Min-Hashing for Probabilistic Frequent Subtree Feature Spaces

Pascal Welke[1], Tamás Horváth[1,2], and Stefan Wrobel[1,2]

[1] Dept. of Computer Science, University of Bonn, Germany
[2] Fraunhofer IAIS, Schloss Birlinghoven, Sankt Augustin, Germany

Abstract. We propose a fast algorithm for approximating graph similarities. For its advantageous semantic and algorithmic properties, we define the similarity between two graphs by the Jaccard-similarity of their images in a binary feature space spanned by the set of frequent subtrees generated for some training dataset. Since the feature space embedding is computationally intractable, we use a probabilistic subtree isomorphism operator based on a small sample of random spanning trees and approximate the Jaccard-similarity by min-hash sketches. The partial order on the feature set defined by subgraph isomorphism allows for a fast calculation of the min-hash sketch, without explicitly performing the feature space embedding. Experimental results on real-world graph datasets show that our technique results in a fast algorithm. Furthermore, the approximated similarities are well-suited for classification and retrieval tasks in large graph datasets.

1 Introduction

A common paradigm in distance-based learning is to embed the instance space into some appropriately chosen feature space equipped by a metric and to define the dissimilarity between two instances by the distance of their images in the feature space. In this work we deal with the special case of this paradigm that the instances are vertex and edge labeled graphs (without any structural restriction), the feature space is the set of vertices of the d-dimensional *Hamming-cube* (i.e., $\{0,1\}^d$) spanned by the elements of a feature set of cardinality d for some $d > 0$, and the metric is defined by the *Jaccard-distance*. This problem class is valid because any vertex of the d-dimensional Hamming-cube can be regarded as a set over a universe of cardinality d.

The crucial step of the generic approach above is the appropriate choice of the feature space. Indeed, the quality (e.g., predictive performance) of this method applied to some particular problem strongly depends on the semantic relevance of the features selected, implying that one might be interested in feature languages of large expressive power. If, however, the features are arbitrary graph patterns and the (binary) value of a pattern P on a graph G is defined by 1 if and only if P is a subgraph of G then this approach suffers from severe computational limitations. Indeed, even in the simple case that P is a path, it is NP-complete to decide whether P is a subgraph of G (i.e., whether P is subgraph isomorphic to G). This implies that the embedding of a graph into the feature space is computationally intractable if the pattern size is not bounded by some constant.

Our goal in this paper is to use *frequent* subgraphs as features, without any structural restriction on the graph class defining the instance space. This is motivated, among others, by the observation that frequent subgraph kernels [3] are of remarkable predictive performance e.g. on the ligand-based virtual screening problem [5]. The features, i.e., the set of frequent subgraphs are generated initially for a training set of graphs. To arrive at a practically fast algorithm, we restrict the pattern language to trees. This restriction alone is, however, not sufficient for a polynomial time algorithm for the following reasons: Mining frequent subtrees from arbitrary graphs is computationally intractable [6] and, as mentioned above, deciding subgraph isomorphism from a tree into a graph is NP-complete. To overcome these computational limitations, we give up the demand on the correctness of the pattern matching operator (i.e., subgraph isomorphism). More precisely, for each training graph we first take a set of k spanning trees generated uniformly at random, where k is some user specified parameter, replace each training graph with the random forest obtained by the vertex disjoint union of its k random spanning trees, and calculate finally the set of frequent subtrees for this forest database for some user specified frequency threshold. Clearly, the output of this probabilistic technique is always *sound* (any tree found to be frequent by this algorithm is a frequent subtree with respect to the original dataset), but *incomplete* (the algorithm may miss frequent subtrees). Since frequent subtrees in forests can be generated with polynomial delay [7], our frequent pattern generation algorithm runs in time polynomial in the combined size of the training dataset $\mathcal{D}$ and the set of frequent subtrees generated, as long as the number k of random spanning trees is bounded by a polynomial of the size of $\mathcal{D}$.

We follow a similar strategy for the embedding step: For an unseen graph G and a frequent tree pattern T, we generate a set F of k random spanning trees of G with the same method as for the frequent pattern mining algorithm and return 1 if T is subgraph isomorphic to F; 0 otherwise. On the one hand, in this way we decide subgraph isomorphism from a tree into a graph with one-sided error, as only a negative answer may be erroneous, i.e., when T is subgraph isomorphic to G but not to F. On the other hand, this subgraph isomorphism test with one-sided error can be performed in polynomial time. In a recent paper [12], we have empirically demonstrated that remarkable predictive performance can be obtained by the method sketched above. We show in this paper that our probabilistic algorithm decides subgraph isomorphism from T into G correctly with high probability if k is chosen appropriately.

Though the method sketched above runs in polynomial time, it is still impractical for large graphs and/or massive graph datasets for the following reasons: The fastest known algorithm for subtree isomorphism into forests runs in $O\left(n^{2.5}/\log n\right)$ time [9]. Although this is polynomial, it is prohibited even for datasets with a few hundred thousands of small graphs [12]. A second reason is the high dimensionality of the feature space, resulting in practically infeasible time and space complexity. Running time and memory can, however, be significantly reduced by using *min-hashing* [1], an elegant probabilistic technique for the approximation of the Jaccard-similarity. Given a binary feature vector $\boldsymbol{f}$ and a permutation π of $\boldsymbol{f}$, the method is based on calculating the min-hash value $h_\pi(\boldsymbol{f})$, i.e., the position of the first occurrence of 1 in the permuted order of $\boldsymbol{f}$.

