SIMU 2001 – Coding Theory Seminar
Project Topics
Draft – June 5, 2001

The following are ideas for the research group projects for the final three weeks of SIMU 2001. I expect we should have three or four groups of three or four students each, so not all of these projects will be "taken". However, there should be a sufficient range of possibilities here that all groups can find something appropriate for their interests, their backgrounds and strengths, their their programming skill and proficiency with Maple, etc.

Schedule

- Project topics handed out to students at start of second week of SIMU with instructions to start looking, see about getting a group together.
- By Wednesday of third week, inform Prof. Little about choice of topic, group membership.
- Full-time work on projects: Weeks 4,5.
- "Pre-presentations" (progress reports) to seminar: beginning of Week 5.
- Week 6: Write-up of results, preparation of final presentations.

A. "Toric Codes"

J.P. Hansen (see reference below) has recently proposed an interesting construction of codes using the geometry of polyhedra (or polytopes) in ordinary real vector spaces ($\mathbf{R}^2, \mathbf{R}^3$, etc.). Some of the same mathematics used here also features in the construction of geometric objects called *toric varieties* in algebraic geometry, hence the name "toric codes". Given a finite field $\mathbf{F}_q$ and a polytope $P \in \mathbf{R}^m$ (for technical reasons, we will assume P has boundary faces defined by linear equations with rational number coefficients), we get a code $C_P(\mathbf{F}_q)$ as follows:

- The entries of the codewords are in 1-1 correspondence with m-tuples of nonzero elements in $\mathbf{F}_q$. If α is a primitive element for $\mathbf{F}_q$, we can index the points in $(\mathbf{F}_q \setminus \{0\})^m$ as follows:
$$Q_{i_1,i_2,\ldots,i_m} = (\alpha^{i_1}, \alpha^{i_2}, \ldots, \alpha^{i_m})$$
where $0 \leq i_j \leq q-2$ for all $j = 1, \ldots, m$. (Recall that $\alpha^{q-1} = 1$.) Hence $n = (q-1)^m$.
- Given P, consider the collection of integer points contained in P, that is: $L = P \cap \mathbf{Z}^m$. For each point $\ell = (\ell_1, \ldots, \ell_m) \in L$, make a monomial in m variables
$$\ell = (\ell_1, \ldots, \ell_m) \leftrightarrow M_\ell = t_1^{\ell_1} \cdots t_m^{\ell_m}$$
.
- Evaluate each monomial M_ℓ at the points $Q_{i_1,i_2,\ldots,i_m}$ in $(\mathbf{F}_q \setminus \{0\})^m$ to make a word of length $n = (q-1)^m$.

- Then $C_P(\mathbf{F}_q)$ is the linear code over $\mathbf{F}_q$ generated by these vectors. (Note: we can think of each one as being obtained from a certain polynomial linear combination of the monomials M_ℓ.)

Note: This can be seen as a sort of generalization of one way we saw to construct the Reed-Solomon codes. There, $m = 1$, and the "polytope" P is a line segment along the x-axis, e.g. $P = [0, k-1]$. We take the integer points in $P - L = \{0, 1, \ldots, k-1\}$, make the monomials $1, t, \ldots, t^{k-1}$, and evaluate those at the points in $\mathbf{F}_q \setminus \{0\}$ to get a Reed-Solomon code word. (Many other sorts of generalizations of this idea have been studied in coding theory too!)

Hansen considers only a few special cases – very simple polytopes P with $m = 2$ – and computes the other parameters k, d for those toric codes. Moreover, he derives his results using some "fancy" tools from the algebraic geometry of toric varieties, *which you should not try to understand in detail*, since they are far removed from the subject matter we have learned in this seminar. So this topic would involve quite a bit of original research.

The first goals would be to rederive Hansen's results using more direct tools and extend the class of examples you have to work with by looking at a number of other P with $m = 2, 3$. Then, you would consider some more general questions (some, or all, as time permits):

1) How is the minimum distance of $C_P(\mathbf{F}_q)$ determined by the geometry of the polytope and the integer points $L = P \cap \mathbf{Z}^m$ in general? Where do the minimum weight vectors come from? How many of them are there? (You would probably want to start with the case $m = 2$ and look at your examples, try to see patterns, test conjectures, etc.)
2) Given q, m, n, k, is there a "best" choice for the polytope P to get a code with high d?
3) What other structures might these codes have? (For instance, which of them cyclic, etc.?)
4) Does taking subfield subcodes (à la BCH) yield interesting codes over $\mathbf{F}_p$?

