Latent Structure Pattern Mining

Andreas Maunz[1], Christoph Helma[2] Tobias Cramer[1], and Stefan Kramer[3]

[1] Freiburg Center for Data Analysis and Modeling (FDM),
Hermann-Herder-Str. 3, D-79104 Freiburg im Breisgau, Germany
maunza@fdm.uni-freiburg.de, cramer@seminar-fr.de
[2] in-silico Toxicology, Altkircherstr. 4, CH-4054 Basel, Switzerland
helma@in-silico.ch
[3] Institut für Informatik/I12, Technische Universität München,
Boltzmannstr. 3, D-85748 Garching bei München, Germany
kramer@in.tum.de

Abstract. Pattern mining methods for graph data have largely been restricted to ground features, such as frequent or correlated subgraphs. Kazius et al. have demonstrated the use of elaborate patterns in the biochemical domain, summarizing several ground features at once. Such patterns bear the potential to reveal latent information not present in any individual ground feature. However, those patterns were handcrafted by chemical experts. In this paper, we present a data-driven bottom-up method for pattern generation that takes advantage of the embedding relationships among individual ground features. The method works fully automatically and does not require data preprocessing (e.g., to introduce abstract node or edge labels). Controlling the process of generating ground features, it is possible to align them canonically and merge (stack) them, yielding a weighted edge graph. In a subsequent step, the subgraph features can further be reduced by singular value decomposition (SVD). Our experiments show that the resulting features enable substantial performance improvements on chemical datasets that have been problematic so far for graph mining approaches.

1 Introduction

Graph mining algorithms have focused almost exclusively on ground features so far, such as frequent or correlated substructures. In the biochemical domain, Kazius *et al.* [6] have demonstrated the use of more elaborate patterns that can represent several ground features at once. Such patterns bear the potential to reveal latent information which is not present in any individual ground feature. To illustrate the concept of non-ground features, Figure 1 shows two molecules, taken from a biochemical study investigating the ability of chemicals to cross the blood-brain barrier, with similar gray fragments in each of them (in fact, due to symmetry of the ring structure, the respective fragment occurs twice in the second molecule). Note that the fragments are not completely identical, but differ in the arrow-marked atom (nitrogen vs. oxygen). However, regardless of this difference, both atoms have a strong electronegativity, resulting in a decreased

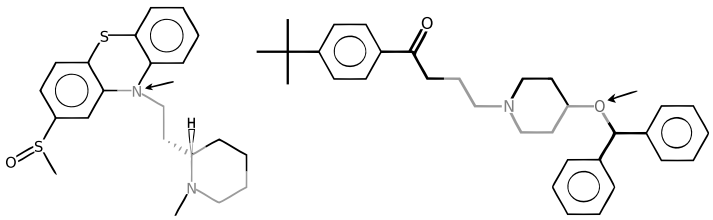

Fig. 1. Two molecules with strong polarity, induced by similar fragments (gray).

ability to cross membranes in the body, such as the blood-brain barrier. So far, the identification of such patterns requires expert knowledge [6] or extensive pre-processing of the data (annotating certain nodes or edges by wildcards or specific labels) [3].

We present a modular graph mining algorithm to identify higher level (latent) and mechanistically interpretable motifs for the first time in a fully automated fashion. Technically, the approach is based on so-called *alignments* of features, i.e. orderings of nodes and edges with fixed positions in the structure. Such alignments may be obtained for features by controlling the feature generating process in a graph mining algorithm with a canonical enumeration strategy. This is feasible, for instance, on top of current a-priori based graph mining algorithms. Subsequently, based on the canonical alignments, ground features can be stacked onto each other, yielding a weighted edge graph (see the left and middle panel of Figure 2). In a final step, the weighted edge graph is reduced again (in our case by singular value decomposition) to reveal the latent structure of the feature (see the right panel of Figure 2). In summary, we execute a pipeline with the steps (a) align, (b) stack, and (c) compress. A schematic overview of the algorithm, called LAST-PM (Latent Structure Pattern Mining) in the following, is shown in Figure 2 (from left to right).

The goal of LAST-PM is to find chemical substructures that are chemically meaningful (further examples not shown due to lack of space) and ultimately useful for prediction. More specifically, we compare LAST-PM favorably to the complete set of ground features from which they were derived in terms of classification accuracy and feature count (baseline comparison), while the tradeoff between runtime and feature count reduction remains advantageous. We also compare accuracy to other state-of-the-art compressed and abstract representations. Finally, we present the results for QSAR endpoints for which data mining approaches have not reached the performance of classical approaches (using physico-chemical properties as features) yet: bioavailability [12] and the ability to cross the blood-brain barrier [7, 4]. Our results suggest that graph mining approaches can in fact reach the performance of approaches that require the careful selection of physico-chemical properties on such data.

