# 16 Lecture 16, November 26, 2014

## 16.1 Local lemma application: van der Waerden lower bound

Here is another "inevitability" theorem in combinatorics; as before, the local lemma will provide a counterpoint.

**Theorem 69 (van der Waerden)** *For every integer $k \geq 1$ there is a finite $W(k)$ such that every two-coloring of $\{1, \ldots, W(k)\}$ contains a monochromatic arithmetic sequence of $k$ terms.*

The current upper bound on $W(k)$, due to Gowers, is

$$W(k) \leq \underbrace{2^{2^{2^{2^{2^{k+9}}}}}}_{\text{five two's}}.$$

The gap in our knowledge for this problem is even worse than for the graph Ramsey numbers: the current lower bound, due to Lovász, is $W(k) \geq 2^{k-1}/((k+2)e)$. (But a better bound is known for prime $k$.) First we show an elementary lower bound:

**Theorem 70** $W(k) \geq 2^{(k-1)/2}\sqrt{k-1}$.

**Proof:** Color uniformly iid. The probability of any particular sequence being monochromatic is $2^{1-k}$. The union bound shows that all these events can be avoided if

$$2^{1-k}n(n-1)/(k-1) < 1$$

(count $n$ places the sequence can start, while the step size is bounded by $(n-1)/(k-1)$), which is implied by $n \leq 2^{(k-1)/2}\sqrt{k-1}$. □

Now for the improved bound through the local lemma:

**Theorem 71** $W(k) \geq 2^{k-1}/((k+2)e)$.

**Proof:** Again color uniformly iid. For a dependency graph, connect any two intersecting sequences. The degree of this graph is bounded by

$$(n-1)k^2/(k-1)$$

($k^2$ choices for which elements they have in common, $(n-1)/(k-1)$ possible step sizes). Thus all the bad events can be avoided if

$$2^{1-k} < \frac{1}{e(1+k^2(n-1)/(k-1))},$$

which in turn is implied by the bound in the statement of the lemma.

The improvement here came because a union bound over approximately $n^2/k$ terms was replaced by a smaller factor of about $nk$. □

## 16.2 Heterogeneous events and infinite dependency graphs

There are two generalized forms of the local lemma that come fairly easily.

### 16.2.1 Heterogeneous events

It is not necessary that we use the same bound on $\Pr(B_j)$ for all $j$. Instead, we can allow events of various probabilities. Those which are more likely to occur, must have in-edges from events of smaller total probability. On the other hands less likely events can tolerate more in-edges (as measured by total probability). This is formulated, in a slightly circuitous way, in the following version of the lemma.

**Lemma 72** *Let events $B_j$ and dependency edges $E$ be as before. If there are $x_j < 1$ such that for all $j$,*

$$\Pr(B_j) \leq x_j \prod_{(k,j)\in E} (1 - x_k)$$

*Then*

$$\Pr(\bigcap B_j^c) \geq \prod_j (1 - x_j).$$

The proof is basically the same. Show inductively on $m$ that (for any subcollection of $m$ events and any ordering on them), $\Pr(B_m | \bigcap_{j\leq m-1} B_j^c) \leq x_m$.

### 16.2.2 Infinite dependency graphs

Typically, the restriction that $\mathcal{S}$ is finite can be dropped due to compactness of the probability space. Specifically, suppose that—as in most applications—there is an underlying space of independent rvs $X_k$, $k$ ranging over some index set $U$, and each $X_k$ ranging in some compact topological space $R_k$. Moreover suppose that every one of the bad events $B_j$ is a function of only finitely many of the $X_k$'s, say of $k \in U_j \subset U$, $U_j$ finite. Suppose moreover that each $B_j$ is an *open set* in $\prod_{k\in U_j} R_k$. Then each $B_j^c$ is a closed set in the product topology on $\prod_{k\in U} R_k$. Since the product topology is itself compact by Tychonoff's theorem, it satisfies the Finite Intersection Property: a collection of closed sets of which any finite subcollection has nonempty intersection, has nonempty intersection. Consequently, under the additional topological assumptions made here—which are trivially satisfied if each $X_k$ takes on only finitely many values—the supposition in the local lemma (in either the formulation 63 or 72) that $S$ is finite may be dropped.

