

# Mining Interpretable Subgraphs

Siegfried Nijssen

Institut für Informatik, Albert-Ludwigs-Universität,  
Georges-Köhler-Allee, Gebäude 097, D-79110, Freiburg im Breisgau, Germany.  
`snijsen@informatik.uni-freiburg.de`

**Abstract.** We present a measure that estimates the interpretability of a frequent subgraph. We show that a feature selection algorithm that uses this measure creates a set of features that is smaller and equally predictive as features obtained in earlier studies. A significant number of the selected features turn out to be trees or cyclic graphs, leading us to the conclusion that such features are not as useless as suggested in some earlier studies. Finally, we show that a constraint on this measure can be pushed in the mining process, thus leading to faster discovery of interesting subgraphs.

## 1 Introduction

One of the most important applications of graph mining is the analysis of molecular datasets, where the aim is to predict accurately if an unseen molecule exhibits a certain chemical activity or not. Traditional methods proposed in the cheminformatics community rely mostly on paths, but the data mining community has started the use of subgraphs in the hope of obtaining better classifiers, where both frequent pattern mining [10, 7] and kernel approaches [6, 9] have been investigated.

In two recent papers, however, it was suggested that the additional complexity of taking into account subgraphs may not be justified [2, 9]. It was reported that for a large number of classifiers and a large number of datasets, features based on paths yield more accurate classifiers.

In the local pattern mining approach, which is used for example in [2], classification is typically approached as follows. First, given a dataset, a set of patterns (in this case, paths or subgraphs) is computed. During this search, a constraint is enforced such that only patterns are found that have sufficient correlation with the activity of the molecules; for instance, the subgraph should occur more often in the active molecules than in the inactive ones.

Second, each pattern is used to create a new feature for the molecules: if a molecule contains a certain pattern, the feature corresponding to the pattern gets value 1 (or true), otherwise the feature has value 0 (or false). On the resulting set of features, a classifier, such as a decision tree, is learned. This decision tree is finally used to predict the activity of new molecules.

It is indeed an interesting observation that a set of paths can contain the same amount or even more information than a set of subgraphs. An equally

interesting question is how these results can be explained. We believe that there are several possible explanations:

- in [2] feature sets of equal size were determined for each type of pattern; if one considers the contingency tables that achieve the best correlation values, there are usually significantly more subgraphs with exactly that table than there are subpaths, yielding features that are more correlated among each other in case a fixed number of subgraphs are used;
- if subgraphs are considered, the amount of possible features is much larger; thus the risk of overfitting to the training data is also much larger.

In the study of [2] it was reported that by relaxing the correlation constraint, a more accurate classifier can be obtained. One could explain this by the argument that the resulting additional features are in this case less correlated among each other.

In this paper, we investigate these observations further. Concretely, we propose a measure to drastically cut down the number of frequent subgraphs that are used as features, thus yielding classifiers that are more interpretable, and we propose a feature selection method to avoid redundancy among the features. We show that the interpretability constraint can be pushed in the mining process, thus showing that there are efficient ways to search for patterns that are still simple, but also cyclic.

The paper is organized as follows. In section 2 we briefly reintroduce the formal concepts that are important in this paper. In section 3 we introduce our notion of interpretable subgraphs. In section 4 we show how a constraint on interpretable subgraphs can be pushed in the mining process. Section 5 introduces our feature selection algorithm. Section 6 concludes.

## 2 Concepts

An essential task in the local pattern mining scenario is the computation of a relevant set of patterns. In the data mining community, these features are often determined using a *frequent subgraph mining* algorithm. To formalize what a frequent subgraph is, we need the following definitions.

**Definition 1 (Graphs).** *An undirected, labeled graph  $G(V, E, \lambda, \Sigma)$  consists of a finite set  $V$  of vertices, a set  $E \subseteq \{\{u, v\} | u, v \in V, u \neq v\}$  of edges, an alphabet  $\Sigma$  and a labeling function  $\lambda : (V \cup E) \rightarrow \Sigma$ .*

**Definition 2 (Connected Graphs).** *A graph  $G(V, E, \lambda, \Sigma)$  is called connected iff for each pair of nodes  $u, v \in V$  there is a sequence of nodes  $v_1, \dots, v_n$  in  $V$  with  $v_1 = u$  and  $v_n = v$  and  $\{v_i, v_{i+1}\} \in E$ .*

In this paper, we only consider connected graphs.

