Probabilistic Frequent Subtree Kernels

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

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

Abstract. We propose a new probabilistic graph kernel. It is defined by the set of frequent subtrees generated from a small random sample of spanning trees of the transaction graphs. In contrast to the ordinary frequent subgraph kernel it can be computed efficiently for any arbitrary graphs. Due to its probabilistic nature, the embedding function corresponding to our graph kernel is not always correct. Our empirical results on artificial and real-world datasets, however, demonstrate that the graph kernel we propose is much faster than other frequent pattern based graph kernels, with only marginal loss in predictive accuracy.

1 Introduction

Over the past decade, graph kernels (see, e.g., [6]) have become a well-established approach in graph mining for their excellent predictive performance. One of the early graph kernels, the *frequent subgraph kernel* [5], is based on an explicit embedding of the graphs into a feature space spanned by the set of *all* connected subgraphs that are frequent w.r.t. the input graph database. It was shown in [5] that remarkable predictive accuracies can be obtained with this type of graph kernels on the molecular graphs of small pharmacological compounds.

One of the main drawbacks of frequent subgraph kernels is that the preprocessing step of generating *all* frequent connected subgraphs is computationally intractable [8]. Many of the practical implementations ignore this complexity limitation, implying that such systems may become infeasible in practice even for small datasets. For example, the general-purpose frequent subgraph mining algorithm FSG [5] requires more than one day to compute all frequent patterns even for small-sized datasets (50 graphs) of random *sparse* graphs with 25 vertices on average. Approaches that do not disregard the computational limitation mentioned above resort either to various heuristics for traversing the search space that result in incomplete output (e.g. [3]) or restrict the input graphs to some tractable graph class to guarantee both completeness and efficiency (e.g. [9]).

In this work we propose a new approach different from the ones above. We present a *randomized frequent subtree* kernel that is *not* restricted to any particular graph class and is computable in time *polynomial* in the number of frequent patterns generated by the algorithm. It follows from the negative complexity result [8] on mining frequent connected subgraphs from arbitrary graphs that, unless P = NP, such a frequent subgraph kernel can only be achieved by relaxing the requirements. For this work, we give up the demand on completeness and

calculate a binary feature vector for each graph with the following method: (i) We represent each input graph by a forest formed by k random spanning trees for some *small* k, (ii) compute the set of subtrees frequent in the forest database generated in step (i), and (iii) map the input graphs to the vertices of the Hamming cube in the space spanned by the set of frequent subtrees calculated in (ii). For complexity reasons, a frequent tree pattern is regarded as a subtree of a particular graph in step (iii) only if it is a subtree of the random forest generated for the graph. Combining the facts that a random sample of k spanning trees is always a forest and that it can be generated in polynomial time with the positive result that frequent subtree mining in forests can be solved with polynomial delay (see, e.g., [2,9]), we arrive at an algorithm computing the feature vector for any graph in time *polynomial* in the combined size of the input database and the set of generated tree patterns.

Regarding steps (i)–(ii), our approach is sound, but incomplete for two reasons: First, the pattern language is restricted to trees. Second, some frequent subtrees may be missed by the algorithm, as they are not necessarily frequent w.r.t. the random forest generated in step (i). Regarding (iii), this step is sound, but incomplete as well. We still resort to this probabilistic strategy, as deciding if a tree is subgraph isomorphic to an arbitrary graph is NP-complete. Our somewhat unusual idea is motivated by the intuition that any tree found by our mining algorithm is not only frequent with respect to the database, but with high probability it has a relatively high frequency also in the set of spanning trees for each transaction graph containing it. Thus, there must be a high chance that such a tree pattern will be detected with this method in a query graph as well, if it is part of it.[3] Hence we call our method probabilistic subtree kernel. It is significantly different from other techniques commonly called "tree kernels" (e.g. [10,12]), as these (i) use homomorphism instead of subgraph isomorphism as the embedding operator and (ii) are not frequency based.

We have empirically evaluated the proposed method on random and on benchmark molecular graph datasets. In particular, we first generated sparse random graphs in the Erdős-Rényi model [11] and investigated the recall of the set of frequent tree patterns for various k (i.e., number of random spanning trees per graph), as well as for different edge factors. (Notice that precision is always 100% for the soundness of the algorithm.) We found that at least 20% of *all* frequent subtrees can be recovered from a *single* random spanning tree per graph; for $k = 20$ the recall varies at around 90%. A similar or even better recall could consistently be observed on different real-world molecular datasets. Not surprisingly, our technique is faster by at least one order of magnitude from an average edge density of around 1.6. In all of our experiments we used the FSG frequent subgraph mining algorithm [5]. From an edge factor of around 2.0, FSG had to be aborted after one day while our algorithm required less than three hours (for frequency threshold 10%). Using different real-world molecular datasets, in a second step we then compared the predictive performance of our

[3] We assume that the query graph has been selected from the same (unknown) probability distribution as the graphs in the input database.

probabilistic approach to that of the ordinary frequent subgraph kernel [5]. We observed only a marginal loss in predictive performance on all datasets. In all of these experiments k was at most 20, in accordance with the recall results on random graphs. We found that with increasing dataset size we needed smaller and smaller values of k to obtain a close approximation of the frequent subgraph kernel's predictive performance. In particular, for the NCI-HIV dataset consisting of more than 40,000 molecular graphs, $k = 5$ sufficed. Putting together, the empirical results suggest that a careful composition of our simple probabilistic technique with some other fast graph kernel might result in a fast graph kernel of high predictive performance.

