Min-Hash Fingerprints for Graph Kernels: A Trade-off among Accuracy, Efficiency, and Compression

Carlos H. C. Teixeira, Arlei Silva, Wagner Meira Jr.

Computer Science Department
Universidade Federal de Minas Gerais, Brazil
{carlos,arlei,meira}@dcc.ufmg.br

Abstract. Graph databases that emerge from several relevant scenarios (e.g., social networks, the Web) require powerful data management algorithms and techniques. A fundamental operation in graph data management is computing the similarity between two graphs. However, due to the large scale and high dimensionality of real graph databases, computing graph similarity becomes a challenging problem in real settings. Many graph data management tasks, such as graph mining, classification, and retrieval, can be contextualized in the framework of graph kernels. A graph kernel is, roughly speaking, a function that computes the similarity between graph structures as means to enable the application of linear methods to graph data. Nevertheless, large databases usually require the use of compact representations of graphs known as graph fingerprints (or signatures). Graph fingerprinting techniques provide a solution that is a trade-off among accuracy, efficiency, and compression in graph kernels. In this article, we study the problem of generating fingerprints for graph kernels. We introduce a graph fingerprinting technique based on the min-hashing scheme, which is a powerful strategy for computing the similarity between large sets of items using a small amount of data. An algorithm for the generation of graph fingerprints as vectors of min-hash values is presented and integrated into the framework of graph kernels. Results show that graph fingerprinting achieves efficiency gains of up to one order of magnitude with up to 97% space savings when compared against the complete set of graph substructures. Moreover, the proposed technique is up to 9 times more accurate than a baseline method.

Categories and Subject Descriptors: H.2 [**Database Management**]: Database applications; H.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

Keywords: Kernel, Graphs, Indexing, Querying, Databases

1. INTRODUCTION

Graph databases that emerge from several scenarios, such as social networks, bioinformatics, cheminformatics, multimedia content, and the Web, have motivated many challenging problems in Computer Science over the last few years. As a motivational example, chemical repositories, such as PubChem[1] and Zinc[2], contain millions of chemical compounds that can be modeled as graphs. Storing, indexing, querying, and analyzing such large graph databases requires powerful data management technology.

A fundamental operation in graph database management is computing the similarity between two graphs [Bunke and Shearer 1998; Gärtner et al. 2004]. Several relevant graph mining, classification, and retrieval tasks rely on efficient and accurate graph similarity computations. However, due to the large scale of such databases and the high dimensionality that is intrinsic to graph data, computing graph similarity becomes a challenge in many real settings. Various of these tasks that involve graph similarity computations can be contextualized in the general framework of graph kernel methods.

[1]http://pubchem.ncbi.nlm.nih.gov/
[2]http://zinc.docking.org/

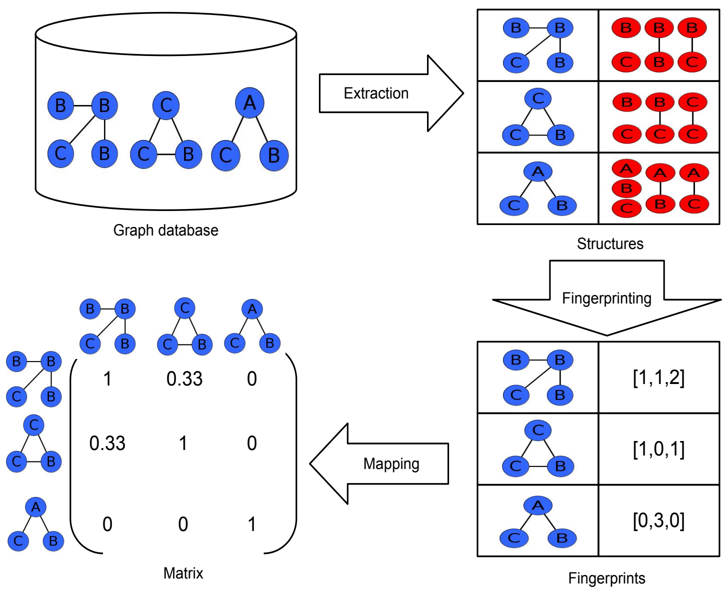


Fig. 1. Illustrative example of a graph kernel, which maps graph structures from a database into a linear space defined by the kernel matrix. This process involves three main steps: (1) extraction, (2) fingerprinting, and (3) mapping. In this article, we propose a new graph fingerprinting method.

A graph kernel [Vishwanathan et al. 2010] is, roughly speaking, a function that computes the similarity between graph structures. Pairwise similarities are represented as an $n \times n$ Gram (or kernel) matrix, where n is the number of graph objects. Figure 1 illustrates the process of mapping a graph database into a kernel matrix with a toy example that will be used through this article. The application of kernel methods to large databases requires an important step known as *fingerprinting* [Ralaivola et al. 2005]. A graph fingerprint (or signature) is a compact descriptor for a graph structure. Effective fingerprints are expected to be accurate, compact, and support efficient similarity computations.

In this work, we study the problem of generating effective graph fingerprints. In particular, we propose a new fingerprinting technique based on *min-hashing* (or min-wise independent locality sensitive hashing) for efficient graph comparision. Min-hashing is a traditional scheme for representing and computing the similarity between large sets of items in general [Cohen et al. 2001]. Graph substructures (e.g., paths, cycles, trees) are examples of large sets that are stored and compared in graph kernel methods. To the best of our knowedge, there are no scientific studies on the effectivity of min-hash values as graph fingerprints.

We propose a very intuitive fingerprinting method based on min-hash values of graph substructures. Min-hashing is an approximate technique with very interesting properties. For instance, its similarity metric is an estimate of the Jaccard similarity. Moreover, previous work [Cohen 1997] have derived upper bounds on the estimated error of this similarity measure. We evaluate the proposed fingerprinting technique using real graph datasets. Our hypothesis is that graph fingerprints based on min-hash values provide a good trade-off among accuracy, efficiency, and compression in real settings. The results obtained in our experimental evaluation strongly support this hypothesis.

The main contribution of this article is the introduction of a graph fingerprinting technique for graph kernel methods. This major contribution may be organized into three key components:

—Proposal of a method for graph fingerprinting based on min-hash values.

—Integration of this method into the larger framework of graph kernels.

—Empirical evaluation of the proposed technique w.r.t. accuracy, efficiency, and compression capacity.

We show the effectivity of min-hash fingerprints in two important graph data management tasks: graph classification and retrieval. The results show that min-hash fingerprints achieve accuracy values that are comparable to the results obtained using a large set of graph substructures with gains up to one order of magnitude and up to 97.5% space savings.

The remaining of this article is organized as follows. Section 2 provides background knowledge on graph kernels. Some definitions and theoretical aspects related to graph kernel methods are covered succinctly, two existing kernel functions are described, and fingerprints are contextualized in the general framework of graph kernels. Section 3 reviews the min-hashing scheme and presents the main contribution of this work, which is a new method for graph fingerprinting based on min-hashing. An extensive empirical evaluation of the proposed fingerprinting technique in terms of accuracy, efficiency and compression power using several real datasets is presented in Section 4. Section 5 covers relevant related work from the literature. Finally, Section 6 provides conclusions and future work.

