Expanders Lecture Notes

William Blair

Wednesday 20th November, 2013

1 Probabilistic Algorithms

Suppose we are given an algorithm $\mathcal{A}(r, x)$ that decides whether $x \in L$ for some $L \subseteq \{0,1\}^*$. Assume r is a random string chosen uniformly at random from $\{0,1\}^n$. Let $\mathcal{A}(r,x) = 1$ mean $\mathcal{A}$ claims that $x \in L$ and 0 otherwise. $\mathcal{A}$ has the following characteristics

$$Pr\left(\mathcal{A}(r,x) = 1 | x \notin L\right) \leq \frac{1}{3}$$
$$Pr\left(\mathcal{A}(r,x) = 0 | x \in L\right) \leq \frac{1}{3}$$

That is, $\mathcal{A}$ gives an incorrect answer with probability at most 1/3 independent of the input.

1.1 (Q). If error is two sided, how can we reduce the likelihood that $\mathcal{A}$ is incorrect using repetition?

Let us run $\mathcal{A}$ k times and take the majority answer as $\mathcal{A}$'s decision. This yields

$$\mathcal{A}(r_1, x), \mathcal{A}(r_2, x), \ldots, \mathcal{A}(r_k, x)$$

Using the Chernoff bound, this will be correct with probability $\frac{1}{2^{O(k)}}$. Let $X_1, X_2, \ldots, X_k$ be defined as follows

$$X_i = \begin{cases} 1, & \mathcal{A}(r_i, x) \text{ is correct} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

$$Pr(\text{majority incorrect}) = Pr\left(S = \sum_i^k X_i \leq \frac{k}{2}\right)$$

This is just a sum of k independent Bernoulli trials where the probability of success is 2/3. Applying the Chernoff bound gives us the following

$$Pr\left(S \leq \frac{k}{2}\right) \leq e^{-\frac{1}{2}k(p-\frac{1}{2})^2} < 2^{-O(k)}$$

Chernoff is useful here because it gives an exponentially decreasing bound on tail distributions of the sums of independent variables. This is definitely useful, but it only works because each r_i is independent.

1.2 (Q). How many bits did we need to achieve this bound? Why should we improve this?

$$O(nk)$$

Obtaining truly random bits in practice is costly and error prone. If we want an error bound of, say 2^{-10}, we need 10x the random bits needed for one run of $\mathcal{A}$. We also need to ensure each bit is chosen uniformly at random. This may not be feasible if n is large.

Recall the $S - T$ connectivity problem where our algorithm chose a starting node uniformly at random. For very large graphs, it may be cumbersome to repeatedly choose truly random initial vertices.

2 Random Walks for Success Amplification

Algorithm Sketch

We can achieve a probability of error of $2^{-O(k)}$ using only $n+O(k)$ random bits by performing a random walk over a special kind of graph. In this method, we use a graph traversal to fetch all k random strings needed in the previous section with the advantage of using less random bits but maintaining the same success bound.

Consider the following algorithm that runs on a graph $G = (V, E)$. The following algorithm jumps to a random vertex in this graph, and defines the strings r_i for $\mathcal{A}$ as nodes it encounters along a random walk of the graph.

```
$v \overset{R}{\leftarrow} V$;
for $k$ times do
    for $\beta$ times do
        $u \overset{R}{\leftarrow}$ neighbors($v$);
        go to $u$;
        $v := u$;
    end
    $r_i := v$;
end
```

Algorithm 1: Collecting k random strings via random walks

2.1 (Q). What are some necessary properties of the graph G in order for this method to amplify the success probability of $\mathcal{A}$?

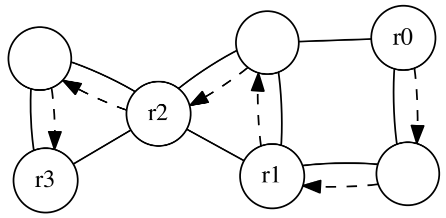

Figure 1: Running our Algorithm on a graph with $\beta = 2$

Graph Properties

- Each node should have bounded degree.
- The graph must have 2^n vertices so all possible random strings may be considered.
- Each r_i must be random. After β steps in the random walk, we should be at any node with uniform probability.

Space Usage

We need n bits to start at a vertex chosen uniformly at random. We then need $O(k)$ bits to perform the random walk. If the degree of every vertex is bounded by some constant d, we just need $log(d)$ bits at each steps to select a neighbor at random. The total number of bits needed for generating k random strings using this method is

$$n + log(d)\beta k = n + O(k)$$

which is a good improvement over the naive approach we first considered. We now introduce special kinds of graphs called expanders which enable us to realize this space bound.

