

On the Application of Locality Sensitive Hashing to
Behavioral Web Analytics

Shantanu Gore

Thomas Jefferson High School for Science and Technology

June - August 2014

Information Management and Decision Architectures Branch

Code 5584

Dr. Myriam Abramson

Abstract

In today's constantly connected world, the dependence on web-based technologies is ubiquitous, creating opportunities for both malicious and benign activity. As a result, it is essential that we be able to identify users on the web. Although simple methods, such as tracking a user by userid or by IP address exist, these methods can easily be evaded if the user so desires, by creating multiple ids or operating from different IP addresses. However, due to habit, a user does not often change the way he or she browses the web, such as the time of day that she visits various genres of pages. In this project, we evaluate locality sensitive hashing (LSH) to uniquely identify and authenticate users based only on the day of the week, time of day, and genres of websites. In addition, we provide a novel extension of LSH, Mode Closest Hash (MCH).

1 Introduction / Motivation

In a wide variety of applications, such as cyber security and authentication, it is often necessary to be able to uniquely identify users. We currently have systems for this, such as usernames and passwords, as well as physical biometrics, such as fingerprint or iris sensors. However, a commonality between these systems is that they can be broken. Usernames and passwords can be stolen, by something as simple as a camera installed at a user's desk, while physical biometrics can be relatively expensive and time consuming to implement [3]. What we need is a way to identify users based on their behavior, which is not something that can be stolen easily from another user, and as such is more difficult to be compromised.

We investigated LSH to approach this problem, because it excels at finding near neighbors in sub-linear time. Furthermore, by purposefully adding noise (through the use of random vectors), LSH also serves to mitigate existing noise in a dataset. It does so by projecting all the vectors onto multiple random vectors, reducing the impact of noisy elements.

LSH is better for finding near or nearest neighbors in high dimensions than other algorithms, because it scales well. That is, it is not impeded by the Curse of Dimensionality, which says that finding nearest neighbors efficiently in higher dimensions (i.e. without comparing a query point to every other point) is not well defined, because all points are near neighbors of all other points, which makes it difficult to find the nearest neighbor. In other words, the relative contrast (the ratio of the average distance between all pairs of vectors in a dataset to nearest neighbor distances) declines as the dimension of a dataset increases (Figure 1).

The remainder of this paper is organized as follows. Section 2 covers some related work, and compares it to our approach. Section 3 provides an introduction to LSH, which Section 4, proposed approach, builds upon. Section 5 goes over the conclusions and possible further work.

2 Related Work

Most previous work with locality sensitive hashing has dealt with finding duplicates in large datasets [7], such as one of the webpages on the web. Due to the size of this dataset, it is often impractical to do a brute force nearest neighbor search – one that involves comparing a given vector to every other vector to find which is the closest. Instead, LSH is used – the locality sensitive hashes of all the documents in the dataset are precomputed (this only needs to be done once), and then the hash of the test vector is computed.

Gionis et al. argue that LSH addresses the Curse of Dimensionality by placing the data into “bins,” and then determining which bin the query point would fall into[4]. They demonstrate that if appropriate hash functions are used, similar points are likely to fall into the same bin, while distant points are not. Due to this, the Curse of Dimensionality is avoided – one needs only compute the hash of a query point, not compare it to every other (or a subset of these) points. This method is somewhat similar to clustering, finding which vectors are in the same bin. However, just because two vectors are in the same bin does not guarantee that they should be in the same group, as this can just be an effect of the random vectors. Gionis et al. conduct two experiments of their method on data, which validate their approach.

A paper by Slanley et al. provides a method to compute analytically the values of parameters used for LSH (k , w , and L)[7]. It asserts that these parameters are a function of the number of data points, n , the acceptable error tolerance, δ , the average distance from a point to its nearest neighbor, d_{nn} , the average distance from a point to a random point, d_{any} , and the acceptable Hamming radius for multiprobe (which allows for nearest neighbors to be in bins that are a certain distance from a query point’s bin), r . It does so by computing collision probabilities, which indicate how likely two points are to fall in the same bin, and then computing the probability that a point and its nearest neighbor fall in the same bin, as well as adjacent bins. These parameters cannot simply be optimized individually, as they are dependent on one another. For example, changing w will result in a different optimal

value for k . Stanley et al. conduct experiments to validate their results, and find that the parameters determined by their method are indeed the optimal parameters for LSH.

