Mining Fragments with Fuzzy Chains in Molecular Databases

Thorsten Meinl[1], Christian Borgelt[2], and Michael R. Berthold[3]

[1] University of Erlangen-Nuremberg
Computer Science Department 2
Martensstr. 3, 91058 Erlangen, Germany
meinl@cs.fau.de

[2] School of Computer Science
Otto-von-Guericke-University of Magdeburg
Universitätsplatz 2, 39106 Magdeburg, Germany
borgelt@iws.cs.uni-magdeburg.de

[3] Department of Computer and Information Science
University of Konstanz
Universitätsstr. 10, 78464 Konstanz, Germany
berthold@inf.uni-konstanz.de

Abstract. This paper discusses methods to discover frequent, discriminative connected subgraphs (fragments) in a database of molecular structures. We present an extension to a well-known algorithm that allows for the discovery of fragments that contain chains of atoms of varying length. This is particularly important for real-world applications (for example drug discovery or synthetic success prediction) where the exact length of chains connecting two or more otherwise rigid substructures is not critical for the biological or chemical activity of the overall substructure. We demonstrate how the proposed extension successfully discovers fragments with several polymethylene bridges.

1 Introduction

1.1 Motivation

Finding common features in large sets of molecules is a frequently reoccurring problem in many biological or chemical applications. Examples include drug discovery, where the goal is to identify common properties that are shared by molecules and have been identified as “active” in a so-called High-Throughput Screen. Such screens typically produce activity information for hundreds of thousands of molecules. Other examples are compound synthesis, i.e. the generation of new molecules based on so-called virtual libraries. The ability to predict chances of a successful synthesis before it is attempted can save valuable resources. Again, results for the synthesis of hundreds of thousands of various molecules exist. In all these cases many possible modes of action exist: reasons why a specific molecule interacts with the sample or why a synthesis fails or succeeds, are manifold. This makes it extremely hard to identify the right features to use. In sharp contrast to many other data mining problems, this is not a simple problem of feature reduction but really a problem of finding suitable ways to describe molecules. In the past, biologists and chemists have spent much time developing just the right ways to describe molecules, ranging from simple one-dimensional measurements (such as molecular weight, number of rotatable bonds, etc.) over 2D and 3D models to enormously complex thousand-dimensional descriptors. It is obvious that none of these methods will be able to model all possible aspects of possible interactions between molecules. Sometimes a simple 3D shape is sufficient — although quite often this is only applicable for a part of a molecule, making matters complicated again since a possible similarity measure would need to weight different parts of the molecules differently. Sometimes the part of a molecule that matters can be described by the combination of a few bits of a fingerprint or by a small 2D fragment, literally a subgraph of the entire molecule. This approach is particularly interesting to the chemist because the resulting model can easily be interpreted.

1.2 Mining Molecular Fragments

For the extraction of frequent or discriminative fragments, various methods have been described recently. All of them are based on methods borrowed from the association rule mining community, in particular the Apriori algorithm [1] and the Eclat approach [2]. Whereas Apriori essentially implements a breadth-first search, Eclat follows a depth-first approach. The difference to the classical application of these algorithms — finding frequent occurrences of bits in large collections of high-dimensional bit-vectors — can be summarized nicely when looking at the two main steps of both methods, namely Candidate Generation and Support Computation.

- *Candidate Generation*: Generating new fragments is inherently based on the previous set of smaller fragments. In a bit-vector-based domain, such candidate generation is relatively straightforward. For graphs this becomes a more challenging task, since there are potentially many different candidates to consider and it is not at all trivial to avoid generation of duplicates.
- *Support Computation*: Again, this step is relatively easy for bit-vectors. For graphs, however, the support computation requires a test on subgraph isomorphism, which essentially requires embedding a fragment into each molecule in the database. Subgraph embedding is NP-complete [3], so this becomes prohibitively expensive, especially for larger fragments.

Quite a number of different approaches exist to date to find frequent fragments in molecular databases. Most of them only concentrate on a subset of the problems mentioned above. In particular most of them ignore the problem of support computation and rely on available graph-embedding toolkits, which restricts their applicability to only finding small fragments since graph embedding is computationally extremely expensive, especially if not optimized carefully.

