DotHash: Estimating Set Similarity Metrics for Link Prediction and Document Deduplication

Igor Nunes
University of California, Irvine
Irvine, CA, USA
igord@uci.edu

Mike Heddes
University of California, Irvine
Irvine, CA, USA
mheddes@uci.edu

Pere Vergés
University of California, Irvine
Irvine, CA, USA
pvergesb@uci.edu

Danny Abraham
University of California, Irvine
Irvine, CA, USA
dannya1@uci.edu

Alex Veidenbaum
University of California, Irvine
Irvine, CA, USA
alexv@ics.uci.edu

Alex Nicolau
University of California, Irvine
Irvine, CA, USA
nicolau@ics.uci.edu

Tony Givargis
University of California, Irvine
Irvine, CA, USA
givargis@uci.edu

ABSTRACT

Metrics for set similarity are a core aspect of several data mining tasks. To remove duplicate results in a Web search, for example, a common approach looks at the *Jaccard index* between all pairs of pages. In social network analysis, a much-celebrated metric is the *Adamic-Adar index*, widely used to compare node neighborhood sets in the important problem of predicting links. However, with the increasing amount of data to be processed, calculating the exact similarity between all pairs can be intractable. The challenge of working at this scale has motivated research into efficient estimators for set similarity metrics. The two most popular estimators, *MinHash* and *SimHash*, are indeed used in applications such as document deduplication and recommender systems where large volumes of data need to be processed. Given the importance of these tasks, the demand for advancing estimators is evident. We propose DotHash, an unbiased estimator for the intersection size of two sets. DotHash can be used to estimate the Jaccard index and, to the best of our knowledge, is the first method that can also estimate the Adamic-Adar index and a family of related metrics. We formally define this family of metrics, provide theoretical bounds on the probability of estimate errors, and analyze its empirical performance. Our experimental results indicate that DotHash is more accurate than the other estimators in link prediction and detecting duplicate documents with the same complexity and similar comparison time.

CCS CONCEPTS

• **Information systems** → **Near-duplicate and plagiarism detection**; **Social networks**.

KEYWORDS

set similarity, Jaccard index, Adamic-Adar index, MinHash, SimHash, link prediction, document deduplication.

KDD '23, August 6–10, 2023, Long Beach, CA, USA

ACM ISBN 979-8-4007-0103-0/23/08.
https://doi.org/10.1145/3580305.3599314

ACM Reference Format:
Igor Nunes, Mike Heddes, Pere Vergés, Danny Abraham, Alex Veidenbaum, Alex Nicolau, and Tony Givargis. 2023. DotHash: Estimating Set Similarity Metrics for Link Prediction and Document Deduplication. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3580305.3599314

1 INTRODUCTION

Many current challenges—and opportunities—in computer science stem from the sheer scale of the data to be processed [19, 26]. Among these challenges, one of the most outstanding is comparing collections of objects, or simply *sets*[1]. This demand arises, for example, in important problems such as comparing text documents or social media profiles. The challenge is often not the size of each set, but the number of pairwise comparisons over a large dataset of sets. This has motivated research on *estimating* existing set similarity measures, the main subject of this paper.

The search for methods to compare sets of elements is longstanding: more than a century ago, Gilbert [23] and Jaccard [35] independently proposed a measure that is still widely used, known as the *Jaccard index*. The metric is defined as the ratio between the sizes of the intersection and the union of two sets. With the explosion of available data, brought about mainly by the advent of the Web, the Jaccard index has become prevalent as an essential tool in data mining and machine learning. Important applications include information retrieval [55, 63], natural language processing [66, 72], and image processing [48, 57], among several others [9].

Another academic field in which set similarity has become crucial is *network science*. This field studies network representations of physical, biological and social phenomena and is used to understand complex systems in various disciplines [3]. Such networks are modeled using graphs, a mathematical abstraction tool where sets are ubiquitous. Marked by its growing relevance, this area also gave rise to one of the most famous set similarity metrics: the *Adamic-Adar index* [1]. The index was proposed as an alternative to Jaccard for the problem of predicting links, such as friendship or co-authorship, in social networks.

[1]We shall refer to *sets*, indistinctly, as collections with or without repeated elements.

Link prediction is a widely studied problem, with applications in several Web-related tasks, including hyperlink-prediction [74], recommender systems in e-commerce [45], entity resolution [51], and friend recommendation in social networks [15, 21], among others [27]. In this problem, each node is characterized by its set of adjacent nodes, or *neighbors*. The intuition is that nodes of similar neighborhoods tend to become neighbors. The Adamic-Adar index is used to compare these sets of neighbors, but unlike Jaccard, it assigns different weights to neighbors (see Section 2.2). Adamic-Adar is known to be superior to Jaccard for modeling the phenomenon of link emergence in networks in various real-world applications [47, 53]. The success of Adamic-Adar also motivated the emergence of several metrics with different ways of weighting neighbors [7, 16, 49], which will be discussed in Section 2.3.

A second prime example of an application marked by demanding an enormous number of set comparisons is the removal of (near-)duplicate pages in Web search. Eliminating such pages saves network bandwidth, reduces storage costs and improves the quality of search results [52]. In this domain, each document is commonly treated as a set of word sequences. To find duplicates in a corpus of 10 million pages, a relatively small scale for Web applications [52], it would already be necessary to compute the set similarity metric about 50 trillion times. It was precisely in the face of this challenge that the problem of *estimating* set similarity metrics has become highly relevant and has triggered numerous scientific endeavors.

MinHash [4] and *SimHash* [8], the two best-known estimators, were initially developed for the above-mentioned problem and used respectively in the AltaVista and Google Web search engines. Other current applications taking advantage of set similarity estimation include genomic and metagenomic analysis [41, 56], graph comparison [68], collaborative filtering [14], natural language dataset preparation [42], and duplicate detection of other types of data such as images [12]. SimHash is also used in *locality sensitive hashing* (LSH), a technique applied to detect if the similarity of sets exceeds a given threshold, used in problems such as dimensionality reduction, nearest neighbor search, entity resolution and fingerprint matching [44]. An important remark is that, despite being related, LSH and the problem addressed in this paper of estimating metrics directly are distinct and both individually relevant as we will discuss in Section 3. This wide range of relevant applications illustrates the importance and potential of set similarity estimators to push boundaries of problems where they are applied.

Despite the importance of the estimators mentioned above, previous works reveal limitations of these techniques. As is common with estimators, their accuracy is a function of the input as well as the value to be estimated. In the original MinHash paper, Broder [4] indicates that the estimator's accuracy is at its worst when the Jaccard index is around 0.5. Koslicki and Zabeti [41] show that the probability of the estimator deviating from the true value increases exponentially with the difference in size between the sets to be compared. Nonetheless, Shrivastava and Li [65] provides theoretical arguments for the superiority of MinHash over the other baseline, SimHash. More importantly, these techniques are not able to estimate indices like Adamic-Adar. In Section 5, we will show experimentally that this limitation makes them less appropriate for the link prediction and near-duplicate detection problems.

Commendable effort has been devoted to reshaping these techniques to overcome these limitations for specific applications as we discuss in Section 3.3. Nevertheless, given its relevance, we argue that it is also important to pursue alternative techniques that explore new perspectives on the problem. With this in mind, we propose DotHash—a novel set similarity estimator. Like its predecessors, DotHash is based on creating a fixed-size representation of sets (often called a *sketch*, *signature* or *fingerprint* in the literature) that can be compared to estimate the similarity between the sets. Creating these compressed representations introduces preprocessing time, but dramatically mitigates the time for each comparison. The central idea of DotHash is to exploit valuable features of high-dimensional random vector spaces, in particular their *tendency towards orthogonality* [37], to create fixed-dimension sketches while retaining as much information from the original space as possible. In Theorem 2, we show that the cardinality of the intersection of two sets can be estimated, without bias, by a simple *dot product* between their DotHash sketches. As the dot product is such a fundamental operation, we argue that DotHash can take advantage of recent progress in modern hardware platforms for enhanced performance and energy efficiency [54, 71].

In addition to the theoretical contribution of a new baseline framework to the problem of comparing sets in its general formulation, we also show that DotHash has prompt practical relevance. We conducted experiments with several popular datasets for the problems of link prediction and near-duplicate detection, which show that the accuracy obtained by DotHash is higher than that obtained with SimHash and MinHash. For this, we exploit the fact that DotHash is able to estimate a more general family of metrics, which are better adapted to applications, as is the case of Adamic-Adar for link prediction. It is worth noting that the time complexity for each comparison is linear in the size of the DotHash sketches regardless of the metric, since it consists of computing the dot product between them. As previously mentioned, these pair-wise comparisons between sketches dictate the overall time complexity.

