Expander Codes

The Sipser–Spielman Construction

Priti Shankar

Expander graphs are graphs in which every set of vertices has an unusually large number of neighbours. It is a remarkable fact that graphs of this kind exist. Even more remarkable is the spectrum of applications of these graphs, ranging from providing new insights in the field of computational complexity theory to the solution of problems in communication. In this article we show how expander graphs can be used for designing efficient error correcting codes which have fast decoding algorithms.

Priti Shankar is with the Department of Computer Science and Automation at the Indian Institute of Science, Bangalore. Her interests are in theoretical computer science and error correcting codes.

Introduction

The fifty year old history of error correcting codes can be characterized by a sequence of incremental results punctuated by occasional leaps forward. The most recent dramatic development, is the recognition of the fact that one can derive excellent codes from graphs which have a high degree of local connectivity but simple structural descriptions that facilitate iterative decoding. One such class consists of *expander graphs* which were introduced in the 1970's and have turned out to be very versatile tools in both theory and practice. In the mid 1990's a clever construction for efficiently decodable codes was discovered by Sipser and Spielman, and this approach has been improved and refined over the last few years. In the first part of this article we attempt to illustrate the power of this technique in constructing efficiently decodable codes.

Keywords
Expander graphs, error correction, efficiently decodable codes.

Codes and Channels

A noisy communication channel is illustrated in *Figure* 1. The aim is to reliably transmit information through

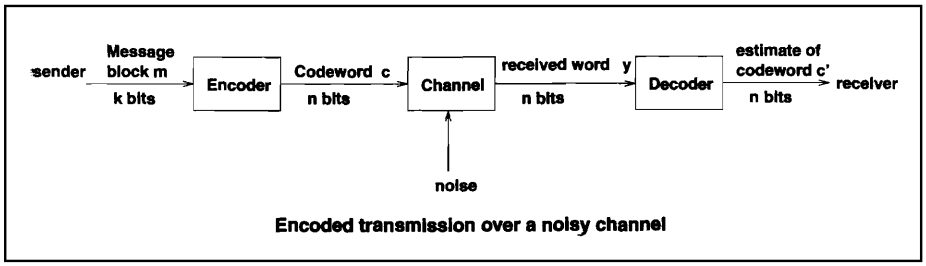

Figure 1. A noisy communication channel.

this channel. Information is transmitted as a sequence of symbols. For example if we wanted to transmit messages made up of English letters and assuming the channel can transmit only binary digits, we might encode each of the twenty six letter using five bits, and use the remaining six combinations of the 32 possible combinations of five bits for other symbols like blanks, punctuation marks and so forth. The message will now look like a sequence of 0's and 1's. Suppose we want to transmit this message over the unreliable communication channel so that even if the channel corrupts some of the bits we are able to recover the original message with high probability. One solution is to transmit *blocks* of symbols rather than individual symbols after the addition of *redundant* symbols for protection against errors. For example we might want to break up a message into blocks each 5 bits long. Once we have done this, we add redundant symbols to the block for error protection. A simple scheme would be one that repeats each block three times. Thus instead of transmitting a block of 5 bits we transmit a larger block of 15 bits. It is easy to see here that a single error in any bit of the larger block can be corrected, as there are three copies of each bit and a majority vote can be taken. However we have added 10 redundant symbols in the process, and the fraction of bits carrying real information is only 1/3. Now suppose we replicated each block five times before transmission, we would be reducing the rate to 1/5 but can now correct two errors in each block. So if we are willing to accept a drop in rate, we can correct as many errors as we wish.

Suppose we want to transmit a message over the unreliable communication channel so that even if the channel corrupts some of the bits we are able to recover the original message with high probability. One solution is to transmit *blocks* of symbols rather than individual symbols after the addition of *redundant* symbols for protection against errors.

Given a code and the decoding rule, it is possible to compute the probability of decoding error if a probabilistic model of the channel is assumed. One such model is the binary symmetric channel (BSC), shown in *Figure* 2. On the left are the channel input symbols 0 and 1 and on the right, the channel output symbols, and these are connected by edges. The labels on the edges represent *transition probabilities*. Thus if a 0 or a 1 was sent, the probability of receiving a 1 or a 0 respectively, is p. This is the probability that a bit is flipped in transmission through the channel, and is called the *raw bit error probability*. The channel is said to be *symmetric* as the probability of its flipping a 0 or a 1 are the same. In general the input alphabet set A and the output alphabet set B may be diferent and have different sizes.