Reference

1) Hansen, J. P. *Toric Surfaces and Error-Correcting Codes*, in: *Coding Theory, Cryptograph, and Related Areas (Proceedings of 1998 Guanajuato Conference)*, Buchmann, J., Hoholdt, T., Stichtenoth, H., Tapia-Recillas, H. eds., Berlin: Springer-Verlag, 2000.

B. List, or Bounded Distance, Decoding for Reed-Solomon Codes

The basic decoding method for Reed-Solomon codes we discussed in class (leading up to the Berlekamp-Massey algorithm) is designed to correct up to $e = \left[\frac{d-1}{2}\right] = \left[\frac{n-k}{2}\right]$ errors in a received word. By our results on the error-correcting capacity of a code in terms of its minimum distance, this restriction on the number of errors is necessary if we ask for a method that returns only one closest codeword for each received word.

However, it is easy to see that for Reed-Solomon codes in fact *some received words with more than e errors will still be correctable.* Reed-Solomon codes are not perfect codes so the Hamming balls of radius e do not fill out the whole space $\mathbf{F}_q^{q-1}$. Adding an error of weight $> e$ to a codeword may still not take us closer to any other codeword. So it is natural to ask whether there are other decoding methods that might take advantage of this "extra, unused" error-correction capacity. But, on the other hand, if we increase the allowed number of errors, then there are some received words that will end up in *more than one Hamming ball with that radius* centered at codewords.

Taking both of these observations into account leads to the *list decoding*, or *bounded distance decoding* problem. Namely, given a received word, and a distance bound e (possibly larger than $\left[\frac{d-1}{2}\right]$), we ask for algorithms that will produce the list of *all codewords at distance* $\leq e$ from the received word. (If there is only one item in the list, then the received word is uniquely decodable, and if $e > \left[\frac{d-1}{2}\right]$ we have taken advantage of the extra, unused error-correcting capacity mentioned above.)

There has been a surge of interest in such algorithms for Reed-Solomon and other codes in recent years, starting with work of Sudan in the late 1990's. (There were apparently similar ideas developed by others before Sudan, but he really "got the ball rolling," so to speak.) See references listed below for the sources you would consult for this project.) Consider the usual Reed-Solomon code

$$RS(q, k) = \{(f(1), f(\beta), \ldots, f(\beta^{q-2})) : f \in \mathbf{F}_q[x],\ \deg(f) < k\}.$$

Given a received word $v = (v_0, \ldots, v_{q-2})$, Sudan's method starts by constructing a 2-variable polynomial $Q(x, y)$ (of smallest possible degree) such that $Q(\beta^j, v_j) = 0$ for $j = 0, \ldots, q-2$. Then, under suitable hypotheses, the closest codewords to the received word come by evaluating $f(x)$ such that $y - f(x)$ *divides* $Q(x, y)$. Sudan's original work gives improvements over the usual approach only if the rate of the code is relatively small – $\frac{k}{n} \leq \frac{1}{3}$. This is unfortunate because the Reed-Solomon codes used in practice typically have much higher rates. However, the original approach has recently been improved by Guruswami and Sudan so that corresponding improvements are obtained for all rates. The original approach has also been reinterpreted by Roth and Ruckenstein so that the Sudan decoding becomes a process of computing syndromes and solving an *Extended Key Equation* similar to the Key Equation for Berlekamp-Massey (but different in several key ways). There are a number of possible project topics here (and these could be combined in different ways depending on whether groups wanted to work on more implementation-oriented or theory-oriented aspects).

1. *Implementing the Roth-Ruckenstein Algorithm* – A first possibility would be to study the sources mentioned above and implement the Roth-Ruckenstein algorithm, test it, compare with Berlekamp-Massey, etc. Their article includes reasonably complete descriptions of the steps involved in solving their Extended Key Equation, so this would be primarily a programming project (but a very good exercise!).

2. *Recovering the Error Values* – The original version of Sudan's method was not "optimized" for efficiency. For instance, the final step, once the $y - f(x)$ dividing $Q(x, y)$

were found, was to recover the corresponding codewords by "re-encoding" – that is, evaluating the $f(x)$ at the elements of $\mathbf{F}_q^*$ to get the corresponding Reed-Solomon codewords. This is clearly less efficient than what we were able to do in the case we studied in class. There the *Forney Formula* was available to compute the error values from the error locations, and we only had to subtract the error vector from the received word to recover the codeword. The relation of the sets of error values to the solution of the Extended Key Equation is less clear. To my knowledge, no one has discovered an analog of the Forney Formula yet. A very interesting theoretical problem is to try to find one. Understanding the case of lists of length $\ell = 1$ in Roth-Ruckenstein would be the first step (and it's not too hard). Exactly how does this relate to the usual Berlekamp-Massey algorithm? Then, what happens when $\ell > 1$?

