Mining Molecular Fragments: Finding Relevant Substructures of Molecules

Christian Borgelt
School of Computer Science
University of Magdeburg, Universitätsplatz 2
D-39106 Magdeburg, Germany
e-mail: borgelt@iws.cs.uni-magdeburg.de

Michael R. Berthold
Data Analysis Research Lab
Tripos, Inc., 601 Gateway Blvd, Suite 720
South San Francisco, CA 94080, USA
e-mail: berthold@tripos.com

Abstract

We present an algorithm to find fragments in a set of molecules that help to discriminate between different classes of, for instance, activity in a drug discovery context. Instead of carrying out a brute-force search, our method generates fragments by embedding them in all appropriate molecules in parallel and prunes the search tree based on a local order of the atoms and bonds, which results in substantially faster search by eliminating the need for frequent, computationally expensive reembeddings and by suppressing redundant search. We prove the usefulness of our algorithm by demonstrating the discovery of activity-related groups of chemical compounds in the well-known National Cancer Institute's HIV-screening dataset.

1. Introduction

Many data mining tasks in bioinformatics consist in analyzing large collections of molecules with the goal to find some regularity among molecules of a specific class. Possible applications are manifold. One example is drug discovery, where the biologist wants to find new drug candidates based on experimental evidence of activity against a certain disease gathered by screening hundreds of thousands of molecules. A second, more recent emphasis comes from chemical synthesis success prediction, where the goal is to find molecular features that inhibit the desired reaction.

Current approaches to find regularities among molecules are often based on so-called descriptors, which usually consist of thousands of binary features, representing (sometimes in a hashed manner) certain substructures of interests, such as aromatic rings or some other predefined small group of atoms [4]. Other descriptors model pairwise atom distances or 3D molecule arrangements. Prediction algorithms then simply use a distance function on these descriptors to define similarity between molecules. More sophisticated algorithms attempt to find boolean combinations of some of these features that are related to different classes [10]. Approaches that try to regard molecules as graphs and derive similarity measures based on transformations of these graphs were also proposed [9]. However, such notions of similarity, based on a particular descriptor with a corresponding metric, only model limited aspects of molecular similarity well. Therefore attempts to directly extract relevant substructures from a collection of molecules are of persistent interest.

Recently an approach was presented that finds linear fragments [7], i.e. chains of atoms, using an algorithm similar to the well-known Apriori association rule mining method [1]. However, the restriction to linear fragments is limiting in many real-world applications, since substructures of interest often contain rings or branching points. Nevertheless, the idea to use an association rule mining algorithm by regarding the molecules as a set of nodes (instead of the usual bit sets) has sparked considerable interest. A more recent approach [5] finds arbitrary connected substructures by deriving canonical labels for each graph. The search is again based on the Apriori algorithm and hence still relies on frequent reembeddings of fragments in order to determine valid intermediate candidates throughout the search.

In this paper we present an algorithm that also finds arbitrary connected substructures but avoids frequent reembeddings by using a different search strategy. The algorithm maintains parallel embeddings of a fragment into all molecules throughout the growth process and exploits a local order of the atoms and bonds of a fragment to prune the search tree, which results in faster search and allows for a restricted depth first search algorithm, similar to the Eclat association rule mining algorithm [12].

We first present the main algorithm, followed by a discussion of results obtained on the HIV-screening dataset from the National Cancer Institute [11] and conclude with a brief discussion of possible extensions of our method.

2. The Induction Algorithm

In this section we describe our algorithm by developing it from algorithms for the well-known task of association rule induction. We start by reviewing the search schemes for fre-

0-7695-1754-4/02 $17.00 © 2002 IEEE

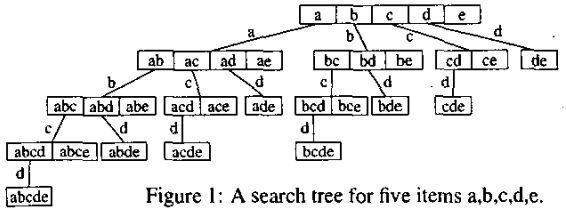

Figure 1: A search tree for five items a,b,c,d,e.

quent itemsets in Section 2.1 and then transfer their ideas to the more complicated case of finding molecular substructures in Section 2.2. In Section 2.3 we describe how our algorithm prunes the search tree based on a local order of the atoms and bonds of a fragment. In section 2.4 we discuss a simple example and in Section 2.5 we study why it is impossible to avoid *all* redundant search. Finally, in Sections 2.6 and 2.7, we describe how to embed core structures to start the search from and how to find contrast structures that distinguish between groups of molecules.