2. BACKGROUND: GRAPH KERNELS

This section gives a background on graph kernels, which is related to the problem studied in this work. The so called *kernel trick* consists of a data transformation from a non-linear to a linear space so that a linear method can be applied to such data. In this article, we are particularly interested in graph kernels (i.e., mapping graph structures into a linear space). Kernel methods may be applied to various problems [Scholkopf and Smola 2001; Hofmann et al. 2008; Herbrich 2001]. For a detailed coverage on graph kernels, please refer to [Ralaivola et al. 2005; Vishwanathan et al. 2010; Gärtner et al. 2004]. In case the reader is already familiar with graphs, Section 2.1 can be skipped. Moreover, Section 2.2 provides background theory on graph kernels, which is not essential to the understanding of the main contributions of this article. The content of Sections 2.3, 2.4 and 2.5 is of interest of those readers who are not familiar with graph kernels and graph fingerprints.

2.1 Definitions

In this section, we introduce some important definitions related to graph kernels. Such definitions are employed through this article. Graphs can be defined in several forms according to specific application scenarios. In this work, we focus on graphs in an undirected labeled form. This definition is general enough for the analysis of a broad spectrum of real graph databases.

Definition 2.1. (**Undirected labeled graph**) An undirected labeled graph is a 4-tuple $G = (V, E, A, L)$, where V is the set of vertices, E is the set of (undirected) edges, A is the set of vertex labels, and $L : V \to A$ is the vertex label function that gives the label of a vertex.

The class of graph kernels studied in this article is based on *simple paths* in graphs. Paths are particular cases of *walks* where no vertex is visited more than once.

Definition 2.2. (**Walk and simple path**) A walk is a sequence of vertices ($[v_1, v_2, \ldots v_c]$) where pairs of consecutive vertices are adjacent (i.e., $(v_{i-1}, v_i) \in E$). Walks always begin and end with a vertex. A simple path is a walk in which no vertices are repeated. In other words, simple paths do not allow cycles. In the remaining of this article, we refer to simple paths as *paths*. The label l of a path is defined as the sequence of labels of its vertices (i.e., $l([v_1, v_2, \ldots v_c]) = [L(v_1), L(v_2), \ldots L(v_c)]$).

2.2 Kernel Theory

Graph kernel methods handle graph data using linear models. These methods require the definition of a kernel function that maps graphs into a linear feature space. This section summarizes some theoretical properties of graph kernels.

Definition 2.3. (**Positive definite kernel**) A function $k \in \mathbb{R}^{\mathcal{X}\times\mathcal{X}}$, where $\mathcal{X}$ is a non-empty space, is said to be a positive definite kernel iff it is continuous, symmetric, and the $n \times n$ Gram (or kernel) matrix $K = (k(\mathbf{u}_i, \mathbf{u}_j))$ is positive semi-definite (i.e., all its eigenvalues are non-negative) for all the input instances $\mathbf{u}_1, \mathbf{u}_2, \ldots \mathbf{u}_n$.

Positive definite kernels ensure that kernel algorithms (e.g., Gaussian processes, support vector machines) converge to a relevant solution. Moreover, positive definite kernels can be associated with *Hilbert spaces*, which are generalizations of Euclidean spaces with a finite or infinite number of dimensions. This mapping between positive definite kernels and Hilbert spaces is called *Mercer's property* [Burges 1998]. The kernel function $k(\mathbf{u}_i, \mathbf{u}_j)$ is usually defined as a dot product (i.e., $k = \langle \mathbf{u}_i, \mathbf{u}_j \rangle$).

Kernel methods perform two independent tasks: (1) computing the Gram matrix ($K = k(\mathbf{u}_i, \mathbf{u}_j)$), which gives the similarity between two objects in the linear space and (2) applying a linear method (e.g., regression, classification, dimensionality reduction) to the input objects based on K. The main contribution of this work regards to the first task. More specifically, we propose an efficient strategy for representing graph objects in a compact form using min-hashing.

A graph kernel is a special case of a *convolution kernel*. Convolution kernels are defined, recursively, in terms of simpler kernels over parts of two given objects [Haussler 1999]. In the case of graphs, objects are represented as a set of graph substructures. These substructures may be paths, trees, cycles or any arbitrary set of subgraphs. Without loss of generality, this work focuses on graph kernels that apply paths of limited size c as subgraphs (see Definition 2.2). Paths have shown to be powerful substructures for graph kernel methods. However, our technique may be applied to kernels that are based on other types of substructures as well.

2.3 Graph Kernel Functions

In a feature space, graphs can be seen as feature vectors. In other words, graph kernel methods produce vector representations for graphs. Considering graph kernels based on paths, a feature vector $(f_1, f_2, \ldots f_k)$ associates each of its n positions with a possible graph path. Therefore, the size of feature vectors is determined by the number of substructures from graphs in the database. Binary feature vectors describe the presence of a given path using binary variables (i.e., $f_i \in \{0, 1\}$). We say that a kernel that applies binary feature vectors is a *binary kernel*, while a *counting kernel* applies feature vectors with the frequency of paths in a graph. In this work, we apply two graph kernel functions from the literature, the *Tanimoto* and the *MinMax kernel* [Ralaivola et al. 2005]. It is important to notice that our fingerprinting technique may be applied to other kernel functions. The choice for these particular functions is due to their reported performance in the classification of chemical compounds, which is the context considered in the evaluation of our method.

Definition 2.4. (**Tanimoto**) Let G and H denote two graphs and c be an integer. The Tanimoto kernel k^t_c is defined by:

$$k^t_c(G, H) = \frac{\langle G, H \rangle_c}{\langle G, G \rangle_c + \langle H, H \rangle_c - \langle G, H \rangle_c}$$

where $\langle ., . \rangle_c$ is the dot product between two feature vectors based on graph paths of size limited to c.

The Tanimoto function gives the ratio between the number of paths shared by G and H, and the total number of paths in G and H.

Definition 2.5. (**MinMax**) Let G and H denote two graphs and c be an integer. Consider the function $\phi_p(G)$, which returns the number of occurrences of the path p in G, and the set $\mathcal{P}(c)$ to be the set of all possible paths of size limited to c. The MinMax kernel k_c^m is defined by:

$$k_c^m(G,H) = \frac{\sum_{p \in \mathcal{P}(c)} min(\phi_p(G), \phi_p(H))}{\sum_{p \in \mathcal{P}(c)} max(\phi_p(G), \phi_p(H))}$$

MinMax is a generalization of the Tanimoto function that considers path counts. In fact, MinMax is identical to Tanimoto for binary feature vectors.

2.4 Graph Fingerprints

An important property of the kernel functions described in the last section is that they are based on feature vectors that have their sizes determined by the number of possible paths in the input graph database. As a consequence, storing and processing feature vectors may not be feasible in realistic settings, where the number of possible paths is expected to be very large. This limitation can be addressed by the use of graph fingerprints, which are short descriptors for graphs.