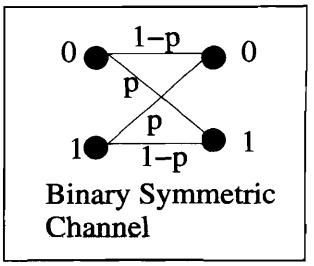

Figure 2. Channel model.

Suppose now, that we wish to compute the bit error probability for the simple repetition scheme described above, where each bit is replicated three times. A mesage bit is decoded erroneously if two or three of its copies are received in error. If P_e denotes the *bit error probability* then $P_e = P\{2 \text{ channel errors}\} + P\{3 \text{ channel errors}\} = 3p^2(1-p) + p^3$. That is,

$$P_e = 3p^2 - 2p^3. \qquad (1)$$

Since $p < 1/2$ this is less than the raw bit error probability p, and if p is very small, the improvement in the reliability of the channel is quite significant. It is easy to see that even higher reliability is obtained by repeating each bit more times. For this repetition code, the error probability can be made as small as we wish but with a corresponding decrease in rate. This is not particularly interesting and we can do much better.

In practical systems as illustrated in *Figure* 1, a message of length k is *encoded* into a *codeword* of length n, $n > k$. This is transmitted over the noisy channel which may corrupt some of the bits. The corrupted version y of the codeword is received at the other end and is

Given a code and the decoding rule, it is possible to compute the probability of decoding error if a probabilistic model of the channel is assumed.

The corrupted version y of the codeword is received at the other end and is *decoded* to form an estimate c' of the transmitted codeword from which the original message can be recovered. A *decoding error* is said to occur if the estimated codeword differs from the transmitted one.

decoded to form an estimate c' of the transmitted codeword from which the original message can be recovered. A *decoding error* is said to occur if the estimated codeword differs from the transmitted one. The block length of the code is n and the *rate* $R = k/n$ is the fraction of real information symbols in a codeword.

Let us look at another simple code, a Hamming code of block length 7 and rate 4/7. For each message block of length 4 there are three extra bits added as checks. If x_0, x_1, x_2, x_3 are the message bits, let x_4, x_5, x_6 be the check or *parity* bits. These are determined by the equations

$$x_4 \equiv x_1 + x_2 + x_3 (\mathrm{mod}\ 2),$$

$$x_5 \equiv x_0 + x_2 + x_3 (\mathrm{mod}\ 2),$$

$$x_6 \equiv x_0 + x_1 + x_3 (\mathrm{mod}\ 2).$$

Thus for example if $(x_0, x_1, x_2, x_3) = (0111)$ then $(x_4, x_5, x_6) = (100)$ and the codeword sent over the channel is (0111100). Let us rewrite the parity check equations above in the following way:

$$\begin{array}{cccccccc} & x_1 & +x_2 & +x_3 & +x_4 & & & = 0, \\ x_0 & & +x_2 & +x_3 & & +x_5 & & = 0, \\ x_0 & +x_1 & & +x_3 & & & +x_6 & = 0. \end{array}$$

Define the binary matrix H as follows:

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

For each of the 16 codewords c, we have

$$H^T c = \mathbf{0}, \qquad (2)$$

where the right hand side is the all zero vector with three components. This just says that for each codeword, all the linear constraints are satisfied. An interesting observation here is that equation(2) is a linear dependence relation among the columns of H which picks up column i if the ith bit in the codeword is a 1. Thus for every linear dependence among w columns of H there is a corresponding codeword with w non-zero components, and every codeword with w non-zero components corresponds to a linear dependence among w columns of H Each codeword is said to be *orthogonal* to every vector that can be generated by linear combinations of rows of H

A binary linear block code of length n, is a set of 2^k vectors of length n all of which satisfy a set of $n-k$ linearly independent constraints.

A binary linear block code of length n, is a set of 2^k vectors of length n all of which satisfy a set of $n-k$ linearly independent constraints. The *Hamming distance* between two vectors is the number of positions in which they differ. The *minimum distance* of the code is the minimum of the Hamming distances between all pairs of codewords. A binary linear block code of length n, whose message length is k and whose minimum distance is d is termed an (n,k,d) code. It should be clear that the all zero vector is always a member of any linear block code.

