*Journal of Computer-Aided Molecular Design*, **12:** 471–490, 1998.
*KLUWER/ESCOM*



# Feature trees: A new molecular similarity measure based on tree matching

Matthias Rarey[a,*] & J. Scott Dixon[b,**]
[a]*German National Research Center for Information Technology (GMD), Institute for Algorithms and Scientific Computing (SCAI), Schloß Birlinghoven, D-53754 Sankt Augustin, Germany;* [b]*SmithKline Beecham Pharmaceuticals, Physical and Structural Chemistry, 709 Swedeland Road, King of Prussia, PA 19406, U.S.A.*

Received 28 October 1997; Accepted 20 March 1998

*Key words:* database screening, molecular descriptors, molecular similarity, molecular superposition, structural alignment

**Summary**

In this paper we present a new method for evaluating molecular similarity between small organic compounds. Instead of a linear representation like fingerprints, a more complex description, a *feature tree*, is calculated for a molecule. A feature tree represents hydrophobic fragments and functional groups of the molecule and the way these groups are linked together. Each node in the tree is labeled with a set of features representing chemical properties of the part of the molecule corresponding to the node. The comparison of feature trees is based on matching subtrees of two feature trees onto each other. Two algorithms for tackling the matching problem are described throughout this paper. On a dataset of about 1000 molecules, we demonstrate the ability of our approach to identify molecules belonging to the same class of inhibitors. With a second dataset of 58 molecules with known binding modes taken from the Brookhaven Protein Data Bank, we show that the matchings produced by our algorithms are compatible with the relative orientation of the molecules in the active site in 61% of the test cases. The average computation time for a pair comparison is about 50 ms on a current workstation.

## Introduction

In the absence of a three-dimensional structure of a target protein, molecular similarity between small organic compounds is the major concept in the search for new lead structures in drug design. There are several important applications for this concept. First of all, possible new lead structures can be retrieved from a database by searching for molecules similar to a known substrate or inhibitor [1]. Other applications are the clustering of molecules with known activity to identify common ways of binding and the search for a set of dissimilar molecules which may form a good set for screening experiments [2].

Because in most applications the number of molecules is very large, a time-efficient way of comparing them must be chosen. The most common approach is the generation of a linear descriptor, usually a bit- or an integer-string, for each molecule. The strings usually store the absence or occurrence of specific features of the molecule. The features are based either on the two-dimensional structural formula of the molecule (structural keys as used in [3] or hashed fingerprints as used in [4]) or on three-dimensional properties (line segments or triangles between possible pharmacophore points [5–9] or the ability of the molecule to bind to a specific protein [10]). A recent evaluation of a large variety of these methods by Brown and Martin [11] has shown that 2D structural descriptors have the best performance in separating active from inactive compounds.

At the other end of the scale of molecule comparison functions are methods based on a structural alignment of two (or a set of) molecules in three-

*To whom correspondence should be addressed. E-mail: Rarey@gmd.de. The Feature Trees software is available for SUN and SGI workstations. Interested readers should visit our WWW page http://cartan.gmd.de/Ftrees or contact the corresponding author.
**Present address: Metaphorics LLC, 419 East Palace #2, Santa Fe, NM 87501, U.S.A.

dimensional space (see for example [12–15] for algorithms). Because of their lengthy computational time, these methods are used for the identification of pharmacophores and in 3D-QSAR/CoMFA [16] analysis applied to small datasets rather than for the computation of similarity indices. A similarity value can be derived from the structural alignment by comparing the functional groups lying close to each other in three-dimensional space. For the intended application, a similarity measure which is derived by matching groups forming interactions with the same parts of the protein is highly appropriate.

In this paper, we present a new class of descriptors lying somewhere in between the classical descriptors mentioned above and the structural alignment tools. A molecule is described by a tree, called a *feature tree*. The nodes of the feature tree represent fragments of the molecule. Edges connect nodes which correspond to fragments which are connected in the two-dimensional structure of the molecule. Thus, the feature tree is a rough approximation to the structural formula of the molecule. Each node of the tree contains a set of *features* which are derived from the molecule fragment corresponding to the node.

To compare two feature trees, a matching between subtrees is computed and a similarity value is derived from this matching. In the Methods section, we describe two different comparison algorithms, called the *split-search* and the *match-search* algorithm. We have combined these algorithms with various features describing shape or physico-chemical properties of molecular fragments.

The concept of reducing molecular graphs to simpler forms is already used for representing and matching generic chemical structures for database retrieval [17]. In this work, we use this concept for the detection of molecular similarity. The reduced graph generation, the feature values, and the comparison algorithms are specially developed for this purpose resulting in an algorithmic framework allowing a very high degree of flexibility in the matching process.

In the Results section, we present the application of feature trees to two different datasets. The first set has been assembled by Briem [10] and contains 972 molecules from the MDDR (MACCS Drug Data Report) data base [18], 35% of them are from 5 different inhibitor classes and the remainder are randomly selected. With this dataset, we verify the ability of our method to identify inhibitors from the same inhibitor class and compare the results to the Daylight fingerprint approach [4]. The second dataset has been assembled by Klebe and Lemmen (discussed in [15]) and contains 58 molecules taken from the PDB (Brookhaven Protein Data Bank) [19]. Because binding geometries relative to a protein are known, this dataset can be used to evaluate the quality of the matchings produced by the comparison algorithms.

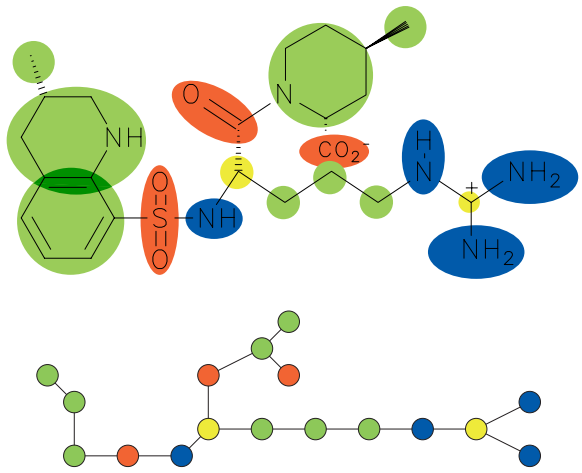



*Figure 1.* Example of a feature tree. Disks overlaid on the molecular structure mark which atoms are combined into one node. The color coding of the disks and corresponding nodes is with respect to an exemplary feature describing the ability of forming interactions: green: mainly hydrophobic, red: hydrogen bond acceptors, blue: hydrogen bond donors, yellow: no direct interactions.

## Methods

### *Feature trees*

A *feature tree* represents a molecule in the following way: a node of the tree describes a set of atoms of the molecule which are connected in the molecular graph. Each atom of the molecule is associated with at least one node. Two nodes which have atoms in common or which contain atoms connected in the molecular graph are connected. Note that in order to have a well-defined tree, the atom sets must be appropriately selected such that no cycles occur. An example of a feature tree is shown in Figure 1. In the following, the terms nodes and subtrees are used for describing feature trees, the term fragment is used to describe a part of the molecule. Subtrees and fragments are always assumed to be connected parts.

Feature trees are able to describe the molecule at various levels of resolution. At the lowest level, the whole molecule can be represented as a single atom set resulting in a feature tree with only one node. At the

highest level, each acyclic atom forms a single node and rings are collapsed to single nodes (the handling of complex ring systems will be explained later). All levels of resolution can be derived from the feature tree of the highest level by taking a subtree and replacing it by a single node representing the union of the atom sets of the nodes belonging to the subtree. This hierarchical nature of feature trees is the first central property and will be used in both comparison algorithms.

The reason for using trees instead of graphs is an algorithmic issue. The goal is the development of efficient comparison algorithms and trees have the property that they can be divided into two independent subtrees just by cutting a single edge. This is the second central property which is important for the comparison algorithms.

*Generation of feature trees*

To generate a feature tree from a molecule, we first identify the ring systems by a biconnected component algorithm [20]. Each ring system is decomposed into a set of rings with the following algorithm: In the first step, a so-called cycle-graph is constructed. For each atom of the ring system, the set of cycles with minimal length containing the atom is determined and collected in a set of initial cycles forming the nodes of the cycle graph. The minimal length cycles for an atom can be computed with a breadth-first-search algorithm similar to the biconnected component algorithm [20]. This algorithm should not be explained in detail here. An edge is added to the cycle graph for each pair of nodes whose cycles share at least one atom.

If the cycle graph is a tree, the algorithm is finished and the cycle graph is the feature tree of the ring system. Figure 3a is an example for this case.

In the case that the cycle graph itself is not acyclic, a second step is necessary to transform the cycle graph to a tree. Here, the biconnected component algorithm is used to decompose the nodes of the cycle-graph. The biconnected components cover exactly all cycles in the cycle-graph. Representing each biconnected component by a single node therefore results in a tree structure. By merging the cycles represented by the nodes of the biconnected components of the cycle-graph, we get a tree which is the feature tree for the ring system.

Figure 2 illustrates the decomposition algorithm on a complex ring system. Examples of some ring systems and the resulting feature trees are shown in Figure 3.

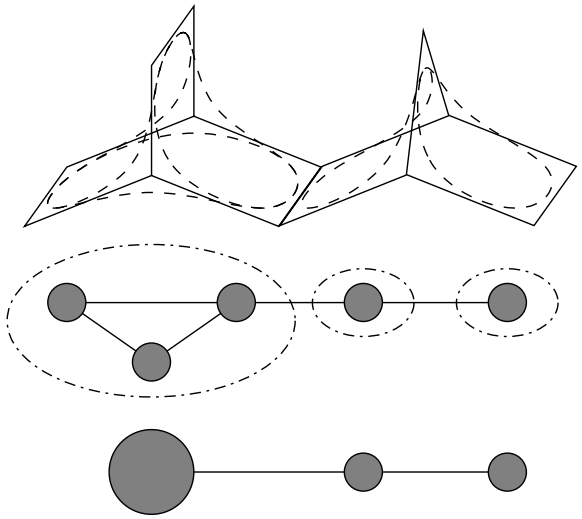


*Figure 2.* Illustration of the ring system decomposition algorithm on two fused bridged rings. All minimal cycles are shown in dashed lines. Below the structures, the cycle graph and its biconnected components (dashed ellipses) are shown. The resulting feature tree is drawn below.