2 BACKGROUND

In this section, we introduce the most relevant set similarity metrics for the purpose of our work. We start with the Jaccard and Adamic-Adar indices, for which we show theoretical and empirical results. Then, we provide an overview of other metrics that DotHash is able to estimate directly. It is worth noting that many alternative set similarity metrics have been proposed, specifically tailored to better suit specific scenarios by adapting the basic metrics. For an extensive overview of set similarity metrics we recommend Martínez et al. [53] and Lü and Zhou [50].

2.1 Jaccard

Consider the sets A and B, and let $A \cap B$ denote their intersection and $A \cup B$ their union. The Jaccard index [35] between A and B is defined as:

$$\mathrm{J}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

where the vertical bars denote the cardinality of the enclosed set. This is one of the oldest and most established ways of comparing sets. Over time, numerous adaptations of this simple metric have

emerged, specializing it for particularly interesting applications. Next, we describe one of these adaptations which is widely used, especially in network science [3].

2.2 Adamic-Adar

The Adamic-Adar index was created for the problem of link prediction in social graphs [1]. Let $G = (V, E)$ be a graph, composed of a set of nodes V and a set of edges E. Each $e = (u, v) \in E$ represents an edge (or link) between nodes $u, v \in V$. With $\Gamma(v) \subseteq V$ we denote the subset of nodes that are adjacent to v, i.e., the *neighbors* of v. The cardinality of a neighborhood $|\Gamma(v)|$ is referred to as the *degree* of v.

In the context of graphs, Jaccard can be used to compare pairs of nodes by looking at how many connections they have in common, normalized by the size of the union of their neighborhoods. Adamic-Adar seeks to improve this comparison based on the intuition that the more popular a node in the intersection is (i.e., the higher its degree), the less informative it is about the similarity of the nodes being compared. In the case of social networks, for example, a common connection with a celebrity says little about the chance of two people connecting with each other compared to a less popular mutual friend. To account for this, Adamic-Adar penalizes the number of connections that each shared connection has by taking the logarithm of its degree. Formally, the Adamic-Adar between two nodes is defined as:

$$\mathrm{A}(u, v) = \sum_{x \in \Gamma(u) \cap \Gamma(v)} \frac{1}{\log |\Gamma(x)|}$$

2.3 Link prediction metrics

Given the importance of link prediction, several other metrics have emerged for the purpose of comparing neighborhoods. Similar to Adamic-Adar, these metrics take the local properties of the nodes in the intersection into account. Many of these metrics can also be directly estimated using DotHash. One such example is the Resource Allocation index, used to evaluate the resource transmission between two unconnected nodes through their neighbors which is defined as:

$$\mathrm{RA}(u, v) = \sum_{x \in \Gamma(u) \cap \Gamma(v)} \frac{1}{|\Gamma(x)|}$$

In Section 4.5, we provide a formal description of the family of set similarity metrics that DotHash can directly estimate.

3 RELATED WORK

In this section we describe the two most popular estimators, MinHash and SimHash, which will then be used as baselines for the evaluation of DotHash. It is important to highlight that we compare DotHash with these two methods because they constitute the state of the art for the problem in its most general formulation and are used in current applications as already mentioned.

3.1 MinHash

MinHash [4] is a probabilistic method for estimating the Jaccard index. The technique is based on a simple intuition: if we *uniformly sample* one element x from the set $A \cup B$, we have that:

$$\Pr(x \in A \cap B) = \frac{|A \cap B|}{|A \cup B|} = \mathrm{J}(A, B)$$

which makes the result of this experiment an estimator for Jaccard. However, an important problem remains: *how to uniformly sample from $A \cup B$?* Explicitly computing the union is at least as expensive as computing the intersection, that is, it would be as expensive as calculating the Jaccard index exactly. The main merit of MinHash is an efficient way of circumventing this problem.

Let $h : A \cup B \to \mathbb{N}$ denote a min-wise independent hash function, i.e., for any subset of the domain, the output of any of its elements is equally likely to be the minimum (see Broder et al. [5] for a detailed discussion). Then, we have:

$$\Pr\left(\min_{a \in A} h(a) = \min_{b \in B} h(b)\right) = \frac{|A \cap B|}{|A \cup B|} = \mathrm{J}(A, B)$$

Given the above, the problem of uniformly sampling an element of the union and checking if it belongs to the intersection can be emulated as follows: hash the set elements and check if the smallest value obtained in both sets is the same. Although the result of this random variable is an unbiased estimator of the Jaccard index, its variance is high when the Jaccard is around 0.5. The idea of MinHash is therefore to do k such experiments with independent hash functions and return the sample mean to reduce the variance.

3.2 SimHash

SimHash [8], sometimes indistinctly called *angular LSH*, is another popular estimator of set similarity. The sketches of sets are fixed-length binary vectors and are generated as follows: 1) all elements of the superset S are mapped uniformly to vectors in $\{-1, 1\}^d$; 2) for each set $X \subseteq S$ a d-dimensional vector is created by adding the vectors of its elements; 3) the *SimHash sketch* of the set is a bit string obtained by transforming each positive entry to one and the non-positive entries to zero. The similarity between pairs of sets is then measured by the Hamming distance between these sketches.

Despite being a general estimator for set similarity metrics, SimHash owes its popularity largely to a specific use. Manku et al. [52] showed an efficient way to solve the following problem: *in a collection of SimHash sketches, quickly find all sketches that differ at most k bits from a given sketch, where k is a small integer.* This particular formulation is very useful in the context of duplicate text detection and its efficient solution led to SimHash being used by Google Crawler.

The problem described above is an instance of the problem known as *locality sensitive hashing* (LSH) which, in general, tries to detect pairs of objects whose similarity exceeds a threshold by maximizing the collision probability of the hashes of these objects [32, 33]. Note that LSH is used to group similar objects and the output is binary, i.e. *two objects are either similar or not* [44]. Therefore, we emphasize that the problem of estimating the metrics directly, addressed in this paper, is *different* from LSH. Directly estimating similarity has other possible outcomes, such as ordering pairs by similarity, which is crucial in some applications like query optimization [13] and link prediction as we will discuss in Section 5.

Although SimHash is much more popular in the context of LSH, for the sake of completeness, but underscoring the above, we consider SimHash as a baseline of the general problem as the method was originally proposed by Charikar [8]. Despite this, we make it clear that the other baseline, MinHash, is much more common in the literature for the problem of estimating the actual value of metrics as we discuss in the next section.

3.3 Adjacent research and developments

Our primary focus in this study is to address the task of estimating set similarity metrics in its broadest sense. However, it is important to acknowledge other advancements in the field that are not directly aligned with our specific contribution. These advancements, discussed below, primarily involve enhancements tailored to specific contexts and applications. It is important to note that our method is not intended to directly compete with these notable developments in each individual application, but rather aims to serve as a new baseline for the general problem. We emphasize, however, that DotHash also brings practical contributions to the state of the art. This is achieved by supporting a wider range of metrics, which is formally defined in Section 4.5. As we will show in Section 5, this allows for greater accuracy in important problems such as link prediction and document deduplication.

While MinHash remains the standard framework for estimating the actual value of set similarity metrics, several techniques have been proposed to enhance its accuracy and efficiency in specific application contexts. For instance, Chum and Matas [11] propose an efficient method to compute MinHash sketches for image collections using inverted indexing. Another technique, introduced by Koslicki and Zabeti [41], employs Bloom filters for fast membership queries and is known as *containment MinHash*. They demonstrate the superiority of this technique in metagenomic analysis by more accurately estimating the Jaccard index when dealing with sets that significantly differ in size.

Several other works have focused on a variant of the problem that involves estimating the *weighted* Jaccard index [70]. For datasets with sparse data, Ertl [18] and Christiani [10] have explored the concept of *consistent weighted sampling* (CWS) [34] with their respective *BagMinHash* and *DartMinHash* techniques. Conversely, when dealing with dense data, methods based on *rejection sampling* have been demonstrated to be more efficient [46, 64].

Another important recent endeavor has been to develop LSH techniques based on deep learning, known as "learn to hash" methods. These include *Deep Hashing* [73] and various others [17, 28, 40]. In general, these techniques do not estimate any particular metric directly, but seek to create sketches that allow for approximate nearest neighbor search. Another key distinction lies in the methodology employed by these approaches. They rely on constructing trained models through annotated data, where the concept of similarity is derived from the mapping of training examples to a specific target. Consequently, this similarity may not necessarily extend to other datasets, making it potentially non-generalizable. In contrast, the methods discussed in this paper estimate the similarity between two sets solely based on the sets themselves.