2.1. Association Rules and Frequent Itemsets

The induction of association rules is a powerful method for *market basket analysis*, which aims at finding regularities in the shopping behavior of customers of supermarkets, mail-order companies, on-line shops and the like. With it one tries to find sets of products that are frequently bought together, so that from the presence of certain products in a shopping cart one can infer (with a high probability) that certain other products are present. The main difference to our task of finding frequent substructures of molecules is that in market basket analysis we have only simple *sets of items* to deal with, while in molecular substructure analysis we have to take the chemical connectivity, i.e. the bonds connecting individual atoms, into account as well.

The best-known association rule algorithms are *Apriori* [1] and *Eclat* [12]. Both work in two steps: First the *frequent itemsets* (often misleadingly called *large itemsets*) are determined. These are sets of items that have at least a given minimum support, i.e., occur in at least a given percentage of all transactions. In the second step association rules are generated from these frequent itemsets. Here we focus on the first step, because we are not concerned with generating rules. However, research in association rule induction usually does the same, because finding the frequent itemsets accounts for the greater part of the processing time.

In order to find frequent itemsets, one has to count the transactions different itemsets are contained in. This task consists in traversing a tree like the one shown in Figure 1 and determining the values of the counters in its nodes. Each box represents a counter. The edge labels on a path from the root to a node indicate the common part of the itemsets for which there are counters in that node. The tree is unbalanced, because we are dealing with sets, not sequences: *abc*, for instance, is the same as *bca* and thus only one of these counters is needed. Mathematically, the search tree is a substructure of the subset lattice, having exactly one path to any itemset. In addition, both Apriori and Eclat exploit the simple observation that no superset of an infrequent itemset can be frequent. This observation can be used to further prune the tree, because all counters for itemsets having an infrequent subset can be discarded.

The main differences between Apriori and Eclat are how they traverse this tree and how they determine the counter values. Apriori does a breadth first search and determines the support of an itemset by explicit subset tests on the transactions. An efficient implementation can use a data structure like the tree shown in Figure 1 to store the counters [2, 3]. However, the need to build a data structure like this can also be a severe disadvantage, as it can consume a lot of memory. Furthermore, the subset tests can be costly.

On the other hand, Eclat does a depth first search and determines the support of an itemset by intersecting the transaction lists for two subsets, the union of which is the itemset. The advantage is that not all counters have to be kept in memory, especially if one allows for some unnecessary tests, which could be avoided only by checking *all* subsets. A disadvantage is that several transaction lists have to be kept in memory at the same time—lists that can be very long, especially for small itemsets. More sophisticated approaches try to combine the advantages, using Apriori for the first levels of the tree (usually only 2 or 3) and Eclat as soon as the transaction lists are short enough [12, 6].

2.2. Frequent Substructures of Molecules

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 consider only *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. 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. With this we go beyond [7], who considered only *linear substructures*, i.e., simple chains of atoms without branches. Such simple chains are rarely sufficient in real-world applications.

Most naturally, the search is carried out by traversing a tree of fragments of molecules, similar to the tree of itemsets shown in figure 1. The root of the tree is the 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 the tree shown in Figure 1 means adding an item to an itemset. 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.

As indicated above, there are basically two ways in which the search tree can be traversed: We can use either a breadth first search and explicit subset tests (Apriori) or a depth first search and intersections of transaction lists (Eclat). For our task the Eclat approach is clearly preferable, because the disadvantages of the Apriori approach become considerably more severe: 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, are 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 suffers, because 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 longer. 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 hash table and an array of lists of embeddings to sort the extensions. The hash table associates an embedding with an array index, using a hash code that is computed from the type of the bond that was added in the preceding step, the position (in the substructure) of the atom it starts from, and the position and the type of the atom it leads to. 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.