For the acyclic part of the molecule, each atom with more than one bond forms a separate node in the feature tree. The remaining terminal atoms form a single node with the atom they are connected to. Hydrogen atoms are considered explicitly which results, for example, in single nodes for amino- and hydroxygroups. As mentioned before, edges are added between nodes containing atoms that are connected in the molecule. In this step, cycles of size three might

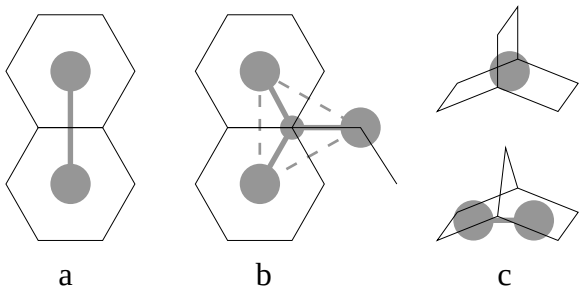



*Figure 3.* Examples for ring systems and the corresponding part of a feature tree (shown in grey). (a) shows a simple ring system composed out of two elementary rings. Note that the atoms that occur in both rings will also be contained in both fragments. (b) shows an example where a zero node (small grey disc) is added in order to resolve the cycle (dashed grey lines) introduced by connecting an acyclic part of the molecule to two rings. In (c), two bridged rings and the corresponding feature trees are shown. In the lower case, the ring system can be decomposed into two five-membered rings. In the upper case, three six-membered rings are found which cannot be uniquely decomposed.

occur as shown in Figure 3b. These cycles are eliminated by introducing a node representing an empty set of atoms which is connected in a star-shaped manner to the three nodes of the cycle. Because the node represents an empty set of atoms, it will be called a zero node in the following.

The resulting tree is unique for the molecule and is used as the feature tree at the highest level of resolution in all further computations.

*Features*

*Feature properties.* The role of a feature is to describe a property of the part of the molecule contained in a feature tree node. Based on the kind of information, features are divided into two classes, steric and chemical features. Features must have the following properties in order to be compliant with the hierarchical nature of the trees. These can be best described by a set of functions used in the comparison algorithms to work with features:

*Generator function.* Given a fragment of the molecule, the generator function calculates a feature value for it.

*Compare function.* Given two feature values, the compare function calculates a similarity value from the range $[0, 1]$, where 1 means very similar (identical) and 0 means dissimilar.

*Combine function.* The combine function takes two feature values and merges them into a single one. The combine function must be consistent with the generator function, i.e. if a molecule can be divided into two pieces (with respect to the edges of the feature tree), then the combine function applied to the feature values of the pieces results in the feature value of the molecule.

A feature value is always associated with a node in the feature tree. Thus we know the corresponding molecule fragment and there is always a consistent combine function: apply the generator function to the merged molecular fragments of the nodes of the feature values to be combined.

*Steric features.* Two steric features describing the size of the fragment are currently used. The first is the number of atoms in the fragment. The generator function simply adds for each atom the reciprocal value of the number of fragments in which it is contained (atoms in ring systems can be contained in multiple fragments, see Figure 3a for an example). The combine function is a simple addition and the comparison function is

$$c(a,b) = \begin{cases} 1, & \text{if } a+b=0, \\ \dfrac{2\min(a,b)}{a+b}, & \text{otherwise.} \end{cases}$$

where $a, b$ are the number of atoms of the compared fragments. The case $a+b=0$ occurs if both fragments have no atoms, i.e. if two zero nodes are matched.

The second feature is an approximated van der Waals volume. First, the volume of an atom $a$ is set to the volume of the corresponding van der Waals sphere. Then, for each outgoing bond to a non-hydrogen atom, the plane of intersection between the two connected atoms is computed and the volume of the cap lying away from the center of $a$ is subtracted. This procedure is a highly simplified variant of volume computation algorithms based on voronoi tesselations [21]. It yields sufficient approximation of volume and is additive over atoms. The generator function adds the computed volume of the atom divided by the number of fragments the atom is contained in. Combine and compare function are defined as above.

Currently, there is no feature describing the shape of a fragment. The generator function of a feature should be independent of the conformation of the molecule which makes a shape description difficult.

*Chemical features.* To describe chemical properties of molecular fragments, we focus on the interactions a fragment is able to form with a potential receptor. For comparison, interactions are divided into a fixed number of interaction types. The feature itself is an array where the $i$-th entry describes the ability of the fragment to form an interaction of type $i$. This representation is called an interaction profile. For all features based on profiles, we use the vector addition as the combine function and the following compare function:

$$c(a,b) = \begin{cases} 1, & \text{if } \sum_i (a_i + b_i) = 0, \\ 2\dfrac{\sum_i \min(a_i, b_i)}{\sum_i (a_i + b_i)}, & \text{otherwise.} \end{cases}$$

The first type of interaction profile is derived from the interaction model of the docking tool FLEXX [22].

*Table 1.* Profiles used as chemical features

| Atom type profile | | FLEXX interaction profile | |
|---|---|---|---|
| C sp3 | 1 | H-donor, metal | 3 |
| C sp2, sp1 | 1 | H-acceptor | 3 |
| N sp3 | 1 | Aromatic-ring-center | 1 |
| N sp2, sp1 | 1 | Aromatic-ring-atom, Methyl, Amide | 1 |
| O | 1 | Hydrophobic | 1 |
| P | 1 | | |
| S | 1 | | |
| Fl, Cl, Br, I | 1 | | |
| Other non-hydrogen | 1 | | |

Columns: atom type profile class description (1) and weighting (2), interaction profile class description (3) and the corresponding weighting (4).

Weights are set by increasing the half interaction energy associated with an interaction type to the next higher integer. Therefore, a weight represents the importance of an interacting group correctly matched. The interaction types and the weighting scheme used are shown in Table 1, columns 3, 4.

For generating a FLEXX interaction profile, the assignment algorithm for interaction types and the corresponding data file from FLEXX is applied. Then, for each atom $a$ of a fragment, the profile entry for each interaction type is set to the number of interactions of this type times the corresponding weight factor.

To evaluate the impact of using interaction based similarity measures, we have also generated atom type profiles as shown in Table 1, columns 1, 2. In this case, no weighting is used.

*Comparison algorithms*

The comparison of two feature trees is based on matching the trees onto each other. A match assigns a subtree of one feature tree to a subtree of the other. The matching is valid if each node occurs in at most one match and the matching is *topology-maintaining*, i.e. if the subtrees are collapsed to single nodes, there is an edge between two nodes of the first tree exactly if there is an edge between the corresponding nodes in the second. An example of a topology-maintaining matching is shown in Figure 4.

To achieve reasonable results, further restrictions on the matches should also be applied. We denote the size of the molecular part represented by a subtree as the size of the subtree. The size of a molecular part

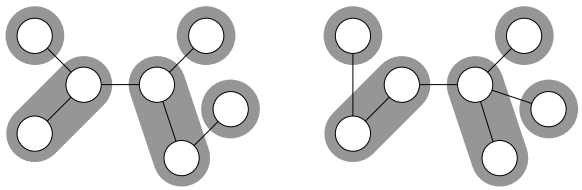


*Figure 4.* Example for a topology-maintaining matching of two feature trees. The grey shapes show the subtrees, the matching is described by the identical relative orientation of the shapes.

is measured by the number of non-hydrogen atoms contained in it.

The first constraint restricts the number of nodes of the subtree and the size of a subtree to be below given thresholds. With this constraint, a comparison algorithm is forced to break down the trees to subtrees of a limited size. Therefore, these thresholds determine the minimum level of resolution achieved by the algorithm.

Second, the size of two matched subtrees must be balanced. A match is considered to be balanced if the ratio of the sizes of the matched subtrees is within an interval $[1/\beta, \beta]$, where $\beta$ is the balance threshold. An exception is a null-match where one of the subtrees is an empty graph.

Given a matching $\mathcal{M}$ between two feature trees $A, B$, a similarity value is computed in the following way: For each match and each feature, the combine function is used to summarize the feature values over the subtrees. Then, a similarity value for the feature is computed with the compare function. The similarity value of a match is then computed by a weighted summation of the similarity values of the features. Currently, we are using only two features simultaneously, a steric feature and a chemical feature, such that the similarity of a match is $s\times$ (steric similarity) $+(1-s)\times$ (chemistry similarity) where $s$ is an appropriate weighting factor. The resulting similarity value for a match $\mathrm{sim}(m)$ is also called the *direct similarity* between the two subtrees.

The similarity value of the feature trees is computed by a weighted average over the similarity values of the matches. The weight factor for each match is the sum of the subtree sizes $\mathrm{size}(m)$ of the match. The total weight is the sum of the sizes of the matched subtrees plus a scaling factor $u$ times the total size of the unmatched subtrees:

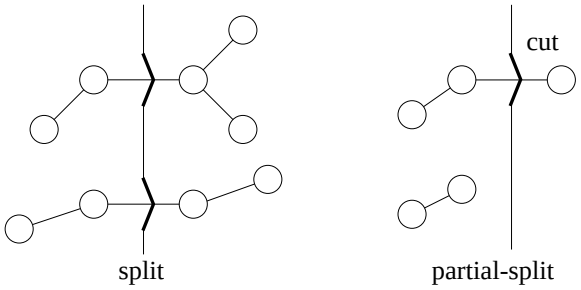



*Figure 5.* Examples of cuts and splits. A directed cut, shown by the wedge, divides a feature tree into two parts, a source part (left) and a target part (right). A split consists of two cuts, one in each of the trees. A partial split assigns only the source part of the cut the whole other tree.

$$S_M(A, B) = \frac{\sum_{m \in \mathcal{M}} \text{size}(m)\text{sim}(m)}{u(\text{size}(A) + \text{size}(B)) + (1 - u) \sum_{m \in \mathcal{M}} \text{size}(m)}.$$

This function works very well in practice but it should be mentioned that it causes some problems for $u \neq 1$ from the theoretical point of view. In the comparison algorithms described later on, matchings in subtrees are optimized independently from each other. Because in $S_M(A, B)$ the denominator depends on the size of the matching, suboptimal matchings of subtrees can form an optimal matching for the whole trees. For now, we neglect this problem and will return to it at the end of the Results section.

To describe the comparison algorithms, we use the following terms: The edges of the tree are described by a pair of anti-parallel directed edges. A *directed cut* is associated with a directed edge $e$ and corresponds to cutting $e$ as well as its anti-parallel edge. Cutting the edges defines two subtrees, one at the source of $e$, the other at the target of $e$. Note that, because of directionality, there are two ways of cutting a tree into two identical pairs of subtrees.