Although "learn to hash" methods have demonstrated promising accuracy in situations where supervised learning is viable, their broader adoption has also been hindered by other challenges. These limitations encompass high costs associated with training and inference, the inherent unpredictability due to unknown bounds in estimation error, and their high sensitivity to data distribution, often concealed by the reliance on purely empirical assessments [39, 69]. As a result, traditional methods such as MinHash and SimHash continue to be utilized in important applications such as the ones mentioned earlier. Despite the inherent differences and the difficulty of setting up an accurate and unbiased study that delves deep into both approaches, we believe that a comparative study between these traditional methods and learning-based approaches would yield significant value for the scientific community.

4 DOTHASH

We begin by describing a simple method to compute the cardinality of the intersection between two sets. This provides the basis from which we describe the intuition for DotHash. The intuition builds on a generalization of the simple method and a subtle feature of high-dimensional vector spaces. From it, we show how we can create an estimator for the intersection cardinality. We emphasize that virtually all set similarity metrics are direct functions of the intersection cardinality, combined with other easily obtained quantities such as the size of the sets [53]. The fact that DotHash estimates the intersection cardinality directly makes it naturally extendable to all these metrics. One of the few exceptions is a family of metrics that assign different weights to the intersection items, such as the Adamic-Adar index. We conclude this section showing how DotHash can be adapted to estimate this larger family of metrics as well, being the first estimator to enable this.

4.1 Computing the intersection size

A common way of representing sets is by using bit strings [20, 61]. In this representation, an arbitrary order $[x_1, x_2, \ldots, x_d]$ of the elements of the superset S is established. Then, each set $X \subseteq S$ is represented by an $|S|$-dimensional binary vector whose i-th bit is one if $x_i \in X$, and zero otherwise. Table 1 illustrates this representation for sets $A = \{a, b, c, d\}$ and $B = \{b, c, d, e\}$. This representation is especially common for graphs, where it is called an *adjacency matrix*, and each set consists of the neighborhood of a node.

Table 1: Bit string representation of sets A and B

	a	b	c	d	e
A	1	1	1	1	0
B	0	1	1	1	1

It is easy to observe that the size of the intersection between A and B is given by the number of columns where both elements are one. This provides a straightforward way to get the size of the intersection of sets: calculate the *dot product* between their bit strings [6, 20].

4.2 Generalization to orthogonal vectors

An alternative way of visualizing the set bit strings, important for the generalization we propose, is: consider that each *element* $x_i \in S$

is encoded by an $|S|$-dimensional vector of value one in position i, and zero elsewhere. This representation is usually called *one-hot encoding*. From this, we can define the bit string of a set as *the sum of its one-hot encoded elements*, as illustrated in Figure 1.

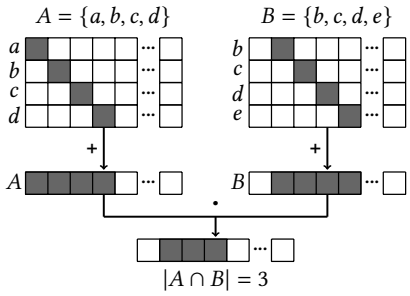

Figure 1: Intersection calculation using one-hot encoding.

In Theorem 1, we show that the dot product results in the intersection of sets not only when they are the sum of one-hot encoded elements, but more generally when we encode the elements using any orthonormal basis of $\mathbb{R}^{|S|}$, of which one-hot is a particular case—the *standard basis*. Although this, arguably trivial, generalization alone may not seem advantageous at this point, in the next section we show how it is fundamental in the transition from exact to estimation with our method.

Theorem 1. *Consider arbitrary sets $A, B \subseteq S$. Let $\phi : S \to \mathbb{R}^{|S|}$ be any injective map of their elements to vectors to one of the orthonormal bases of $\mathbb{R}^{|S|}$, and let*

$$\boldsymbol{a} = \sum_{a \in A} \phi(a) \quad \text{and} \quad \boldsymbol{b} = \sum_{b \in B} \phi(b).$$

Then, $\boldsymbol{a} \cdot \boldsymbol{b} = |A \cap B|$.

While the above yields a way of representing sets so that we can compute intersection sizes by simple dot products, notice that each set is represented using $|S|$ bits, resulting in a time and space complexity of $O(|S|)$. Although this can be useful in certain scenarios, clearly this method becomes prohibitively expensive for very large supersets S, for example, in the large scale applications described in the previous sections. Another problem is that in many real applications, such as those related to social networks, the sets change over time, so there is no way to establish the superset size a priori.

In some cases the time and space complexity can be improved to $O(|A| + |B|)$ by restricting $\phi(\cdot)$ to the standard basis encoding, represented using a sparse vector format so that only the non-zero elements are stored. With this modification the dot product can be computed by iterating over both vectors at once, similar to merging lists. However, because of the overhead of sparse vector representations, this is mainly useful when $|A| + |B| \ll |S|$.

There are still limitations to the sparse vector improvement, especially when even $|A| + |B|$ is very large. This happens both in social networks where nodes can have more than a million connections, and in document deduplication where large documents can hold many word sequences. In the next section we present a method to improve the time complexity to a constant value by giving up the exact intersection size in favour of an estimate.

4.3 Exploiting *quasi*-orthogonality

The method described above seems unsuitable for large scale applications as it requires a number of dimensions equal to the size of the superset. This constraint is imposed by the fact that the smallest real vector space with $|S|$ orthogonal vectors is $\mathbb{R}^{|S|}$. Intuitively, we need orthogonality so that $\phi(x) \cdot \boldsymbol{a} = 1$ if, and only if, $x \in A$, and zero otherwise. This ensures that $\boldsymbol{a} \cdot \boldsymbol{b} = |A \cap B|$ (for a detailed discussion see the proof of Theorem 1 in the Appendix A.1). From an information theory perspective, this guarantees a lossless representation of the sets by the sum of the encoded elements, since by the above operation we can verify exactly which elements make up the set.

Our proposed estimator, DotHash, relies on a very interesting property of high-dimensional vector spaces: *uniformly sampled vectors are nearly orthogonal*, or *quasi*-orthogonal, *to each other with high probability* [38]. This valuable feature has been explored in several other domains, especially to model human cognition and memory [22, 37]. It is based on this insight that DotHash turns the above method into an estimator for the size of the intersection of sets.

Instead of using a *precisely* orthonormal basis of vectors of $\mathbb{R}^{|S|}$ to encode the elements of S, DotHash uses unit vectors sampled from $\mathbb{R}^d$, with $d < |S|$. The set *sketches* (fixed-length representations) are then built in the same way, by adding the encodings of their elements, as depicted in Figure 2. Intuitively from the above, each encoded element would be quasi-orthogonal to all others, allowing to approximate the dot product relations mentioned above. In Theorem 2 we formalize this idea, showing that the the dot product between the DotHash sketches of sets is an unbiased estimator for their intersection cardinality.

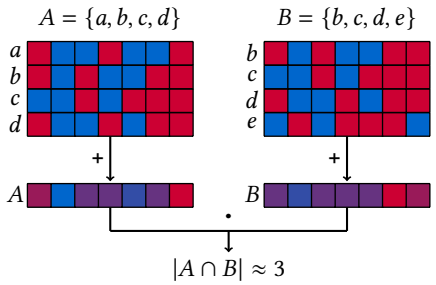

Figure 2: Intersection calculation using quasi-orthogonal encoding.

Theorem 2. *Consider arbitrary sets $A, B \subseteq S$ and any constant $d \in \mathbb{N}^+$. Let $\psi : S \to \mathbb{R}^d$ be a uniform random mapping of elements in S to unit vectors which are the vertices of a d-dimensional hypercube, and let*

$$\boldsymbol{a} = \sum_{a \in A} \psi(a) \quad \text{and} \quad \boldsymbol{b} = \sum_{b \in B} \psi(b).$$

Then, $\mathbb{E}[\boldsymbol{a} \cdot \boldsymbol{b}] = |A \cap B|$, and

$$\mathrm{Var}(\boldsymbol{a} \cdot \boldsymbol{b}) = \frac{1}{d}\Big(|A||B| + |A \cap B|^2 - 2|A \cap B|\Big)$$

Using the variance provided in Theorem 2 and the Chebychev inequalty we can bound the probablity of error by:

$$\Pr(|X-\mu| \geq \epsilon\mu) \leq \frac{\mathrm{Var}(\boldsymbol{a}\cdot\boldsymbol{b})}{(\epsilon|A\cap B|)^2}$$

If we use the observation that each dimension can be interpreted as an independent sample, we can use the Central Limit Theorem (CLT) to approximate the probability of error as follows:

$$\lim_{d\to\infty}\Pr(|X-\mu|\geq\epsilon\mu) = 2\left(1-\Phi\left(\frac{\epsilon|A\cap B|}{\sqrt{\mathrm{Var}(\boldsymbol{a}\cdot\boldsymbol{b})}}\right)\right)$$

where $\Phi(\cdot)$ denotes the standard normal cumulative distribution function. In Figure 3 we provide the CLT estimate (solid line) and the empirical probability (dashed line).