In their paper, Pang et al. assert that simply removing or changing the MAC address (explicit identifiers) from an 802.11 device, such as a router, is not enough to protect the identity of users, as they can be identified on the web based solely on “implicit identifiers” [5]. They propose four new implicit identifiers: network destinations, network names identified in 802.11 probes, configurations of 802.11 options, and sizes of broadcast packets. They ask two questions – did a particular sample come from a particular user, and did a particular user connect to a network today. They use a naïve Bayes classifier to attempt to identify users, and use one feature for each identifier. They find that 64% of users can be determined with 90% accuracy in public networks with less than 100 users. Further, they determine that some users are detectable with high accuracy – between 12% and 37% of users can be identified with 99% accuracy in networks with 25 users.

A paper by Roussev et al. explores the use of bloom filters, a similar concept as LSH [6]. It gives a brief overview of the existing problem, that MD5 hashing is often impractical for forensic (file system) analysis. Specifically, they calculate that the hashes for a relatively small dataset (512 GB) would take 32 GB to store, while most workstations have only a few gigabytes of RAM. Thus, the size of each hash needs to be reduced by a factor of at least 8. To do so, they propose the use of Bloom Filters, which can efficiently store and query a large number of hashes, and answer queries about whether two files are identical. Bloom filters create an array of k bits, and then use k separate hash functions to hash the input vector and get k hashes. These k bits are set to one in the output vector. This process creates a false negative rate of zero, as two vectors with differing hash values cannot be identical. However, there remains some false positive rate, as the fact that the same bits are one could just be a coincidence. This false positive rate is roughly equal to the following:

$$p_{FP} = (1 - (e^{-kn/m}))^k$$

A paper by Abramson & Gore investigates the use of recurrent neural networks to identify users on the web[1]. They convert two months of user data into vector form, taking into account the genre of a page a user visited, time visited, and day of the visit. They split this up temporally into training and testing, in an attempt to identify a user’s testing vector from a set of training vectors. They use a Hopfield Network to find the closest training vector to a given testing vector. They also use other methods to accomplish this, such as a simple Hamming distance (the number of different bits). Furthermore, in order to improve the accuracy of the Hopfield Network, they use a “Tournament Approach,” where the testing vector is compared to the training vectors, taken in groups of two, and the closer training vector in each pair advances to the next “round,” with the final one remaining chosen as the training vector closest to the testing vector. They find that it is indeed sufficient to use the genre vs. time of day approach in order to identify users on the web, with an accuracy of up to 81%.

3 Introduction to LSH

Locality Sensitive Hashing, or LSH, differs from regular hashing in that the LSH of two vectors that are closely related is more likely to be the same than the LSH of two vectors that are far apart. The LSH of a vector \vec{x} is defined as follows, where \vec{x} is a d dimensional data vector, \vec{v} is a d dimensional vector containing random real elements picked from a Gaussian distribution, w is the window size, and b is a random real number in the range $[0, w]$.

$$LSH(\vec{x}) = \lfloor \frac{\vec{x} \cdot \vec{v} + b}{w} \rfloor$$

This method essentially transforms a d -dimensional vector into an integer. It is clear that for identical \vec{x} , using the same \vec{v} , b , and w will provide identical hashes. Also, for similar \vec{x} , using the same \vec{v} , b , and w will provide either identical or similar hashes, depending on the magnitude of w . Furthermore, for a vector \vec{y} that is distant from \vec{x} , LSH will provide a

different hash, depending on the magnitude of w . In fact, for an appropriate hashing function (such as the dot product, among others), LSH guarantees the following probabilities.

$$LSH(\vec{x}) = LSH(\vec{y}) \geq P_1 \text{ for } |\vec{x} - \vec{y}| \leq R_1$$

$$LSH(\vec{x}) = LSH(\vec{y}) \leq P_2 \text{ for } |\vec{x} - \vec{y}| \geq R_2$$

$$P_1 > P_2$$

Due to the inherent randomness present in LSH (through the elements of \vec{v} as well as the value of b), it is often essential to do multiple, k , random projections for the LSH algorithm. This produces a k -dimensional integral vector, as opposed to a single integer. To further mitigate the effects of the randomness, it is beneficial to run L iterations of the LSH algorithm to get L k -dimensional vectors[2].