2.3. 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, when we reviewed the search for frequent itemsets, we also considered a third type of pruning, which we refer to as *structural pruning*. It is responsible for the unbalancedness of the search tree shown in Figure 1: As pointed out above, we do not need a counter for bca, because it is the same itemset as abc. Structural pruning ensures that every itemset is considered in one branch only, even though adding items in different orders can yield the same itemset. In the following we consider how such structural pruning can be done in the search for frequent substructures of molecules, because, obviously, adding bonds in different orders can result in the same substructure.

In order to find a structural pruning scheme, let us analyze the structural pruning of the itemset tree in more detail. Figure 1 shows the basic idea very clearly. The items are ordered, which is indicated by the symbols a, b, etc. This order is, of course, arbitrary. But once it is fixed, the itemsets processed in a node can be constructed as follows: Extend the set of items used as edge labels on the path to the node with an item following the last edge label. Consider, for example, the second node on the third level (count from top to bottom and from left to right): The path to this node has labels a and c. Therefore the set $\{a, c\}$ has to be extended by items following c, i.e. by d and e. Consequently there are counters for the sets $\{a, c, d\}$ and $\{a, c, e\}$ in this node. The same holds for any other node. Obviously, this scheme fixes an order in which items can be added and thus each itemset can be reached only in one possible way.

We organize the nodes of the search tree for molecular substructures in a very similar way. The main difference is that we cannot define a *global* order of the atoms of the molecules, which would correspond directly to the order of the items. Rather, we 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.

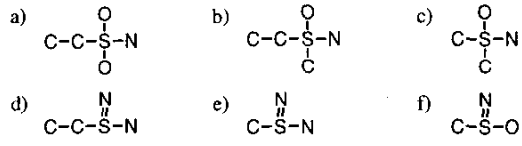

Figure 2: A set of six example molecules.

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 itemset. 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 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.

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.

2.4. An Illustrative Example

As an illustration we consider how our algorithm finds the frequent substructures of the six example molecules shown

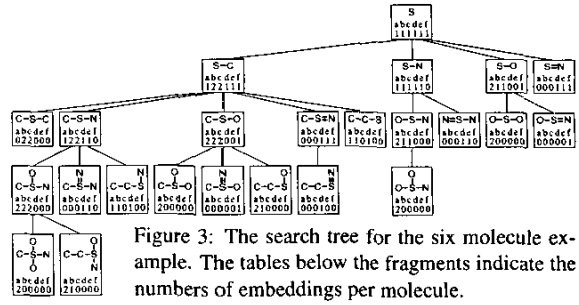

Figure 3: The search tree for the six molecule example. The tables below the fragments indicate the numbers of embeddings per molecule.

in Figure 2[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.

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 3; 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 the 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

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

[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.

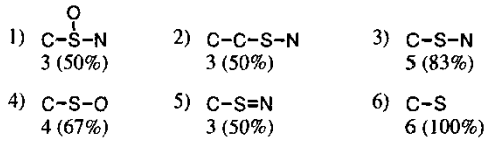

Figure 4: The six frequent substructures that are found in the example in the order in which they are generated.

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.

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 4. 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. This 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.

2.5. 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. 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.

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 have 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 2).

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. We are currently trying to tackle this problem by collapsing rings into special vertices.

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 in only one branch. 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.

2.6. 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 the following 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).

2.7. Finding Contrast Structures

Our approach to find frequent substructures can easily be extended to find *contrast structures*, that is, substructures that are frequent in a predefined subset of the molecules and infrequent in the complement of this subset. Finding contrast structures 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 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 those substructures that do not satisfy the requirements for a contrast structure.