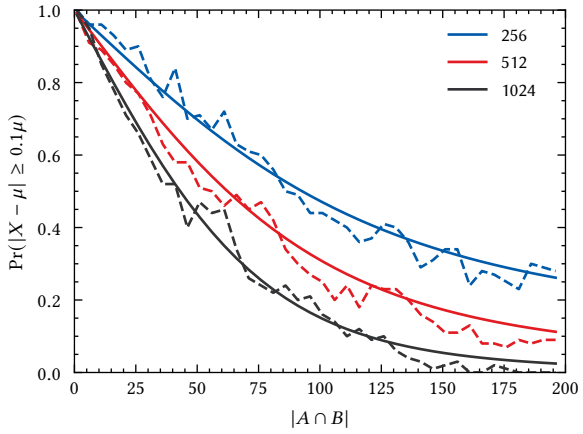

Figure 3: Intersection estimate bounds for DotHash. With $|A| = |B| = 200$ and $d = \{256, 512, 1024\}$. Dashed lines indicate experimental results.

We can rewrite the CLT (or the Chebychev inequality) to get the required number of dimensions d to obtain an error greater or equal to $\epsilon|A\cap B|$ with a given probability p:

$$d \approx \mathrm{Var}(\boldsymbol{a}\cdot\boldsymbol{b})\left(\frac{\Phi^{-1}\left(1-\frac{p}{2}\right)}{\epsilon|A\cap B|}\right)^2$$

where $\Phi^{-1}(\cdot)$ denotes the standard normal percent point function.

4.4 Estimating Adamic-Adar

Now that we have established a method for estimating the size of the intersection, we describe how to adapt DotHash to estimate the Adamic-Adar index. The idea starts from the fact that $\boldsymbol{a}$ and $\boldsymbol{b}$ are sums of the vectors that encode the elements of their respective sets. Given this and the distributive property of the dot product over addition, we have:

$$\boldsymbol{a}\cdot\boldsymbol{b} = \sum_{a\in A}\sum_{b\in B}\psi(a)\cdot\psi(b)$$

Observing that $\mathbb{E}[\psi(a)\cdot\psi(b)] = 1$ if $a = b$, and zero otherwise (see the proof of Theorem 2):

$$\mathbb{E}[\boldsymbol{a}\cdot\boldsymbol{b}] = \sum_{a\in A}\sum_{b\in B}\mathbb{E}[\psi(a)\cdot\psi(b)] = \sum_{x\in A\cap B}1$$

The right-hand side of this equation is similar to Adamic-Adar in that both sum values over the intersection items. The key missing part is that the value to be summed must be a function of the size of the neighborhoods, not a constant.

In the above case, the summation parameter is one because every element is encoded to a unit vector. However, the construction of DotHash allows us to adjust the summation parameter by modifying the magnitude of the vectors used to represent each element. To obtain the Adamic-Adar index, we want each element, in this case each *node*, to be encoded in such a way that:

$$\psi(v)\cdot\psi(v) = \frac{1}{\log|\Gamma(v)|}$$

Theorem 3 shows how to adapt the vector magnitudes to obtain the Adamic-Adar index.

Theorem 3. *Consider an arbitrary graph $G = (V, E)$ with nodes V and edges E. Take any constant $d \in \mathbb{N}^+$ and let $\Gamma(v) \subseteq V$ denote the neighbors of node $v \in V$. Let $\psi : V \to \mathbb{R}^d$ be a uniform random mapping of nodes in V to unit vectors which are the vertices of a d-dimensional hypercube, and let*

$$\boldsymbol{u} = \sum_{x\in\Gamma(u)}\psi(x)\sqrt{\frac{1}{\log|\Gamma(x)|}}$$

and

$$\boldsymbol{v} = \sum_{y\in\Gamma(v)}\psi(y)\sqrt{\frac{1}{\log|\Gamma(y)|}}.$$

Then, $\mathbb{E}[\boldsymbol{u}\cdot\boldsymbol{v}] = \mathrm{A}(u, v)$.

4.5 General family of supported metrics

Building upon the result presented in the previous section, we naturally extend it to encompass a general formulation of all set similarity metrics that can be directly estimated using DotHash. This family includes all metrics of the form: $\sum_{x\in A\cap B} f(x)$, where $f : S \to \mathbb{R}$ is any function on intersection elements. Besides the item x, the function f can use any global parameters such as the cardinalities of A, B or S. The DotHash sketches for this general set similarly metric are given by:

$$\boldsymbol{a} = \sum_{a\in A}\psi(a)\sqrt{f(a)} \quad \text{and} \quad \boldsymbol{b} = \sum_{b\in B}\psi(b)\sqrt{f(b)}$$

and the estimate is obtained by $\boldsymbol{a}\cdot\boldsymbol{b}$. Observe that the intersection size, the Adamic-Adar index, and the Resource Allocation index all fit into this general framework: the intersection size corresponds to $f(x) = 1$, Adamic-Adar to $f(x) = \frac{1}{\log|\Gamma(x)|}$, and Resource Allocation to $f(x) = \frac{1}{|\Gamma(x)|}$. This group of metrics directly supported by DotHash includes the majority of metrics listed in Martínez et al. [53] and Lü and Zhou [50].

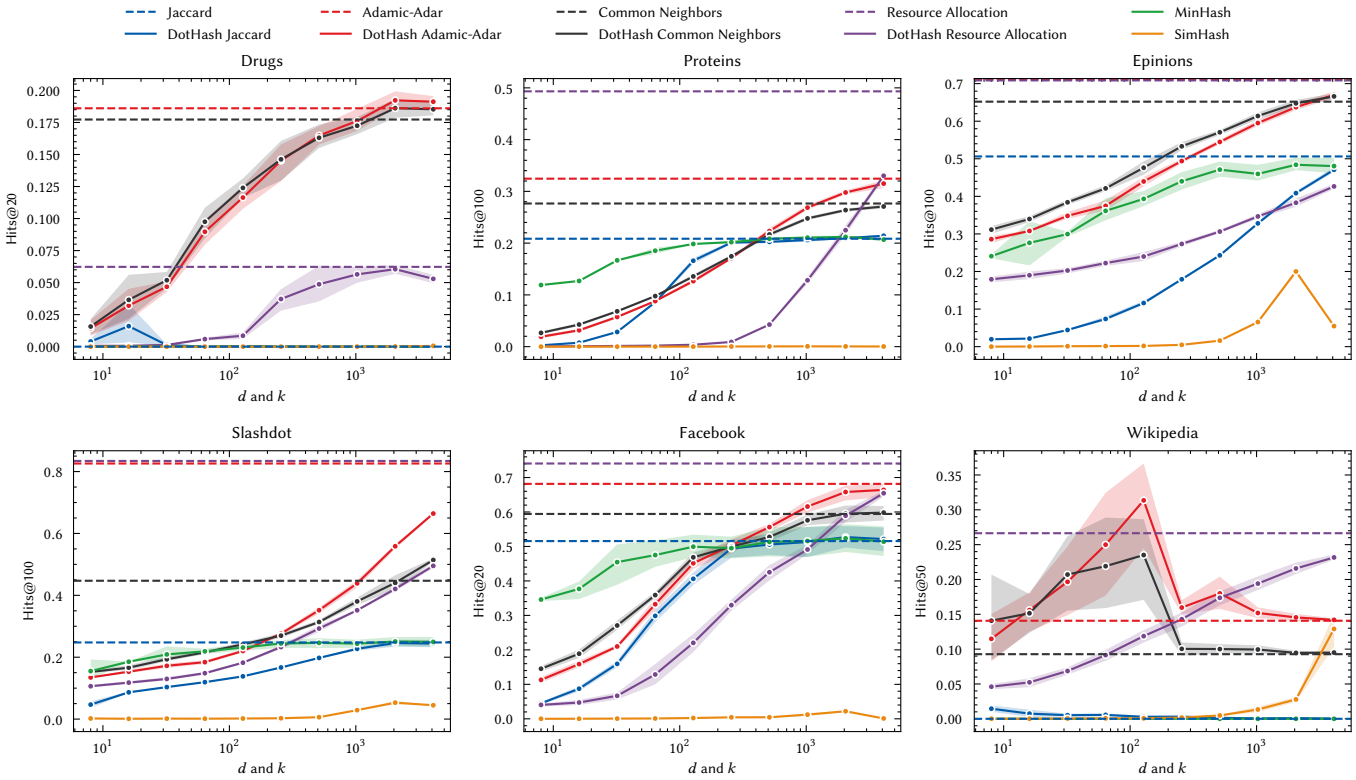

Figure 4: Link prediction accuracy results while varying the number of dimensions d and hashes k.