For the feature set formed by the set of all paths up to a *constant* length, min-hashing has already been applied for graph similarity estimation by performing the embedding *explicitly* [11]. We show for the more general case of tree patterns of *arbitrary* length that for a feature vector $\boldsymbol{f}$ and permutation π, $h_\pi(\boldsymbol{f})$ can be computed *without* calculating $\boldsymbol{f}$. On the one hand, we can utilize the fact that we are interested in the first occurrence of a 1 in the order of π; once we have found it, we can stop the calculation, as all patterns after $h_\pi(\boldsymbol{f})$ are irrelevant for min-hashing. Beside this straightforward speed-up of the algorithm, the computation of the min-hash can further be accelerated utilizing the facts that a tree pattern T need not be evaluated if T or a subtree of T has already been considered for this or another permutation. These facts allow us to define a linear order on the patterns to be evaluated and to avoid redundant subtree isomorphism tests.

Our experimental results clearly demonstrate that using our technique, the number of subtree isomorphism tests can dramatically be reduced with respect to the min-hash algorithm performing the embedding explicitly. It is natural to ask how the predictive performance of the approximate similarities compares to the exact ones. We show that even for a few random spanning trees per chemical compound, remarkable precisions of the active molecules can be obtained by taking the k nearest neighbors of an active compound for $k = 1, \ldots, 100$ and that these precision values are close to those obtained by the *full* set of frequent subtrees. In a second experimental setting, we analyze the predictive power of support vector machines using our approximate similarities and show that it compares to that of state-of-the-art related methods. The stability of our incomplete probabilistic technique is explained by the fact that a subtree generated by our method is frequent not only with respect to the training set, but, with high probability, also with respect to the set of spanning trees of a graph.

The rest of the paper is organized as follows. In Section 2 we collect the necessary notions and sketch the min-hashing technique. In Section 3 we present our algorithm for calculating min-hashing in probabilistic tree feature spaces. In Section 4 we report our empirical results and conclude finally in Section 5 along with mentioning some interesting problems for future work.

2 Notions

In this section we collect the necessary notions and fix the notation. The set $\{1, \ldots, n\}$ will be denoted by $[n]$ for all $n \in \mathbb{N}$. The following basic concepts from graph theory are standard (see, e.g., [4]). An *undirected* (resp. *directed*) *graph* G is a pair (V, E), where V (vertex set) is a finite set and E (edge set) is a subset of the family of 2-subsets of V (resp. $E \subseteq V \times V$). Unless otherwise stated, by graphs we mean undirected graphs. An *unrooted* (or *free*) tree is a connected graph that contains no cycle. For simplicity, we restrict the description of our method to unlabeled graphs, by noting that all concepts can naturally be generalized to labeled graphs.

Among the classical embedding (or pattern matching) operators, subgraph isomorphism is the most widely used one in pattern mining. For this reason, in the next section we will present our method for *subgraph isomorphism* and discuss potential generalizations to other embedding operators in Section 5. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be graphs. They are *isomorphic* if there exists a bijection $\varphi : V_1 \to V_2$ with $\{u, v\} \in V_1$ if and only if $\{\varphi(u), \varphi(v)\} \in V_2$ for all $u, v \in V_1$. G_1 is *subgraph isomorphic* to G_2, denoted $G_1 \preccurlyeq G_2$, if G_2 has a subgraph isomorphic to G_1; $G_1 \prec G_2$ denotes that $G_1 \preccurlyeq G_2$ and G_1 is not isomorphic to G_2. It is a well-known fact that subgraph isomorphism is NP-complete. This negative result holds even for the case that the patterns are restricted to trees.

For any graph class $\mathcal{H}$ containing no two isomorphic graphs, $(\mathcal{H}, \preccurlyeq)$ is a *poset*. Since $\mathcal{H}$ will be finite for our case, we represent $(\mathcal{H}, \preccurlyeq)$ by a directed graph $(\mathcal{H}, E)$ with $(H_1, H_2) \in E$ if and only if $H_1 \preccurlyeq H_2$ and there is no $H \in \mathcal{H}$ with $H_1 \prec H \prec H_2$ for all $H_1, H_2 \in \mathcal{H}$.

We will also use concepts and algorithms from frequent subgraph mining. For any graph class $\mathcal{H}$ (the *pattern* class), finite set $\mathcal{D}$ of graphs, and for any frequency threshold $\theta \in (0, 1]$, a pattern $H \in \mathcal{H}$ is *frequent* if $|\{G \in \mathcal{D} : H \preccurlyeq G\}| \geq \theta|\mathcal{D}|$. Given $\mathcal{H}$, $\mathcal{D}$, and θ, the problem of frequent subgraph mining is to *generate* all patterns from $\mathcal{H}$ that are frequent. This listing problem is computationally intractable [6]. It follows from the proof of this negative result that the problem remains intractable if $\mathcal{H}$ is restricted to trees.[3] If, however, $\mathcal{D}$ is restricted to forests then frequent subgraphs (i.e., subtrees) can be generated with polynomial delay [7].

We will measure the similarity between two graphs by the Jaccard-similarity of their images in the Hamming-cube $\{0, 1\}^{|\mathcal{F}|}$ spanned by the elements of some finite feature set $\mathcal{F}$. The binary feature vectors can then be regarded as the characteristic vectors of subsets of $\mathcal{F}$. Given two feature vectors $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ representing the sets S_1 and S_2, respectively, we define their similarity by the *Jaccard-similarity* of S_1 and S_2, i.e., by

$$\textsc{Sim}_{\text{Jaccard}}(\boldsymbol{f_1}, \boldsymbol{f_2}) := \textsc{Sim}_{\text{Jaccard}}(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

with $\textsc{Sim}_{\text{Jaccard}}(\emptyset, \emptyset) := 0$ for the degenerate case. As long as the feature vectors are low dimensional (i.e., $|\mathcal{F}|$ is small), the Jaccard-similarity can quickly be calculated. If, however, they are high dimensional, it can be approximated with the following fast probabilistic technique based on *min-hashing* [1]: For a permutation π of $\mathcal{F}$ and feature vector $\boldsymbol{f}$, define $h_\pi(\boldsymbol{f})$ to be the index of the first

[3] We note that the crucial property implying the negative complexity result in [6] is not necessarily the intractability of subgraph isomorphism; there are cases when efficient frequent subgraph mining is possible even for NP-hard pattern matching operators [7].