3. Experimental Results

We applied the presented approach to a number of confidential data sets with substantial success. In order to be able to report results in more detail, we used a well-known, publicly available dataset from the National Cancer Institute, the DTP AIDS Antiviral Screen dataset. This screen utilized a soluble formazan assay to measure protection of human CEM cells from HIV-1 infection. Full details were published in [11]. Compounds able to provide at least 50% protection to the CEM cells were retested. Compounds that provided at least 50% protection on retest were listed as moderately active (**CM**). Compounds that reproducibly provided 100% protection were listed as confirmed active (**CA**). Compounds not meeting these criteria were listed as confirmed inactive (**CI**). Available online [8] are screening results and chemical structural data on compounds that are not covered by a confidentiality agreement. Available are 41,316 compounds of which we used 37,171. Out of these, a total of 325 belongs to class **CA**, 877 are of class **CM** and the remaining 35,969 are of class **CI**.

Atom		CA	CM and CI
C	Carbon	325 (100.0%)	36828 (99.95%)
O	Oxygen	311 (95.7%)	33029 (89.64%)
N	Nitrogen	276 (84.9%)	29234 (79.34%)
S	Sulfur	143 (44.0%)	10926 (29.65%)
...	...		
Se	Selenium	6 (1.9%)	132 (0.36%)

Table 1: Some single-atom fragments occurring in molecules of the HIV database, together with their occurrence frequencies.

NCI lists 75 known active compounds, which are grouped into seven classes: (1) Azido Pyrimidines, (2) Natural Products or Antibiotics, (3) Benzodiazepines, Thiazolobenzimidazoles and related Compounds, (4) Pyrimidine Nucleosides, (5) Dyes and Polyanions, (6) Heavy Metal Compounds, and (7) Purine Nucleosides.

As described above, our molecular fragment mining algorithm requires a seed fragment, which may be empty. For the HIV dataset an empty core results in numerous embeddings of trivial fragments, such as single carbon atoms or small combinations of carbon atoms only. However, fragments of interest contain at least one non-carbon atom, which enabled us to seed the algorithm using the remaining atoms. The list of atoms can be obtained through various methods. We simply started from an empty core, restricted the fragment size to one atom, and ran our molecular fragment miner. Parts of the resulting list of atoms together with their occurrence frequencies are listed in Table 1.

Obviously, due to space constraints, we cannot report in detail about seeding the algorithm with each of these atoms. In the following we therefore concentrate on a few experiments to demonstrate how the proposed method picks out relevant fragments in some of these groups.

3.1. Nitrogen based Fragments

First we focus on compounds containing a nitrogen atom. We used a minimum support of 15.0% for compounds of class **CA** and a maximum support of 0.1% for the complement[3] (classes **CM** and **CI**). 171 fragments were generated

[3]Thresholds were selected "top-down", i.e., starting with large settings (that quickly resulted in no reported fragments) the thresholds were subsequentially lowered until a small number of fragments was reported.

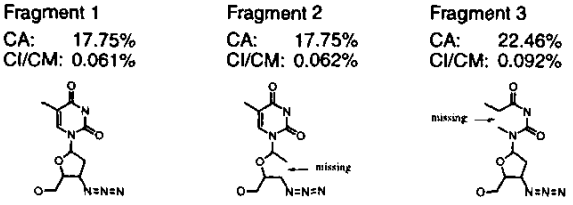

Figure 5: The three largest fragments containing a nitrogen atom.

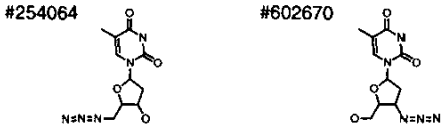

Figure 6: One particular compound (#254064) along with a representative of the normal structure (#602670).

within approximately 20 minutes.[4] The three largest fragments found are shown in Figure 5.

Note how the first two fragments have essentially the same coverage. The only difference is one additional compound of class **CI** that contains fragment 2. In this compound the three nitrogen atoms are connected to the 4-carbon-oxygen ring through an intermediate carbon. This results in a fragment where the carbon connected to the three nitrogen atoms is part of a ring in all cases but one, which prevents the search algorithm from closing the ring. The ring was closed in the first fragment, however, resulting in one less inactive compound being covered. Figure 6 shows this specific compound along with another compound of class **CA** that exhibits the more typical structure.