**Definition 3 (Graph Isomorphism).** *Graphs  $G(V, E, \lambda, \Sigma)$  and  $G'(V', E', \lambda', \Sigma')$  are called isomorphic if there exists a bijective function  $f : V \rightarrow V'$  such that:  $\forall v \in V : \lambda(v) = \lambda'(f(v)) \wedge E = \{\{f(v_1), f(v_2)\} | \{v_1, v_2\} \in E'\} \wedge \forall e \in E : \lambda(e) = \lambda'(f(e))$ .*

**Definition 4 (Subgraph).** Given a graph  $G(V, E, \lambda, \Sigma)$ ,  $G'(V', E', \lambda', \Sigma')$  is called a subgraph of  $G$  iff  $V' \subseteq V \wedge E' \subseteq E \wedge \forall v \in V' : \lambda(v') = \lambda(v) \wedge \forall e \in E' : \lambda(e') = \lambda(e)$ .

If a graph  $H$  has a subgraph  $H'$  such that a graph  $G$  is isomorphic with  $H'$ , then  $G$  is said to be *subgraph isomorphic* with  $H$ , denoted by  $H \succeq G$ ; function  $f$  defines how subgraph  $G$  can be *embedded* in the larger graph  $H$ . The set of all embeddings of  $G$  in  $H$  is denoted by  $F_{H \succeq G}$ .

**Definition 5 (Frequent Subgraph).** Given a collection  $\mathcal{D}$  of graphs and a graph  $G$ , the frequency of  $G$ , denoted by  $\text{freq}(G, \mathcal{D})$ , is the cardinality of the set  $\{G' \in \mathcal{D} | G' \succeq G\}$ . A graph  $G$  is frequent if  $\text{freq}(G, \mathcal{D}) \geq \text{minsup}$ , for a predefined threshold  $\text{minsup}$ .

The problem that is solved by a *frequent subgraph mining* algorithm is to find *all* subgraphs that are frequent in a dataset. In the molecular setting the set of frequent subgraphs is usually computed on the active set of molecules, and their corresponding frequencies on the inactive set of molecules are computed later.

Several algorithms have been proposed to solve the frequent subgraph mining problem, among which gSpan [10], GASTON [7] and MoFA [5, 8]. All these algorithms exploit the fact that the frequency constraint is anti-monotonic: if  $G \succeq H$ , then  $\text{freq}(G) \leq \text{freq}(H)$ . As a consequence of this property, algorithms do not need to consider all possible subgraphs of a database to find all frequent ones: it is possible to search from small to large subgraphs, and stop searching if a large graph is not frequent any more.

An important issue that subgraph miners have to address, is that there are many ways to construct isomorphic subgraphs. For instance, a molecular fragment consisting of an oxygen (O) connected to a carbon (C), can be obtained by connecting either a carbon to an oxygen, or by connecting an oxygen to a carbon. To avoid duplicates, only one of these two extensions should be allowed. Most subgraph miners solve this issue by using a *canonical code* that restricts in what ways a subgraph can be extended.

The *degree* of a node  $v$ , denoted by  $\text{deg}(v)$ , is the number of nodes to which the node is directly connected by the edges in the graph. Several simple classes of subgraphs have been studied. A (*free*) *tree* is a connected graph in which  $|V| = |E| + 1$ ; a *path* is a tree in which no node has a degree higher than 2.

Assume that there are two graphs  $H$  and  $G$ , such that  $H \succeq G$ , and  $\text{freq}(G) = \text{freq}(H)$ , then for the purpose of classification graph  $H$  can be considered redundant: it covers exactly the same set of molecules as  $G$ , but is more specific, and therefore more likely to overfit. Therefore, it is often useful to restrict the search to subgraphs  $G$  for which there is no subgraph  $H$  for which  $\text{freq}(G) = \text{freq}(H)$ . Such subgraphs are called *free subgraphs*. If there is no supergraph  $H$  for which  $\text{freq}(G) = \text{freq}(H)$ , then subgraph  $G$  is called a *closed subgraph*.

Finally, if we know the frequency of a subgraph in both a set of active and inactive molecules, we can compute its *correlation* with that activity; this score—typically the result of a  $\chi^2$  test—can be used to rank patterns. A *top-k* pattern miner finds all  $k$  subgraphs that obtain the highest positions in this

ranking. The motivation is that only the highest scoring subgraphs are expected to be important to classify molecules.

