CS 880: Advanced Complexity Theory 1/23/06

Lecture 1: Introduction

Instructor: Dieter van Melkebeek Scribe: Michael Kowalczyk

In this lecture the course outline is presented. The definition of an expander is introduced and several applications of expanders are discussed. All of these applications will be covered in detail in later lectures.

1 Course Outline

In this installment of advanced complexity theory, we will study expander graphs and their applications. Our motivation for studying expanders is that they provide an abundance of applications in computer science. In particular, we will focus on complexity theoretic upper and lower bounds in which expanders play an important role. After introducing the original definition and spectral characterization of expander graphs, we will prove the existence of expanders and study some explicit constructions. Most of the time will be spent on applications. Time permitting, related topics such as extractors may also be covered.

2 Expander graphs

We start with the original definition of expander graphs. Informally, an expander graph is a sparse graph which is highly connected. Even though such a graph has few edges, it has some properties similar to the complete graph. The main idea is that sufficiently small subsets of vertices always have relatively large neighbor sets. Formally, we give the following definition (when we say "graph" we will always mean an undirected multigraph unless stated otherwise).

Definition 1. *A graph $G = (V, E)$ is (K, c)-expanding if for all subsets $A \subseteq V$ with $|A| \leq K$, we have $|\Gamma(A)| \geq c \cdot |A|$, where $\Gamma(A) = \{w \in V | (\exists v \in A)((v, w) \in E)\}$.*

For this concept to be nontrivial, we need $c > 1$. Otherwise, the definition is easily satisfied by a graph with self loops everywhere. Furthermore, note that we need the size restriction on subsets in order for the definition to make sense, because we can never have $|\Gamma(V)| > |V|$. Typically, we will take $K = \alpha \cdot N$ where $N = |V|$ and α is some constant between 0 and 1. When not specified, α will be assumed to be $\frac{1}{2}$. Typically we will require that G is a d-regular graph for some constant d. Of course, this implies that the graph is sparse.

The previous definition also applies for bipartite graphs, but in this case we will often relax the requirement and only require the expanding property on subsets of the left vertex set, as follows.

Definition 2. *A bipartite graph $G = (L, R, E)$ is (K, c)-expanding if for all subsets $A \subseteq L$ with $|A| \leq K$, we have $|\Gamma(A)| \geq c \cdot |A|$, where $\Gamma(A) = \{w \in R | (\exists v \in A)((v, w) \in E)\}$.*

Note that for reasonable settings of the parameters, it is not yet clear that these objects even exist. An existence proof will be given in the next lecture.

3 Upper bounds

3.1 Superconcentrators

Expanders were first invented for their application in communication networks. One such example is the construction of linear size superconcentrators. This problem arrives from building switches for telephone networks. A superconcentrator is a graph $G = (V, E)$ with n input vertices and n output vertices such that for any subset I of the inputs and O of the outputs with $|I| = |O|$, there exist $|I|$ vertex disjoint paths connecting I and O. In application, we can imagine n phone lines that need to be connected to n other phone lines. The idea is that no matter what subset of phone lines are active, there is a way to set the intermediate switches to provide service through the superconcentrator.

Of course, if it doesn't matter how many edges the graph has, the complete bipartite graph will suffice. However, such a design would imply quadratic growth in the size (and cost) of the superconcentrator as the number of inputs and outputs increase. It is obvious that the smallest possible growth in the number of edges is linear, but it remained an open problem for some time whether this was actually achievable. It turns out that it is possible to build these graphs with only a linear number of vertices, and with each vertex having constant degree. The construction is roughly based on the idea that in an expander graph, any subset can reach the whole graph without taking too many steps.

3.2 Error correcting codes

A second application, also in the realm of communication, is error correcting codes. Important parameters in the design of error correcting codes include the rate and the relative distance. The rate is defined to be $\frac{k}{n}$ where k is the size the of an information word and n is the size of a codeword. Ideally, this ratio is kept as close to 1 as possible, so that the total bandwidth used is small. Relative distance is defined to be $\frac{h}{n}$ where h is the minimum hamming distance between two codewords. This is important, as we can correct t errors in any codeword if $h = 2t + 1$. Of course, these two requirements oppose each other, but we would like to be able to correct a constant fraction of errors without causing more than a constant factor increase in message size. Finally, we would like to have efficient algorithms for encoding and decoding. We will be able to realize such constructions by using expander graphs to construct the code.

Now we sketch the idea. One natural way of describing a linear error correcting code is the generator matrix, which maps any information word to the corresponding code word. Equivalently, the code can be represented by the parity check matrix, which is simply a $(0, 1)$-matrix where the kernel is precisely the set of codewords. It is easy to see why this is called the parity check matrix, as each row in the matrix is equivalent to the requirement that the sum of particular bit positions in the codeword has even parity. Another way to model these conditions is to consider a bipartite graph where each vertex on the left is a bit position in the codewords, each vertex on the right is a parity-check test, and any given test is the exclusive-or of its adjacent vertices. We will construct an error correcting code by using a bipartite expander.

3.3 Deterministic amplification

In general, derandomization refers to techniques which reduce the total number of random bits required by some randomized process. Expanders become a useful tool in this endeavor, and a specific case is deterministic amplification.

To illustrate the concept of deterministic amplification, imagine a randomized algorithm, A, that solves some decision problem with error at most $\frac{1}{3}$. We would like to reduce the error, and one way of doing this is running A a number of times independently, and taking a majority vote of the outputs. This is effective, but requires a high number of random bits, which may be limited in the given application. There are other alternatives for reducing the number of random bits required, but in deterministic amplification, we focus on reducing error without using any extra random bits at all. That is, you are only given a string r with enough random bits to run A once.