Let us compute the minimum distance of the Hamming code described above. Note that the seven columns of H consist of all distinct nonzero combinations of three bits. Since no two columns are identical, the minimum distance of the code is at least two. Further we have several linear combinations of three columns that give the all 0 vector, therefore the minimum distance of the code is 3.

A common rule for decoding is the *maximum-likelihood* decoding rule which, given a received vector y decodes into the codeword c' such that the probability of y given c' is the largest. For a binary symmetric channel model

It turns out to be convenient to imagine that the BSC adds (mod 2) either a 0 or a 1 to each transmitted bit. If it adds a 0, the bit is received correctly; if it adds a 1 the bit is in error.

with $p < 1/2$ a maximum likelihood decoding rule would decode a received vector into a codeword at smallest Hamming distance from the received vector.

We now describe a maximum-likelihood decoding algorithm for the Hamming code above. It turns out to be convenient to imagine that the BSC adds (mod 2) either a 0 or a 1 to each transmitted bit. If it adds a 0, the bit is received correctly; if it adds a 1 the bit is in error. The sequence of 0's and 1's is the *error pattern* denoted by z. A single error pattern has a single 1 and all other bits 0.

Now the receiver who knows y and wants to find an estimate of c, computes the vector $s = (s_0, s_1, s_2)$ such that $s^T = Hy^T = H(c+z)^T = Hc^T + Hz^T = Hz^T$ Here s is called the *syndrome* of the received vector. The syndrome of a codeword is always the zero vector. Else if the i^{th} constraint is not satisfied by the received vector, then the i^{th} bit of the syndrome is a 1. The syndrome thus depends only on the error pattern z. However if z is known then c can be computed as $c = y + z$. The syndrome for each single error pattern corresponds to a column of H Since the columns exhaust all possible syndromes, and since for $p < 1/2$ a single error pattern is always more likely than a multiple error pattern, maximum likelihood decoding for the Hamming code consists of the following steps:

1. Compute the syndrome $s = Hy^T$ from the received vector y.
2. If s is 0, output the received vector as the codeword estimate and exit, else identify the unique column i of H equal to the syndrome.
3. Form the error pattern z which has a 1 in the i^{th} position and zeroes elsewhere.
4. Compute $c' = y + z$.

So all single errors can be corrected. What if two errors occur? Two errors in fact will always cause the received vector to be closer to a codeword *other* than the one transmitted. For example if the codeword (0111000) is transmitted and there are two errors in the last two bits, (0111011) is received. The decoder will decode to (0110011) which is at a Hamming distance of 1 from the received vector. If we draw n dimensional spheres of Hamming radius 1 around each codeword, each sphere containing vectors at Hamming distance 1 fron the center, it turns out that the spheres are non intersecting (that is why we can correct all single errors), and between them they cover the whole space (which is why a double error in one codeword will give a vector in the Hamming sphere around another). It is easy to see that if $d = 2t+1$ for some t, then one can draw n-dimensional spheres of radius t around each codeword and not have any of them intersect. With the assumption that a small number of errors is more likely than a large number, any received vector that falls within a sphere should be decoded to the codeword at the center of the sphere. While this is a simple rule, its implementation is difficult for the general class of linear block codes. Note that the number of possible syndromes for an (n, k, d) binary code is 2^{n-k}, a quantity that is exponential in the code parameters. So a scheme that uses table look-up where each syndrome indexes into the most likely error pattern, is quite impractical for large values of $n - k$.

It is easy to see that if $d = 2t + 1$ for some t, then one can draw n-dimensional spheres of radius t around each codeword and not have any of them intersect.

Before we proceed further, let us compute the error probability of the Hamming code. We can compute the *block error probability* i.e. the probability that the transmitted vector is not equal to the decoded vector. But this does not tell the whole story as even if the two are different, some of the bits may be correct. If we denote the bit error probability $P\{c_i \neq c_i'\}$ by P_e^i it is possible to show by a combinatorial argument which we omit here that

$$P_e^i = 9p^2 + 26p^3 + \ldots \qquad (3)$$

In his classic paper of 1948, Shannon considered a large class of probabilistic channels and showed the rather surprising result that there was a quantity called the *channel capacity* such that one could achieve an *arbitrarily small* error probability at any rate less than the channel capacity.

where the higher order terms are negligible for small p. When we compare this with the crude repetition scheme we find that for small values of p the Hamming code at rate $4/7$ performs as well as the crude repetition scheme of rate $1/3$. So this is some improvement. However, the curious reader may wonder what is the *best* that one can do, given the channel model. The answer was provided by Claude Shannon in 1948.