entry with value 1 in the permuted order of $\boldsymbol{f}$. One can show that the following correspondence holds for the feature vectors $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ above (see [1] for the details):

$$\textsc{Sim}_{\text{Jaccard}}(S_1, S_2) = \mathbf{Pr}\left[h_\pi(\boldsymbol{f_1}) = h_\pi(\boldsymbol{f_2})\right] \quad ,$$

where the probability is taken by selecting π uniformly at random from the set of permutations of $\mathcal{F}$. This allows for the following approximation of the Jaccard-similarity between $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$: Generate a set $\pi_1, \ldots, \pi_K$ of permutations of the feature set uniformly at random and return K'/K, where K' is the number of permutations π_i with $h_{\pi_i}(\boldsymbol{f}_1) = h_{\pi_i}(\boldsymbol{f}_2)$. The approximation of the Jaccard-distance with min-hashing results in a fast algorithm if the embedding into the feature space can be computed quickly.

3 Efficient Min-Hash Sketch Computation

In this section we present our method for *approximating* the similarity between two graphs. We define this similarity by the Jaccard-similarity of their binary feature vectors for its advantageous semantic and algorithmic properties. The underlying feature space is spanned by a certain set of frequent tree patterns fixed in advance. More precisely, given a finite training set $\mathcal{D}$ of graphs and a frequency threshold $\theta \in (0, 1]$, in a *preprocessing step* we first generate a subset $\mathcal{F}$ of the set of frequent subtrees of $\mathcal{D}$. Clearly, $\mathcal{F}$ is finite. It will span the Hamming feature space $\{0, 1\}^{|\mathcal{F}|}$ and serve as the universe for the Jaccard-similarity. Given $\mathcal{F}$ and two graphs G_1, G_2, the similarity $\textsc{Sim}(G_1, G_2)$ between G_1 and G_2 is then defined by

$$\textsc{Sim}(G_1, G_2) := \textsc{Sim}_{\text{Jaccard}}\left(\boldsymbol{f_1}, \boldsymbol{f_2}\right),$$

where $\boldsymbol{f_i}$ is the characteristic vector of the set $\{T \in \mathcal{F} : T \preccurlyeq G_i\}$. The preprocessing step and the above definition of similarity raise the following three computational problems:

(P1) *The Frequent Subtree Mining Problem*: *Given* a finite set $\mathcal{D}$ of graphs and a frequency threshold $\theta \in (0, 1]$, *generate* the set of all *trees* that are frequent in $\mathcal{D}$.

(P2) *The Subtree Isomorphism Problem*: *Given* a tree T and a graph G, *decide* whether or not $T \preccurlyeq G$.

(P3) *Computing the Jaccard-similarity*: *Given* two binary feature vectors $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ as defined above, *compute* $\textsc{Sim}_{\text{Jaccard}}(\boldsymbol{f}_1, \boldsymbol{f}_2)$.

Since we have no restrictions on $\mathcal{D}$ and G, problems (P1) and (P2) are computationally intractable. In particular, unless P = NP, the frequent subtree mining problem cannot be solved in output polynomial time [6] and deciding whether a tree is subgraph isomorphic to a graph is NP-complete. Regarding (P3), as $|\mathcal{F}|$ is typically some large set, computing the Jaccard-similarity makes the above algorithmic definition practically infeasible if a huge number of similarity queries must be evaluated.

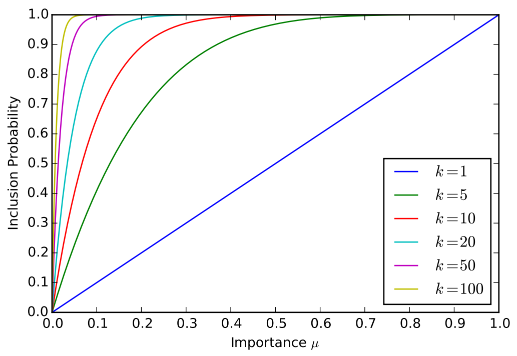

Fig. 1. The function $1-(1-\mu)^k$ for different values of k.

3.1 Probabilistic Tree Patterns

We first focus on problems (P1) and (P2). To overcome the complexity limitations of these two problems, we follow our approach described in [12] and give up the demand on the completeness of (P1) and on the correctness of the subtree isomorphism test for (P2), and use the following *probabilistic* technique: We replace each graph in $\mathcal{D}$ for problem (P1) and G for problem (P2) by a set of its k spanning trees generated uniformly at random. For an empirical evaluation of this method, the reader is referred to [12]; a probabilistic justification together with an appropriate choice of k is given below. According to this probabilistic technique, a tree T will be found as frequent in $\mathcal{D}$ (P1) if it is a subtree of at least $\theta|\mathcal{D}|$ forests, each formed by the vertex disjoint union of the k random spanning trees. Similarly, T will be found to be subgraph isomorphic to G (P2) if T is subgraph isomorphic to the forest of G formed by the vertex disjoint union of the k random spanning trees generated for G.

Regarding (P1), the probabilistic technique above results in a *sound*, but *incomplete* algorithm. Indeed, any tree found to be frequent by the mining algorithm in the forest dataset obtained is a frequent tree in $\mathcal{D}$, but we have no guarantee that all frequent trees of $\mathcal{D}$ will be generated. Thus, our technique may ignore frequent tree patterns. Regarding (P2), we decide subtree isomorphism by one-sided error: If T is subgraph isomorphic to any of the k spanning trees of G then T is subgraph isomorphic to G; otherwise T may or may not be subgraph isomorphic to G.