3. *Incorporating the Guruswami-Sudan Improvements into the Roth-Ruckenstein setup* – This would also be a mainly theoretical topic, and almost certainly *quite a bit more difficult*. Roth and Ruckenstein derive their Extended Key Equation based on Sudan's original formulation of the decoding method. Can their approach be modified to take the Guruswami-Sudan improvements into account?

4. *Connections with Gröbner Basis Algorithms* – There is also a potential connection here with topics considered in Prof. Laubenbacher's Computational Algebra seminar. It might be possible to put together an *"interdisciplinary" project* that links the two seminars if you wanted to try to work with people from the other seminar. One way to understand both the Berlekamp-Massey algorithm we studied in class and the Roth-Ruckenstein algorithm is to view them as coming from a method of computing a Gröbner basis for a particular *module* over a polynomial ring (a solution module for the key equation). For this project topic, you would work through the details first for the Berlekamp-Massey algorithm (work by Fitzpatrick), then try to extend this to the Roth-Ruckenstein algorithm. This would be a new result, as far as I am aware.

References

1. Sudan, M. *Decoding Reed Solomon Codes Beyond the Error-Correction Diameter*, *Proceedings of the 35th Annual Allerton Conference on Communication, Control, and Computing*, University of Illinois, 1997.
2. Roth, R.M. and Ruckenstein, G. *Efficient Decoding of Reed-Solomon Codes Beyond Half the Minimum Distance*, *IEEE Transactions on Information Theory*, **46, no. 1**, (2000), 246-257.
3. Guruswami, V. and Sudan, M. *Improved Decoding of Reed-Solomon and Algebraic-Geometry Codes*, *IEEE Transactions on Information Theory*, **45, no. 6**, (1999), 1757-1767.
4. Fitzpatrick, P. *On the Key Equation*, *IEEE Transactions on Information Theory*, **41, no. 5**, (1995), 1290-1302.
5. Fitzpatrick, P. *Solving a Multivariable Congruence by Change of Term Order*, *Journal of Symbolic Computation*, **24**, (1997), 575-589.

6. Cox, D., Little, J. and O'Shea, D. *Using Algebraic Geometry*, Graduate Texts in Mathematics 185, Springer-Verlag 1998. Chapter 5 discusses the necessary background on modules over polynomial rings and Gröbner bases, and Chapter 8 includes a presentation of Fitzpatrick's work on applications to decoding algorithms.

C. Low-Density Parity Check Codes

This is another area where there are a number of possible project topics (and they could be combined in different ways depending on whether groups wanted to work on more implementation-oriented or more theory-oriented aspects).

In the past two years or so, one of the "hot topics" in coding theory has been the subject of *low-density parity check codes*, or LDPC codes for short. The idea is not new – it goes back, in fact to work of Gallagher published in 1961. However, the new element is the realization (through experimental simulations) that long binary LDPC codes yield communications systems that actually perform very close to the theoretical Shannon capacity on Gaussian channels. Hence, in some sense, these codes are almost as good as theoretically possible, so they are very attractive potential candidates for use in communications systems or data storage systems where very high degrees of accuracy are necessary.

The construction of a binary LDPC code starts from the parity check matrix. This should be a "sparse" matrix of zeroes and ones. The ratio of the total number of ones to the number of entries is called the *density*. The good codes here often have small densities – roughly .05 (i.e. 5 out of every 100 entries are 1's) or less. In the past, to obtain examples of codes of this type, coding theorists often simply chose *random* parity check matrices with the desired density of 1's. The problem with this is that *not every random matrix gives a good code* – for instance the minimum distance might turn out to be smaller than expected, or the code might have other undesirable properties (e.g. cycles of small order in its *Tanner graph*).