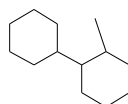
In [2], the classification of molecules was approached using a *top-k free* pattern mining algorithm. For several values of  $k$  the highest scoring patterns were determined; these patterns were used to build classifiers. Trees, graphs and paths were considered as pattern domains. It was concluded in many cases that classifiers based on paths achieved the highest predictive accuracy.

### 3 Interpretable subgraphs

A situation that occurs in many molecular data mining applications, is illustrated by the following example. Assume that we have a (hypothetical) database containing the following two highly active molecules:



Then we can derive the following substructure, which tries to encode a disjunction of these two graphs, but which is chemically hard to interpret, as it contains only part of an aromatic ring:



In this subgraph, some edges have been selectively included or excluded to obtain the best fit to the data. This subgraph shows that there is sometimes a tension between *interpretability* and predictive *accuracy* of subgraphs:

- if this subgraph obtains a superior accuracy score, one could believe that it is desirable that it is found;
- if this subgraph cannot be interpreted without looking through an enormous set of embeddings in the data, it may not be useful that it is found.

Thus, by *interpretability* we mean that less additional information has to be considered before the true value of the pattern can be determined by an expert.

A further problem is the size of the search space. If one allows individual removal or inclusion of edges, the search space becomes larger. As a result of the additional features, accuracy can increase, but on the other hand, if there are so many features to consider, it will be harder to interpret the features and to determine which of the features are really of interest.

These issues also important when building accurate classifiers. On the one hand, tweaking of substructures can be desirable, as it could yield a better fit to the data; on the one hand, if too much tweaking is allowed, we might *overfit* to the data. It is a well-known problem to balance these issues.

Until now, two solutions are prevalent. In the approaches based on gSpan or GASTON [10, 7] the interpretability of subgraphs is neglected, thus allowing

structural tweaking as in the example above. In MoFA [5, 8] the issue is addressed by hard-coding structural preference in the search: for example, rings of length 5 or 6 are marked as special structural elements, and edges of such rings can never be added individually, yielding more interpretable subgraphs and smaller search spaces, but disallowing potentially interesting subgraphs such as in the example above.

In this paper, we investigate an approach in which there is a parameter to balance interpretability and structural freedom. We wish to allow structural freedom, but want to implement a measure that allows us to express our preference for structures that are not too incomplete. There are many conceivable measures. Here, we take an approach that is based on considering how graphs match to the data. What we can observe in general for the example fragment, is that it is *incomplete*: in every embedding of this subgraph, we can connect an additional edge to the fragment. If we consider the original molecules as fragments, they are more ‘complete’: there would usually be less possibilities to connect additional edges to an embedding. Therefore, a reasonable measure seems to be based on the number of edges that can be connected to an embedding of the subgraph.

Formally, we can compute this as follows:

$$missing(f) = \sum_{v \in V_G} deg(f(v)) - deg(v),$$

which is the number of edges that can be connected to the embedded subgraph. For a database  $\mathcal{D}$  of graphs we can compute

$$missing(G) = \frac{\sum_{H \in \mathcal{D}, G \subseteq H} \min_{f \in \mathcal{F}_{H \supseteq G}} missing(f)}{|\{H \in \mathcal{D} | H \supseteq G\}|}.$$

We believe that it is reasonable to consider only the best possible embedding (for example, an average over all embeddings would be skewed as the number of embeddings can be exponential), therefore we choose to minimize the *missing* value. The measure clearly reflects the number of possibilities for extending the pattern in the minimal case.

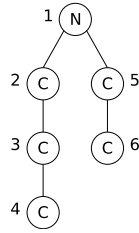
The use of this measure can be two-fold. We can use it to express a preference between structures that are more or less equivalent otherwise, thus in principle still allowing unlimited structural freedom. Another possibility is to enforce a constraint on this measure. In this case, we might not find some well-scoring substructures any more. On the other hand, structures satisfying such a constraint would have the advantage that the resulting substructures are usually easier to interpret, and the possibilities for overfitting are reduced.

It is important to observe that this constraint is neither monotonic nor anti-monotonic. A given subgraph with a low *missing* value can obtain a high *missing* value after extension: assume that the newly added node is always connected to many other nodes in the data, then *missing* will go up. On the other hand, the *missing* value can also go down. If we add an edge to the example fragment, we can obtain the second example molecule. This graph has less missing edges.