The rationale behind our probabilistic technique is as follows. For a graph G, let $\mathfrak{S}(G)$ be the set of spanning trees of G. Let $\mu \in (0,1]$. A tree T is μ-*important* in G if

$$\frac{|\{S \in \mathfrak{S}(G) : T \preccurlyeq S\}|}{|\mathfrak{S}(G)|} \geq \mu \ .$$

Thus, the probability that a μ-important tree in G is subtree isomorphic to a spanning tree of G generated uniformly at random is at least μ. Notice that

$\mu = 1$ for any subtree of the forest formed by the set of bridges of G (i.e., by the edges that do not belong to any cycle in G). Let $\mathfrak{S}_k(G)$ denote a sample of k spanning trees of G generated independently and uniformly at random. Then

$$\mathbf{Pr}\left[\exists S \in \mathfrak{S}_k(G) \text{ such that } T \preccurlyeq S\right] \geq 1 - (1-\mu)^k \ . \tag{1}$$

The bound in (1) implies that for any graph G and μ-important tree pattern T in G for some $\mu \in (0, 1]$, and for any $\delta \in (0, 1)$,

$$\mathbf{Pr}\left[\exists S \in \mathfrak{S}_k(G) \text{ such that } T \preccurlyeq S\right] \geq 1 - \delta$$

whenever

$$k \geq \frac{\ln \delta}{\ln(1-\mu)}$$

(see, also, Figure 1 for the function $1 - (1-\mu)^k$ for different values of k). Thus, if k is appropriately chosen, we have a probabilistic guarantee in terms of the confidence parameter δ that all μ-important tree patterns will be considered with high probability. As an example, we need less than 20 random spanning trees to correctly process a 0.15-important tree pattern with probability at least 0.95. Clearly, a smaller value of μ results in a larger feature set.

3.2 Fast Min-Hashing for Tree Patterns

We now turn to problem (P3) of computing the Jaccard-similarity between two binary feature vectors corresponding to the set $\mathcal{F}$ of tree patterns generated in the preprocessing step. More precisely, given a set $\mathcal{F}$ of trees that index our full feature space and two graphs G_1, G_2, we want to answer similarity queries of the form $\textsc{Sim}(G_1, G_2)$ by calculating $\textsc{Sim}_{\text{Jaccard}}(\boldsymbol{f}_1, \boldsymbol{f}_2)$, where $\boldsymbol{f}_i$ is the binary feature vector representing the set of trees from $\mathcal{F}$ that are subgraph isomorphic to the forest formed by the vertex disjoint union of the k random spanning trees generated for G_i ($i = 1, 2$). Instead of using the naive brute-force algorithm, i.e., calculating first the explicit embedding of G_i into the feature space and computing then the exact value of $\textsc{Sim}_{\text{Jaccard}}(\boldsymbol{f}_1, \boldsymbol{f}_2)$, we follow Broder's probabilistic *min-hashing* technique [1] sketched in Section 2.

Min-hashing was originally applied to text documents using q-shingles as features (i.e., sequences of q contiguous tokens for some $q \in \mathbb{N}$), implying that one can calculate the explicit embedding in linear time by shifting a window of size q through the document to be embedded. In contrast, a naive algorithm embedding a graph with n vertices into the feature space corresponding to $\mathcal{F}$ would require $O\left(|\mathcal{F}| n^{2.5} / \log n\right)$ time by using the fastest subtree isomorphism algorithm [9], which is practically infeasible for large-sized feature sets. Another important difference between the two applications is that while the set of q-shingles for text documents forms an anti-chain (i.e., the q-shingles are pairwise incomparable), subgraph isomorphism induces a natural *partial order* on $\mathcal{F}$. The transitivity of subgraph isomorphism allows us to safely ignore features from $\mathcal{F}$ that do not influence the outcome of min-hashing, resulting in a much faster algorithm.

More precisely, in the preprocessing step, directly after the generation of $\mathcal{F}$, we generate K random permutations $\pi_1, \ldots, \pi_K : \mathcal{F} \to [|\mathcal{F}|]$ of $\mathcal{F}$ (see [2] for the details) and fix them for computing the min-hash values that will be used for similarity query evaluations (cf. Section 2). Since the computation of the embeddings is the most expensive operation, we can allow preprocessing time and space that is polynomial in the size of the pattern set $\mathcal{F}$. Therefore, we explicitly compute and store $\pi_1, \ldots, \pi_K$, and do not apply any implicit representations of them. This is particularly true, as we compute $\mathcal{F}$ explicitly in the preprocessing step and spend time that is polynomial in $\mathcal{F}$ anyway.

For a graph G and permutation π of $\mathcal{F}$, let

$$h_\pi(G) = \operatorname*{argmin}_{T \in \mathcal{F}} \{\pi(T) : T \preccurlyeq G\} \quad .$$

The *sketch* of G with respect to $\pi_1, \ldots, \pi_K$ is then defined by

$$\textsc{Sketch}_{\pi_1,\ldots,\pi_K}(G) = (h_{\pi_1}(G), \ldots, h_{\pi_K}(G)) \quad .$$[4]

The rest of this section is devoted to the following problem: Given $\pi_1, \ldots, \pi_K$ and G as above, compute $\textsc{Sketch}_{\pi_1,\ldots,\pi_K}(G)$. The first observation that leads to an improved algorithm computing $\textsc{Sketch}_{\pi_1,\ldots,\pi_K}(G)$ is that for any $i \in [K]$, π_i may contain trees that can never be the first matching patterns according to π_i, for any query graph G. Indeed, suppose we have two patterns $T_1, T_2 \in \mathcal{F}$ with $T_1 \preccurlyeq T_2$ and $\pi_i(T_1) < \pi_i(T_2)$. Then, for any query graph G, either

1. $T_1 \preccurlyeq G$ and hence T_2 cannot be the first matching pattern in π_i or
2. $T_1 \not\preccurlyeq G$ and hence, by the transitivity of subgraph isomorphism, we have $T_2 \not\preccurlyeq G$ as well.

