is interesting....)

Basic notation and terms

A **code** is a linear block code over a finite field $\mathbb{F} = GF(q)$, i.e., a subspace of $\mathbb{F}^n$ with a fixed basis. In the exact sequence

$$0 \to \mathbb{F}^k \xrightarrow{G} \mathbb{F}^n \xrightarrow{H} \mathbb{F}^{n-k} \to 0, \tag{1}$$

- G represents a generating matrix (and $\mathbf{m} \longmapsto \mathbf{m}G$ the **encoder**)
- H represents a check matrix,
- $C = Image(G) = Kernel(H)$ is the code.

Hamming weight, etc.

Hamming metric is the function $d : \mathbb{F}^n \times \mathbb{F}^n \to \mathbb{R}$,

$$d(\mathbf{v}, \mathbf{w}) = |\{i \mid v_i \neq w_i\}| = d(\mathbf{v} - \mathbf{w}, \mathbf{0}).$$

- the **weight** is $wt(\mathbf{c}) = d(\mathbf{c}, \mathbf{0})$
- **minimum distance of** C is defined to be the number $d(C) = \min_{\mathbf{c} \neq \mathbf{0}} wt(\mathbf{c})$.
- **weight distribution** (or spectrum) of C is the $(n+1)$-tuple $spec(C) = (A_0, A_1, ..., A_n)$, where

$$A_i = |\{\mathbf{c} \in C \mid wt(\mathbf{c}) = i\}|.$$

Corresponding GAP functions.

Some associated GAP functions

- `AClosestVectorCombinationsMatFFEVecFFECoords` (for $d(C)$)
- `DistancesDistributionMatFFEVecFFE` (for $spec(C)$, GUAVA manual has typo)
- `WeightVecFFE`, `DistanceVecFFE` (for $wt(v)$, $d(v,w)$)
- `ConwayPolynomial` (calls Frank's GPL'd database of polynomials used to construct $GF(q)$)
- `RandomPrimitivePolynomial` (for random cyclic codes?)

Associated GUAVA functions

Some associated GUAVA functions

- `MinimumDistance`
- `MinimumDistanceLeon` (does not call Leon's C code)
- `MinimumDistanceRandom`
- `CoveringRadius`
- `WeightDistribution` (for *spec*(C), should call Leon?)
- `DistancesDistribution` (the distribution of the distances of elements of C to a vector w)

Automorphism group of a code

What is an automorphism of a code?

Let S_n denote the symmetric group on n letters. The **(permutation) automorphism group** of a code C of length n is simply the group

$$\mathrm{Aut}(C) = \{\sigma \in S_n \mid (c_1, ..., c_n) \in C \implies (c_{\sigma(1)}, ..., c_{\sigma(n)}) \in C\}.$$

There are no known methods for computing these groups which are polynomial time in the length n of C.

Automorphism group of a code

What is an automorphism of a code?

Let S_n denote the symmetric group on n letters. The **(permutation) automorphism group** of a code C of length n is simply the group

$$\mathrm{Aut}(C) = \{\sigma \in S_n \mid (c_1, ..., c_n) \in C \implies (c_{\sigma(1)}, ..., c_{\sigma(n)}) \in C\}.$$

There are no known methods for computing these groups which are polynomial time in the length n of C.

Automorphism group of a code

What is an automorphism of a code?

Let S_n denote the symmetric group on n letters. The **(permutation) automorphism group** of a code C of length n is simply the group

$$\mathrm{Aut}(C) = \{\sigma \in S_n \mid (c_1, ..., c_n) \in C \implies (c_{\sigma(1)}, ..., c_{\sigma(n)}) \in C\}.$$

There are no known methods for computing these groups which are polynomial time in the length n of C.

Automorphism group of a code

If

(a) $C_1, C_2 \subset \mathbb{F}^n$ are codes, and
(b) $\exists \sigma \in S_n$ for which $(c_1, ..., c_n) \in C_1 \iff (c_{\sigma(1)}, ..., c_{\sigma(n)}) \in C_2$,

then $C_1 \cong C_2$ (i.e., C_1 and C_2 are **permutation equivalent**).