The rest of the paper is organized as follows. In Section 2 we present our algorithm with some important implementation details. Section 3 describes the empirical evaluation of our approach and Section 4 concludes with some interesting questions for further work.

2 The Probabilistic Frequent Subtree Kernel

Several graph kernels have been developed over the past decade for predictive graph mining. A broad range of these graph kernels belong to the class of *convolution kernels* [7]. That is, the input graphs are first decomposed into certain sets of substructures determined by some pattern language and the graph kernels are then defined by the intersection kernel over such sets. Depending on the particular choice of the substructure class (i.e., the pattern language), different graph kernels can be defined in this way. One of the first such graph kernels was defined by means of *frequent connected subgraphs* [5]. That is, the feature space corresponding to the kernel is spanned by the set of connected graphs that occur in at least a certain proportion of the graphs in the input database. The first step of this approach is to generate *all* frequent connected subgraphs, i.e., to solve the following pattern mining problem:

Frequent Connected Subgraph Mining (FCSM) Problem: *Given* a finite set $D \subseteq \mathcal{G}$ for some graph class $\mathcal{G}$ and a threshold $t \in (0, 1]$, *list* the set $F \subseteq \mathcal{P}$ of all pairwise non-isomorphic graphs from some graph class $\mathcal{P}$ that are subgraph isomorphic to at least $\lceil t \cdot |D| \rceil$ graphs in D.

In what follows, $\mathcal{G}$ and $\mathcal{P}$ will be referred to as *transaction* and *pattern* classes. The set F of frequent patterns in D naturally yields a binary vector representation for any arbitrary graph G: We map G to its characteristic vector $\overline{v}_G$ over the universe F, i.e., $\overline{v}_G$ is indexed by F and for all $H \in F$, $\overline{v}_G[H] = 1$ if and only if H is subgraph isomorphic to G. To avoid redundancies in the characteristic vectors over F, the patterns in F are required to be pairwise non-isomorphic.

One of the main limitations of the frequent subgraph kernel [5] is the computational intractability of the FCSM problem: If there is no restriction on the transaction graphs in $\mathcal{G}$ and $\mathcal{P}$ consists of all connected graphs of $\mathcal{G}$ then, unless P = NP, the FCSM problem cannot be solved in output polynomial time [8].

As our empirical results of the next section clearly indicate, this negative complexity result makes the frequent subgraph kernel practically infeasible even for small-sized datasets of small sparse graphs. To overcome this limitation, below we propose a probabilistic frequent *subtree* kernel that can be calculated in time polynomial in the combined size of G_1, G_2, and the set of frequent patterns for any *arbitrary* graphs G_1 and G_2.

2.1 Probabilistic Frequent Subtrees

To achieve the goal above, we restrict the pattern language to trees and relax the correctness constraint of the FCSM problem by giving up the requirement of completeness relative to the constrained pattern language of trees. Just restricting the pattern language to trees is not sufficient to get rid of the computational intractability mentioned above; mining frequent trees in arbitrary graphs is not possible in output polynomial time, as otherwise it could solve the Hamiltonian path problem in polynomial time [8]. To overcome this problem, we propose a simple probabilistic approach that proved fast yet powerful enough in all of our empirical experiments. For the sake of simplicity, we assume that the transaction graphs are connected by noting that our algorithm can naturally be generalized to disconnected transaction graphs.

Our approach is very simple: For each transaction graph we first generate a forest formed by a sample of k random spanning trees for some small k, then solve the FCSM problem for this random forest database, and finally use the set of output patterns to define the underlying feature space. With this problem relaxation we arrive at an easy to implement and, as shown in Section 3, practically effective frequent subgraph mining algorithm (see Algorithm 1). In addition to the transaction database D and the frequency threshold t given in the definition of the FCSM problem, the input contains an additional parameter $k \in \mathbb{N}$ defining an upper bound on the number of spanning trees to be generated for each transaction graph. The algorithm starts by sampling k spanning trees for each graph in the database. Instead of mining frequent patterns in the input database D directly, we replace each graph G by a forest F_G formed by the vertex disjoint union of the random spanning trees generated for G. This effectively reduces the problem of mining frequent subtrees in arbitrary graph databases D to that of mining frequent subtrees in a database D' consisting of forests. A tree T is regarded to be t-frequent in this setting if and only if it is subgraph isomorphic to at least $\lceil t \cdot |D'| \rceil = \lceil t \cdot |D| \rceil$ forests in D'. As frequent subtree mining in forest databases can be done with polynomial delay [2,9], we arrive at an algorithm that runs in time polynomial in the combined size of D and the set of frequent subtrees in D'.

To distinguish between the output F of the frequent subgraph problem and the output F' of Algorithm 1 on D and t, we will refer to the former set as *frequent patterns* and to the later one as *probabilistic (subtree) patterns* with respect to a threshold t. Clearly, for any D, t, and k, the output of Algorithm 1 is a subset of the set of frequent trees in D, i.e., Algorithm 1 is sound. However,

it will not necessarily find all frequent patterns, i.e., it is not complete in general. Thus, with this technique, on the one hand we obtain a polynomial time algorithm that is fast for small values of k, on the other hand, however, loose a number of frequent patterns.