A *split* is a pair of directed cuts, one in each of the trees which should be compared. The split defines which parts are matched onto each other, namely the two source subtrees and the two target subtrees. A *partial split* is a single directed cut. It implies that the source subtree of the cut tree is matched onto the whole other tree while the target subtree is matched to nothing forming a *null-match*. Examples for these terms are given in Figure 5.

*The split-search algorithm*

The first comparison algorithm is based on a divide & conquer strategy and is called the split-search algorithm. The input of the algorithm consists of two subtrees and the output is a similarity value for the two subtrees and two lists: a list of splits and a list of matches defining how the subtrees are subdivided and matched onto each other. The first call of the recursive algorithm is performed with the complete feature trees as the input. During a run, the introduced splits always imply a topology-maintaining matching.

Here, we will give an informal outline of the algorithm. Some details on the algorithm and a more formal description can be found in Appendix A.

At the beginning of the algorithm, the two given subtrees are checked to see if they fulfill all conditions for forming a match such that the recursion can be stopped. Because the matching is always topology-maintaining, only the number of nodes in the subtrees, the size of the subtrees and the size ratio are checked against the given thresholds. In addition, we stop the recursion if the direct similarity between the two subtrees is very low. This avoids spending computation time on comparing two parts of the molecules which are dissimilar even at a low level of resolution. If the recursion is stopped, a match containing the two subtrees is generated and the direct similarity value is returned.

In the recursive case, the algorithm starts by computing a set of splits inside the subtrees. All splits including partial splits are enumerated and checked as to whether they are topology-maintaining. Topology-maintaining splits are exactly those where all previously applied splits are on the same side (source or target) in both subtrees (see Figure 6). We also check that the split is either a partial split or that the resulting subtrees are balanced. Then for each split a score is computed by taking the direct similarities of the cut subtrees and the balance of the two cuts into account. The best scoring $k$ splits are selected for further investigation.

For each of the selected splits, the split-search algorithm is recursively called twice, once for the source and once for the target subtrees of the split and the results are combined to a single similarity value. The split achieving the highest similarity value is selected and appended to the combined list of splits returned by the two recursive calls. Finally, the similarity value is returned.

With the split-search algorithm, it is possible that null-matches occur between matches. This means that

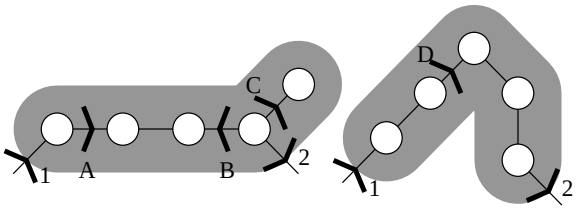



*Figure 6.* Two subtrees with two previous splits 1, 2, and four cuts A, B, C, and D are shown. (A,D) is a topology-maintaining split because split 1 lies on the source side and split 2 on the target side in both subtrees. (B,D) and (C,D) are not topology-maintaining.

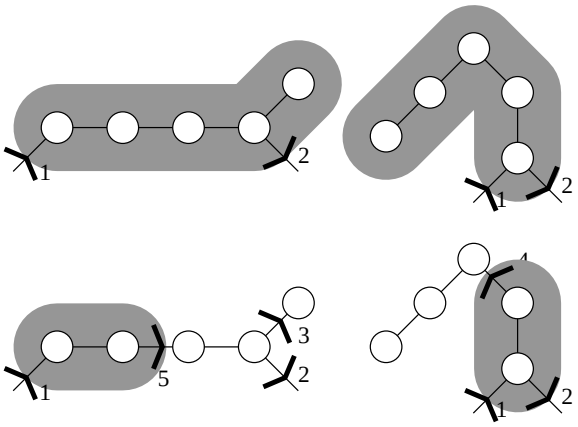



*Figure 7.* The upper part shows two subtrees where balancing the matches makes splitting difficult. After three additional partial splits, the matching shown in the lower part may result which contains unmatched nodes between splits.

there may be unmatched nodes somewhere in the middle of the tree (see Figure 7 for an example). Although these matchings are topology-maintaining, central null-matches are difficult to motivate from the application point of view. Forbidding them without giving up the balance criterion can cause situations where no further splitting is possible. To compensate for this, the number of splits considered at each level of the recursion must be increased which results in higher computation times.

*The match-search algorithm*

The match-search algorithm was developed to be able to generate balanced matches without producing null-matches between them. Because the algorithm explicitly uses the fact that the matches are continuously connected, it is an alternative to the split-search algorithm rather than a replacement. The phases of the match-search algorithm are illustrated in Figure 8.

The underlying idea of the match-search algorithm is the following: At the beginning, the match-search algorithm searches for a set of initial splits as described for the split-search algorithm. The only exception is that partial splits are not allowed (Figure 8, phase 1). Each split implies two pairs of subtrees. In each pair the algorithm searches now for a match, called an extension match, which contains the node lying at the initial split (Figure 8, phase 3). In order to form the extension match new splits have to be introduced to separate the matched subtree from the remaining parts. At this point, the recursion takes place to find the next level of extension matches. The recursion terminates if no remaining parts are left.

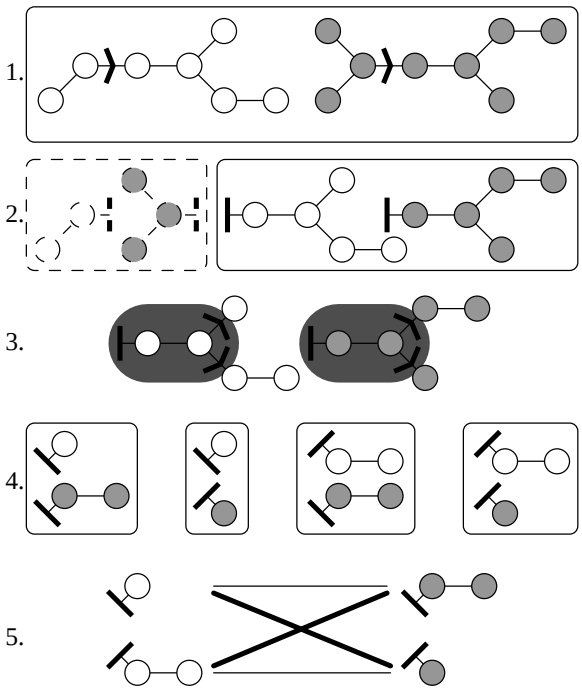



*Figure 8.* Phases of the match-search algorithm as explained in the text. Boxes mark the input of a recursive match-search call. (1) Search for initial splits; (2) First recursive call for matching rooted subtrees; (3) Extending the matching by a match containing both root nodes; (4) Recursive calls for all combinations of subtrees resulting from separating the extension match; (5) Assignment of subtree by searching a maximum weighted matching. The weights are computed by the recursive calls in phase 4. The resulting assignment is shown with bold lines.

Based on Figure 8, we will now describe the phases of the match-search algorithm precisely. Phase 1 is the search for an initial set of splits already described before. The remaining phases are summarized in a recursive procedure. The input to the recursive match-search algorithm is a pair of rooted subtrees. A rooted subtree differs from a subtree by the fact that one node is explicitly marked as the root. In the match-search algorithm, this will always be the node at the split

performed directly before the recursive call. After the initial splitting in phase 1, the recursive match-search procedure is called for each split twice, namely for each pair of rooted subtrees. In Figure 8, the input of the recursive procedure is always surrounded by a box (see phase 2).

The output of the recursive procedure is a similarity value together with a matching of the subtrees. In contrast to the split-search algorithm, the matchings are restricted to those containing the roots of both subtrees in one match. Because this restriction holds at each level of the recursion, a connected set of matches is produced.

The first step of the recursive part of the match-search algorithm is the search for an extension match (phase 3). Starting at the two roots, all matches containing them and fulfilling the size- and balance-criteria are enumerated and scored. A set of the best scoring $k$ extension matches are selected for further investigation.

For each extension match, we have a set of cuts in both subtrees which separates the match from the rest of the subtrees (see wedges in phase 3 of Figure 8). If one subtree is completely contained in the extension match, the similarity value for the subtrees is given by the extension match alone. The remaining parts of the other subtree form null matches.

If there are remaining parts not contained in the match in both subtrees, the algorithm has to compute an assignment of the cuts in order to form new splits from them. There are no constraints on the assignment of cuts but because the assignment determines the matching of subtrees on the next recursion level, it has a large impact on the result. To find the assignment, we perform a recursive call of the match-search algorithm for all combinations of cut subtrees (phase 4). As a result we get a weighted, complete bipartite graph in which the two sets of nodes are the cut subtrees and the edge weights are the similarity values returned by the recursive calls for the corresponding pair of cut subtrees. In this bipartite graph, we compute the assignment yielding the maximum overall similarity value (phase 5 in Figure 8, the final assignment is shown with bold edges). Finally, the maximal similarity value over all examined extension matches is returned.

After the recursion we get a matching and a corresponding similarity value for each initially selected split. The algorithm selects the split resulting in the highest similarity value and returns the value and the corresponding matching.

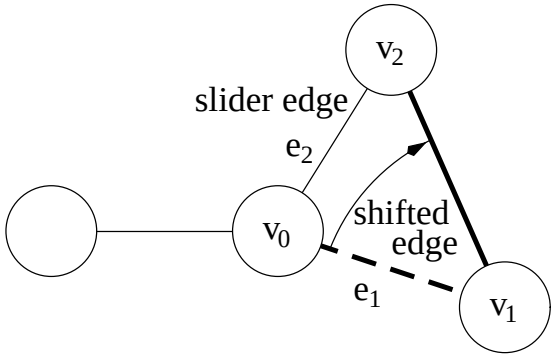



*Figure 9.* Definition of 1-shifts: at nodes of degree higher than two, an edge can be moved to a neighboring node.

A more exact description of the match-search algorithm and some details concerning a time-efficient implementation are given in Appendix B.

*Split-search algorithm with 1-shifts*

The main assumption behind the comparison of feature trees is that two molecules binding to the same active site can be roughly aligned on the basis of their overall topology. Although we are looking at the molecules at a low level of approximation, there are of course cases where this assumption does not hold. One example are thrombin inhibitors which have mainly a Y-shape (see Figure 15) and the location of the central atom varies drastically. To tackle this problem, we have developed a variant of the split-search algorithm which allows minor changes in the topology of the trees, called *1-shifts*.