Definition 2.6. (**Graph fingerprint**) A graph fingerprint $\zeta(G) = (\zeta_1, \zeta_2, \ldots \zeta_k)$ is a feature vector representation of a graph G with limited size k.

Graph fingerprints restrict the feature space defined by graph kernels in the sense that such feature space is constrained to the space of fingerprints. Different from feature vectors, graph fingerprints have their size defined by the user. As a consequence, the size of these fingerprints represent a trade-off between accuracy (i.e., description power) and performance in graph kernel methods. More specifically, fingerprints are compact enough to be loaded to main memory – even when the entire feature vector representations are too large – and also support efficient similarity computation in graph kernel methods. Given a graph database D and fingerprints of size k, the space required to store the fingerprints is $O(|D|k)$ and the time for computing the dot product between two fingerprints is $O(k)$. In this article, we introduce a new efficient strategy for the generation of accurate graph fingerprints.

2.5 Computing Graph Similarity using Fingerprints

Fingerprinting is part of the process of computing the similarity between two graphs efficiently. Figure 1 shows the main steps involved in the computation of graph kernel functions – represented as a Gram matrix – using fingerprints. We summarize such steps as follows:

(1) **Extraction:** Extracts a set of substructures from the graphs in the graph database. The features applied in this article are paths, but they could be any arbitrary type of subgraph.
(2) **Fingerprinting:** Given the representation of input graphs in the feature space, the fingerprinting step generates a fingerprint for each input graph. The main contribution of this work is proposing the use of min-hashes in the generation of graph fingerprints for graph kernel techniques.
(3) **Mapping:** In the last step, a kernel function computes the similarity between pairs of graphs based on fingerprints. The particular kernel functions applied in this work are the Tanimoto and the MinMax, which were described in Section 2.3.

In the next section, we describe in detail how min-hashing can be applied in the computation of effective graph fingerprints in the general framework described in Figure 1.

3. GRAPH SIMILARITY VIA MIN-HASHING

This section presents a new strategy for computing the similarity between graphs in graph kernel methods. The idea is to apply min-hashes in the generation of graph fingerprints, which are compact and accurate graph descriptors. These fingerprints enable efficient similarity computation with low memory space requirements. We begin with a short introduction to min-hashing and then describe an algorithm for the generation of graph fingerprints using min-hashes.

3.1 Min-Hashing

Min-hashing (or min-wise independent permutations locality sensitive hashing) is an efficient approximate technique for assessing the similarity between two sets using a small amount of data. Originally, this idea has been proposed in order to estimate the size of transitive closure and reachability sets [Cohen 1997]. The min-hashing scheme has been successfully applied in the generation of fingerprints for sets in varied scenarios, such as duplicate web page detection [Broder et al. 1997], itemset mining [Cohen et al. 2001], news personalization [Das et al. 2007], image processing [Chum et al. 2008], and tree-structured data mining [Tatikonda and Parthasarathy 2010]. Nevertheless, there is no evidence of the effectivity of min-hashing in graph kernels, which is the main topic of this research.

General hash functions map objects to keys of fixed length. The min-hashing scheme applies k different hash functions over a set of items and considers only the minimum hash values in the comparison between two sets. Given two sets $A = \{a_1, a_2, \ldots a_n\}$ and $B = \{b_1, b_2, \ldots b_m\}$, and a set of hash functions $H = \{h_1, h_2, \ldots h_k\}$, let $h_i^{min}(X)$ be the item of X with the minimum value of h_i. We say that $h_i^{min}(X)$ is a min-hash of X. The following proposition gives the probability that A and B will generate the same min-hash for a hash function h_i.

Proposition 3.1. ***(Probability of two sets to generate the same min-hash value for one hash function)*** *Given two sets, A and B, and a hash function h_i, the probability of two sets to generate the same min-hash value is given by [Cohen et al. 2001]:*

$$Pr[h_i^{min}(A) = h_i^{min}(B)] = \frac{|A \cap B|}{|A \cup B|}$$

We can see a hash function h_i as a particular random permutation of the items from $A \cup B$. As a consequence, $h_i^{min}(X)$ defines the first item from X according to such permutation. Proposition 3.1 comes directly from the fact that the probability of A and B to share a min-hash value is equal to the probability of picking up an item shared by A and B in $A \cup B$. It is interesting to notice that the probability $Pr[h_i^{min}(A) = h_i^{min}(B)]$ is the Jaccard similarity between the sets A and B. Therefore, this probability increases with the Jaccard similarity between the two sets. This property has important implications for the estimates of the kernel functions described in Section 2.3, as we will discuss in more detail in Section 3.2.

The use of multiple (k) independent min-hash values, instead of a single one, is expected to increase the accuracy of the min-hash scheme. In other words, the average value of $Pr[h_i^{min}(A) = h_i^{min}(B)]$ gets closer to the Jaccard similarity between A and B as we increase the number of hash functions $h_i \in H$ considered. Definition 3.2 gives a similarity measure for sets based on multiple min-hashes.

Definition 3.2. (**Similarity between two sets based on k min-hash values**) Given two sets, A and B, and a set of hash functions H, the similarity between them is defined as [Cohen et al. 2001]:

$$sim(A, B) = \frac{|\{i | 1 \leq i \leq k \wedge h_i^{min}(A) = h_i^{min}(B)\}|}{k}$$

In other words, the similarity between two sets is estimated as the number of common min-hash values between them. The theoretical upper bound of this estimate is a function of the number of hash functions applied. The value of $sim(A,B).k$ follows the distribution of a sum of independent variables $X_1, X_2, \ldots X_k$, where the expected value of X_i is given by Proposition 3.1 (i.e., $E[X_i] = Pr[h_i^{min}(A) = h_i^{min}(B)]$). Therefore, the error of $sim(A,B)$ can be estimated as follows.

THEOREM 3.3. ***(Estimated error of** sim(A,B))* *The number k of min-hash values needed in order to achieve an estimated error of $sim(A,B)$ upper bounded by θ ($0 < \theta \leq 1$) with confidence $1-\delta$ ($0 < \delta \leq 1$) must satisfy the following constraint [Cohen et al. 2001]:*

$$k \geq \frac{2+\theta}{\theta^2} ln(\frac{2}{\delta})$$

The probability that $sim(A,B)$ deviates from the Jaccard similarity between A and B by up to a factor θ is given by $Pr[|sim(A,B) - p| \geq \theta]$, where p is the given Jaccard similarity. The Chernoff bound on $Pr[|sim(A,B) - p| \geq \theta]$ gives that:

$$Pr[|sim(A,B) - p| \geq \theta] \leq 2e^{-\frac{\theta^2}{2+\theta}k}$$

Thus, $Pr[|sim(A,B) - p| \geq \theta] \leq \delta$ if k satisfies Theorem 3.3.

Given a set A, of size n, and k min-hashes, the space required to store the min-hash representations of A is $O(k)$. Furthermore, such min-hashes are generated in $O(nk)$ and two sets are compared in $O(k)$ computational time. An interesting property of min-hashing is that neither the comparison time nor the estimated error depend on the size of sets. Hence, min-hashing is convenient for the representation of large sets, such as the substructures extracted from real graphs.