A further complexity problem arises when calculating the explicit embedding of a graph into the feature space defined above: Given a set F' of probabilistic tree patterns generated by Algorithm 1 and a graph G, the embedding of G into the feature space spanned by F' cannot be computed in polynomial time (if P $\neq$ NP). The reason is that deciding subgraph isomorphism from a tree into an arbitrary graph is NP-complete. Therefore, we allow the embedding to be incorrect and use the following probabilistic embedding based also on a random sample of k spanning trees: Given a tree pattern $T \in F'$ and a graph G,

1. use F_G generated in steps 3–4 of Algorithm 1, if $G \in D$; o/w generate a random forest F_G for G as in steps 3–4 of Algorithm 1,
2. set the entry $\overline{v}_G[T] = 1$ in the binary feature vector $\overline{v}_G$ of G to 1 if and only if T is a subgraph of F_G.

Clearly, this embedding is not unique because it depends on the randomly generated forest F_G. In the application context of graph kernels, the incompleteness of Algorithm 1 and the incorrectness of the probabilistic embedding sketched above raise two important questions:

1. How stable is the output of Algorithm 1 and what is its recall with respect to *all* frequent subtrees?
2. How does our probabilistic approach influence the predictive performance of the graph kernel obtained?

Regarding the first question, we show in the next section on artificial and real-world chemical graph datasets that (i) the output is very stable even for $k = 1$ and (ii) more than 75% of the frequent patterns can be recovered by using only ten random spanning trees per graph (i.e., for $k = 10$). The high stability and recall together indicate that the probabilistic embedding of G calculated by the above method has a small Hamming distance to the exact one defined by F.

Regarding the second question, we show on different real-world benchmark graph datasets that our experimental results are comparable with those obtained by the FSG algorithm [5], even for the full set of frequent *subgraphs*. Before presenting these and other empirical results in Section 3, we first discuss some implementation issues and analyse the time complexity of Algorithm 1.

2.2 Implementation Issues and Runtime Analysis

Line 3 of Algorithm 1 can be implemented using Wilson's algorithm [13], which has an expected runtime that is linear in the *mean hitting time* of a graph and returns each spanning tree of G with the same probability. It is $\Theta(n^3)$ in the worst case, but conjectured to be much smaller for most graphs [13]. The set of all sampled spanning trees *up to isomorphism* (Line 4) can be computed from

Given: A graph database $D \subseteq \mathcal{G}$ an integer $k > 0$ and an integer threshold $t > 0$.
Output: A set of t-frequent subtrees of D.

1: $D' := \emptyset$
2: **for all** $G \in D$ **do**
3: Sample k spanning trees of G uniformly at random
4: Add the forest F_G of those trees up to isomorphism to D'
5: List all t-frequent subgraphs in D'

Algorithm 1: The Probabilistic Subtree Mining Algorithm

the set of sampled spanning trees using some canonical string representation for trees and a prefix tree as data structure (see, e.g., [2] for more details on canonical string representations for labeled graphs). We follow this approach to practically reduce the runtime of the subsequent frequent subtree mining step, as isomorphic spanning trees yield the same subtrees and can safely be omitted. For each tree, this can be done in $O(n \log n)$ time by computing first the tree center and then applying a canonical string algorithm for rooted trees as in [2]. These canonical strings are then stored in and retrieved from a prefix tree in time linear in their size.

Thus, the sampling step of our algorithm runs in expected $O(kn^3)$ time. If we do not require the spanning trees to be drawn uniformly, we can improve on this time and achieve a deterministic $O(km \log n)$ runtime, where m denotes the number of edges. This is achieved by choosing a *random* permutation of the edge set of a graph and then applying Kruskal's minimum spanning tree algorithm using this edge order. It is not difficult to see that this technique can generate random spanning trees with non-uniform probability. As our experimential results on molecular graphs of pharmacological compounds show, this has no significant impact on the predictive performance of the graph kernel obtained.

Finally we note that for Line 5, we can use almost any one of the existing algorithms generating frequent connected subgraphs (i.e., subtrees) from forest databases (see, e.g., [2] for an overview on this topic).

3 Experiments

In this section we empirically evaluate our probabilistic approach on artificial and real-world datasets. We start with artificial datasets to study various features of the ordinary and our probabilistic approach on increasingly complex datasets. We then evaluate the two methods on real-world datasets to complement our results on the artificial datasets and to investigate the suitability of our graph kernel for predictive tasks.

In all our experiments, we used FSG [5] in the version provided by the authors[4] to generate frequent subgraphs. In Line 5 of Algorithm 1, we also used FSG to generate frequent subtrees. In this way, we can consistently compare the

[4] `http://glaros.dtc.umn.edu/gkhome/pafi/overview`

runtimes of the two methods, as none of them is affected by some specific heuristic not used in the other one. However, we expect a significant improvement of our probabilistic method over the traditional one, once a specialized tree mining algorithm is applied. All our experiments were conducted on an Intel i7 CPU with 3.40GHz and 16GB of RAM.