The third fragments coverage is substantially different, even though its structure is almost identical. The only difference is the double bond between two carbons that closes the second ring in the first fragment, which is missing in Fragment 3. However, some active compounds have a single bond instead of a double bond between these carbons and hence not closing this ring results in a slightly smaller fragment with a much higher coverage. This fragment successfully picks out compounds of class Azido Pyrimidines, a well-known inhibitor of HIV-1. Below we will discuss how "softening" the matching criteria allows us to tolerate such small differences between otherwise identical fragments, which makes this approach also more useful for chemists, who tend to regard such structures as similar.

3.2. Sulfur based Fragments

Next we seeded the algorithm with a sulfur atom. We chose the thresholds support=10% and complement=0.5%, which

[4]We used a Java implementation of our algorithm on a 1Ghz Xeon Dual-Processor machine with 1GB of main memory using jre1.3.1.

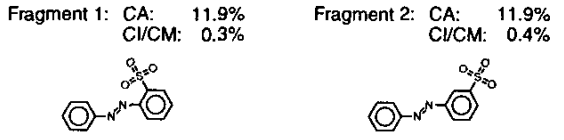

Figure 7: The two largest fragments with a sulfur atom. These fragments are common to 11 of the 13 Dyes and Polyanions.

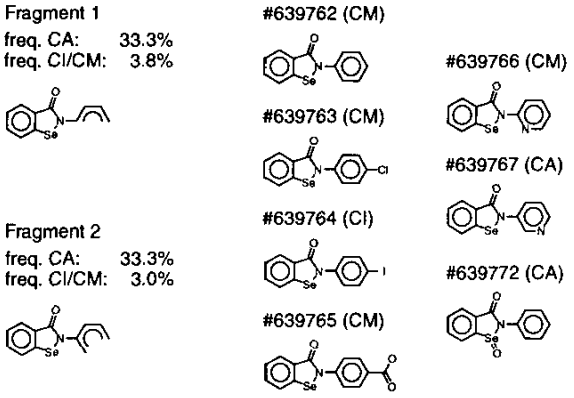

Figure 8: Fragments (left) and corresponding compounds (right) containing a Selenium atom. The two compounds of class **CA** are members of the group of Heavy Metal Compounds.

generated a list of 122 fragments in under one minute. The first two (which also happen to be the largest ones with 18 atoms and 19 bonds, resp.) are shown in Figure 7.

Note how these two fragments differ only in the location of the SO_3 group. Both fragments exhibit a lift of well above 25 and pick out 11 of the 13 molecules listed as Dyes and Polyanions. We miss only two of the remaining Dyes and Polyanions (#9617 and #65849), which contain uncommon structures for this family of compounds.

3.3. Selenium based Fragments

An interesting effect of the current method to find fragments can be seen when seeding the algorithm with a Selenium atom (Se). Figure 8 (left) shows the two fragments that are found for a minimum support of 30% and a maximum complement support of 5%.

Clearly the first fragment is sufficient to pick out all 7 compounds from the database (shown on the right of Figure 8). However, the second fragment covers one compound less (#639766) and tries to complete the aromatic ring in both directions in parallel. This results in a conflict with the nitrogen atom in compound #639766 and a fragment which is neither a subset of the other fragment nor has exactly the same coverage. For our algorithm these two fragments are therefore unique and are not pruned.

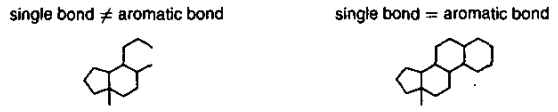

Figure 9: Two fragments extracted from a set of steroids. On the left, single and aromatic bonds were treated as different bond types, on the right, they were treated as the same bond type.