Several algorithms to discover arbitrary connected subgraphs were proposed in [4–6]. The main contribution of these approaches are sophisticated ways to compute canonical labels for fragments. This allows them to quickly test for isomorphisms between graphs and to easily eliminate duplicate candidates. However, these procedures also rely on a full subgraph embedding test to compute fragment supports and hence are not applicable to larger fragments since subgraph tests scale exponentially with the size of the embedded fragment. More recent work, as described in [7] for example, reduces the number of required embeddings but still only pushes back the underlying problem.

A different approach was chosen in [8]. The presented algorithm (which will be explained in more detail in section 2) performs an Eclat-based search and stores all possible embeddings for a fragment in parallel. While this is rather expensive from a memory-consumption point of view, it allows for a very fast way of generating new candidates. Due to the consideration of all possible embeddings in parallel, this approach is not able to handle very small and frequent fragments such as, for example, a single carbon atom. The authors therefore propose to "seed" the algorithm with small, initial fragments such as nitrogen atoms or an aromatic ring. Recently, the authors discussed an extension of this method [9], where a special treatment of rings solves some of these problems, however starting the search with single carbon atoms still remains cumbersome.

Other algorithms that also use embedding lists have been presented in [10, 11]. Whereas the former – FFSM – still generates some candidate fragments that do not exist in the database, the latter – Gaston – seems to be very fast by using several refinement steps during the search process.

Other related approaches were discussed in (among others) [12–14]. Whereas the first only finds frequent trees and not arbitrary graphs, the others concentrate on finding geometrical or 3D substructures and variations. Since we will concentrate on finding complete 2D substructures in this article we will not go into detail about those algorithms.

Besides these graph-based algorithms there are also other approaches relying, for instance, on methods from Inductive Logic Programming (ILP), where molecules are essentially encoded as lists of basic facts and the result is a combination of facts (usually based on first order logic) that is compatible with both positive and negative examples [15–17].

1.3 Finding Fuzzy Fragments

Relying on full graph embeddings to compute support values is not only a problem for larger fragments. It also makes it hard to find "fuzzy" fragments, that is, is not possible to match certain small graph pieces with each other even if they are not perfectly identical. If one wanted to introduce such means of "fuzziness" into the subgraph embedding process it would become even more time consuming and would restrict the maximum size of identifiable fragments even further.

Such fuzzy matches are important for real world applications in order to find substructures that a chemist or biologist would consider equivalent. Examples are simple wild card atoms/bonds, rings that contain only carbon atoms or maybe 1-2 nitrogen atoms or chains of carbon atoms (polymethylene bridges) that connect two otherwise relatively rigid structures. The first two problems have already been addressed in [9]. There the authors have shown how the *MoFa* -algorithm first described in [8] can be extended to find fragments containing fuzzy meta-atoms.

The third issue – finding fragments with chains of variable lengths – is much more complicated than the first two. There, not only atom or bond types have to be mapped, worse, even the size of the equivalent fragments may differ. If one uses algorithms that conduct subgraph isomorphism tests, this problem is extremely difficult to handle. Nevertheless this is an important aspect of frequent substructure searches. All current algorithms only find connected subgraphs. However, the ability to tolerate variations in length of chains that connect two or more interesting substructures can help to find larger, more meaningful chemical fragments.

To our knowledge there is only one approach presented in [18] that would be able to solve the issue of fuzzy chains. It stems from the group of ILP-based algorithms. In this paper we present a further extension of the *MoFa* algorithm which is able to find all frequent fragments that differ only in the length of one or more carbon chains. As *MoFa* does not perform explicit subgraph isomorphism tests but keeps lists of embeddings, which are grown piece by piece, it is better suited for this type of problem than most other approaches.