So a portion of the recent work on LDPC codes has dealt with other, more systematic, ways to construct these codes by using ideas from algebra or geometry. In particular, one method that has been suggested is to use codes defined by the incidence matrices of points and lines in a *finite geometry*. In a finite geometry, there are two finite sets, a set P of *points* with $|P| = p$ and a set L of *lines*, with $|L| = \ell$. Each line contains a fixed number ρ of points, and each point is contained in a fixed number λ of lines. The paper by Kou, Lin, and Fossorier below contains quite a few ideas on two classes of codes of this type, where the geometry is either the affine or projective geometry over a field. We get the parity check matrix H for a code, for instance, by making the $\ell \times p$ matrix $H = (h_{ij})$ with

$$h_{ij} = \begin{cases} 1 & \text{if line } i \text{ contains point } j \text{ and} \\ 0 & \text{if line } i \text{ does not contain point } j \end{cases}$$

Note that each row of H contains ρ 1's and each column of H contains λ 1's.

1. *LDPC Codes from Hermitian Curves* – Another construction of LDPC codes that does not seem to have received much attention as of yet is to use another sort of finite geometry. Namely, consider a field of square order $\mathbf{F}_{m^2}$ (e.g. $\mathbf{F}_4, \mathbf{F}_{16}, \mathbf{F}_{64}, \ldots$ in

characteristic 2). The *(affine) Hermitian curve* over $\mathbf{F}_{m^2}$ is the set of pairs (x, y) that solve the equation

$$x^{m+1} = y^m + y.$$

The points of the geometry in question are the solutions of this equation with $(x, y) \in \mathbf{F}_{m^2}^2$. The lines are the sets of points on some line $ax + by + c = 0$ in $\mathbf{F}_{m^2}^2$ and on the Hermitian curve. (There is also a projective Hermitian curve defined by the homogeneous equation

$$x^{m+1} = y^m z + yz^m.)$$

and a corresponding projective version.)
These finite geometries have some very interesting combinatorial properties, and one interesting project topic will be to investigate them and the properties of the LDPC codes obtained from them. For instance, how many points and lines are there? How many points are on each line? How many lines contain at each point. Can we perhaps remove some lines and or points to get ρ, λ to be the same for all points and lines? What are the parameters of the corresponding codes? What extra structure might they have? (Note: There is a large group of automorphisms of the Hermitian curve that will induce automorphisms of these codes.) What is the shortest nontrivial cycle in the *Tanner graph* for one of these codes? It will be good to start with the cases $m = 4, 16$, and perhaps $m = 64$, work out everything in detail by hand/with Maple to get some intuition for how this all works. Then try to form some conjectures about the general situation and prove them. (N.B. The answers to many of these questions are known in the sense that they have been worked out by others previously, but the application to LDPC codes is probably new.)

In order to use good LDPC codes, efficient encoding and decoding algorithms are necessary. This is not an easy requirement, though, because the codes of interest here have quite large n, k. For instance, there are several fast iterative decoding algorithms (the "belief propagation" method is one) that appear to work extremely well in practice but that are not completely understood from the theoretical point of view.

2. *Iterative Decoding Methods* – Another good project (more computationally-oriented) would be to study and implement one or more of the iterative decoding algorithms discussed in the Kou, Lin, Fossorier paper below and test them on random LDPC matrix codes, or on the Hermitian geometry codes as in topic 1 above.

3. A number of other possible project topics exist here based on other methods of constructing LDPC codes by applying ideas on the *Cayley graphs* of the matrix groups $SL_2(\mathbf{F}_q)$, or Ramanujan graphs, etc.) See Prof. Little if this sounds interesting.

References

1) Kou, Y., Lin, S., and Fossorier, M. *Low Density Parity Check Codes Based on Finite Geometries: A Rediscovery and More*, preprint, 2000.

2) Assmus, E. and Key, J. *Designs and Their Codes*, Cambridge Tracts in Mathematics 103, Cambridge U. Press, 1992.

D. Algebraic Geometry Codes and "Elkies Codes"

One of the big developments in coding theory from the late 1980's and 1990's was the construction by Tsfasman, Vladut, and Zink of long codes over large finite fields that exceed the Gilbert-Varshamov bound. The construction involved ideas about *algebraic curves* over finite fields and their algebraic geometry. The original idea came from work by V.D. Goppa, so codes of this class are often called AG Goppa codes, or AG codes for short.