#### 4 Pushing the *missing* constraint

Although the  $missing(G) \leq maxmissing$  constraint is neither monotonic nor anti-monotonic, it is still possible to push the constraint in the mining process. The reason is that in subgraph mining, the canonical form limits how we are allowed to refine a subgraph. It is not possible to connect a new edge to every possible node in a subgraph to avoid duplicates. Therefore, for a certain embedding, we can determine that we will never get rid of some of the ‘missing’ edges as the search procedure does not allow for adding these missing edges. If the number of missing edges that we are not allowed to get rid off, is too large, we can stop refining the subgraph and prune the search space.

As an example, we use the DFS canonical code of gSpan; for more details about this canonical code, consult [10]. Consider the subgraph given below.



We assume that the label  $N$  is lower than the label  $C$ . Then in gSpan the canonical DFS code of this graph is (assuming all edges are labeled with  $\lambda$ ):

$$(1, 2, N, \lambda, C)(2, 3, C, \lambda, C)(3, 4, C, \lambda, C)(1, 5, N, \lambda, C)(5, 6, C, \lambda, C),$$

where  $(x, y, \sigma_1, \sigma_2, \sigma_3)$  denotes that there is an edge from node  $v_x$  to node  $v_y$  with label  $\sigma_2$ ; the label of  $v_x$  and  $v_y$  is  $\sigma_1$  and  $\sigma_3$ , respectively. According to gSpan’s canonical code, we can only connect new edges to the *rightmost* path. Thus, nodes 2, 3 and 4 cannot be extended any more. Furthermore, in the DFS code, among siblings, the lowest label must be listed first. This means that we cannot connect a node with label  $N$  to node 1 of the example graph.

For a given node  $v$  in a subgraph, and an embedding  $f$  of this subgraph, let  $sdeg_f(v)$  be the *static degree* of the node. We define the static degree to be the number of sibling edges of  $f(v)$  that can not be added to the subgraph, either because they are already in the subgraph, or because the canonical code does not allow it. Then for an embedding we can compute

$$smissing(f) = \sum_{v \in V_G} sdeg_f(v) - deg(v).$$

For a given graph  $H$  in the database and a pattern graph  $G$ , we can determine whether

$$\min_{f \in \mathcal{F}_{H \supseteq G}} smissing(f) \leq maxmissing.$$

If there is no database graph in which this condition holds, we do not need to refine the subgraph, as the average *missing* value can never become low enough to satisfy the constraint.

Name	Size class 1	Size class 2
NCI HIV	417	1069
Mutagenicity I [7]	2401	1936
Mutagenicity II [4]	341	343
Biodegradability [1]	143	185
PTE [10]	340	—

**Table 1.** Properties of the datasets used in the experiments

In principle, to decide which edges can be connected to a subgraph according to the canonical form is computationally hard, as we need to solve graph isomorphism. However, the above given two rules can be used to approximate  $sdeg_f(v)$ : nodes are static if either (1) they are a sibling of a node not on the rightmost path or (2) they are a siblings of a node on the rightmost path, but the label is too low. This approximation will not influence the correctness of the algorithm, as it underestimates the number of missing edges.

*Experimental Evaluation* To test the influence of this pruning rule we have performed several experiments with an implementation of gSpan that includes this constraint. The properties of the datasets are listed in Figure 1.

We first performed experiments on the predictive toxicology dataset (PTE). Runtime experiments were performed on an Intel Pentium M processor running at 1.1Ghz, and are listed in Table 2. In this table, the number of processed subgraphs is the number of subgraphs that is enumerated by the graph mining algorithm, including subgraphs for which *missing* is too high. We can observe

$min$ - $sup$	$max$ - $missing$	runtime	# processed subgraphs
32	$\infty$	5.9s	930
32	2.0	3.7s	706
16	$\infty$	46.9s	4405
16	2.0	9.7s	2197
10	$\infty$	261.8s	22758
10	4.0	108.9s	13472
10	3.0	57.0s	9449
10	2.0	19.9s	5344

**Table 2.** Experiments on the PTE dataset

$min$ - $sup$	$max$ - $missing$	runtime	# processed subgraphs
200	$\infty$	298.7s	2244
200	3.0	298.9s	2231
200	2.0	279.5s	2115
150	$\infty$	590.5s	6367
150	2.0	533.9s	5790

**Table 3.** Experiments on the Mutagenesis I dataset

that the runtime improvement is much larger than the number of processed subgraphs. The reason is that especially large subgraphs, for which computing subgraph isomorphism is more expensive, are pruned.