3 Expander Graphs

definition

Let an expander be a connected, bipartite, multi-graph $G(X, Y, E)$ where $|X| = |Y| = \frac{n}{2}$. Recall that if G is connected then every subset $S \subseteq X$ has at least one neighbor outside of S. Every expander is given by a triple (n, d, c)

- n - number of vertices in the graph
- d - the degree of every vertex v. Such a graph is called d-regular.
- c - the measure of connectedness in the graph

The last parameter is important, but requires a bit of explanation. For any subset $S \subseteq X$, let

$$\Gamma(S) = \text{Vertices that share an edge with S}$$

or in terms of sets of vertices

$$\Gamma(S) = \{v \in Y | u \in S \text{ s.t. } (u, v) \in E\}$$

We can now define a lower bound for $\Gamma(S)$ for any $S \subseteq X$ in terms of c.

$$|\Gamma(S)| \geq \left(1 + c\left(1 - \frac{2|S|}{n}\right)\right)|S|$$

Where $1 \geq c \geq 0$. Thus, the connectedness parameter c controls how many neighbors every possible subset S must have. If c is small, then $|\Gamma(S)| \approx |S|$. Likewise, if c is closer to 1 then $|\Gamma(S)| > |S|$. As an example, consider the following subset S of an $(n, d, 1/4)$ expander.

$$|S| = \frac{n}{3}$$

$$|\Gamma(S)| = \left(1 + \frac{1}{4}\left(1 - \frac{2(n/3)}{3}\right)\right)|S|$$

$$|\Gamma(S)| = \left(1 + \frac{1}{4}\left(\frac{1}{3}\right)\right)|S|$$

$$|\Gamma(S)| = \left(1 + \frac{1}{12}\right)|S|$$

The consequence of this is that the set of neighbors S' connected to S must be 1/12 larger than S. This makes c an expansion constant for the neighborhood of any subset in the expander.

3.1 (Q). For our purpose of obtaining strings from a random walk over an expander, would we want large or small d? Would we want large or small c. Why?

Recall that d dictates how many random bits we need at each step of our random walk. Since G is d-regular, we'll need $log(d)$ random bits to select the next neighbor. Given our goal to reduce the need for true randomness as much as possible, we would like to minimize d.

The expansion factor c defines the size of the neighborhood of every possible subset in the graph. The larger the neighborhood, the more vertices we can reach in just a few steps of a random walk. The randomness requirement for our algorithm necessitates a larger value of c as we want to reach any nodes with uniform probability in constant steps.

We will later define this more formally, but first we need to relate the expansion factor c to the eigenvalues of the graph G.

Eigenvalues and Expansion Factor

Since any expander G is a bipartite graph, its adjacency matrix is represented by the following $n \times n$ matrix.

$$A(G) = \begin{bmatrix} 0 & B \\ B^T & 0 \end{bmatrix}$$

Where 0 a matrix of zeros and $B_{i,j}$ gives the number of edges between vetices i and j in G. Since A is a symmetric matrix its n eigenvalues have the following property

$$\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$$

and since G is a connected graph its first eigenvalue is greater than the second. Since G is d-regular, $\lambda_1 = d$ and so $\lambda_2 < d$. If we can bound λ_2 by our expansion factor c, we can relate the expansion factor of G to the Markov Chain that represents a random walk over it. This will allow us to prove our bit generation algorithm from a random walk actually produces random bits. The following is the bound of λ_2 for a (n, d, c) expander.

$$|\lambda_2| \leq d - \frac{c^2}{1024 + 2c^2}$$

Constructing Expanders

Now that we have outlined what makes a graph an expander, we provide an explicit construction of one. Suppose for an arbitrary m, the size of our graph is $n = 2m^2$. We can define a $(n, 5, \alpha)$ expander as follows using a Gabber-Galil construction.

1. Pick an arbitrary integer m and let our graph have size $n = 2m^2$
2. Create m^2 vertices in X. Give each a label (x, y) where $a, b \in \mathbb{Z}_m$. Do the same for Y.
3. For each vertex (x, y) in X, add an edge between it and the following edges in Y.

$$\{(x, y), (x, x + y), (x, x + y + 1), (x + y, y), (x + y + 1, y)\}$$

   where addition is all done modulo m.

3.2 (Q). How does this solve our problem of needing an exponentially large graph to represent all possible strings?

This definition allows us to do a random walk on the graph from a random start point without having to explicitly store the entire graph. Starting at some node v, we can just compute its 5 nearest neighbors and select the next node randomly from them.