In his classic paper of 1948, Shannon considered a large class of probabilistic channels and showed the rather surprising result that there was a quantity called the *channel capacity* such that one could achieve an *arbitrarily small* error probability at any rate less than the channel capacity. For example for the binary symmetric channel if $p = 0.1$ then the channel capacity is 0.531, and according to Shannon's theorem there should exist a code with rate $R \geq 0.5$ and overall error probability less than 10^{-1000}! This was a counter-intuitive result and it took communication engineers some time to understand its importance.

The outstanding problem for many years was the explicit construction of such codes, as Shannon only showed their *existence*. In the last few years explicit constructions have appeared. One of these arose from the PhD thesis of Robert Gallager in 1961 which was finally recognized as a deeply prophetic work. Gallager had developed techniques for recursively developing long codes calld *low density* parity check codes (LDPC), and had proposed several supoptimal techniques for decoding these codes. His technique is best described by means of a bipartite graph.

A bipartite graph is a graph with two set of nodes. Let us call these the *left* nodes and the *right* nodes. Edges exist only between left nodes and right nodes. Every linear block code can be described by a bipartite graph as follows.

Assume that the message symbols are binary digits. The n nodes on the left called *codeword* nodes and r nodes on the right called *constraint* nodes. In Gallager's original version of an LDPC the nodes on the left are in one-to-one correspondence with the positions $1, 2, \ldots n$ of a codeword. A binary n-bit vector $c = (c_1, c_2, \quad c_n)$ is a codeword if and only if for each constraint node the modulo 2 sum of the values of its adjacent message nodes is 0. Each constraint node imposes a *linear* constraint on the bits of the codeword. The rate of the code is at least $(n - r)/n$ with equality if all the constraints are linearly independent. Each constraint can be thought of as a linear equation in the variables corresponding to each codeword position. The code is called "low density" because the equivalent parity check matrix, is chosen to be *sparse*, containing ay most a constant number of ones in each row and column. We illustrate with a bipartite graph for the Hamming code just described. Though the code is not a low density code, it gives an idea of how graphs for Gallager codes are constructed. *Figure* 3 displays the bipartite graph for the Hamming code.

Each constraint node imposes a *linear* constraint on the bits of the codeword. The rate of the code is at least $(n-r)/n$ with equality if all the constraints are linearly independent.

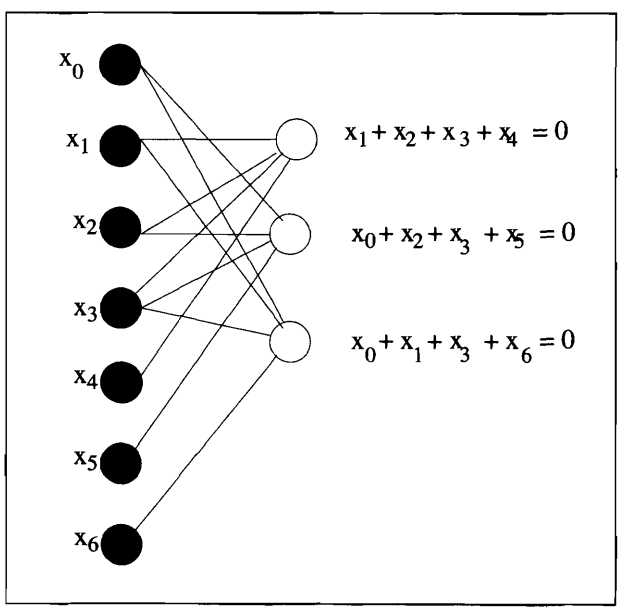

Figure 3. Bipartite graph for the Hamming code.

Sipser and Spielman showed in 1996 that it was possible to construct a family of asymptotically good linear error-correcting codes which could be decoded in linear time.

For an (n, k, d) code the following two problems are of interest

- The unique bounded distance decoding problem: Given a received word $y \in B^n$ (where B is the output alphabet set), find the unique codeword c if any, that lies within a distance $(d-1)/2$ from y. We saw a geometric interpretation of this problem earlier.

- The list decoding problem: Given a received word y and an error bound t, find all codewords of C within distance t of the received vector. Here we draw an n dimensional sphere of radius t around the *received vector* and output the *set of codewords* inside the sphere.