The remainder of the paper is organized as follows: Section 2 will introduce the graph-theoretic concepts needed to explain the approach. In Section 3, we will present the workflow and basic components (conflict detection, conflict

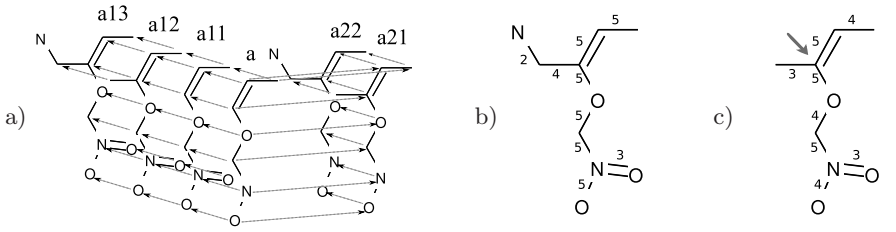

Fig. 2. Illustration of the pipeline with the three steps (a) align, (b) stack, and (c) compress. Left: Aligned ground features in the partial order. Center: Corresponding weighted graph. Right: Latent structure graph.

resolution, the stopping criterion and calculating the latent structure graph). Section 4 discusses the algorithm and also briefly the output of the method. Subsequently, we will present experimental results on blood-brain barrier, estrogen receptor binding and bioavailability data, and compare against other types of descriptors. Finally, we will discuss LAST-PM in the context of related work (Section 6) and come to our conclusions (Section 7).

2 Graph Theory and Concepts

We assume a graph database $R = (\mathbf{r}, a)$, where $\mathbf{r}$ is a set of undirected, labeled graphs, and $a : \mathbf{r} \to \{0, 1\}$ is a function that assigns a class value to every graph (binary classification). Graphs with the same classification are collectively referred to as *target classes*. Every graph is a tuple $r = (V, E, \Sigma, l)$, where $l : V \cup E \to \Sigma$ is a label function for nodes and edges. An *alignment* of a graph r is a bijection $\phi_r : (V, E) \to P$, where P is a set of distinct, strictly ordered, identifiers of size $n = |V| + |E|$, such as (pairs of) natural numbers. Thus, the alignment function applies to both nodes and edges. We use the usual notion of edge-induced subgraph, denoted by $\subseteq$. If $r' \subseteq r$, then r' is said to *cover* r. This induces a partial order on graphs, the more-general-than relation "$\preceq$", which is commonly used in graph mining: for any graphs r, r', s,

$$r' \preceq r, \text{ if } r \subseteq s \Rightarrow r' \subseteq s. \tag{1}$$

Subgraphs are also referred to as *(ground) features*. The subset of $\mathbf{r}$ that a feature r covers is referred to as the *occurrences* of r, its size as *support* of r in $\mathbf{r}$. A node refinement is an addition of an edge and a node to a feature r. Given a graph r with at least two edges, a *branch* is a node refinement that extends r at a node adjacent to at least two edges. Two (distinct) features obtained by node refinements of a specific parent feature are called *siblings*. Two aligned siblings r and s are called *mutually exclusive*, if they branch at different locations of the parent structure, i.e. let v_i and v_j be the nodes where the corresponding node refinements are attached in the parent structure, then $\phi_r(v_i) \neq \phi_s(v_j)$. Conversely, two siblings r and s are called *conflicting*, if they refine at the same location of the parent structure.

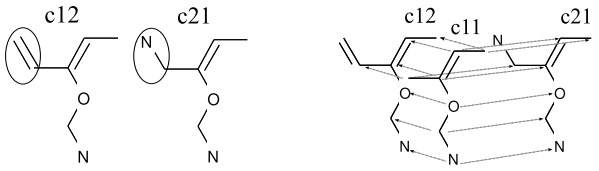

Fig. 3. Left: Conflicting siblings $c12$ and $c21$. Right: Corresponding partial order.

For several ground features, alignments can be visualized by overlaying or *stacking* the structures. It is possible to count the occurrences of every component (identified by its position), inducing a weighted graph. Assume a collection of aligned ground features with occurrences significantly skewed towards a single target class, as compared to the overall activity distribution. A "heavy" component in the associated weighted graph is then due to many ground features significant for a specific target class. Assuming correct alignments, the identity of different components is guaranteed, hence multiple adjacent components with equal weight can be considered equivalent in terms of their classification potential.