For both cases, T_2 will never be the first matching pattern according to π_i and can therefore be omitted from this permutation. Algorithm 1 implements this idea for a single permutation π of $\mathcal{F}$. The proof of the following result is immediate from the remarks above:

Lemma 1. *Let $\sigma = \langle T_1, \ldots, T_l \rangle$ be the output of Algorithm 1 for a permutation π of $\mathcal{F}$. Then, for any graph G,*

$$h_\pi(G) = \operatorname*{argmin}_{T_i \in \sigma} \{i : T_i \preccurlyeq G\} \quad .$$

Algorithm 1 runs in time $O(|\mathcal{F}|)$. Loop 5 can be implemented by a DFS that does not recurse on the visited neighbors of a vertex. In this way, each edge of F is visited exactly once during the algorithm.

We now turn to the computation of $\textsc{Sketch}_{\pi_1,\ldots,\pi_K}(G)$. A straightforward implementation of calculating $\textsc{Sketch}_{\pi_1,\ldots,\pi_K}(G)$ for the evaluation sequences

[4] In practice, we do not store the patterns in $\textsc{Sketch}_{\pi_1,\ldots,\pi_K}(G)$ explicitly. Instead, we define some arbitrary total order on $\mathcal{F}$ and represent each pattern by its position according to this order.

Input: directed graph $F = (\mathcal{F}, E)$ representing a poset $(\mathcal{F}, \preccurlyeq)$ and permutation π of $\mathcal{F}$
Output: evaluation sequence $\sigma = \langle T_1, \ldots, T_l \rangle \in \mathcal{F}^l$ for some $0 < l \leq |\mathcal{F}|$ with $\pi(T_i) < \pi(T_j)$ for all $1 \leq i < j \leq l$

```
1: init $\sigma$ := empty list
2: init $visited(T) := 0$ for all $T \in \mathcal{F}$
3: for all $T \in \mathcal{F}$ in the order of $\pi$ do
4:     if $visited(T) = 0$ then
5:         for all $T' \in \mathcal{F}$ (including $T$) that are reachable from $T$ in $F$ do
6:             set $visited(T') := 1$
7:         append $T$ to $\sigma$
8: return $\sigma$
```

Algorithm 1: Poset-Permutation-Shrink

$\sigma_1, \ldots, \sigma_K$ computed by Algorithm 1 for $\pi_1, \ldots, \pi_K$ just loops through each evaluation sequence, stopping each time the first match is encountered. This strategy can further be improved by utilizing the fact that a pattern T may be evaluated redundantly more than once for a graph G, if T occurs in more than one permutation before or as the first match. Lemma 2 below formulates necessary conditions for avoiding redundant subgraph isomorphism tests.

Lemma 2. *Let G be a graph, $F = (\mathcal{F}, E)$ be a directed graph representing a poset $(\mathcal{F}, \preccurlyeq)$, and let $\sigma_1, \ldots, \sigma_K$ be the evaluation sequences computed by Algorithm 1 for the permutations $\pi_1, \ldots, \pi_K$ of $\mathcal{F}$. Let $\mathfrak{A}$ be an algorithm that correctly computes $\textsc{Sketch}_{\pi_1,\ldots,\pi_K}(G)$ by evaluating subgraph isomorphism in the pattern sequence $\Sigma = \langle \sigma_1[1], \ldots, \sigma_K[1], \sigma_1[2], \ldots, \sigma_K[2], \ldots \rangle$. Then $\mathfrak{A}$ remains correct if for all $i \in [K]$ and $j \in [|\sigma_i|]$, it skips the evaluation of $\sigma_i[j] \preccurlyeq G$ whenever one of the following conditions holds:*

1. *$\sigma_i[j'] \preccurlyeq G$ for some $j' \in [j-1]$,*
2. *there exists a pattern T before $\sigma_i[j]$ in Σ such that $\sigma_i[j] \preccurlyeq T$ and $T \preccurlyeq G$,*
3. *there exists a pattern T before $\sigma_i[j]$ in Σ such that $T \preccurlyeq \sigma_i[j]$ and $T \not\preccurlyeq G$.*

Proof. If Condition 1 holds then the min-hash value for permutation π_i has already been determined. If $\sigma_i[j] \preccurlyeq T$ and $T \preccurlyeq G$ then $\sigma_i[j] \preccurlyeq G$ by the transitivity of subgraph isomorphism. For the same reason, if $T \preccurlyeq \sigma_i[j]$ and $T \not\preccurlyeq G$ then $\sigma_i[j] \not\preccurlyeq G$. Hence, if Condition 2 or 3 holds then $\mathfrak{A}$ can infer the answer to $\sigma_i[j] \preccurlyeq G$ without explicitly performing the subgraph isomorphism test.

Algorithm 2 computes the sketch for a graph G along the conditions formulated in Lemma 2. It maintains a state for all $T \in \mathcal{F}$ defined as follows: 0 encodes that $T \preccurlyeq G$ is unknown, 1 that $T \preccurlyeq G$, and -1 that $T \not\preccurlyeq G$.

Theorem 1. *Algorithm 2 is correct, i.e., it returns $\textsc{Sketch}_{\pi_1,\ldots,\pi_K}(G)$. Furthermore, it is non-redundant, i.e., for all patterns $T \in \mathcal{F}$, it evaluates at most once whether or not $T \preccurlyeq G$.*

Input: graph G, directed graph $F = (\mathcal{F}, E)$ representing a poset $(\mathcal{F}, \preccurlyeq)$ and K evaluation sequences $\sigma_1, \ldots, \sigma_K$ computed by Algorithm 1 for the permutations $\pi_1, \ldots, \pi_K$ of $\mathcal{F}$
Output: $\textsc{Sketch}_{\pi_1,\ldots,\pi_K}(G)$

```
1: init $sketch := [\bot, \ldots, \bot]$
2: init $state(T) := 0$ for all $T \in \mathcal{F}$
3: for $i = 1$ to $|\mathcal{F}|$ do
4:     for $j = 1$ to $K$ do
5:         if $|\sigma_j| \geq i \wedge sketch[j] = \bot$ then
6:             if $state[\sigma_j[i]] \neq 0$ then
7:                 if $state[\sigma_j[i]] = 1$ then $sketch[j] = \sigma_j[i]$
8:             else if $\sigma_j[i] \preccurlyeq G$ then
9:                 $sketch[j] = \sigma_j[i]$
10:                for all $T' \in \mathcal{F}$ (including $T$) that can reach $T$ in $F$ do
11:                    set $state(T') := 1$
12:            else
13:                for all $T' \in \mathcal{F}$ (including $T$) that are reachable from $T$ in $F$ do
14:                    set $state(T') := -1$
15: return $sketch$
```