Both these problems have elegant algebraic solutions for some classes of codes eg. Reed-Solomon codes, BCH-codes, and algebraic-geometry codes, though these algorithms do not run in linear time. By a linear time algorithm is meant one whose running time is linear in n,the length of the code.

The question was: Can we design codes with decoding algorithms having linear complexity and good performance in terms of their error probability?

Sipser and Spielman showed in 1996 that it was possible to construct a family of asymptotically good linear error-correcting codes which could be decoded in linear time. Their construction used *expander graphs* and their codes are a special subclass of low density parity check codes. We now describe these graphs.

Expander Codes

Expanders are sparse graphs with very good connectivity properties. Suppose we have a bipartite graph which is d-regular, meaning thereby that every vertex has degree d. One could have regular graphs with degree d on the left and c on the right. Such graphs are both *left regular* and *right regular*. Now given any subset of

vertices of size N on the left, if we count its neighbours on the right, the maximum value this can have is Nd. We say that a graph is an expander if given a not too large subset on the left the number of neighbours on the right is at least γ times the size of the subset. When γ is very close to d we say the expander is *lossless*. More formally:

We say that a graph is an expander if given a not too large subset on the left, the number of neighbours on the right is at least γ times the size of the subset.

Definition: A bipartite graph $G = (X, Y, E)$ is said to be an $(n, m, d, \alpha, \gamma)$-expander if $|X| = n, |Y| = m$, the degree of each node in X is d and for every $A \subseteq X, |A| \leq \alpha n$ the set of neighbours $N(A)$ of A in Y satisfies $|N(A)| > \gamma|A|$.

An expander is illustrated in *Figure* 4.

We now show how expanders can lead to linear time decodable codes.

Given a sparse bipartite graph (i.e. a graph with a small number of edges), representing a linear block code with vertices on the left representing positions of the codeword, vertices on the right representing parity checks or constraints, with an edge between a vertex a on the left and vertex b on the right if position a participates in

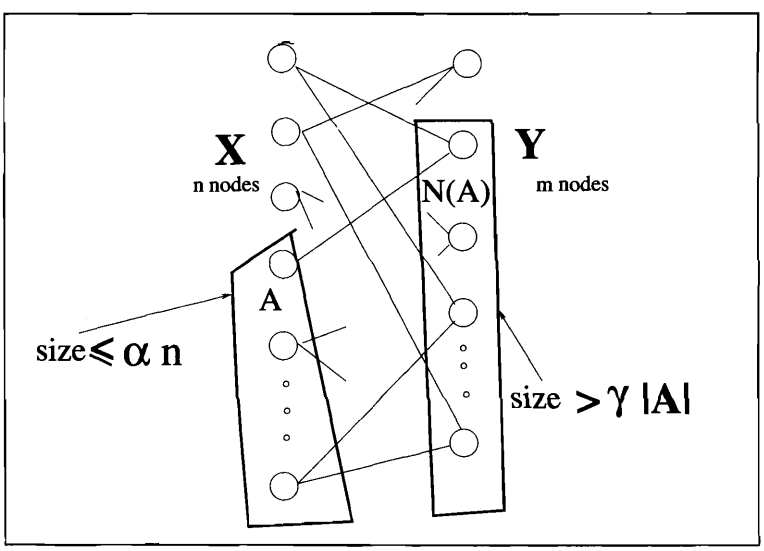

Figure 4. Expander graphs.

M Sipser

D Spielman

parity check b, let us assume that each position participates in r parity checks i.e. degree of each vertex on left is r, where r is a constant. The following result is useful.

Lemma 1. *Let $G = (X, Y, E)$ be a bipartite $(n, m, r, \alpha, r/2)$ expander. Then every set S in X of size at most αn satisfies the Unique Neighbour Property, i.e. there exists $y \in Y$ such that $|N(y) \cap S| = 1$.*

Proof. Consider a set S of size at most αn. Suppose the unique neighbour property does not hold for S then every $y \in N(S)$ has at least two neighbours in S. Therefore the number of edges leaving S is at least $2|N(S)| > 2(r/2)|S| = r|S|$, which contradicts the assumption of the left regularity of G.

The next theorem due to Sipser and Spielman establishes a lower bound on the minimum distance of the code, and is interesting as it is based entirely on the expansion property of the graph.

