International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 06 Issue: 02 | Feb - 2022 ISSN: 2582-3930

Hashing Algorithm Techniques

Ms. Nupur Gaikwad
PG student,
Computer Engineering Department,
Alamuri Ratnamala College of Engineering, Mumbai

Prof. Sanjay Jadhav
Associate Professor,
Dept. of Computer Engineering,
MGM College of Engineering Technology, Mumbai

***Abstract*:-** In this thesis, we show that the traditional idea of hashing goes far be-yond near-neighbor search and there are some striking new possibilities. We show that hashing can improve state of the art large scale learning algorithms, and it goesbeyond the conventional notions of pairwise similarities. Despite being a very well studied topic in literature, we found several opportunities for fundamentally improvingsome of the well know textbook hashing algorithms. In particular, we show that the traditional way of computing minwise hashes is unnecessarily expensive and without loosing anything we can achieve an order of magnitude speedup. We also found that for cosine similarity search there is a better scheme than SimHash. In the end, we show that the existing locality sensitive hashing frameworkitself is very restrictive, and we cannot have efficient algorithms for some important measures like inner products which are ubiquitous in machine learning. We propose asymmetric locality sensitive hashing (ALSH), an extended frame- work, where we show provable and practical efficient algorithms for Maximum Inner Product Search (MIPS). Having such an efficient solutions to MIPS directly scales up many popular machine learning algorithms

INTRODUCTION

- What is Hashing?

Hashing Algorithm are the functions that generate a fixed length result .

For instant, think of paper documents that you keep crumpling to the point where you aren't even able to read its content anymore. Its almost impossible to restore the original data or file input without knowing what the starting Data was.We could discuss if it is secured algorithm. Ever input number is Individual .

Hashing Algorithm Explained

A hash function algorithm is designed to be a one-way function, Infeasible to invert. However in recent years several Hashing algorithms have been compromised. This happened to MD5 , for example a widly know function designed to be cryptographic hash function , which is know

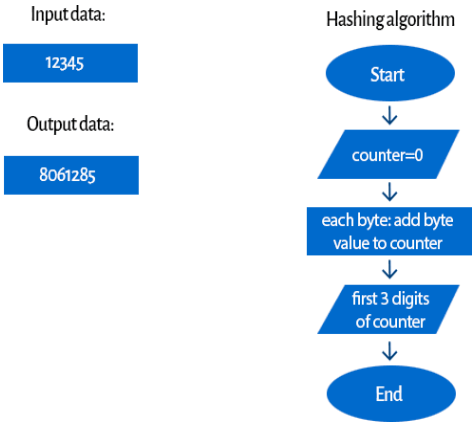

Fig1: Hashing Algorithm Example with a simple hash function

very easy to reverse that we could only use for verifying data against unintentional corruption .

Its easy to find out what the ideal cryptographic function should be like :

1. It should be fast to compute the hash value for any kind of data.
2. It should be impossible to regenerate a message from its value
3. It should be infeasible to find two message with same hash like collision.
4. Every change to a message, even a smallest one , should change the hash value. It should be completely different. Its called the Avalanche Effect.

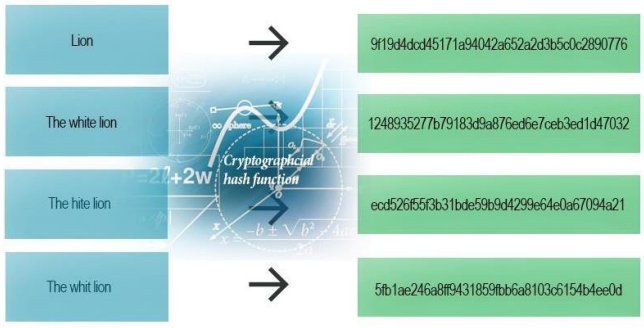

Fig2: Shows how any small changes can affect the whole hash function

- Hashing Algorithm Example

We're sending file to our friend. Its very important file and we have to ensure that it delivered only in one piece, that where our Hashing Function comes in. But first ,lets see how our file transfer will look without it.

Step 1: User1 sends file to User2.

