CS369E: Expanders	April 4, 2005

Lecture 1: Expanders – an Introduction

Lecturer: Cynthia Dwork *Scribe: Arpita Ghosh*

In this lecture, we start with some basic definitions and notation, and then discuss some motivating applications of expanders.

A graph $G = (V, E)$ has vertex set V, with $|V| = N$, and edge set E. G is undirected, unless specified otherwise, and can be a multigraph. A bipartite graph will be denoted as $G = (L \cup R, E)$, *i.e.*, its vertex sets are partitioned as L and R (L and R do not necessarily have the same size).

For each vertex $v \in V$, the neighborhood of v is denoted $\Gamma(v)$, and defined as

$$\Gamma(v) = \{u \in V \mid (u, v) \in E\}.$$

The neighborhood of a subset $S \subseteq V$ is defined as the union of the neighborhoods of the vertices in S, *i.e.*,

$$\Gamma(S) = \cup_{v \in S} \Gamma(v).$$

We also define

$$\Gamma^{'}(S) = S \cup \Gamma(S),$$

i.e., $\Gamma^{'}(S)$ consists of vertices in S as well as their neighbors.

Definition 1.1. *A graph G is said to have vertex expansion (K, A) if*

$$|\Gamma(S)| \geq A \cdot |S|, \qquad \forall S \subseteq V \text{ with } |S| \leq K.$$

We will then say that G is a (K, A)-expander

Clearly, a complete graph has the best possible expansion. However, we will be interested in constant degree graphs. We will typically want $K = cN$, for some constant c, and A to be of the order of the degree d of the graph. When $K = N/2$ (this will be the most common setting in this course), we will refer to G as an A-expander. Informally, an expander graph is one where all subsets of V (under some constraint on their size) have large neighborhoods. Now we will see some applications and results related to expanders. Surprisingly, such graph do exisit. For the present we will assume the existence of such graphs (and also that we can construct them efficiently). We will defer the actual constructions of such expanders to the second half of this course.

We will now consider applications of such graphs in five different contexts.

- Time-space tradeoffs
- Byzantine Agreement
- Saving Random Bits
- Error Correcting Codes
- Metric Embeddings

Two of these applications (the first and the last) are lower bounds while the other two are upper bounds. These examples illustrate the use of expanders in a wide variety of contexts, both for positive and negative results. We will not furnish full proofs in these examples, however.

1.1 Time-space tradeoffs in computing functions

One of the first uses of expanders in computer science was in studying the computational hardness of certain functions. Given a specific method for computing a function we can create an acyclic computational circuit describing the computation. Valiant was studying the hardness of computing certain linear transforms, and observed that every circuit computing these functions was a *superconcentrator* [Val]. Intuitively, these are graphs with very great flow from inputs to outputs: An n-superconcentrator is a graph with n inputs and n outputs, such that for all subsets S of the input, and all subsets T of the output with $|T| = |S|$, there exist vertex disjoint paths connecting S to T. Valiant hoped to obtain lower bounds for the linear transforms by obtaining high lower bounds for superconcentrators. Valiant conjectured that any superconcentrator must have a superlinear number of edges. However, Valiant himself disproved this conjecture and gave construction of superconcentrators with $O(n)$ edges using expanders. This happens to be the first context when expanders where used in Computer Science. For an interesting account of the details that went into the discovery of expanders, refer the writeup on Reingold's recent "SL=L" result by Sara Robinson [Rob].

However, it turns out that superconcentrators exhibit interesting time/space tradeoffs, leading to time/space tradeoffs for computations (see the work of Paul, Tarjan and Celoni [PTC, PT]). We will show how a pebbling game on such graphs leads to some lower bounds. Consider the following pebbling game played on a DAG (directed acyclic graph). To start with, the DAG has some specified input nodes and output nodes, and some auxiliary nodes connecting inputs to output. All edges are directed and go from left to right (inputs are on the left, and outputs on the right). We are given S pebbles, with the following rules:

- A pebble can be placed on an input at any time.
- A pebble can be placed on a non-input node if all its predecessors (*i.e.*, nodes with edges leading into it) currently hold pebbles.
- A pebble can be removed at any time.

