MoSS: A Program for Molecular Substructure Mining

Christian Borgelt
Dept. of Knowledge Processing and Language Engineering
University of Magdeburg
Universitätsplatz 2,
39106 Magdeburg, Germany
borgelt@iws.cs.uni-magdeburg.de

Thorsten Meinl
Computer Science Department 2
University of Erlangen-Nuremberg
Martenstraße 3,
91058 Erlangen, Germany
meinl@cs.fau.de

Michael Berthold
Dept. of Computer and Information Science
University of Konstanz
78457 Konstanz, Germany
berthold@inf.uni-konstanz.de

ABSTRACT

Molecular substructure mining is currently an intensively studied research area. In this paper we present an implementation of an algorithm for finding frequent substructures in a set of molecules, which may also be used to find substructures that discriminate well between a focus and a complement group. In addition to the basic algorithm, we discuss advanced pruning techniques, demonstrating their effectiveness with experiments on two publicly available molecular data sets, and briefly mention some other extensions.

1. INTRODUCTION

A frequent task in biochemistry is the search for common features in large sets of molecules. Examples are drug discovery, where one is interested in identifying properties shared by molecules that have been classified as "active" (and rarely shared by those classified as "inactive") w.r.t., for example, the protection of human cells against a virus, and compound synthesis, where one tries to identify properties that enable or inhibit the synthesis of new molecules, so that one can predict the chances for a successful synthesis.

Since the features one may use to describe molecules are manifold, there are approaches in abundance, ranging from simple one-dimensional measurements to complex thousand-dimensional descriptors. The molecular weight and the number of hydrogen donors or acceptors are examples of simple features. More complex ones include binary vectors, which can be several thousand bits long with each bit representing a specific constellation of atoms like aromatic rings or amino groups, as well as shape descriptors that try to capture geometric properties of a molecule.

In this paper we focus on an approach that models molecules as attributed graphs, thus taking the connection structure, though not the 3-dimensional structure into account. The resulting set of graphs is then searched for common subgraphs, that is, molecular fragments that appear with a user-specified minimum frequency. For this approach several algorithms have been proposed recently, with many of them based on methods developed for association rule mining. In particular, the Apriori algorithm [1] and the Eclat algorithm [21] are taken as starting points. The general ideas of these algorithms can be transferred to molecular substructure mining, even though the fact that the input consists of graphs instead of sets poses some problems. Examples of algorithms developed in this way are Subdue [5], MolFea [12], FSG [13], MoFa [2], gSpan [20], FFSM [9], and Gaston [16]. Other approaches rely, for instance, on principles from inductive logic programming [6] and describe the graph structure by logical expressions.

Of course, all of the mentioned approaches are generally applicable to attributed graphs, with molecules being only one specific application example. However, the Java implementation discussed in this paper is geared to molecules as it supports reading molecules in standard description languages like SMILES (Daylight, Inc.) and SLN (Tripos, Inc.). It also has to be mentioned that the first version of the presented program was developed in cooperation with the biochemical software company Tripos, Inc., St. Louis, MO, USA. Finally, it should be noted that the algorithm described here is also known as MoFa (**Mo**lecular **Fra**gment miner). The Java program presented here, however, is called MoSS (**Mo**lecular **S**ub**S**tructure miner) in order to distinguish it from other implementations.

2. BASIC ALGORITHM

In order to capture the bond structure of molecules, we model them as attributed graphs, in which each vertex represents an atom and each edge a bond between atoms. Each vertex carries attributes that indicate the atom type (i.e., the chemical element), a possible charge, and whether it is part of an aromatic ring. Each edge carries an attribute that indicates the bond type (single, double, triple, or aromatic).

Our goal is to find substructures that have a certain minimum support in a given set of molecules, i.e., are part of at least a certain percentage of the molecules. However, in order to restrict the search space, we usually consider only

OSDM'05, August 21, 2005, Chicago, Illinois, USA.

connected substructures, i.e., graphs having only one connected component. For most applications, this restriction is harmless, because connected substructures are most often exactly what is desired. (Note, however, that the program considered here also allows for starting from a disconnected core structure, though not for extending a substructure by an unconnected atom.) We do *not* constrain the connectivity of the graph in any other way: The graphs may be chains or trees or may contain an arbitrary number of cycles.

2.1 Search Tree Traversal

Most naturally, the search is carried out by traversing a tree of fragments of molecules, similar to the tree of item sets that is traversed in frequent item set mining. The root of the tree is a core structure to start from, which for now we assume to be a single atom (more complex cores are discussed below). Going down one level in the search tree means to extend a substructure by a bond (and maybe an atom, if the bond does not close a ring), just like going down in an item set tree means adding an item to an item set. That is, with a single atom at the root of the tree, the root level contains the substructures with no bonds, the second level those with one bond, the third level those with two bonds and so on.

There are basically two ways in which such a search tree can be traversed: We can use either a breadth first search and explicit subset tests (like Apriori) or a depth first search and intersections of transaction lists (like Eclat). For our task the Eclat approach seems preferable, because the Apriori approach has certain drawbacks: even subset tests can be costly, but substructure tests, which consist mathematically in checking whether a given attributed graph is a subgraph of another attributed graph, can be extremely costly. Furthermore, the number of small substructures (1 to 4 atoms) can be enormous, so that even storing only the topmost levels of the tree can require a prohibitively large amount of memory. Of course, the Eclat approach also has its drawbacks, for example, the transaction lists are now lists of embeddings of a substructure into the given molecules. Since there can be several embeddings of the same substructure into one molecule, these lists tend to get fairly long. This drawback can make it necessary to start from a reasonably sized core structure (see below).

To be more specific, our algorithm searches as follows: The given core structure is embedded into all molecules, resulting in a list of embeddings. Each embedding consists of references into a molecule that point out the atoms and bonds that form the substructure. Remember that a list of embeddings may contain several embeddings for the same molecule if the molecule contains the substructure in more than one place or if the substructure is symmetric. In a second step each embedding is extended in every possible way. This is done by adding all bonds in the corresponding molecule that start from an atom already in the embedding (to ensure connectedness and, of course, to reduce the number of bonds that have to be considered). This may or may not involve adding the atom the bond leads to, because this atom may or may not be part of the embedding already. More technically, by following the references of an embedding the atoms and bonds of the corresponding molecule are marked and only unmarked bonds emanating from marked atoms are considered as possible extensions.