Algorithm 2: Min-Hash Sketch

Proof. The correctness is immediate from Lemmas 1 and 2. Regarding non-redundancy, suppose $T \preccurlyeq G$ has already been evaluated for some pattern $T \in \mathcal{F}$ with $T = \sigma_i[j]$. Then, as $T \preccurlyeq T$, for any $\sigma_{i'}[j'] = T$ after $\sigma_i[j]$ in Σ either Condition 2 or 3 holds and hence $T \preccurlyeq G$ will never be evaluated again.

Once the sketches are computed for two graphs G_1, G_2, their Jaccard similarity with respect to $\mathcal{F}$ can be approximated by the fraction of identical positions in these sketches. (The similarity of G_1 and G_2 with $\textsc{Sketch}_{\pi_1,\ldots,\pi_K}(G_1) = \textsc{Sketch}_{\pi_1,\ldots,\pi_K}(G_2) = (\bot, \ldots, \bot)$ is defined by 0.)

4 Experimental Evaluation

We have conducted experiments on several real-world datasets. Since our method is restricted to connected graphs, disconnected graphs have been omitted. To obtain the feature sets of probabilistic tree patterns, we have applied the method in [12] to a randomly sampled subset of 10% of the graphs in each dataset. We have restricted the maximum size of a tree pattern to 10 vertices, as this bound seemed optimal for the predictive/ranking performance for all chemical datasets used in our experiments by noting that our method is not restricted to constant-sized tree patterns. The same observation is reported in [11]. We have empirically investigated (i) the *speed* measured by the number of subtree isomorphism tests performed, (ii) the *quality* of our method for the retrieval of positive molecules in the highly skewed NCI-HIV dataset, and (iii) the *predictive performance* of support vector machines using a kernel function based on similarity measure.

Datasets: We have used the chemical graph datasets MUTAG, PTC, DD, NCI1, and NCI109 obtained from `http://www.di.ens.fr/~shervashidze/`, and NCI-HIV from `https://cactus.nci.nih.gov/`. MUTAG is a dataset of 188 connected compounds labeled according to their mutagenic effect on Salmonella typhimurium. PTC contains 344 connected molecular graphs, labeled according to the carcinogenicity in mice and rats. DD consists of $1,187$ protein structures, of which $1,157$ are connected. Labels differentiate between enzymes and non-enzymes. NCI1 and NCI109 contain $4,110$ resp. $4,127$ compounds of which $3,530$ resp. $3,519$ are connected. Both are balanced sets of chemical molecules labeled according to their activity against non-small cell lung cancer (resp. ovarian cancer) cell lines. NCI-HIV consists of $42,687$ compounds of which $39,337$ are connected. The molecules are annotated with their activity against the human immunodeficiency virus (HIV). In particular, they are labeled by "active" (A), "moderately active" (M), or "inactive" (I). While the first five datasets have a balanced class distribution, the class distribution of NCI-HIV is heavily skewed: Only 329 molecules (i.e., less than 1%) are in class A, 906 in class M, and the remaining $38,102$ in class I.

Before going into the details, we first note that the similarities obtained by our method approximate the *exact* Jaccard-similarities quite closely on average. On a sample of roughly 1,500 graphs from the NCI-HIV dataset, the exact Jaccard similarities based on the full set of frequent trees and the similarities based on our min-hashing method showed a mean-squared-error of at most 0.005 for sketch size $K = 32$ and for an average Jaccard-similarity of 0.1396. For space limitations, we omit a detailed discussion of these results.

4.1 Speed-Up

The main goal of our method is to reduce the number of subgraph isomorphism tests during the computation of the min-hash sketch for a graph. We now show the effectiveness of our method from this aspect. To this end, we have compared our method given in Algorithm 2 not only with the *brute-force* explicit embedding, but also with the following *naive* embedding algorithm utilizing the partial order on the feature set $\mathcal{F}$: Given the poset $(\mathcal{F}, \preccurlyeq)$ as a directed graph F and a graph G, we traverse F starting at the vertices with in-degree 0. If a pattern T does not match G, we prune away all patterns reachable from T in F. In this way, we obtain the complete feature set of G with respect to $\mathcal{F}$ and compute the min-hash sketch accordingly. It is important to note that our algorithm may perform more subgraph isomorphism tests than the naive algorithm; this is due to the fact that, in contrast to the naive algorithm, we do not traverse F systematically. We leave a detailed discussion for the long version of this paper. We have compared our algorithm also with the naive method above in terms of the number of subgraph isomorphism tests performed. Table 1 shows the average number of subtree isomorphism tests per graph together with the pattern set size, for different datasets and parameters. The last four columns are the results of our algorithm for sketch size $K = 32, 64, 128, 256$, respectively. It can be seen that our algorithm (MH32–MH256) performs dramatically less subtree