Let $e_1 = (v_0, v_1), e_2 = (v_0, v_2)$ be two edges of a feature tree with one node in common. Then, a 1-shift of $e_1$ over $e_2$ is the change of $e_1$ to $(v_2, v_1)$ (see Figure 9). $e_1$ is called the shifted edge and $e_2$ the slider edge. The following restrictions are applied to 1-shifts. First, the node degree of $v_0$ must be larger than two. Second, an edge can be shifted only once and there is only one shift over an edge allowed.

We have integrated the concept of 1-shifts in the split-search algorithm. 1-shifts influence the resulting similarity value only if the slider edge is cut. We therefore extend the definition of a split by the edges which are shifted above it. With this concept, the following algorithmic parts in the search for splits must be extended to consider 1-shifts: the test, whether a split is topology-maintaining, the enumeration of splits, and the scoring of splits.

Details on 1-shifts can be found in Appendix C.

*Table 2.* Briem ligand dataset

| Class | #ligands | Average #atoms | Receptor |
|---|---|---|---|
| ACE | 40 | 63 | Angiotensin-converting enzyme |
| TXA2 | 49 | 56 | Thromboxane A2 receptor |
| 5HT3 | 52 | 48 | 5-$HT_3$ receptor |
| HMG | 114 | 64 | HMG CoA reductase |
| PAF | 136 | 70 | PAF receptor |
| RND | 579 | 57 | Random selection |

Columns: Name of the class, number of ligands, average number of atoms (including hydrogen), receptor type. Two molecules from Briem's random selection contain macrocycles and are omitted from the set.

*Table 3.* PDB ligand dataset

| Receptor | #ligands |
|---|---|
| Carboxypeptidase-A | 5 |
| Dihydrofolate-reductase | 2 |
| Endothiapepsin | 5 |
| g-Phosphorylase | 4 |
| Immunoglobulin | 6 |
| Rhinovirus | 8 |
| Streptavidin | 5 |
| Thermolysin | 12 |
| Thrombin | 4 |
| Trypsin | 7 |

Columns: Name of the class, number of ligands.

## Results and discussion

For the validation of the feature tree approach, two datasets are used. The first dataset is assembled by Briem [10]. It contains 972 molecules taken from the MDDR database [18], 391 of them belonging to one of five inhibitor classes and 581 selected randomly (see [10] for details). Two molecules from the randomly selected set contain macrocycles which cannot be handled with feature trees and are therefore omitted. The resulting numbers of molecules and their average size are shown in Table 2.

The second dataset is assembled by Lemmen and Klebe [15] and contains 58 ligands taken from the PDB. With respect to the protein a ligand binds to, the dataset can be divided in ten classes listed in Table 3. For each protein, the corresponding ligands are superimposed with respect to their orientation in the active site. This dataset is used to measure the relative distance of the matched subtrees in the active site.

All calculations described in the following are performed with one set of parameters from which the most important ones are listed in Table 4.

### *Calculation of enrichment factors*

The ability of feature trees to detect molecules belonging to the same class of inhibitors is analyzed using the dataset and protocol described in [10].

For each molecule $m$ of the dataset, an enrichment factor is computed for the percent levels 1 to 10. To compute the enrichment factor for $m$, $m$ is compared to all molecules from the dataset. Let $N$ be the total number of molecules in the dataset, $n(A)$ is the number of molecules in class $A$ that contains $m$, and $h(m, p)$ is the number of molecules from the same class as $m$ within the first $p$% of the dataset sorted by the similarity to $m$. Then, the enrichment factor is

$$E(m, p) = \frac{h(m, p)/(pN)}{(n(A) - 1)/(N - 1)}$$

and the average enrichment factor of class $A$ is

$$E(A, p) = 1/n(A) \sum_{m \in A} E(m, p).$$

Intuitively, the average enrichment factor is the factor by which the average number of hits is increased in the first $p$% of the sorted database using a given similarity measure compared to a random selection.

Figure 10a–c shows the average enrichment factors for the inhibitor classes ACE, TXA2, and 5HT3 at percent levels 1 to 10. Curves are shown for the split-search algorithm using the FLEXX interaction profile as the chemical feature and the van der Waals volume as the steric feature, the split-search algorithm using the atom type profile, and for 1024 bit fingerprints generated with the Daylight software package [4].

The enrichment factor plots for the PAF and HMG inhibitor classes are very similar for the feature tree algorithms and the Daylight fingerprints and are therefore omitted here. Also, enrichment factors for the random class are not shown. As expected, they lie slightly above 1 for the higher percent values (starting from 1.1 at the 1% level).

For the ACE class the enrichment factors produced by the fingerprint approach are higher than those based on feature trees. While for the one and two percent level the difference is quite small (factor 1.04 and 1.11 respectively), it increases to a factor of about 1.3 at the six percent level.

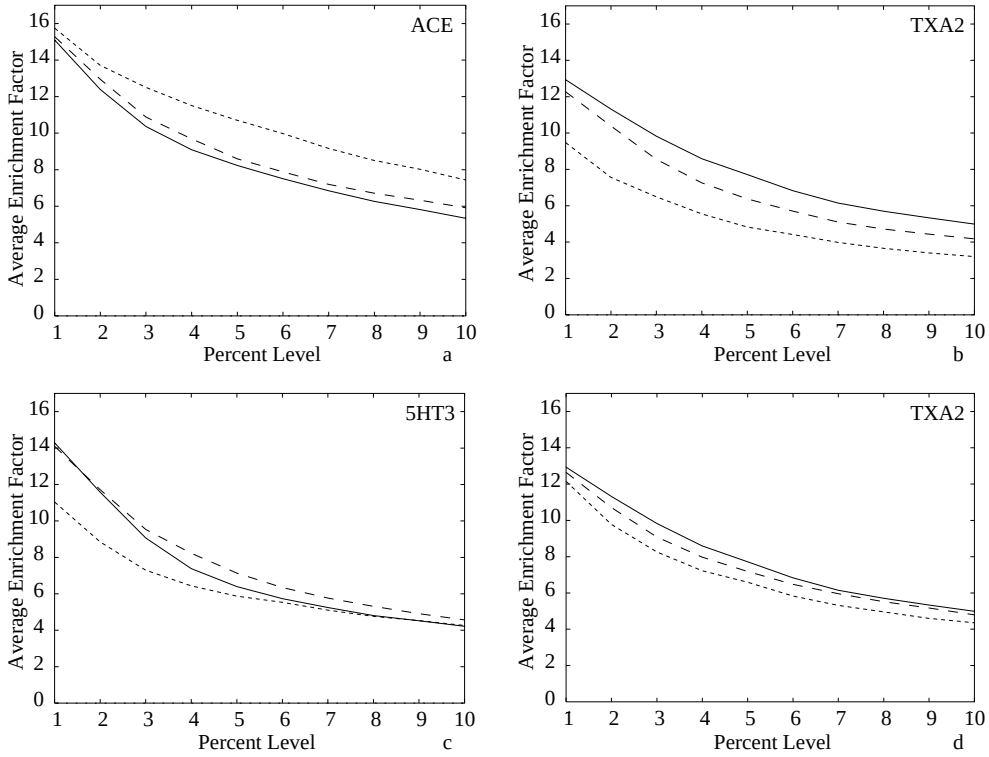



*Figure 10.* Enrichment factors for classes ACE (a), TXA2 (b), 5HT3 (c); graphs for the split-search algorithm with FLEXX interaction profiles are drawn solid, graphs for the split-search algorithm with atom type profiles are drawn dashed, and graphs for the fingerprint approach are drawn dotted. (d) contains the TXA2 class again, the enrichment factors are computed with the split-search algorithm (solid), the match-search algorithm (dashed), and the split-search algorithm with 1-shifts (dot-dashed).

For the TXA2 and 5HT3 class, the feature tree approach produces better enrichment factors over the whole range of percent levels. The differences are largest for the TXA2 class, where the enrichment factor achieved with feature trees is increased by a factor of 1.35 at the 1% level and by a factor of 1.5 for higher percent levels with respect to fingerprints.

In order to analyze the similarity of the hits extracted with feature trees and Daylight fingerprints, we compare the hitlists at the 1% level (first ten molecules). Figure 11 displays the average number of hits for each class, the numbers of common hits are shown in light grey. On the average, about 50% of the hits found with feature trees and Daylight fingerprints are identical. In cases where the feature trees approach show a significant larger number of hits (TXA2 and 5HT3), about 75% of the Daylight hits are also found in the feature tree hitlist.

Top ranking molecules in the feature tree hitlist which have a low rank in the Daylight list are of special interest. These cases demonstrate the ability of feature trees to detect similarities between molecules with less similar two-dimensional structures. Two examples with very large differences in rank are shown in Figures 12 and 13.

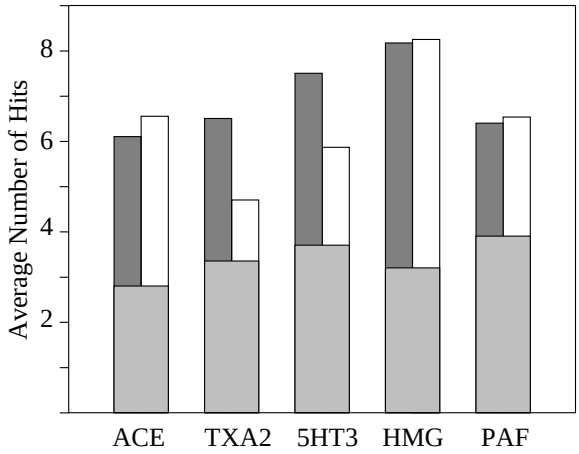



*Figure 11.* Average number of hits under the ten highest ranking molecules with feature trees (dark grey) and Daylight fingerprints (white). The numbers of common hits are shown in ligh grey.