The resulting extended embeddings are then sorted into equivalence classes, each of which represents a new substructure. This sorting is very simple, because only the added bond and maybe the added atom have to be compared. In our implementation we use a sorted array of lists of embeddings to group the extensions. After all extended embeddings have been processed, each array element contains the list of embeddings of a new substructure. Each of these new substructures corresponds to a child node in the search tree, each of which is then processed in turn by searching recursively on the list of embeddings corresponding to it.

2.2 Basic Search Tree Pruning

Of course, subtrees of the search tree can be pruned if they refer to substructures not having enough support, i.e., if too few molecules are referred to in the associated list of embeddings. We call this *support based pruning*. We may also prune the search tree if a user-defined threshold for the number of atoms in a fragment has been reached. We call this *size based pruning*. However, the most important pruning type is *structural pruning*, which is meant to ensure that each substructure is considered only once in the search tree. In frequent item set mining such structural pruning can easily be achieved by drawing on an (arbitrary, but fixed) order of the items and disallowing extensions by items preceding the item added last to the set (cf. [3] for more details).

In molecular substructure mining, however, such an approach is impossible, since there is no possible *global* order of, say, the atoms, as we have to take the bond structure into account. However, we can number the atoms *in a substructure* and record how a substructure was constructed in order to constrain its extensions. The number we assign to an atom reflects the step in which it was added. That is, the core atom is numbered 0, the atom added with the first bond is numbered 1 and so on. Note that this number does not tell anything about the type of the atom, as two completely different atoms may receive the same number, simply because they were added in the same step.

Whenever an embedding is extended, we record in the resulting extension the number of the atom from which the added bond started. When the extended embedding is to be extended itself, we consider only bonds that start from atoms having numbers no less than this recorded number. That is, only the atom extended in the preceding step and atoms added later than this atom can be the starting point of a new bond. This rule is directly analogous to the rule that only items following the item added last may be added to an item set. With this simple scheme we immediately avoid that two bonds, call them A and B, which start from different atoms, are added in the order A, B in one branch of the search tree and in the order B, A in another. Since either the atom A starts from or the atom B starts from must have a smaller number, one of the orders is ruled out.

However, two or more bonds can start from the same atom. Therefore we also have to define an order on bonds, so that we do not add two different bonds A and B that start from the same atom in the order A, B in one branch of the search tree and in the order B, A in another. This order on bonds is, of course, arbitrary. In our implementation, single bonds precede aromatic bonds, which precede double bonds, which

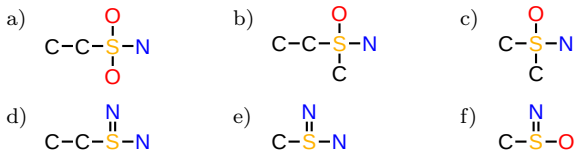

Figure 1: A set of six example molecules.

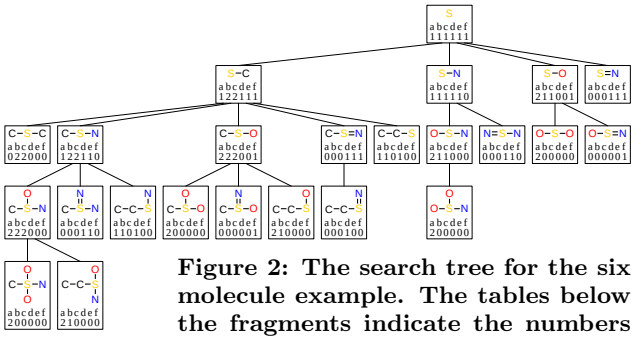

Figure 2: The search tree for the six molecule example. The tables below the fragments indicate the numbers of embeddings per molecule.

precede triple bonds. Finally, within extensions by bonds of the same type starting from the same atom, the order is determined by (1) whether the atom the bond leads to is already in the substructure or not and (2) the type of this atom. To take care of the bond type etc., we record in each embedding which bond was added last.

As a final rule, we disallow extensions by bonds leading to an atom already in the substructure if the number of the destination atom is higher than the number of the source atom. All bonds leading to already captured atoms (that is, all bonds closing rings) must lead "backward", that is, from an atom with a higher number to an atom with a lower one.

The above rules provide us with a structural pruning scheme, but unfortunately this scheme is not perfect and making it perfect would be very expensive computationally. The problem is that we do not have any precedence rule for two bonds of the same type starting from an atom with the same number and leading to atoms of the same type, and that it is not possible to give any precedence rule for this case that is based exclusively on locally available information. We consider the problems that result from this imperfection and our solution below, but think it advisable to precede this consideration by an illustrative example of the search process as we defined it up to now.

2.3 An Illustrative Example

As an illustration we consider how our algorithm finds the frequent substructures of the six example molecules shown in Figure 1[1], starting from a sulfur atom. We use a minimum support of 50%, i.e., a substructure must occur in at least three of the six molecules to qualify as frequent.

[1]Please note that these structures were made up to demonstrate certain aspects of the search scheme. None of them has any real (i.e., chemical) meaning.

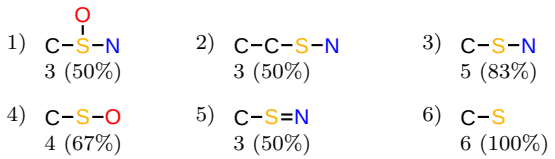

Figure 3: The six frequent substructures that are found in the order in which they are generated.

First the sulfur atom is embedded into the six molecules. This results in six embeddings, one for each molecule, which form the root of the search tree (see Figure 2; the table in the root node records that there is one embedding for each molecule). Then the embeddings are extended in all possible ways, which leads to the four different substructures shown on the second level (i.e., S–C, S–N, S–O, S=N). These substructures are ordered, from left to right, as they are considered by our algorithm, i.e., extensions by single bonds precede extensions by double bonds, and within extensions by bonds of the same type the element type of the atom a bond leads to determines the order. Note that there are two embeddings of S–C into both the molecules b and c and two embeddings of S–O into the molecule a.

