

A new randomized algorithm for Document Fingerprinting

Sumit Ganguly

Department of Computer Science
and Engineering
Indian Institute of Technology
Kanpur, UP, INDIA - 208016
Email: sganguly@cse.iitk.ac.in

Gaurav Veda

Department of Computer Science
and Engineering
Indian Institute of Technology
Kanpur, UP, INDIA - 208016
Email: gveda@cse.iitk.ac.in

Abstract— In today’s world, copy detection is a major problem. Students plagiarize assignments from the web and from each other. In such a scenario, we need a technique that can detect even partial copies between assignments subject to relocation. This problem also finds uses in the context of the web. Search engines are highly interested in detecting copies of entire web documents to avoid displaying the same content multiple times in the result.

Document fingerprinting is an efficient technique for the accurate detection of full and partial copies between documents. We have come up with a new randomized algorithm that provides a guarantee that with very high probability, any match of greater than or equal to W characters (an input parameter) will be detected. Moreover, we have small deterministic bounds on the amount of space needed for our algorithm. This is the key way in which it differs from previous work, where either there are no guarantees [1] or the space bounds are very poor [2].

I. INTRODUCTION

Detecting full and partial copies between a set of documents is a frequently occurring and important problem. Consider, for example, a typical university setting. Students often end up copying assignments (especially programming assignments) from each other and from the web. It is easy to detect exact copies between documents by just comparing a full document checksum. However, it is rarely the case that whole documents (read assignments) are copied. Generally, students use loads of tricks to avoid getting detected by a copy checking mechanism. Some of these are relocating parts of the code, changing variable names, rewriting parts of the code and/or changing the algorithm for a minor task (such as sorting etc.). Because of this, copy detection becomes quite challenging.

Document fingerprinting is an efficient technique for accurately detecting full and partial copies of documents. Here, the idea is to store a small sketch (that is, a representative set of numbers) such that by comparing the sketches between two documents, we will be able to identify whether they have a substantial overlap. In addition to the university example above, this technique finds use in many real-world applications. The Google search engine, uses the technique outlined by Broder[1] to detect copies of webpages while crawling the web. It then tries to display only unique results (removing mirror sites etc.) so that the user has a better selection.

In document fingerprinting, we are interested in storing a small sketch since otherwise copy detection would become computationally infeasible. We want the detection mechanism to be fast so that we can compare all the documents in a given set for detecting copies.

Most of the existing techniques for copy detection use a notion of w -grams [2] or *shingles* [1]. We also use the notion of w -grams. A w -gram is a set of w contiguous characters in the document. Refer to following figure for an example.

Jack and Jill

(a) Some text

jackandjill

(b) The text after conversion to the canonical input format

jacka ackan ckand kandj andji ndjil djill

(c) The set of all possible 5-grams derived from (b)

Figure 1: Introducing w -grams

In the technique of document fingerprinting, each of the w -grams so identified is treated as a single unit and is hashed using a one-one hash function (note that the number of w -grams and hence the number of hashes is close to the size of the document. To be precise, it is equal to $m - w + 1$, where m is the size of the document). After hashing, a subset of these hashes is chosen as a *fingerprint* of the document and forms the sketch. The various techniques for fingerprinting differ in the way in which they choose a subset as the sketch and the number of hashes present in that subset. Once sketches are formed, to detect similarity between documents, we just compare their sketches.

In the existing approaches for creating sketches, either the space needed is not bounded (only a probabilistic bound assuming uniformity of content exists for [2])

II. EXISTING APPROACHES

One popular approach for obtaining constant size sketches is to simply choose the lowest or highest s sketches, where s is the required size of sketch [1]. However, it is easy to see that though this technique might work well when we

expect the documents to be very small (depending on s) or have substantial similarity, it would fail in many cases. At a philosophical level, if one stores a very small amount of information about a document as a sketch, then there is only little that one can do with this limited information.

Another technique suggested in [1] is to store all hashes that are $0 \bmod p$. However, here, although on expectation we expect that the size of the sketch would only be $\frac{1}{p}$ of the document size, in the worst case, it could be as large as the document itself. In particular, consider the case in which the entire document contains a repeating sequence. So here, if a w -gram hashes to $0 \bmod p$, we will be storing all copies of it and the storage would be large. However, if a w -gram does not hash to $0 \bmod p$, then despite numerous occurrences, we will not store even a single occurrence of this substring. Moreover, no bounds can be obtained limiting the distance between two consecutive hashes thus chosen. In [2], it is reported that experiments on web data have shown that the gap between 2 successive hashes can indeed be very large. Therefore, no guarantees in terms of copy detection for a match exceeding a certain number of bits can be given. This makes the approach not so practical, since we are generally interested in tagging the documents as similar even if there is a single substring that is greater than a certain size (input parameter) and occurs in both documents.

