CSE 533: Error-Correcting Codes **(Autumn 2006)**

Lecture 13: Expander Codes

Nov 15, 2006

Lecturer: Venkatesan Guruswami *Scribe: Laura Elisa Celis*

In this lecture we define $(n, m, d, \gamma, \alpha)$ expander graphs and codes. We show that expander codes exist with rate $1 - m/n$ and distance $2(1 - \varepsilon)\gamma n$. We analyze the Sipser-Spielman degree decoding algorithm and show that it decodes correctly up to almost half the minimum distance, approaching this bound as α approaches 1.

1 Expander Graphs and Codes

Consider a bipartite graph $G(L, R, E)$ where every vertex in L has degree d, $|L| = n$ and $|R| = m \leq n$. The graph G is said to be a $(N, M, d, \gamma, \alpha)$ expander if every $S \subset L$ where $|S| \leq \gamma n$ has at least $d\alpha|S|$ neighbors. Ideally we want α close to 1 (note that α can never be greater than 1) and $\gamma >> 0$. The expansion property holds trivially for $\gamma \leq 1/n$. If $1/n < \gamma \leq 1$ and $\alpha = 1 - \varepsilon$ for constant ε, we say that G is a lossless expander.

Let us quickly introduce some notation that will become useful later. Let $N(S)$ denote the set of neighbors of S. In general we will consider $S \subseteq L$, so $N(S) \subseteq R$. A vertex $v \in R$ is said to be a unique neighbor of S if there is a single edge from v to S. Let $U(S)$ denote the collection of unique neighbors of S.

Since G is bipartite, we can consider its $n \times m$ bipartite adjacency matrix where the zero rows and columns are disregarded. Let H be the transpose of this matrix. The code C obtained by viewing H as the parity check matrix is an expander code. In this view, each codeword corresponds to a subset $S \subseteq L$. Specifically, if a given codeword c has k nonzero entries, the corresponding set S is of size k and contains $v_i \in L$ if the ith entry of c is nonzero. This vector is called the *characteristic vector of* S.

Notice this means a subset $S \subseteq L$ corresponds to a codeword if and only if every $y \in N(S)$ is adjacent to an even number of elements of S. Thus, the minimum distance of C is the size of the smallest non-empty subset S that has an even number of edges (possibly 0) to every vertex in R. Additionally, since we place at most m linearly independent constraints on the codewords of C, our code has dimension at least $n-m$. Hence the rate of the code is at least $(n-m)/n = 1-m/n$.

2 Existential Knowledge

Borrowing results from combinatorics, we know that lossless expanders exist. More precisely, the following theorem holds.

Theorem 1. *For all $\varepsilon > 0$, and all $m \leq n$, there exist $\gamma > 0$ and d such that a $(n, m, d, \gamma, 1-\varepsilon)$-expander exists. Additionally,*

$$d = \Theta\left(\frac{\log 2N/M}{\varepsilon}\right) \text{ and } \gamma N = \Theta\left(\frac{\varepsilon M}{d}\right).$$

The proof of this theorem is done by probabilistic method, and can be found in *Randomness Conductors and Constant-Degree Lossless Expanders* by Capalbo, Reingold, Vadhan, and Wigderson written in 2002. The authors also show that these codes are polynomial time constructible for constant ε and ratio n/m. Thus we can build arbitrarily large $1-\varepsilon$ expander codes with fixed positive rate. Additionally their construction results in lossless expander graphs with constant degree D .

3 Rate and Distance

A factor graph is a parity check matrix viewed as a bipartite graph. An expander code is a code whose factor graph is an expander graph.

Theorem 2. *An expander code where the factor graph is a $(n, m, d, \gamma, 1-\varepsilon)$ - expander where $\varepsilon < 1/2$ has rate $R \geq 1 - \frac{m}{n}$ and distance $d \geq 2(1-\varepsilon)\gamma n$.*

The proof for the bound on the rate is simple and shown above. We will prove the bound on the distance distance using the following lemmas.

Lemma 1. *For any $S \subset L$ of size at most γn,*

$$d|S| \geq |N(S)| \geq |U(S)| \geq d(1-2\varepsilon)|S|.$$

Proof. Clearly $|N(S)| \geq |U(S)|$ since the number of unique neighbors of S cannot exceed the total number of neighbors of S. The number of edges out of S is $d|S|$, so $d|S| \geq |N(S)|$. Of these, due to the expansion property of the graph, $d(1-\varepsilon)|S|$ edges go to distinct vertices. The remaining $d\varepsilon|S|$ edges are arbitrarily distributed, so they can eliminate the uniqueness of at most $d\varepsilon|S|$ vertices in $N(S)$. Hence $|U(S)| \geq d(1-\varepsilon)|S| - \varepsilon|S| = d(1-2\varepsilon)|S|$. □

More precisely, this shows that if $|S| < \gamma n$, then S has a unique neighbor. Recall that if a set $S \subseteq L$ has a unique neighbor, then the characteristic vector of S cannot belong to C. Thus the minimum distance is at least γn. We can improve this bound with the following lemma.

Lemma 2. *Every set $T \subset L$ where $|T| < 2(1-\varepsilon)\gamma n$, has a unique neighbor.*

Proof. If $|T| \leq \gamma n$, the statement hold from the above lemma. Thus we can assume $|T| > \gamma n$. Consider some $S \subseteq T$ such that $|S| = \gamma n$. From the above lemma, we know that $|U(S)| \geq d(1-2\varepsilon)\gamma n$. Notice that a vertex $v \in U(S)$ lies in $U(T)$ if it is not in $N(T \setminus S)$. However, $|T \setminus S| < 2(1-\varepsilon)\gamma n - \gamma n = (1-2\varepsilon)\gamma n$. Since the degree is d, we know $|N(T \setminus S)| < d(1-2\varepsilon)\gamma$. Thus, $|U(S)| > |N(T \setminus S)| > 0$ since $\varepsilon < 1/2$. Thus $U(T)$ is nonempty. □

Hence, by the same argument as above, the minimum distance of an expander code is at least $2(1-\varepsilon)\gamma n$. Thus good codes with positive distance and rate exist.

