CSE525: Randomized Algorithms and Probabilistic Analysis May 14, 2013

Lecture 12

Lecturer: Anna Karlin *Scribe: Daniel Yang Li, Noah Siegel*

1 Constructive proof of Lovasz Local Lemma(LLL)

Since the constructive proof of Lovasz Local Lemma is complicated, we use a constructive proof for an application of LLL. Specifically, we will construct a solution for solving SAT. This proof illustrates the key ideas in the constructive proof of LLL.

Theorem 1. *Let S be an arbitrary set of m length k clauses such that the support of each clause intersects at most 2^{k-c} clauses, where c is a sufficiently large constant. Then there exists a satisfying assignment for S.*

1.1 Proof of the theorem

Initialize variables $x_1, x_2, \cdots, x_n$ to random values. While there exists clauses not satisfied, pick an arbitrary clause not satisfied, assign new random values to the variables in it.

Claim 2. *This process terminates in polynomial time with high probability.*

We imagine that there is a table R with n columns and an infinite number of rows. The columns are indexed by the variables $x_1, x_2, \cdots, x_n$. In the beginning, the table is filled with random and independent 0/1 values. After that, when the algorithm need a new random value for a variable, it considers the column corresponding to the variable, and uses the first unused entry in the column.

Consider the sequence of unsatisfied clauses encountered by the algorithm, $C_1, C_2, \cdots$. Assume that C_t is the clause visited in the t-th step. C_t can be viewed as a tree T_t labelled by clauses. It is built from step t to 1. At step i, where $t \geq i \geq 1$, we have the tree T_t^i. If C_i does not intersect any of the clauses in T_t^{i+1}, then $T_t^i = T_t^{i+1}$; otherwise, C_i is appended to the clause C_j deepest in the tree T_t^{i+i} that shares a variable with C_i.

Claim 3. *T_t^1 uniquely determines which locations in R the random values for the variables of each clause are taken from, for all C_i in the tree.*

Proof Simply start from the deepest node in the tree, allocate to the respective clause the values from the respective columns in the first row of R, erase these values and shift the respective columns down by one row and repeat. ■

Observation 4. *If 2 clauses C_i and C_j $(i < j)$ in tree are at the same depth in tree, then they are disjoint.*

Proof Otherwise, C_i would have been placed deeper and been a descendent of C_j. ■

Tree is feasible if all clauses in tree are not satisfied by the corresponding values in R. Then we have

$$Pr(\text{tree } T_t^1 \text{ with } q \text{ clauses is feasible}) = p^q,$$

where $p = 2^{-k}$.

Then we have the following lemma.

Lemma 5. *Given the table R and $q \geq 1$, for the algorithm to run for qm steps, it must be the case that R contains a feasible tree of size at least q.*

Proof For the qm clauses visited, there are at least q of them representing the same clause. All q copies of the same clause must belong to the tree associated with the last appearance of this clause in the sequence of visited clauses. ■

From this lemma, we know that

$$Pr(\text{algorithm runs at least } qm \text{ steps}) \leq Pr(R \text{ contains feasible tree of size at least } q).$$

We also have the following claim.

Claim 6. *The number of legally labelled trees of size q is at most $m\binom{dq}{q-1}$.*

(A tree is legally labelled if all nodes are labelled with clauses, adjacent nodes overlap, and nodes at the same level don't intersect.)

Proof

Fix some ordering of the m clauses. Take some arbitrary legally labelled tree containing q clauses. First, there are m possibilities of the root. Next, we will create a unique data structure to represent this tree. Specifically, we will represent the tree as a boolean vector of length dq, where d is the maximum number of clauses that any clause intersects (bounded by 2^{k-c} by our assumption).

Start with an empty vector. Then, for each clause in the tree (using a postorder traversal; the traversal used doesn't matter, it only matters that it's constant), append a zero vector of length d. The entries of this vector will represent the clause's neighbors; this is always possible, since the number of neighbors of any clause is at most d. Finally, for each clause in the tree, mark all of the entries representing its children as 1. Since each node except the root has exactly 1 parent, our vector contains exactly $q-1$ entries that are 1. Any legal tree can be represented in this way, since we took an arbitrary legally labelled tree, and any such tree corresponds to at most one legal tree, since it fully specifies the clauses, their ordering, and the structure of the tree. The number of legal trees is at most the number of vectors of length dq with exactly $q-1$ 1s, which is $m\binom{dq}{q-1}$. ■

Using Stirling's formula, we know that $m\binom{dq}{q-1} \leq m(de)^q$. So the probability that there is a feasible legal tree of size Q or more is at most

$$m\sum_{q=Q}^{\infty}(de)^q p^q = m(dpe)^Q/(1-dpe).$$

For large enough Q (say, $\Theta(\log m)$), this is $o(1)$. So we know that

$$Pr(\text{number of clauses resampling at least } \Theta(\log m))) = o(1).$$

This implies both that the formula is satisfiable, and that the randomized algorithm will find a satisfying assignment in expected polynomial time.