In the third step the extensions of the substructure S–C are constructed. This leads to the first five substructures on the third level (i.e., C–S–C, C–S–N, C–S–O, C–S=N and C–C–S). Again the order of these substructures, from left to right, reflects the order in which they are considered. Since we search depth first, the next substructure to be extended is C–S–C.[2] However, this substructure does not have enough support and therefore the subtree is pruned.

The substructure C–S–N is considered next etc. However, we confine ourselves to pointing out situations in which specific aspects of our method become obvious. Effects of structural pruning can be seen, for instance, at the fragment C–S–N, which does not have a child in which a second carbon atom is attached to the sulfur atom. The reason is that the extension by the bond to the nitrogen atom rules out all single bonds leading to atoms of a type preceding nitrogen (like carbon). Similarly, C–S=N does not have children with another atom attached to the sulfur atom by a single bond, not even an oxygen atom, which follows nitrogen in the periodic table of elements. The reason is that a double bond succeeds a single bond and thus the extension by the double bond to the nitrogen atom rules out all single bonds emanating from the sulfur atom. Finally, the structure C–C–S has no children at all, even though it has enough support. The reason is that in this substructure a bond was added to the carbon atom adjacent to the sulfur atom. This carbon atom is numbered 1 and thus no bonds can be added to the sulfur atom, which has number 0. Only the carbon atoms can be starting points of a new bond, but there are no such bonds in the molecules a, b, and d. Note that ruling out extensions of the sulfur atom in this branch is not harmful, since all fragments having another atom attached to the sulfur atom are considered in the subtrees to the left of C–C–S.

[2]It may seem strange that there are two embeddings of this substructure into both the molecules b and c. The reason for this is explained below.

During the recursive search all frequent substructures encountered are recorded. The resulting set of six frequent substructures, together with their absolute and relative support is shown in Figure 3. Note that C–C–S is not reported, because it has the same support as its superstructure C–C–S–N. Likewise, O–S–N, S=N, and S are not reported. Transferring a common term from frequent item set mining, we may say that our algorithm reports only *closed fragments*, that is, fragments no superstructure of which has the same support (see also below, Section 3.1).

The example makes it clear that our algorithm can find arbitrary substructures, even though it does not show how cyclic structures are treated. Unfortunately, search trees for cyclic structures are too big to be depicted here.

2.4 Incomplete Structural Pruning

We indicated above that our structural pruning is not perfect. In order to understand the problems that can arise, consider two molecules A and B with the common substructure N–C–S–C–O. We try to find this substructure starting from the sulfur atom. Since the two bonds emanating from the sulfur atom are equivalent, we have no precedence rule and thus the order in which they are added to an embedding depends on the order in which they occur in the corresponding molecule (which we have no real control over) .

Suppose that in molecule A the bond going to the left precedes the bond going to the right, while in molecule B it is the other way round. As a consequence, in embeddings into molecule A the left carbon atom will precede the right one, while in embeddings into molecule B it will be the other way round. Now consider the substructure C–S–C and its extensions. In molecule A the carbon numbered 1 (the left one) will be extended by adding the nitrogen atom and thus the oxygen atom can be added in the next step (to the carbon on the right, which is numbered 2), resulting in the full substructure. However, in molecule B the nitrogen atom has to be added by extending the carbon atom numbered 2 (again the left one; in embeddings into molecule B the right carbon is numbered 1). Hence it is not possible to add the oxygen atom in the next step, because this would mean adding a bond starting at an atom with a lower number than the atom extended in the preceding step. Therefore the common substructure is not found. This example also shows that it does not help to look "one step ahead" to the next atom, because there could be arbitrarily long equivalent chains, which differ only at the ends. There are no "local" criteria, which would enable us to decide how to order and thus number equivalent extensions of the same atom.

If, however, we accept to reach identical substructures in different branches of the search tree in cases like this, we can correct the imperfection of our structural pruning. Whenever we extended an embedding by following a bond, we allow adding an equivalent bond in the next step, regardless of whether it precedes or succeeds, in the corresponding molecule, the bond added in the preceding step. This relaxation explains why there are two embeddings of the substructure C-S-C into both the molecules b and c of our example. In one embedding the left carbon atom is numbered 1 and the one at the bottom is numbered 2, while in the other it is the other way round (cf. Figure 1).

Note that considering the same substructure several times cannot lead to wrong results, only to multiple reporting of the same substructure. Multiple reporting, however, can be suppressed by maintaining a list of frequent substructures and suppressing new ones that are identical to already known ones. It is more important that the missing rule for equivalent bonds can lead to considerable redundant search in certain structures, especially molecules containing one or more aromatic rings. (A solution to this is discussed below.)

However, it should be noted that even if we could amend the weakness of our structural pruning, we would still be unable to guarantee that each substructure is considered only once. If, for instance, some substructure X can be embedded twice into some molecules and if there are frequent substructures that contain both embeddings (and thus X twice), then these substructures can be grown from either embedding. If the connection between the two embeddings of X is not symmetric, the same substructure is reached in two different branches of the search tree in this case. Obviously, there is no simple way to avoid such situations.

2.5 Embedding a Core Structure

Up to now we assumed that we start the search from a single atom. This usually works fairly well as long as this atom is rare in the molecules to work on. For example, sulfur or phosphorus are often good starting points in biochemical applications, while starting with carbon is a bad idea: Every organic molecule contains several carbon atoms, often twenty or more, and thus we end up with an already very high number of embeddings of the initial atom. As a consequence, the algorithm is likely to run out of memory before reaching substructures of reasonable size.

However, if we cannot start from a rare element, it is sometimes possible to specify a core—for instance, an aromatic ring with one or two side chains—from which the search can be started. Provided the core structure is specific enough, there are only few, at best only one embedding per molecule, so that the list of embeddings is short.