2 Fragment Mining with *MoFa*

As stated above, the goal of molecular fragment mining is to find discriminative fragments in a database of molecules, that are classified as either active or inactive. To achieve this goal, the *MoFa* algorithm (Molecular Fragment Miner, [8]) represents molecules as attributed graphs and carries out a depth-first search on a tree of fragments. Going down one level in this search tree means extending a fragment by adding a bond and maybe an atom to it. For each fragment a list of embeddings into the available molecules is maintained, from which the lists of embeddings of its extensions can easily be constructed. An embedding is a copy of the particular subgraph where each atom and bond has a reference to the atom/bond in the molecule it is mapped to. As a consequence, expensive re-embeddings (i.e. subgraph isomorphism tests) of fragments are not necessary as $n - 1$ atoms/bonds of the new embedding are already mapped. The support of a fragment (the number of molecules it is contained in) is then determined by simply counting the number of different molecules the embeddings refer to. If the support of a fragment is high in the set of active molecules and low in the set of inactive molecules, it is reported as a discriminative fragment. The important ingredients of the algorithm are different search tree pruning methods, which are called *size-based pruning*, *support-based pruning*, and *structural pruning*. Among these the latter is the most important and most complicated. It is based on a schema that defines a local order of the extensions of a fragment, which is used to avoid redundant search. See [8] for details.

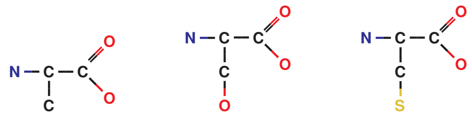


Fig. 1. The amino acids clycin, cystein, and serin

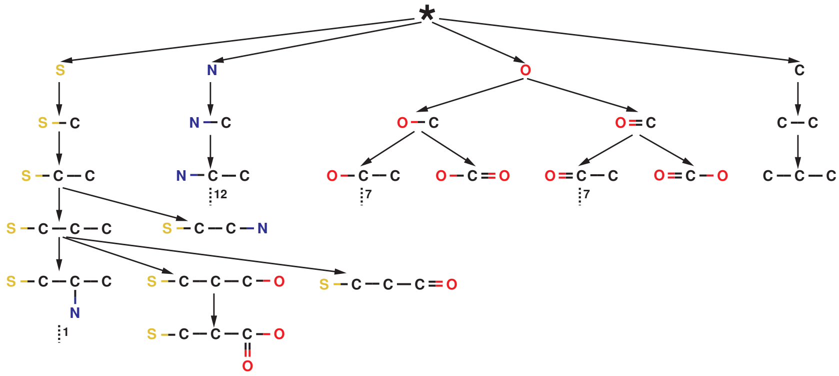


Fig. 2. The tree of fragments for the amino acids example

Let us illustrate the search tree for a simple example. Figure 1 shows the molecular database consisting of the amino acids clycin, cystein and serin with hydrogens and charges neglected. The upper part of the tree (or forest if the empty fragment at the root is removed) that is traversed by *MoFa* for these molecules is shown in figure 2. The first level contains individual atoms, the second connected pairs of atoms, and so on. The dots indicate subtrees that are not depicted, in order to keep the figure understandable. The numbers next to these dots state the number of fragments in these subtrees, to give an idea of the total size of the tree.

The order, in which the atoms on the first level of the tree are processed, is determined by their frequency of occurrence in the molecular dataset. The least frequent atom type is considered first. Therefore the algorithm starts on the left by embedding a sulfur atom into the example molecules. That is, the molecules are searched for sulfur atoms and their locations are recorded (which is not shown in the figure). In our example there is only one sulfur atom in cystein, which leads to one embedding of this (one atom) fragment. This fragment is then extended (depth-first search) by a single bond and a carbon atom (`-C`), which produces the fragment `S-C` on the next level. All other extensions of fragments that go down one level in the tree are created analogously.

If a fragment allows for more than one extension (as is the case, for instance, for the fragments `O-C` and `S-C-C-C`), we sort them according to the local ordering rules (see [8] for details). The main purpose of this local order is to prevent certain extensions to be generated, in order to avoid redundant searches. For instance, the fragment `S-C-C-C-O` is not extended by adding a single bond to a nitrogen atom at the second carbon atom, because this extension has already been considered in the subtree rooted at the left sibling of this fragment. Furthermore, in the subtree rooted at the nitrogen atom, extensions by a bond to a sulfur atom are left out, since all fragments containing a sulfur atom have already been considered in the tree rooted at the sulfur atom. Similarly, neither sulfur nor nitrogen are considered in the tree rooted at the oxygen atom, and the rightmost tree contains fragments that consist of carbon atoms only. *MoFa* does not create this (hypothetical) search tree completely as for real-world datasets it is too expensive. Therefore pruning is an essential part of the algorithm. Since a discriminative fragment must be frequent in the active molecules, and extending a fragment can only reduce its support in the active molecules (because only fewer molecules can contain it), subtrees can be pruned as soon as the support falls below a user-defined threshold (support-based pruning). Furthermore, the depth of the tree may be restricted, thus limiting the size of the fragments to be generated (size-based pruning).