3.1 Datasets

Any general frequent subgraph mining algorithm is expected to process a broad spectrum of graph databases. Most empirical evaluations, however, concentrate on some particular type of graph data, mostly representing small molecules. These graphs share certain properties, e.g. sparsity, small vertex degree, near planarity, and, in particular, a natural set of frequent patterns corresponding to functional groups. While all these properties (especially the last one) motivate frequent subgraph mining in the first place, it is also important to observe the behavior of a mining technique on data that may or may not have such properties. We therefore conducted experiments on artificial as well as on real-world molecular datasets.

Artificial Datasets All these datasets consist of unlabeled sparse graphs of varying number of vertices and edges that were generated in the Erdős-Rényi random graph model [11]. The datasets generated are of different structural complexity, where the structural complexity is defined as the *expected edge factor* $q = \frac{m}{n}$ (n is the number of vertices and m the number of edges). For a given q, each graph G in the corresponding dataset is generated as follows: We first draw the number n of vertices uniformly at random between 2 and 50, set the Erdős-Rényi edge probability parameter $p = \frac{2q}{n-1}$, and then generate G on n vertices in the usual way with this p. If the resulting graph is connected, we add it to the dataset.

MUTAG is a dataset of 188 connected compounds labeled according to their mutagenic effect on Salmonella typhimurium. On average, each graph has 20 vertices and 22 edges.

PTC contains 344 connected molecular graphs, labeled according to the carcinogenicity in mice and rats. The graphs have 26 vertices and edges on average.

NCI1, NCI109 consist of 4,110 (resp. 4127) compounds of which 3530 (resp. 3519) 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. The average number of vertices is 30, the average number of edges is 32 in both datasets[5].

NCI-HIV consists of 42,687 compounds of which 39,337 are connected[6]. The average number of vertices and edges per graph are 41 and 43, respectively.

[5] MUTAG, NCI1, NCI109, and PTC were obtained from `http://www.di.ens.fr/~shervashidze/code.html`.

[6] `http://cactus.nci.nih.gov/`

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). We consider the following three usual binary classification problems: (AMvsI) A and M together versus I, (AvsMI) A versus M and I, and (AvsI) A versus I where instances labeled by M are removed.

ZINC is a subset of $8,946,757$ ($8,946,755$ connected) so called 'Lead-Like' molecules from the zinc database of purchasable chemical compounds[7]. The molecules in this subset have a molar mass between $250g/mol$ and $350g/mol$ and have an average number of vertices and edges 43 and 44, respectively.

3.2 Runtime

In this section, we compare the runtime of FSG and our algorithm (using FSG as the mining subroutine) on artificial datasets and on subsets of the ZINC dataset. We use Wilson's method [13] to generate the random spanning trees and report the combined time for sampling and frequent pattern generation. As already noted, one could use a specialized frequent subtree mining algorithm in combination with our sampling method to further increase the speedup. We experimented with several such publicly available tree mining algorithms but, somewhat surprisingly, they were not able to beat the speed of FSG on the tree dataset.

Figure 1 shows the processing times for expected edge factors (q) varying between 1.0 and 5.0. (Note the log scale used for the y-axis.) We report average execution times over three runs for computing the set of frequent patterns and probabilistic patterns for various numbers of sampled spanning trees (k). It turns out that FSG is very sensible to the parameter q. In order to be able to get any result in reasonable time, we had to restrict the number of graphs in each dataset to 50. Still we had to terminate FSG in several cases where it took more than 24 hours (86,400s), which was consistently the case once q exceeded 1.8. Up to 20 sampled spanning trees, our probabilistic approach is always faster; for $q > 1.4$ it is faster even for $k = 50$ (more precisely, in contrast to FSG, it is able to terminate in significantly less than a day).

Figure 2 shows the time in seconds for our algorithm with $k = 1$ in black and for FSG in gray. It reports results with subsets of the ZINC dataset of increasing size. Though FSG was able to process much larger datasets than in the experiments with artificial datasets, our method always outperformed FSG in runtime on all datasets and for all frequency thresholds. Furthermore, with decreasing frequency threshold, the runtime of our method increases with a much smaller speed. Last but not least, in contrast to our method, FSG has a clear limitation beyond $200,000$ graphs for $t = 5\%$ and beyond $100,000$ graphs for $t = 2\%$.