A first attempt might be to run A on r and on some fake random strings calculated deterministically from r in a straightforward way (say, all the strings at hamming distance one from r). But then it isn't too hard to cook up a decision problem for which this amplification procedure doesn't amplify the confidence much at all. This is because you can create the set in such a way that the neighbors (under the deterministic operation) remain inside the set of random strings that give the wrong answer.

However, this looks like an application for expanders. When using an expander, the neighbor set cannot be forced to remain small. If we use a constant degree expander and retrieve all of the strings within $\log(n)$ steps of r on the expander graph, then we have a polynomial number of random strings to use as input to A under the majority vote. We will prove that this decreases the error of A in general.

3.4 Pseudorandom generators

Another application for expanders is pseudorandom generators for space bounded computation. These are deterministic procedures that take a random string as input and expand it to a significantly longer string that still looks random to a space bounded algorithm. As an example, for randomized algorithms that use logarithmic space, we can construct pseudorandom generators that use a random seed of length $\log^2(n)$. The pseudorandom generator will expand random seeds of this length to polynomial length strings which will cause the logspace computation to have about the same output distribution as it would have on truly random bits.

3.5 Space efficient deterministic algorithm for st-connectivity

One specific problem that can be solved with only a logarithmic amount of workspace is undirected st-connectivity. We are given an undirected graph with two distinguished vertices s and t, and we want to know whether these two vertices are connected. It can be solved easily with breadth first search or depth first search, but this requires a linear amount of workspace. On the other hand, a random walk will only take a logarithmic amount of space, and it can be shown that if s and t are connected, then the walk will visit t within $O(n^3)$ steps with high probability.

A recent breakthrough was that the same problem can be solved with a deterministic algorithm that uses only a logarithmic amount of space. We make an observation here that comes into play. Namely, this problem can easily be solved in deterministic logarithmic space on expander graphs. The expanding property of the graph can guarantee that the longest distance between two vertices is only logarithmic. Thus, when applying, say, a depth first search, there would only need to be a logarithmic number of recursion levels. If the degree of the graph is constant then only a constant amount of information needs to be stored per recursion level, and this gives us a logarithmic space bounded algorithm. Thus the challenge is to transform an arbitrary graph into an expander graph without changing its connectivity.

3.6 Simple proof of the PCP theorem

The PCP theorem says that given a boolean formula ϕ in conjunctive normal form, we can efficiently transform it into another formula ϕ' such that if ϕ is satisfiable, then so is ϕ', and if ϕ is not satisfiable, then not only is ϕ' not satisfiable, but no assignment can satisfy more than half of the clauses of ϕ'.

To see that this is connected with probabilistically checkable proofs, think of these formulas as statements (say, some whether some graph is 3-colorable). Now note that a satisfying assignment to the formula constitutes a proof of that statement. So now we have a formula ϕ' with the property that either there exists an assignment that satisfies all the clauses, or no assignment can satisfy more than half of the clauses. If somebody gives you a purported proof, you can be convinced with high confidence by just checking a few clauses uniformly at random.

This is one of the most important results in computational complexity, not only because the result itself is quite an achievement, but also because it is the basis for many inapproximability results. For example, consider the clique problem. We know that solving it exactly is $\mathsf{NP\text{-}hard}$, but that doesn't tell us anything about how closely we can approximate a solution in polynomial time. However, it can be shown (using the PCP theorem) that under reasonable conjecture, we cannot get a nontrivial approximation for clique in polynomial time.

The PCP theorem is quite involved. We will see a transformation based on expanders that iteratively boosts the unsatisfiability gap, resulting in a significantly simpler proof.

4 Lower Bounds

4.1 Metric embeddings

One lower bound that we will discuss is metric embeddings. A metric embedding is a mapping from a discrete metric (a distance function on a finite space that satisfies the triangle inequality) to Euclidean space. We require that every point in the original space (say, the vertices of a graph where distance is taken to be the shortest path) maps to a point in Euclidean space, such that the distances are roughly preserved. One important result in metric embeddings is that for any finite metric space with n points, an embedding in Euclidean space can be constructed such that the largest factor that distances can be expanded or contracted is logarithmic. It turns out that expander graphs can be used to show that this is the best possible. If you want to embed the shortest distance metric from an expander graph into Euclidean space with a bounded number of dimensions, you cannot do better than logarithmic distortion.

4.2 Proof complexity

A number of applications exist in proof complexity. In this area, the sizes of proofs for tautologies are investigated. In other words, a propositional formula is considered that is true under all possible assignments, and we would like to find proofs for that tautology that are as short as possible (polynomial in the length of the formula). If there is such a proof system, then this is equivalent to NP being closed under complement. Progress is attempted on $\mathsf{NP} \neq \mathsf{coNP}$ by considering several candidate proof systems (theorem solvers, in practice), and prove that for these particular systems, there are tautologies that have no short proofs. Some such proofs are based on expander graphs.

5 More on Expanders

We briefly mention the spectral characterization of expander graphs, which will be described in detail next lecture. Given a d-regular graph's normalized adjacency matrix, we consider the spectrum of that matrix. It turns out that d-regular expanders are d-regular graphs with a constant gap between the largest and second largest eigenvalue. This characterization forms the basis for many of the upper bound applications.

We will also look at constructions of expanders in the next lecture. We will prove that expander graphs exist, and in that proof we will use a probabilistic method, where if we pick a random graph with certain parameters, then with high probability it will be an expander graph. Note that given a graph, determining whether it is an expander is actually $\mathsf{coNP\text{-}hard}$. Also, since in many of the applications the expander graph will be quite large, we often cannot construct the graph explicitly. Rather, we want a procedure that efficiently tells us what the neighbors are for a given vertex.