FINGERPRINTING SETS

Richard J. Lipton

CS-TR-212-89

March 1989

Fingerprinting Sets

Richard J. Lipton*
Computer Science Department
Princeton University

Abstract

We show how to efficiently compute hash functions that are invariant under permutations. These hash functions have a variety of applications to distributed computing problems.

1 Introduction

An interesting application of "randomness" is the random pattern-matcher of Karp and Rabin [1]. One way to understand their work is to realize that is based on a new kind of data structure. In particular, they show how to efficiently implement the following operations:

- *add* an element to the bottom of a queue;
- *remove* an element from the top of a queue;
- *test* if two queues are equal.

The operations are all done in $O(1)$ time. The key is performing the last operation efficiently. They do this by constructing a *fingerprint* for each queue. By a fingerprint we mean that if the two queues are equal, then they must have the same fingerprints; on the other hand, if they are unequal, then their fingerprints are very unlikely to be the same. It is easy to extend their ideas to similar data structures such as stacks and queues.

The central idea used in their data structure is that strings can easily be fingerprinted. Let us select a random prime of suitable size. We agree that the fingerprint of x is simply $x \bmod p$. Then if x and y are unequal it is very unlikely (as a function of the size of p) that $x \equiv y \bmod p$ i.e. that x and y have the same fingerprints. Indeed Karp and Rabin show that $p \leq O(\log(n))$ is enough to make agreement of unequal $n - bits$ numbers very unlikely.

Another interesting application of such fingerprints is the following [2]. Assume two separate locations have two large files x and y. How can they tell if the files are the same, i.e. that $x = y$? In the worst case one must send many bits. However, there is a very short random strategy: just choose a random prime p and exchange the fingerprints of x and y. Then if the fingerprints are different the files must be different, and if they are the same the files are very likely to be the same. This is a result of Yao and has recently been extended by Galil, Landau, Yung [4] to multiple sets of files.

*Work supported by DARPA and ONR contracts N00014-85-C-0456 and N00014-85-K-0465, and by NSF Cooperative Agreement DCR-8420948

The problem we study is a different generalization. Assume that x is a file of records and so is y. Thus, $x = x_1, x_2, \ldots, x_k$ and $y = y_1, y_2, \ldots, y_l$ where x_i is the *ith* record of file x and the y_i is the *ith* record of file y. Then our problem is: are x and y equal as multi-sets, i.e. is there a permutation π so that

$$x_1 = y_{\pi_1}, x_2 = y_{\pi_2}, \ldots, x_k = y_{\pi_k}$$

and $k = l$. We wish, therefore, to generalize the fingerprint method from strings to sets. The chief difficulty, of course, is that now our fingerprints must be invariant under arbitrary permutations.

2 Solutions

We first consider a number of "obvious solutions" to our fingerprint problem before we present our approach. Each of these solutions fails to correctly solve the problem.

(a) *Hash Sums*

One simple idea is to use $x_1 \oplus \ldots \oplus x_k$ as the fingerprint where $\oplus$ denotes the exclusive-or sum. While in practice this may be a good heuristic it clearly fails to solve the problem. For example, x_1, x_1, x_1 and x_1, x_2, x_2 both always get the same fingerprint. Even without duplicates it is easy to see that this simple method does not always work.

(b) *Other Sums*

Another simple idea is to use

$$\sum_{i=1}^{k} f(x_i)$$

as the fingerprint of $x_1, x_2, \ldots, x_k$ where $f(x)$ is a fixed polynomial. Again, perhaps this is a good heuristic but it cannot be a fingerprint in our sense. One can prove that for any d there are two distinct sets $x_1, \ldots, x_k$ and $y_1, \ldots, y_k$ so that

$$\sum_{i=1}^{k} f(x_i) = \sum_{i=1}^{k} f(y_i)$$

holds for all $f(x)$ with degree at most d.

(c) *Random Hash Sums*

This is a method due to Wegman and Carter [3]. Their idea is to associate with each record r a random bit pattern say $f(r)$. Then the fingerprint of $x_1, \ldots, x_k$, is

$$f(x_1) \oplus \ldots \oplus f(x_r).$$

Wegman and Carter show that these are indeed fingerprints in our sense provided duplicates are not allowed. The problem with their method is simple: it requires *global* knowledge of the map r to $f(r)$. Therefore, it cannot be used in our distributed example since to transfer this map from one site to the other is as hard or harder than the original problem.