3.2 Generating Graph Fingerprints using Min-Hashing

The main contribution of this work is proposing a new graph fingerprinting technique based on min-hashing. Fingerprints are compact descriptors for graphs and may support the application of graph kernels to large graph databases in real settings. Effective fingerprints must satisfy three main requirements: accuracy, efficiency, and compression. Because compression and efficiency are expected to be achieved at the expense of accuracy, graph fingerprints represent a trade-off between these three requirements imposed by real graph databases. Our hypothesis is that min-hashing enables the generation of graph fingerprints with high compression, efficiency, and accuracy.

Graph fingerprints are applied by graph kernel methods in the computation of similarity between two graph structures (see Figure 1). Therefore, accurate fingerprints must preserve, as much as possible, the distances between objects in the feature space. In other words, fingerprints of graphs with similar features (or substructures) should be similar. The opposite is also true, fingerprints of graphs with distinct features should be distinct as well. In this context, min-hashes have the convenient property that the expected probability of a match between two sets for a given hash function is the Jaccard similarity between such sets. Otherwise stated, min-hashing is appropriate for the comparison of graphs originally represented as sets of features. Moreover, the fact that neither the comparison time nor the estimated error of min-hash-based comparison depend on the size of the sets involved is a motivation to the use of min-hashes as fingerprints for the large graphs that emerge from real settings.

The general idea of min-hashes as graph fingerprints is very intuitive. As discussed in Section 2.3, in the feature space, graphs are represented as a set of substructures (e.g., paths, walks, cycles). For each graph, we produce min-hash values based on these substructures using k hash functions. The similarity between two graphs is proportional to the number of min-hash values shared by such graphs (see Definition 3.2). These similarities are aggregated in a Gram (or kernel) matrix, which contains the similarity between each pair of graphs from the database.

Algorithm 1: Graph Fingerprinting Algorithm

```
  Input: Substructures $S(G)$ from a graph $G$; Set of hash functions $H$
  Output: $fingerprint$
1 $k = |H|$;
2 foreach $i \in 1, 2, ...k$ do
3     $fingerprint[i] = \infty$;
4 $set = \emptyset$;
5 foreach $s \in S(G)$ do
6     $set = set \cup int(s)$
7 foreach $i \in 1, 2, ...k$ do
8     $fingerprint[i] = h_i^{min}(set)$;
```

Algorithm 1 is a high level description of an algorithm for generating graph fingerprints using min-hashing. The algorithm receives the set of substructures $S(G)$ from a graph G and a set of k hash-functions H. The output of the algorithm is the fingerprint of G, which is represented as a vector of min-hash values. Each position of the fingerprint vector is initialized as infinite (line 3). Moreover, a set of integer values (set) is obtained from the substructures $S(G)$ using the function int (line 6).

Given a substructure from a graph, the function int can be implemented using several strategies. Since we are particularly interested in graph kernels based on paths (see Section 2.3), we choose the following function to map a path $[v_1, v_2, \dots v_c]$ into an integer value:

$$int([v_1, v_2 \dots v_c]) = (a_1 L(v_1) + a_2 L(v_2) + \dots a_c L(v_c)) \pmod P \tag{1}$$

where a_j is a random integer such that $0 < a_i < P$, P is a big prime[3] number and L is the label function (see Section 2.1).

Given the representation of G as a set of integers, we compute the fingerprint of G as a vector of k min-hash values (line 8). Min-hash values are produced using the following permutation function:

$$h_i(x) = (a_i x + b_i) \pmod M$$

where x is an item from set, a_i and b_i are random integers such that $0 < a_i < M$ and $0 \le b_i < M$, and M is a big prime number such that $M \ge P$.

Values of a_i and b_i are generated once for each hash function $h_i \in H$. Given a hash function h_i, we define the min-hash of a set of integers as:

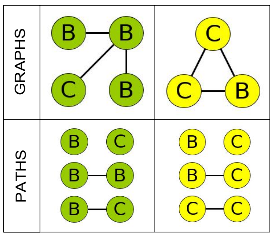
$$h_i^{min}(set) = \{h_i(x) | x \in set \wedge h_i(x) \le h_i(y), \forall y \in set\}$$

The proposed algorithm computes graph fingerprints using $O(k)$ memory space and $O(|S(G)|k)$ execution time. The memory space required is proportional to the number of min-hash values used in the generation of fingerprints. The execution time depends on the size of the sets for which the min-hashes are computed, which is the number of substructures from the input graph ($S(G)$).

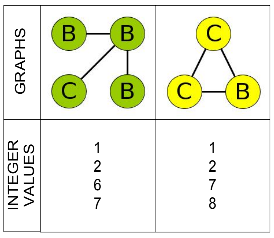
Considering a graph database $D = \{G_1, G_2, \dots G_n\}$, such that the average number of substructures from a graph from D is $\bar{S}$, the graph fingerprints are generated using $O(n\bar{S}k)$ time and $O(nk)$ space. We expect that these fingerprints will be computed only once. Given the fingerprints, the $n \times n$

[3]The expected number of collisions of the method depends on how large the selected prime number is.

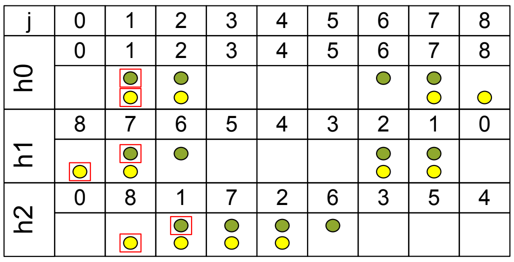
Gram matrix of the graph is computed in $O(n^2k)$ time. The fingerprints and the Gram matrix can be computed only once and then stored for further usage. On the other hand, computing the Gram matrix using the features (substructures) takes $O(n^2\bar{S}^2)$ time. Since the average number of substructures is expected to grow exponentially with the size of the substructures considered, graph fingerprints enables the application of kernel methods to large graph databases that would not otherwise be processed using the graph substructures. In particular, in case the substructures used are paths of limited size d, the number of substructures $S(G)$ is, in the worst case, $O(|G|^d)$.



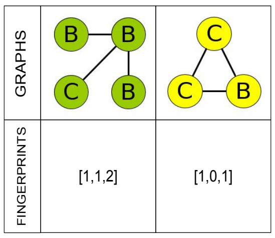
(a) I - Substructures



(b) II - Integer sets



(c) III - Min-hashes



(d) IV - Fingerprints

Fig. 2. Generating Graph Fingerprints using Min-Hashing (steps I, II, III, and IV). Min-hash values are the indexes (j) of the items of sets in the random permutations defined by the hash functions h0, h1, and h2. The similarity between the two graphs is 0.33

Figure 2 shows a running example of the computation of graph fingerprints using min-hashing. The graphs involved were extracted from the database given in Figure 1. Fingerprints are generated according to three min-hash values, where each hash function is a particular permutation of the universe of items (0-8). The hash function h0, for instance, defines the permutation $(0, 1, \ldots 8)$. We summarize each step of the process of generating the fingerprints as follows:

I. **Substructures:** A set of substructures from each graph is enumerated. The substructures considered are paths of size (i.e., number of vertices) limited to two.