Figure 2 illustrates the pipeline consisting of the three steps (a) align, (b) stack, and (c) compress, which exploits these relationships. It shows aligned ground features a, $a11$, $a12$, $a13$, $a21$, and $a22$ in the partial order (*search tree*) built by a depth-first algorithm. The aligned features can be stacked onto each other, yielding a weighted edge graph. Subsequently, latent information (such as the main components) can be extracted by SVD. Inspecting the partial order, we note that refining a branches the search due to the sibling pair $a11$ and $a21$. Siblings always induce a branch in the partial order. Note that the algorithm will have to backtrack to the branching positions.

However, in general, the proposed approach is not directly applicable. In contrast to $a11$ and $a21$, which was a mutually exclusive pair, Figure 3 shows a conflicting sibling pair, $c12$ and $c21$, together with their associated part of the partial order (matching elements are drawn on corresponding positions). It is not clear a priori, how conflicting features could be stacked, thus a conflict resolution mechanism is necessary.

The introduced concepts (alignment, conflicts, conflict resolution, and stacking) will now be used in the workflow and algorithm of LAST-PM.

3 Workflow and Basic Steps

In this section, we will elaborate on the main steps of latent structure pattern mining:

1. Ground features are repeatedly stacked, resolving conflicts as they occur. A pattern representing several ground features is created.
2. The process in step 1. is bounded by a criterion to prevent the incorporation of too diverse features.

id	label
0	7
1	6
2	8
3	6
4	6
5	6
6	8
7	8

id1	id2	label
0	1	1
0	6	1
0	7	2
1	2	1
2	3	1
3	4	2
4	5	1

(a) $a11$ node and edge lists

id	label
0	7
1	6
2	8
3	6
4	6
5	6
6	8
7	6

id1	id2	label
0	1	1
0	6	1
1	2	1
2	3	1
3	4	2
3	7	1
4	5	1

(b) $a21$ node and edge lists

Fig. 4. Node and edge lists for conflicting nodes $c12$ and $c21$, sorted by id (position). Underlined entries represent core nodes and adjacent edges.

3. The components with the least information are removed from the structure obtained after step 2. Then the result (*latent structure*) is returned.

In the following, we describe the basic components of the approach in some detail.

3.1 Efficient Conflict Detection

We detect conflicts based primarily on edges and secondarily on nodes. A node list is a vector of nodes, where new nodes are added to the back of the vector during the search. The edge list first enumerates all edges emanating from the first node, then from the second, and so forth. For each specific node, the order of edges is also maintained. Note, that for this implementation of alignment, the ground graph algorithm must fulfill certain conditions, such as partial order on the ground features as well as canonical enumeration (see Section 4). In the following, the *core component* of two siblings denotes their maximum subgraph, i.e. the parent.

Figure 4 shows lists for features $a11$ and $a21$, representing the matching alignment. Underlined entries represent core nodes and adjacent edges. In line with our previous observations, no distinct nodes and no distinct edges have been assigned the same position, so there is no conflict. The node refinement involving node identifier 7 has taken place at different positions. This would be different for the feature pair $c12/c21$.

Due to the monotonic addition of nodes and edges to the lists, conflicts between two ground features become immediately evident through checking corresponding entries in the alignment for inequality. Three cases are observed:

1. Edge lists of f_1 and f_2 do not contain exactly the same elements, but all rows with identical positions, i.e. pairs of ids, are equal. This does not indicate a conflict.
2. There exists a row in each of the lists with the same position that differs in the label. This indicates a conflict.

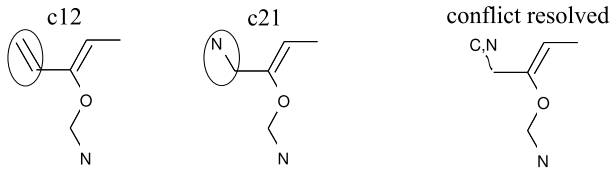

Fig. 5. Conflict resolution by logical OR.

3. No difference is observed between the edge lists at all. This indicates a conflict, since the difference is in the node list (due to double-free enumeration, there must be a difference).

For siblings $a11$ and $a21$, case 1. applies, and for $c12$ and $c21$, case 2. applies. A conflict is equivalent to a missing maximal feature for two aligned search structures (see Section 3.2). Such conflicts arise through different embeddings of the conflicting features in the database instances. Small differences (e.g., a difference by just one node/edge), however, should be generalized.