The goal is to evaluate the outputs, *i.e.*, pebble each of the outputs. For example, suppose the DAG represents a computational circuit. Clearly, with a large number of pebbles S, the outputs can be computed in a small number of steps. If every graph to evaluate a function is such that it cannot be pebbled quickly with a small number of pebbles, then the function is hard to compute with a small amount of memory. For further details, refer to the work of Celoni, Tarjan and Paul [PTC, PT].

As stated earlier, superconcentrators with $O(n)$ eedges can be constructed from constant degree expanders (we will not discuss this construction here). This construction uses the Hall's Theorem which states the necessary and sufficient condition for a graph to have a perfect matching. Since this condition is similar in flavor to the definition of expanders, we state and prove Hall's theorem here.

Theorem 1.2. *Let $G = (L \cup R, E)$ be a bipartite graph. Then, G has a perfect matching if and only if*

$$\Gamma(S) \geq |S|, \qquad \textit{for all } S \subseteq L. \tag{1}$$

Proof. The only if direction is obvious (if there is a subset of S with not as many neighbors as vertices, S cannot have a matching). We will show the if direction by induction on the size of L. The base case, where L has at most one vertex is trivial. Suppose now that the claim is true for some $L \leq m$. We will consider two cases:

- **Case 1:** All proper subsets S of L, $S \neq \emptyset$, expand strictly, *i.e.*, $|\Gamma(S)| > |S|$. Consider any vertex x in L, and let (x, y) be an edge in E. Now, consider the bipartite graph G^* with vertex sets $L^* = L - \{x\}$ and $R^* = R - \{y\}$. Since every $S \subset L$ satisfies (1) with strict inequality, every subset of L^* satisfies (1), since only a single vertex y has been removed from R. Therefore, by the induction hypothesis, the smaller graph G^* has a matching. To this matching add the edge (x, y); this gives a perfect matching in G.

- **Case 2:** There exists a proper subset $T \subset L$, $T \neq \emptyset$, with $|\Gamma(T)| = |T|$. Consider the induced graphs G_1 and G_2 on the vertex sets $T \cup \Gamma(T)$, and $L \setminus T \cup R \setminus \Gamma(T)$ respectively. By the induction hypothesis, G_1 has a perfect matching. (Note that the induction hypothesis cannot be used directly on G_2.) Let $S \subseteq L \setminus T$. Then,

$$\begin{aligned} \Gamma_{G_2}(S) &= \Gamma_G(S \cup T) \setminus \Gamma_G(T) \\ \Rightarrow |\Gamma_{G_2}(S)| &\overset{(a)}{\geq} |S \cup T| - |T| \\ &= |S|, \end{aligned}$$

  where (a) is true since $S \cup T$ satisfies $|\Gamma_G(S \cup T)| \geq |S \cup T|$, and by assumption, $|\Gamma(T)| = |T|$. Therefore, the graph G_2 also satisfies (1), and by the induction hypothesis, has a matching. The unions of the perfect matchings in G_1 and G_2 is a matching for G.

□

1.2 Almost Everywhere Byzantine Agreement

In the Byzantine agreement problem each of n processors begins with an input value, say, $v_i \in \{0, 1\}$. During the course of the computation each processor must irreversibly *decide* on an output value d_i. The requirements are

- (Unanimty): All non-faulty processors must produce the same decision.
- (Non-triviality): If all non-faulty processors begin with the same value, say v, then every non-faulty processor must output v.

Another application of expanders occurs in the context of solving Byzantine agreement in general networks, where the processors correspond to nodes and a processor can only communicate with its immediate neighbors. Dolev showed that for t-resilient Byzantine agreement, connectivity greater equal $2t + 1$ is necessary [Dol].

When t is linear in the number n of vertices in the network, this requires $O(n^2)$ edges, which is unreasonable for large n. Consider, instead, a constant-degree expander on n vertices. In such a network, if not too many nodes are faulty, then it can be shown that there is a large fraction of non-faulty nodes that can communicate "as if" they were in a completely connected network. Intuitively this is because expanders do not have small cuts.