In GUAVA:

`IsEquivalent( C1, C2 )` and `CodeIsomorphism(C1, C2)`

The parameters dimension and minimum distance are *invariants*:
$C_1 \cong C_2 \implies \dim(C_1) = \dim(C_2)$ and $d(C_1) = d(C_2)$.

Automorphism group of a code

If

(a) $C_1, C_2 \subset \mathbb{F}^n$ are codes, and
(b) $\exists \sigma \in S_n$ for which $(c_1, ..., c_n) \in C_1 \iff (c_{\sigma(1)}, ..., c_{\sigma(n)}) \in C_2$,

then $C_1 \cong C_2$ (i.e., C_1 and C_2 are **permutation equivalent**).

In `GUAVA`:

`IsEquivalent( C1, C2 )` and `CodeIsomorphism(C1, C2)`

The parameters dimension and minimum distance are *invariants*:
$C_1 \cong C_2 \implies \dim(C_1) = \dim(C_2)$ and $d(C_1) = d(C_2)$.

Automorphism group of a code

C be a $[n, k]$-code, $G = \mathrm{Aut}(C) =$ (perm.) aut. gp of C.

Define $\rho : G \to GL_k(\mathbb{F})$, by

$$\sigma \longmapsto ((c_1, ..., c_n) \in C \longmapsto (c_{\sigma^{-1}(1)}, ..., c_{\sigma^{-1}(n)}) \in C).$$

Therefore, C is a (modular) representation space of G.

Open Problem: Determine explicitly this representation for common families of codes.

Automorphism group of a code

C be a $[n, k]$-code, $G = \mathrm{Aut}(C) =$ (perm.) aut. gp of C.

Define $\rho : G \to GL_k(\mathbb{F})$, by

$$\sigma \longmapsto ((c_1, ..., c_n) \in C \longmapsto (c_{\sigma^{-1}(1)}, ..., c_{\sigma^{-1}(n)}) \in C).$$

Therefore, C is a (modular) representation space of G.

Open Problem: Determine explicitly this representation for common families of codes.

Automorphism group of a code

C be a $[n, k]$-code, $G = \mathrm{Aut}(C) =$ (perm.) aut. gp of C.

Define $\rho : G \to GL_k(\mathbb{F})$, by

$$\sigma \longmapsto ((c_1, ..., c_n) \in C \longmapsto (c_{\sigma^{-1}(1)}, ..., c_{\sigma^{-1}(n)}) \in C).$$

Therefore, C is a (modular) representation space of G.

Open Problem: Determine explicitly this representation for common families of codes.

Automorphism group of a code

C be a $[n, k]$-code, $G = \mathrm{Aut}(C) =$ (perm.) aut. gp of C.

Define $\rho : G \to GL_k(\mathbb{F})$, by

$$\sigma \longmapsto ((c_1, ..., c_n) \in C \longmapsto (c_{\sigma^{-1}(1)}, ..., c_{\sigma^{-1}(n)}) \in C).$$

Therefore, C is a (modular) representation space of G.

Open Problem: Determine explicitly this representation for common families of codes.

Leon's code.

Leon's C code for computing automorphism groups of matrices and designs and linear codes is now GPL'd. Good news:

- it's GPL'd, optimized C code,
- new developers are working on GUAVA!

Drawbacks:

- it has memory leaks and "home-brewed" finite fields (should use Conway polynomials),
- GUAVA only interfaces a small part of what it does.

Robert Miller and Tom Boothby recently worked on fixing up Leon's code.

Leon's code.

GUAVA functions interfacing with Leon's code:

- `IsEquivalent`,
- `CodeIsomorphism`,
- `AutomorphismGroup`,
- `ConstantWeightSubcode`,
- `PermutationDecode` - see below.