While it is trivial to embed a single atom into a molecule, embedding a core structure can be much more difficult. In our implementation we rely on a simple observation: embedding a core structure is the same as finding a common substructure of the molecule and the core that is as big as the core itself. This leads to the idea to grow a substructure into both the core and the molecule until it completely covers the core. That is, we do a substructure search for the core and the molecule starting from an arbitrary atom of the core and requiring a support of 100% (i.e., both the core and the molecule must contain the substructure). In addition, we can restrict the search to one embedding of a substructure into the core at all times, since we know that it must be completely covered in the end. (For the molecule, however, we must consider all possible embeddings.)

Note that the same mechanism of growing a substructure into two molecules can also be used for substructure tests as they are needed to suppress multiple reporting of the same fragment (see above) as well as reporting redundant fragments (fragments that are substructures of some other fragment and have the same support as this fragment).

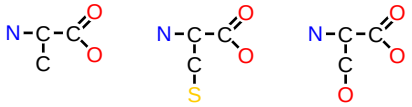

Figure 4: The amino acids clycin, cystein and serin.

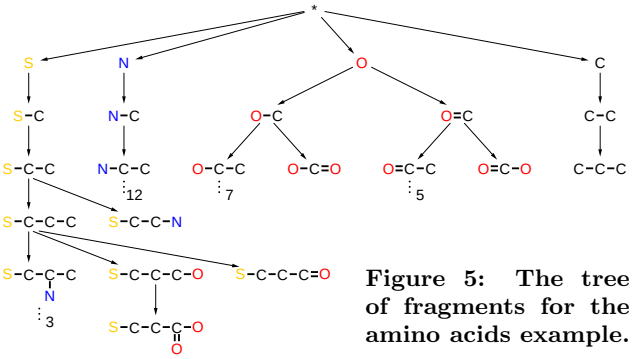

Figure 5: The tree of fragments for the amino acids example.

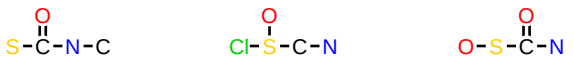

Figure 6: Some (fictitious) example molecules.

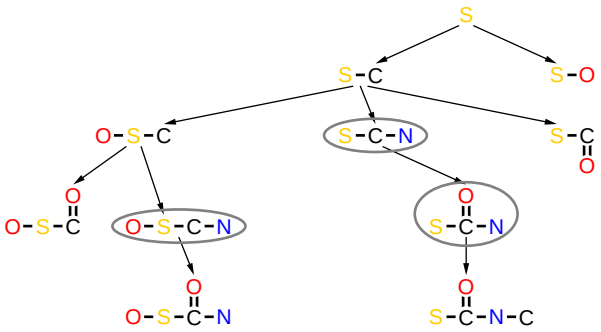

Figure 7: Search tree for the molecules in Figure 6.

2.6 Starting with an Empty Core

Up to now we assumed that we are given a core structure to start the search from. However, in [7, 8] an extension to empty cores was suggested. We explain this approach with a simple example. Figure 4 shows the amino acids clycin, cystein and serin. The upper part of the tree (or forest if the empty fragment at the root is removed) that is traversed by our algorithm for these molecules is shown in Figure 5. The first level contains individual atoms, the second connected pairs 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, indicating 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 molecules. 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. 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 in the way described above. Next the subtree rooted at the nitrogen atom is processed. However, in this subtree extensions by a bond to a sulfur atom are ruled 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. The program offers the option to skip this last tree, because for biochemical molecules it can be very large and thus may govern the search time.

3. ADVANCED PRUNING

In the preceding section we considered basic search tree pruning, which is meant to ensure that each substructure is considered as few times as possible. In this section we discuss further optimizations, suggested in [14, 4], which consist in two advanced pruning strategies, namely *equivalent sibling pruning* and *perfect extension pruning*. The first of these optimizations can be applied even if the search is not restricted to closed fragments (see below), while the second presupposes that non-closed fragments can be discarded.

3.1 Closed Molecular Fragments

As already mentioned above, the notion of a *closed fragment* is derived from the corresponding notion of a *closed item set*, which is defined as an item set no superset of which has the same support, i.e., is contained in the same number of transactions. Analogously, a *closed fragment* is a fragment no superstructure of which has the same support, i.e., is contained in the same number of molecules. As an example consider the three example molecules shown in Figure 6 and the corresponding (unpruned) MoFa search tree (starting from sulfur as a seed) shown in Figure 7: the closed fragments (for a minimum support of two molecules, i.e., 66%) in inner nodes are circled. (It should be clear that every leaf of the search tree is necessarily a closed fragment).

As for item sets, restricting the search for molecular fragments to closed fragments does not lose any information: all frequent fragments can be constructed from the closed ones by simply forming all substructures of closed fragments that are not closed fragments themselves and assigning to them as their support the maximum of the support values of those closed fragments of which they are substructures. Consequently, closed fragment mining is a very convenient way to reduce the size of the output. In addition, chemists are usually not interested in all frequent fragments, but only in the closed ones, presumably because they contain all relevant information without redundancy. Here, however, we are interested in the fact that in closed fragment mining certain pruning techniques become applicable, which can speed up the search considerably. These and other advantages of closed fragment mining were also studied in [20, 14, 4].

3.2 Equivalent Sibling Pruning

In order to suppress all redundant search, one would have to check whether any two subtrees of the search tree have the

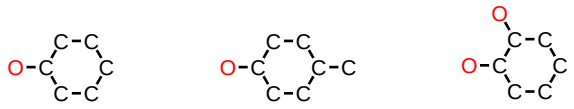

Figure 8: Three phenols: phenol, p-cresol, and catechol.

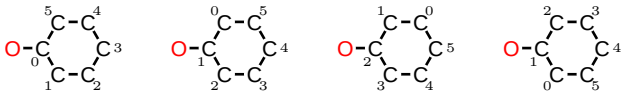

Figure 9: Equivalent extended embeddings.

same fragment as their root (or fragments which only differ in the restrictions placed on their possible extensions), so that only one (namely the least restricted one) is actually searched. Such a general check, however, is costly, because one would have to store all fragments that have been considered in the search so far. In addition, one needs an efficient way of checking whether these stored fragments contain one that is equivalent to the one currently considered. Finally, it is difficult to find out whether a subtree to be searched is the least restricted one appearing in the whole search tree, even though the depth first traversal applied in our algorithm always considers the least restricted sibling node first.