(d) *Sort*

By this we mean just sort the files $x_1, \ldots, x_k$ and $y_1, \ldots, y_l$ and then fingerprint the files. Of course this works but it is quite costly in computing time.

We will now present our fingerprint method. Again let $x_1, x_2, \ldots, x_k$ be the set of records and assume each is at most an $n - bit$ record. Choose two random numbers p a prime and $r \in \{0, 1, \ldots, p-1\}$; we will discuss their size in a moment. Then the *fingerprint* of $x_1, x_2, \ldots, x_k$ is

$$\prod_{i=1}^{k}(x_i + r) \bmod p.$$

There are several key remarks we wish to make. First, this can clearly be computed in linear time. Second, it is also clear that our fingerprint is *invariant* under permutations, i.e. $x_{\pi_1}, \ldots, x_{\pi_k}$ has the same fingerprint for all permutations π. Finally, we claim that the probability of two unequal sets getting the same fingerprint is very small. This is stated precisely in our theorem:

Theorem 2.1 *The probability that $x_1, x_2, \ldots, x_k$ and $y_1, y_2, \ldots, y_l$ are unequal and get the same fingerprints is at most*

$$O\left(\frac{\log(n) + \log(m)}{nm} + \frac{1}{n^2 m}\right)$$

where all records are $n - bit$ numbers at most and $m = \max(k, l)$ provided only the prime p is selected randomly from the interval

$$[(nm)^2, 2(nm)^2].$$

Thus, the prime p and hence the fingerprint has at most $O(\log(n) + \log(m))$ bits. Before we can prove this theorem we need one definition and a simple lemma. The *height* $H(\Phi)$ of a polynomial $\Phi(x) = \Phi_d x^d + \ldots + \Phi_0$ is the maximum of $|\Phi_d|, \ldots, |\Phi_0|$. Then,

Lemma 2.1 *(1) Let $\Phi_1(x)$ and $\Phi_2(x)$ be polynomials. Then, $H(\Phi) \le H(\Phi_1) + H(\Phi_2)$. (2) Let $\Phi(x) = (x - a_1) \ldots (x - a_m)$. Then,*

$$H(\Phi) \le (1 + |a_1|) \ldots (1 + |a_m|)$$

.

Proof: (1) is obvious. (2) follows by induction and the simple observation that

$$H(\Phi(x)(x - a)) \le H(\Phi)(1 + |a|).$$

□

3 Proof of Theorem:

Proof: Let $f(u) = (u - x_1) \ldots (u - x_k)$ and $g(u) = (u - y_1) \ldots (u - y_l)$ be polynomials in u. Also, let $\Phi(u) = f(u) - g(u)$. Since $x_1, \ldots, x_k$ and $y_1, \ldots, y_l$ are unequal as multisets, the polynomial $\Phi(u)$ is not identically zero. Clearly, its degree is at most m. By the lemma its height is at most,

$$\prod_{i=1}^{k}(1 + |x_i|) + \prod_{j=1}^{l}(1 + |y_j|).$$

Since each record is at most $n - bits$ it follows that $H(\Phi)$ is at most 2^{nm+1}. Now let $t = (nm)^2$ and let p be a random prime with $t \le p \le 2t$ and let r be a random residue modulo p.

We must now show that it is unlikely that $\Phi(r) \equiv 0 \bmod p$ since this implies that $x_1, \ldots, x_k$ and $y_1, \ldots, y_l$ have different fingerprints. Since Φ is not identically 0 it has at least one coefficient say c that is not 0. By the lemma, c is at most 2^{nm+1}. Therefore, c has at most $nm+1$ prime factors. Thus, the probability that $c \equiv 0 \bmod p$ is at most

$$O\left(\frac{nm}{\frac{t}{log(t)}}\right)$$

since there are approximately $\frac{t}{\log(t)}$ primes in the given interval. Since $t = (nm)^2$ it follows that this probability is

$$O\left(\frac{log(n) + log(m)}{nm}\right).$$

Next we need to compute the probability that $\Phi(r) \equiv 0 \bmod p$ given that $c \not\equiv 0 \bmod p$. Since Φ is not identically zero it follows that it has at most m roots. Thus, the probability that $\Phi(r) \equiv 0 \bmod p$ is at most m/p or $1/n^2m$. Therefore, the total probability the two sets have the same fingerprint is at most

$$O\left(\frac{\log(n)+\log(m)}{nm} + \frac{1}{n^2m}\right).$$

□

4 Applications

There are several immediate applications for our ability to fingerprint sets.