II. **Integer sets:** Each substructure is mapped to an integer value in the interval [0,8]. Graphs are represented as the set of integers corresponding to their substructures.

III. **Min-hashes:** Each hash function defines a permutation of integers in the interval [0,8]. Integers associated with each graph are positioned (as circles) according to the permutations. The min-hash value for a hash function is the index j of the position of the first item associated with such graph in the permutation defined by the given hash function. Min-hash positions are marked with squares. The min-hash of the graph associated to the set $\{1, 2, 7, 8\}$ is 1 for the hash function h2 because the first item of such set in the permutation $(0, 8, 1, 7, 2, 6, 3, 4, 5)$ has index (j) 1.

IV. **Fingerprints:** Fingerprints are vectors of min-hash values for each of the three hash functions (h0, h1, and h2) applied in the fingerprinting process. The similarity between the two graphs based on the fingerprints (see Definition 3.2) is 0.33.

The similarity metric presented in Definition 3.2 is exactly the Tanimoto kernel function (see Definition 2.4). In order to apply the proposed fingerprinting technique to the MinMax kernel function, it is necessary to represent a fingerprint as a vector of pairs (h_i^{min}, f_i), where f_i is the number of substructures that produced a min-hash value h_i^{min} for a given graph. Other kernel functions based on set similarity may be easily adapted in order to support similarity computations based on min-hash values. Since the min-hash values are a compact representation of sets of substructures, these new kernel functions must consider shared min-hash values instead of substructures. Theorem 3.3 gives an estimate of the quality of these compact representations.

In this section, we presented a new fingerprinting technique for graph kernels. The idea is to apply min-hash values as fingerprints. Min-hashing supports the generation of compact and accurate fingerprints for large sets, but there is no study on its performance in graph kernel methods. Therefore, an extensive evaluation of the proposed strategy using real graphs is presented in the next section.

4. EXPERIMENTAL RESULTS

We conducted several experiments in order to evaluate the application of min-hashing in the generation of graph fingerprints using real graph databases. Our main goal is to check our hypothesis that min-hash fingerprints are effective for graph kernels in terms of the trade-off among accuracy, efficiency, and compression. As discussed in Section 2.3, we apply paths as graph substructures. The kernel functions applied in this evaluation are Tanimoto and MinMax.

We compare our technique against a naive fingerprinting approach, similar to the one applied in [Ralaivola et al. 2005]. This naive technique, which we call *baseline*, works as follows:

(1) Initialize the fingerprint as a vector of size k where each position is set as 0;
(2) Generate a unique ID for each graph substructure (see Equation 1);
(3) Map the set of integers from step 2 to a limited size vector of k positions by computing such integers module k;
(4) Set the vector positions associated with each path to 1.

The baseline fingerprinting strategy can be generalized to counting kernel functions by setting fingerprint positions to the number of corresponding integers module k, instead of binary values.

4.1 Datasets

We evaluated our graph fingerprinting technique using seven real datasets of chemical compounds. These datasets are composed of sets of molecules represented as graphs, where atoms are represented as vertices and bounds as edges. The label of a vertex corresponds to a chemical element. A molecule of water, for instance, is represented as a graph with three vertices and two edges, where two vertices labeled as Hydrogen are connected to one vertex labeled as Oxygen. Some of these datasets were previously applied in the evaluation of graph kernel techniques [Ralaivola et al. 2005] and graph pattern mining algorithms [Yan et al. 2008]. It is known that chemical graphs are sparse – average degree is slightly above two – and small – with a few dozen vertices on average. Nevertheless, we are still able to assess the effectivity of our fingerprint technique using these chemical datasets.

Table I shows some important properties of the datasets evaluated in this study. The Mutag dataset is composed of 188 chemical compounds from *Salmonella typhimurium*. The Predictive Toxicology Evaluation (PTE) dataset contains 340 compounds classified in terms of carcinogenicity for Mices and Rats. This dataset presents many labels (66) compared to the other datasets employed in this evaluation. The Predictive Toxicology Challenge (PTC) dataset contains the same kind of data that PTE, but it is divided into different animals groups – Male Mice (MM), Female Mice (FM), Male Rats (MR), and Female Rats (FR). AIDS is a dataset made available by the National Cancer Institute

dataset	$\|D\|$	average $\|V\|$	average $\|E\|$	$\|A\|$
Mutag	188	17.93	19.79	7
PTE	340	27.02	27.40	66
PTC-FM	349	25.25	25.62	19
AIDS	422	39.60	42.30	21
MCF-7	2294	60.00	63.00	27
MOLT-4	3140	57.00	60.00	34
OVCAR-8	2079	62.00	64.00	33

Table I. $|D|$ is the number of graphs in the database, $|V|$ and $|E|$ are the number of vertices and edges of graphs, respectively, and $|A|$ is the number of vertex labels in the database

(NCI) from the USA. It provides the screening results for the ability of 42,668 compounds against HIV. The database AIDS is a sample of it containing 422 active compounds (graphs). Finally, the PubChem website provides the datasets MCF-7, MOLT-4 and OVCAR-8 containing active chemical compounds from anti-cancer screen tests with breast, leukemia and ovarian cancer cells, respectively.

4.2 Fingerprint Accuracy

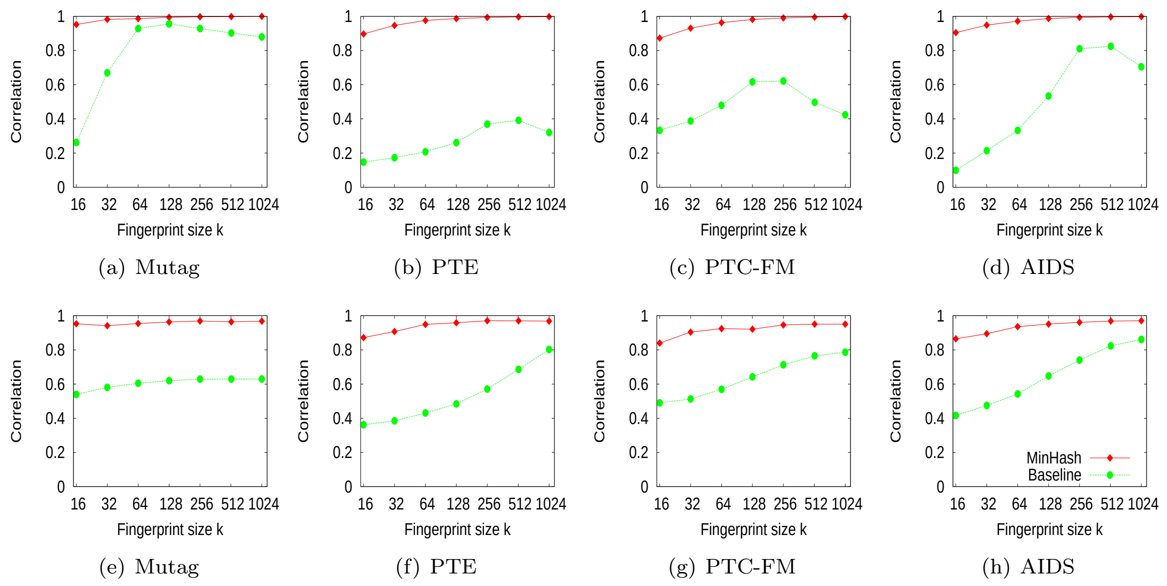
In this section we evaluate the accuracy of min-hash values as graph fingerprints. We measure the correlation between graph similarities computed using fingerprints and the entire sets of substructures. An accurate fingerprint is expected to produce similarity values that are close to the those computed considering the whole set of substructures. The databases considered are Mutag, PTE, PTC-FM, and AIDS. Moreover, we vary the fingerprint size (k) from 16 to 1024 in order to assess the trade-off between accuracy and size of fingerprints. The maximum size of substructures (paths) was set to 10. As discussed in Section 2.4, fingerprints are expected to be compact and accurate descriptors for graphs. However, compression is expected to be achieved at the expense of accuracy.