Example (Aut gp of a code)

$GL(2,\mathbb{C})$ acts on the projective line $\mathbb{P}^1$ by: $z \longmapsto \frac{az+b}{cz+d}$, $\begin{pmatrix} a & b \\ c & d \end{pmatrix} \in GL(2,\mathbb{C})$.

$$\mathrm{Aut}(\mathbb{P}^1) = PGL(2,F)$$

divisor on $\mathbb{P}^1$ = element of $\mathbb{Z}[\mathbb{P}^1]$
= formal $\mathbb{Z}$-linear sum of points

Example (Aut gp of a code)

$GL(2,\mathbb{C})$ acts on the projective line $\mathbb{P}^1$ by: $z \longmapsto \frac{az+b}{cz+d}$, $\begin{pmatrix} a & b \\ c & d \end{pmatrix} \in GL(2,\mathbb{C})$.

$$\mathrm{Aut}(\mathbb{P}^1) = PGL(2, F)$$

divisor on $\mathbb{P}^1$ = element of $\mathbb{Z}[\mathbb{P}^1]$
= formal $\mathbb{Z}$-linear sum of points

Example (Aut gp of a code)

divisor of f = $\mathrm{div}(f)$ = formal sum of zeros of f minus the poles.

$D = n_1 P_1 + ... + n_k P_k$ a divisor then $\mathrm{supp}(D) = \{P_1, ..., P_k\}$ is the **support** of D.

Example: f = polynomial of degree n in x $\implies$ $\mathrm{div}(f) = P_1 + ... + P_n - n\infty$, $\mathrm{supp}(\mathrm{div}(f)) = \{P_1, ..., P_n, \infty\}$, where $zeros(f) = \{P_1, ..., P_n\}$.

The abelian group of all divisors is denoted $\mathrm{Div}(\mathbb{P}^1)$.

Example (Aut gp of a code)

divisor of f = $\mathrm{div}(f)$ = formal sum of zeros of f minus the poles.

$D = n_1 P_1 + \ldots + n_k P_k$ a divisor then $\mathrm{supp}(D) = \{P_1, \ldots, P_k\}$ is the **support** of D.

Example: f = polynomial of degree n in x $\Longrightarrow$ $\mathrm{div}(f) = P_1 + \ldots + P_n - n\infty$, $\mathrm{supp}(\mathrm{div}(f)) = \{P_1, \ldots, P_n, \infty\}$, where $zeros(f) = \{P_1, \ldots, P_n\}$.

The abelian group of all divisors is denoted $\mathrm{Div}(\mathbb{P}^1)$.

Example (Aut gp of a code)

divisor of f = $\mathrm{div}(f)$ = formal sum of zeros of f minus the poles.

$D = n_1P_1 + ... + n_kP_k$ a divisor then $\mathrm{supp}(D) = \{P_1, ..., P_k\}$ is the **support** of D.

Example: f = polynomial of degree n in $x \implies$ $\mathrm{div}(f) = P_1 + ... + P_n - n\infty$, $\mathrm{supp}(\mathrm{div}(f)) = \{P_1, ..., P_n, \infty\}$, where $zeros(f) = \{P_1, ..., P_n\}$.

The abelian group of all divisors is denoted $\mathrm{Div}(\mathbb{P}^1)$.

Example (Aut gp of a code)

divisor of f = $\mathrm{div}(f)$ = formal sum of zeros of f minus the poles.

$D = n_1 P_1 + ... + n_k P_k$ a divisor then $\mathrm{supp}(D) = \{P_1, ..., P_k\}$ is the **support** of D.

Example: f = polynomial of degree n in $x \implies$
$\mathrm{div}(f) = P_1 + ... + P_n - n\infty$, $\mathrm{supp}(\mathrm{div}(f)) = \{P_1, ..., P_n, \infty\}$,
where $zeros(f) = \{P_1, ..., P_n\}$.