*Table 4.* Parameters of the split-search and match-search algorithms

| Parameter | Value |
|---|---|
| **Recursive stop criteria** | |
| Minimal similarity necessary to initiate a recursive subdivision of two trees | 0.1 |
| Minimal size both subtrees must have to initiate a recursive subdivision | 3 atoms |
| Minimal number of nodes both subtrees must have to initiate a recursive subdivision | 2 nodes |
| **Split-search algorithm** | |
| Maximal number of splits considered in a recursive call ($k$) | 3 |
| Percentage of similarity (versus balancing) in split scoring ($\alpha$) | 60 % |
| **Match-search algorithm** | |
| Maximal number of initial splits | 10 |
| Maximal number of extension matches considered in a recursive call ($k$) | 3 |
| Percentage of the similarity of the extension match versus the remaining unmatched parts in the scoring of extension matches ($\alpha$) | 80 % |
| **Similarity computation and balancing matches** | |
| Percentage of the steric feature versus the chemistry feature in the calculation of the direct similarity value of a match ($s$) | 30 % |
| Factor by which the sizes of two matched subtrees are allowed to differ ($\beta$) | 2.0 |
| Factor by which null-matches are scaled during the calculation of the overall similarity value ($u$) | 0.3 |

The letter in parentheses is the name of the parameter as used in the Methods section.

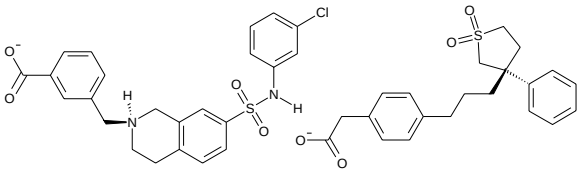


*Figure 12.* An example of a feature tree hit with low two-dimensional similarity from the TXA class. Left: search structure (MDDR Reg. 148448), right: extracted structure (MDDR Reg. 167674) at rank 7 (Daylight fingerprints: rank 700).

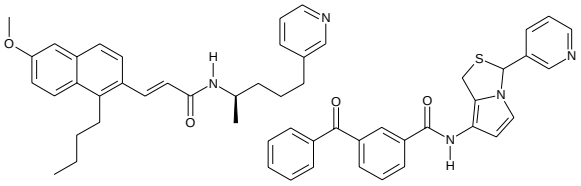


*Figure 13.* An example of a feature tree hit with low two-dimensional similarity from the PAF class. Left: search structure (MDDR Reg. 143082), right: extracted structure (MDDR Reg. 172584) at rank 5 (Daylight fingerprints: rank 729).

There is no clear answer in the comparison of FLEXX interaction profiles to atom type profiles. While the atom type profile is slightly better for the ACE class, the FLEXX interaction profile is significantly better for the TXA2 class. For the ACE and TXA2 class, enrichment factors based on atom type profile are closer to enrichment factors based on fingerprints than enrichment factors based on interaction types are. This makes sense since fingerprints are based on atom types, too. Taking the whole dataset into account, using interaction profiles yield only a minor advantage over atom type profiles. For practical usage however, using interaction profiles enables similarities to be found between molecules with greater differences in two-dimensional structure.

Figure 10d shows the enrichment factors for the TXA2 class for the split-search algorithm, match-search algorithm, and split-search algorithm with 1-shifts. In the TXA2 class, the differences between the three comparison algorithms are largest.

In general, the introduction of 1-shifts lowers the resulting enrichment factors slightly. Therefore, in this case, the 1-shifts cause slightly more unrelated molecules to be identified as similar. Nevertheless, we think that the concept of 1-shifts can be helpful in the search

for molecules with a low degree of two-dimensional structural similarity.

Although the differences in the matchings produced using the split-search algorithm or the match-search algorithm can be quite large for individual cases, there are only small differences in the achieved enrichment factors. For the TXA2 class, the split-search algorithm performs slightly better. The difference between the matchings computed by these algorithms are that the match-search algorithm always produces matchings where all matched subtrees are connected while the split-search algorithm allows null-matches in the middle of the tree. In the TXA2 class, this higher flexibility in the mapping process helps to identify the class.

Enrichment factors for the number-of-atoms shape descriptor are not shown. In general, using van der Waals volumes instead of number of atoms improve the enrichment factors by a small amount.

*RMS distances between matched subtrees*

The goal of the second test is to verify whether the matchings produced by the comparison algorithms correlate with the relative orientation of two corresponding molecules inside an active site. All 173 comparisons between two ligands of the PDB dataset belonging to the same protein class (see Table 3) are computed.

The location of a feature tree node is the center of the smallest sphere enclosing the atoms the node represents. For a subtree, the center is defined to be the centroid of all nodes, weighted by the number of atoms of the node. To measure the quality of a matching, the root-mean-square distance between the centers of all matched subtrees is computed. This RMS value is called the average node distance. In addition, we will define and analyze a superposition RMS distance later in this section.

Figure 14 shows the distribution of the average node distance for the three comparison algorithms. The RMS values fall roughly into three groups. 61% of the test cases are in the first group spanning the range from 0 to 4 Å. The second group, going from 4 to 10 Å, contains 27% of the cases and the third group with RMS deviations above 10 Å contains 12% of the cases (numbers are given for the match-search algorithm).

Nearly all test cases in the third group are comparisons of rhinovirus inhibitors. These inhibitors have nearly a two-fold symmetry and are actually very difficult to align correctly in the absence of the active site [23].

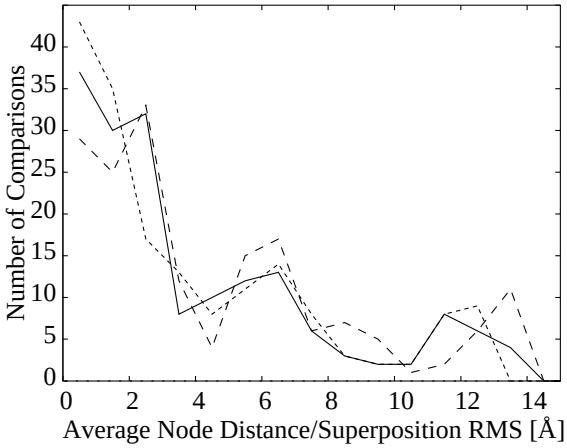



*Figure 14.* Histogram of average node distance values and superposition RMS values. Average node distances are shown for the match-search algorithm (solid) and the split-search algorithm (dashed). The superposition RMS is given only for the match-search algorithm (dotted). Average node distance and the superposition RMS are defined in the text.

Taking the low level of approximation of feature trees into account, matchings with an average node distance of below 4.0 Å map parts of the molecules onto each other which bind into the same region of the active site. As an example, the feature trees for NAPAP (1dwd) and argatroban (1dwc) taken from the thrombin class are shown in Figure 15. To visualize the matches, two nodes of each match representing the subtrees are connected by a line. Thin tree edges represent cut edges, therefore all nodes connected by thick drawn edges form a subtree. The matching in the shown example has an average node distance of 2.0 Å.

Matchings with an average node distance above 4 Å map at least parts of molecules onto each other which bind to different locations in the active site. Two examples of this group are shown in Figures 16 and 17. In these figures, the nodes are color-coded by their interaction profile. Hydrophobic nodes are green, hydrogen bond donor nodes are blue, hydrogen bond acceptor nodes are red, and nodes having a mixed profile have a corresponding mixed color (i.e. a hydroxy-group would be violet).

Figure 16 shows the feature trees of endothiapepsin ligand molecules taken from 4er1 and 5er2. The matching shown has an average node distance of 4.1 Å. While the matches on the right side of the figure are correct, the matches in the middle section are

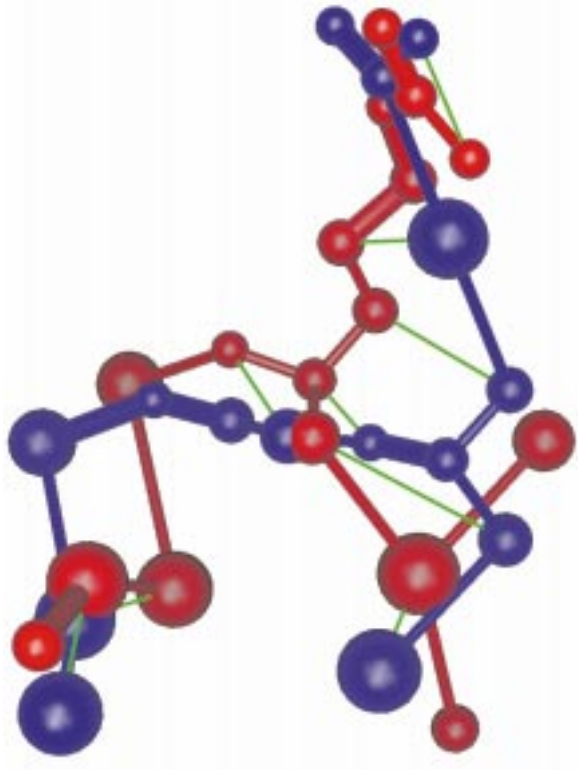


*Figure 15.* Feature trees of NAPAP (blue) and argatroban (red). Nodes are located at the centroids of the corresponding molecule fragments positioned in the active site of thrombin. Thin edges are cut, all nodes connected by thick edges form a subtree. Matched subtrees are connected by green lines.

shifted. On the left side, two branches of the trees are mapped in a crossed manner.

Figure 17a shows two feature trees generated from two carboxypeptidase ligands (3cpa and 7cpa). The molecules vary drastically in size and the matching procedure maps the smaller tree incorrectly to the lower part of the larger tree resulting in an average node distance of 8.5 Å. In Figure 17b, the smaller feature tree is superimposed to the larger one. Looking at the colors of the matched subtrees shows that although it is an incorrect matching, at the approximation level of feature trees it is a quite reasonable matching.

In contrast to the calculation of enrichment factors, the match-search algorithm shows the best performance in this experiment. Forming the average over all comparisons, the node distance is 0.53 Å smaller for the match-search algorithm compared to the split-search algorithm.

Although the introduction of 1-shifts into the split-search algorithm lowers the overall performance, there are some examples, where 1-shifts allow the correct

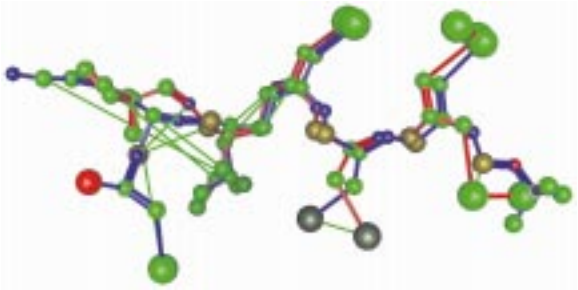


*Figure 16.* Feature trees of endothiapepsin inhibitors, taken from 4er1 (red edges) and 5er2 (blue edges). Nodes are located at the centroids of the corresponding molecule fragments positioned in the active site of endothiapepsin. The nodes are color-coded by the interaction profile (see text). Matched subtrees are connected by green lines. The long green lines on the left hand side indicate matches not agreeing with the relative orientation of the molecules in the active site.

match to be found. One case, the comparison between methotrexate and dihydrofolate, is shown in Figure 18. Here, an amino group of methotrexate is moved from one ring to the neighboring ring inside the pteridine unit. Together with the ring it was shifted to, the amino group is correctly mapped onto the corresponding ring of dihydrofolate.