Figure 3 shows that min-hashing generates accurate fingerprints for the two kernel functions considered. Min-hashing achieves similarity values with correlations of at least 80%, even when small (k=16) fingerprints are applied. The proposed method outperforms the baseline technique systematically, with gains up to 800%. Furthermore, the baseline is more sensitive to the size of fingerprints than our method. In fact, the accuracy of the baseline does not grow steadily with the fingerprint size for the Tanimoto kernel. As a consequence, finding an optimal fingerprint size for the baseline method may be a challenging task in real settings. On the other hand, the accuracy of the min-hash fingerprints became stable after a small number of experiments.

These results lead us to conclude that min-hashing is able to generate accurate graph fingerprints. A small number of min-hash values may replace a large set of graph paths with a neglectable loss in description power. Nevertheless, the ultimate goal of graph fingerprints is supporting efficient and accurate kernel methods in the solution of real problems, such as classification, regression, and dimensionality reduction. Therefore, it is relevant to assess the impact of our fingerprinting technique in a more practical setting, as described in the next section.

4.3 Applications: Graph Classification and Retrieval

Two important applications of graph kernel methods are graph classification [Ralaivola et al. 2005] and retrieval [He and Singh 2006]. The graph classification problem consists of predicting the class of a graph based on its features. This problem appears in many real scenarios, such as chemical informatics, bioinformatics, information retrieval, and social network analysis. In this evaluation, we considered the task of predicting mutagenicity and carcinogenicity in the Mutag and the PTC-FM dataset, respectively. Graph kernels based on the complete set of substructures and the graph fingerprints



(a) Mutag (b) PTE (c) PTC-FM (d) AIDS

(e) Mutag (f) PTE (g) PTC-FM (h) AIDS

Fig. 3. Fingerprint accuracy of the technique based on min-hashing (MinHash) and the baseline, measured as the Person's correlation between similarities computed using fingerprints and the entire set of substructures, for the Tanimoto (a,b,c, and d) and MinMax (e,f,g, and h) kernel functions and four different datasets varying the fingerprint size. Min-hashing produces similarity values that are strongly correlated with the exact similarities, outperforming the baseline method.

were given as input to an SVM classifier[4], which is a state-of-the-art classification algorithm [Boser et al. 1992]. The reported accuracy is the average accuracy achieved through a 10-fold cross validation procedure. Path sizes were limited to 10, as done by previous work [Ralaivola et al. 2005].

The results shown in Figures 4(a), 4(b), 4(c), and 4(d) are consistent with those discussed in the last section. Because min-hash fingerprints are accurate, graph classifiers that apply kernel methods based on min-hash fingerprints achieve accuracy levels that are comparable to those achieved using the complete set of substructures. While the baseline and the min-hash fingerprint techniques performed similarly on the PTC-FM dataset, which is harder to classify, results on the Mutag dataset show that our method achieves improvements up to 10% when compared against the baseline strategy.

The second task considered in this section is graph retrieval (or graph database query), which consists of, given a query graph, identifying a set of similar graphs. Application scenarios for graph retrieval are very similar to those for graph classification. The datasets applied in graph retrieval are the same used in graph classification. However, in order to simulate graph retrieval, we select a graph from the database as the query graph and add noise to the dataset by changing the labels of 3% of the vertices, selected at random, from each graph. Our objective is to find the original graph in the presence of noise. The reported accuracy corresponds to the frequency in which the original graph is returned in the top 10 positions of a rank based on graph kernel similarity in a leave-one-out experiment. Path sizes were limited to 10, as done by previous work [Ralaivola et al. 2005].

Figures 4(e), 4(f), 4(g), and 4(h) show that min-hash fingerprints achieve high accuracy in graph retrieval. Using small fingerprints (k=16), our technique performs similar to graph kernels based on the whole set of structures, achieving accuracy levels up to 100% and outperforming the baseline. Though PTC-FM is harder to classify, their graphs are easier to retrieve in the presence of noise.

[4]`www.csie.ntu.edu.tw/~cjlin/libsvm/`

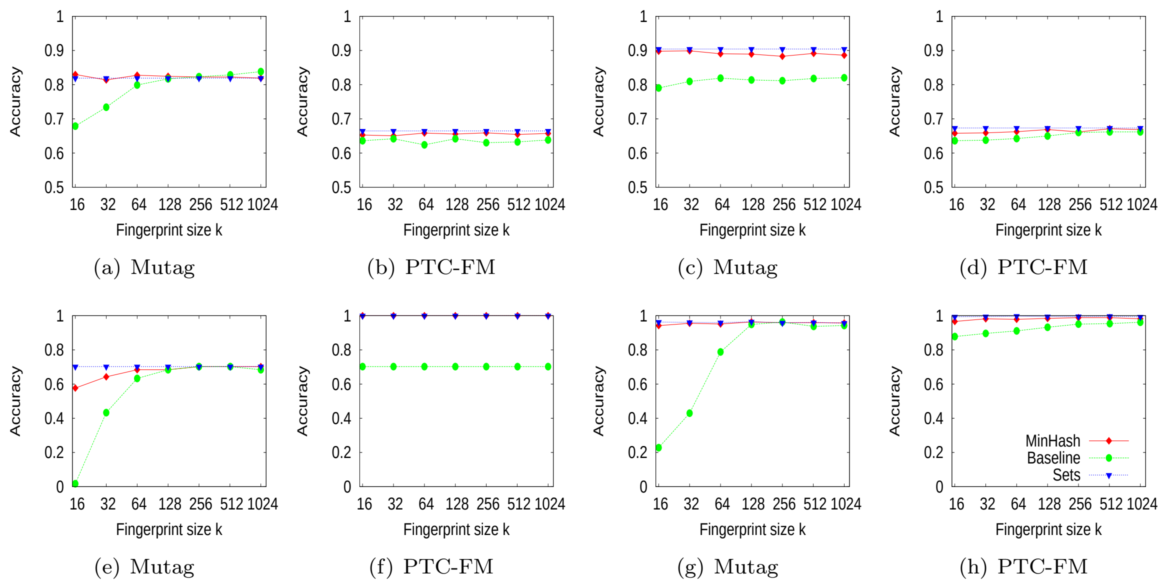


Fig. 4. Graph classification (a,b,c, and d) and retrieval (e, f, g, h) accuracy for the Tanimoto (a, b, e, and f) and the MinMax (c, d, g, and h) kernel functions varying the fingerprint sizes. Different from the baseline method, min-hash fingerprints achieve accuracy values that are comparable to the results obtained using the complete set of graph substructures for all the datasets and kernel functions.