However, what can be checked fairly easily is whether *two sibling nodes* in the search tree correspond to fragments that are equivalent (represent the same substructure) and differ only in the restrictions placed on their possible extensions. That such a situation can actually occur can be seen from the example molecules shown in Figure 8. Suppose we start the search by embedding a benzene ring into these molecules. Since such a ring exhibits a high symmetry (which is a necessary prerequisite for equivalent siblings), it can be embedded into each of the molecules in twelve different ways (the ring can be rotated to six positions and it can be traversed in two directions). Consider now the extensions of this benzene ring: each of the twelve embeddings is extended by a bond and an oxygen atom. This leads to twelve new fragments, all of which are equivalent and differ only in the label of the ring atom that was extended (see Figure 9 for examples). Obviously it suffices in this situation to consider the fragment in which the ring atom with the smallest number was extended. All other fragments must lead to redundant search, because they allow for a subset of the possible extensions, but no additional ones.

Since our algorithm considers sibling nodes w.r.t. a local order that is based on the extension information of the parent fragment (i.e. bond and atom type, number of the atom the bond is incident to) and processes the least restricted extensions first, it is fairly easy to implement the described pruning approach: for each node its preceding sibling nodes are checked for equivalence and if an equivalent sibling is found, the current node is skipped.

The equivalence test is carried out as follows: for the fragment corresponding to the current node an arbitrary embedding into an arbitrary molecule is selected. In the corresponding molecule all bonds and atoms belonging to this embedding are marked and all other bonds and atoms are unmarked. Then all embeddings of the fragment corresponding to a sibling node, which is to be checked for equivalence, are traversed. For each of these embeddings it is checked whether it refers only to marked atoms and bonds, that is, whether it essentially represents the same substructure. If such an embedding can be found, the two fragments are equivalent and consequently the current node (the extensions of which are more severely restricted than those of its already processed sibling) can be skipped.

Note that it suffices to check one molecule, because if there are identical embeddings into one molecule there must be identical embeddings into all molecules referred to by the fragment. As a consequence the check is comparatively cheap and thus does not degrade performance much even if the pruning cannot be carried out.

3.3 Perfect Extension Pruning

Perfect extension pruning is based on the observation that sometimes there is a fairly large common fragment in *all* currently considered molecules. As long as the search has not grown a fragment to this maximal common one, it is not necessary to branch in the search tree.

From the definition of a closed fragment it is clear that if there is a common substructure in *all* currently considered molecules of which the current fragment is only a part, then any extension that does not grow the current fragment towards the maximal common one can be postponed until this maximal common fragment has been reached. The reason is, obviously, that the maximal common fragment is part of all closed fragments that can be found in the currently considered set of molecules. Consequently, it suffices to follow one path in the search tree that leads to this maximal common fragment and to start branching only from there.

As an example consider again the set of molecules shown in Figure 6. If the search is seeded with a single sulfur atom, considering extensions by a single bond starting at the sulfur atom and leading to an oxygen atom can be postponed until the structure `S-C-N` common to all molecules has been grown (provided that the extensions of this maximal common fragment are not restricted in any way—see below).

Technically, the search tree pruning is based on the notion of a *perfect extension*. An extension of a fragment, consisting of a bond and possibly an atom (if the bond does not close a ring), is called *perfect* if all of its embeddings can be extended in exactly the same way by this bond and atom and if it is a *bridge*[3] in all embeddings into the molecules or if it closes a ring. (Note that there may be multiple ways of extending an embedding by this bond and atom. In this case all embeddings must be extendable in the same number of ways and in all the bond must be a bridge or must close a ring. The additional condition that the bond has to be a bridge or must close a ring in all embeddings was unfortunately not recognized in [4].) If there is a perfect ex-

[3] A bridge in a graph is an edge that, if removed, splits the graph into two unconnected parts.

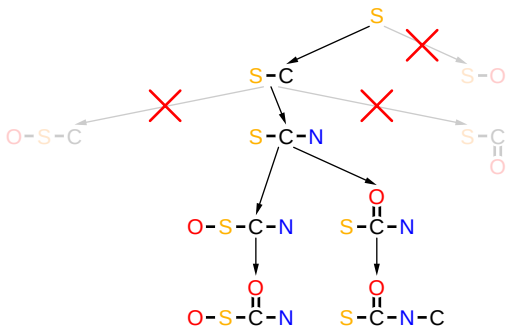

Figure 10: Search with perfect extension pruning.

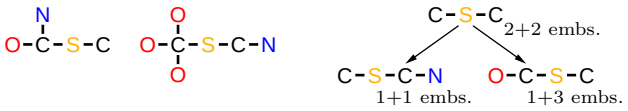

Figure 11: Examples of non-perfect extensions.

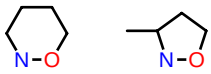

Figure 12: Perfect extensions may fail in rings.

tension of a fragment, all closed fragments can, in principle, be found by searching only the corresponding branch.

There is, however, a minor complication, because perfect extension pruning interferes with the normal structural pruning done in MoFa. Normal structural pruning prevents extensions of a fragment by bonds that start from atoms with smaller numbers than the one extended in the preceding step. However, a perfect extension should not lead to such a restriction, because otherwise some search results are lost.

As an example consider again the search tree shown in Figure 7. If we simply confined the search to the subtree rooted at the fragment `S-C-N`, we would lose the fragment `O-S-C-N` in the left branch. The reason is that the extension of `S-C` to `S-C-N`, due to the normal structural pruning rules, prevents an extension of the sulfur atom in this subtree, because an atom with a higher number, namely the carbon atom, has been extended in the preceding step.

However, a perfect extension should not restrict possible future extensions of the fragment in this way. Therefore the extension information of a fragment obtained by a perfect extension are set to those of its parent fragment, bypassing the normal structural pruning rules. In the considered example, this allows us to extend the fragment `S-C-N` by bonds starting from the sulfur atom and thus we get the search tree shown in Figure 10, in which the fragment `O-S-C-N` is found.

