CS 6200: Algorithmics II	**Fall 2020**
Instructor: Avah Banerjee, S&T.	Scribe: Self
Lectures: 1-3	**Dates: 8/24 - 8/28**

1 Verifying Matrix Product (Freivalds' algorithm)

Given three $n \times n$ matrices A, B, C our task is to verify whether $C = AB$. Trivially, we can compute the product first using any standard matrix multiplication algorithm. Then determine if $C - AB = 0$. However, the best known matrix multiplication algorithm takes $O(n^{2.372})$ (as of 2020). Even though it is conjectured that we can perform matrix multiplication in $O(n^{2+\epsilon})$ time (for every $\epsilon > 0$) we do know how to do this currently. Since we are tasked to only verify an equality, can we avoid having to multiply A and B.

Using randomization we will devise a simple algorithm that correctly verifies the equality with high probability. We say that some statement about an algorithm holds with high probability (w.h.p.) if the probability tends to 1 as n (size of the input) tends to infinity.

Algorithm 1 A Randomized Matrix Product Verifier

```
1: Input: A, B, C (all n × n matrices).
2: Output: returns true if C = AB else returns false.
3: Let 𝔹^n = {0,1}^n
4: Pick an element r from 𝔹^n uniformly at random
5: Compute D ← A(Br) − Cr
6: if D == 0 then
7:     return true
8: else
9:     return false
10: end if
```

Sampling r from $\mathbb{B}^n$ takes $O(n)$ given access to n random bits. Only "real"computation happens at line 5 which is a sequence of three matrix-vector multiplication followed by a subtraction. Each of these operations take $O(n^2)$. Thus a single invocation of Algorithm 1 takes $O(n^2)$ time.

Theorem 1.1 *If $AB \neq C$ then Algorithm 1 fails with probability $\leq 1/2$.*

Proof: Let $D = AB - C$. If $AB \neq C$ then $D \neq 0$. Then some entry in D, say d_{ij} must be non-zero. Suppose the algorithm fails. Then we have $ABr = Cr$, which implies $Dr = 0$. Thus for all $i \in [n]$,

$$\sum_{k=1}^{n} d_{ik} r_k = 0 \tag{1}$$

In particular,

$$r_j = -\frac{\sum_{k=1,k\neq j}^{n} d_{ik} r_k}{d_{ij}} \tag{2}$$

Now we use the *principle of deferred decision*. Suppose we have selected all r_k's before selecting r_j. At this stage RHS of Equation 2 is fixed. There is at most one value of r_j for which the equality in Equation 2 holds. Since r_j can either 0 or 1 with equal probability, we have probability of failure $\leq 1/2$. ■

Although a $\leq 1/2$ probability of failure seems bad we can easily improve our odds of success by running the algorithm multiple times. We can do this easily since Algorithm 1 has one sided error. At each run we choose a random vector r independently from the other runs. Thus the event that a particular run fails is independent of the rest of the runs. This meta-algorithm fails if every individual run fails; probability of this happening is $\leq \frac{1}{2^t}$, if we perform t runs of Algorithm 1. The meta-algorithm takes $O(tn^2)$ time. If $t = \log n$ we see that the algorithm succeeds with high probability with a running time of $O(n^2 \log n)$, still better than the current best known matrix multiplication algorithm.

Remark 1.1 *Suppose we are given only n random bits. Can we reuse these random bits to get a similar result as above?*

2 Method of Conditional Expectation

In this section we will use the method of conditional expectation to derandomize a randomized algorithm. First we recall the definition of conditional expectation (for discrete sample space). Suppose X, Y are two random variables on the same probability space. The conditional expectation $\mathbf{E}[X|Y]$ is a random variable Z; it is a function of Y.

$$Z(y) = \mathbf{E}[X|Y = y] = \sum_x x\mathbf{Pr}[X = x|\ Y = y]$$

We can write $Z = \sum_x x\mathbf{Pr}[X = x|Y]$. Conditional expectation generalizes the notion of conditional probability where X and Y are event random variables (a.k.a indicator random variables).

Example 2.1 *Suppose we roll two 6-sided dice. The value of the first roll is a r.v, say X_1. Similarly we define X_2. Let, $X = \max(X_1, X_2)$, which is the r.v that takes the maximum of the two rolls. We want to determine the conditional expectation $\mathbf{E}[X|X_1]$. Let us first calculate $\mathbf{E}[X|X_1 = i]$. If $X_1 = i$ then either $X = i$ or $X > i$. Thus,*

$$\mathbf{E}[X|X_1 = i] = i.\mathbf{Pr}[X = i|X_1 = i] + \sum_{x=X_1+1}^{6} x\mathbf{Pr}[X = x|X_1 = i]$$

The first term is simply $i^2/6$, since the probability that $X_2 \leq X_1$ is $X_1/6$. Similarly we find $\mathbf{Pr}[X = x|X_1 = i] = 1/6$. Putting these values in the above equation we get,

$$\mathbf{E}[X|X_1] = \frac{X_1^2}{6} + \frac{\sum_{X_1+1}^{6} x}{6} = \frac{X_1^2 - X_1 + 42}{12} \tag{3}$$

