CS265, Winter 2025

Class 12: Agenda and Questions

1 Announcements

- HW5 out now! (Due 2/28)

2 Questions?

Any questions from the minilectures and/or the quiz? (Constructive LLL)

3 You prove the constructive LLL for another problem!

In the mini-lectures we proved the constructive LLL for the special case of SAT. Now we'll do it for another problem. (Of course, it can be done in general, but two examples basically makes a theorem...)

Consider the following problem (which has featured on a quiz). You are coloring the integers $\{1, \ldots, n\}$ either blue or red. You are given as input a collection of sets $S_1, S_2, \ldots, S_m \subseteq \{1, \ldots, n\}$, so that:

- Each set S_i has size at most k.
- Each set S_i intersects at most d other sets S_j, for some $d > 1$.

Our goal is to color the points $\{1, \ldots, n\}$ so that there is no monochromatic set S_i. (A set S_i is *monochromatic* if every element of it is either red or blue).

Group Work

1. Mimic the proof of the constructive LLL that we saw for k-SAT to give a randomized algorithm that does the following.

   Suppose that $k \geq \log_2 d + 10000$. (Here, 10000 is a stand-in for "some big enough constant.") Then there is a randomized algorithm that proceeds by re-randomizing the sets S_i, (that is, it will iteratively look at different sets S_i and randomly re-color all of the points in that set), so that:
   - If the algorithm terminates, then all of the numbers $\{1, \ldots, n\}$ will be colored so that there is no monochromatic set S_j.
   - The expected number of times that the algorithm re-randomizes a set S_j is poly(m).

Don't worry about giving a complete proof with all the details, just work it out with enough detail that you believe it. As we did in the minilecture video, you may use the (informal) fact that "there is no function $f : \{0,1\}^X \to \{0,1\}^Y$ so that (a) $Y \ll X$ and (b) with high probability over a uniformly random $x \in \{0,1\}^X$, it is possible to recover x given $f(x)$."

***Hint**: It's not* quite *as straightforward as relabeling "clauses" ← "sets"...why not?*

2. What happens to your proof if the number of possible colors grows from two (blue and red) to some number t? In particular, can you get the same guarantee as above, but under a weaker guarantee (eg, $k \geq$ [something smaller than $\log d + 10000$]).

3. How does the answer that you got in the previous part compare to what Corollary 3 in the lecture notes would give you for this problem?

   As a reminder, that Corollary says:

   > Let V be a finite set of independent random variables. Let $\mathcal{A}$ be a finite set of events determined by the random variables in V. If for all $A \in \mathcal{A}$, $|\Gamma(A)| \leq d + 1$, and $\Pr[A] \leq \frac{1}{e(d+1)}$, then the algorithm from the lecture notes (the general one, not just for k-SAT) will find an assignment to the variables V such that no event of $\mathcal{A}$ occurs. Additionally, the expected number of "re-randomizations" performed by the algorithm is bounded by $O(|\mathcal{A}|/(d+1))$.

4. [**This question is open-ended and may be difficult—think about it after you finish the others if you still have time.**] What happens to your proof if the sets can have variable size? (e.g., if all but a few of them have size k, and a few can be really small? Or if they have average size k? Or....?)

Group Work: Solutions

1. Here is our algorithm with print statements that will help in the proof. We've just copy-pasted the algorithm from the lecture notes (made slightly informal for ease of discussion), but changed all the talk of satisfying formulas to coloring sets. The only change is the line in red: we needed to give more information about what went wrong.