The abelian group of all divisors is denoted $\mathrm{Div}(\mathbb{P}^1)$.

Example (Aut gp of a code)

$X = \mathbb{P}^1$
$F(X)$ = function field of $X \cong F(x)$, x a local coord.
D a divisor on X

Define: **Riemann-Roch space** $L(D)$:

$$L(D) = L_X(D) = \{f \in F(X)^\times \mid \mathrm{div}(f) + D \geq 0\} \cup \{0\},$$

"zeros allowed, poles required"

Example: polynomial of degree n in $x \in L(n\infty)$.

Example (Aut gp of a code)

$X = \mathbb{P}^1$
$F(X)$ = function field of $X \cong F(x)$, x a local coord.
D a divisor on X

Define: **Riemann-Roch space** $L(D)$:

$$L(D) = L_X(D) = \{f \in F(X)^\times \mid \mathrm{div}(f) + D \geq 0\} \cup \{0\},$$

"zeros allowed, poles required"

Example: polynomial of degree n in $x \in L(n\infty)$.

RiemannRochSpaceBasisP1

Example

```
gap> F:=GF(11);; R1:=PolynomialRing(F,["a"]);;
gap> var1:=IndeterminatesOfPolynomialRing(R1);;
gap> a:=var1[1];; b:=X(F,"b",var1);;
gap> var2:=Concatenation(var1,[b]);;
gap> R2:=PolynomialRing(F,var2);;
gap> crvP1:=AffineCurve(b,R2);
rec( ring := PolynomialRing(...,[a,b]),polynomial:=b)
gap> D:=DivisorOnAffineCurve([1,2,3,4],
     [Z(11)^2,Z(11)^3,Z(11)^7,Z(11)],crvP1);
rec( coeffs := [ 1, 2, 3, 4 ],
     support := [ Z(11)^2, Z(11)^3, Z(11)^7, Z(11) ],
     curve := rec( ring := PolynomialRing(..., [ a, b ]),
                         polynomial := b ) )
```

This sets up a divisor $D = 1 \cdot P_1 + 2 \cdot P_2 + 3 \cdot P_3 + 4 \cdot P_4$ on $\mathbb{P}^1$.

RiemannRochSpaceBasisP1

We compute a basis for $L(D)$ on $\mathbb{P}^1$ local coordinate a.

Example

```
gap> B:=RiemannRochSpaceBasisP1(D);
[ Z(11)^0, (Z(11)^0)/(a+Z(11)^7), (Z(11)^0)/(a+Z(11)^8),
  (Z(11)^0)/(a^2+Z(11)^9*a+Z(11)^6),
  (Z(11)^0)/(a+Z(11)^2),
  (Z(11)^0)/(a^2+Z(11)^3*a+Z(11)^4),
  (Z(11)^0)/(a^3+a^2+Z(11)^2*a+Z(11)^6),
  (Z(11)^0)/(a+Z(11)^6),
  (Z(11)^0)/(a^2+Z(11)^7*a+Z(11)^2),
  (Z(11)^0)/(a^3+Z(11)^4*a^2+a+Z(11)^8),
  (Z(11)^0)/(a^4+Z(11)^8*a^3+Z(11)*a^2+a+Z(11)^4) ]
```

DivisorAutomorphismGroupP1

Next, we compute a subgroup $Aut(D) \subset Aut(\mathbb{P}^1)$ preserving D.

Example

```
gap> agp:=DivisorAutomorphismGroupP1(D);; time;
7305
gap> IdGroup(agp);
[ 10, 2 ]
```

The automorphism group in this case is the dihedral group of order 10.

Example (Aut gp of a code)

X a curve, $D \in Div(X)$, $P_1, ..., P_n \in X(\mathbb{F})$ distinct points and $E = P_1 + ... + P_n \in \mathrm{Div}(X)$.

Assume $\mathrm{supp}(D) \cap \mathrm{supp}(E) = \emptyset$.