Theorem 1. *Let $G = (X, Y, E)$ be a bipartite $(n, m, r, \alpha, r/2$ expander graph defining a code $C(G)$. Then the minimum distance of $C(G)$, $dist((C(G)) > \alpha n$.*

Proof. Assume that $dist(C(G)) \leq \alpha n$. Then there is a nonzero codeword whose weight is at most αn. Let w be such a codeword and let Z be its support, i.e. the set of coordinates that are not zero. Let y be a vertex in $N(Z)$ satisfying the unique neighbour property. The constraint imposed by constraint node y is that the sum of all the variables that it checks is 0 mod 2, but in the assignment defined by w only one of those variables is assigned the value 1. Therefore the constraint cannot be satisfied and w cannot be a codeword, giving a contradiction.

Figure 5 illustrates the above argument. Note that in the figure all the non-zero components of the codeword have been gathered into the set Z so that all the components

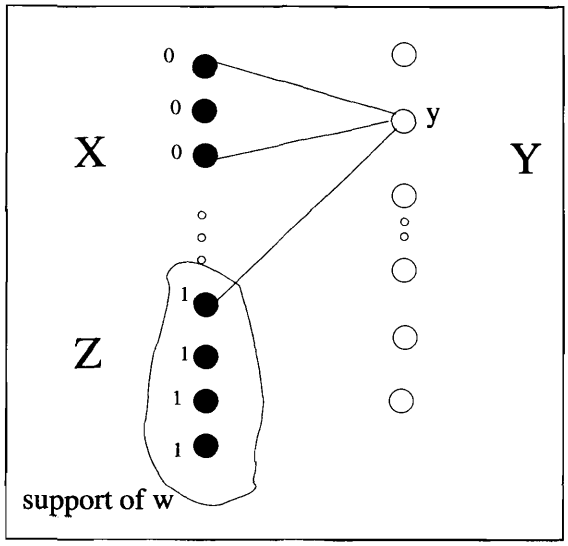

Figure 5. Illustrating the lower bound on distance.

outside Z are 0. Since there is no particular ordering of the vertices on the left, this is always possible.

Having established the lower bound on the minimum distance, let us look at the incredibly simple decoding algorithm proposed by Sipser and Spielman. We say that a constraint is *satisfied* by a setting of a set of variables if the sum of those variables is 0. Else it is *unsatisfied.* Suppose we receive some word and consider a variable(i.e. node on left hand side of G) and its constraints. If the variable has more unsatisfied than satisfied constraints and we flip the variable, then the situation is reversed, i.e. it has more satisfied than unsatisfied constraints. The goal of the algorithm is to keep flipping variables, ensuring that the total number of unsatisfied constraints keeps decreasing until no unsatisfied constraints are left, i.e. we are left with a codeword. We will show that if the graph is a good expander and the number of errors is not too large, this can be achieved. The simple sequential decoding algorithm is described as follows.

We say that a constraint is *satisfied* by a setting of a set of variables if the sum of those variables is 0. Else it is *unsatisfied.*

• If there is a variable that is in more unsatisfied than satisfied constraints, then flip the value of that variable.

If the expansion of sets of size at most αn is greater than $(3/4)r$ and if the number of errors in a received word w' is at most $(\alpha/2).n$ then the simple sequential algorithm will decode correctly in linear time.

- Repeat until no such variable remains.

We now show that the algorithm decodes in linear time. It requires an expansion greater than what is enough to prove Lemma 1 and Theorem 1. So if such an expansion is guaranteed, we can use the lemma and theorem.

Theorem 2. *If the expansion of sets of size at most αn is greater than $(3/4)r$ and if the number of errors in a received word w' is at most $(\alpha/2).n$ then the simple sequential algorithm will decode correctly in linear time.*

Proof. Let $\mathbf{w} = (w_1, w_2, \quad . w_n)$ be the transmitted codeword and $\mathbf{w}' = w'_1, w'_2, \quad . w'_n$ be the received vector, and at any point in the algorithm, let A be the set of errors in $\mathbf{w}'$. That is $A = \{v : w_v \neq w'_v\}$. If A is empty then we are done. Otherwise assume that $|A| \leq \alpha.n$. A has two kinds of neighbours, satisfied neighbours S and unsatisfied neighbours U so that $N(A) = S \cup U$ By assumption

$$|U| + |S| > \frac{3}{4}r|A|. \tag{4}$$

Now count the edges between A and $N(A)$. There must be an odd number of edges going into every node in U and an even number of edges going into every node in S from A as the constraints are modulo 2 sums. Therefore there are at least $|U|$ edges leaving U and at least $2|S|$ edges leaving S and entering A. Since this a r-regular graph on the left,