3.2 Conflict Resolution

Let r and s be graphs. A *maximum refinement* m of r and s is defined as $(r \preceq m) \wedge (s \preceq m) \wedge (\forall n \succeq r : m \succeq n) \wedge (\forall o \succeq s : m \succeq o)$.

Lemma 1. *Let r and s be two aligned graphs. Then the following two configurations are equivalent:*

1. *There is no maximum refinement m of r and s with alignment ϕ_m induced by ϕ_r and ϕ_s, i.e. $\phi_m \supseteq \phi_r \cup \phi_s$.*
2. *A conflict occurs between r and s, i.e. either*
   (a) $v_i \neq v_j$ for nodes $v_i \in r$ and $v_j \in s$ with $\phi_r(v_i) = \phi_s(v_j)$, or
   (b) $e_i \neq e_j$ for edges $e_i \in r$ and $e_j \in s$ with $\phi_r(e_i) = \phi_s(e_j)$.

Proof. Two directions:
"1. $\Rightarrow$ 2.": Assume the contrary. Then the alignments are compatible, i.e. no unequal nodes $v_i \neq v_j$ or edges $e_i \neq e_j$ are assigned the same position. Thus, there is a common maximum feature m with $\phi_m \supseteq \phi_r \cup \phi_s$.
"1. $\Leftarrow$ 2.": Since ϕ is a bijection, there can be at most one value assigned by ϕ for every node and edge. However, the set $\phi_m \supseteq \phi_r \cup \phi_s$ violates this condition due to the conflict. Thus, there is no m with $\phi_m \supseteq \phi_r \cup \phi_s$.

In Figure 3, the refinements of $c11$ have no maximum element, since they include conflicting ground features $c12$ and $c21$. In contrast, refinements of a in Figure 2 do have a maximum element (namely feature $a13$).

As a consequence of Lemma 1, conflicts prove to be barriers when we wish to merge several features to patterns, especially in case of patterns that stretch beyond the conflict position. A way to resolve conflicts and to incorporate two

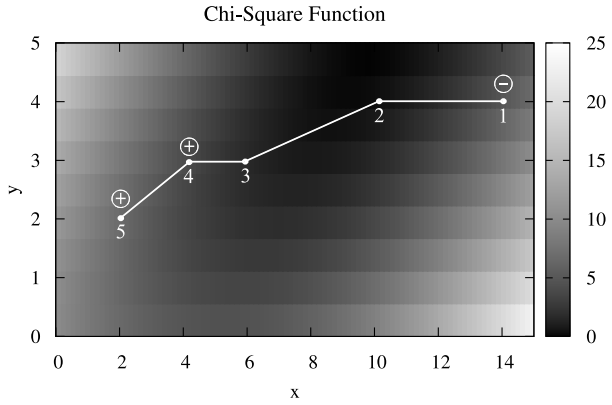

Fig. 6. Contour map of χ^2 values for a balanced class distribution and possible values for a refinement path.

conflicting features in a latent feature is by logical OR, i.e. any of the two labels may be present for a match. For instance, $c12$ and $c21$ can be merged by allowing either single or double bond and either node label of $\{N, C\}$ at the conflicting edge and node, as shown in Figure 5, represented by a curly edge and multiple node labels.

Conflicts and mutually exclusive ground features arise from different embeddings of the features in the database, i.e. the anti-monotonic property of diminishing support is lost between pairs of conflicting or mutually exclusive features. This also poses a problem for directly calculating the support of latent patterns.

3.3 Stopping Criterion

Since the alignment, and therefore equal and unequal parts, are induced by the partial order of the mining process, which is in turn a result of the embeddings of ground features in the database, we employ those to mark the boundaries within which merging should take place. Given a ground feature f, its support in the positive class is defined as $y = |\{r \in \mathbf{r} \mid covers(f, r) \wedge a(r) = 1\}|$, its (global) support as x. We use χ^2 values to bound the merging process, since they incorporate a notion of *weight*: a pattern with low (global) support is downweighted, whereas the occurrences of a pattern with high support are similar to the overall distribution. Assuming $n = |\mathbf{r}|$ the number of graphs, define the *weight* of a feature as $w = \frac{x}{n}$. Moreover, assuming $m = \{r \in \mathbf{r} \mid a(r) = 1\}$, define the *expected support* in the positive [negative] class as wm $[w(n-m)]$. The function

$$\chi^2_d(x, y) = \frac{(y - wm)^2}{m} + \frac{(x - y - w(n-m))^2}{w(n-m)} \tag{2}$$

calculates the χ^2 value for the distribution test as the sum of squares of deviation from the expected support for both classes. Values exceeding 3.84 ($\approx 95\%$

	1	2	3	4	5	6	7	8	9	10
1	0	5	0	0	0	0	0	0	0	0
2	5	0	5	0	0	0	0	3	0	0
3	0	5	0	5	0	0	0	0	0	0
4	0	0	5	0	5	0	0	0	0	0
5	0	0	0	5	0	5	0	0	4	0
6	0	0	0	0	5	0	5	0	0	0
7	0	0	0	0	0	5	0	0	0	0
8	0	3	0	0	0	0	0	0	0	0
9	0	0	0	0	4	0	0	0	0	2
10	0	0	0	0	0	0	0	0	2	0

(a) Weighted original adjacency matrix.