Choose an $\mathbb{F}$-rational basis for $L(D)$ and let $L(D)_{\mathbb{F}}$ denote the corresponding vector space over $\mathbb{F}$.

Example (Aut gp of a code)

X a curve, $D \in Div(X)$, $P_1, ..., P_n \in X(\mathbb{F})$ distinct points and $E = P_1 + ... + P_n \in \mathrm{Div}(X)$.

Assume $\mathrm{supp}(D) \cap \mathrm{supp}(E) = \emptyset$.

Choose an $\mathbb{F}$-rational basis for $L(D)$ and let $L(D)_{\mathbb{F}}$ denote the corresponding vector space over $\mathbb{F}$.

Example (Aut gp of a code)

Goppa's idea in the case of $X = \mathbb{P}^1$.

The **algebraic geometric code** (`AGCode`):

$C = C(D, E)$ = image of $L(D)_{\mathbb{F}}$ under the evaluation map

$$\mathrm{eval}_E : L(D) \to F^n, \quad f \longmapsto (f(P_1), ..., f(P_n)).$$

Example (Aut gp of a code)

Properties:

- *generator matrix for $C \iff$ basis of $L(D)$.*
- $\mathrm{length}(C) = \deg(E) = n$.
- eval_E $1-1 \implies C \cong L(D)$ as $\mathbb{F}[G]$-modules.
- $X = \mathbb{P}^1$ gives Reed-Solomon codes, which are MDS codes used in CDs.

Codes with “large” aut gps can be constructed this way.

J+Ksir+Traves paper (available on web) classifies concretely the aut. groups which can arise (in the $\mathbb{P}^1$ case).

GUAVA's non-linear codes

"Unrestricted" codes:

- `ElementsCode`, `RandomCode`
- `HadamardCode` (assumes GUAVA has associated Hadamard matrix in it database to construct `HadamardMat( ... )`)
- `ConferenceCode`
- `MOLSCode` (from mutually orthogonal Latin squares)
- `NordstromRobinsonCode` (discovered by a HS student)
- `GreedyCode`, `LexiCode`

General linear code constructions.

From the check/generator matrix or tables:

- `GeneratorMatCode`
- `CheckMatCodeMutable`, `CheckMatCode`
- `RandomLinearCode`
- `OptimalityCode`, `BestKnownLinearCode`

The last command uses tables developed by Cen Tjhal. Much larger “best known” codes tables are needed.

Common linear code constructions.

- `HammingCode`, `ReedMullerCode`,
- `SrivastavaCode`, `GeneralizedSrivastavaCode`
- `FerreroDesignCode` (uses `SONATA`)
- (classical) `GoppaCode`

Figure: Richard Hamming (1915-1998)

Special covering codes.

The covering radius of a linear code C is the smallest number r with the property that each element $\mathbf{v} \in \mathbb{F}^n$ there must be a codeword $\mathbf{c} \in C$ with $d(\mathbf{c}, \mathbf{c}) \leq r$.

- `GabidulinCode`
- `EnlargedGabidulinCode`
- `DavydovCode`
- `TombakCode`
- `EnlargedTombakCode`

Much larger covering codes tables are needed.

Golay codes.

- `BinaryGolayCode`
- `ExtendedBinaryGolayCode`
- `TernaryGolayCode`
- `ExtendedTernaryGolayCode`

Figure: Marcel Golay (1902-1989)

Cool example (on self-dual codes).

Group theory arises in the study of self-dual codes.

Consider the group G generated by

$$g_1 = \begin{pmatrix} 1/\sqrt{q} & 1/\sqrt{q} \\ (q-1)/\sqrt{q} & -1/\sqrt{q} \end{pmatrix}, g_2 = \begin{pmatrix} i & 0 \\ 0 & 1 \end{pmatrix}, g_3 = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix},$$

with $q = 2$. This group leaves invariant the weight enumerator of any self-dual doubly even binary code, e.g., `ExtendedBinaryGolayCode`.

Cool example (on self-dual codes).