Since $\mathbf{E}[X|X_1]$ is a random variable we can apply the expectation operator on it, giving

$$\mathbf{E}[\mathbf{E}[X|X_1]] = \mathbf{E}[X] = \frac{\mathbf{E}[X_1^2] - \mathbf{E}[X_1] + 42}{12} \approx 4.5$$

2.1 Maximum Satisfiability (MaxSat)

Let f be a Boolean function on the set $V = \{x_1, \ldots, x_n\}$ of variables. f is given as a conjunction of disjunctions. That is, $f = \bigwedge_{c \in C} C$. Here C is the set of clauses and each cluse $c \in C$ is a disjunction: $c = x_{i_1} \vee \neg x_{i_2} \vee \ldots \vee x_{i_{n_c}}$. Additionally each clause c has a positive real weight $w(c)$. In the MaxSat our goal is to maximize the total weight of the satisfied clauses:

$$\max_{v \in \{0,1\}^n} \sum_{c \in C} I_c w(c)$$

Here v is a truth assignment and I_c is the indicator random variable that c is satisfied.

Algorithm 2 A Simple Randomized 2-approximation Algorithm

1: **Input:** A MaxSat instance f.
2: **Output:** A truth assignment of the variables.
3: Set each variable independently with probability 1/2 to 1 (true).
4: **return** this truth assignment.

First we observe that a clause is not satisfied iff all its literals are 0 (false). If the size of a clause c is $k \geq 1$ then $\mathbf{Pr}[I_c] = 1 - 1/2^k \geq 1/2$. Let, $W = \sum_{c \in C} I_c w(c)$.

Theorem 2.2 *Algorithm 2 produces a truth assignment such that $\mathbf{E}[W] \geq OPT/2$. Here OPT is the optimal value of the instance.*

Proof: We have,

$$\mathbf{E}[W] = \sum_{c \in C} \mathbf{Pr}[I_c] w(c) \geq \frac{1}{2} \sum_{c \in C} w(c) \geq \frac{1}{2} OPT.$$

■

However, the above result is in expectation. Now we look at a strategy where we can guarantee that the total weight of the satisfied clauses is at least $\mathbf{E}[W]$. The idea is to choose the truth assignment for the variables sequentially from 1 to n and build up a complete solution from a sequence of partial ones. We use the fact that *Satisfiabilty* is self-reducible[1]. We determine the

[1]There is a poly-time reduction from the search version to the decision version of the problem.

truth value for the variable x_i based on the following strategy. Let $\mathbf{E}[W|V_i]$ be the conditional expectation of W with respect to the subset of variables $V_i = \{x_1, \ldots, x_i\}$ which already been assigned a truth value. For every i we can compute $\mathbf{E}[W|V_i]$ easily: let C_T be the set of satisfied clauses and C_I be the remaining set of clauses after removing all the false literals. Then

$$\mathbf{E}[W|V_i] = \sum_{c \in C_T} w(c) + \sum_{c \in C_I} P(I_c) w(c)$$

Algorithm 3 A Derandomized version of Algorithm 2

Input: A MaxSat instance f.
Output: A truth assignment of the variables.
$i \leftarrow 0$, $V_0 = \emptyset$
while $i < n$ **do**
  if $\mathbf{E}[W|V_i \wedge x_{i+1} = \mathbf{true}] \geq \mathbf{E}[W|V_i \wedge x_{i+1} = \mathbf{false}]$ **then**
    $V_{i+1} \leftarrow V_i \cup \{x_{i+1} = \mathbf{true}\}$
  else
    $V_{i+1} \leftarrow V_i \cup \{x_{i+1} = \mathbf{false}\}$
  end if
  $i \leftarrow i + 1$
end while
return V_n.

Theorem 2.3 *The following invariant holds for Algorithm 3: for all i we have $\mathbf{E}[W|V_{i+1}] \geq \mathbf{E}[W|V_i]$.*

Proof: We prove this by induction on i. The base case, $i = 0$ is trivially true. Since we choose the assignment for x_{i+1} with equal probability, we have:

$$\mathbf{E}[W|V_i] = \frac{1}{2}\mathbf{E}[W|V_i \wedge \{x_{i+1} = true\}] + \frac{1}{2}\mathbf{E}[W|V_i \wedge \{x_{i+1} = false\}] \tag{4}$$

Thus we have $\max(\mathbf{E}[W|V_i \wedge \{x_{i+1} = true\}], \mathbf{E}[W|V_i \wedge \{x_{i+1} = false\}]) \geq \mathbf{E}[W|V_i]$, which proves the claim of the theorem. ■

3 Tail Bounds: A Randomized Median Finding Algorithm

Consider the following randomized algorithm to find the median of a set X of elements, where X has an unknown total order. In the following algorithm we ignore the floor and ceiling operations for notational simplicity, this does not affect our analysis.