When it comes to implementing perfect extension pruning, one should bear in mind that checking whether an extension is perfect by testing the extensions of all embeddings is costly. Therefore we precede the actual test by cheap tests in order to, if possible, *fail early* or *succeed early*. The ideas underlying these tests are fairly simple: an extension leading to a fragment cannot be perfect (1) if the number of molecules referred to by the fragment differs from those referred to by its parent and (2) if the number of embeddings of the fragment is not an integer multiple of the number of embeddings of its parent. On the other hand, if these tests do not indicate that the extension is not perfect, we can immediately conclude that it is perfect if the fragment refers only to one molecule. The costly test whether each parent embedding leads to the same number of extended embeddings is, of course, carried out only if the number of molecules referred is not equal the number of embeddings (otherwise there is exactly one embedding per molecule). Only afterwards we test whether the bond either closes a ring or is a bridge in all embeddings. (Bridges are, of course, found and marked in a preprocessing step, so that the test only has to check this flag for the bonds referred to by the embeddings.)

Note that it is actually necessary to count the embeddings per molecule. Checking only whether the total number of embeddings into all molecules coincides with (or is an integer multiple of) the parent embeddings does not suffice as can be seen from the example shown in Figure 11. Even though the total number of embeddings is the same in the right branch, the extension is not perfect. The left extension is not perfect, because the number of extended embeddings, even though the same for each parent embedding, is reduced from the number of extensions of its parents. This indicates that some symmetry has been destroyed by the extension, which therefore cannot be perfect.

Why it is necessary to constrain perfect extensions to bridges if the extension does not close a ring can be seen from the example in Figure 12. Suppose we want to find all common substructures of these two molecules (minimum support 100%). We seed the algorithm with a nitrogen atom. Then adding the oxygen atom is a perfect extension in the sense that it can be done in the same way in all molecules. The same holds for a subsequent extension of the oxygen atom with a carbon atom, or generally, to walk around the rings in counterclockwise order. This yields the fragment `N-O-C-C-C-C` as the only result. However, `C-C-N-O-C-C` and `C-C-C-N-O` are also common substructures. These we would lose if we allowed such extensions in rings to be considered as perfect.

4. OTHER EXTENSIONS

Apart from the advanced pruning strategies described above, there are several ways in which the basic algorithm can be extended, some of which are already incorporated in the Java implementation referred to here. Due to reasons of space these extensions are only briefly mentioned here. Details can be found in [2, 7, 8, 14, 15].

4.1 Finding Discriminative Fragments

Our approach to find frequent substructures can easily be extended to find *discriminative fragments*, that is, substructures that are frequent in a predefined subset of the molecules and infrequent in the complement of this subset. Finding discriminative fragments requires two parameters: a minimum support for the focus subset and a maximum support for the complement. The search is carried out in exactly the same way as described above. The only difference is that two

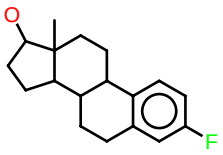

Figure 13: Example of a steroid, which can lead to problems due to the large number of rings.

Figure 14: Aromatic and Kekulé representations of benzene.

support numbers are determined: one for the focus subset and one for the complement. Only the support in the focus subset is used to prune the search tree. The support in the complement determines whether a frequent substructure is recorded or not, thus filtering out substructures that do not satisfy the requirements for a discriminative structure.

4.2 Rings

An unpleasant problem of the search algorithm as we presented it up to now is that the local ordering rules for suppressing redundant searches are not complete. Two extensions by identical bonds leading to identical atoms cannot be ordered based on locally available information alone (see above). As a consequence, a certain amount of redundant search has to be accepted in this case, because otherwise incorrect support values would be computed and it could not be guaranteed that all frequent substructures are found.

Unfortunately, situations leading to such redundant search occur almost always in molecules with rings, whenever the fragment extension process enters a ring. Consequently, molecules with a lot of rings, like the steroid shown in Figure 13—despite all clever search tree pruning—still lead to a prohibitively high search complexity.

To cope with this problem, we added an (optional) preprocessing step to the algorithm, in which the rings of the molecules are found and marked. The implementation described here offers the possibility to specify a range for the ring size, so that this feature can be restricted to chemically meaningful rings, for example, of size 5 and 6. In the search process itself, whenever an extension adds a bond to a fragment that is part of a ring, not only this bond, but the whole ring is added (all bonds and atoms of the ring are added in one step). As a consequence, the depth of the search tree is reduced (the basic algorithm needs five or six levels to construct a ring bond by bond) and many of the redundant searches of the basic algorithm are avoided, which leads to enormous speed-ups. Details can be found in [7, 8].

Besides a considerable reduction of the running time, treating rings as units when extending fragments has other advantages as well. In the first place, it makes a special treatment of atoms and bonds in rings possible, which is especially important for aromatic rings. For example, benzene can be represented in different ways as shown in Figure 14. On the left, all six bonds are represented as aromatic bonds, while the other two representations—called

pruning	# nodes	# frags.	# embs.
neither	48762	48762	78209
equivalent sibling	22423	23122	38475
perfect extension	4355	6581	16731
both	1577	2947	7786

Table 1: Effects of pruning on the steroids data set.

Kekulé structures—use alternating single and double bonds. A chemist, however, considers all three representations as equivalent. With the basic algorithm this is not possible, because it would require treating single, double, and aromatic bonds alike, which obviously leads to undesired side effects. By treating rings as units, however, these special types of rings can be identified in the preprocessing step and then be transformed into a standard aromatic ring.

4.3 Variable Length Chains

Often the exact length of a carbon chain in a molecule is not important, as long as it is within a certain range. In this case, the fragment may contain only an indication of a chain, which match a certain number of chained carbon atoms in a molecule, where the number may differ for different molecules. This type of imprecise match can be handled in a similar way as the ring extension we studied in the previous section: Instead of bond by bond, a carbon chain is added in one step, allowing for different lengths. The implementation described here offers this possibility, but does not yet take a range for the acceptable lengths of the chain.