[7] `http://zinc.docking.org/subsets/lead-like`

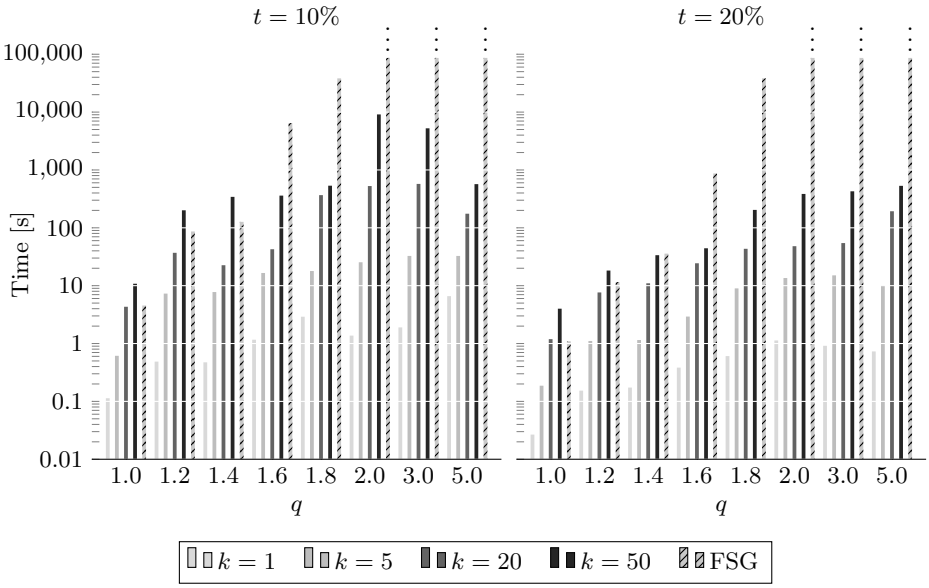

Fig. 1. Runtime of our method, compared to FSG on artificial datasets of varying expected edge factor q. Dots over bars signal that the run was terminated after 24 hours.

3.3 Recall

As discussed in Section 2, for any graph database the pattern set F' found by our algorithm is a subset of all frequent subtrees F_T, which in turn is a subset of all frequent subgraphs F. We now analyze the recall of our method, i.e. the amount of frequent subtree patterns that are found when applying Algorithm 1 for various k and t. To this end, let $R(k,t) := \frac{|F'|}{|F_T|}$ be the fraction of t-frequent tree patterns that are found if Algorithm 1 selects k random spanning trees. Using the FSG algorithm, on each dataset we first compute all frequent connected patterns, including non-tree patterns as well, and then filter out all frequent subgraphs that are not trees.

Figure 3 shows the recall $R(k,t)$ of our method of one run on artificial datasets for frequency thresholds 10% and 20%. It is restricted to expected edge factors $q \leq 1.8$, as beyond this value FSG was not able to compute the full set of frequent patterns in less than a day. Even for a single spanning tree (i.e., for $k = 1$), the recall is always above 20%; in most cases actually above 40%. The recall for $k = 5$ sampled spanning trees is drastically higher than for $k = 1$; in fact the increase in recall between $k = 5$ and $k = 50$ is much lower. This suggests that $k = 5$ might be a good compromise in the trade-off between runtime and accuracy of our method.

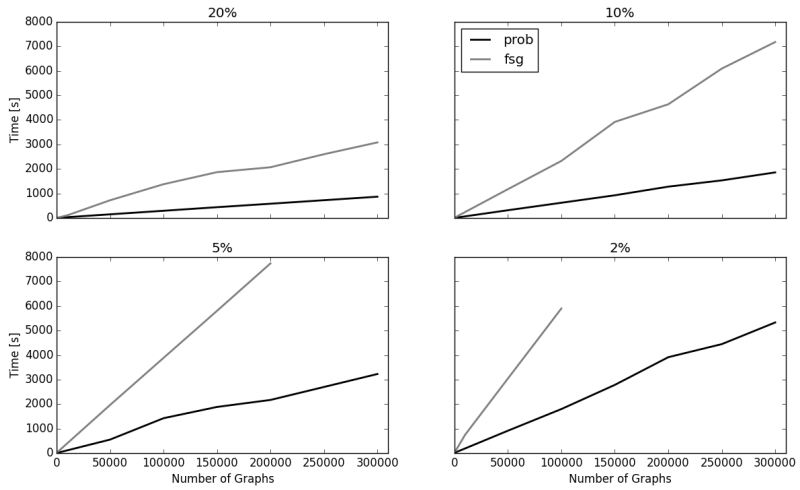

Fig. 2. Runtime results in seconds for our method (black) and FSG (grey), for different frequency thresholds. The x-values show the number of graphs in the subsets of ZINC that were used.

For NCI-HIV and ZINC, we sample 10 subsets of 100 graphs each and report the average value of $R(k,t)$ and its standard deviation. The results on the two datasets can be found in Table 1 for different values of k with frequency thresholds 5%, 10%, and 20%. We have found that at least 95% of all frequent subgraphs are trees. It can be seen as well that the fraction of the retrieved tree patterns rapidly grows with the number of sampled spanning trees per graph. Sampling 10 spanning trees per graph already results in around 90% recall for the ZINC dataset and in a recall of 80% for the NCI-HIV dataset.

3.4 Stability of Probabilistic Subtree Patterns

The results of Section 3.3 above indicate that a relatively high recall of the frequent tree patterns can be achieved on molecular graphs and our artificial datasets, even for a very small number of random spanning trees. In this section we report empirical results showing that the output pattern set of Algorithm 1 is quite stable (i.e., independent runs of our probabilistic tree mining yield similar sets of frequent patterns). To empirically demonstrate this advantageous property, we ran Algorithm 1 several times on the same values of the parameters k and t and observed how the union of the probabilistic tree patterns grows.