One way to construct these codes can be seen as a generalization of the construction of Reed-Solomon codes by evaluation of polynomials. To illustrate this, let X be a curve over a finite field $\mathbf{F}$, the subset of $\mathbf{F}^2$ defined by a polynomial equation $G(x, y) = 0$. For example, the Hermitian curve over $\mathbf{F}_{m^2}$ mentioned in topic C above has the defining equation

$$x^{m+1} - y^m - y = 0.$$

The Hermitian curve has one point Q "at infinity", and the coordinate functions x, y have pole singularities at Q of orders $m, m + 1$ respectively. (This means, intuitively, that in a local coordinate t at Q, $x = \frac{1}{t^m}$ + higher degree terms and $y = \frac{1}{t^{m+1}}$ + higher degree terms. Given an integer $a \geq 0$, let $L(aQ)$ denote the vector space of polynomials spanned by monomials $x^r y^s$ with $rm + s(m + 1) \leq a$. (In more intrinsic terms, $L(aQ)$ can also be defined as the space of all functions on X with poles of order $\leq a$ at Q and no other poles, and then the description of such functions as polynomials in x, y is something that must be proved.) In any case, we get linear codes over $\mathbf{F}_{m^2}$ as follows. Let $P_1, P_2, \ldots, P_N$ be a list of the $\mathbf{F}_{m^2}$-rational points on the curve X apart from Q (or some subset of them), and consider the mapping

$$\begin{aligned} ev : L(aQ) &\to \mathbf{F}_{m^2}^N \\ f &\mapsto (f(P_1), \ldots, f(P_N)) \end{aligned}$$

The image is one of the AG Goppa codes. (This type of code is often called a "one-point" code because the construction of the vector space $L(aQ)$ involves only the one point Q. We can evaluate $f \in L(aQ)$ at any other $\mathbf{F}_{m^2}$-rational point on X and get something well-defined because f has no other poles besides Q. There is also a more general "recipe" for constructing AG Goppa codes that uses vector spaces of functions with poles at several different points.) The (extended) Reed-Solomon codes are obtained in the case $X = \mathbf{P}^1$ (the projective line, $Q = \infty$, and $a = k + 1$.

The very good codes that Tsfasman, Vladut, and Zink constructed came from particular curves called *modular curves.* Their definition involves some "heavy-duty" number theory and algebraic geometry, so we will not try to reproduce it here. And unfortunately, the modular curves do not have simple equations as plane curves. (The equations are *much* more complicated than the equation of the Hermitian curve given above.) So it is not easy to give explicit constructions of some of the most interesting AG Goppa codes. Moreover, AG codes have not yet found many applications, partly because it is only relatively recently that decoding algorithms with efficiency approaching that of the Berlekamp-Massey algorithm have been developed for the codes that can be explicitly constructed.

Noam Elkies has recently proposed a construction of non-linear codes from algebraic curves over finite fields that is closely related to Goppa's construction, but that sometimes yields superior results. The basic idea is to use the same sort of evaluation mapping above, but to evaluate a collection of rational functions defined by degree bounds on the numerator and denominator rather than conditions on the pole set.

There are several possible project topics here, but if you want to work on these, be aware that the mathematical prerequisites will be somewhat higher than in the other topics above. You'll probably need to do a fair amount of preliminary study on the language of divisors, "line bundles" etc. to get started on these. If you're interested in seeing an unexpected application of a "mainsteam" of classical pure mathematics (algebraic geometry and number theory), though, this might be a good choice of topic for you.

1. *Elkies Codes from Hermitian Curves* – A good project topic would be to look at his paper (see below), try to understand the construction, work out the details for the "genus zero" case (i.e. for X the affine or projective line – as in construction of Reed-Solomon codes). Then see what this construction does in the case of Hermitian codes. What are the parameters for the codes you get? Also, good research questions here are:
2. *The Recognition Problem* – Is there a good (i.e. efficiently computable) way to determine if a given word is even in one of these code? Note: the codes are not linear so we do not have a parity check matrix to use for this!
3. *The Encoding Problem* – How could a good (i.e. efficiently computable) encoding method be developed for codes of this type? Since they are not linear in general, there will not be a generator matrix to use. How could an easily computable 1-1 correspondence between information strings and codewords be set up?
4. *Automorphisms* – The Hermitian curve has a large group of automorphisms that permute the $\mathbf{F}_{m^2}$-rational points, and hence permute the coordinates of codewords of these codes. What, if anything, does the presence of these automorphisms do for you (even though the code itself is not linear)?

References

1. Elkies, N. *Excellent nonlinear codes from modular curves*, preprint 2001.
2. Pretzel, O. *Codes and Algebraic Curves*, Oxford University Press, 1998.
3. Walker, J. *Codes and Curves*, Student Mathematical Library, 7. IAS/Park City Mathematical Subseries. American Mathematical Society, Providence, RI; Institute for Advanced Study (IAS), Princeton, NJ, 2000. A good place to start to see what is involved with the Goppa codes, but this book doesn't go too far.