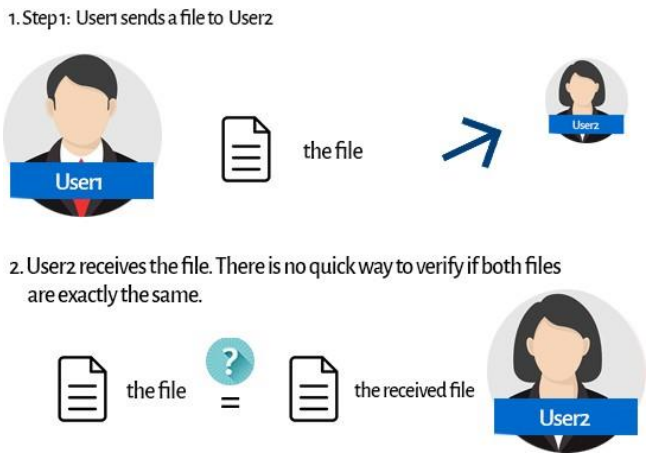

Fig3: Example without Hashing Function

We can figure out some trivial ideas. You could, for instance , call user2 and you could check file content together.But then what's the point in sending ? Checksum are our godsend here.

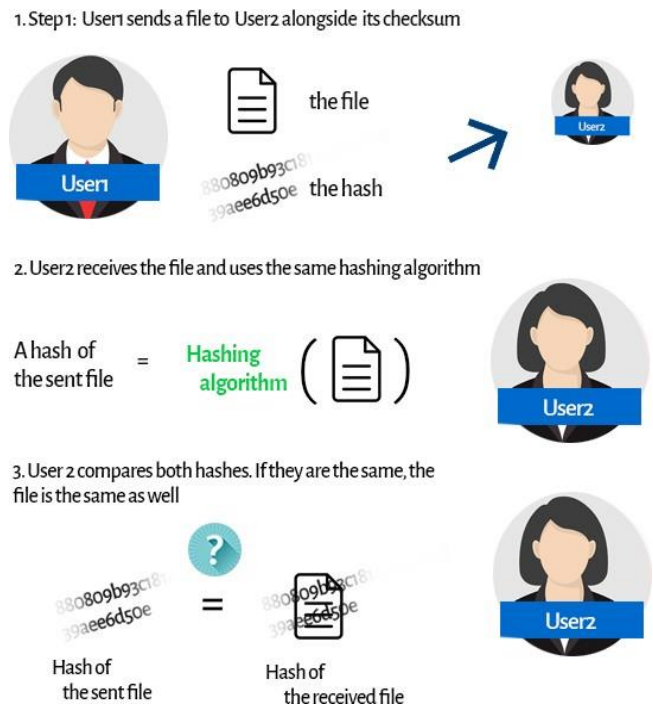

Fig4: Example with Hash Algorithm

Before sending a file, User1 uses a hashing algorithm to generate a checksum for a file. Then he/she sends it alongside the file itself. User2 receives both the file and the checksum. Now he/she can use the same hashing algorithm on the received file. What's the point? We already know that a hash is individual (so there can't be any other file with the same hash) and has to be always the same for an individual file. No matter how many times you use the hash algorithm, itwill always give you the same result. So now, User2 can compare both hashes. If they're the same, it means it's generated from the same file. There is no way that any other file has the samehash and there is no chance for a hash to be different for the same file.

This way, User2 can verify if the file isn't in any way corrupted. Easy? Certainly. A lot of downloading services use checksums to validate the integrity of their files. Thanks to that, you can find out if your downloaded file isn't corrupted.

Popular Hashing Algorithms

MD5

Before we go any further – **MD5 is completely broken**!

If you ever learned any programming language and it was some time ago, you surely knowthis algorithm. It's one of the most widely known.

This hash algorithm used to be widely used and is still one of the most widely known hashing algorithms. But despite initially being designed to be used as a cryptographic algorithm function, it is no longer considered safe to use for cryptographic purposes, as it is compromised. In particular, it is possible to quickly generate collisions on ordinary computers.