4 Random Walks on Expanders

4.1 (Q). How can we transform a (n, d, c)-expander's adjacency matrix,$A(G)$, into a Markov Chain?

$$P = A(G)/d$$

$$P_{u,v} = Pr(X_{n+1} = v|X_n = u)$$

Observe that all eigenvalues of P are now

$$1 \geq \lambda_i \geq -1$$

as we scaled down the matrix by d.

4.2 (Q). What's the problem with performing a random walk on an expander G if we want a uniform stationary distribution?

So far we've defined expanders as bipartite graphs, and all bipartite graphs are periodic. That is, for any state i, let its period be

$$k = gcd\{n : \; Pr(X_n = i|X_0 = i) > 0\}$$

The period of i is the greatest common divisor of the steps needed to reach i from i. For any vertex i in a bipartite graph, this must be even since each step brings us to a node in the opposite partition. The period is greater than 1 as no node has an edge with itself.

A periodic Markov Chain does not have a unique stationary distribution. This implies that if we perform a random walk on G, we are not guaranteed to hit the uniform distribution. Therefore, the strings we collect to be random input to $\mathcal{A}$ will not be random.

4.3 (Q). How can we modify P so that it is periodic?

We can reduce all transition probabilities by 1/2. Next, add a self loop at each vertex with transition probability 1/2. This defines a new markov chain as follows

$$Q = (I + P)/2$$

Q is double stochastic, and has a unique stationary distribution π that is equal to the uniform distribution.

$$\pi = \pi Q$$

4.4 (Q). Why is π important?

Once we reach π, each r_i we walk to will be selected uniformly at random. Now, the last question we need to answer is, "how quickly do we converge to π?"

Let $q^{(t)}$ be the state probability vector for our Markov chain at step t from the initial distribution $q^{(0)}$. $q_i^{(t)}$ gives us the probability of being at state i at step t. Define the relative pointwise distance of the chain at step t as follows

$$\Delta(t) = \max_i \frac{|q_i^{(t)} - \pi_i|}{\pi_i}$$

This measures the distance of our state probability vector at any step from the stationary distribution. We use the state i that is furthest from its stationary value as the measure. Note, this is independent of where we start in the graph.

For our Q, we can bound $\Delta(t)$ as follows.

$$\Delta(t) \leq n^{1.5}(\lambda_2)^t$$

4.5 (Q). Recall that in Q, the eigenvalues range from 1 to -1 where $\lambda_1 = 1$ and $\lambda_2 < \lambda_1$. Why is the bound of $\Delta(t)$ here important?

Since $\lambda_2 < 1$, $\Delta(t)$ converges exponentially to 0. No matter where we start in the graph, we'll reach the uniform distribution very quickly. In fact, for any constant $0 < \delta < 1$, we only need to take $O(log(n))$ steps before the relative pairwise distance at time t falls below δ.

5 Final Algorithm

We have all the pieces now to make our final random string generator. Let us modify the decision algorithm $\mathcal{A}$ for the language L and make it a little more accurate.

$$P(A(r,x) = 1 | x \in L) \leq \frac{1}{100}$$

$$P(A(r,x) = 0 | x \notin L) \leq \frac{1}{100}$$

Where $|r| = n$. Assume n is odd. If it is not, we can always add an additional bit to make it so. Use the Gabber-Gahlil expander technique mentioned before to make a $(N, 7, 2\alpha)$ expander G where

$$m = 2^{\frac{(n-1)}{2}}$$

$$N = 2m^2 = 2^n$$

Label each vertex with a unique id from $\{0,1\}^n$ in any order. Let A be the adjacency matrix for G and let the Markov Chain of the random walk be defined as follows

$$Q = (I + A/7)/2$$

We begin the random walk at a vertex v selected uniformly at random. Pick a positive integer β such that

$$\lambda_2^\beta \leq \frac{1}{10}$$

The choice of β is very important. If we satisfy the above inequality, we will satisfy the property that each string we choose will be selected uniformly at random.

Now, we run the random walk algorithm we defined at the beginning of the lecture on G with a slight modification. We walk $7k\beta$ random steps starting from v. Every β steps, stop and sample the current node as $r_{i\beta}$. Let $X_{i\beta}$ be that state in the Markov chain Q. We then run $\mathcal{A}(r_{i\beta}, x)$ to get $7k$ answers and take the majority result as our decision

5.1 (Q). How many steps are there total? How many random bits are used?

This algorithm takes $7k\beta$ random steps and uses $n + \log(7) \cdot 7k\beta$ random bits.

We will now end by showing the following

$$Pr(\text{incorrect decision}) < \frac{1}{2^k}$$

5.2 (Q). What problem can you see with this approach?