In principle, the matching produced by a feature-tree comparison algorithm can be used for a structural superposition of molecules. For each of the comparisons in the PDB-dataset, the centers of the subtrees are computed with respect to the molecule in the given conformation. Then, the smaller one of the two molecules is positioned to the larger one by superimposing the subtree centers with a weighted rigid-body fit routine [24]. The superposition RMS is defined as the RMS deviation of the node locations (see above for the definition of node locations) in the calculated position relative to the crystallographically observed position of the feature tree. The superposition RMS is shown for the match search algorithm in Figure 14.

Despite a slight shift towards lower RMS values, the overall shape of the histogram is similar to those previously computed for the average node distance. About 50% of the test cases can be aligned with an RMS deviation of less than 2 Å.

Currently, only the highest scoring matching is returned by the comparison algorithms and is therefore used for the alignment. An improvement can be expected by keeping a set of matchings instead of just one. The matchings can then be used to generate starting orientations for structural superposition algorithms

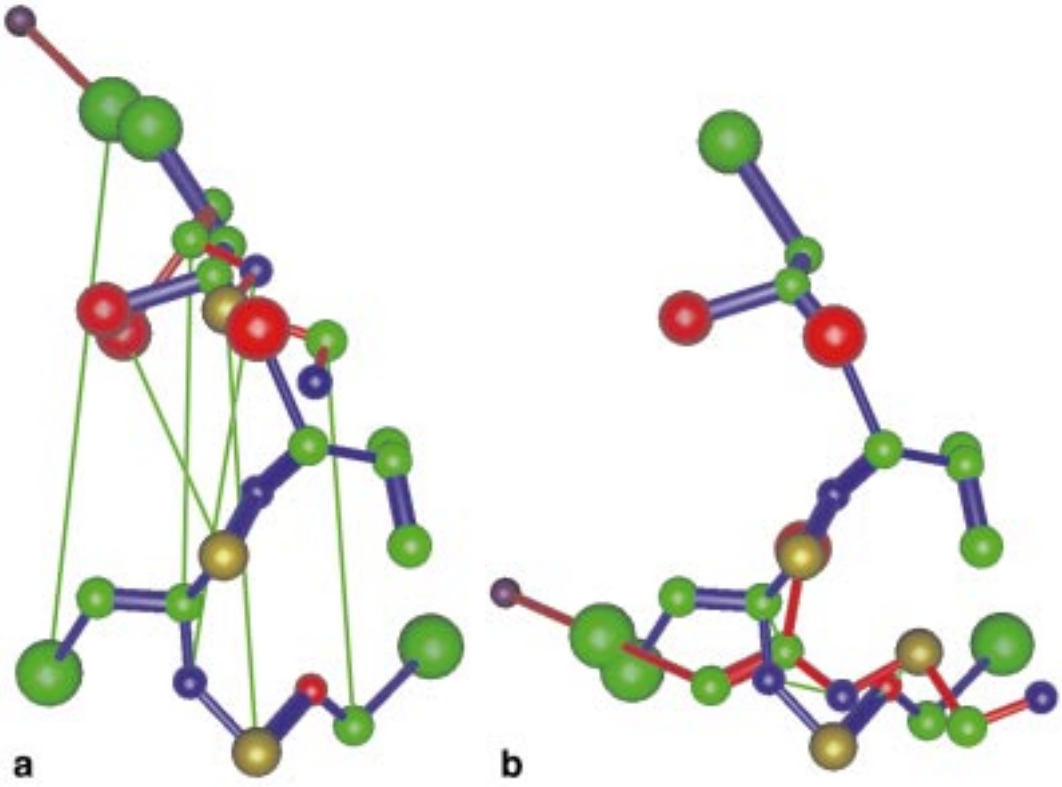



*Figure 17.* Feature trees of carboxypeptidase A inhibitors, taken from 3cpa (red edges) and 7cpa (blue edges). Nodes are color-coded by the interaction profile (see text). (a) shows the two trees in the relative orientation to a superimposed binding site. (b) shows the two trees superimposed on the basis of the calculated subtree matches.

based on numerical optimization (for example SEAL [12]).

*Computation time*

For analyzing the computation time needed to compare feature trees, we have chosen the MDDR dataset because of its larger variety of structures and molecular size. The selection of the steric and chemical features used in the comparison has only a minor influence on the computation time. We therefore used the features with most predictive power, FLEXX interaction profiles combined with van der Waals volume. All computations are performed on a SGI Indigo2 Workstation with an R10000, 195 MHz processor and 64 MB of main memory.

Computing the enrichment factors for all molecules from the first 5 classes results in 378879 feature tree comparisons. The average computation time for a comparison is 80 ms for the split-search algorithm, 38 ms for the match-search algorithm, and 54 ms for the split-search algorithm with 1-shifts. Figure 19 shows the computation time over the input size described by the minimum number of nodes of the two feature trees compared.

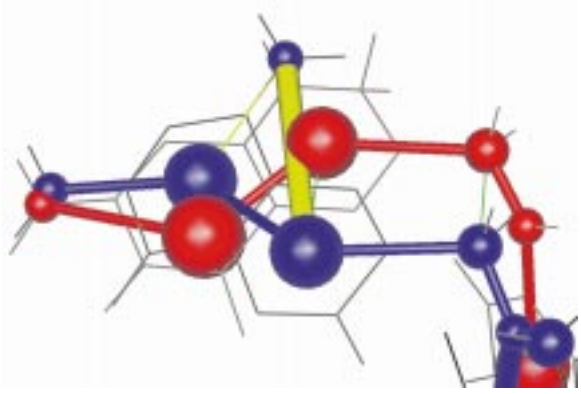


*Figure 18.* Parts of the feature trees of methotrexate (blue) and dihydrofolate (red). The molecular structures are underlaid in grey. The node representing an amino group (top of figure) is shifted to a neighboring ring. The yellow thin line represents the previous location, the thick line the current location after shifting.

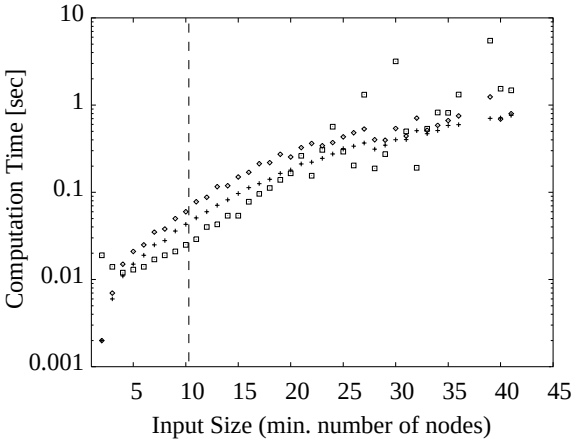



*Figure 19.* Computation time of the comparison algorithms: split-search algorithm (◊), split-search algorithm with 1-shifts (+), match-search algorithm (□).

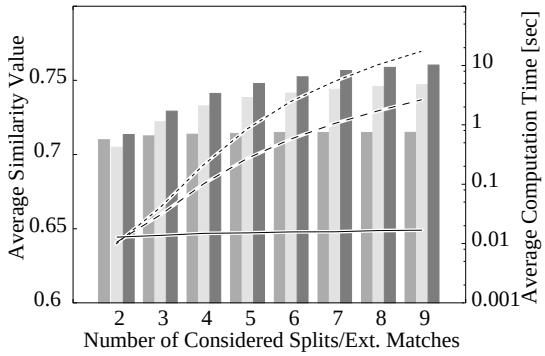



*Figure 20.* The average computation time (lines) and average similarity values (bars) of the comparison algorithms dependent on the number of considered splits (split-search algorithm) or extension matches (match-search algorithm).

For the analyzed range of input sizes, the computation time of each algorithm shows an exponential behavior. Because of the low slope, the algorithms are practical even for larger feature trees containing 30 to 40 nodes. For large feature trees, the match-search algorithm shows much larger deviations in the computation time. This might be an effect of reusing the results from recursive calls or from unconsidered characteristics of the input like the maximum path length or the average node degree of the trees.

1-shifts introduce more degrees of freedom to the matching algorithm. Due to the higher number of possible splits which must be investigated, we would assume an increase in computation time. On the other hand, 1-shifts result in more balanced splits which in summary explains the lower average computation time.

The most important parameter for the computation time as well as the accuracy of the computation is the number of splits or extension matches $k$ which are further investigated on each level of recursion. To study the influence of this parameter, we have reduced the MDDR dataset to three molecules of each class. All pair comparisons between the 18 molecules are performed with each algorithm setting $k$ to the values 2 to 9. The results are summarized in Figure 20.

While the split-search algorithm clearly shows an exponential growth of computation time for small $k$, the match-search algorithm does not. This results from reusing the results of recursive calls which becomes more effective when $k$ increases.

Figure 20 also displays the average similarity value computed with the three algorithms and different $k$ values. The similarity values of different algorithms are not comparable, because the restrictions on the matchings decrease from the match-search over the split-search to the split-search with 1-shifts algorithm (resulting in an increase of the average similarity value in this order). For each of the three algorithms, changing $k$ from 2 to 9 results in a only minor increase of the average similarity value of about 3% which justifies the standard setting of $k = 3$ for the computations presented here.

As mentioned in the Methods section, the described algorithms are heuristic for $u \neq 1$ in the overall similarity function. We therefore counted the number of cases in which an increase of $k$ by 1 causes a lower similarity value by at least 0.001. For the split-search algorithm, this happens in 0.4%, for the split-search algorithm with 1-shifts in 0.3%, and for the match-search algorithm in none of the comparisons.

## Conclusions and outlook

A feature tree is a new way of describing a molecule, allowing a time-efficient computation of molecular similarity. The description is very flexible since the basic tree structure can be combined with several features representing the chemistry of molecular fragments. Two different comparison algorithms, both based on the generation of matchings between two trees, are introduced. Together with the extension of 1-shifts, matchings with different levels of restrictiveness can be produced.