Algorithm 1. FIND ASSIGNMENT AND PRINT STUFF

Given $S_1, \ldots, S_m$:

- *Choose a random assignment σ that assigns a color to each of the numbers $1, \ldots n$.*
- *For each set S_i that is monochromatic under σ:*
  - *print "Running FIX on Set i"*
  - *Run* FIX(i, σ) *on S_i and σ.*
- *Return σ.*

Algorithm 2. FIX(i, σ)

Given $S_1, \ldots, S_m$, i, and σ:

- *If we have run for too long (T re-randomizations), print "I give up; I've got σ" and halt.*
- *Re-randomize the colors of all of the variables in S_i.*
- *Suppose that $S_{i_1}, \ldots, S_{i_{d+1}}$ are the sets that intersect with S_i (including S_i itself).*
- *For $\ell = 1, 2, \ldots, d+1$:*
  - *If S_{i_ℓ} is monochromatic (say all [color], where [color] is either blue or red):*
    - *Print ``Trying to fix the ℓ'th child..."*
    - *Print ``the problem was that it was all [color]''*
    - *Run* FIX(i_ℓ, σ)
- *print "All done, moving back up a level."*

To do the proof, we need to do the following three steps:

A We need to argue that we can recover the random bits that go in from the print statements taht come out.

B We need to count the number of random bits that go in to our algorithm in T re-randomizations (say in one call to FIX)

C We need to count the number of bits that come out in T re-randomizations (in one call to FIX).

Thing A is why we needed to add the extra info about the color. With that, let's see why A is true. Suppose that we are working backwards from the final σ returned. Inductively assume that we know the assignment *after* the t'th re-randomization, and we want to know the assignment σ' *before* that randomization, and we want to know the outcome of the random bits used in the t'th re-randomization. We know which set S_i was re-randomized at the t'th step, since we are using the "fixing ℓ'th child" statements to follow along through the execution tree. Thus, every number that was *not* in S_i didn't change, so we know what colors all of those were in σ'. For the items in S_i, our extra print statement told us that before they were all monochromatic [blue or red], so we know what those should have been in σ' as well,

and we know what the randomizing bits were. Continuing in this way, we can work backwards all the way to the original assignment σ, and we can recover the random bits that were flipped.

For B, the number of random bits we use is $n + Tk$, since there are n bits for the original assignment, and k for each of the T re-randomizations.

For C, the number of bits of print statements that come out are:

- $m(C + \log m)$ at the beginning of highest-level calls to Fix to say which clause we are starting on.
- $T(\log(d+1)+1+C)$ bits for "trying to fix the ℓ'th child because it was red" sorts of calls. (Notice that "because it was red" is only one extra bit of information).
- $n + C$ bits for "I give up, here's σ"

where above C is some constant. Thus, the total is at most

$$m(\log m + C) + T(\log(d + 1) + C) + n + C$$

Now, we put it all together. In order to get a contradiction (because the number of bits out would be way less than the number in), we want

$$n + kT \gg m(\log m + C) + T(\log(d + 1) + C) + n + C.$$

After some algebra, we want

$$m(\log m + C) \ll T(k - \log(d + 1) + C)$$

Thus, provided that $k \geq \log(d+1)+C$ for some constant C, the RHS is a constant, and we win when $T \approx m \log m$. This is what we wanted to show.

2. If the number of colors grows from 2 to t, then we just have to change the algorithm to say "because it was green" or "because it was purple" or whatever. This is $\log t$ bits of communication. Working that into our computations above, we get that we need

$$n \log t + kT \log t \gg m(\log m + C) + T(\log(d + 1) + \log(t) + C) + n \log t + C.$$

aka

$$m(\log m + C) \ll T(k \log t - \log(d + 1) + \log t + C)$$

Thus, we'd need $k \geq \log_t(d + 1) + C$ for some constant C, and again we get $O(m \log m)$ re-randomizations.

3. In the context of Cor. 3, we have $V = \{1, \ldots, n\}$ and $\mathcal{A} = \{A_i\}_{i \in [m]}$ where A_i is the event that S_i is monochromatic. If there are t colors, then the probability that A_i is monochromatic is $t^{-(k-1)}$. Thus, Cor. 3 says we need

$$t^{-(k-1)} \leq 1/(e(d+1)),$$

or that $k \geq \log_t(e(d+1)) + 1$, or that

$$k \leq \log_t(d+1) + C,$$

where $C = 1 + \log_t(e)$. This is roughly the same as what we got above (maybe with a different additive constant).