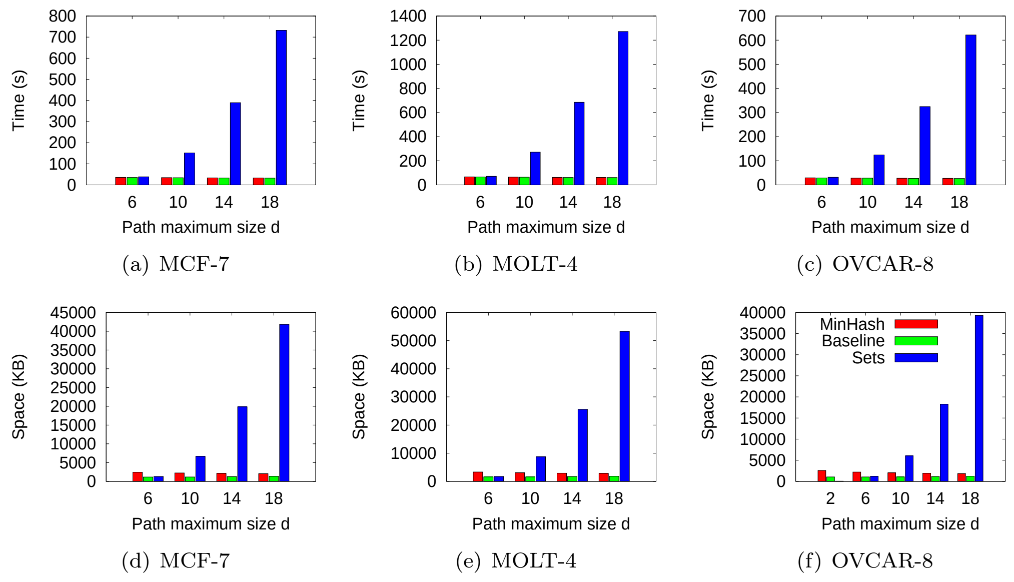
The results presented in this and the last sections confirm part of our hypothesis. Graph fingerprints based on min-hash values have shown to be accurate descriptions for graph structures that emerge from real datasets. In the next section, we evaluate the performance of our fingerprinting technique in terms of execution time and compression power.

4.4 Fingerprint Compression and Efficiency

The main hypothesis of this work is that min-hash fingerprints achieve a good trade-off among accuracy, efficiency, and compression. The first two sections of this evaluation focused on the analysis of the accuracy of the proposed graph fingerprints as descriptors for graph structures. In this last part, we evaluate min-hash graph fingerprints in terms of efficiency and compression. Otherwise stated, we compute the execution time and memory space required in order to generate the Gram (or kernel) matrix and store the fingerprint representations of the graphs from a database, respectively. Effective fingerprints are expected to be compact and support efficient graph similarity computations as means to enable the analysis of large graph databases.

The min-hash fingerprinting technique presented similar execution times and storage requirements compared to the baseline. Moreover, both fingerprinting techniques have shown to require less execution time and memory space for small and intermediate fingerprint sizes on all datasets. Nevertheless, we decided to focus our analysis on scenarios that rely more on effective graph fingerprints, which are those that involve both large databases and number of substructures. Figure 5 shows the execution time and the required memory space for computing the kernel matrix and storing the fingerprints, respectively, for the MCF-7, MOLT-4, and OVCAR-8 datasets. Fingerprints are able to keep both execution time and storage space constant despite the increase in maximum path size. In addition, the min-hash fingerprinting technique and the baseline achieve comparable results in terms of efficiency and compression, with time gains up to one order of magnitude and up to 97.5% space savings.

Based on the evaluation described along this section, we confirmed our hypothesis regarding the ef-



(a) MCF-7 (b) MOLT-4 (c) OVCAR-8

(d) MCF-7 (e) MOLT-4 (f) OVCAR-8

Fig. 5. Execution time to compute the kernel matrix (a,b, and c) and space required to store sets of paths and their fingerprints (d, e, and f) varying the maximum size of paths for three datasets.

fectivity of min-hashing in the generation of graph fingerprints. The results have shown that min-hash fingerprints are an accurate, compact and efficient alternative to the graph structure representation in graph kernel methods. These fingerprints are specially suitable for large datasets, being able to support efficient similarity computation and high compression with a small loss in descriptive power when compared to the original set of graph substructures.

5. RELATED WORK

This section discusses related work on graph similarity, graph kernels, and fingerprinting techniques for structured-data. Graph similarity has been an intense research topic, specially in mathematics and computer science. The first methods for graph similarity computation came from graph theory and were based on graph isomorphism and edit distance. However, these methods present some critical drawbacks such as poor performance and complex parametrization [Bunke and Allermann 1983; Bunke and Shearer 1998; Zhao et al. 2012]. These drawbacks motivated further studies on graph kernels.

The basic idea behind kernel methods was proposed about 40 years ago [Kimeldorf and Wahba 1971]. Graph kernels map graph data into a new representation domain (usually Euclidean) through a kernel function. These mapping – from a non-linear to a linear space – is known as kernel trick. The kernel trick enables the application of linear methods, such as support vector machines (SVMs), to non-linear problems. In this direction, convolution kernels support the use of the kernel trick in structured data such as graphs. The central idea of convolution kernels is dividing the graph into a set of simple components and defining a simple kernel function based on these components.

There are several graph kernel functions in the literature. A common strategy is decomposing graphs into paths, trees or even subgraphs. Among these strategies, paths and its variants – random walks, simple paths and shortest paths – have achieved relative success in the literature. Basically, a path-based kernel computes the similarity between two graphs by the number of common paths between them. Graph kernels using random walks were introduced by two different studies, product graph kernel [Gärtner et al. 2003] and marginalized graph kernel [Kashima et al. 2003]. The former presents a definition of graph product for which the result is a matrix containing the number of common paths between graphs. The second method explores a Markov model through a matrix of transition probabilities between nodes in the graph. The use of simple paths was proposed as an alternative to random walks, providing a more efficient solution. Results have shown that simple paths achieve good results in the classification of chemical graphs [Ralaivola et al. 2005]. More recently, a new strategy based on shortest paths was proposed in the literature. However, the enumeration of shortest paths is less efficient than the enumeration of simple paths [Borgwardt and Kriegel 2005].

