Randomized Algorithms 2013A
Lecture 11 – Importance Sampling, and the Local Lemma[*]

Robert Krauthgamer

1 Counting DNF solutions via Importance Sampling

Problem definition: The input is a DNF formula f with m clauses $C_1, \ldots, C_m$ over n variables $x_1, \ldots, x_n$, i.e. $f = \vee_{i=1}^m C_i$ where each C_i is the conjunction of literals like $x_2 \wedge \bar{x}_5 \wedge x_n$.

The goal is the estimate the number of Boolean assignments that satisfy f.

Theorem 1 [Karp and Luby, 1983]: Let $S \subset \{0,1\}^n$ be the set of satisfying assignments. Then $|S|$ can be estimated within factor $1+\varepsilon$ in time that is polynomial in $m + n + 1/\varepsilon$.

1.1 A first attempt

Random assignments: Sample t random assignments, and let Z count how many of them are satsifying. We can estimate $|S|$ by $Z/t \cdot 2^n$.

Formally, let Z_i be an indicator for the event that the i-th sample satisfies f. Then $Z = \frac{1}{t}\sum_i (Z_i \cdot 2^n)$. We can see it is an unbiased estimator:

$$\mathbb{E}[Z \cdot 2^n/t] = \sum_{i=1}^t \mathbb{E}[Z_i] \cdot 2^n/t = |S|.$$

Observe that $\operatorname{Var}(Z) = \frac{1}{t^2}\sum_i \operatorname{Var}(Z_i \cdot 2^n) = \frac{1}{t}\operatorname{Var}(Z_1 \cdot 2^n)$. But even though we can use Chernoff-Hoeffding bounds since Z_i are independent, it's not very effective because the variance could be exponentially large.

Exer: Show that the standard deviation of Z_1 (and thus Z) could be exponentially large.

1.2 A second attempt

Let $S_i \in \{0,1\}^n$ be all the assignments that satisfy the i-th clause, hence $|S_i| = 2^{n-\operatorname{len}(C_i)}$.

[*]These notes summarize the material covered in class, usually skipping proofs, details, examples and so forth, and possibly adding some remarks, or pointers. The exercises are for self-practice and need not be handed in. In the interest of brevity, most references and credits were omitted.

Remark: The naive approach does not use the DNF structure at all. Using this structure, we can write $S = \cup_i S_i$, which can be expanded using the inclusion-exclusion formula, but it would be too complicated to estimate efficiently.

Algorithm E:

1. Choose a clause C_i with probability proportional to $|S_i|$ (namely, $|S_i|/M$ where $M = \sum_i |S_i|$).
2. Choose at random an assignment $a \in S_i$.
3. Compute the number y_a of clauses satisfied by a.
4. Output $Z = \frac{M}{y_a}$.

We proved in class the following two claims.

Claim 1a: $\mathbb{E}[Z] = |S|$.

Claim 1b: $\sigma(Z) \le n \cdot \mathbb{E}[Z]$.

Exer: Show that $|S|$ can be approximated within factor $1 \pm \varepsilon$ with success probability at least 3/4, by averaging $O(m^2/\varepsilon^2)$ independent repetitions of the above.

Exer: Show how increasing the number of repetitions by an $O(\log \frac{1}{\delta})$ factor can improve the success probability to $1 - \delta$.

1.3 Importance sampling

It's a tool to reduce variance when sampling. The idea is to have a more "focused" sample, rather than uniform distribution, and then "factor out" the bias in this sample.

Setup: We want to estimate $z = \sum_{i \in [s]} z_i$ without reading all the z_i values. The main concern is that the z_i are unbounded, and thus most of the contribution might come from a few unknown elements, but we have a "good" lower bound on each element.

Theorem 2 [Importance Sampling]: Let $z = \sum_{i \in [s]} z_i$, and $\lambda \ge 1$. Let $\hat{Z}$ be an estimator computed by sampling a single index $i \in [s]$ with probability p_i and setting $\hat{Z} = z_i/p_i$, where each $p_i \ge \frac{z_i}{\lambda z}$ and $\sum_{i \in [s]} p_i = 1$. Then

$$\mathbb{E}[\hat{Z}] = z \quad \text{and} \quad \sigma(\hat{Z}) \le \sqrt{\lambda}\mathbb{E}[\hat{Z}].$$

Proof: was seen in class.

Exer: Let $z = \sum_{i \in [s]} z_i$ and suppose that for each z_i we already have an estimate within factor $b \ge 1$, i.e., some $z_i \le y_i \le b z_i$. How many samples are needed to compute, with probability at least 3/4, a $1 \pm \varepsilon$ factor estimate for z?

Exer: Explain our DNF counting algorithm above using the importance sampling theorem.

Hint: Assignments a that satisfy no clause are chosen with zero probability.

2 The Local Lemma

Setup: We have many "bad" events, each happening with a small probability, but they are NOT independent. Instead, each event "depends" (in a very strong sense, see more below) only on a few other events.

Example: n events "organized" in a 2d-grid, where dependencies are only between nodes connected by an edge.