4.4 Wildcard Atoms

In principle it is possible to extend the search algorithm to handle wildcard atoms, which may match any atom in a user-defined group of atoms. Details can be found in [7, 8]. This possibility is available in a sibling implementation of the algorithm described here. This implementation is, however, property of Tripos, Inc., and thus cannot be made available. The Java implementation described here currently lacks this capability.

5. EXPERIMENTAL RESULTS

In our experiments we restrict ourselves to demonstrating the effects of the two advanced pruning approaches described above. We carried out experiments on three data sets, all of which are publicly available. The first is a very small data set consisting of 17 steroid molecules.[4] The effects of the two pruning methods are shown in Table 1 (all experiments were carried out with a minimum support of one molecule). As can be seen, both pruning methods have considerable effects, with those of perfect extension pruning being much more pronounced. However, this is presumable due to the very low minimum support, which makes perfect extension pruning highly effective. At higher support values equivalent sibling pruning seems to be more effective (see below).

In a second test we ran the program on the well-known HIV data set available from the National Cancer Institute [10]. This library contains about 44,000 molecules tested for their activity against the HI-virus, which are grouped into three

[4]See Section 7 below for a download URL.

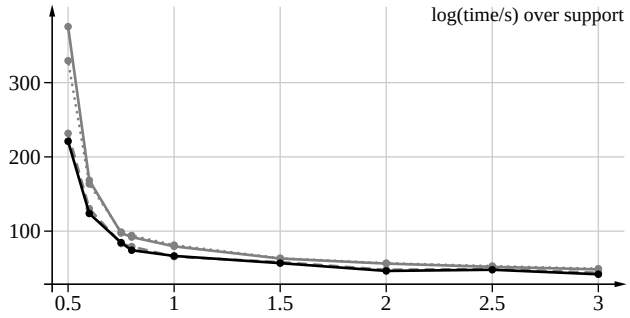

Figure 15: Execution times on NCI's HIV data.

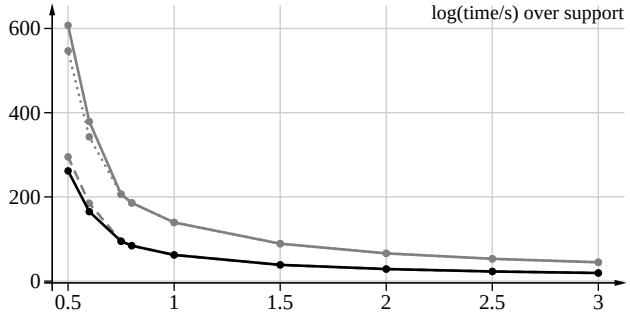

Figure 16: Numbers of nodes on NCI's HIV data.

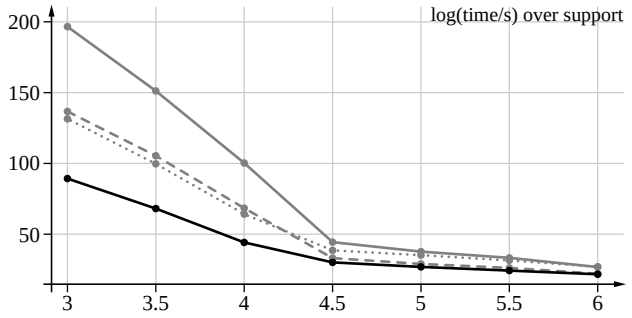

Figure 17: Execution times on the IC93 data.

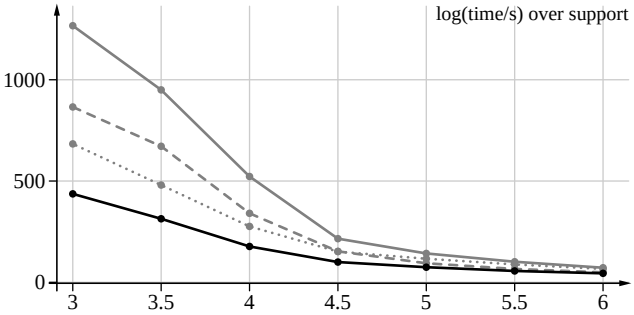

Figure 18: Numbers of nodes on the IC93 data.

classes: about 400 belong to class CA (confirmed active), about 1000 to CM (confirmed medium active) and the rest belongs to CI (confirmed inactive). In the experiments we report here, however, we neglected these class assignments and tried to find common substructures of all molecules at minimum support thresholds ranging from 0.7 to 3%. In addition, we tested on the IC93 data set [11]. For all experiments we used a Pentium 4C 2.6GHz system with 1GB of main memory running S.u.S.E. Linux 9.3 and Java 1.5.0.

Results on the NIC's HIV data are shown in Figures 15 and 16. These results were obtained with activated ring mining for rings of size 5 and 6. Results on the IC93 data are shown in Figures 17 and 18 and were obtained without ring mining. The former figure of each pair (i.e., 15 and 17) shows the execution time in seconds, the latter (i.e., 16 and 18) the number of search tree nodes (in thousands) over the minimum support in percent. Solid grey lines refer to the basic algorithm with neither of the two advanced pruning strategies. The dashed grey line refers to the results for equivalent sibling pruning only, the dotted grey line to the results with perfect extension pruning only. Finally, the black line shows the results obtained with both pruning methods activated.

As can be seen from Figures 15 and 16, equivalent sibling pruning yields the highest gains for the HIV data set, while perfect extension pruning yields only fairly small gains. On the IC93 data the picture is reversed: perfect extension pruning is more effective. However, both methods yield considerable gains, which add nicely when both pruning strategies are activated. It is interesting to note that the effects of perfect extension pruning are more pronounced for the number of nodes (see Figure 18), which indicates that part of the gains resulting from this pruning type are eaten up by the somewhat costly test for a perfect extension.