To this end, we fix two sets of graphs, each of size approximately $40,000$, as follows: We take all connected graphs in NCI-HIV, as well as a random subset ZINC_{40k} of ZINC that contains $40,000$ graphs. We run Algorithm 1 10 times for the datasets obtained with parameters $k = 1$ and $t = 10\%$. Each execution results in a set F_i' of probabilistic subtree patterns, from which we define

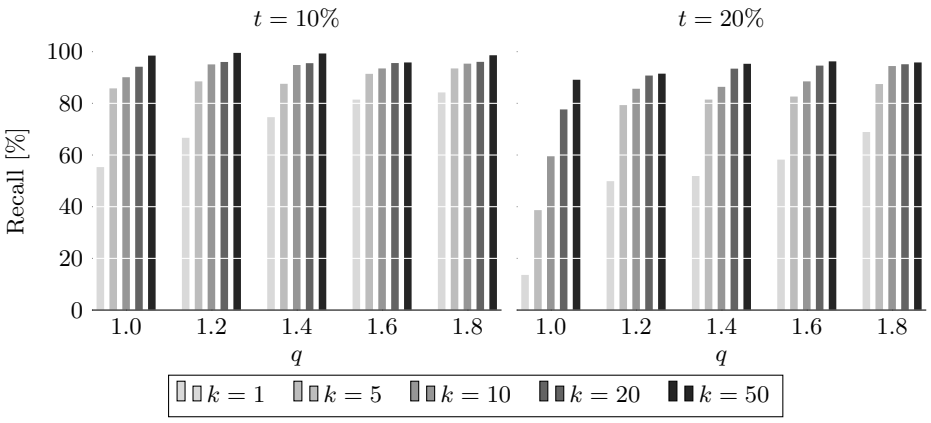

Fig. 3. Recall of our method on artificial graph databases with varying expected edge factor q, for frequency thresholds 10% and 20%.

		$k=1$	$k=2$	$k=3$	$k=10$	$k=20$
NCI-HIV	$t=5\%$	20.13 ± 1.20	35.53 ± 1.34	46.48 ± 0.51	78.32 ± 0.85	91.11 ± 1.29
	$t=10\%$	20.26 ± 2.22	34.45 ± 1.42	45.40 ± 1.59	79.94 ± 1.82	92.44 ± 1.34
	$t=20\%$	24.45 ± 1.38	39.76 ± 1.68	50.41 ± 1.14	83.38 ± 1.40	94.72 ± 1.31
ZINC	$t=5\%$	36.80 ± 0.87	56.70 ± 1.65	68.42 ± 0.94	92.50 ± 0.45	97.92 ± 0.55
	$t=10\%$	32.77 ± 1.89	51.36 ± 1.84	64.47 ± 1.40	92.49 ± 1.18	86.70 ± 22.83
	$t=20\%$	31.03 ± 2.59	48.99 ± 3.05	61.41 ± 3.41	90.53 ± 1.28	97.89 ± 0.40

Table 1. Recall with standard deviation of the probabilistic tree patterns on the NCI-HIV and ZINC datasets for frequency thresholds 5%, 10%, and 20%

$U_i = \bigcup_{j=0}^{i} F'_j$ with $F'_0 := \emptyset$. Table 2 reports $|F'_i \setminus U_{i-1}|$, i.e., the number of *new* probabilistic subtree patterns found in iteration i for $i = 1, \ldots, 10$ on the left. For an initial number of $3,920$ (NCI-HIV) and $9,898$ (ZINC_{40k}) probabilistic patterns, the number of newly discovered patterns drops to at most 22 for the following iterations.

We observed this behavior consistently on the artificial graphs (over all observed edge factors, all numbers of sampled spanning trees, and all frequency thresholds). Due to lack of space, Table 2 only shows the results for $t = 10\%$, $k = 10$, and 5 iterations on the right. As in the evaluation in Section 3.3, each artificial dataset consists of 50 graphs.

These results together show, that the generated feature set does *not* depend too much on the particular spanning trees selected at random. Overall, we have found that independent runs of our algorithm yield similar feature sets on the same data. This observation, combined with the remarkable recall results of the previous section, is essential for our kernel; high recall and stability together indi-

Iteration	1	2	3	4	5	6	7	8	9	10
NCI-HIV	3920	20	5	10	14	7	2	6	7	2
$ZINC_{40k}$	9898	18	17	11	10	22	7	7	9	1

Iteration	1	2	3	4	5
$q = 1.0$	692	2	5	8	3
$q = 1.2$	750	2	0	0	11
$q = 1.4$	806	18	0	0	0
$q = 1.6$	824	1	0	0	0
$q = 1.8$	824	2	0	0	0
$q = 2.0$	850	0	0	1	0
$q = 3.0$	814	26	1	4	0
$q = 5.0$	822	4	0	0	20

Table 2. Repetitions of the experiment with $t = 10\%$ and $k = 1$ sampled trees on NCI-HIV and ZINC (left), and $k = 10$ for random graphs with different edge factors q (right). The numbers reported are the number of probabilistic patterns that were not in the union of all probabilistic patterns found up to the current iteration.

cate that the predictive performance of the (computationally intractable) *exact* frequent subtree kernel can closely be approximated by our (computationally feasible) *probabilistic* frequent subtree kernel even for small values of k.

3.5 Predictive Performance

In this section we show that the predictive performance of the probabilistic subtree kernel compares favorably with that of the frequent subgraph kernel. We deliberately consider the more expressive complete output of FSG, including also cyclic patterns, because we compare the runtime of our method to that of FSG needed to compute *all* frequent subgraphs. We choose, as does most related work, a wrapper method and report the achieved area under the ROC-curve (AUC) of a well trained support vector machine (SVM) [4]. To this end, we consider the seven binary classification problems described in Section 3.1. We compare the predictive performance of (i) the frequent subgraph kernel computed by FSG [5] with that of (ii) the probabilistic frequent subtree kernel for different k and for different frequency thresholds. For (ii), we use only the results with Wilson's random spanning tree sampling algorithm [13]; we obtained nearly identical accuracy and runtime results with the greedy sampling algorithm based on Kruskal's method. For our evaluation, we use the SVM provided by the libSVM package [1] with a radial basis function kernel.