GAP code (which calls Singular's `finvar.lib` package) for computing the invariants of G:

Example

```
gap> q := 2;; a := Sqrt(q);; b := 4;; z := E(b);;
gap> gen1 := [[1/a,1/a],[(q-1)/a, -1/a]];;
gap> gen2 := [[1,0],[0,z]];; gen3 := [[z,0],[0,1]];;
gap> G := Group([gen1,gen2,gen3]); Size(G);
Group(
[ [ [ 1/2*E(8)-1/2*E(8)^3, 1/2*E(8)-1/2*E(8)^3 ],
    [ 1/2*E(8)-1/2*E(8)^3, -1/2*E(8)+1/2*E(8)^3 ] ],
  [ [ 1, 0 ], [ 0, E(4) ] ], [ [ E(4), 0 ], [ 0, 1 ] ] ])
192
```

Cool example (on self-dual codes).

GAP code (cont'd):

Example

```
gap> R:=PolynomialRing(CyclotomicField(8),2);
PolynomialRing(..., [ x_1, x_2 ])
gap> LoadPackage("singular");
true
gap> GeneratorsOfInvariantRing(R,G);
[ x_1^8+14*x_1^4*x_2^4+x_2^8,
  1025*x_1^24+10626*x_1^20*x_2^4+735471*x_1^16*x_2^8+
  2704156*x_1^12*x_2^12 + 735471*x_1^8*x_2^16+
  10626*x_1^4*x_2^20+1025*x_2^24 ]
```

The GAP interface to Singular was written by Marco Costantini and Willem A. de Graaf.

Cool example (on self-dual codes).

The above result implies that any such weight enumerator must be a polynomial in

$$x^8 + 14x^4y^4 + y^8$$

and

$$1025x^{24} + 10626x^{20}y^4 + 735471x^{16}y^8 + 2704156x^{12}y^{12} + 735471x^8y^{16} + 10626x^4y^{20} + 1025y^{24}.$$

(Consistent with a well-known result in coding theory.)

Cyclic codes.

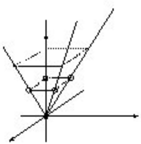
From the check/generator poly, etc:

- `GeneratorPolCode`, `CheckPolCode`
- `RootsCode`, `FireCode`
- `ReedSolomonCode`
- `BCHCode`, `AlternantCode`
- `QRCode`, `QQRCodeNC`
- `CyclicCodes`, `NrCyclicCodes`

Figure: Irving Reed, Gustave Solomon

Evaluation codes

- `EvaluationCode`
- `GeneralizedReedSolomonCode`
- `GeneralizedReedMullerCode`
- `ToricCode`
- `GoppaCodeClassical`
- `EvaluationBivariateCode`, `EvaluationBivariateCodeNC`
- `OnePointAGCode`



ToricCode example

This code was once best known:

Example

```
gap> C := ToricCode([ [0,0],[1,1],[1,2],[1,3],[1,4],\
 [2,1],[2,2],[2,3],[3,1],[3,2],[4,1]],GF(8));
a linear [49,11,1..39]25..38  toric code over GF(8)
```

min. dist. = 28. (Diego Ruano searched for other “new and good” toric codes but found none.)

Toric codes arise from “Riemann-Roch spaces” via the AG code construction above. Choosing the polytope containing the monomial’s exponents carefully, the code can be constructed to have a large automorphism group.

Decoding methods

`Decode(C,r)` uses syndrome decoding or nearest-neighbor except for:

- Hamming codes (the usual trick),
- GRS codes - see below,
- cyclic codes (error-trapping - sometimes), and
- BCH codes (Sugiyama decoding).

generalized Reed-Solomon codes

Decoding methods

The default algorithm used for generalized Reed-Solomon codes is the interpolation algorithm. Gao's decoding method for GRS codes is also available as an option.

generalized Reed-Solomon codes

Decoding codes obtained from evaluating polynomials at lots of points “should be easy”.

