CMPUT 675: Randomized Algorithms	**Fall 2005**

Lecture 17: November 3

Lecturer: Mohammad R. Salavatipour — *Scribe: Bruce Fraser*

17.1 Bloom Filters

Hash table is good to find the object given its key. What if we only want to check the existence of the object? In general, with hash tables of size n, with a universe U with $|U| = m$, $\log m$ bits are needed to store each key in the table, so total size is $O(n \lg m)$ bits.

To save space, we can map each item to a single bit. The problem with this approach is that collisions cause false positives (saying an item is in the set while it really is not). Another approach is to use bit vectors of size n, and many (k) random hash functions. The multiple functions add redundancy to algorithm, and form a method for dealing with the inevitable collisions in such a small hash table.

Given item x, compute $h_1(x)$ up to $h_k(x)$ and set all set these bits in the bit vector to 1. To lookup an item: Check all $h_i(x)$ locations. If any is 0, say no, otherwise say yes.

Clearly if the algorithm returns no then x is not in the set. What is the probability of false positives?

For m distinct items we have km bit changes. Probability that a fixed bit remains 0:

$$\left(1 - \frac{1}{n}\right)^{km} \sim e^{\frac{-km}{n}} = p$$

Expected number of 0-bits is thus approximately np, and $(1-p)^k = f$ is the probability of false positives. We have to optimize k to minimize f.

Let $f = e^g$ then $g = k\ln(1 - e^{-km/n})$, $\frac{dg}{dk} = \ln(1 - e^{-km/n}) + \frac{km}{n} \cdot \frac{e^{-km/n}}{1-e^{-km/n}}$ The zero is at $k = \ln 2(n/m)$. Thus with this value $p = \frac{1}{2}$ and $f \sim \frac{1}{2}^k = (0.6185)^{n/m}$. If n is approximately $8m$, the probability of a false positive is less than 2%.

17.2 Fingerprinting

Basic idea is simple and we saw it in Lecture 1: to compare two items x, y from a huge universe U, any deterministic algorithm clearly needs $O(\log |U|)$ bit comparisons. However, we can map u to a smaller set V and then compare images of x, y in V. This will require only $O(\log |V|)$ bits.

Some preliminaries: recall the algebraic definition of a field, $\mathbf{F}$ a set over which two operations are defined, addition and multiplication. We will be working with fields in this topic.

17.2.1 Freivalds technique

Consider the following simple problem: given matrices A, B and C, it would be desirable to have an efficient decision procedure for confirming if $AB = C$. Checking the equality in the obvious brute force manner would be in $O(n^3)$, dominated by the cost of multiplying A and B (this can be improved to $O(n^{2.376...})$). We want a faster (simple) algorithm to check $AB = C$, this is good as a test at the end of complex multiplication programs.

Consider $r \in 0, 1^n$. We could instead check if $ABr = Cr$. If $AB = C$ there is no problem with this procedure, but what if $AB \neq C$ but $ABr = Cr$? To rephrase, let $D = (AB - C)$. What if $Dr = 0$ but D is non-zero? If D is non-zero there is a non-zero row, without loss of generality assume $d_1 \neq 0$.

$$\sum_{i=1}^{n} d_i r_i = 0 \Rightarrow r_i = \frac{-\sum_{i=2}^{n} d_i r_i}{d_1}$$

Assume that all $r_i \geq 2$'s are chosen before r_1, then there is a unique value $v \in F$ for r, and the probability that r_1 takes that value is less than $\frac{1}{2}$. If $r \in S \subset \mathbf{F}$ then probability is less than $\frac{1}{|S|}$.

17.2.2 Comparing Polynomials

Given polynomials P, Q, R over $\mathbf{F}$, we want to check if $P(x)Q(x) = R(x)$, where P and Q have degree at most n, and R has degree of at most $2n$. Using FFTs you can multiply in $O(n \lg n)$ time to evaluate $R(x)$.

Random test:

- Let $S \subset \mathbf{F}$, and pick $r \in S$.
- Compute $P(r)Q(r) - R(r)$.
- If non-zero we know that the original equality does not hold,
- otherwise we conclude the equality does hold.

What is the probability of error? if $D(x) = P(x).Q(x) - R(x)$ is not identically zero but the value r that we chose is one of its roots. Since degree of $R(x)$ is at most $2n$, degree of $D(x)$ is at most $2n$ and so has at most $2n$ roots. Thus the probability of error is at most $\frac{2n}{|S|}$.