When MD5 is used to hash passwords directly, there is an even easier way to break it... Google. By typing the hash in the search box, there's a good chance you'll receive its before-state within milliseconds! Now let's look at this example:

You could think you are secure if your passwords are stored as MD5 hashes, but if somebodygets access to your database, he/she can just type the hash to Google and get its real value!

	id	login	email	password
Edit Copy Delete	1	Tester	itsjustanotheremail@mail.com	cdb662ff7eaa975e2990760b1a5c0834
Edit Copy Delete	2	Tester2	tester@mail.com	b5500dfb568d36e779c64846004d3b76
Edit Copy Delete	3	Tester3	tester3@mail.com	48503dfd58720bd5ff35c102065a52d7
Edit Copy Delete	4	Admin	tester4admin@mail.com	2e33a9b0b06aa0a01ede70995674ee23

The CMU Software Engineering Institute considers MD5 essentially "cryptographically broken and unsuitable for further use". It was accepted for many years, but it's now mainly used for verifying data against unintentional corruption

SHA-family

Secure Hash Algorithm is a cryptographic hash function designed by the United States' NSA. **SHA-0** (published in 1993) has been compromised many years ago. SHA-1 (1995)
produces a 160-bit (20-byte) hash value. It's typically rendered as a 40 digits long hexadecimalnumber. It has been compromised in 2005 as theoretical collisions were discovered, but its real "death" occurred in 2010 when many organizations started to recommend its replacement.

The big three – Microsoft, Google, and Mozilla — have stopped accepting SHA-1 SSL certificates in 2017 on their browsers, after multiple successful attacks. SHA-1 was built on principles similar to those used in the design of the MD4 and MD5. It has a more conservative approach, though.

Safer, for now, is **SHA-2**. SHA-2 includes several important changes. Its family has six hash functions with digests: SHA-224, SHA-256 or 512 bits: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256.

There are numerous reasons why you should move to SHA-2 if you haven't yet. We also find some useful resources that can help you with this move.

As a bottom line, SHA-2 is a lot more complicated and is still considered safe. However, SHA-2 shares the same structure and mathematical operations as its predecessor (SHA-1) — so it's likely that it will be compromised in the near future. As so, a new option for the future is SHA- 3.

SHA-3 (Secure Hash Algorithm 3) designed Guido Bertoni, Joan Daemen, Michaël Peeters and Gilles Van Assche. Their algorithm Keccak won the NIST contest in 2009 and has been adopted as an official SHA algorithm. It was released by NIST on August 5, 2015. One of SHA-3's requirements was to be resilient to potential attacks that could compromise SHA-2.

Keccak is significantly faster than SHA-2 (from 25% to 80%, depending on implementation). It uses the sponge construction. The data is first "absorbed" into the "sponge" and the result is "squeezed" out. While absorbing, message blocks are XORed into a subset of the state. Then it's transformed as one element. While squeezing, output blocks are read from this element, but alternated with state transformations.

A key aspect of SHA-3 is that it was designed to easily replace SHA-2 in applications that currently use that variant. As so, the transition from SHA-2 to SHA-3 should be analyzed in regards to the required security level and the overhead (refactoring/testing) — that greatly depend on the application's structure and architecture.

Probabilistic Hashing Techniques For Big Data

Background: Classical Locality Sensitive Hashing (LSH)

In this technique, we briefly review Locality Sensitive Hashing (LSH) families and their applications in sub-linear time near neighbor search. We refer readers to [36] for detailed description of existing works in LSH literature. This thesis provides several new fundamental results and improvements in the LSH do- main. The concepts described in this chapter will be heavily referred through- out the course of this thesis. The Near-Neighbor Search Problem Near-neighbor search or similarity search is one of the fundamental problems incomputer science.

The Near-Neighbor Search Problem

Near-neighbor search or similarity search is one of the fundamental problems incomputer science. this problem, we are typically given a giant collection $C \subset R^D$ anda query $q \in R^D$. The task is to search for point $x \in C$ which minimizes (or maximizes) thedistance (or similarity) with the query q, i.e., we are interested in $x=\arg\min Dist(x,q)$

Classical LSH Algorithm

To be able to answer queries in sub-linear time, the idea behind the LSH al- gorithm is to create hash tables from the given collection C which we are in- terested in searching. The hash table construction is one time costly operation which makes further querying efficient. The hash tables are generated using the locality sensitive hash family.