3.4. Treatment of Aromatic Bonds

An important aspect of molecular fragment mining is the treatment of aromatic rings. Since aromaticity is not clearly defined and can be modeled differently (i.e. explicit aromatic bonds vs. alternating single and double bonds) it is desirable to be able to take it into account throughout the mining process itself. We achieve this by modeling aromatic bonds as either single or double bonds with a flag that indicates aromaticity. This allows us to choose to ignore this flag during mining and hence to find fragments that contain either aromatic or single resp. double bonds. The following example illustrates why this is desirable.

Using a small set of steroid compounds we derived fragments that occur in all of them (support=100%) using the standard algorithm. Figure 9 (left) shows the corresponding fragment. Note how only two rings with an incomplete third ring are discovered of the four ring structure that is typical for steroids. However, if we model aromatic bonds as single+flag and allow the algorithm to ignore this flag, the resulting fragment contains all four rings (see Figure 9 (right)). For some steroids this fourth ring consists of single bonds, while others have an aromatic ring at this position. However, most chemists still regard this as the same 4-ring structure. Such selective "tolerance" against some mismatches can therefore make the presented algorithm more useful for real applications.

4. Conclusions

We presented an algorithm to find relevant molecular fragments in large chemical structure databases. The algorithm allows us to focus on fragments that help to discriminate between different classes of molecules. The underlying search method, which is based on a depth first search with structural pruning, makes it possible to find such fragments efficiently, without the need for frequent reembeddings of fragment candidates, which is a known problem of previously reported approaches. We have shown how the proposed method finds relevant fragments using data from a well-known HIV-screening compound database. The extracted fragments successfully model several of the activity classes known for this dataset.

Future work will focus on making the presented approach more meaningful for the underlying application. In particular, finding fragments that match exactly is not of prime interest to chemists. As demonstrate above, some types of ring structures are considered functionally equivalent, which should be taken into account by the search algorithm as well. We are currently exploring ways to include such "fuzziness" into the underlying search algorithm directly.

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. Apriori — Finding Association Rules with the Apriori Algorithm (free computer software under the GLPL). http://fuzzy.cs.uni-magdeburg.de/~borgelt/apriori/

[3] C. Borgelt and R. Kruse. Induction of Association Rules: Apriori Implementation. *Proc. 14th Conf. on Computational Statistics (COMPSTAT)*. Berlin, Germany 2002

[4] R.D. Clark. Relative and Absolute Diversity Analysis of Combinatorial Libraries. *Combinatorial Library Design and Evaluation*, 337–362. Dekker, New York, NY, USA 2001

[5] M. Desphande, M. Kuramochi, and G. Karypis. Automated Approaches for Classifying Structures. *Proc. Workshop on Data Mining in Bioinformatics, BioKDD*, 11-18, 2002

[6] J. Hipp, A. Myka, R. Wirth, and U. Güntzer. A New Algorithm for Faster Mining of Generalized Association Rules. *Proc. 2nd Europ. Symp. on Principles of Data Mining and Knowledge Discovery (PKDD'98, Nantes, France)*, 74–82. LNAI 1510, Springer, Heidelberg, Germany 1998

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

[8] http://dtp.nci.nih.gov/docs/aids/aids_data.html

[9] J. W. Raymond, E. J. Gardiner, and P. Willett. Heuristics for Similarity Searching of Chemical Graphs using a Maximum Common Edge Subgraph Algorithm. *Journal of Chemical Information and Computer Sciences*, 42(2):305–316. American Chemical Society, Columbus, OH, USA 2002

[10] W. J. Streich and R. Franke. Topological Pharmacophores, New Methods and Their Application to a Set of Antimalarials, Part 1: The Methods LOGANA and LOCON. *Quant. Struct.-Act. Relat.*, 4:13–18. J. Wiley & Sons, Chichester, United Kingdom 1985

[11] O. Weislow, R. Kiser, D. Fine, J. Bader, R. Shoemaker, and M. Boyd. New Soluble Formazan Assay for HIV-1 Cytopathic Effects: Application to High Flux Screening of Synthetic and Natural Products for AIDS Antiviral Activity. *Journal of the National Cancer Institute*, 81:577–586. Oxford University Press, Oxford, United Kingdom 1989

[12] 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)*, 283–296. AAAI Press, Menlo Park, CA, USA 1997