5 EXPERIMENTS

In this section we present experiments comparing DotHash to the baselines presented in Section 3. The main goal here is to provide empirical evidence on the advantages of DotHash in the link prediction and duplicate detection tasks. All the methods were implemented using PyTorch [58] and the Torchhd library [29], and ran on a machine with 20 Intel Xeon Silver 4114 CPUs, 93 GB of RAM and 4 Nvidia TITAN Xp GPUs. The experiments, however, only used a single CPU or GPU. We repeated each experiment 5 times on the CPU and 5 times on the GPU. The code is available at: https://github.com/mikeheddes/dothash.

Table 2: Statistics of the graph datasets

Dataset	Nodes	Edges	Median degree
Drugs	4,267	1,334,889	446
Wikipedia	11,631	341,691	13
Facebook	22,470	342,004	7
Epinions	75,879	508,837	2
Slashdot	82,168	948,464	6
Proteins	576,289	42,463,862	43

5.1 Datasets

An overview of the datasets used is shown in Table 2. To compare the methods under different circumstances, we consider a range of common benchmarks used in the literature that have different characteristics and are associated with different applications. For the link-prediction task we evaluate each method on the following datasets:

- Drugs [24]: This dataset represents the interaction between drugs, where the joint effect of using both drugs is significantly different from their effects separately.
- Wikipedia [62]: This dataset represents a webpage network where each node represents a web page and the edges represent hyperlinks between them.
- Facebook [62]: A network of verified Facebook pages where nodes correspond to Facebook pages and edges are the mutual likes between pages.
- Proteins [67]: Is a protein network where nodes represent proteins from different species and edges show biological meaningfulness between the proteins associations.
- Epinions [59]: Represents the who-trusts-whom social network of the general consumer review site Epinions.com, where each node represents a user, and each edge is a directed trust relation.
- Slashdot [43]: Represents the Slashdot social network as of February 2009, where each node is a user and each edge is a directed friend/foe link.

For the document deduplication we use the following datasets:

- CORE Deduplication Dataset 2020 [25]: This dataset consists of more than 1.3M scholar documents labeled as duplicates or non-duplicates.
- Harvard Dataverse Duplicate Detection Restaurant dataset [2]. This dataset consists of a collection of 864 restaurant records containing 112 duplicates.
- Harvard Dataverse Duplicate Detection cddb dataset [2]: This dataset contains a set of 9,763 records with 299 duplicated entries, with each row representing information about a particular audio recording.

5.2 Link prediction

This experiment aims to demonstrate a practical advantage of DotHash. While the baseline estimators, MinHash and SimHash, are limited to estimating the Jaccard index, DotHash offers the ability to estimate more complex metrics, known to be more effective for certain applications, including link prediction. We evaluate the accuracy of each estimator in solving the link prediction problem. The problem consists of inferring which links will appear in the future, called the *inference time interval*, given a snapshot of a graph [47]. In practice, the task is seen as a ranking problem for pairs of nodes, i.e., the approaches compare pairs of nodes and predict that the most similar pairs are those that are likely to connect in the future.

The quality of the methods is evaluated based on how well they rank pairs that effectively form in the inference time interval, against random pairs that do not connect. These edges are respectively called the positive and negative samples. The most popular metric, Hits@K, counts the ratio of positive edges that are ranked K or above the negative edges [31]. The Jaccard, Adamic-Adar, Common Neighbors, and Resource Allocation indices are all used in the literature for establishing this ranking.

The results, presented in Figure 4, provide evidence to substantiate the claim that estimating more appropriate metrics makes DotHash a better estimator for the link prediction problem. By employing sufficient dimensions and selecting suitable metrics, DotHash consistently outperforms the baselines across all datasets and approaches the exact indices, shown in dashed lines. Each solid line represents the mean of the values observed in the five repetitions, and the corresponding colored shades show the 95% confidence interval. Importantly, as explained in the previous section, the adoption of DotHash does not impose a significant additional computational burden compared to the baseline methods. Figure 5 shows the normalized execution time of each method on the same datasets. For more details on execution times, please refer to Appendix B.

5.3 Document deduplication

The detection of duplicate documents was the first major application to motivate the development of set similarity estimators. Both MinHash and SimHash were developed and became popular for their use in deduplicating web pages in Google and Alta Vista search engines. With our experiments in this section we seek to show how DotHash compares to these methods in this important problem.

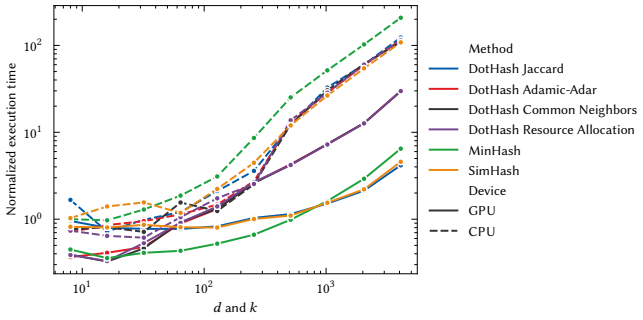

Figure 5: Normalized average execution time of different methods, relative to MinHash with $k = 8$ running on CPU. The average is calculated over all link prediction datasets.

Once again, we took advantage of the broader capability of DotHash to estimate a metric richer than the Jaccard coefficient between documents. While MinHash and SimHash give equal weight to shingles (sequences of words) when comparing documents, with DotHash we can assign different weights to reflect how important each shingle is to each document in the corpus. Our intuition is that the more important a term is to identify each text, the more its presence in different texts indicates their similarity.

One of the most popular ways of evaluating how important a term is to a document is by the *inverse document frequency*, or IDF [36]. The measure is widely used in the information retrieval literature for text mining [30, 60], and is defined as:

$$\mathrm{idf}(x) = \log \frac{|D|}{|\{d \in D : x \in d\}|}$$

where $|D|$ is the number of documents in the entire corpus and $|\{d \in D : x \in d\}|$ the number of documents that contain the term x. Given this, we can compare documents A and B not only by the number, but also by the importance of common shingles, as follows:

$$\mathrm{sim}_{\mathrm{idf}}(A, B) = \sum_{x \in A \cap B} \mathrm{idf}(x)$$

which is in the family of functions that DotHash can estimate.

In Table 3 we show the comparative results between the three estimators for the near-duplicate detection problem in the three different datasets described in Section 5.1. For each estimator we present the near-duplicity detection accuracy in terms of Hits@25 and execution time in seconds. The number of hash values and dimensions for MinHash, SimHash, and DotHash were set to 128, 500, and 10,000, respectively. These values were chosen to ensure comparable accuracy and enable the observation of differences in computational efficiency between the algorithms.

The numerical results presented indicate that DotHash is able to surpass the accuracy of MinHash in all datasets, even with a number of dimensions in which its execution time is between 0.5 and 3× faster. The same is observed in the comparison with SimHash, which obtains the lowest accuracy in all cases. These empirical results reinforce the findings observed in the link prediction experiments, highlighting the advantage of DotHash. By efficiently estimating richer metrics through a single dot product computation between

set sketches, DotHash consistently delivers superior cost-benefit compared to other estimators.

Table 3: Accuracy and computation time (in seconds) results in the detection of duplicate documents.

Dataset	Metric	MinHash	SimHash	DotHash
CORE 20'	Hits@25	0.6246	0.3991	**0.6286**
	Time	0.0269	0.0284	**0.0088**
Rest	Hits@25	0.9598	0.7745	**0.9819**
	Time	0.0014	0.0011	**0.0006**
cddb	Hits@25	0.9058	0.6496	**0.9085**
	Time	0.0094	0.0084	**0.0066**

6 CONCLUSION

We propose DotHash, a new baseline method for estimating the similarity between sets. The method takes advantage of the tendency to orthogonality of sets of random high-dimensional vectors to create fixed-size representations for sets. We show that a simple dot product of these sketches serves as an unbiased estimator for the size of the intersection of sets. DotHash allows estimating a larger set of metrics than existing estimators. Our experiments show that this makes it more appropriate for link prediction and duplicate detection tasks. Adding the theoretical and practical contribution, we see DotHash as a new framework for a problem of increasing relevance in data mining and related areas.

REFERENCES

[1] Lada A Adamic and Eytan Adar. 2003. Friends and neighbors on the web. *Social networks* 25, 3 (2003), 211–230.

[2] Abdulrazzak Ali. 2022. Duplicates Detection. https://doi.org/10.7910/DVN/O7LNDH

[3] Albert-László Barabási. 2013. Network science. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 371, 1987 (2013), 20120375.

[4] Andrei Z Broder. 1997. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*. IEEE, 21–29.

[5] Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. 2000. Min-wise independent permutations. *J. Comput. System Sci.* 60, 3 (2000), 630–659.

[6] Michael Brusco, J Dennis Cradit, and Douglas Steinley. 2021. A comparison of 71 binary similarity coefficients: The effect of base rates. *Plos One* 16, 4 (2021), e0247751.

