15.859-Z Algorithmic Superpower Randomization October 22, 2015

Lecture 14

Lecturer: Bernhard Haeupler *Scribe: Alnur Ali*

1 Recap of the Lovasz Local Lemma (LLL)

Let us first recap what the Lovasz Local Lemma (LLL) states.

Suppose we have a set of m "bad" events (*i.e.*, events we wish would *not* occur) $\mathcal{A} = \{A_1, \ldots, A_m\}$, and a (valid) dependency graph G for $\mathcal{A}$. (Recall that we define a dependency graph as follows. Suppose we have a (undirected) graph $G = (V, E)$, where the set of vertices $V = \mathcal{A}$, and E is some edge set. We say G is a dependency graph for $\mathcal{A}$ if nonadjacent vertices/events are *mutually independent* $\implies$ no edge between vertices/events (see the Lecture 13 notes for a definition of mutual independence).)

Suppose further that

$$\sum_{A_i' \in \Gamma^+(A_i)} \mathbf{Prob}\, A_i' < 1/e \quad \text{for } i = 1, \ldots, m, \tag{1}$$

where $\Gamma^+(Z)$ denotes the set of neighbors of vertex Z including Z. Define $p_i = \mathbf{Prob}\, A_i,\ i = 1, \ldots, m$.

Then, the LLL concludes that

$$\mathbf{Prob}\left(\wedge_{i=1}^m \bar{A}_i\right) > \prod_{i=1}^m (1 - ep_i) > 0, \tag{2}$$

i.e., the probability that none of the bad events happens is nonzero.

Alternatively, instead of (1), we can require that

$$(\Delta(G) + 1) \max_{i=1,\ldots,m} p_i < 1/e, \tag{3}$$

where $\Delta(G)$ is the maximum degree of G, and reach the same conclusion.

2 Application to hypergraph coloring

Here, we will apply the LLL to show that there exists a (valid vertex) *coloring* for a *k-uniform hypergraph*; first we define these terms.

A graph is an ordered pair consisting of a vertex set and an edge set; the elements of edge set must have cardinality two. A hypergraph is defined in the same way, except that the elements of the edge set for a hypergraph (*i.e.*, a *hyperedge*) may have cardinality ≥ 2, *i.e.*, they may be elements of the power set of the vertex set, *i.e.*, a single edge can connect more than 2 vertices. A k-uniform hypergraph is a hypergraph where each hyperedge has cardinality (exactly) k (thus, a graph is a 2-uniform hypergraph).

A coloring for a graph is an assignment of each vertex to a color (*i.e.*, a label) such that no two adjacent vertices share the same color. A coloring for a hypergraph is an assignment of each vertex to a color such that no hyperedge contains vertices that are all the same color (*i.e.*, no hyperedge is *monochromatic*).

Looking back on the assumptions required for (2), we need a set of events, probabilities for the set of events, and a dependency graph, in order to apply the LLL. Since we want to show that there exists a coloring such that none of the (hyper)edges are monochromatic, let us simply take A_i to be the event that edge i is monochromatic.

Suppose we are dealing with k-uniform hypergraphs, and that there at most c different colors. Then $p_i = c/c^k = 1/c^{k-1}$ (*i.e.*, there are c ways for all k vertices in a hyperedge to share the same color, out of all possible c^k color assignments). For our dependency graph, if two hyperedges share a vertex in the hypergraph then we place an edge between their vertices in the dependency graph (note that this is a valid dependency graph as if hyperedges don't share vertices in the hypergraph, then, intuitiuvely, the events that these hyperedges are monochromatic are independent). Suppose the edges in our hypergraph overlap with at most ℓ other edges (equivalently, our dependency graph has degree at most ℓ); then, by (3), if

$$(\ell + 1)/c^{k-1} < 1/e,$$

then we reach our claim, *i.e.*, that there exists a valid coloring for the k-uniform hypergraph. Equivalently, we only need that

$$\ell < c^{k-1}/e - 1$$

for there to exist a coloring.

3 Application to packet scheduling

Here, we consider another application of the LLL. Let us describe the setup first. We consider n paths connecting sources to destinations in a network, where each path may overlap/intersect other paths (but each path never intersects itself). Roughly, the goal is to send a packet from each source so that it reaches its destination in as little time as possible. Let D be the maximum length of any path,

and C be the maximum number of overlaps/intersections of paths anywhere in the network (*i.e.*, the maximum amount of congestion).