Discriminative fragments should also be rare in the inactive molecules, defined formally by an user-specified upper support threshold. However, this threshold cannot be used to prune the search tree: Even if a fragment does not satisfy this threshold, its extension may (again extending a fragment can only reduce the support), and thus it has to be generated. Therefore this threshold is only used to filter the fragments that are frequent in the active molecules. Only those fragments that satisfy this threshold (in addition to the minimum support threshold in the active molecules) are reported as discriminative fragments.

3 Mining Fuzzy Chains

In this section we will discuss how the *MoFa* algorithm can be extended to find so-called *chains* of atoms. As already explained above, the discovery of such atom-chains of flexible length is valuable for many real-world applications in chemistry and biology.

3.1 Definitions

Before discussing details of the underlying algorithm we need to define more precisely what we mean when we refer to a *chain*.

1. Every atom in a chain has the same type (which is carbon, in almost all cases of interest). We call this the chain's atom type.
2. Every atom in the chain must have exactly two single bonds to other atoms (we neglect hydrogen atoms attached to chain atoms). Hence, there are two atoms in a chain (we call them head and tail), that have bonds to atoms (we call them neighbours) that are either not of the chain's atom type or have anything else than two single bonds, or both.
3. A chain always consists of the maximum possible number of atoms satisfying the first two conditions and must have a minimum length of one, i.e. consists of at least one chain-atom.

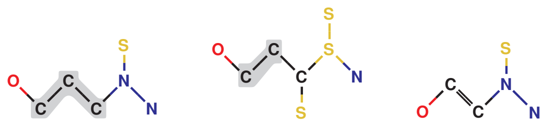


Fig. 3. According to the definition of *chain*, the first two fragments contain (carbon) chains, whereas the third one does not.

Figure 3 shows a few example chains of carbon type. The left fragment contains a chain of carbon atoms of length three. The set of of neighbour atom types of their chain is {`O, N`}. The second fragment contains a chain of length two. In this case the rightmost carbon atom is not a member of the chain because it has more than two bonds. The set of neighbour types is {`O, C`}. According to our definition the last fragment on the right does not contain a chain, because the double bond between the two carbon atoms violates condition 2.

Now we can define the concept of a *fuzzy chain*. Two chains are equivalent, if the following constraints hold:

1. both chains have the same atom type,
2. both chains have the same set of neighbour atoms types, and
3. the lengths of the two chains may differ but they need to be within the user defined range (for example two to four atoms).

Two fragments are considered equivalent if they are isomorph after equivalent chains are (conceptually) removed from them.

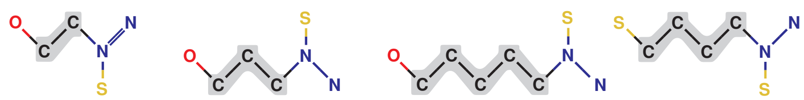


Fig. 4. Only the first two fragments are equivalent given a size range of 2 - 4.

With an acceptable size range of 2 - 4 the first two fragments in figure 4 are equivalent. The other two fragments are not equivalent since the third fragment's chain is too long and the fourth fragment's set of neighbour atom types is different ({`S, N`} instead of {`O, N`}).

With these definitions in mind, there is obviously a straightforward way to extend a subgraph mining algorithm by fuzzy matching: construct the whole set of equivalent framents that only differ in their chains' lengths and then test whether each of the fragments can be embedded into the molecules from the data set. Unfortunately, it is not quite that easy, since the standard subgraph isomorphism is not sufficient any more. Instead of general embedding, (artificially constructed) chain sections need to be embedded to chains of the molecules. An embedding to arbitrary molecular graphs will lead to wrong results.

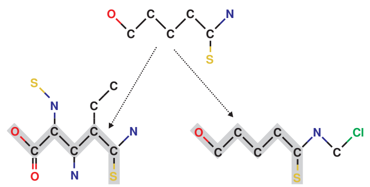


Fig. 5. Only the right molecule contains a chain.