[7] Carlo Vittorio Cannistraci, Gregorio Alanis-Lobato, and Timothy Ravasi. 2013. From link-prediction in brain connectomes and protein interactomes to the local-community-paradigm in complex networks. *Scientific Reports* 3, 1 (2013), 1–14.

[8] Moses S Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM Symposium on Theory of computing (STOC)*. 380–388.

[9] Seung-Seok Choi, Sung-Hyuk Cha, and Charles C Tappert. 2010. A survey of binary similarity and distance measures. *Journal of Systemics, Cybernetics and Informatics* 8, 1 (2010), 43–48.

[10] Tobias Christiani. 2020. DartMinHash: Fast Sketching for Weighted Sets. *arXiv preprint arXiv:2005.11547* (2020).

[11] Ondřej Chum and Jiří Matas. 2012. Fast computation of min-hash signatures for image collections. In *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 3077–3084.

[12] Ondrej Chum, James Philbin, Andrew Zisserman, et al. 2008. Near duplicate image detection: Min-hash and TF-IDF weighting.. In *The British Machine Vision Conference (BMVC)*, Vol. 810. 812–815.

[13] Graham Cormode, Minos Garofalakis, Peter J Haas, Chris Jermaine, et al. 2011. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends® in Databases* 4, 1–3 (2011), 1–294.

[14] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. 2007. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th International Conference on World Wide Web (WWW)*. 271–280.

[15] Nur Nasuha Daud, Siti Hafizah Ab Hamid, Muntadher Saadoon, Firdaus Sahran, and Nor Badrul Anuar. 2020. Applications of link prediction in social networks: A review. *Journal of Network and Computer Applications* 166 (2020), 102716.

[16] Yuxiao Dong, Qing Ke, Bai Wang, and Bin Wu. 2011. Link prediction based on local information. In *2011 International Conference on Advances in Social Networks Analysis and Mining*. IEEE, 382–386.

[17] Anshuman Dutt, Chi Wang, Azade Nazi, Srikanth Kandula, Vivek Narasayya, and Surajit Chaudhuri. 2019. Selectivity estimation for range predicates using lightweight models. *Proceedings of the VLDB Endowment* 12, 9 (2019), 1044–1057.

[18] Otmar Ertl. 2018. BagMinHash-minwise hashing algorithm for weighted sets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*. 1368–1377.

[19] Jianqing Fan, Fang Han, and Han Liu. 2014. Challenges of big data analysis. *National Science Review* 1, 2 (2014), 293–314.

[20] Sam Fletcher, Md Zahidul Islam, et al. 2018. Comparing sets of patterns with the Jaccard index. *Australasian Journal of Information Systems* 22 (2018).

[21] Valerio Freschi. 2009. A graph-based semi-supervised algorithm for protein function prediction from interaction maps. In *International Conference on Learning and Intelligent Optimization (LION)*. Springer, 249–258.

[22] Ross W Gayler. 2003. Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience. In *International Conference on Cognitive Science (ICCS)*. 133–138.

[23] Grove Karl Gilbert. 1884. Finley's tornado predictions. *American Meteorological Journal. A Monthly Review of Meteorology and Allied Branches of Study (1884-1896)* 1, 5 (1884), 166.

[24] Emre Guney. 2017. Reproducible Drug Repurposing: When Similarity Does Not Suffice. *Pacific Symposium on Biocomputing* 22, 132–143. https://doi.org/10.1142/9789813207813_0014

[25] Bikash Gyawali, Lucas Anastasiou, and Petr Knoth. 2020. Deduplication of Scholarly Documents using Locality Sensitive Hashing and Word Embeddings. In *Proceedings of 12th Language Resources and Evaluation Conference*. France European Language Resources Association, 894–903.

[26] Reihaneh H Hariri, Erik M Fredericks, and Kate M Bowers. 2019. Uncertainty in big data analytics: survey, opportunities, and challenges. *Journal of Big Data* 6, 1 (2019), 1–16.

[27] Mohammad Al Hasan and Mohammed J Zaki. 2011. A survey of link prediction in social networks. In *Social Network Data Analytics*. Springer, 243–275.

[28] Shohedul Hasan, Saravanan Thirumuruganathan, Jees Augustine, Nick Koudas, and Gautam Das. 2020. Deep learning models for selectivity estimation of multi-attribute queries. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1035–1050.

[29] Mike Heddes, Igor Nunes, Pere Vergés, Dheyay Desai, Tony Givargis, and Alexandru Nicolau. 2022. Torchhd: An Open-Source Python Library to Support Hyperdimensional Computing Research. *arXiv preprint arXiv:2205.09208* (2022).

[30] Monika Henzinger, Bay-Wei Chang, Brian Milch, and Sergey Brin. 2003. Query-free news search. In *Proceedings of the 12th International Conference on World Wide Web (WWW)*. 1–10.

[31] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in Neural Information Processing Systems (NeurIPS)* 33 (2020), 22118–22133.

[32] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing (STOC)*. 604–613.

[33] Piotr Indyk, Rajeev Motwani, Prabhakar Raghavan, and Santosh Vempala. 1997. Locality-preserving hashing in multidimensional spaces. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing (STOC)*. 618–625.

[34] Sergey Ioffe. 2010. Improved consistent sampling, weighted minhash and l1 sketching. In *2010 IEEE International Conference on Data Mining (ICDM)*. IEEE, 246–255.

[35] Paul Jaccard. 1912. The distribution of the flora in the alpine zone. 1. *New Phytologist* 11, 2 (1912), 37–50.

[36] Karen Sparck Jones. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation* (1972).

[37] Pentti Kanerva. 1988. *Sparse distributed memory*. MIT Press.

[38] Pentti Kanerva. 2009. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation* 1, 2 (2009), 139–159.

[39] Kyoungmin Kim, Jisung Jung, In Seo, Wook-Shin Han, Kangwoo Choi, and Jaehyok Chong. 2022. Learned cardinality estimation: An in-depth study. In *Proceedings of the 2022 International Conference on Management of Data*. 1214–1227.

[40] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. 2018. Learned cardinalities: Estimating correlated joins with deep learning. *arXiv preprint arXiv:1809.00677* (2018).

[41] David Koslicki and Hooman Zabeti. 2019. Improving minhash via the containment index with applications to metagenomic analysis. *Appl. Math. Comput.* 354 (2019), 206–215.

[42] Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. 2021. Deduplicating training data makes language models better. *arXiv preprint arXiv:2107.06499* (2021).

[43] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. 2008. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. arXiv:0810.1355 [cs.DS]

[44] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. 2020. *Mining of massive data sets.* Cambridge University Press.

[45] Xin Li and Hsinchun Chen. 2009. Recommendation as link prediction: a graph kernel-based machine learning approach. In *Proceedings of the 9th ACM/IEEE-CS joint conference on Digital Libraries*. 213–216.

[46] Xiaoyun Li and Ping Li. 2021. Rejection sampling for weighted jaccard similarity revisited. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, Vol. 35. 4197–4205.

[47] David Liben-Nowell and Jon Kleinberg. 2007. The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology* 58, 7 (2007), 1019–1031.

[48] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. 2020. Deep learning for generic object detection: A survey. *International Journal of Computer Vision* 128, 2 (2020), 261–318.

[49] Shuxin Liu, Xinsheng Ji, Caixia Liu, and Yi Bai. 2017. Extended resource allocation index for link prediction of complex network. *Physica A: Statistical Mechanics and its Applications* 479 (2017), 174–183.

[50] Linyuan Lü and Tao Zhou. 2011. Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications* 390, 6 (2011), 1150–1170.

[51] Bradley Malin, Edoardo Airoldi, and Kathleen M Carley. 2005. A network analysis model for disambiguation of names in lists. *Computational & Mathematical Organization Theory* 11, 2 (2005), 119–139.

[52] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. 2007. Detecting near-duplicates for web crawling. In *Proceedings of the 16th International Conference on World Wide Web (WWW)*. 141–150.

[53] Víctor Martínez, Fernando Berzal, and Juan-Carlos Cubero. 2016. A survey of link prediction in complex networks. *ACM Computing Surveys (CSUR)* 49, 4 (2016), 1–33.

[54] Ryan Moughan. 2021. *Parallel Architectures for Hyperdimensional Computing.* Ph. D. Dissertation. Master's thesis. University of California at Berkeley.

[55] Suphakit Niwattanakul, Jatsada Singthongchai, Ekkachai Naenudorn, and Supachanun Wanapu. 2013. Using of Jaccard coefficient for keywords similarity. In *Proceedings of the International Multiconference of Engineers and Computer Scientists*, Vol. 1. 380–384.