LSH is often used to find the nearest neighbor to a given a vector (testing vector) given a set of possibilities (training vectors). To do this, it computes the LSH of all the training vectors, as well as the LSH of the testing vector. Using the k -dimensional hash vector for each of the training vectors, it creates a mapping between the elements at each position and the related training vectors. Then, it computes the k -dimensional hash vector for the testing vector, and retrieves the set of training vectors for each position. Now, it does one of two things, depending on the implementation: it either identifies the mode of the list of possible training vectors as being the same as the testing vector, or it places all the training vectors that share at least one hash into a list and computes the distance to each of the training vectors in the list, and identifies the one with the smallest distance as being the closest to the testing vector. It now has an answer – the closest training vector to the testing vector. By repeating this process L times and taking the mode of the answers, we can identify, with high accuracy, the actual closest vector to the testing vector.

3.1 Example

Consider the set of $n = 4$ training vectors:

$$\left\{ \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \right\},$$

and the set of $k = 3$ random vectors:

$$\left\{ \begin{bmatrix} -1.3 \\ 0.8 \\ 0.3 \\ -1.0 \\ 1.7 \end{bmatrix}, \begin{bmatrix} -0.3 \\ 0.3 \\ -0.6 \\ 0.9 \\ -1.2 \end{bmatrix}, \begin{bmatrix} 0.1 \\ -0.5 \\ -2.4 \\ 0.0 \\ -1.2 \end{bmatrix} \right\},$$

and let w be 1.0 and b be 0.8.

We determine the first hash to be:

$$\lfloor \frac{\vec{x} \cdot \vec{v} + b}{w} \rfloor = \lfloor \frac{1 * -1.3 + 1 * 0.8 + 0 * 0.3 + 0 * -1.0 + 1 * 1.7 + 0.8}{1.0} \rfloor = \lfloor \frac{2.0}{1.0} \rfloor = 2$$

We determine the remaining hashes:

Training Vector	Random Vector	Hash
0	0	2
0	1	-1
0	2	-1
1	0	-2
1	1	1
1	2	0
2	0	3
2	1	-1
2	2	-4
3	0	0
3	1	1
3	2	-2

Creating a list of all the training vectors associated with each random vector, we obtain the following:

Random Vector 0	Random Vector 1	Random Vector 2
2 → 0	-1 → 0, 2	-1 → 0
-2 → 1	1 → 1, 3	0 → 1
3 → 2		-4 → 2
0 → 3		-2 → 3

Now consider the testing vector:

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Using the same random vectors and b , we obtain

$$\begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}$$

for the k -dimensional hash vector. Now, consulting the table created for each element of this vector, we determine that the -1 in the first position does not correspond with any training vector. Moving on, we determine that the 1 in the second position indicates that the testing vector is close to either the 1^{st} or 3^{rd} training vector. Continuing to the third position, we use the 0 to determine that the testing vector is close to the 1^{st} training vector. Now, as mentioned previously, the algorithm will either compute the distance from the testing vector to the two identified training vectors, or simply choose the 1^{st} training vector as being closest to the testing vector, because it came up twice while the 3^{rd} one only came up once. In this example, it does not matter which one the algorithm chooses, as both vectors are equidistant from the testing vector.

4 Proposed Approach

We applied LSH to two problems in behavioral web analytics: identification and authentication. Although these problems are closely related, there are some key differences, such as in the training set and the output values. We conducted analyses for both the binning and MCH. The pseudocode for the MCH algorithm (Algorithm 1) and the authentication algorithm is shown (Algorithm 2). The steps taken to convert the data into a format usable for LSH are shown (Figure 1).