Dataset	k	θ	$\lvert\mathcal{F}\rvert$	naive	MH32	MH64	MH128	MH256
MUTAG	5	10%	452	206.38	49.93	68.24	96.12	127.42
MUTAG	10	10%	543	244.11	42.77	63.77	90.57	125.39
MUTAG	15	10%	562	254.86	45.39	65.96	94.87	133.91
MUTAG	20	10%	573	260.18	55.34	76.32	105.15	135.11
PTC	5	10%	1,430	321.04	70.07	102.62	121.12	156.12
PTC	5	1%	9,619	734.79	236.31	327.27	475.35	611.92
PTC	10	10%	1,566	354.20	79.63	108.59	109.44	147.91
PTC	20	10%	1,712	376.65	17.60	25.81	31.49	39.62
DD	5	10%	8,111	3,547.22	260.47	486.09	846.09	1,374.76
DD	10	10%	18,137	6,670.93	317.82	568.23	1,072.58	1,936.42
DD	20	10%	33,100	11,005.49	344.59	653.66	1,242.03	2,190.15
NCI1	5	10%	1,819	431.19	89.12	137.75	185.22	221.21
NCI1	5	1%	21,306	900.68	615.62	920.17	1,227.52	1,378.18
NCI1	20	10%	2,441	557.70	115.07	183.54	220.14	255.58
NCI109	5	10%	2,182	462.62	115.62	170.43	206.23	254.70
NCI109	5	1%	19,099	886.06	532.38	727.15	1057.18	1,348.27
NCI109	20	10%	2,907	598.36	110.42	175.76	226.07	284.92

Table 1. Average number of subtree isomorphism test per graph for several datasets with varying number k of sampled spanning trees and frequency thresholds θ. The table reports $|\mathcal{F}|$ and the average number of subtree isomorphism tests evaluated by the naive method and by Algorithm 2 for $K = 32, 64, 128, 256$ (last four columns).

isomorphism tests than the brute-force one requiring $|\mathcal{F}|$ and outperforms also the naive algorithm in almost all cases. For example, on DD for $k = 10$ and $\theta = 10\%$, the naive algorithm evaluates subtree isomorphism for 11,006 patterns per graph on average, which is roughly one third of the total pattern set ($|\mathcal{F}|$), while our method evaluates subtree isomorphism 345 times on average for sketch size 32, ranging up to 2190 times for sketch size 256. In general, the naive algorithm performs 4 (resp. 1.7) times as many subtree isomorphism tests as our method for $K = 32$ (resp. $K = 256$). This is a significant improvement in light of the speed $O\left(n^{2.5}/\log n\right)$ of the fastest subtree isomorphism algorithm [9].

4.2 Positive Instance Retrieval

In this section we evaluate the performance of our approach in terms of precision for retrieving similar molecules for a given active compound in the NCI-HIV dataset. We use a simple setup to evaluate the quality of the min-hash based similarity in comparison to the exact Jaccard similarity as well as to the similarities obtained by the path min-hash kernel [11].

For each molecule of class A (i.e., "active"), we retrieve its i nearest neighbors (excluding the molecule itself) from the dataset and take the fraction of the neighbors of class A. This measure is known in the Information Retrieval community as *precision at i*. As a baseline, a random subset of molecules from NCI-HIV is expected to contain less than 1% of active molecules due to the highly skewed

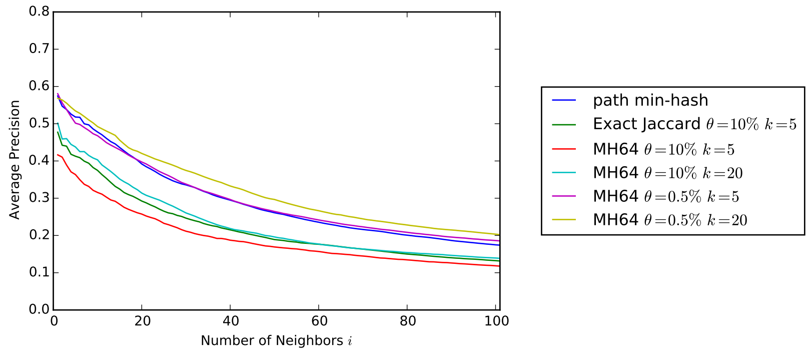

Fig. 2. Average fraction of "active" molecules among the i nearest neighbors of positive molecules in NCI-HIV dataset for path min-hash [11], exact Jaccard-similarity for frequent probabilistic tree patterns, and for our method with $K = 64$.

class distribution. In contrast, all methods show a drastically higher precision for the closest up to 100 neighbors on average.

Figure 2 shows the average precision at i (taken over all 329 active molecules) for i ranging from 1 to 100. The number k of sampled spanning trees per graph, as well as the frequency threshold θ has a strong influence on the quality of our method. To obtain our results, we have sampled 5 (resp. 20) spanning trees for each graph and used a random sample of $4,000$ graphs to obtain pattern sets for thresholds $\theta = 10\%$ and $\theta = 0.5\%$ respectively. We plot the min-hash-based precision for the four feature sets obtained in this way by our algorithm as a function of i for sketch size $K = 64$. We have compared this to the precision obtained by the exact Jaccard-similarity for $\theta = 10\%$ and $k = 5$ as well as to the precision obtained by path min-hash [11], both for the same sketch size $K = 64$.

The average precision obtained by using the exact Jaccard-similarities is slightly better than that of path min-hash. While our method performs comparably to path min-hash for $\theta = 0.5\%$ and $k = 5$, for $\theta = 0.5\%$ and $k = 20$ spanning trees it outperforms all other methods.

We were able to compute the precisions for the exact Jaccard-similarity neither for $\theta = 1\%$ nor for $k = 20$ sampled spanning trees. The Python implementation we used to calculate the similarity computations for exact Jaccard-similarity was not able to deal with the high dimensionality of the feature space, independently of the sparsity of the feature vectors. This indicates that the space required to compute the Jaccard-similarity is crucial for high-dimensional feature spaces.