[56] Brian D Ondov, Todd J Treangen, Páll Melsted, Adam B Mallonee, Nicholas H Bergman, Sergey Koren, and Adam M Phillippy. 2016. Mash: fast genome and metagenome distance estimation using MinHash. *Genome biology* 17, 1 (2016), 1–14.

[57] Rafael Padilla, Sergio L Netto, and Eduardo AB Da Silva. 2020. A survey on performance metrics for object-detection algorithms. In *2020 International Conference on Systems, Signals, and Image Processing (IWSSIP)*. IEEE, 237–242.

[58] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems (NeurIPS)* 32 (2019).

[59] Matthew Richardson, Rakesh Agrawal, and Pedro Domingos. 2003. Trust Management for the Semantic Web. In *The Semantic Web - ISWC 2003*, Dieter Fensel, Katia Sycara, and John Mylopoulos (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 351–368.

[60] Stephen Robertson. 2004. Understanding inverse document frequency: on theoretical arguments for IDF. *Journal of Documentation* (2004).

[61] Kenneth H Rosen. 2012. *Discrete mathematics and its applications.* McGraw-Hill Higher Education.

[62] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. 2019. Multi-scale Attributed Node Embedding. (09 2019).

[63] Athar Sefid, Jian Wu, C Ge Allen, Jing Zhao, Lu Liu, Cornelia Caragea, Prasenjit Mitra, and C Lee Giles. 2019. Cleaning noisy and heterogeneous metadata for record linking across scholarly big datasets. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, Vol. 33. 9601–9606.

[64] Anshumali Shrivastava. 2016. Simple and efficient weighted minwise hashing. *Advances in Neural Information Processing Systems (NeurIPS)* 29 (2016).

[65] Anshumali Shrivastava and Ping Li. 2014. In defense of minhash over simhash. In *Artificial Intelligence and Statistics*. PMLR, 886–894.

[66] Michael Strube and Simone Paolo Ponzetto. 2006. WikiRelate! Computing semantic relatedness using Wikipedia. In *AAAI*, Vol. 6. 1419–1424.

[67] Damian Szklarczyk, Annika L. Gable, David Lyon, Alexander Junge, Stefan Wyder, Jaime Huerta-Cepas, Milan Simonovic, Nadezhda Tsankova Doncheva, John H. Morris, Peer Bork, Lars Juhl Jensen, and Christian von Mering. 2019. STRING v11: protein–protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets. *Nucleic Acids Research* 47 (2019), D607 – D613.

[68] Carlos HC Teixeira, Arlei Silva, and Wagner Meira Jr. 2012. Min-hash fingerprints for graph kernels: A trade-off among accuracy, efficiency, and compression. *Journal of Information and Data Management* 3, 3 (2012), 227–227.

[69] Xiaoying Wang, Changbo Qu, Weiyuan Wu, Jiannan Wang, and Qingqing Zhou. 2021. Are we ready for learned cardinality estimation? *Proceedings of the VLDB Endowment* 14, 9 (2021), 1640–1654.

[70] Wei Wu, Bin Li, Ling Chen, Junbin Gao, and Chengqi Zhang. 2020. A review for weighted minhash algorithms. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 34, 6 (2020), 2553–2573.

[71] Naoya Yamanaka, Takeshi Ogita, Siegfried M Rump, and Shin'ichi Oishi. 2008. A parallel algorithm for accurate dot product. *Parallel Comput.* 34, 6-8 (2008), 392–410.

[72] Wen-tau Yih and Christopher Meek. 2007. Improving similarity measures for short segments of text. In *AAAI*, Vol. 7. 1489–1494.

[73] Han Zhu, Mingsheng Long, Jianmin Wang, and Yue Cao. 2016. Deep hashing network for efficient similarity retrieval. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, Vol. 30.

[74] Jianhan Zhu, Jun Hong, and John G Hughes. 2002. Using markov models for web site link prediction. In *Proceedings of the Thirteenth ACM Conference on Hypertext and Hypermedia*. 169–170.

A THEOREM PROOFS

In this section we provide the proofs of the theorems presented in the main paper.

A.1 Intersection cardinality by dot product

Theorem 1. *Consider arbitrary sets $A, B \subseteq S$. Let $\phi : S \to \mathbb{R}^{|S|}$ be any injective map of their elements to vectors in one of the orthonormal bases of $\mathbb{R}^{|S|}$, and let*

$$\boldsymbol{a} = \sum_{a \in A} \phi(a) \quad \text{and} \quad \boldsymbol{b} = \sum_{b \in B} \phi(b).$$

Then, $\boldsymbol{a} \cdot \boldsymbol{b} = |A \cap B|$.

Proof. By the definitions of $\boldsymbol{a}$ and $\boldsymbol{b}$,

$$\boldsymbol{a} \cdot \boldsymbol{b} = \left(\sum_{a \in A} \phi(a)\right) \cdot \left(\sum_{b \in B} \phi(b)\right)$$

Using the distributivity of the dot product over addition, we can rewrite the above equation as:

$$\boldsymbol{a} \cdot \boldsymbol{b} = \sum_{a \in A} \sum_{b \in B} \phi(a) \cdot \phi(b)$$

Because ϕ is injective, i.e., $\phi(a) = \phi(b)$ if, and only if, $a = b$, and based on the orthogonal property of orthonormal vectors, $\phi(a) \cdot \phi(b) = 1$ if $a = b$, and zero otherwise. Then,

$$\boldsymbol{a} \cdot \boldsymbol{b} = \sum_{a \in A} \sum_{b \in B} \mathbf{1}(a = b) = \sum_{x \in A \cap B} 1 = |A \cap B|$$

□

A.2 Intersection cardinality estimate by dot product

Theorem 2. *Consider arbitrary sets $A, B \subseteq S$ and any constant $d \in \mathbb{N}^+$. Let $\psi : S \to \mathbb{R}^d$ be a uniform random mapping of elements in S to unit vectors which are the points of a d-dimensional hypercube, and let*

$$\boldsymbol{a} = \sum_{a \in A} \psi(a) \quad \text{and} \quad \boldsymbol{b} = \sum_{b \in B} \psi(b).$$

Then, $\mathbb{E}[\boldsymbol{a} \cdot \boldsymbol{b}] = |A \cap B|$, and

$$\mathrm{Var}(\boldsymbol{a} \cdot \boldsymbol{b}) = \frac{1}{d}\Big(|A|\ |B| + |A \cap B|^2 - 2|A \cap B|\Big)$$

Proof. By the definitions of $\boldsymbol{a}$ and $\boldsymbol{b}$,

$$\boldsymbol{a} \cdot \boldsymbol{b} = \left(\sum_{a \in A} \psi(a)\right) \cdot \left(\sum_{b \in B} \psi(b)\right)$$

Applying the linearity of expectation and distributivity of the dot product over addition, we have:

$$\mathbb{E}[\boldsymbol{a} \cdot \boldsymbol{b}] = \mathbb{E}\left[\sum_{a \in A} \sum_{b \in B} \psi(a) \cdot \psi(b)\right] = \sum_{a \in A} \sum_{b \in B} \mathbb{E}[\psi(a) \cdot \psi(b)]$$

Given that ψ maps uniformly to unit vectors which are the points of a d-dimensional hypercube, the elements of $\psi(\cdot)$ are sampled uniformly at random from $\left\{-\frac{1}{\sqrt{d}}, +\frac{1}{\sqrt{d}}\right\}$. When $a = b$,

$$\mathbb{E}[\psi(a) \cdot \psi(b)] = \mathbb{E}[\psi(a) \cdot \psi(a)] = \mathbb{E}\left[\sum_{i=1}^{d} \psi(a)_i^2\right] = \sum_{i=1}^{d} \frac{1}{d} = 1$$

where the right-hand subscripts denote the dimension of the vector. And when $a \neq b$,

$$\mathbb{E}[\psi(a) \cdot \psi(b)] = \mathbb{E}\left[\sum_{i=1}^{d} \psi(a)_i\ \psi(b)_i\right] = \sum_{i=1}^{d} \mathbb{E}[\psi(a)_i\ \psi(b)_i] = 0$$

Thus, $\mathbb{E}[\psi(a) \cdot \psi(b)] = \mathbf{1}(a = b)$, which ensures that:

$$\mathbb{E}[\boldsymbol{a} \cdot \boldsymbol{b}] = \sum_{a \in A} \sum_{b \in B} \mathbf{1}(a = b) = \sum_{x \in A \cap B} 1 = |A \cap B|$$

Moreover, the variance of the estimator is obtained as follows, where $r_i(a, b) = \psi(a)_i\ \psi(b)_i$ and we use the definitions of $\boldsymbol{a}$, $\boldsymbol{b}$, and the dot product,

$$\mathrm{Var}(\boldsymbol{a} \cdot \boldsymbol{b}) = \mathrm{Var}\left(\sum_{i=1}^{d} \sum_{a \in A} \sum_{b \in B} r_i(a, b)\right)$$