While comparing feature trees is more time-consuming than the usage of linear descriptors, the advantages are manifold. First of all, the feature tree

representation reflects chemical intuition about the way in which similar groups in different ligands can bind to a common receptor site. Second, similarity of only parts of the molecules can be easily addressed.

Compared to the structural key approach, feature trees introduce a notion of fuzziness into the comparison of molecules. The critical definition of the substructures represented in the structural key is avoided. Compared to fingerprints, the calculated similarity value is based on a mapping of molecular fragments which simplifies the interpretation of the results.

There are several interesting directions for further research based on the feature tree approach. Because the main focus of this paper lies in the comparison algorithms, the topic of 'features' is only partially addressed. More advanced chemical features describing electrostatics or a more sophisticated description of interaction capabilities could be used. So far, there is no feature describing the shape of a fragment. The problem in this case is that the feature should be conformation-independent and easy to combine.

The comparison algorithms can be extended in several directions, too. Handling macrocycles and very complex ring systems can be done by exchanging the tree structure with a graph which can be divided by cutting two edges. Then selecting *double-splits*, which cut the graphs at two edges simultaneously could be used to divide the graph into smaller subgraphs. From the application point of view the simultaneous comparison of multiple feature trees would be very useful.

Finally, feature trees offer a new way of addressing the structural superposition problem of molecules. Most approaches consider molecular flexibility by either generating a set of conformations in advance or building the conformation while searching for similarities. Because feature trees are conformation-independent, a rough matching between functional groups can be computed before the problem of molecular flexibility has to be considered.

## Acknowledgements

We thank M. Wagener for useful discussions and many constructive comments on this work and the following for making data available: H. Briem (MDDR dataset), M. Wagener (Daylight fingerprint calculations), C. Lemmen and G. Klebe (PDB dataset). M.R. thanks GMD and SmithKline Beecham for funding his research stay at SmithKline Beecham Pharmaceuticals, PA. We also thank an anonymous referee for a remarkable number of helpful suggestions.

## References

1. Willett, P., J. Mol. Recog., 8 (1995) 290.
2. Warr, W.A., J. Chem. Inf. Comput. Sci., 37 (1997) 134.
3. MDL Information Systems Inc., San Leandro, CA, USA. MACCS II.
4. DAYLIGHT Inc., Mission Viejo, California, USA. DAYLIGHT Software Manual, (1994).
5. Nilakantan, R., Bauman, N. and Venkataraghavan, R., J. Chem. Inf. Comput. Sci., 33 (1993) 79.
6. Sheridan, R.P., Miller, M.D., Underwood, D.J. and Kearsley, S.K., J. Chem. Inf. Comput. Sci., 36 (1996) 128.
7. Bemis, G.W. and Kuntz, I.D., J. Comput.-Aided Mol. Design, 6 (1992) 607.
8. Bath, P.A., Poirrette, A.R. and Willett, P., J. Chem. Inf. Comput. Sci., 34 (1994) 141.
9. Good, A.C., Ewing, T.J.A., Gschwend, D.A. and Kuntz, I.D., J. Comput.-Aided Mol. Design, 9 (1995) 1.
10. Briem, H. and Kuntz, I.D., J. Med. Chem., 39 (1996) 3401.
11. Brown, R.D. and Martin, Y.C., J. Chem. Inf. Comput. Sci., 36 (1996) 572.
12. Kearsley, S.K. and Smith, G.M., Tetrahedron Comput. Method., 3 (1990) 6C 615.
13. Klebe, G., Mietzner, T. and Weber, F., J. Comput.-Aided Mol. Design, 8 (1994) 751.
14. Jones, G., Willett, P. and Glen, R.C., J. Comput.-Aided Mol. Design, 9 (1995) 532.
15. Lemmen, C. and Lengauer, T., J. Comput.-Aided Mol. Design, 11 (1997) 357.
16. Kubinyi, H. (Ed.) 3D QSAR in Drug Design. Theory, Methods and Applications, ESCOM, Leiden, (1993).
17. Gillet, V.J., Downs, G.M., Holliday, J.D., Lynch, M.F. and Dethlefsen, W., J. Chem. Inf. Comput. Sci., 31 (1991) 260.
18. MDL Information Systems Inc., San Leandro, CA, USA. MACCS Drug Data Report (MDDR).
19. Bernstein, F.C., Koetzle, T.F., Williams, G.J.B., Meyer, E.F. Jr., Brice, M.D., Rodgers, J.R., Kennard, O., Shimanouchi, T. and Tasumi, M., J. Mol. Biol., 112 (1977) 535.
20. Corman, T.H., Leiserson, C.E. and Rivest, R.L., Introduction to Algorithms, MIT Press, Cambridge, MA (1990).
21. Goede, A., Preissner, R. and Frömmel, C., J. Comput. Chem., 18 (1997) 1113.
22. Rarey, M., Kramer, B., Lengauer, T. and Klebe, G., J. Mol. Biol., 261 (1996) 3 470.
23. Mattos, C. and Ringe, D., In Kubinyi, H. (Ed.), 3D QSAR in Drug Design. Theory, Methods and Applications, ESCOM, Leiden, 1993, pp. 226–254.
24. Kabsch, W., Acta Crystallogr., A32 (1976) 922.

## Appendix A. The split-search algorithm

The overall structure of the split-search algorithm is summarized in Figure 21. For the sake of simplicity, some algorithmic issues are omitted in the methods section and should be discussed here.

*Forbidden splits.* Because a set of splits is considered in each call of the split-search algorithm, care must be taken to avoid the generation of identical matchings. This can be done by a list $\mathcal{F}$ containing splits which are already considered at higher levels in the recursion and are not available for selection at the current level. The maintaining of the list of forbidden splits $\mathcal{F}$ is integrated in the split-search algorithm shown in Figure 21.

*Scoring and selecting splits.* Let $s$ be a split dividing the trees $n_1, n_2$ into the subtrees $(a_1, a_2)$ and $(b_1, b_2)$ respectively. The score of a split consists of two terms, the balance of the two cuts and the similarity of the two pairs of subtrees implicitly assigned to each other by the split.

The balance of a cut is defined as

$$bal(a_1, a_2) \;=\; \begin{cases} 1, & \text{if } |n(a_1) - n(a_2)| \le 2, \\ 1 - \dfrac{|n(a_1) - n(a_2)| - 2}{n(a_1) + n(a_2) - 2}, & \\ & \text{otherwise,} \end{cases}$$

with

$$n(x) \;=\; \text{number of nodes in subtree } x.$$

Finally, the score of the split is

$$\begin{aligned} \text{score}(s) \;=\; & \alpha(bal(a_1, a_2) + bal(b_1, b_2))/2 \\ & +(1-\alpha)(\text{sim}(a_1, b_1) \oplus \text{sim}(a_2, b_2)) \end{aligned}$$

where sim() denotes the function for computing the direct similarity value from the features, $\oplus$ denotes the function for combining similarity values of matches (see Figure 21 also), and $\alpha$ is a parameter which adjust the balance term versus the similarity term in the scoring function.

In contrast to balancing the matches explained in the Methods section, balancing the cuts is a pure algorithmic issue. The maximal recursion level is reduced by selecting more balanced cuts leading to an increased performance of the algorithm.

```
RECURSIVE_SPLIT_SEARCH (subtree n1, subtree n2,
list of splits F)
% calculates similarity between n1, n2
1  if recursion_stop_criterion (n1,n2)
       return sim(n1,n2); fi;
% minimum number of nodes or minimum size
% reached or sim(n1,n2) is too low
2  S ← find_splits(n1,n2,F);
3  F_new ← ∅
4  foreach split s ∈ S do
5     (n11,n12), (n21,n22) ← apply split s to n1 and n2;
6     sv[s] ← recursive_split_search(n11, n21, F ∪
      F_new)
          ⊕ recursive_split_search(n12, n22, F ∪ F_new
7     F_new ← F_new ∪ {s}; od;
8  return max{sv[s]|s ∈ S};
```

*Figure 21.* The split-search algorithm: the function sim() calculates the direct similarity value; $\oplus$ denotes the function for combining similarity values of different matches. The tracing of splits and matches is not shown.

The number of splits selected in the find_splits procedure depends on the recursion level. Starting from a fixed number of splits at level 0, the actual selected number of splits is reduced by one every four recursion levels until the minimum of 2 selected splits is reached. Compared to a constant number of selected splits on each level, this model yields a significant reduction in computation time with only a minor effect on the results.

*Port-strings* The search for new splits (procedure find_splits in Figure 21) is obviously the most time-consuming step of the split-search algorithm. Once a few splits are set, a large portion of possible new splits will not be topology-maintaining. An efficient way to find out whether a split is topology-maintaining is therefore an important component of the split-search algorithm.

Let $n$ be a subtree separated by a set of splits $s_1, \ldots, s_k$. We define the *port* of a directed cut $c$ to be the node of $n$ incident to the edge cut by $c$. The *port-string* for a directed edge $e$ in the subtree contains $k$ bits, the $i$-th bit is set to one exactly if the $i$-th port lies in the direction of the target node of $e$. The definition of port-strings is illustrated in Figure 22.

For two subtrees which should be compared in a call of the split-search algorithm, the port-strings can be computed in linear time. Then, a new split $s$ cut-

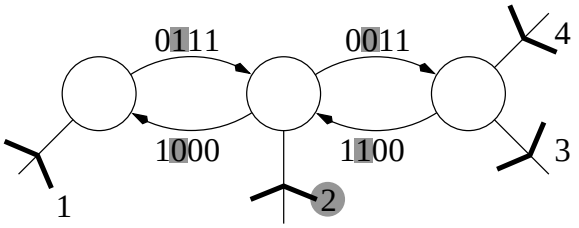



*Figure 22.* Definition of port-strings. The shown subtree is separated by four splits (wedges with numbers 1 to 4). The port-strings therefore contain four bits, the *i*th bit encodes the direction in which the *i*th port is located. The bits belonging to port 2 are highlighted with grey boxes in all port-strings.

ting the directed edges $e_1, e_2$ is topology-maintaining exactly if the port-strings of $e_1$ and $e_2$ are identical.

Port-strings collect data of a whole subtree relative to one edge of it. The same technique is used for all data needed to calculate the score of a split, namely, the number of nodes and the combined feature values on both sides of an edge.

## Appendix B. The match-search algorithm

The match-search algorithm is shown in Figure 23. As in Appendix A, important algorithmic issues not contained in the Methods section can be found here.