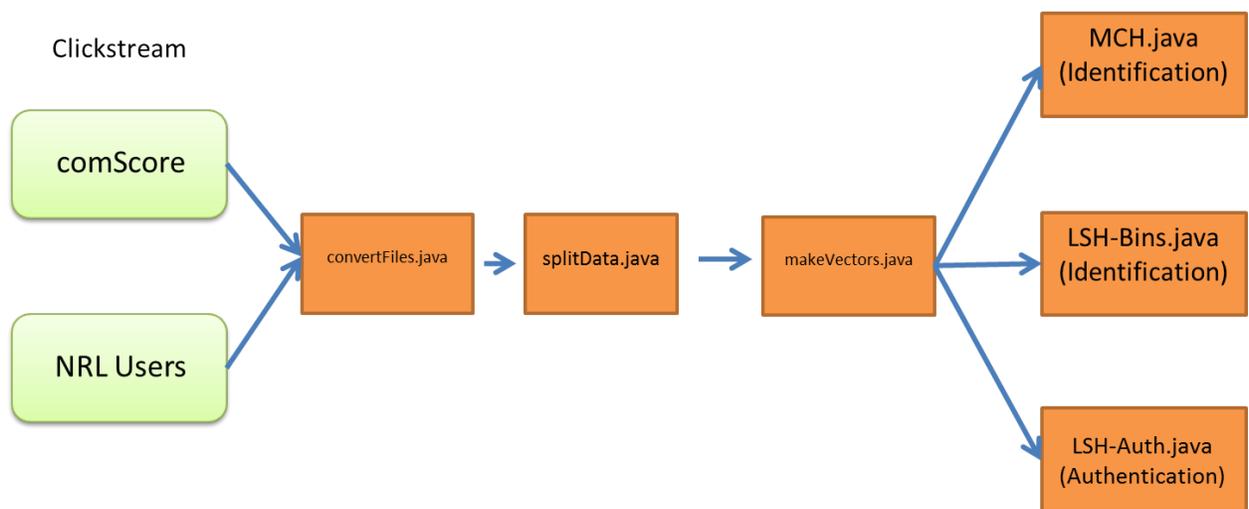


Figure 1: Flowchart demonstrating the path of data.

Algorithm 1 MCH Identification

1: **procedure** TRAINING

Input:

$V \leftarrow$ set of training vectors
 $L \leftarrow$ number of times to repeat
 $w \leftarrow$ window width

Output:

$HashTable \leftarrow$ set of hashes

```
2:   for l in L do
3:     for v in V do
4:        $v \leftarrow$  training vector
5:        $b \leftarrow$  random bias
6:        $K \leftarrow$  k random vectors
7:       for i in K do
8:          $h_i \leftarrow \lfloor \frac{v \cdot i + b}{w} \rfloor$ 
9:        $H \leftarrow (h_1, \dots, h_k)$ 
10:      Return  $HashTable \leftarrow Hash[H]$ 
```

11: **procedure** TESTING

Input:

$HashTable \leftarrow$ set of hashes
 $q \leftarrow$ query vector

Output:

```
 $u \leftarrow$  closest vector to q
12:   $b \leftarrow$  random bias
13:   $K \leftarrow$  k random vectors
14:   $L \leftarrow$  number of times to repeat
15:  for l in L do
16:    for i in K do
17:       $h_i \leftarrow \lfloor \frac{v \cdot i + b}{w} \rfloor$ 
18:     $H \leftarrow (h_1, \dots, h_k)$ 
19:     $curAns \leftarrow Query_{HashTable}closestH$ 
20:   $Answer \leftarrow Mode[curAns]$ 
```

Algorithm 2 Authentication

1: **procedure** TRAINING

Input:

$v \leftarrow$ training vector
 $L \leftarrow$ number of times to repeat
 $w \leftarrow$ window width

Output:

$HashTable \leftarrow$ set of hashes
2: **for** l **in** L **do**
3: $b \leftarrow$ random bias
4: $K \leftarrow$ k random vectors
5: **for** i **in** K **do**
6: $h_i \leftarrow \lfloor \frac{v \cdot i + b}{w} \rfloor$
7: $H \leftarrow (h_1, \dots, h_k)$
8: Return $HashTable \leftarrow Hash[H]$

9: **procedure** TESTING

Input:

$HashTable \leftarrow$ set of hashes
 $q \leftarrow$ query vector
 $u \leftarrow$ provided user training vector

Output:

$b \leftarrow$ whether or not q is u
10: $b \leftarrow$ random bias
11: $K \leftarrow$ k random vectors
12: $p \leftarrow$ acceptable error tolerance
13: **for** i **in** K **do**
14: $h_i \leftarrow \lfloor \frac{v \cdot i + b}{w} \rfloor$
15: $H \leftarrow (h_1, \dots, h_k)$
16: $curAns \leftarrow Distance_{H,u} < p * len(u)$
17: Return $curAns$