In the best case, each path might never intersect, and so we could hope to take no fewer than $\Omega(D)$ rounds by sending each packet (in parallel) along its path (we will assume that a packet advances by one edge on each round). Neglecting additional constants, we might also hope to do (no better than) $\Omega(C)$, since we always need to wait for the most congested edge to clear up.

We can also do no worse than $O(nD)$, which is achieved by sending each (of n) packet(s) along its path (of length at most D) in series. We can also do no worse than $O(DC)$, *i.e.*, if each edge along the path achieves the maximum congestion. We might hope to achieve $O(C + D)$ since not every edge might be congested.

Since the main issue is (clearly) the congestion, let us consider (randomly) delaying the time at which we release each packet. For simplicity, we can choose a delay uniformly at random. Note, however, that if the delay values are too packed together, then we might run into congestion issues again; if they are too spread apart, then we may end up performing as (bad as) a serial strategy does. Thus, let us simply choose the delay values (uniformly at random) from the set $\{0, \ldots, C\}$.

Under this setup, we first show that using the Chernoff (and union) bound we can derive an algorithm that attains $O(C + D)\Omega(\log n)$; then we show using the LLL that there exists an algorithm that can do better.

3.1 First attempt: use the Chernoff (and union) bound

Fix an edge e and a time step t. Let $C_{e,t}$ be the random variable modeling the number of packets that would like to cross through edge e at time step t, *i.e.*, the amount of congestion. Then

$$\mathbf{E}\, C_{e,t} \leq C \cdot 1/C = 1,$$

which makes getting a high probability result difficult. Instead, consider applying the Chernoff bound to the event $C_{e,t} > k \log n > \mathbf{E}\, C_{e,t}$, for some $k > 0$ (say, $k = O(CD)$) (see the Lecture 4 notes for more background on this approach), and get

$$\mathbf{Prob}(C_{e,t} \geq k \log n) \leq \exp(-k \log n) = 1/n^k.$$

Applying a union bound, *i.e.*, summing over all possible edges and time steps (m times for, say, $m = O(C + D)$, as discussed at the beginning of this section when working out the runtime upper bounds), we get that the probability that there exists a congestion along any edge in the graph at any time step that exceeds $k \log n$ is at most $1/n^{k-m}$. Thus, we can conclude that the congestion

anywhere in the graph is at most $\Omega(\log n)$, with high probability.

This leads us to the following algorithm. On each time step, introduce $\Omega(\log n)$ additional time steps to allow each edge to resolve its congestion. The running time of this algorithm is $O(C + D)\Omega(\log n)$, as claimed. However, if $n \to \infty$, then we may want to pursue another algorithm/approach.

3.2 Second attempt: use the Lovasz Local Lemma

Now we turn to the LLL. Let $A_{e,t}$ be the event that $C_{e,t}$ exceeds some constant q, say, $q = 100 \log(CD)$. From the arguments above (*i.e.*, the Chernoff bound), we have that

$$p_i \leq \exp(-20 \log(CD)) = 1/(CD)^{20}.$$

For our dependency graph, we consider the following rule. If there is a path in our network containing both edges e and e', then $A_{e,t}$ and $A_{e',t'}$ are connected in our dependency graph. This dependency graph has

$$\Delta \leq CD \cdot CD = (CD)^2,$$

and so, invoking (3), we get

$$(CD)^2 \cdot 1/(CD)^{20} < 1/e,$$

for large enough C, D. Thus, we conclude that there exists a way to delay the packets such that the congestion anwhere in the network is bounded by $100 \log(CD)$.

We sketch how to proceed from here. We can interpret the above result as a new network with congestion $C' = \log(CD)$ and (an appropriately adjusted) maximum path length $D' = 2D$. Now, break each path up into subpaths of length $\log(CD)$, and apply the above reasoning to get (another) new network with congestion $C'' = \log\log(CD)$ and maximum path length $D'' = 4D$. Recursively doing this until the congestion becomes $O(1)$, we get a maximum path length of $O(2^{\log^*(C)} D)$, which gives an overall runtime of $O\left((C + D)2^{\log^*(C+D)}\right)$, which is close to $O(C + D)$, and does not depend on n, as claimed.