An alternative technique that can give deterministic guarantees for detecting similarities whenever the size of a single common substring exceeds W (an input parameter) is given in [2]. The algorithm is based on sliding windows. It takes 2 parameters as input : W and w . W is the *guarantee threshold*. The algorithm guarantees that any matches that are more than W characters in length will be detected by it. The other parameter is a *noise threshold*. Matches below this are considered irrelevant and will not be detected. This parameter ‘ k ’ is also used as the gram size. Matches of length between k and W , might or might not be detected. The algorithm uses a sliding window of size W . It starts at the beginning of the document and slides the window character by character while picking up the minimum hash in every window. The set of minimum hash values then forms the sketch of the document. If two consecutive windows have the same minimum value and a common element occurs in both the windows having this value, then only a single copy of the element is kept in the sketch. Assuming uniformity of input, the algorithm creates a sketch of $\frac{2m}{W+1}$ in size, where m is the length of the document.

The problem with this approach is that assuming uniformity of input is not a good assumption. Generally the input data (from the web etc.) is highly non-uniform. Therefore, the size of the sketch that the algorithm stores could in practice become very large. In particular, an input can be crafted such that the algorithm will pick almost as many w -grams as the document size.

III. SOME PRELIMINARIES

Before going into our algorithm, here we will briefly describe the synopsis structure begin used by us for maintaining

distinct items and the count-sketch algorithm for estimating the frequency of items using only limited space. These are techniques generally used for processing data-streams.

A. Synopsis data structure

Synopsis data structures are defined as “any data structures that are substantively smaller than their base data sets” [3]. We use such a structure for maintaining distinct samples in the document. We also require a one-one hash function. Let us assume that the hash function gives an l -bit number as output.

$$h : \{w\text{-grams}\} \rightarrow \{0, 1, \dots, |F| - 1\}$$

$$|F| = 2^l$$

We define the level of a w -gram as follows:

$$level(sub) = \begin{cases} 0 & \text{if } h(sub) = 0 \\ i & \text{if } lsb(h(i)) = i \end{cases}$$

where, $lsb(num)$ = position of the last 1 in the l -bit representation of the number

Let $p(l)$ denote the probability that an input w -gram hashes to level l .

$$p(l) = \begin{cases} \frac{1}{2} + \frac{1}{|F|} & \text{if } l = 1 \\ \frac{1}{2^l} & \text{otherwise} \end{cases}$$

To create a distinct sample structure, we follow the following simple algorithm. At every level, we have a set of items. Each item is a tuple, the two entries being the hash of a w -gram and its position in the document. To insert a w -gram, we compute its hash and ascertain its level (as shown above). If that level has not been dropped (explained below), go to that level and add the tuple containing this w -gram and its position to the set of items at that level. Whenever we exceed a given parameter s , the space bound for this structure, we simply drop the lowest level and all the items contained in the set corresponding to that level to reclaim space. The basic idea is that the probability that a given items hashes to level l goes on decreasing exponentially as we increase the level. So in every subsampling operation, we expect to reduce the size of the stored quantites by a factor of 2.

B. Sketches

Sketches are commonly used in the context of data-streams. This is because they can be used to compute most of the quantities about the stream that we are interested in and they take up a lot less space as compared to storing the entire stream. The basic idea is to use a 4-wise independent hash function to map each item of a stream to either 1 or -1. In our case, each item in the stream is a w -gram. That is,

$$h : \{w\text{-gram}\} \rightarrow \{1, -1\}$$

To create a sketch X , initialize X to 0, and on every input i , update X as

$$X = X + h(i)$$

This is known as the sketch of the stream. It can also be expressed as

$$X = \sum_{i \in w\text{-grams}} f_i * h(i)$$

where f_i is the frequency of the i th w -gram in the document (or the stream).

C. COUNTSKETCH algorithm

The Countsketch algorithm is one of a class of algorithms that are used for estimating the frequency of items in a data-stream while using low space. It uses the following interesting observation about sketches. Let $Z = \sum_i f_i x_i$ be a sketch. Then,

$$\begin{aligned} E[Z * x_i] &= f_i \\ \text{and } Var[Z * x_i] &\leq F_2 - f_i^2 \end{aligned}$$

where, $E[y]$ and $Var[y]$ denote the expected value and the variance of the quantity y and F_2 is the second moment

$$F_2 = \sum_i f_i^2$$

This can be seen as follows.

$$\begin{aligned} E[Z * x_i] &= E[x_i * \sum_j f_j * x_j] \\ &= E[f_i * x_i^2 + \sum_{j \neq i} f_j * x_i * x_j] \\ &= f_i + \sum_{j \neq i} f_j * E[x_i * x_j] \\ &= f_i + \sum_{j \neq i} f_j * E[x_i] * E[x_j] \quad (1) \end{aligned}$$

$$\begin{aligned} &= f_i + \sum_{j \neq i} f_j * 0 * 0 \quad (2) \\ &= f_i \end{aligned}$$

Note that (1) follows from the pairwise independence of the hash function used and (2) follows from the fact that since x_i is equally likely to be either 1 or -1 , the expected value of x_i is 0.