By relaxing the unanimity requirement, essentially permitting $O(t)$ non-faulty processors to be "lost" (not to join in the majority decision), we obtain almost-everywhere agreement [DPPU]. Using constant-degree expanders (together with other special graphs), the relaxed problem can be solved in networks of bounded degree. For further details, refer [DPPU] and [Upf].

1.3 Saving random bits

Another application of expander graphs occurs in reducing the number of random bits used by a randomized algorithm. We will study several such applications of expanders in the context of derandomization. For today's introductory lecture, we will study a simple and beautiful application due to Karp, Pippenger and Sipser [KPS].

Recall that a language $\mathcal{L}$ belongs to RP if there exists a polynomial time Turing machine M using a sequence of random bits (of length polynomial in the input size), such that

$$x \in \mathcal{L} \Rightarrow \frac{|W_x|}{2^{p(|x|)}} \geq q \geq \frac{3}{4},$$

and

$$x \notin \mathcal{L} \Rightarrow |W_x| = 0,$$

where

$$W_x = \{y | M(x,y) = 1, |y| = poly(|x|)\}.$$

That is, if $x \in \mathcal{L}$, the probability that $M(x,y) = 0$ (which is the probability of error) is less equal $1-q$, a constant (less than $1/4$). That is, if the algorithm M returns 1 on input (x,y), then surely $x \in \mathcal{L}$, but if it returns 0, then x may or may not belong to $\mathcal{L}$.

Let r be the number of random bits used to generate the string y, $r = poly(|x|)$. Suppose we want the probability of error to be δ. Since the error decreases by a constant fraction each time, to reduce the probability of error down to δ, we can repeat the algorithm $O(\log(\frac{1}{\delta}))$ times. But for this, the total number of random bits used is $O(r\log(\frac{1}{\delta}))$, *i.e.*, we need to use a large number of random bits to make the probability of error small.

Using expanders, it is possible to obtain, in polynomial time, to reduce the error to as low as $\frac{1}{\mathrm{poly}(r)}$ with no extra random bits (*i.e.*using only the original r random bits).

The expander used is a giant $(N/2, A)$-expander G on $N = 2^r$ vertices where A is the expansion of the expander. Note that since the expander is so large, we cannot even afford to write down the entire expander, forget constructing it. However, we will assume that there exists an implicit construction of the expander in the following sense: given any vertex v and any index i in the range $1 \ldots d$ (where d is the degree of the expander), we can in time polynomial in $|v|$ and $|i|$, compute the i^{th}-neighbor of v. The expanders constructions we will discuss later in the course will satisfy such strong properties.

Choose a radius c such that $\frac{1}{4A^c} \leq \delta$, where A is the expansion of G. Choose a vertex v uniformly at random from G. This needs r random bits, since $|V| = 2^r$. The modified RP algorithm is as follows:

1. Run the original RP algorithm M for all strings y lying within a ball of radius c around v.

2. If for all these y, $M(x,y) = 0$, reject x.

3. If $M(x, y) = 1$ for any y, accept x.

We will show that the error of this modified RP algorithm is at most δ.

Suppose $x \in \mathcal{L}$. Define Bad_x to be the set of y for which $M(x, y) = 0$. The output is erroneous only if $\Gamma'_c(v) \subseteq Bad_x$, where the subscript c denotes the set of vertices within a distance c of v. Let

$$B = \{v \mid \Gamma'_c(v) \subseteq Bad_x\}.$$

The algorithm fails only when v is picked from B. Now, by definition of B, for $1 \leq i \leq c - 1$,

$$\Gamma'_i(B) \subseteq \Gamma'_{i+1}(B) \subseteq Bad_x.$$

Also, by definition, since we started with an input $x \in \mathcal{L} \in RP$,

$$|Bad_x| \leq N/4.$$

Since G has an expansion of A,

$$|\Gamma'_c(B)| \geq A^c|B|.$$

Combining all of this, we have

$$\frac{N}{4} \geq |Bad_x| \geq |\Gamma'_c(B)| \geq A^c|B|,$$

and therefore,

$$\frac{|B|}{N} \leq \frac{1}{4A^c} \leq \delta.$$

But the algorithm only fails when the vertex v is picked from the set B, which has a probability of $\frac{|B|}{N}$. Observe that the running time of the new algorithm is $\text{poly}(1/\delta)$. This limits the minimum error that can be attained by this technique. Therefore, we can get any error $\delta = \frac{1}{\text{poly}(r)}$ by choosing c appropriately, with only r bits of randomness.