To illustrate this problem consider a single fragment and the two molecules in figure 5. The standard subgraph isomorphism test returns a positive result for both embeddings. However, only in the right molecule the fragment's carbon chain is mapped to a chain. In the left molecule the four carbon atoms do not form a chain. Therefore the subgraph isomorphism test has to be extended to distinguish between normal atoms and chain atoms.

Fortunately this is not a great problem for *MoFa* as we will see in the next section. The only step where we must take this effect into account is when checking for duplicate fragments (graph isomorphism) because even though *MoFa* has clever pruning algorithms it may happen that a fragment is generated more than once. When we are dealing with chains one fragment may have a part representing a chain while the other one does not.

3.2 Algorithmic solution with *MoFa*

Before we describe how we can extend *MoFa* to consider fuzzy chains, we must first take a deeper look at the main loop of the original algorithm 3.1:

Algorithm 3.1 traverseTree(E_n^i)

$T \leftarrow$ all possible extensions of all embeddings in E_n^i
$G \leftarrow$ build groups (sets) of identical extensions in T
for all $G_j \in G$ **do**
  $E_{n+1}^j \leftarrow \emptyset$
  for all *extension* $\in G_j$ **do**
    $e_{new} \leftarrow$ new embedding generated from *extension*
    $E_{n+1}^j \leftarrow E_{n+1}^j \cup e_{new}$
  end for
end for
for all E_{n+1}^k **do**
  traverseTree(E_{n+1}^k)
end for

For the discussion, consider figure 2 again. Let the function `traverseTree` work on a node of level n of the search tree. On that level, fragments of size n are processed. In the third main branch of the example (the one starting with O), two fragments of size two are considered. For each fragment i, MoFa finds its embeddings in the molecules of the data set and stores these embeddings in a set E_n^i. E_2^1 contains four embeddings of `O-C` into the three amino acids, E_2^2 holds three embeddings of `O=C`.

In the first step all possible extensions (compatible with the pruning criteria) of all embeddings in E_n^i are searched and stored in T (line 1). Again looking at figure 2, after extending `O-C` the set T will contain three times the extension `-C` and three times `=O`. Now these extensions are separated into different groups (or sets) by comparing the new bond-atom pair (or bond only if a ring is closed) that are stored in G (line 2). In lines 3 to 9 new fragment embeddings are generated based on the extensions in the different groups. Every extension in group j creates a new embedding that is moved into E_{n+1}^j. In the example shown in figure 2 E_3^1 will contain the embeddings of `O-C-C` and E_3^2 the embeddings of `O-C=O`. Finally the process is repeated by recursively calling the function with the new embeddings as argument.

To handle fuzzy chains *MoFa* is extended as follows. Every extension that initiates a potential chain (note that we do not know in advance if this is really going to result in a chain) is duplicated. One copy is used in the "normal" branch, the other is put into a new fuzzy branch. The new atom and bond in this fuzzy extension are marked as chain atom and chain bond, respectively. This is important when later on duplicate fragments have to be filtered out as we already mentioned in the last section.

After new fragment embeddings have been created out of the fuzzy extension (lines 3 - 9) the next call to `traverseTree` is made (line 11). Now that we have embeddings with open chains the process differs from the one in the normal branch.

Firstly, we disallow any extension that takes place at any other place except for the marked chain atom (again line 1). Otherwise there would be the possibility that multiple open chains exist in a fragment which would require complicated bookkeeping. Secondly, after grouping the extensions we create new embeddings out of the remaining extensions. There two cases can happen:

1. Some extensions end the chain (i.e. the new atom is not carbon, has more than two bonds or not only single bonds), the others extend the chain further.
2. All extensions end the open chains.

Consider the first three molecules in figure 4 to be the molecular database. Let us assume that we have a search tree that started with the oxygen atom. After adding two carbon chain-atoms during the depth-first search, in the first structure the next extension would add a nitrogen atom with three bonds, whereas in the others the chain will be lengthened. Now the important step happens: As long as there is one extension that will elongate a chain, *all* embeddings will be extended by the same atom-bond pair (both are marked fuzzy). When applied to the example in figure 4, this means that the new embedding in the leftmost structure now contains a third carbon atom instead of the nitrogen atom. The reference of the additional atom in the embedding links to the same atom in the molecule as the one from its left neighbour. In figure 6 the rightmost carbon atom in the fragment at the top is marked fuzzy and points to the same atom in the molecule (indicated by the arrows) as its "real" neighbour. This process is necessary because *MoFa* relies on the fact that corresponding atoms in equiv-

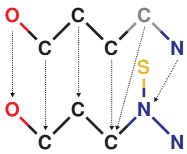


Fig. 6. A fuzzy fragment and its references to the atoms in the molecule.