4.1 Experimental Study

We obtained web browsing data from two sources – a set of 14 volunteers at the Naval Research Lab, and a set of 42 users provided by comScore - consisting of user id, date, time of day, and website visited. We ran this data through DiffBot¹, a web service that classifies websites into genres based on their visual appearance. This resulted in 26 genres (e.g., article, document, recipe, profile, etc.), including one we added, navy, for sites Diffbot could not categorize but had “navy” in their URL. By pairing these genres with the time of day and day of week that they were accessed, we created fourteen datasets. We temporally split these pairs up in a three to one split, such that one set, referred to as the training set, had 70% of the data while the other, the testing set, had 30%. We used the training set to gather information about the fourteen users, and then tried to match each of the testing data sets to the larger ones.

In order to be able to test our approaches on larger datasets, we also created a random data generator. It generates randomized data sets with specific parameters, such as percent sparsity, percent noise, dimension, and number of vectors.

4.2 Identification

In order to identify users, we first computed the hashes of all of the users’ training vectors. Then, we computed the hash of the given testing vector, using the same random vectors and random offset as before. Now that we had a k -dimensional hash vector for the test user, and n k -dimensional hash vectors for the training users, we had to pick the training hash vector that most resembled this testing hash vector. Each of our vectors was 4368 bits long ($24 \cdot 7 \cdot 26$). We computed 4000 hashes for each vector, with $w = 1.0$, $L = 10$.

We used several different methods to choose the training hash vector that was closest to the given testing hash vector. First, we tried performing a simple nearest neighbor search, by finding the closest training vector to a given testing vector. However, this did not perform

¹<http://www.diffbot.com/>

very well because for most users, the training vectors and testing vectors were very different due to temporal drift (users’ behavior changing over time) – in fact, the training vector closest to a given testing vector was not from the same user. We found that for only 2 out of the 14 NRL users the training vector closest to a user’s testing vector was from the same user. Similarly, for only 15 of the 42 comScore users was the training vector closest to a user’s testing vector from the same user. Due to these large differences between users’ training and testing vectors, the nearest training vector was often not from the same user as a given testing vector.

We then tried using the binning method proposed by other work – we iterated through the k hashes for the given testing vector, and found the m training vectors (of the n total training vectors) that had the same hashes at each position. We then computed the Euclidean distance from the testing vector to each of the m selected training vectors, and chose the training vector associated with the smallest distance. However, due to the large value of k (4000), we found that all 14 or 42 of the training vectors appeared in the m selected vectors. In other words, that $m = n$ for this dataset. We attempted modifying the value of k , and tried a variety of values ranging from 1 to 10000, but we found that with small values of k the method was not very accurate, as the correct answer sometimes did not appear in the m selected vectors. On the other hand, when k was sufficiently large to accurately identify users, all n vectors were selected, and we ended up having to compute the distance between the k dimensional hash vector for the test user and all n training vectors.

Due to this, we decided to skip the binning step and simply compute the Euclidean distance from the testing hash vector to each training hash vector. While this method achieved a respectable accuracy (71% for the comScore set, and 64% for the NRL set), we noticed that the algorithm got different users wrong each time. As a result of this, we tried using repeated trials – running the algorithm L times, and taking the mode of the L answers provided. We named this method Mode Closest Hashing (MCH). We calculated that this method has a testing complexity of $O(L \cdot k \cdot d)$. By performing MCH, we achieved an excellent

Table 1: Identification Results for NRL User Dataset

Method	k	w	L	Accuracy (%)
Nearest Neighbor	–	–	–	14.3%
Bins	1000	10.0	100	57.1%
MCH	4000	1.0	10	92.9%

Table 2: Identification Results for comScore User Dataset

Method	k	w	L	Accuracy (%)
Nearest Neighbor	–	–	–	35.7%
Bins	1000	10.0	100	78.6%
MCH	400	1.0	10	95.2%

accuracy that could not have been achieved with a simple nearest neighbor approach (Table 1, Table 2.).

4.3 Authentication

The problem of authenticating users is slightly different than that of identifying users. Specifically, there are two main differences – firstly, the algorithm is only trained with the data for one user; its job is to decide if a given test vector is from that user. Secondly, there needs to be a “reject option” – the algorithm needs a way to indicate that the given testing user is not one of the training users.