1.4 Error correcting codes

The next application comes from error correcting codes. Suppose a sender A is sending a k-bit message to receiver B over a faulty channel, which could flip up to a fraction p of the bits sent over it. The message is encoded by A into an n-bit codeword in $C \subset \{0, 1\}^n$, where the size of the C is 2^k (there is a unique codeword for each message). The decoding is simple: given an n-bit vector, find the closest codeword $w \in C$, and return the k-bit message that maps to w. The rate R of the code is defined to be

$$R = \frac{\log |C|}{n} = \frac{k}{n}.$$

Clearly, if the Hamming distance between every two codewords is strictly greater than $2pn$, then the message can be decoded without error (there is a unique closest codeword $w \in C$ for each n-bit string.) Define the distance δ of a codebook to be

$$\delta = \min_{c_1 \neq c_2 \in C} \frac{d_H(c_1, c_2)}{n},$$

where d_H, the Hamming distance, is the number of bit positions in which the vectors differ.

The communication problem is the following:
Is it possible to define a family of codes, $\{C_k\}_{k=1}^{\infty}$, such that $|C_k| = 2^k$, and for each k, $\delta_k > 0$, and $R_k > 0$?

We will show that codes with good distance can be constructed from expanders which have very good expansion (more precisely, when the expansion factor is greater than half the degree). The first such construction of expander based codes was given by Tanner [Tan]. Suppose we have an $(\alpha N, K)$ d-regular expander G, where $K > d/2$. Consider a bipartite graph with $L = n$, with the vertices in L the components of a (n-bit) codeword. Each vertex in R represents a constraint, which are of the form $\oplus x_v = 0$, where x_v is the vth component of the (n-bit) codeword x. Thus if vertex $j \in R$ has neighbors, say $i_1, i_2, i_3 \in L$, then vertex j represents the constraint $x_{i_1} \oplus x_{i_2} \oplus x_{i_3} = 0$. (Thus, the larger the number of constraints, the smaller the size of the code, and vice versa.)

Note that a code which is the set of Boolean vectors satisfying constraints of the form $\oplus x_v = 0$ is a linear code, since $(0, \ldots, 0)$ belongs to the code, and if c_1 and c_2 belong to the code, then so does $c_1 \oplus c_2$. It can easily be checked (using the above definition) that the distance of a linear code is the weight of the minimum weight (non-zero) codeword.

Now, since G has an expansion of $K > d/2$, for every subset S of L of size less equal αN, there is a $v \in \Gamma(S)$ such that v is adjacent to a unique element of S. Suppose not; then, every vertex in $\Gamma(S)$ is connected to at least two vertices in S. Let E be the number of edges between S and $\Gamma(S)$, then

$$\begin{aligned} & d|S| = |E| \geq 2|\Gamma(S)| \\ \Rightarrow \quad & |\Gamma(S)| \leq \frac{d}{2}|S|. \end{aligned}$$

But this contradicts the $K > d/2$ expansion of G.

Now, this means that there cannot be a codeword with only αN ones in it, since then we can choose S to be this subset of αN ones, and have a violated constraint. Therefore, the minimum weight codeword has a weight strictly greater than αN, and thus $\delta \geq \alpha$.

If there are $n(1-c)$ constraints, then the set of codewords simultaneously satisfying these constraints is a linear subspace of $\{0,1\}^n$, with size $|C| \geq 2^{nc}$. Therefore, the rate of such a code would be $nc/n = c$, which is a constant greater than zero.

Thus, the existence of a family of expanders, with an expansion of $K > d/2$, and $|R| = (1-c)|L|$ allows the construction of a family of codes $\{C\}_k$, with rate and minimum Hamming distance strictly greater than 0 for all k. We will return to the applications of expanders to error-correcting codes in one of the later lectures.