4.3 Predictive Performance

In this section we empirically analyze the predictive performance of our method. The Jaccard-similarity induces a positive semi-definite kernel on sets, also known as a special case of the Tanimoto kernel (see, e.g., [8]). Interestingly, its approximation based on min-hashing is a kernel as well. Hence, we can use the tree pattern sets, resp. the min-hash sketches together with these two kernels in a support vector machine to learn a classifier. We have used 5-fold cross-validation and report the average area under the ROC curve obtained for the datasets MUTAG, PTC, DD, NCI1, and NCI109. We omit the results with NCI-HIV because LibSVM was unable to process the Gram matrix for this dataset. We note, however, that our algorithm required less than 10 (resp. 26) minutes for sketch size $K = 32$ (resp. $K = 256$) for computing the Gram matrix for the full set of NCI-HIV, while this time was 5.5 hours for the exact Jaccard-similarity. The runtimes of the preprocessing step (3 minutes) are not counted for both cases.

To this end, we have fixed the number of random spanning trees per graph to $k = 5$ (resp. $k = 20$) and sampled 10% of the graphs in a dataset to obtain the probabilistic frequent subtree patterns of up to 10 vertices. We report the results for the frequency threshold of $\theta = 10\%$ for our min-hash method (MH*) with sketch sizes K varying between 32 and 256, as well as for the exact Jaccard similarity (Jaccard) based on the same feature set. A lower frequency threshold is practically unreasonable e.g. for MUTAG, as it contains only 188 compounds. We have compared the results obtained with our previous results in [12] that use a radial basis function kernel on the probabilistic subtree features (PSK) and with the frequent subgraph kernel (FSG) [3] using the full set of frequent connected subgraphs of up to 10 vertices with respect to the *full* datasets (i.e., not only for a sample of 10%). We also report results of the Hash Kernel (HK) [10], which uses count-min sketching on sampled induced subgraphs up to size 9 to answer similarity queries.

Table 2 shows the results for frequency threshold $\theta = 10\%$ and $k = 5$ sampled spanning trees per graph. Jaccard and MH256 outperform PSK and even FSG on all datasets. On all datasets, except for PTC, the quality increases with the sketch size K. While HK is the best by a comfortable margin on MUTAG, the contrary is true for MH256, PSK, and Jaccard on DD. Most notably, MH usually outperforms PSK even for smaller sketch sizes and in many cases even the full frequent subgraph kernel FSG, while still being computable on DD, which has an enormous number of frequent patterns (compare Sec. 4.1). Note that the full datasets were used to generate the feature sets for PSK and FSG.

We also conducted experiments for $k = 20$ sampled spanning trees. For identical frequency threshold, the AUC improved by 3% on MUTAG, while only slightly changing for the other datasets.

5 Concluding Remarks

Algorithms 1 and 2 computing the min-hash sketches for a given tree pattern set and for subtree isomorphism can easily be adapted to any finite pattern set and pattern matching operator (e.g., homomorphism) if the pattern matching operator induces a pre-order on the pattern set. While the number of evaluations of the pattern matching operator can drastically be reduced in this way, the complexity of the algorithm depends on that of the pattern matching operator.

The one-sided error of our probabilistic subtree isomorphism test seems to have no significant effect on the experimental results. This raises the question

θ	Method	MUTAG	PTC	DD	NCI1	NCI109
10%	$MH32$	87.84	58.97	77.58	77.36	77.48
10%	$MH64$	87.73	58.68	79.91	78.04	79.54
10%	$MH128$	87.59	56.97	82.07	79.94	79.94
10%	$MH256$	87.78	57.18	83.58	80.76	81.72
10%	Jaccard	89.04	57.72	85.38	82.28	82.41
10%	PSK	84.22	54.17	84.67	79.09	78.05
10%	FSG	87.34	56.76	82.20	81.66	81.55
	HK	93.00	62.70	81.00	n/a	n/a

Table 2. AUC values for our method (MH) for sketch sizes $K = 32, 64, 128, 256$, $k = 5$ spanning trees per graph, and frequency threshold $\theta = 10\%$ to obtain the feature set. "n/a" indicates that the autors of [10] did not provide results for the respective datasets.

whether we can further relax the correctness of subtree isomorphism obtaining an algorithm that runs in at most subquadratic time without any significant negative effect on the predictive/retrieval performance.

References

1. A. Z. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29. IEEE, 1997.
2. A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *J. Comput. Syst. Sci.*, 60(3):630–659, 2000.
3. M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis. Frequent substructure-based approaches for classifying chemical compounds. *Trans. on Knowl. and Data Eng.*, 17(8):1036–1050, 2005.
4. R. Diestel. *Graph Theory*, volume 173. Springer, 2012.
5. H. Geppert, T. Horváth, T. Gärtner, S. Wrobel, and J. Bajorath. Support-vector-machine-based ranking significantly improves the effectiveness of similarity searching using 2d fingerprints and multiple reference compounds. *Journal of Chemical Information and Modeling*, 48(4):742–746, 2008.
6. T. Horváth, B. Bringmann, and L. D. Raedt. Frequent hypergraph mining. In *Inductive Logic Programming, 16th Intern. Conf., ILP 2006*, pages 244–259, 2006.
7. T. Horváth and J. Ramon. Efficient frequent connected subgraph mining in graphs of bounded tree-width. *Theor. Comput. Sci.*, 411(31-33):2784–2797, 2010.
8. L. Ralaivola, S. J. Swamidass, H. Saigo, and P. Baldi. Graph kernels for chemical informatics. *Neural Networks*, 18(8):1093–1110, 2005.
9. R. Shamir and D. Tsur. Faster subtree isomorphism. In *Theory of Computing and Systems, 1997*, pages 126–131. IEEE, 1997.
10. Q. Shi, J. Petterson, G. Dror, J. Langford, A. J. Smola, and S. V. N. Vishwanathan. Hash kernels for structured data. *J. Mach. Learn. Res.*, 10:2615–2637, 2009.
11. C. H. C. Teixeira, A. Silva, and W. M. Jr. Min-hash fingerprints for graph kernels: A trade-off among accuracy, efficiency, and compression. *JIDM*, 3(3):227–242, 2012.
12. P. Welke, T. Horváth, and S. Wrobel. Probabilistic frequent subtree kernels. In *NFMCP 2015, Springer LNCS 9607*, pages 179–193, 2015.