Theorem 3 [Lovász Local Lemma, general version, 1975]: Let $A_1, \dots, A_m$ be random events, and for every i, let $\Gamma(A_i)$ be a subset of the events such that A_i is independent, even mutually, of the events $\{A_1, \dots, A_m\} \setminus (\Gamma(A_i) \cup \{A_i\})$. If there are $0 < x_i < 1$ for every $i \in [m]$ such that

$$\Pr[A_i] \le x_i \prod_{j:A_j \in \Gamma(A_i)} (1 - x_j),$$

then $\Pr[\cap_{i\in[m]} \bar{A}_i] \ge \prod_{i\in[m]}(1 - x_i) > 0$.

For comparison, if the events were completely independent, then the probability would be exactly $\prod_i(\Pr[\bar{A}_i])$ which is at least $\prod_i(1 - x_i)$.

Theorem 4 [LLL, symmetric version]: Let $A_1, \dots, A_m$ be random events, each happening with probability p. Assume that every event A_i is independent (mutually) of all but d other events, i.e. $|\Gamma(A_i)| \le d$. If $p(d + 1) \le 1/e$, then $\Pr[\cap_{i\in[m]} \bar{A}_i] > 0$.

Proof: was seen in class using the general version.

Theorem 5: Every k-CNF formula with n variables in which each variable appears in less than $2^k/ek$ clauses is satisfiable.

Proof: was seen in class using the LLL.

3 Algorithmic version of the local lemma

We will see an algorithmic version for the k-CNF problem. This method works for many, but not all, applications of the LLL, sometimes with somewhat worse parameters.

Theorem 6 [Moser-Tardos'09]: Given a k-CNF formula with n variables in which each variable appears in less than $0.92^k/ek$ clauses, a satisfying assignment can be computed in polynomial time.

Algorithm MT:

(1) Pick a random assignment for each variable.

(2) While there are unsatisfied cluases, pick one such clause arbitrarily, and assign *all* its variables new random values.

The algorithm only terminates when a satisfying assignment is found. But how long does it usually take? The analysis will show that with high probability, the algorithm terminates in polynomial time. Recall that each clause is satisfied with probability $p = 2^{-k}$, and that $p(d + 1) < 0.9/e$.

Main idea: For any specific clause that is not satisfied, we can easily fix the assignment to satisfy this clause, in fact flipping any variable would do. It could have been great if we could always *reduce* the number of unsatisfied clauses, but this is not realistic. Instead, we sample it at random, which should succeed with high probability.

Proof: Suppose the random values for the variables are written down in advance, as a table R that has n columns (one for each variable) and r rows. In particular, the first assignment is just the first row in the table.

Denote the clause "fixed" by the algorithm in iteration i by C_i. (They need not be distinct.) Consider clause C_t from iteration t. We can "trace" the execution backwards from iteration t to 1, building along the way a labeled tree T_t. We start (at $i = t$) with the tree being just C_t, which throughout will be the root of the tree. Next, assuming we already traced iterations $t, t-1, \ldots, i+1$, let us consider iteration i.

If C_i does not overlap any clause in the tree, we make no change to the tree, basically ignoring C_i. Otherwise, we find the *deepest* (furthest from root) node C_j whose label overlaps C_i, and add to it a child labeled C_i. The tree T_t is completed after doing $i = 1$.

We saw the rest of the proof in class, using the following three lemmas.

Claim 5a: If two clauses C_i and C_j for $i < j$, are at the same depth in a tree T_t, then they are disjoint.

The completed tree T_t determines uniquely which locations in R are used to set the values of all clauses C_i in the tree. Indeed, just scan the tree from the leaves (deepest nodes) towards the root, and use the claim to "advance" inside R.

Given a table R, we say that a tree T_t is *feasible*, if a (determinisitc) execution of the algorithm can produce T_t. (The tree is not determined uniquely from R because of the choice of t.) In particular, the clauses appearing in the tree must all be unsatisfied.

We denote the number of clauses in the formula by $m \leq dn$.

Claim 5b: Fix $q^* \geq 1$. If for a given table R, the algorithm runs for $\geq q^* m$ iterations, then R must have a feasible tree of size $\geq q^*$. Thus,

$$\Pr_R[\#\text{iterations} \geq q^* m] \leq \Pr_R[R \text{ has a feasible tree of size} \geq q^*].$$

Claim 5c: The number of labeled trees of size q is $\leq m\binom{(d+1)q}{q-1} \leq m((d+1)e)^q$, and each one is feasible for a random R with probability $\leq p^q$.

Putting it together: Using Claims 5a and 5b and setting $q^* = \Theta(\log m)$, we have

$$\Pr_R[\#\text{iterations} \geq q^* m] \leq \sum_{q \geq q^*} m((d+1)e)^q p^q \leq m \sum_{q \geq q^*} (p(d+1)e)^q \leq m \sum_{q \geq q^*} 0.9^q \leq 1/2.$$

Exer: Use this analysis to prove that the expected runtime is polynomial.