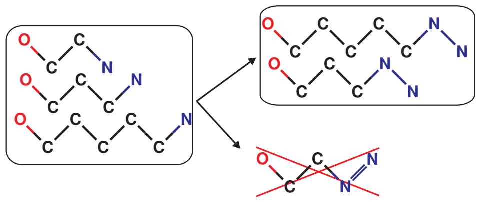
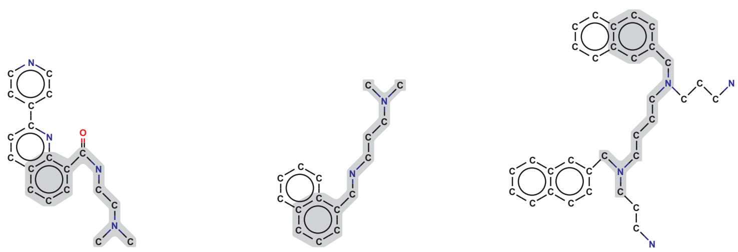


Fig. 7. Deletion of a former fuzzy fragment.

alent embeddings have the same index. This implies that all embeddings in a node of the search tree must have the same size.

This process of adding such "fuzzy" atom-bond pairs during the recursive search continues until the second case happens, i.e. all new extensions end the chain. In our example, that means that the nitrogen atom has been reached in all four structures. Now the embeddings are checked against the user-defined chain length criterion. All embeddings that contain chains *in the underlying molecule* where lengths are outside the range, are removed. This information can easily be retrieved by not counting all marked fuzzy atoms in an embedding.

Lastly the remaining extensions are grouped as in the normal branch (line 2 in algorithm 3.1) and will form new – now again "unfuzzy" – branches in the search tree. However, there is one last issue one has to be aware of. Either already at this point or sometimes later in the search process, it may happen that a group of new embeddings will only contain elements with just one chain length. This is illustrated in figure 7. The group on the left consists of three equivalent (but not identical) embeddings, which occur in the first three structures of figure 4. Now the embeddings in this group are extended but there is no common extension in all three structures. In the leftmost the new nitrogen atom has a double bond whereas in the others it has a single bond. So the new group at the bottom of figure 7 is not a fuzzy group any more and is therefore removed. It will be discovered in one of the standard nonfuzzy branches.



molecule #617931 molecule #658480 molecule #685982

Fig. 8. Three molecules of the NCI-H23 dataset with a fragment group whose members only differ in the length of the carbon chain

4 Experimental Results

To prove the applicability of our algorithm to large datasets we searched for fragments with fuzzy chains in two well-known datasets, namely NCI-HIV[4] and the H23 subset of NCI-Cancer[5] with 35,982 compounds. However, only the H23 dataset contains molecules that are equivalent except for chain lengths. One example is shown in figure 8. The three molecules contain fragments (indicated by the gray shadows) that only differ in the length of the carbon chain. In total this fragment group occurs in 140 molecules, the greater part being active.

One reason why no larger substructure is found, lies in the aromatic ring attached to the ring in the fragment. In the left fragment it contains one nitrogen atom instead of only carbon atoms like in the other two molecules. If we had also applied fuzzy atom matching (as described in [9]) we would have probably found the bigger fragment as well.[6]

5 Conclusions

The presented extension of the *MoFa* algorithm allows for the discovery of fragments with chains of (carbon) atoms of varying length. This is another step towards the goal of finding not only frequent fragments but discriminative fragments that help chemists better understand the behaviour of certain molecules thus making *MoFa* even more useful for practical applications.

Acknowledgements

We thank Michael Philippsen for his work and advice on improving readability and presentation of this article.