1.5 Metric embeddings

Another use of expanders occurs in metric embeddings. Let M on (X, D) (*i.e.*, a metric on X with distance measure D) and M' on (X', D') be two metrics. An embedding $f : M \to M'$ has distortion c if

$$\forall x, y \in X, \quad D(x,y) \leq D'(f(x), f(y)) \leq cD(x,y).$$

Bourgain showed that any n-point metric space can be embedded into l_2-metric [1] with distortion $O(\log n)$ and dimension at most $O(\log n)$ [Bou].

Embeddings are used in several contexts. For instance, it might be the case that a problem (say the Travelling Salesman Problem) is hard in an arbitrary metric, but is slightly more tractable in a well-understood metric such as l_2. In this case, it is plausible that embedding the arbitrary metric into l_2, solving the problem in l_2 and retransforming the problem back to the original metric gives some insight into the solution of the problem in the "hard" metric.

The following is another application of metric embeddings into l_2. Suppose we are given a graph $G = (V, E)$ with weights (or distances) on edges, and the distance between an arbitrary pair of nodes i and j is the shortest path distance between i and j on G. We want to be able to quickly answer (approximately) a query of the form 'what is the shortest path distance between vertices i and j in G', where i and j are arbitrary. To answer this question exactly, the storage required is $O(n^2)$ (store all the shortest path distances). However, the l_2 embedding allows us to simply compute the l_2 distance between the queried vertices in the l_2 embedding, which is a $O(\log n)$ approximation to the actual shortest path distance between the same vertices in G. The storage required for the l_2 embedding is $O(n \log n)$; since the l_2 embedding is into a $O(\log n)$ dimensional space (which is, of course, better than the $O(n^2)$ storage for the exact solution).

Expanders happen to be some of the worst case examples for embedding. In this sense, expanders are sometimes used to show the limits of embedding. For instance, London, Linial and Rabinovich showed that this is actually tight, by constructing expanders for which the distortion is $\Omega(\log n)$ [LLR].

References

[Bou] Jean Bourgain: "On Lipschitz embedding of finite metric spaces in Hilbert space", Israel J. Math 52, 46–52 (1985).

[Dol] Danny Dolev: "The Byzantine Generals Strike Again". J. Algorithms 3(1): 14-30 (1982)

[DPPU] Cynthia Dwork, David Peleg, Nicholas Pippenger, Eli Upfal: "Fault Tolerance in Networks of Bounded Degree". SIAM J. Comput. 17(5): 975-988 (1988)

[KPS] Richard Karp, Nicholas Pippenger and Michael Sipser: "A time-randomness tradeoff", AMS Conference on Probabilistic Computational Complexity, 1985.

[LLR] Nathan Linial, Eran London, Yuri Rabinovich: "The Geometry of Graphs and Some of its Algorithmic Applications". Combinatorica 15(2): 215-245 (1995)

[PT] Wolfgang J. Paul, Robert Endre Tarjan: "Time-Space Trade-Offs in a Pebble Game". Acta Inf. 10: 111-115 (1978)

[PTC] Wolfgang J. Paul, Robert Endre Tarjan, James R. Celoni: "Space Bounds for a Game on Graphs". Mathematical Systems Theory 10: 239-251 (1977)

[Rob] Sara Robinson: "Computer Scientist Finds Small-Memory Algorithm for Fundamental Graph Problem", SIAM News, Volume 38, Number 1, January/February 2005.

[1] l_2 is $\mathbb{R}^n$ equipped with the usual Euclidean metric.

[Tan] Robert M. Tanner: “A recursive approach to low complexity codes”, IEEE Trans. Information Theory, 27(5): 533-547, 1981.

[Upf] Eli Upfal: “Tolerating a Linear Number of Faults in Networks of Bounded Degree” Inf. Comput. 115(2): 312-320 (1994)

[Val] Leslie G. Valiant: “Graph-Theoretic Properties in computational Complexity.” J. Comput. Syst. Sci. 13(3): 278-285 (1976)