Table1 Table L

h^1_1	...	h^1_K	Buckets
00	...	00	...
..00	...	01	...
00	...	10	Empty

h_{L1}	...	h^L_K	Buckets
00	...	00	...
00	...	01	...

We assume that we have an access to the appropriate locality sensitive family H forthe similarity search problem of interest. The classical (*K*, *L*) parameter- ized LSH algorithm, which we also refer to as *b*ucketing algorithm, works as follows. We generate *L* different meta-hash functions. Each of these meta-hash functions is formed by concatenating *K* sampled hash values from H.

$B_j(x) = [h_{j1}(x); h_{j2}(x); ...; h_{jK}(x)]$,

Non-Linear Learning In Linear Time Via Hashing

With the advent of the Internet, many machine learning applicationsare faced with very large and inherently high-dimensional datasets, resulting in challenges in scaling up training algorithms and storing the data. Especially in the context of search and machine translation, corpus sizes used in industrial practice have long exceeded the main memory capacity of single machine. For example, discusses training sets with 10^{11} items and 10^9 distinct features, requiring novel algorithmic approaches andarchitectures.

As a consequence, there has been a renewed emphasis on scaling up machine learning techniques.
In this chapter we show that locality sensitive hashing techniqueswhich are used for efficient indexing, for sub-linear time near-neighbor search, also lead to very efficient and practical large scale learning algorithms. In particular, we show that a locality sensitive hashing family directly leads to kernel features for the associated similarity function. These kernel features can then be used for approximate learning with (non-linear) kernels in linear time, which otherwise is quadratic and proh expensive.

MINHASH or SIMHASH?

Locality Sensitive Hashing (LSH)) scheme is tightly coupled with the underlying hash function which in turn is defined with respect to a similar- ity measure. An LSH scheme for one similarity measure cannot be used in gen-eral for a different similarity measure. Therefore, it is taken for granted that the two popular hashing schemes MinHash and SimHash, defined in section, are incomparable and the choice between them is based on whether the desired notion of similarity is resemblance or cosine similarity.

In this chapter, we show that for sparse binary data, which is usually of interest over the web, there is actually a fixed choice among these two hashing schemes. Our theoretical and empirical results show a counter-intuitive fact that for sparse data MinHash is the preferred choice even when the desired measure is cosine similarity. Although the decade old concept of LSH comes with a rich theoretical analysis, there is no machinery to mathematically compare two LSH schemes for different similarity measures. By showing that MinHash is prov- ably superior to SimHash for retrieval with cosine similarity, we provide the first evidence that two LSHs for different similarity measures can be compared.

MinHash and SimHash for Sparse Binary Data

Current web datasets are typically very sparse and extremely high dimensional, mainly due to the wide adoption of the “Bag of Words” (BoW) representations for documents and images. In BoW or shingle representations, it is known that the word frequency within a document follows power law, indicating that most of the words occur rarely in the document.In “BoW” representations most of the higher order shingles in the document occur only once.

It is often the case that just the presence or absence information suffices in practice]. Thus, high dimensional web data are almost always binary or binary like. The most information is in the sparsity structure of a vector rather than the magni- tude of its components. Many leading search companies use only sparse binary representations in their large data systems. Furthermore, there are many empirical studies which suggest that binary quantizations of datasets achieve good performance in practice. All these factors lead to an increased emphasis on techniques which are capable of exploiting binary datasets.

MinHash an LHS for Cosin Similarity

We would like to higjlight that the well known ρ value for MinHash and SimHash from equation which determine the worst case query complexity of these algorithms, are not directly comparable because they are in the context of different similarity measures i.e. resemblance and cosine similarity. To make the comparison possible,we fix our gold standard similarity measure to be the cosine similarity $S\,im = S$. Theorem 2 leads to two simple corollaries:

Corollary *If* $S(x, y) \geq S_0$, *then we have*

$$Pr\left[h^{min}(x) = h^{min}(y)\right] = R(x, y) \geq S(x, y)^2 \geq S_0^2$$

Hashability For K-Way Similarities

Existing notions of similarity in search problems mainly work with pairwise similarity functions. In this chapter, we go beyond this notion and look at the problem of *k*-way similarity search, where the similarity function of interest in- volves *k* arguments ($k \geq 2$). An example of higher order similarity is the **3-way Jaccard**similarity which is defined as:

$$R_{3way} = \frac{|S1 \cap S2 \cap S3|}{|S1 \cup S2 \cup S3|},$$

Importance of *k*-way Resemblance

We list **four** practical scenarios where *k*-way resemblance search would be a natural choice and in the later section provide some empirical support.