Next, we performed a set of experiments on the Mutagenesis I dataset. Experiments were performed on an Intel Pentium IV running at 3.2Ghz. The minimum support threshold applies to the set of active molecules. We observe that on this

dataset the influence of the constraint is almost negligible. A possible explanation is that in this dataset the molecules were encoded without hydrogens (contrary to the PTE dataset). Consequently, the branching factor of nodes in the data is lower. A larger amount of the nodes therefore already have low *missing* value. The pruning opportunities are therefore reduced.

Finally, we performed an experiment in which the NCI HIV dataset was used; we determined the frequent patterns in the active molecules, and evaluated the patterns also on the moderately active ones. The experiments were performed on an Intel Pentium IV 3.2Ghz again. For a minimum support of 6% we determined that the runtime without constraint was 1500s, while it was only 691s if a *maxmissing* constraint of 3.0 was applied. In this dataset the hydrogens were not included in the encoding. Apparently, the amount of speed-up is not only dependent on this encoding.

Dataset	<i>minsup</i>	Frequent Interpretable	
Mutagenesis I	2.5%	48583	132
Mutagenesis II	2.5%	7577	208
Mutagenesis II	5.0%	1005	103
HIV	6.0%	371374	350
HIV	7.5%	58380	264
Biodegradability	2.5%	168033	206
Biodegradability	5.0%	34839	88

**Table 4.** Experiments showing the reduction in subgraphs by applying the *maxmissing* constraint

Finally, in Table 4 the results are reported of an experiment in which we compared the number of frequent subgraphs in the output of gSpan with and without a *maxmissing* = 3.0 constraint. This shows that the *maxmissing* constraint is very effective in reducing the number of subgraphs.

## 5 Ranking patterns

As discussed in the introduction, the simplest approach to rank subgraphs is to order them according to the  $\chi^2$  statistic of their correlation with the activity of molecules. However, such an absolute ordering may not always be justified. Assume that we have two subgraphs of which the TID lists (i.e., the lists of graphs in the database to which they map) are *almost* equal, then it might not be justified to prefer the one pattern above the other pattern; the small difference might just be noise in the data. For every pattern, we can therefore define a set of patterns that is close to it in terms of TID list, and for which we have no reason to prefer one pattern above the other.

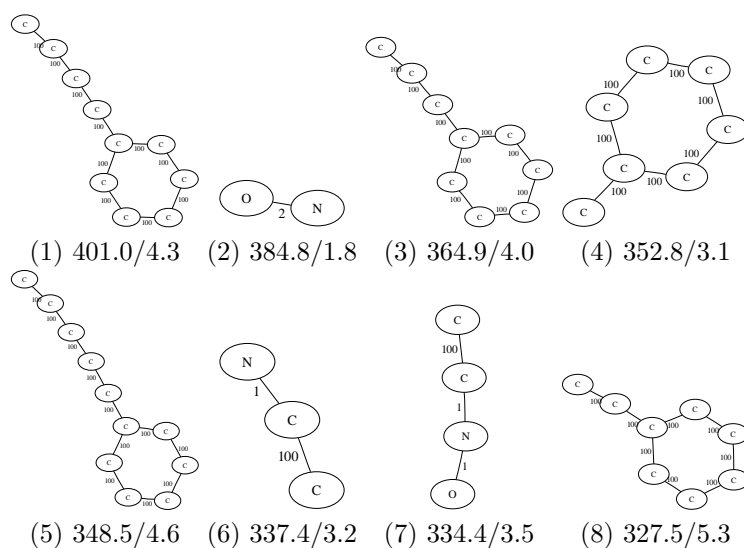
One way then to cut the ties between patterns with almost equal TID lists, is to use the *missing* score. Thus, assuming that we have sorted all patterns on  $\chi^2$ , we can apply this procedure:

1. Among the set of all patterns within a close distance from the topmost pattern, pick the pattern that scores best according to the *missing* score. Add this pattern to the result set.
2. Remove all patterns that are close to the chosen pattern (which includes the best pattern), including the chosen pattern.
3. Go to step 1 for the remaining sorted list of patterns.