6. CONCLUSIONS

In this paper we presented a Java implementation of an efficient graph substructure mining algorithm that finds closed frequent fragments of molecules. The algorithm is based on a depth first search in a tree of substructures, in which lists of embeddings into the set of molecules are processed and extended. The core ingredients of the approach, which make it efficient, are a structural pruning scheme, which tries to minimize the number of times a fragment is considered in the search, and two advanced pruning techniques, which speed up the search considerably. In addition, the implementation offers other extensions, like ring and chain mining, which make it very useful for biochemical applications.

7. PROGRAM

The implementation of the molecular substructure mining algorithm described in this paper (executable Java archive as well as the source code, distributed under the LGPL) can be downloaded free of charge at

http://fuzzy.cs.uni-magdeburg.de/~borgelt/software.html

Explanations of how to apply the program can be found at

http://www.inf.uni-konstanz.de/bioml/projects/mofa/

It takes a simple set of 17 steroids (also used above) as an example, which can also be retrieved from the above URLs.

8. REFERENCES

[1] R. Agrawal, T. Imielienski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. *Proc. Conf. on Management of Data*, 207–216. ACM Press, New York, NY, USA 1993

[2] C. Borgelt and M.R. Berthold. Mining Molecular Fragments: Finding Relevant Substructures of Molecules. *Proc. IEEE Int. Conf. on Data Mining (ICDM 2002, Maebashi, Japan)*, 51–58. IEEE Press, Piscataway, NJ, USA 2002

[3] C. Borgelt. Efficient Implementations of Apriori and Eclat. *Proc. 1st IEEE ICDM Workshop on Frequent Item Set Mining Implementations (FIMI 2003, Melbourne, FL)*. CEUR Workshop Proceedings 90, Aachen, Germany 2003. http://www.ceur-ws.org/Vol-90/

[4] C. Borgelt, T. Meinl, and M.R. Berthold. Advanced Pruning Strategies to Speed Up Mining Closed Molecular Fragments. *Proc. IEEE Conf. on Systems, Man and Cybernetics (SMC 2004, The Hague, Netherlands)*, CD-ROM. IEEE Press, Piscataway, NJ, USA 2004

[5] D.J. Cook and L.B. Holder. Graph-Based Data Mining. IEEE Trans. on Intelligent Systems 15(2):32–41. IEEE Press, Piscataway, NJ, USA 2000

[6] P.W. Finn, S. Muggleton, D. Page, and A. Srinivasan. Pharmacore Discovery Using the Inductive Logic Programming System PROGOL. *Machine Learning*, 30(2-3):241–270. Kluwer, Amsterdam, Netherlands 1998

[7] H. Hofer, C. Borgelt, and M.R. Berthold. Large Scale Mining of Molecular Fragments with Wildcards. *Proc. 5th Int. Symposium on Intelligent Data Analysis (IDA 2003, Berlin, Germany)*, LNCS 2810:380–389. Springer-Verlag, Heidelberg, Germany 2003

[8] H. Hofer, C. Borgelt, and M.R. Berthold. Large Scale Mining of Molecular Fragments with Wildcards. *Intelligent Data Analysis* 8:495–504. IOS Press, Amsterdam, Netherlands 2004

[9] J. Huan, W. Wang, and J. Prins. Efficient Mining of Frequent Subgraphs in the Presence of Isomorphism. *Proc. 3rd IEEE Int. Conf. on Data Mining (ICDM 2003, Melbourne, FL)*, 549–552. IEEE Press, Piscataway, NJ, USA 2003

[10] *HIV Antiviral Screen*. National Cancer Institute. http://dtp.nci.nih.gov/docs/aids/aids_data.html

[11] *Index Chemicus — Subset from 1993*. Institute of Scientific Information, Inc. (ISI). Thomson Scientific, Philadelphia, PA, USA 1993

[12] S. Kramer, L. de Raedt, and C. Helma. Molecular Feature Mining in HIV Data. *Proc. 7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD 2001, San Francisco, CA)*, 136–143. ACM Press, New York, NY, USA 2001

[13] M. Kuramochi and G. Karypis. Frequent Subgraph Discovery. *Proc. 1st IEEE Int. Conf. on Data Mining (ICDM 2001, San Jose, CA)*, 313–320. IEEE Press, Piscataway, NJ, USA 2001

[14] T. Meinl, C. Borgelt, and M.R. Berthold. Discriminative Closed Fragment Mining and Perfect Extensions in MoFa. *Proc. 2nd Starting AI Researchers' Symposium (STAIRS 2004, Valencia, Spain)*, 3–14. IOS Press, Amsterdam, Netherlands 2004

[15] T. Meinl, C. Borgelt, and M.R. Berthold. Mining Fragments with Fuzzy Chains in Molecular Databases. *Proc. 2nd Int. Workshop on Mining Graphs, Trees and Sequences (MGTS 2004, Pisa, Italy)*, 49–60. University of Pisa, Pisa, Italy 2004

[16] S. Nijssen and J.N. Kok. A Quickstart in Frequent Structure Mining can Make a Difference. *Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD2004, Seattle, WA)*, 647–652. ACM Press, New York, NY, USA 2004

[17] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering Frequent Closed Itemsets for Association Rules. *Proc. 7th Int. Conf. on Database Theory (ICDT'99, Jerusalem, Israel)*, LNCS 1540:398–416. Springer-Verlag, Heidelberg, Germany 1999

[18] T. Washio and H. Motoda. State of the Art of Graph-Based Data Mining. *SIGKDD Explorations Newsletter* 5(1):59–68. ACM Press, New York, NY, USA 2003

[19] X. Yan and J. Han. gSpan: Graph-Based Substructure Pattern Mining. *Proc. 2nd IEEE Int. Conf. on Data Mining (ICDM 2003, Maebashi, Japan)*, 721–724. IEEE Press, Piscataway, NJ, USA 2002

[20] X. Yan and J. Han. Closegraph: Mining Closed Frequent Graph Patterns. *Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD 2003, Washington, DC)*, 286–295. ACM Press, New York, NY, USA 2003

[21] M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New Algorithms for Fast Discovery of Association Rules. *Proc. 3rd Int. Conf. on Knowledge Discovery and Data Mining (KDD'97, Newport Beach, CA)*, 283–296. AAAI Press, Menlo Park, CA, USA 1997