$$|U| + 2|S| \leq r|A|. \tag{5}$$

Combining (4) and (5) we have

$$r|A| - |U| \geq 2|S| > 2(\frac{3}{4}r|A| - |U|). \tag{6}$$

Therefore

$$|U| > \frac{r}{2}|A|. \tag{7}$$

So the total number of unsatisfied neighbours of the $|A|$ members of A is greater than $\frac{r}{2}|A|$ if $|A| \leq \alpha n$. There-

fore there is at least one element of A that has greater than $r/2$ unsatisfied neighbours. Recall r is the degree of every vertex on the left. So what this means is that as long as there are errors, there will be at least one variable in A that has more unsatisfied than satisfied neighbours provided the expansion condition is satisfied and the size of the set A at any time is $\leq \alpha n$. However this does not mean that the decoding algorithm will decide to flip a corrupt variable. All it guarantees is that if $|A| \leq \alpha n$ the algorithm will decide to flip *some* variable and thereby *reduce* the size of the set U

So what this means is that as long as there are errors, there will be at least one variable in A that has more unsatisfied than satisfied neighbours provided the expansion condition is satisfied and the size of the set A at any time is $\leq \alpha n$.

To show that the size of the set A never exceeds αn during the course of the algorithm we observe that at each step we flip some variable. Now if the variable was in the error set in the first place, we have reduced the size of the error set A and therefore we stay within the bound. On the other hand if we flip a variable not originally in A then the size of A increases by 1. If at any time the size of A exceeds αn then at the previous step it must have been αn. Therefore by the previous argument

$$|U| > \frac{r}{2}\alpha n. \tag{8}$$

Now in the beginning

$$|U| \leq |N(A)| \leq r\frac{\alpha}{2}n. \tag{9}$$

(The second inequality follows from the fact that this is a left r-regular graph and the size of A is initially bounded by $\alpha n/2$). Also during the course of the algorithm the size of the set U keeps decreasing by definition. (Either by flipping bits in the original error set, or bits not in the original error set). Hence at any point of the algorithm

$$|U| \leq r\frac{\alpha}{2}n. \tag{10}$$

Thus we have a contradiction and hence the size of the set A can never exceed αn. Therefore the algorithm will

Since the degrees of the bipartite graph are constant, checking satisfied and unsatisfied neighbours is a constant time operation. Also since the size of the set U decreases at each iteration and the number of constraints is upper bounded by the length of the code, the algorithm runs in linear time.

eventually terminate with all constraints satisfied. We summarize the steps so far in the proof as follows:

1. The expansion property of the graph, and the upper bound on the size of A guarantee that some variable in A is eligible for flipping.

2. An eligible variable flipped at each iteration implies that the size of U decreases.

3. The upper bound on the assumed number of errors implies an upper bound on size of A. Thus U must finally become empty and we get the decoded codeword.

We have still to show that the algorithm decodes to the *correct* codeword under all the assumptions made. We can assume without loss of generality that the all-0 codeword was sent as this is a linear code. We have seen that at no point in the algorithm does the size of the set A which contains all the erroneous bits exceed αn. But there is no codeword of weight $\leq \alpha n$, so such a word can never be output by the algorithm. Since the degrees of the bipartite graph are constant, checking satisfied and unsatisfied neighbours is a constant time operation. Also since the size of the set U decreases at each iteration and the number of constraints is upper bounded by the length of the code, the algorithm runs in linear time.

Sipser and Spielman also introduced a constructive family of asymptotically good linear error-correcting codes together with a simple parallel algorithm that will always correct a constant fraction of errors. In the next part of this article we will explain their result and also describe another very clever construction of an expander code with a simple and beautiful decoding algorithm.

Address for Correspondence
Priti Shankar
Department of Computer Science and Automation
Indian Institute of Science
Bangalore 560 012, India
Email:priti@csa.iisc.ernet.in

Suggested Reading

[1] **Stephen B Wicker and Saejoon Kim, *Fundamentals of Codes, Graphs, and Iterative Decoding*, Kluwer Academic Publishers, 2003.**

[2] **M Sipser and D Spielman, Expander codes, *IEEE Transactions on Information Theory*, Vol.42, No.6, pp.1710-1722, 1996.**