## 16.3 Moser-Tardos "branching process" algorithm for the local lemma

Now we describe an algorithm for finding satisfying assignments to the local lemma. The algorithm works in great generality and achieves the same limiting threshold (whenever the algorithm is applicable) as the full local lemma; however, for simplicity, we will describe it here in a slightly more restricted setting. (Most notably we'll have no asymmetry between events.)

Let $H = (V, E)$ be a SAT instance. (We can encode most applications of the local lemma in these terms.) That is, $V$ is a set of boolean variables; a *literal* is a variable $v \in V$ or its negation. $E$ is a collection of *clauses*, each $T \in E$ being a set of literals, which is satisfied if at least one of them is satisfied. $H$ is satisfied if *all* $T \in E$ are satisfied. Say that two clauses are neighbors in the dependency graph if they share any variable (not necessarily literal).

We have from last time a corollary of the local lemma:

**Corollary 73** *Suppose every clause in $T \in E$ has size $k$ and has at most $d$ neighbors. If $d + 1 \leq 2^k/e$ then $H$ is satisfiable.*

In the following pseudo-code, two clauses are *neighbors* if they share a common variable (not necessarily common literal). A clause is its own neighbor.

*Moser-Tardos Algorithm [52, 53]*
*Pick* a random assignment to $V$
*While* there is an unsatisfied clause, pick any such clause $T$ and run Fix$(T)$.

*Fix*$(T)$
*Recolor* the variables of $T$ u.a.r.
*While* $T$ has an unsatisfied neighbor, pick any such neighbor $T'$ and run Fix$(T')$.

There is some ambiguity in both of the above procedures as to the order in which unsatisfied clauses are attended to. In the analysis below the ambiguity can be resolved in any deterministic manner (even depending on the history of the algorithm so far). E.g., for simplicity one might choose among available clauses in lexicographic order. Observe that Fix essentially implements a DFS.

We suppose that $|V| = n$ (number of variables), $|E| = m$ (number of clauses).

**Theorem 74** *If $8d \leq 2^k$ then the Moser-Tardos algorithm finds a satisfying assignment to $H$ in time $\tilde{O}(n + mk)$.*

We are not being careful in this presentation about two things. One is the leading constant of "8". This can be improved (without changing the algorithm) and we will point out where that comes up in the analysis.

The second is in the run-time details. The above bound applies to the number of random bits the algorithm uses. The actual run-time, which includes all the necessary data structure management, will be a little be a little larger but only by some factor of about $\log nm$.

Before presenting the proof, let's see why what we are studying is very similar to a branching process. Fixing some clause as the root, there is an implicit tree extending out first to neighboring clauses, then to neighbors of those, and so on. (Of course there may be repetition but that works out in our favor.) The degree of this tree is $d + 1$, but our DFS needs to explore only a subtree of it, generated at random, in which the expected number of children of a node is bounded by $(d + 1)/2^k < 1$. So, intuitively, what is going on is that a Fix call that is initiated by the main procedure, tends to terminate after generating a finite DFS tree.

This is only of course intuition, and the formal proof follows.

**Proof:** The algorithm is implemented with the aid of an (infinite) random string $z = z_1 \ldots$. The first $n$ bits are used for the initial assignment. Then, successive bits are used in batches of $k$ for the Fix procedure.

The choice of $z$ amounts to uniformly choosing a path down a non-degenerate binary tree (no vertices with one child), whose leaves represent successful terminations of the algorithm. (Note, this is the tree of random bits, and we only descend in it—it is not the tree in which we are performing DFS!) Of course the tree is infinite (we might continually "Fix" bits badly.) However, we will argue that with high probability we reach a leaf fairly soon.