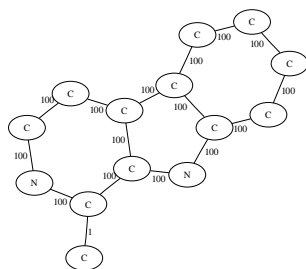
Observe that if two graphs have equal TID lists, and one is a subgraph of the other, the *missing* score can still cut ties. However, one cannot predict whether the subgraph or the supergraph will be preferred. Thus, we do not imply either closed subgraph mining or free subgraph mining.

Parameters in this procedure are the distance measure and the threshold on this measure. Currently, we have chosen symmetric difference as distance measure. As a rule of thumb, the threshold on the distance is lower than the minimum support threshold.

We first performed an experiment on the Mutagenesis I dataset. We used a minimum support of 150, a distance threshold of 30, and no interpretability constraint. The total number of frequent patterns was 6367. If we apply the selection procedure given above, 297 fragments remain. The best 8 fragments are given below. For each fragment its  $\chi^2$  value and its *missing* value are given.

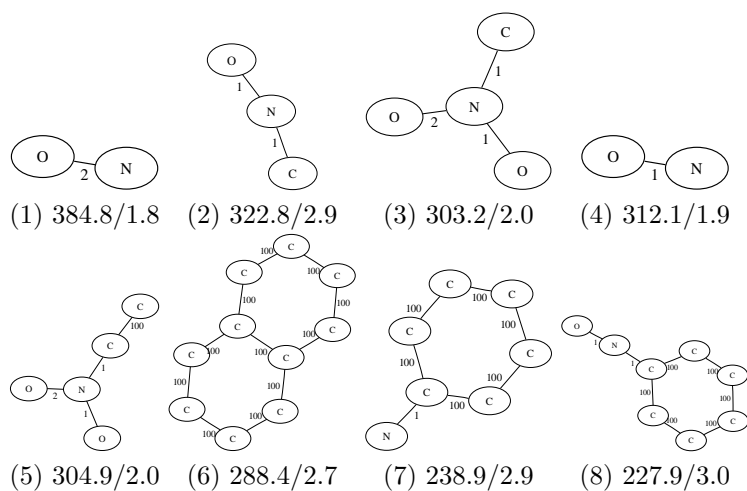


We can observe that there is a large number of fragments involving aromatic bonds, which are labeled with '100' in these figures. The first, third and fourth fragment still have significant differences in TID lists. Please note that the long chains in the patterns are *not* chains in the data. The method of [8] would not prune them. An example of a molecule that does not contain the first fragment, but does contain the third fragment, is given below:



The problem is that the information of how this pattern matches (i.e., if the non-branching chain in the pattern is also a non-branching chain in the data, as one could be tempted to think), is not readily available from the pattern and a closer inspection of patterns and data is required; there might also be other explanations. It is not obvious for what reasons this particular fragment matches as good as it does.

If however we apply a maximum value of 3.0 as constraint on the missing value, the number of frequent fragments that satisfies the constraint falls down to 54. With the algorithm given above, using a threshold distance of 30, 44 become selected. The best 8 fragments are given below:



To a certain extent, we still have similar effects as in the previous experiment. The fifth fragment only contains one aromatic bond, because the given group can connect to any aromatic ring, not only a six-ring, like in the seventh fragment. The effects are however much smaller.

Next, we performed several experiments on this dataset to compare the classification accuracies of several classes of fragments. To make the results comparable to the results obtained in [2], we took the same settings as in that paper, except that we used our new algorithm to select the cyclic subgraphs that are used as features; so, the results are obtained after 10 fold crossvalidation, using implementations of Weka [3]. We restricted ourselves to the C4.5 decision tree

Algorithm	Mutagenesis I		Mutagenesis II		Biodegradability	
	Number of features	Accuracy	Number of features	Accuracy	Number of features	Accuracy
Paths	100	76.37	100	70.90	100	76.22
	1000	79.73	1000	71.63	1000	74.10
Trees	100	70.94	100	70.39	100	71.96
	1000	74.83	1000	71.55	1000	76.07
Graphs	100	70.79	100	70.53	100	71.82
	1000	74.68	1000	72.06	1000	71.20
Selection 2.5%	82	79.94	93	76.02	83	72.56
Selection 5.0%	48	78.35	58	70.91	54	72.87

**Table 5.** Accuracies on the Mutagenesis I, Mutagenesis II and Biodegradability datasets; Selection  $x\%$  denotes that a minimum support of  $x\%$  was used