β is a constant, and we previously showed that we need $O(log(n))$ steps to reach the uniform stationary distribution π. Observe that we start at the stationary distribution by selecting a beginning vertex at random. This will help us achieve the error bound we want.

Let the probability distribution vector for the string $r_i = X_i\beta$ be $p^{(i)}$ for $i = 1, 2, \ldots, 7k$. Let $B = Q^\beta$ be the state probability matrix after β random walks.

$$p^{(i)} = p^{(0)} B^i$$

where $p^{(0)}$ is the uniform distribution. Let $\mathcal{W}$ be the set of witnesses for x.

$$\mathcal{W} = \{r \in \{0,1\}^n | A(r,x) = L(x)\}$$
$$\overline{\mathcal{W}} = \{r \in \{0,1\}^n | A(r,x) \neq L(x)\}$$

From the definition of how correct $\mathcal{A}$ is we have

$$|\mathcal{W}| \geq 0.99N$$
$$|\overline{\mathcal{W}}| \leq 0.01N$$

Define W as $N \times N$ diagonal matrix s.t. $W_{i,i} = 1$ if the string r_i in G is a witness. Likewise, $\overline{W} = I - W$ defines the strings that are non-witnesses or cause $\mathcal{A}$ to answer incorrectly. From our previous definitions we have the following

$$Pr(r_i \text{ is a witness}) = ||p^{(i)} W||_1$$

$$Pr(r_i \text{ is a non-witness}) = ||p^{(i)}\overline{W}||_1$$

Consider our random-walk's output is $7k$ "random" strings.

$$r_1, r_2, \ldots, r_{7k}$$

where each r_i is either a witness or a non-witness. Consider the corresponding event sequence of matrices

$$S = \{S_1, S_2, \ldots, S_{7k}\} \in \{W, \overline{W}\}^{7k} \text{ s.t. } S_i = W \text{ iff } r_i \in \mathcal{W}$$

5.3 (Q). How can we define an event sequence S to mean we make an error?

The event sequences we are interested in are those where $\overline{W}$ is the majority of $\mathcal{S}$. These correspond to finding more non-witnesses than witnesses. We will use the following theorems to bound the probability of this event occurring. For any $\mathcal{S}$

$$Pr(\mathcal{S} \text{ occurs}) = ||p^{(0)}(BS_1) \cdot (BS_2) \cdot \ldots \cdot (BS_{7k})||_1$$

and for any probability state vector $p \in \mathbb{R}^N$

$$||pBW|| \leq ||p||$$

$$||pB\overline{W}|| \leq \frac{1}{5}||p||$$

$$Pr(\text{majority incorrect}) \leq \sqrt{N}||p^{(0)}(BS_1) \cdot (BS_2) \cdot \ldots \cdot (BS_{7k})||$$

In order for an event sequence to be incorrect, at least $\mathcal{K} = 7k/2$ of the events must produce non-witnesses to achieve a majority. We can apply the appropriate theorem above $\mathcal{K}$ times to get

$$Pr(\text{majority incorrect}) \leq \sqrt{N}\left(\frac{1}{5}\right)^{\mathcal{K}}||p^{(0)}||$$

$$Pr(\text{majority incorrect}) \leq \sqrt{N}\left(\frac{1}{5}\right)^{\frac{7k}{2}}||p^{(0)}||$$

Since we defined $p^{(0)}$ as the uniform distribution, its L_2 norm is simply $\frac{1}{\sqrt{N}}$. Suppose we over-estimate that the number of event sequences with a majority of non-witnesses $\mathcal{S}$ is 2^{7k}. Combining these two observations we have

$$Pr(\text{majority incorrect}) \leq 2^{7k}\sqrt{N}\left(\frac{1}{5}\right)^{\frac{7k}{2}}\frac{1}{\sqrt{N}}$$

$$Pr(\text{majority incorrect}) \leq \frac{2^{7k}}{5^{7k/2}}$$

$$Pr(\text{majority incorrect}) < \frac{1}{2^k}$$

Which gives us the error bound exponentially small with respect to k using only $n + O(k)$ random bits.

6 References

The randomized algorithm $\mathcal{A}$ we mentioned in the first section is a decision algorithm for a language L in the complexity class BPP. A more in depth description of BPP and other classes such as RP, $Co - RP$, and ZPP may be found in Chapter 7 of Arora and Barak.

For in depth proofs of theorems stated in these notes, along with a description of the random walk algorithm, see Section 6.7 in Motwani. More details on Expanders and their properties can be found in Chapter 21 of Arora and Barak.

References

[1] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, 2009.

[2] Rajeev Motwani. *Randomized Algorithms.* Cambridge University Press, 1995.