References

1. Agrawal, R., Imielinski, T., Swami, A.N.: Mining association rules between sets of items in large databases. In Buneman, P., Jajodia, S., eds.: Proceedings of the 1993 ACM SIGMOD Intl. Conf. on Management of Data, Washington, D.C., USA, ACM Press (1993) 207–216
2. Zaki, M.J., Parthasarathy, S., Ogihara, M., Li, W.: New algorithms for fast discovery of association rules. In Heckerman, D., Mannila, H., Pregibon, D., Uthurusamy, R., Park, M., eds.: 3rd Intl. Conf. on Knowledge Discovery and Data Mining, AAAI Press (1997) 283–296
3. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Company (1979)

[4] The dataset can be found at http://dtp.nci.nih.gov/docs/aids/aids_data.html ...
[5] ... and this one at http://dtp.nci.nih.gov/docs/cancer/cancer_data.html .
[6] Unfortunately the two implementations currently differ quite a lot thus causing some severe problems when they are used together.

4. Kuramochi, M., Karypis, G.: Frequent subgraph discovery. In: Proceedings of the IEEE Intl. Conf. on Data Mining ICDM, Piscataway, NJ, USA, IEEE Press (2001) 313–320
5. Yan, X., Han, J.: gSpan: Graph-based substructure pattern mining. In: Proceedings of the IEEE Intl. Conf. on Data Mining ICDM, Piscataway, NJ, USA, IEEE Press (2002) 51–58
6. Inokuchi, A., Kashima, H.: Mining significant pairs of patterns from graph structures with class labels. In: Proceedings of the Third IEEE International Conference on Data Mining, IEEE Computer Society (2003) 83
7. Deshpande, M., Kuramochi, M., Karypis, G.: Automated approaches for classifying structures. In: Proceedings of the 2nd ACM SIGKDD Workshop on Data Mining in Bioinformatics (BIOKDD '02), ACM Press (2002) 11–18
8. Borgelt, C., Berthold, M.R.: Mining molecular fragments: Finding relevant substructures of molecules. In: Proceedings of the IEEE Intl. Conf. on Data Mining ICDM, Piscataway, NJ, USA, IEEE Press (2002) 51–58
9. Hofer, H., Borgelt, C., Berthold, M.R.: Large scale mining of molecular fragments with wildcards. In: Advances in Intelligent Data Analysis V. Number 2810 in Lecture Notes in Computer Science (LNCS). Springer Verlag (2003) 380–389
10. Huan, J., Wang, W., Prins, J.: Efficient mining of frequent subgraphs in the presence of isomorphism. In: Proceedings of the 3rd IEEE Intl. Conf. on Data Mining ICDM, Piscataway, NJ, USA, IEEE Press (2003) 549–552
11. Nijssen, S., Kok, J.N.: A quickstart in frequent structure mining can make a difference. Technical report, LIACS, Leiden University, Leiden, The Netherlands (2004)
12. Kramer, S., Raedt, L.D., Helma, C.: Molecular feature mining in HIV data. In: Proceedings of the seventh ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, ACM Press (2001) 136–143
13. Kuramochi, M., Karypis, G.: Discovering frequent geometric subgraphs. In: Proceedings of the IEEE Intl. Conf. on Data Mining ICDM, Piscataway, NJ, USA, IEEE Press (2002) 258–265
14. Parthasarathy, S., M.Coatney: Efficient discovery of common substructures in macromolecules. In: Proceedings of the IEEE Intl. Conf. on Data Mining ICDM, Piscataway, NJ, USA, IEEE Press (2002) 362–369
15. Finn, P.W., Muggleton, S., Page, D., Srinivasan, A.: Pharmacophore discovery using the inductive logic programming system PROGOL. Machine Learning **30** (1998) 241–270
16. King, R., Muggleton, S., Srinivasan, A., Sternberg, M.: Structure-activity relationships derived by machine learning: the use of atoms and their bond connectivities to predict mutagenicity by inductive logic programming. Proceedings of the National Academy of Sciences **93** (1996) 438–442
17. Marchand-Geneste, N., Watson, K., Alsberg, B., King, R.: A new approach to pharmacophore mapping and qsar analysis using inductive logic programming. application to thermolysin inhibitors and glycogen phosphorylase b inhibitors. Journal of Medicinal Chemistry **45** (2002) 399–409
18. Lee, S.D., Raedt, L.D.: Constraint based mining of first-order sequences in SeqLog, University of Alberta, Edmonton, Canada (2002) 80–96