Algorithm	Number of features	Accuracy	Algorithm	Number of features	Accuracy
Paths	100	77.46	Selection 6.0% 4	107	82.77
	1000	83.21	Selection 6.0% 8	93	82.30
Trees	100	77.06	Selection 7.5% 4	94	82.77
	1000	75.95	Selection 7.5% 8	83	82.55
Graphs	100	77.06			
	1000	75.95			

**Table 6.** Accuracies on the NCI HIV dataset; Selection  $x\%$   $y$  denotes that a minimum support of  $x\%$  was used and a maximum distance of  $y$

learner, as this algorithm achieved the best results in all experiments in [2]. Results are listed in Table 5 and 6; results for Paths, Trees and Graphs are copied from [2].

In all cases *maxmissing* was fixed to 3.0. The distance threshold was 60 on Mutagenesis I, 2.5% on Mutagenesis II, and 2.5% on Biodegradability. Interestingly, we achieve similar classification accuracies as for the paths in most cases; only on the biodegradability dataset our results are disappointing. The set of features that is passed to C4.5 is smaller in all cases. Also in terms of the size of the decision tree, our classification method seems more interpretable; for example, a decision tree learned on the entire Mutagenesis I dataset using 1000 paths, contains 196 leaves, while the tree that is built on our features contains 104 leaves.

Comparing these tables to table 4, the largest reduction in the number of features stems from the *maxmissing* constraint.

In addition to the minimum support threshold, we also varied the distance threshold on the NCI HIV dataset. Most results do not seem to be very sensitive to the choice of the parameters, with the exception perhaps of the *minsup* threshold on the Mutagenesis II dataset.

## 6 Conclusions

We argued that there is a general trade-off between pattern representations, mining efficiencies and classification accuracies. To illustrate this, we developed a measure to quantify the interpretability of a subgraph, and showed that this measure could be pushed in the mining process, with mixed success. We performed experiments with the features that were selected using a feature selection method that takes this measure into account, and found that the classifiers were accurate in most cases, and possibly more interpretable.

There are many open issues in this approach. A broader study of interpretability measures might yield measures that approximate the average chemist's idea of interpretability better. For instance, one could also base an interpretability measure on how well examples cluster on the frequent supergraphs of a subgraph. We used an ad-hoc method to prefer structures that achieve a higher interpretability score, but a more principled approach would be desirable. Finally, in most experiments we applied feature selection only on pruned features. Our feature selection method did not work efficiently on large amounts of features. How to optimize this method is also an open question.

**Acknowledgements** This work was supported by the EU FET IST project IQ ("Inductive Querying"), contract number FP6-516169. We would like to thank Taneli Mielikäinen and Jeroen Kazius for discussions.

## References

1. H. Blockeel, S. Dzeroski, B. Kompare, S. Kramer, B. Pfahringer, and W. Van Laer. Experiments in predicting biodegradability. In *Appl. Art. Int.* 18, pages 157–181, 2004.
2. B. Bringmann, A. Zimmermann, L. De Raedt, and S. Nijssen. Don't be afraid of simpler patterns. In *PKDD*, 2006.
3. E. Frank, M. Hall, L.E. Trigg, G. Holmes, and I.H. Witten. Data mining in bioinformatics using weka. In *Bioinformatics* 20, pages 2479–2481, 2004.
4. C. Helma, T. Cramer, S. Kramer, and L. De Raedt. Data mining and machine learning techniques for the identification of mutagenicity inducing substructures and structure activity relationships of noncongeneric compounds. In *Journal of Chemical Information and Computer Systems* 44, pages 1402–1411, 2004.
5. H. Hofer, C. Borgelt, and M. Berthold. Large scale mining of molecular fragments with wildcards. In *IDA*, pages 380–389, 2003.
6. T. Horvath, T. Gärtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *KDD*, pages 158–167, 2004.
7. J. Kazius, S. Nijssen, J.N. Kok, T. Bäck, and A. IJzerman. Substructure mining using elaborate chemical representation. In *Journal of Chemical Information and Modeling* 46, 2006.
8. T. Meinel, C. Borgelt, and M. Berthold. Mining fragments with fuzzy chains in molecular databases. In *MGTIS*, pages 49–60, 2004.
9. N. Wale and G. Karypis. Acyclic subgraph-based descriptor spaces for chemical compound retrieval and classification. In *Technical report, Univ. Minnesota*, 2006.
10. X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *ICDM*, pages 721–724, 2002.