Let g be a function chosen from a family of pair-wise independent hash functions. such that it maps each item i in the domain to one of the buckets between 0 and $B - 1$.

$$g : \{0, 1, \dots, N - 1\} \rightarrow \{0, 1, \dots, B - 1\}$$

Assume that the hash table has B buckets and in each bucket we maintain the sketch of all the items that hash to that bucket.

Then, by Chebychev's bounds and boosting confidence, it follows that if we keep $O(\log \frac{1}{\delta})$ copies of sketches (each using a different hash function) and we predict the frequency of a particular item i as the median (\hat{f}_i) of the frequencies estimated by each such structure ($Z * x_i$), then the following holds

$$if \quad f_i \geq \frac{\Delta}{\epsilon},$$

$$then, \quad Pr\{\hat{f}_i \in (1 \pm \epsilon)f_i\} \geq 1 - \delta$$

where, $Pr a \in (1 \pm \epsilon)b \geq 1 - \delta$ means that the probability that the quantity a lies between $(1 - \epsilon)b$ and $(1 + \epsilon)b$ is greater than or equal to $1 - \delta$ and

$$\Delta = \left(\frac{8F_2^{res}}{B} \right)^{1/2}$$

$$F_2^{res} = F_2 - f_i^2$$

IV. OUR APPROACH

We start from a canonical form of the document. We convert the input document to this format to remove undesirable differences between documents. This is the form obtained after removing whitespaces and doing domain specific processing on the document. An example of domain specific processing is to replace all the occurrences of variable names in a program by a single identifier V . This ensures that a match will be found between 2 programs even if the author of one program has changed variable names to avoid copy detection. Another example might be the conversion of all characters to lowercase and the removal of punctuation and common phrases / w -grams. This would be the processing done for a text document.

Our approach for determining the signature of a document is based on keeping a distinct sample of the items that do not occur very frequently in the document. In addition we keep a frequent items data structure that contains items that occur frequently within a single window and a global frequent items data structure containing items that are globally frequent. We explain each of these data structures, how to construct them and their role below.

We take as input the parameters W and s . W is the *guarantee threshold* [2]. Our aim is to be able to determine matches between 2 documents if the match exceeds more than W consecutive characters. s is a *sketch size parameter* that determines the size of the distinct sample. The minimum value of s depends on W and will be bounded later. Intuitively, a larger value of s implies better detection of matches. We need 2-passes over the input document to determine the sketch to be kept. This is a big drawback since this makes the algorithm unapplicable for data-streams. We are working to remove this limitation. The algorithm for static documents is as follows.

In our algorithm, we use a sliding window of size W . Therefore, our aim is to be report 2 documents as being partial copies of each other if they contain atleast one window in common.

A. Distinct Samples Data Structure

We keep a distinct samples data structure of the kind described in Section III above. The space given to this data-structure is s (the input parameter).

B. Frequent Items Data Structure

All w -grams that have a frequency greater than a parameter k in a single window are classified as Window Frequent Items. These are stored in a separate data structure. This data structure is what we term as the Frequent Items Data Structure.

Here, each entry is a tuple. The first entry of the tuple is a w-gram and the second entry is the position of the window in which it occurs frequently, i.e. with frequency $\geq k$. We describe below how the value of the parameter k is determined.

C. Global Frequent Items Data Structure

All w-grams that have a frequency greater than a parameter L in the entire document are classified as Global Frequent Items. These are also stored in a separate structure that we label as the Global Frequent Items Data Structure. Here, each entry is just a w-gram. We do not store any position information.

D. The algorithm

Use the Countsketch method described above, in the first pass. At the end of the pass, retrieve all w-grams that have frequency at least L , where,

$$L = \frac{7}{16} \frac{s}{\log \frac{1}{\delta}}$$

Insert all the w-grams thus identified into the Global Frequent Items Data Structure.

Make a second pass over the document and create the distinct samples structure. Do not insert the w-grams already identified as globally frequent into this structure. Also, simultaneously fill the frequent items structure using

$$\begin{aligned} k &= \frac{W}{t} \\ t &= \frac{2}{p} * \log \frac{1}{\delta} \end{aligned}$$

where, p is the sampling probability. This can be estimated from the first pass (since $p = \frac{1}{2^l}$, where l is the final level after the document has been sampled). or be approximated as $\frac{m}{s}$.