We repeat Algorithm 1 four times using different sets of sampled trees and report the average and standard deviation of AUC values from a 3-fold cross validation for each resulting feature set. The same procedure is applied to the frequent subgraph pattern set, here we use a different splitting for the cross validation in each run.

Table 3 shows the results for the classification problems on MUTAG, NCI1, NCI109, and PTC. We can see that the frequent subgraph kernel outperforms our probabilistic subtree kernels for all frequency thresholds and all choices of k. However, if we select $k = 20$ spanning trees, the accuracy is fairly close to

t	k	MUTAG	PTC	NCI1	NCI109
1%	1	81.72 ± 1.22	56.20 ± 1.54	79.73 ± 0.26	78.64 ± 0.20
1%	2	82.98 ± 0.46	57.03 ± 0.88	81.74 ± 0.22	80.89 ± 0.15
1%	5	85.47 ± 0.80	59.18 ± 0.54	83.45 ± 0.12	83.07 ± 0.14
1%	10	88.33 ± 0.30	59.67 ± 0.26	84.09 ± 0.10	83.79 ± 0.15
1%	20	89.32 ± 0.14	60.10 ± 0.09	84.43 ± 0.06	84.23 ± 0.05
1%	FSG	91.18 ± 0.46	63.62 ± 1.01	86.87 ± 0.10	86.84 ± 0.09
5%	1	80.79 ± 1.26	54.92 ± 1.69	76.90 ± 0.40	75.67 ± 0.23
5%	2	82.30 ± 0.41	55.05 ± 1.25	78.87 ± 0.17	77.73 ± 0.17
5%	5	84.20 ± 0.90	56.12 ± 0.67	80.75 ± 0.17	80.31 ± 0.16
5%	10	86.35 ± 0.15	56.14 ± 0.29	81.60 ± 0.10	81.12 ± 0.13
5%	20	87.66 ± 0.26	56.34 ± 0.19	82.15 ± 0.05	81.73 ± 0.05
5%	FSG	89.01 ± 0.64	58.00 ± 1.86	83.76 ± 0.13	83.86 ± 0.06
10%	1	80.99 ± 1.23	54.05 ± 1.84	75.41 ± 0.43	74.10 ± 0.28
10%	2	82.60 ± 0.44	54.35 ± 1.48	77.28 ± 0.22	76.08 ± 0.17
10%	5	84.22 ± 0.86	54.17 ± 0.87	79.09 ± 0.16	78.05 ± 0.14
10%	10	86.23 ± 0.16	53.94 ± 0.28	79.95 ± 0.09	79.01 ± 0.10
10%	20	86.95 ± 0.11	53.99 ± 0.19	80.44 ± 0.05	79.61 ± 0.07
10%	FSG	87.34 ± 0.46	56.76 ± 1.96	81.66 ± 0.10	81.55 ± 0.24
20%	1	81.02 ± 1.43	53.36 ± 2.16	72.78 ± 0.35	70.84 ± 0.32
20%	2	83.12 ± 0.53	53.05 ± 0.79	74.94 ± 0.22	73.77 ± 0.17
20%	5	84.68 ± 0.82	52.34 ± 0.89	77.05 ± 0.15	76.13 ± 0.11
20%	10	86.92 ± 0.16	51.86 ± 0.52	77.79 ± 0.06	76.90 ± 0.10
20%	20	88.10 ± 0.06	51.97 ± 0.22	78.15 ± 0.06	77.33 ± 0.08
20%	FSG	88.36 ± 0.00	55.82 ± 2.59	77.41 ± 0.09	77.92 ± 0.02

Table 3. AUC values [%] of an SVM classifier on MUTAG, NCI1, NCI109, and PTC for frequency thresholds t between 1% and 20% when using features generated by FSG and our method for $k \in \{1, 2, 5, 10, 20\}$.

that of the exact frequent subtree kernel for all datasets and for all frequency thresholds. Furthermore, the results suggest that we can achieve or perhaps even increase the predictive accuracy of the exact frequent subgraph kernel at a certain frequency threshold t by using the probabilistic frequent subtree kernel with parameters $k = 20$ and frequency threshold $t/2$. It is also worth noting that the increase of accuracy slows down in the function of k; the gain of moving from 1 to 5 spanning trees is much larger than that from 5 to 10 on all datasets except MUTAG, where the second increase is comparable to the first. We assume that this behavior on MUTAG is due to the small number of molecules in the dataset.