4 Decoding

A natural greedy algorithm was proposed and analyzed by Sipser and Spielman in *Expander Codes* in 1996. For this algorithm, we think of the vertices in L as variables, and the vertices in R as constraints. As discussed above, the constraint is that an even number of vertices adjacent to each element of R must be nonzero. In other words, the constraint given by a vertex $r \in R$ is that $\sum_{v \in N(r)} v = 0$. In this algorithm we repeatedly look for a variable which is in more unsatisfied constraints that satisfied constraints and flip it. This will terminate either when we reach a codeword (if no unsatisfied constraints remain) or when we fail to find a variable to flip, in which case the decoding algorithm fails. Notice that for the complexity analysis we are assuming this algorithm is run in on a code with factor graph constructed as shown by Capalbo, Reingold, Vadhan, and Wigderson, which specifically have the property that m/n is a constant, and the graph has maximum degree D.

4.1 Greedy Algorithm

For this algorithm, we consider a node v_i to be the element of L that corresponds to the ith index in the codewords of C. Let $n(i)$ be the number of satisfied neighbors of v_i, and let $u(i)$ be the number of unsatisfied neighbors of v_i. Let $C \subseteq R$ be the set of unsatisfied constraints. The actual algorithm is as follows:

```
while |C| > 0
    if exists i such that u(i) > n(i)
        flip entry at i
    else
        fail
return codeword
```

To analyze the algorithm, we will keep track of the set of corrupt variables S and unsatisfied constraints C, whose sizes we will denote by s and c. We also note here that for this algorithm to work, we must have $\varepsilon < \frac{1}{4}$.

4.2 Correctness

To show this algorithm is correct, we must show that it terminates with the correct result when the input is within half the distance of a codeword.

Lemma 3. *If $s < \gamma n$, there is a variable which is in more unsatisfied constraints that satisfied constraints.*

Proof. Since $s < \gamma n$, from Lemma 1 we know $U(S) \geq d(1 - 2\varepsilon)s$. Thus an average vertex $v \in S$ has at least $d(1 - 2\varepsilon) > d/2$ unique neighbors (recall that for this algorithm we have assumed $\varepsilon < 1/4$), and hence v lies in more unsatisfied than satisfied constraints. □

Thus if the algorithm has not converged to a codeword, there will always be some variable to flip. Additionally, c decreases with each iteration, so the algorithm will terminate. In order to prove that the algorithm terminates at the correct codeword, we must show that the number of corrupt variables s also decreases. In other words, we are not introducing so many new errors that we will might converge to a different word.

Notice that c is upper bounded by $|N(S)|$, and lower bounded by $|U(S)|$. Thus, from Lemma 1, c and s are related by

$$ds \geq c \geq d(1-2\varepsilon)s.$$

In the ideal case when $\varepsilon = 0$, $c = sd$, thus if c decreases, so does s. When $\varepsilon > 0$, a decrease in c does not imply an immediate decrease in s. However, if $s \leq \gamma n$ after every step in the iteration, then the above inequality holds throughout. Thus, since c decreases to 0, s would also decrease to 0.

Let s_t and c_t be the number of corrupt variables and unsatisfied constraints after t steps of decoding. So $s_0 = s$ and $c_0 = c$.

Theorem 3. *If $s_0 < (1-2\varepsilon)\gamma n$, then $s_t < (1-2\varepsilon)\gamma n$ for all t.*

Proof. Initially, $s_0 < (1-2\varepsilon)\gamma n$ since we assume the received word is within half the minimum distance of the correct codeword. Thus $d|S| = d(1-2\varepsilon)\gamma n \geq c_0$. Assume for sake of contradiction that at some time $t \geq 1$, the number of corrupted variables $s_t \geq (1-2\varepsilon)\gamma n$. Since s_t must be an integer and $\varepsilon < 1/4$, this means $s_t \geq \gamma n$. Additionally, from the way the algorithm works, at each time step we flip exactly one variable. Thus, at some time $0 < t' < t$ we had $s_{t'} = \gamma n$. Then $c_{t'} \geq d(1-2\varepsilon)\gamma n$. However, $c_{t'} < c_0$, so this gives a contradiction. Hence $s_t < (1-2\varepsilon)\gamma n$ for all t. □

Hence the greedy decoding algorithm terminates at the correct codeword if the input is within half the minimum distance of the original word.

4.3 Complexity

Let us now determine the runtime of the decoding algorithm. Recall that m/n is fixed. Thus $m \in O(n)$. Clearly $c_0 \leq m$. Since c_t is monotonically decreasing, the while loop runs at most m times. Thus we need only determine how long it takes us to find a vertex v with more unsatisfied than satisfied constraints. We can use a bucketing technique where each bucket U_k contains all vertices with k more unsatisfied than satisfied neighbors. Note that these buckets can be set up initially in $O(dn)$ time. Then, we can simply grab a vertex from the bucket U_k with highest k. Since flipping v may affect other vertices, we now need to rebucket. However, v is in exactly d constraints, and each of these constraints contains at most D distinct vertices. No other vertices can be affected, so we need to rebucket no more than dD vertices. Since this is a constant, the entire algorithm runs in $O(n)$ time!