Algorithm 4 A simple randomized median finding algorithm

```
1: Input: A set X, (|X| = n)
2: Output: The (lower) median of X or FAIL
3: Uniformly, independently and with replacement sample a set of n^{3/4} elements from X. Let Y
   be this sampled set.
4: Sort Y
5: Let l = (1/2 n^{3/4} − √n)^{th} smallest element in Y
6: Let h = (1/2 n^{3/4} + √n)^{th} smallest element in Y
7: Determine the set C = {x ∈ X | l ≤ x ≤ h} and let n_l = |{x ∈ X | x < l}| and n_h = |{x ∈
   X | x > h}|
8: if n_l > n/2 or n_h > n/2 or |C| > 4n^{3/4} then
9:    return FAIL
10: else
11:    Sort C
12:    Output the (⌊n/2⌋ − n_l + 1)^{th} element in the sorted order of C.
13: end if
```

Correctness and Running Time: Correctness follows from the if statement at line 8 of Algorithm 4. Only time the algorithm would fail to produce the correct median if C does not contain it. This can only happen if either $n_l > n/2$ or $n_h > n/2$. It is easy to verify that all the steps takes $O(n)$ time in total unless C is large. But in this case the condition $|C| > 4n^{3/4}$ is satisfied and the algorithm fails. Hence the algorithm either returns the median or fails and takes $O(n)$ time (comparisons).

Using tail inequality (here we use the Chebyshev's inequality) we will upper bound the failure probability.

Theorem 3.1 *Algorithm 4 fails with probability $O(n^{-1/4})$.*

Proof: The algorithm fails if any of the condition at line 8 holds. Let E_1 be the event that $n_l > n/2$. Similarly we define E_2 and E_3 ($|C| > 4n^{3/4}$). Then failure probability $\mathbf{Pr}[E_1 \cup E_2 \cup E_3] \leq \mathbf{Pr}[E_1] + \mathbf{Pr}[E_1] + \mathbf{Pr}[E_1]$, using the union bound. Due to symmetry $\mathbf{Pr}[E_1] = \mathbf{Pr}[E_2]$ so we only have to find $\mathbf{Pr}[E_1]$ and $\mathbf{Pr}[E_3]$.

Let m be the median of X. If $n_l > n/2$ then it must be the case that $l > m$. Since at most $\frac{1}{2}n^{3/4} - \sqrt{n}$ elements in Y are less than l there are at most this many elements in Y which can be less than m. Let X_i be the following indicator random variable:

$$X_i = \begin{cases} 1, & \text{if the } i^{\text{th}} \text{ sample is} < m. \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

Since there are $\lfloor n/2 \rfloor$ elements $< m$ and at least as many $> m$ we have $\mathbf{Pr}[X_i = 1] \approx 1/2$. Thus X_i's are distributed according to the Bernoulli distribution with $p = 1/2$. Thus $\mathbf{E}[X_i] = 1/2$ and $\mathbf{Var}[X_i] = 1/4$. Now let, $Z = \sum_i^{|Y|} X_i$. Z is the number of samples in Y less than m. The event E_1 is equivalent to saying $Z < \frac{1}{2}n^{3/4} - \sqrt{n}$. We want to show that $\mathbf{Pr}[Z < \frac{1}{2}n^{3/4} - \sqrt{n}] \leq \frac{1}{4}n^{-\frac{1}{4}}$.

For this we use the Chebyshev's inequality: Let $X_1, \ldots, X_n$ are independent random variables with $\mathbf{E}[X_i] = \mu_i$ and $\mathbf{Var}[X_i] = \sigma_i^2$ and $Z = \sum X_i$ then,

$$\mathbf{Pr}[|Z - \mathbf{E}[Z]| \geq \delta] \leq \frac{\sum \sigma_i^2}{\delta^2} \tag{6}$$

Now,

$$\mathbf{Pr}[Z - \mathbf{E}[Z] \leq -\delta] \leq \mathbf{Pr}[(Z - \mathbf{E}[Z] \leq -\delta) \vee (Z - \mathbf{E}[Z] \geq \delta)] = \mathbf{Pr}[|Z - \mathbf{E}[Z]| \geq \delta] \leq \frac{\sum \sigma_i^2}{\delta^2} \tag{7}$$

In our case $\mathbf{E}[Z] = \sum_{i=1}^{|Y|} \mathbf{E}[X_i] = \frac{1}{2}n^{3/4}$, $\sum \sigma_i^2 = \sum_{i=1}^{|Y|} \frac{1}{4} = \frac{1}{4}n^{3/4}$ and $\delta = \sqrt{n}$. Substituting these values in Equation 7 we get:

$$\mathbf{Pr}[E_1] \leq \frac{1}{4}n^{-\frac{1}{4}} \tag{8}$$

Similarly we can show that $\mathbf{Pr}[E_3] \leq \frac{1}{2}n^{-\frac{1}{4}}$, which is left as an exercise. ■

References and Further Reading

[1] Mitzenmacher, M., & Upfal, E. (2017). Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis. Cambridge university press. [Chapter 1, Chapter 2, Chapter 3, Chapter 6- Section 6.3, 6.3]