Now consider multivariate polynomials, composed of n variables. The degree of a term is defined as the sum of included variables degree. For example, $deg(x^2yz^3) = 6$. We may have an implicit representation of these polynomials (e.g. given an matrix A, with variables being the entries, the polynomial might be the permanent of this matrix).

Theorem 17.1 (Shwartz and Zippel) *Let $Q(x_1, ..., x_n)$ be a multivariate polynomial of total degree d. Fix $S \subset \mathbf{F}$ and $r_1, r_2, ..., r_n$ be selected uniformly randomly from S. Then*

$$\Pr[Q(r_1, ..., r_n) = 0 | Q(x_1, ..., x_n) \not\equiv 0] \leq \frac{d}{|S|}$$

There is no known deterministic polytime algorithm for identity checking for multivariate polynomials.

Proof: We prove by induction on n. Base Case: $n = 1$, was done in the previous subsection.

Consider Q and pick one variable, say x_1 that affects Q and factor it:

$$Q = \sum_{i \leq k} x_1^i Q_i(x_2, ..., x_n)$$

where k is largest exponent of x_1. We have $Q_k(x_2, \ldots, x_n) \not\equiv 0$ by the choice of x_1 and the total degree of Q_k is at most $d - k$. First assume that $Q_k(r_2, \ldots, r_n) \neq 0$. Let $q(x_1) = Q(x_1, r_2, \ldots, r_n) = \sum_{i=0}^{k} x_1^i Q_i(r_2, \ldots, r_n)$. Since $q(x_1)$ is a singe variable polynomial, the probability that $q(x_1)$ equals to zero is at most $\frac{k}{|S|}$. Also, by induction hypothesis and since total degree of Q_k is at most $d - k$:

$$\Pr[Q_k(r_2, ..., r_n) = 0] \leq \frac{d-k}{|S|}.$$

Thus:

$$\Pr[Q_k(r_2, ..., r_n) = 0] \leq \frac{d-k}{|S|}$$

$$\Pr[Q(r_1, ..., r_n) = 0 | Q_k(r_1, ..., r_n) \neq 0] \leq \frac{k}{|S|}$$

For any two events: E_1 and E_2:

$$\Pr[E_1] = \Pr[E_1 \cap \overline{E_2}] + \Pr[E_1 \cap E_2] \leq \Pr[E_1 | \overline{E_2}] + \Pr[E_2].$$

Thus $\Pr[Q(r_1, \ldots, r_n) = 0] \leq \frac{d-k}{|S|} + \frac{k}{|S|} = \frac{d}{|S|}$, as claimed. ■

17.3 Perfect Matchings

In this section, we look at efficient randomized algorithms for checking existence of and finding perfect matchings in bipartite graphs. The algorithm uses the results on polynomial identity checking described before.

A bipartite graph $G(U \cup V, E)$ is a graph in which the vertex set is partitioned into two parts U and V and all the edges have exactly one end point in each of U and V. With $|U| = |V| = n$, a perfect matching $M \subseteq E$ is a set of n edges such that the subgraph of G induced by M is 1-regular.

Definition 17.2 *A Tutte matrix of a bipartite graph $G(U \cup V, E)$ is an n by n matrix M, such that:*

$$M_{ij} = \begin{cases} 0 & \textit{if } u_i, v_j \notin E \\ x_{ij} & \textit{if } u_i, v_j \in E \end{cases}$$

The determinant of the Tutte matrix can be used to check existence of perfect matchings in G.

Lemma 17.3 *$Det(M) \neq 0 \Leftrightarrow G$ has a perfect matching*

Proof: Using the definition of determinant, $Det(M) = \sum_{\pi \in P_n} (-1)^{sign(\pi)} \prod_{i=1}^{n} M_{i,\pi(i)}$, where P_n is the set of all permutations of $\{1, ..., n\}$. There is a trivial one-to-one correspondence between perfect matchings in G and permutations of $\{1, \ldots, n\}$: $\{(v_1, u_{\pi(1)}), \ldots, (v_n, u_{\pi(n)})\}$. Thus:

$$Det(M) = \sum_{\pi \in P_n} (-1)^{sign(\pi)} \prod_{i=1}^{n} x_{i,\pi(i)}.$$

This implies that, every term in the above summation is non-zero iff and only if it corresponds to a perfect matching of G. Since every produce in the summation is unique, no two terms cancel each other out. ■

This gives an easy way of checking for the existence of a perfect matching in G. $Det(M)$ is a degree n polynomial, so if we pick S with $|S| = 2n$ then

$$Pr[G \text{ has no perfect matching and we accept}] = 1$$

$$Pr[G \text{ has a perfect matching and we reject}] \leq \frac{1}{2}$$

This topic is continued in the next lecture.