Since each dimension of $\psi(\cdot)$ is sampled independently from an identical distribution (i.i.d.),

$$\mathrm{Var}(\boldsymbol{a} \cdot \boldsymbol{b}) = \sum_{i=1}^{d} \mathrm{Var}\left(\sum_{a \in A} \sum_{b \in B} r_i(a, b)\right)$$

We then separate the equal from the non-equal pairs of a and b and note that when $a = b$ their elements are identical thus their outcome has no variance, this gives:

$$\begin{aligned}\mathrm{Var}(\boldsymbol{a} \cdot \boldsymbol{b}) &= \sum_{i=1}^{d} \mathrm{Var}\left(\sum_{x \in A \cap B} \psi(x)_i^2\right) + \mathrm{Var}\left(\sum_{a \in A} \sum_{b \in B \setminus \{a\}} r_i(a, b)\right) \\ &= \sum_{i=1}^{d} \mathrm{Var}\left(\sum_{a \in A} \sum_{b \in B \setminus \{a\}} r_i(a, b)\right)\end{aligned}$$

Using the property of linear combination of random variables we get,

$$\begin{aligned}\mathrm{Var}(\boldsymbol{a} \cdot \boldsymbol{b}) = \sum_{i=1}^{d} \sum_{a \in A} \sum_{b \in B \setminus \{a\}} &\mathrm{Var}(r_i(a, b)) \\ + \sum_{x \in A \setminus \{a\}} \sum_{y \in B \setminus \{x, b\}} &\mathrm{Cov}(r_i(a, b), r_i(x, y))\end{aligned}$$

Lastly, we observe that the covariance is $\frac{1}{d^2}$ when $(a, b) = (y, x)$ giving the variance:

$$\begin{aligned}\mathrm{Var}(\boldsymbol{a} \cdot \boldsymbol{b}) &= \sum_{i=1}^{d} \frac{1}{d^2}\Big(|A|\ |B| + |A \cap B|^2 - 2|A \cap B|\Big) \\ &= \frac{1}{d}\Big(|A|\ |B| + |A \cap B|^2 - 2|A \cap B|\Big)\end{aligned}$$

□

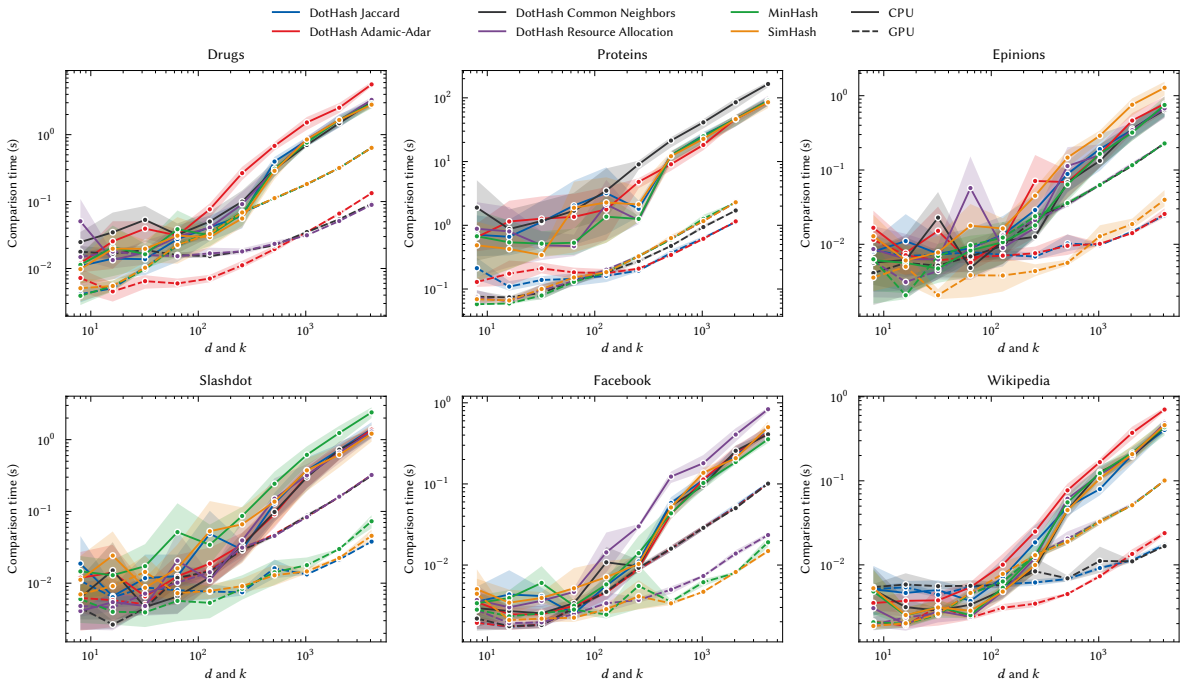

Figure 6: Node signature comparison time in the link prediction tasks.

A.3 Adamic-Adar estimate by dot product

Theorem 3. *Consider an arbitrary graph $G = (V, E)$ with nodes V and edges E. Take any constant $d \in \mathbb{N}^+$ and let $\Gamma(v) \subseteq V$ denote the neighbors of node $v \in V$. Let $\psi : V \to \mathbb{R}^d$ be a uniform random mapping of nodes in V to unit vectors pointing to the corners of a d-dimensional hypercube, and let*

$$\boldsymbol{u} = \sum_{x \in \Gamma(u)} \psi(x) \sqrt{\frac{1}{\log|\Gamma(x)|}} \quad \text{and} \quad \boldsymbol{v} = \sum_{y \in \Gamma(v)} \psi(y) \sqrt{\frac{1}{\log|\Gamma(y)|}}$$

Then, $\mathbb{E}[\boldsymbol{u} \cdot \boldsymbol{v}] = \mathrm{A}(u, v)$.

Proof. By the definitions of $\boldsymbol{u}$ and $\boldsymbol{v}$,

$$\boldsymbol{u} \cdot \boldsymbol{v} = \left(\sum_{x \in \Gamma(u)} \psi(x) \sqrt{\frac{1}{\log|\Gamma(x)|}} \right) \cdot \left(\sum_{y \in \Gamma(v)} \psi(y) \sqrt{\frac{1}{\log|\Gamma(y)|}} \right)$$

Applying the linearity of expectation and distributivity of the dot product over addition, we have:

$$\begin{aligned}
&\mathbb{E}[\boldsymbol{u} \cdot \boldsymbol{v}] \\
&= \mathbb{E}\left[\sum_{x \in \Gamma(u)} \sum_{y \in \Gamma(v)} \left(\psi(x) \sqrt{\frac{1}{\log|\Gamma(x)|}} \right) \cdot \left(\psi(y) \sqrt{\frac{1}{\log|\Gamma(y)|}} \right) \right] \\
&= \sum_{x \in \Gamma(u)} \sum_{y \in \Gamma(v)} \mathbb{E}\left[\left(\psi(x) \sqrt{\frac{1}{\log|\Gamma(x)|}} \right) \cdot \left(\psi(y) \sqrt{\frac{1}{\log|\Gamma(y)|}} \right) \right] \\
&= \sum_{x \in \Gamma(u)} \sum_{y \in \Gamma(v)} \sqrt{\frac{1}{\log|\Gamma(x)|}} \sqrt{\frac{1}{\log|\Gamma(y)|}} \mathbb{E}[\psi(x) \cdot \psi(y)]
\end{aligned}$$

From the previous proof we have that $\mathbb{E}[\psi(x) \cdot \psi(y)] = \mathbf{1}(x = y)$, giving:

$$\begin{aligned}
&\mathbb{E}[\boldsymbol{u} \cdot \boldsymbol{v}] \\
&= \sum_{x \in \Gamma(u)} \sum_{y \in \Gamma(v)} \sqrt{\frac{1}{\log|\Gamma(x)|}} \sqrt{\frac{1}{\log|\Gamma(y)|}} \mathbf{1}(x = y) \\
&= \sum_{x \in \Gamma(u) \cap \Gamma(v)} \frac{1}{\log|\Gamma(x)|} = \mathrm{A}(u, v)
\end{aligned}$$

□

B DETAILED EXPERIMENTAL RESULTS

In this section we present additional experimental results. We believe that the results presented in the main paper are sufficient to substantiate our main claims. Still, the results below should reveal more details to those interested in, for example, exploring the DotHash framework in future work.

B.1 Time to compare node signatures in link prediction

The results in the Figure 6 show the time taken to compare the sketch of the sets created by each method in the link prediction datasets. For each of the methods and datasets, we show the CPU and GPU times for different sketch dimensions d, in the case of DotHash and SimHash, and minwise hash functions k for MinHash. The lines represent the mean over 5 runs of calculating the represented metric for all edges in the test set.