*Scoring initial splits.* Because initial splits are not allowed to be partial, splits with high score will always cut the two feature trees somewhere in the middle. This is inappropriate if the sizes of the two molecules which should be compared is very different. We therefore change the balance term in the split scoring function:

$$\text{score}(s) = (1-\alpha)(\text{sim}(a_1,b_1) \oplus \text{sim}(a_2,b_2)) + \alpha \begin{cases} bal(a_1,a_2), & \text{if } n(a_1)+n(a_2) < n(b_1)+n(b_2), \\ (bal(a_1,a_2)+bal(b_1,b_2))/2, & \text{if } n(a_1)+n(a_2) = n(b_1)+n(b_2), \\ bal(b_1,b_2), & \text{if } n(a_1)+n(a_2) > n(b_1)+n(b_2). \end{cases}$$

With this adaptation, only the cut in the subtree with the smaller number of nodes must be balanced to achieve a high score.

*Enumerating extension matches.* The central part of the find_matches procedure is the enumeration of extension matches. An extension match consists of two subtrees, one from each input subtree containing the root node. Therefore, the key problem is the enumeration of subtrees from a tree containing the root node.

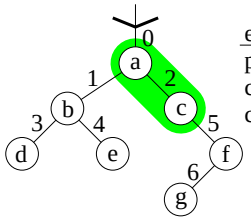
The first step of the algorithm is to sort the edges in a breadth-first order starting from the cut edge at the root node (see Figure 24). A subtree is now represented by a *cut-string* where the *i*th character describes the status of the $i$-th edge: 1 (cut), 0 (not cut), or x (part of a cut subtree).

Based on this description, the increase operation which leads from an arbitrary cut-string to the next one is a simple algorithm shown in Figure 25. The enumer-

```
MATCH_SEARCH_SIMILARITY
(subtree n1, subtree n2)
% initiates similarity computation with the
% match-search algorithm by setting an initial,
% non-partial split
1  S ← finds_splits(n1,n2, {partial splits});
2  foreach split s ∈ S do
3     (n11, n12), (n21, n22) ← apply split s;
4     sv(s) ← recursive_match_search(n11, n21)
         ⊕ recursive_match_search(n12, n22); od;
5  return max{sv(s)|s ∈ S};

RECURSIVE_MATCH_SEARCH
(subtree n1, subtree n2)
% finds a match containing root nodes of n1, n2
% and calculates similarity recursively
1  M ← find_matches(n1,n2);
2  foreach match m ∈ M do
3     cut all edges necessary to seperate m;
4     Ri ← {subtrees from ni seperated from m};
5     foreach pair of subtrees r1, r2 from R1, R2 do
6        rv[r1, r2] ← recursive_match_search(r1, r2);
         od;
7     π ← maximal weighted bipartite matching
         between R1 and R2 using rv[r1, r2] as
         weights and ⊕ to add them;
8     Ui ← subtrees of Si unmatched in π;
9     sv[m] ← sim(m) ⊕ ⊕r∈U1∪U2 sim(r, NULL) ⊕
         ⊕r∈R1\U1 rv[r, π(r)]; od;
10    return max{sv[m]|m ∈ M};
```

*Figure 23.* The match-search algorithm: the function sim() calculates the direct similarity value; $\oplus$ denotes the function for combining similarity values of different matches. The tracing of splits and matches is not shown. The find_splits procedure is the same as in the search-split algorithm (Appendix A) with setting the list of forbidden splits to all partial splits.




| edge | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| parent | | 0 | 0 | 1 | 1 | 2 | 5 |
| child node | a | b | b | d | e | f | g |
| cut-string | 0 | 1 | 0 | x | x | 1 | x |

*Figure 24.* Definition of cut-strings. The edge numbers in the shown tree are in a breadth-first order. The cut-string for the subtree containing nodes $a, b$ is shown in the Table on the right, the subtree itself is highlighted by a grey box.

INCREASE_CUTSTRING (cut-string $c$, integer-array parent)
% *calculates the cut-string following c directly in c*
% *returns TRUE if increase was successful*
1 **if** $\exists i : c[i] \neq 1$ **then return** FALSE; **fi**;
2 head $\leftarrow \min\{i|c[i] = 1\}$; tail $\leftarrow \max\{i|c[i] = 1\}$;
3 **if** head = tail **then** increase lower bounds; **fi**;
4 $c$[tail] $\leftarrow 0$;
5 **foreach** $i >$tail **do**
6 **if** parent$[i]$ = tail **then** $c[i] = 1$;
7 **else if** $c$[parent$[i]$] $= 0$ **then** $c[i] = 1$;
**else** $c[i]$ = x; **fi**; **fi**; **od**;
8 **return** TRUE;

*Figure 25.* The cut-string enumeration: starting with the initial cut-string '1xxx...', the increase operation shown here enumerates subsequently all cut-strings and therefore all subtrees containing the root of a tree.

ation scheme starts with the empty subtree represented by the string '1xxx...'. The first six cut-strings for the tree shown in Figure 24 are listed in Table 5.

*Table 5.* Example of a cut-string sequence

| No. | cut-string | | | | | | | lower bound |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
| initial | 1 | x | x | x | x | x | x | |
| 1. | 0 | 1 | 1 | x | x | x | x | a |
| 2. | 0 | 1 | 0 | x | x | 1 | x | a |
| 3. | 0 | 1 | 0 | x | x | 0 | 1 | a |
| 4. | 0 | 1 | 0 | x | x | 0 | 0 | a |
| 5. | 0 | 0 | 1 | 1 | 1 | x | x | a+b |
| 6. | 0 | 0 | 1 | 1 | 0 | x | x | a+b |

The corresponding tree is shown in Figure 24. The Table shows only the first six of 20 cut-strings. Column 9 (lower bound) contains the nodes which will be contained in all subsequent subtrees and can therefore be used for computing a lower bound on the subtree size.

The algorithm is designed to enumerate subtrees from the root to the leaves of the tree. Once a leading 1 in the cut-string is converted into a 0, it will never be converted again. Therefore, the child node of the corresponding edge (the incident node furthest away from the root) will be part of all following subtrees in the enumeration scheme. Thus a lower bound on the size can be computed by adding the size of all child nodes of edges corresponding to a leading 0.

*Scoring extension matches.* According to the nomenclature in Figure 23, let $n_1, n_2$ be the input subtrees, $m$ be an extension match and $\mathcal{R}_1, \mathcal{R}_2$ be the sets of cut subtrees. At the time when a score for $m$ in the procedure find_match should be computed, the matching $\pi$ between the subtrees in $\mathcal{R}_1, \mathcal{R}_2$ is unknown. In principle, a matching $\pi'$ could be computed in the same way $\pi$ is computed later but with using direct similarity values instead of the results of recursive calls for the weighting (see Figure 23, 5–7). Because scoring must be done for every possible extension match, this would be too expensive in computation-time.

Instead of computing $\pi'$, the following more time-efficient scoring is used. The subtrees in $\mathcal{R}_i$ are virtually linked together such that they can be treated like one subtree $v_i$. Then the score is computed from the similarity of the extension match and the direct similarity of $v_1$ and $v_2$:

$$\text{score}(m) = \alpha\text{sim}(m) + (1 - \alpha)\text{sim}(v_1, v_2).$$

The parameter $\alpha$ is used to adjust the extension match similarity against the similarity of the remaining part of the molecule.

*Reusing results of a recursive match-search.* In the split-search algorithm the input is approximately divided in half on each recursion level yielding a logarithmic depth of the recursion. In contrast, the match-search algorithm cuts only a constant piece of the input and performs the recursive call on the rest of it resulting in a more linear relationship between input and recursion depth.

High computation times can be avoided by reusing results of the recursive match-search procedure. The key observation for a speed up is that all subtrees which occur as input of the recursive match-search procedure are separated from the tree by only one cut. If two feature trees with $n$ and $m$ nodes should be compared with the match-search algorithm, there are only $4(n-1)(m-1)$ different inputs to the recursive match-search procedure. Therefore, the results of the

recursive calls can be stored in a table which will be addressed by the pair of directed cuts separating the input subtrees. Whenever the recursive match-search procedure is called again with the same input, the result can be taken from the table instead of performing the recursive call again.

## Appendix C. The split-search algorithm with 1-shifts

The introduction of 1-shifts in the split-search algorithm is mostly straightforward to do. Two issues should be explained here, the first concerning the scoring of splits, the second concerning the port-string mechanism to check whether a split is topology-maintaining.

*Scoring splits containing 1-shifts.* In the selection of splits, there should be a preference for selecting splits without 1-shifts. We therefore reduce the score for splits containing 1-shifts to 95%. The effect is that 1-shifts are introduced only if they improve the matching while the resulting similarity value is not affected.

*Updating port-strings.* When 1-shifts are performed, precomputed data on the edges like port-strings, subtree sizes, accumulated feature values must be updated appropriately. Normally, only the data at the slider edge has to be changed. As an example, the influence of a 1-shift on the port-strings at the slider edge are shown in Figure 26.

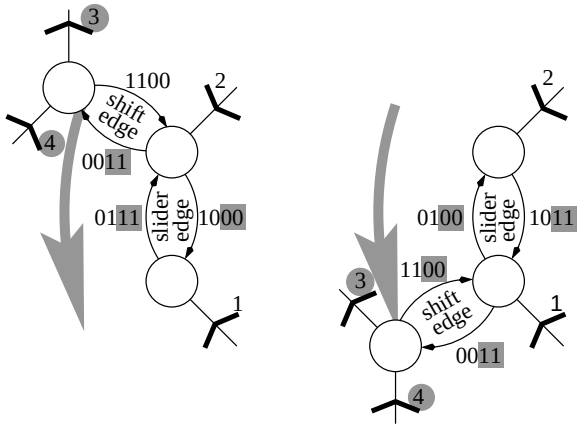



*Figure 26.* On the left hand side, a subtree separated by four cuts (1–4) and the corresponding port-strings are shown. The grey arrow indicates the shift which leads to the subtree shown on the right hand side. The port-strings at the slider edge can be updated by an XOR-operation with the port-string of the shifted edge (pointing away from the slider edge).

The shift edge port-strings contain the information which cuts (3 and 4 in the example) are located at the shifted subtree. After the shift, these cuts will lie at the opposite side of the slider edge. Therefore, the port-strings at the slider edge can be updated by switching the bits corresponding to the moved cuts. The port-strings of all other edges are not affected by the 1-shift.