2 Random-walk based algorithm for 2-SAT

We will construct an algorithm for solving 2-SAT in polynomial time with high probability, where n is the number of variables in our input expression. Our algorithm is as follows:

- First, start with an arbitrary assignment of our n variables.
- While not all clauses are satisfied, repeat the following up to cn^2 times for some constant c:
  - Choose an arbitrary clause that is not satisfied.
  - Pick one of the two variables in that clause uniformly at random and switch its value.

- If we found a satisfying assignment, return it, otherwise return "unsatisfiable."

Theorem 7. *Our algorithm solves 2-SAT time with high probability.*

Proof If there is no satisfying assignment, then clearly our algorithm is correct. Therefore, assume there is some satisfying assignment S. We will bound the probability that our algorithm finds this specific satisfying assignment. We think of the algorithm as a random walk on a line between 0 and n, where our position represents the number of variables on which our current assignment agrees with S. When we change the value of a variable, we either change it from agreement to disagreement or disagreement to agreement, so we either increase or decrease our position on the line by 1. Let X_t denote our position in iteration t of the loop.

Claim 8. $\Pr(X_{t+1} = i+1 | X_t = i) \geq \frac{1}{2})$ *and* $\Pr(X_{t+1} = 1 | X_t = 0) = 1$

Equivalently, during any iteration of the loop, the probability of switching a variable from disagreement with S to agreement with S is at least $\frac{1}{2}$, and this probability is 1 when all variables are in disagreement.

First, we choose an arbitrary clause that is not satisfied. This clause not being satisfied means that one or both of its variables differ from their value in S (or else this clause would be satisfied). If one of its variables differ, then the probability we switch the differing variable is 1/2. If both variables differ, then the probability we switch a differing variable is 1. Therefore, the probability of switching a variable from disagreement with S to agreement is at least 1/2. If all variables are in disagreement, the whatever variable we switch, we are sure to change it from disagreement to agreement.

Claim 9. *In any random walk between* 0 *and* n *where* $\Pr(X_{t+1} = i+1 | X_t = i) \geq \frac{1}{2})$ *and* $\Pr(X_{t+1} = 1 | X_t = 0) = 1$, *we claim that* $E(\textit{earliest } t | X_t = n) \leq n^2$

Consider the pessimistic version where $\Pr(X_{t+1} = i+1 | X_t = i) = \frac{1}{2})$. Let h_j = expected number of steps for this random walk to reach n starting at j. At each j between 0 and n noninclusive, we have a 1/2 chance of moving to h_{j-1}, and a 1/2 chance of moving to h_{j+1}. Therefore,

$$h_j = \frac{1}{2}h_{j-1} + \frac{1}{2}h_{j+1} + 1$$

$$2h_j = h_{j-1} + h_{j+1} + 2$$

$$h_j - h_{j+1} = h_{j-1} - h_j + 2$$

$$h_n = 0$$

$$h_0 - h_1 = 1$$

By induction on j,

$$h_j - h_{j+1} = 2j + 1$$

$$h_0 = h_0 - h_n = \sum_{i=0}^{n-1} h_i - h_{i+1} = \sum_{i=0}^{n-1} (2i+1) = 2\frac{(n-1)n}{2} + n = n^2$$

Therefore, since the expectation is n^2 in the worst case, the expectation is always at most n^2. By the Markov bound,

$$Pr[(\text{earliest } t | X_t = n) > 2n^2] \leq \frac{1}{2}$$

In other words, the probability we fail to find a satisfying assignment in $2n^2$ steps if one exists is at most $1/2$. We run our loops cn^2 times. This is no worse than running the loop for $2n^2$ steps independently $c/2$ times, since in each iteration we're starting from 0, the worst possible starting point in expectation. The probability that all $c/2$ independent trials fail is at most $1/2^{c/2}$. Therefore, the probability that our algorithm fails to find a solution when one exists is at most

$$\frac{1}{2^{c/2}}$$

So if we run the loop 100 times, for example, our probability of failure is at most $\frac{1}{2^{50}}$, a very small probability. Our algorithm runs with high probability of success. ■