The results on NCI-HIV are presented in Table 4. On the one hand, one can see that from a frequency threshold of 10%, the results with the frequent subgraph kernel are more stable than those with the probabilistic frequent subtree kernel on all three problems. Though the frequent subgraph kernel outperforms the probabilistic frequent subtree kernel on the same frequency threshold, the difference seems marginal once we compare the best results on each problem,

t	k	AvsI	AMvsI	AvsMI
5%	FSG	o.o.m	o.o.m	o.o.m
5%	1	89.27 ± 0.20	72.35 ± 0.23	88.23 ± 0.24
5%	2	89.94 ± 0.12	74.09 ± 0.69	89.09 ± 0.74
5%	5	91.17 ± 0.13	75.65 ± 0.27	90.63 ± 0.17
10%	FSG	91.31 ± 0.38	75.29 ± 0.24	90.82 ± 0.31
10%	1	88.53 ± 0.81	71.32 ± 0.54	87.45 ± 1.18
10%	2	88.28 ± 1.51	71.09 ± 0.21	87.29 ± 0.62
10%	5	91.11 ± 0.23	74.30 ± 0.18	90.27 ± 0.08
20%	FSG	91.35 ± 0.39	74.24 ± 0.26	90.57 ± 0.17
20%	1	86.75 ± 0.76	68.55 ± 0.73	86.00 ± 0.74
20%	2	86.40 ± 1.00	68.79 ± 0.61	85.79 ± 0.74
20%	5	90.29 ± 0.28	73.17 ± 0.56	90.27 ± 0.53

Table 4. Average AUC values for the three learning problems on the NCI-HIV benchmark dataset for the frequent subgraph kernel and the probabilistic frequent subtree kernel for $k = 1, 2$ and for different frequency thresholds.

especially in light of the runtime benefits presented above. On the other hand, however, for the frequent subgraph kernel, the results could be calculated only for $t = 10\%$, while for the probabilistic frequent subtree kernel we obtained the result in half of the time for $t = 5\%$. For this frequency threshold, FSG was unable to produce any result because it ran out of memory. For larger frequency thresholds, we had difficulties with training the SVM using all frequent patterns because of its excessive memory usage. These observations clearly show the limitation of the frequent subgraph kernel over the probabilistic frequent subtree kernel when the predictive performance required can be achieved only for low frequency thresholds. Finally we note that there is no improvement when sampling two instead of one spanning tree per graph, but a drastic increase when increasing k to 5. This result fits well with the evaluation of our method on the artificial datasets.

4 Conclusion and Future Work

We have presented a kernel for graph structured data that is based on probabilistic subtree patterns, i.e., on frequent subtrees in a forest database obtained by randomly selecting k spanning trees for each transaction graph in the input database and for some small value of k. Our empirical results on various random graph datasets generated in the Erdős-Rényi model show that even for small values of k ($k \leq 20$), the output of the probabilistic frequent subtree mining algorithm is of high recall and stability. This implies that the predictive performance of the corresponding probabilistic frequent subtree kernel must closely approximate to that of the exact frequent subtree kernel. Runtime comparisons with the FSG frequent subgraph mining algorithm clearly demonstrate the superiority of our probabilistic approach; the speed of the probabilistic frequent subtree mining algorithm is faster by at least one order of magnitude.

Our empirical results on various real-world benchmark graph datasets show that the probabilistic feature space considered is expressive enough in terms of predictive performance compared to that of the ordinary frequent subgraph kernel. Furthermore, our graph kernel is not only faster than the frequent subgraph kernel, but has a much smaller memory footprint in all stages.

We are currently working on some formal properties of the proposed method, e.g., on probabilistic guarantees for (t_1, t_2)-frequent subtrees where t_1 is a frequency threshold for a tree pattern within the set of spanning trees of a graph, whereas t_2 within the database. These results will then be turned into an algorithm that, for a given confidence value δ specified by the user, generates each frequent subtree with probability at least δ. We are also considering the design and implementation of a frequent subtree mining algorithm for unlabeled free trees that is able to effectively process massive forest transaction datasets.

One of the strengths of our method is that it is not restricted to any particular graph class. This advantageous property allows us to empirically investigate the proposed graph kernel on more complicated graph classes beyond molecular graphs, such as the k-neighborhood graphs of the web graph or RDF graphs.

References

1. C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.
2. Y. Chi, R. R. Muntz, S. Nijssen, and J. N. Kok. Frequent subtree mining - an overview. *Fundam. Inform.*, 66(1-2):161–198, 2001.
3. D. J. Cook and L. B. Holder. Substructure discovery using minimum description length and background knowledge. *J. Artif. Intell. Res. (JAIR)*, 1:231–255, 1994.
4. C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.
5. M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis. Frequent substructure-based approaches for classifying chemical compounds. *Knowledge and Data Engineering, IEEE Transactions on*, 17(8):1036–1050, 2005.
6. T. Gärtner. *Kernels for structured data*. PhD thesis, Universität Bonn, 2005.
7. D. Haussler. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, Univerisity of California - Santa Cruz, July 1999.
8. T. Horváth, B. Bringmann, and L. De Raedt. Frequent hypergraph mining. In *Inductive Logic Programming*, pages 244–259. Springer, 2007.
9. 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.
10. P. Mahé and J.-P. Vert. Graph kernels based on tree patterns for molecules. *Machine learning*, 75(1):3–35, 2009.
11. A. Rényi and P. Erdős. On random graphs. *Publicationes Mathematicae*, 6(290-297):5, 1959.
12. N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *The Journal of Machine Learning Research*, 12:2539–2561, 2011.
13. D. B. Wilson. Generating random spanning trees more quickly than the cover time. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 296–303. ACM, 1996.