Rough idea: codewords are values of polynomial and $\#$ values is $>$ $\deg$(polynomials), so the vector overdetermines the polynomial. If the number of errors is “small” then the polynomial can still be reconstructed....

McGowan’s (undergrad) thesis has details fo the GUAVA implementation.

generalized Reed-Solomon codes

Syntax: `Decodeword( C, r )`, where C is a GRS code. This does “interpolation decoding”.

`GeneralizedReedSolomonDecoderGao` is a version which uses an algorithm of Gao.

`GeneralizedReedSolomonListDecoder( C, r, tau )` implements Sudan’s list-decoding algorithm for “low rate” GRS codes. It returns the list of all codewords in C which are a distance of at most τ from r.

Permutation decoding

Permutation decoding

This method also applies to the decoding of certain AG codes (see John Little's ("The Algebraic Structure of Some AG Goppa Codes", "Automorphisms and Encoding of AG and Order Domain Codes", for example).

Here is the basic idea.

C is a code, $v \in \mathbb{F}^n$ is a received vector, $G = Aut(C)$ is the perm. automorphism group.

Permutation decoding

The algorithm runs through the elements g of $G = Aut(C)$ checking if the weight of $H(g \cdot v)$ is less than $(d-1)/2$. If it is then the vector $g \cdot v$ is used to decode v: assuming C is in standard form then $c = g^{-1} \cdot Gm$ is the decoded word, where m is the information digits part of $g \cdot v$.

If no such g exists then “fail” is returned.

- This generalizes “error-trapping” for decoding cyclic codes,
- In some cases, only a *subset* of the elements g of G are required.

GUAVA functions: `PermutationDecodeNC( C, v, G )`, `PermutationDecode( C, v )`

Permutation decoding

The algorithm runs through the elements g of $G = Aut(C)$ checking if the weight of $H(g \cdot v)$ is less than $(d-1)/2$. If it is then the vector $g \cdot v$ is used to decode v: assuming C is in standard form then $c = g^{-1} \cdot Gm$ is the decoded word, where m is the information digits part of $g \cdot v$.

If no such g exists then "fail" is returned.

- This generalizes "error-trapping" for decoding cyclic codes,
- In some cases, only a *subset* of the elements g of G are required.

GUAVA functions: `PermutationDecodeNC( C, v, G )`, `PermutationDecode( C, v )`

SAGE and GUAVA

In SAGE , bad news :

- most GUAVA functions are not wrapped,
- most Leon functions are not wrapped.

Lots of work to be done.

SAGE and GUAVA

In SAGE , good news :

- GUAVA in included,
- there are some new coding-theoretic functions (related to computing "Duursma zeta functions" of codes).

Figure: Tom Hoeholdt talking to Iwan Duursma at the IMA coding theory conference, May 2007.

SAGE and GUAVA

C is an $[n, k, d]_q$ code
$C^\perp$ is an $[n, k^\perp, d^\perp]_q$ code
Motivated by local CFT, Iwan Duursma introduced the zeta function $Z = Z_C$ associated to C:

$$Z(T) = \frac{P(T)}{(1-T)(1-qT)}, \tag{2}$$

where $P(T)$ is a polynomial of degree $n + 2 - d - d^\perp$, called the zeta polynomial.

My "ACA talk" (pdf slides available online) surveyed some of its properties and gave examples using SAGE ....

GUAVA 2do list.

In GUAVA, my subjective list of priorities:

1. Leon's code needs to be rewritten and better utilized,
2. Database of codes (and Hadamard mat" and ...) should be
   - "certified" (and much larger ...),
   - in a more standard, transferable format (such as xml? ...),
   - "open" (as it is now) but "trademarked".
3. Constructions to be added ("Construction X/XX/Zinov'ev").
4. More and better (generalized) self-dual code algorithms.
5. More AG+LDPC codes and their decoding algorithms.
6. Codes over rings.

The end.

Have fun with GUAVA!