(a) *Distributed Equality of Sets*

This is just the original problem we considered. The key point is we can now tell efficiently if two distributed data bases contain the same set of records *not* necessarily stored in the same way. Just as Galil et al [4] generalizes Yao [2], we can also have more than two data bases. Suppose $P_1, P_2, \ldots, P_m$ are processors each with a part of a data base and $Q_1, Q_2, \ldots, Q_m$ are also processors each with a part of the data base. Then, we can check efficiently to see if

$$\cup_{R\, record\, in\, P_i} R = \cup_{S\, record\, in\, Q_j} S$$

we do this by exploiting the simple fact we can compute the fingerprint of

$$P_1, \ldots, P_m(\text{resp.} Q_1, \ \ldots, \ Q_m x)$$

by just computing locally and then taking the product of the fingerprints mod$p$.

(b) *A Set Data Structure*

As in Karp and Rabin [1] consider the following data structure:

1. *add* an element to a set;
2. *delete* an element from a set;
3. *test* if two sets are equal.

Our fingerprint method allows us to easily implemented this data structure. Note, we do the deletion (2) by dividing modulo p. Thus, a data base can keep track of the fingerprints of its records as updates to the data base occur. In this manner fingerprints are always available.

(c) *Remote Sorting*

Our fingerprint method is quite useful in another kind of distributive computing problem. Suppose we wish to sort a large file of records $x_1, \ldots, x_k$. Assume we send it to another site over a high-speed link and get back $y_1, \ldots, y_k$. How can we be sure that this is indeed the sorted version of our original file? Clearly, our fingerprint method allows us to do this easily. Note, even if the other site was an *enemy*, the probability that we are fooled can be made arbitrarily small. (We of course keep p and r secret in this protocol.)

(d) *Lost Messages*

Suppose we wish to check that a network has correctly delivered certain messages. In many networks messages will *not* always arrive in the same order they were sent. But, our fingerprint method will allow one to check that the messages were not changed, and it will do this without the need to sort them into the original order. We only need maintain the fingerprints of the sets of messages that we have received and the set of messages that we have sent. Then periodically we can test as in (a) that certain sets are equal. If they are not, then an error must have occurred; if they are equal, then with high probabilty no messages have been lost or changed by the network.

5 Conclusions

Finally, let us close with a generalization of the problem considered here. Let, G be a group of permutations that act on $\{1, 2, ..., k\}$. Then, can we fingerprint $x_1, x_2, \ldots, x_k$ so that the fingerprint is invariant under all of G? More precisely, can we efficiently compute a hashing function h that has the following two properties:

1. $h(x_1, \ldots, x_k) = h(x_{\pi_1}, \ldots, x_{\pi_k})$ for any π in the group G;
2. if there is no π in the group G so that $x_1 = y_{\pi_1}, \ldots, x_k = y_{\pi_k}$, then it is likely that $h(x_1, \ldots, x_k) \neq h(y_1, \ldots, y_k)$.

Thus, Karp and Rabin, used the case where G is the identity group. We handle the case where G is the full symmetry group. It appears to be open what happens in general. Indeed consider the following group P_k: it acts on the set of pairs (x, y) with x and y in the range of 1 to k. It performs both row and column permutations. Essentially, P_k is the permutation group for bipartite graph isomorphism. Then, it is easy to see that the following is true:

Theorem 5.1 *If P_k has an fingerprint function that can be computed in random polynomial-time, then graph isomorphism is in random-polynomial time.*

References

[1] R.M. Karp and M.O. Rabin, Efficient randomized pattern-matching algorithms, Tech. Rept., Center for Research in Computing Technology, Harvard University, 1981.

[2] A.C. Yao, Some complexity questions related to distributive computing, *Proc. ACM on Theory of Computing* (1979) 209-213.

[3] M.N. Wegman and J.L. Carter, New classes and applications of hash functions, *20th Annual Symposium on Foundations of Computer Science*, (1979) 175-182.

[4] Z. Galil, G.M. Landau and M.M. Yung, Distributed algorithms in synchronous broadcasting networks, Theoretical Computer Science 49 (1987) 171-184.

894