We decided to continue with our method of distances (computing the distance from a given test vector to training vectors). We first computed the k -dimensional hash vector for a training user vector. We then computed the k -dimensional hash vector for a testing user vector. We found the distance between these two vectors, and if it was less than some threshold, said that the two vectors were from the same user, “accepting” the testing vector. If not, we said they were not from the same user, “rejecting” the testing vector.

By modifying the value of the threshold, we could calculate the false acceptance rate (FAR, when a different test vector from the training vector was thought to be from the same user), and the false rejection rate (FRR, when a test vector from the same user as the

Table 3: Authentication Results

NRL Users	comScore Users
EER: 26%	EER: 24%
Accuracy: 74%	Accuracy: 76%

training vector was thought to be from a different user). These two metrics share an inverse correlation – a decrease in one leads to an increase in the other. The goal of authentication is to minimize both of these metrics. We can compute the accuracy from the FAR and FRR by first computing the equal error rate (equal error rate), where the FAR and FRR are equal, and then subtracting this from 1.

$$Accuracy = 1 - EER$$

This method has a similar accuracy as that of other methods (Figure 2, Figure 3, Table 3).

4.4 Randomized Datasets

To validate our approach, we also generated and analyzed random data sets. Specifically, we generated data with $n = 100$, $d = 15,625$, a percent sparsity of 30%, and a noise percentage of 15%. We found that the both the bins approach and the MCH approach achieved a perfect accuracy (100%) for both identification and authentication. By performing further analysis on this dataset, we determined that both these methods work well when the correct answer is the same as the nearest neighbor. Specifically, we noticed that while the level of noise in the two user datasets (15.1% and 5.3% for the comScore and NRL User datasets, respectively), the percent sparsity, defined as the number of zeros in the training set, was very high (80.3% and 93.4% for the comScore and NRL User datasets, respectively). Due to this, the percent noise was not actually acting on the whole vector, but rather on the portion of the vector that had information. We thus defined the true noise in a dataset as



Figure 2: NRL Users ROC Curve



Figure 3: comScore Users ROC Curve

$$TrueNoise = \frac{Noise}{1 - Sparsity}.$$

We determined the true noise to be 76.6% and 80.3% for the comScore and NRL User datasets, respectively, and 21.4% in the randomized dataset.

5 Conclusion

In this paper we evaluated different LSH algorithms for identifying and authenticating users based on their web browsing behavior. We built upon our previous method of analyzing web browsing behavior based on genre and time of access by adding day of week to the data vectors. While previous research used machine learning techniques, such as recurrent neural networks to attempt to remove noise from the dataset, we attempted to add noise ourselves (through the use of random vectors) to mitigate the noise already present. We concluded that it is indeed possible to identify users based on the genres of webpages they access and the time of day and day of week they access them. Future work could repeat this study with more users, as well as use different representations of the dataset.

6 Acknowledgements

We would like to thank Dr. Myriam Abramson and Mrs. Natalie Haman for their help in completing this paper. We would also like to thank the fourteen anonymous NRL volunteers for providing the data used in this paper.

References

- [1] Myriam Abramson and Shantanu Gore. Associative patterns of web browsing behavior. In *2013 AAAI Fall Symposium Series*, 2013.

- [2] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 459–468. IEEE, 2006.
- [3] Whither Biometrics Committee et al. *Biometric Recognition:: Challenges and Opportunities*. National Academies Press, 2010.
- [4] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *VLDB*, volume 99, pages 518–529, 1999.
- [5] Jeffrey Pang, Ben Greenstein, Ramakrishna Gummadi, Srinivasan Seshan, and David Wetherall. 802.11 user fingerprinting. In *Proceedings of the 13th annual ACM international conference on Mobile computing and networking*, pages 99–110. ACM, 2007.
- [6] Vassil Roussev, Yixin Chen, Timothy Bourq, and Golden G Richard III. `ij` `md5bloomij/ij`: Forensic filesystem hashing revisited. *digital investigation*, 3:82–90, 2006.
- [7] Malcolm Slaney and Michael Casey. Locality-sensitive hashing for finding nearest neighbors [lecture notes]. *Signal Processing Magazine, IEEE*, 25(2):128–131, 2008.