Improving retrieval quality**:** We are interested in finding images ofa particular type of object, and we have two or three(possibly noisy) represen- tative images. In such a scenario, a natural expectation is that retrieving im- ages simultaneously similar to all the representative images should be more re- fined than just retrieving images similar to any one of them. In Section 5.1.1, we demonstrate that in cases where we have more than one elements to search for, we can refine our search quality using *k*-way resemblance search. In a dynamic

Beyond pairwise clustering: While machine learning algorithms of- ten utilize the data through pairwise similarities (e.g., inner product or resem- blance), there are natural scenarios where the affinity relations are not pairwise.

GoogleSets:
(http://googlesystem.blogspot.com/2012/11/google-sets-still- available.html) *Google Sets* is among the earliest google projects, which allows users to generate list of similar words by typing only few related key- words. For example, if the user types “mazda” and “honda” the application will automatically generate related words like “bmw”, “ford”, “toyota”, etc. This application is currently available in google spreadsheet.

Joint recommendations: Users A and B would like to watch a movie together. The profile of each person can be represented as a sparse vector over a giant universe of attributes. For example, a user profile may be the set of actors, actresses, genres, directors, etc, which she/he likes. On the other hand, we can represent a movie M in the database over the same

CONCLUSION

Hashing algorithms can be pretty useful. However, IT is a really fast-changing industryand this entropy also extends to hashing algorithms.

MD5, once considered really safe now it’s completely compromised. Then there wasSHA-1, which is now unsafe. The same thing will surely happen to the widely used SHA-2 someday.

We investigate probabilistic hashing techniques for addressing computational and memory challenges in large scale machine

learning and data mining sys- tems. In this thesis, we show that the traditional idea of hashing goes far be- yond near-neighbor search and there are some striking new possibilities. We show that hashing can improve state of the art large scale learning algorithms, and it goes beyond the conventional notions of pairwise similarities. Despite being a very well studied topic in literature, we found several opportunities for fundamentally improving some of the well know textbook hashingalgorithms. In particular, we show that the traditional way of computing minwise hashes is unnecessarily expensive and without loosing anythingwe can achieve an order of

REFERENCE:

[1] Christina Brandt, Thorsten Joachims, Yisong Yue, and Jacob Bank. Dy- namicranked retrieval. In *WSDM*, pages 247–256, 2011.

[2] Harald Baayen.*Word Frequency Distributions*, volume 18 of *Text, Speechand Language Technology*. Kulver Academic Publishers,2001

[3] Yoram Bachrach, Yehuda Finkelstein, Ran Gilad-Bachrach, Liran Katzir, Noam Koenigstein, Nir Nice, and Ulrich Paquet. Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14,2014

[4] Hajishirzi, H., Yih, W.-t., and Kolcz, A. (2010). Adaptive near-duplicate detection via similarity learning. In Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval, pages 419–426. ACM.

[5] Jimmy Ba and Brendan Frey.Adaptive dropout for training deep neural networks. In Advances in Neural Information Processing Systems, pages 3084– 3092, 2013.

[6] http://www.howtogeek.com/howto/15799/how-to-use-autofill- on-a- google-docs-spreadsheet-quick-tips/.

[7] S. Agarwal, Jongwoo Lim, L. Zelnik-Manor, P. Perona, D. Kriegman, and S. Belongie. Beyond pairwise clustering. In *CVPR*, 2005

[8] Nir Ailon and Bernard Chazelle. Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform. In *STOC*, pages 557–563, Seattle, WA, 2006

[9] Alexandr Andoni and Piotr Indyk. E2lsh: Exact euclidean locality sensitivehashing. Technical report,2004.

[10] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash func-tions. In *STOC*, pages 106–112, 1977.