Creating both the distinct samples data structure and the frequent items data structure simultaneously can be done as follows. Consider partitions of the document of size $2W$ that overlap in W , the window size.

$$P = [1, 2W][W + 1, 3W] \dots$$

Now check if any w-gram occurs with frequency $\geq k$ in any partition. If it does, add it to the frequent items data structure and do not add it in the distinct samples data structure.

E. Analysis

This is the final sketch that we obtain. The space needed by the frequent items data structure is $O(\frac{m}{k})$. The number of globally frequent items appearing in the sample is at most $\frac{m}{L}$. Therefore, the number of globally frequent items that can occur in the distinct samples data structure is on expectation, at most $p * \frac{m}{L}$ where p is the sampling probability. So by Hoeffding's bounds, with probability $\geq 1 - \delta$, at most $O(\frac{m}{L} * p * \log \frac{1}{\delta})$ globally frequent items occur in the distinct sample. Let

$$\frac{m}{L} * p * \log \frac{1}{\delta} \leq \frac{s}{8}$$

Therefore, by Hoeffding's bounds, with probability $\geq 1 - \delta$, size of the distinct samples structure excluding globally frequent items (D) is

$$D \leq \max\left(2m' * p * \log \frac{1}{\delta}, 2L * \log \frac{1}{\delta}\right)$$

$$\begin{aligned} \Rightarrow \frac{7}{8}s &\geq 2m' * p * \log \frac{1}{\delta} \\ \text{and } \frac{7}{8}s &\geq 2L * \log \frac{1}{\delta} \end{aligned}$$

Substituting p from here into the bound calculated for frequent items data structure, we finally obtain

$$\begin{aligned} s &= O\left(\frac{m}{\sqrt{W}} * \sqrt{\log \frac{1}{\delta}}\right) \\ \text{and } L &\leq \frac{7}{16} \frac{s}{\log \frac{1}{\delta}} \end{aligned}$$

V. FUTURE WORK

The first thing that needs to be done is to implement this algorithm and run it on a large collection of documents to gauge its effectiveness. Once this is done, we need to compare it with the approaches in [1] and [2]. The next step could be to generalize this algorithm to a one-pass algorithm. This would allow it to be used for data-streams as well.

Another minor improvement could be that instead of deciding L , k etc. in the way it is being currently being done, we simply ask the user for the sketch size (S) that he/she wants and then generate the best possible sketch. We internally decide how to partition this S into s (the distinct sample size), the frequent items data structure and the global frequent items structure. In other words, the algorithm should decide the values of the parameters k & L .

A problem with the algorithm is that it needs $O(\frac{m}{\sqrt{W}})$ space in the worst case. If instead of this we could reduce the maximum space needed to $O(\frac{m}{W})$, then in every case we would store a sketch that is approximately equal to or much smaller than the sketch that will be stored by the algorithm described in [2]. One possible way to do this is as follows. Instead of storing all distinct items, in the first pass we make an estimate of the probability p with which items are to be sampled. Then, in the next pass, we store the first item with level greater than or equal to the calculated level. After that, we look at the next W items (the guarantee threshold) and store *only 1* of them - maybe the rightmost item with level greater than or equal to the calculated level. This should at least intuitively reduce the storage from $O(\frac{m}{\sqrt{W}})$ to $O(\frac{m}{W})$. The idea here is that to detect similarity between documents, even if there is exactly one *w-gram* stored in every window of size W , then we can use it as the seed of similarity and expand the match. We don't need more than a single *w-gram* per window.

VI. CONCLUSION

We have come up with a promising new approach to address the problem of detecting full and partial copies. The most notable feature of our algorithm is that we are able to give guarantees (to any level of accuracy) while still using a small amount of storage that is bounded deterministically. The frequent items and global frequent items data structure ensure that we will be able to handle any kind of documents. We make no assumption on the input. In particular, we do not assume it to be uniformly distributed, unlike in previous work [2].

ACKNOWLEDGMENT

Gaurav Veda would like to thank Dr. Sumit Ganguly (CS497 guide) for introducing this interesting problem to him and allowing him to work on it for the seminar course CS497 - Special Topics in Computer Science.

REFERENCES

- [1] A. Z. Broder, "On the resemblance and containment of documents," in *Proceedings of the Compression and Complexity of Sequences 1997*, 1997, p. 21.
- [2] S. Schleimer, D. Wilkerson, and A. Aiken, "Winnowing: Local algorithms for document fingerprinting," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2003, pp. 76–85.
- [3] P. B. Gibbons and Y. Matias, "Synopsis data structures for massive data sets." in *SODA*, 1999, pp. 909–910.