Though graph kernels have been pointed as a promising alternative for efficient graph similarity computation, the number of substructures that describe the graphs imposes both memory and performance constraints to graph kernels in real settings. Fingerprinting techniques, such as bloom filters and locality sensitive hashing, constitute an interesting solution for this problem. Bloom filters are a quite intuitive scheme that became popular in several scenarios, including database applications and networking [Kirsch and Mitzenmacher 2006; Broder and Mitzenmacher 2004]. This method employs a certain number of hash functions in order to generate a binary size fixed fingerprint. A similar technique for graph fingerprinting where paths are used to set positions of a binary vector has been studied by previous work [Ralaivola et al. 2005]. A drawback of this technique is that bloom filters are sensitive to false positives. On the other hand, locality sensitive hashing (LSH) is a hashing scheme that maps similar data to the same bucket (hash value) with high probability and theoretical guarantees [Andoni and Indyk 2008]. LSH methods are frequently used to enhance nearest neighbor search algorithms and to estimate the similarity between objects (e.g., images, XML files, trees) [Chum et al. 2008; Tatikonda and Parthasarathy 2010]. In fact, min-hashing approach is a particular case of LSH. To the best of our knowledge, this is the first study on min-hashing for graph fingerprinting.

Besides graph kernel methods, graph similarity can be computed using other techniques. In particular, feature-based methods, such as [Yan et al. 2006], [Yan et al. 2008] and [Zhao et al. 2012], have achieved promising results in graph similarity computation. Comparing the effectivity of graph kernel and feature-based techniques for graph similarity is beyond the scope of this work.

6. CONCLUSIONS AND FUTURE WORK

In this article, we have proposed a new graph fingerprinting technique for graph kernels. Graph fingerprints are compact descriptors for graph structures and enable the application of graph kernel methods to large graph databases. Our technique brings the power of min-hashing scheme to the generation of graph fingerprints. A throughout empirical evaluation of the effectivity of min-hash fingerprints in terms of accuracy, efficiency, and compression power has shown that the proposed solution defines a good trade-off among these three aspects in real settings. In fact, we have given strong evidences that the information loss associated with the use of min-hash fingerprints – instead of the complete set of graph substructures – is small for short ($k = 16$) and neglectible for intermediate size ($k = 64$) fingerprints, outperforming a baseline strategy in up to 800%. In addition, graph fingerprinting achieves efficiency gains up to one order of magnitude with up to 97% space savings.

Promising directions for future work include: (1) the application of min-hash fingerprints to other graph databases; (2) the use of different graph substructures (e.g., frequent subgraphs); (3) the evaluation the proposed framework in other scenarios (e.g., graph clustering, outlier detection); and (4) a comprehensive study of several graph fingerprinting strategies (e.g., bloom filters, locality sensitive hashing) and a wide range of distance metrics for graph kernel methods.

REFERENCES

Andoni, A. and Indyk, P. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM* 51 (1): 117–122, 2008.

Borgwardt, K. M. and Kriegel, H.-P. Shortest-path kernels on graphs. In *Proceedings of the International Conference on Data Mining.* Washington, DC, USA, pp. 74–81, 2005.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. A training algorithm for optimal margin classifiers. In *Proceedings of the Workshop on Computational Learning Theory*. New York, NY, USA, pp. 144–152, 1992.

Broder, A. and Mitzenmacher, M. Network applications of bloom filters: A survey. *Internet Mathematics* 1 (4): 485–509, 2004.

Broder, A. Z., Glassman, S. C., Manasse, M. S., and Zweig, G. Syntactic clustering of the web. *Computer Networks and ISDN Systems* vol. 29, pp. 1157–1166, 1997.

Bunke, H. and Allermann, G. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters* 1 (4): 245 – 253, 1983.

Bunke, H. and Shearer, K. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters* 19 (3-4): 255–259, 1998.

Burges, C. J. C. A tutorial on support vector machines for pattern recognition. *Data Mining Knowledge Discovery* 2 (2): 121–167, 1998.

Chum, O., Philbin, J., and Zisserman, A. Near duplicate image detection: min-hash and tf-idf weighting. In *Proceedings of the British Machine Vision Conference*. Malvern, England, 2008.

Cohen, E. Size-estimation framework with applications to transitive closure and reachability. *Journal of Computer and System Sciences* 55 (3): 441–453, 1997.

Cohen, E., Datar, M., Fujiwara, S., Gionis, A., Indyk, P., Motwani, R., Ullman, J. D., and Yang, C. Finding interesting associations without support pruning. *Transactions on Knowledge and Data Engineering* 13 (1): 64–78, 2001.

Das, A. S., Datar, M., Garg, A., and Rajaram, S. Google news personalization: scalable online collaborative filtering. In *Proceedings of the International Conference on World Wide Web*. New York, NY, USA, pp. 271–280, 2007.

Gärtner, T., Flach, P., and Wrobel, S. On graph kernels: Hardness results and efficient alternatives. In *Proceedings of the Annual Conference on Computational Learning Theory and Kernel Workshop*. Washington, DC, USA, pp. 129–143, 2003.

Gärtner, T., Lloyd, J. W., and Flach, P. A. Kernels and distances for structured data. *Machine Learning* 57 (3): 205–232, 2004.

Haussler, D. Convolution Kernels on Discrete Structures. Tech. rep., University of California, Santa Cruz, 1999.

He, H. and Singh, A. K. Closure-tree: An index structure for graph queries. In *Proceedings of the International Conference on Data Engineering*. ACM, Washington, DC, USA, 2006.

Herbrich, R. *Learning Kernel Classifiers: Theory and Algorithms*. Cambridge, MA, USA, 2001.

Hofmann, T., Schölkopf, B., and Smola, A. Kernel methods in machine learning. *Annals of statistics* 36 (3): 1171–1220, 2008.

Kashima, H., Tsuda, K., and Inokuchi, A. Marginalized kernels between labeled graphs. In *International Conference on Machine Learning*. New York, NY, USA, pp. 321–328, 2003.

Kimeldorf, G. and Wahba, G. Some results on tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications* 33 (1): 82 – 95, 1971.

Kirsch, A. and Mitzenmacher, M. Distance-sensitive bloom filters. In *Proceedings of the Workshop on Algorithm Engineering and Experiments*. Philadelphia, PA, USA, 2006.

Ralaivola, L., Swamidass, S., Saigo, H., and Baldi, P. Graph kernels for chemical informatics. *Neural Networks* 18 (8): 1093–1110, 2005.

Scholkopf, B. and Smola, A. J. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA, 2001.

Tatikonda, S. and Parthasarathy, S. Hashing tree-structured data: Methods and applications. In *Proceedings of the International Conference on Data Engineering*. Washington, DC, USA, pp. 429–440, 2010.

Vishwanathan, S. V. N., Schraudolph, N. N., Kondor, R., and Borgwardt, K. M. Graph kernels. *Journal Machine Learning Research* 99 (1): 1201–1242, 2010.

Yan, X., Cheng, H., Han, J., and Yu, P. S. Mining significant graph patterns by leap search. In *Proceedings of the International Conference on Management of Data*. New York, NY, USA, pp. 433–444, 2008.

Yan, X., Zhu, F., Yu, P. S., and Han, J. Feature-based similarity search in graph structures. *Transactions on Database Systems* 31 (4): 1418–1453, 2006.

Zhao, X., Xiao, C., Lin, X., and Wang, W. Efficient graph similarity joins with edit distance constraints. In *Proceedings of the International Conference on Data Engineering*. Los Alamitos, CA, USA, pp. 834–845, 2012.