If Fix$(T)$ terminates, then $T$ is satisfied. (Otherwise we never "pop" it off the DFS stack.) And, so is every clause that was satisfied before the call to Fix$(T)$. (If a clause $T'$ was made unsatisfied during Fix$(T)$ then that can only be because it was a neighbor of some clause we were running Fix on, hence, Fix$(T')$ will be reached by our DFS traversal. So we'll satisfy $T'$ by the time Fix$(T')$ terminates.)

Hence, the main procedure calls Fix at most $m$ times.

Let $N_t$ be the number of nodes that the algorithm tree has at depth $t$. Since the algorithm always runs the first $n$ steps and then operates in batches of $k$ bits, leaves of the tree occur at the levels $n + sk$, after $s$ calls to Fix (whether from the main procedure or recursively from within Fix).

For any such node which actually exists in the tree, what is the probability that the algorithm reaches it? Since all random seeds $z$ are equally likely, this probability is precisely $2^{-n-sk}$.

So what is the expected runtime of the tree? It is the sum of the probabilities that we reach each node. Namely, $\sum_{t \geq 0} N_t 2^{-t} = n + k \sum_{s \geq 1} 2^{-n-sk} N_{n+sk}$. You can see that if we had, say, $N_{n+sk} = 2^{n+sk}$, this sum would diverge, as of course it should, since then the tree has no leaves at all. But even if there are some leaves, the sum can readily diverge. We need to show the tree is actually "thin" enough that the sum converges.

The key to the proof is to bound in a clever way the number of random bits we need to provide, in order to specify the first $n + sk$ bits of $z$. Our encoding will be concise enough to imply that $N_{n+sk}$ is small.

The obvious way to specify a node at depth $n + sk$ is to use the first $n + sk$ bits of $z$.

But here is another way. Suppose that rather than being told all the bits of $z$, we're instead told, in order, the arguments (names of clauses) to Fix; plus the assignment to all the variables at the time we reach the current node (the one at depth $n + sk$).

Then we can determine the identity of that current node by "working backwards." The last clause, before its recoloring, had to have been in its unique unsatisfied assignment. In turn, the penultimate clause, before its recoloring, had to have been in its unique unsatisfied assignment. And so forth.

How many bits are required to specify the node at depth $n + sk$ in this alternative way?

1. $n$ bits for the last assignment.
2. Each call to Fix is made either from the main procedure or recursively. We can (although this is not the most efficient encoding if one wants to push the limits of the "$2^k/3$" factor in the lemma) spend one bit per call to specify which of these types it is, and then an amount that depends on the type, as follows:
   (a) Each call to Fix from the main procedure can be specified with $\lceil \lg m \rceil$ bits. However, since we potentially may make all of these calls, and since the calls will occur in a pre-specified order (say, lexicographically), it is actually more efficient to provide a binary vector of length $m$ which specifies for each clause whether or not it, in its turn, is called from the main procedure.
   (b) Each call to Fix from within Fix (i.e., by DFS) takes $\lceil \lg d \rceil \leq 1 + \lg d$ bits.

So,

$$\lg N_{n+sk} \leq \min\{n + sk, n + s + m + s(1 + \lg d)\}$$

Now, as above measuring runtime in terms of how many bits of $z$ we read, we have,

$$\begin{aligned} E(\text{runtime}) &= n + k \sum_{s \geq 1} 2^{-n-sk} N_{n+sk} \\ &\leq n + k \sum_{s \geq 1} 2^{-n-sk} 2^{n + \min\{sk, m + s(2 + \lg d)\}} \\ &= n + k \sum_{s \geq 1} 2^{\min\{0, m + s(2 + \lg d - k)\}} \end{aligned}$$

Since we have assumed $k \geq 3 + \lg d$, the first $m$ summands here are simply 1; the sum of all remaining terms is again 1. So,

$$E(\text{runtime}) \leq n + k(m + 1).$$

(The analysis may be tightened by encoding the DFS slightly more efficiently.) □