	1	2	3	4	5	6	7	8	9	10
1	0	4	0	0	0	0	0	0	0	0
2	4	0	5	0	0	0	0	3	0	0
3	0	5	0	4	0	0	0	0	0	0
4	0	0	4	0	5	0	0	0	0	0
5	0	0	0	5	0	5	0	0	3	0
6	0	0	0	0	5	0	4	0	0	0
7	0	0	0	0	0	4	0	0	0	0
8	0	3	0	0	0	0	0	0	0	0
9	0	0	0	0	3	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0

(b) Latent structure adjacency matrix.

Fig. 7. Input (left) and output (right) of latent structure graph calculation, obtained by aligning the features $a11 - a22$.

significance for $1df$) are considered significant. Here, we consider significance for each target class individually. Thus, a significant feature f is correlated to either (a) the positive class, denoted by $f_{\oplus}$, if $y > wm$, or (b) the negative class, denoted by $f_{\ominus}$, if $x - y > w(n - m)$.

Definition 1. *Patch*
Given a graph database $R = \{r, a\}$, a patch P is a set of significant ground features, where for each ground feature f there is a ground feature in P that is either sibling or parent of f, and for each pair of ground features $(f_X, g_Y) : X = Y, \quad X, Y \in \{\oplus, \ominus\}$.

The contour map for equally balanced target classes, a sample size of 20 and occurrence in half of the compounds in Figure 6 illustrates the (well-known) convexity of the χ^2 function and a particular refinement path in the search tree with features partially ordered by χ^2 values as $1_{\ominus} > 2 < 3 < 4_{\oplus} < 5_{\oplus}$.

3.4 Latent Structure Graph Calculation

In order to find the latent (hidden) structures, a "mixture model" for ground features can be used, i.e. elements (nodes and edges) are weighted by the sum of ground features that contain this element. It is obtained by stacking the aligned features of a specific patch, followed by a compression step. To extract the latent information, singular value decomposition (SVD) can be applied. It is recommended by Fukunaga to keep $80\% - 90\%$ of the information [2].

The first step is to count the occurrences of the edges in the ground features and put them in an adjacency table. For instance, Table 7(a) shows the pattern that results from the aligned features $a11$, $a12$, $a13$, $a21$, and $a22$ (see Figure 2). As a specific example, edge $1 - 2$ was present in all five ground features, whereas edge $9 - 10$ occurred in two features only. We applied SVD with 90% to the corresponding matrix and obtained the latent structure graph matrix in

Figure 7(b). Here, we removed spurious edges that were introduced by SVD (compression artifacts). As can be seen, the edges leading to the two nodes with degree 3 are fully retained, while the peripheral ones are downweighted. In fact, edge $9 - 10$ is even removed, since it was downweighted to weight 0. In general, SVD downweights weakly interconnected areas, corresponding to a blurred or downsampled picture of the original graph, which has previously proven useful in finding a basic motif in several ground patterns [13].

Definition 2. *Latent Structure Pattern Mining (LAST-PM)*
Given a graph database R, and a user-defined minimum support m, calculate the latent structure graph of all patches in the search space, where for each ground feature f, $supp(f) \geq m$.

4 Algorithm

Given the preliminaries and description of the individual steps, we are now in a position to present a unified approach to latent structure pattern mining, combining alignment, conflict resolution, and component weighting. The method assumes (a) a partial order on ground features (vertical ordering), and (b) canonical representations for ground features, avoiding multiple enumerations of features (horizontal ordering). A depth-first pattern mining algorithm, possibly driven by anti-monotonic constraints, can be used to fulfill these requirements. We follow a strategy to extract latent structures from patches. A latent structure is a graph more general than defined in Section 2: the edges are attributed with weights, and the label function is replaced by a label relation, allowing multiple labels. Since patches stretch horizontally (sibling relation), as well as vertically (parent relation), we need a recursive updating scheme to embed the construction of the latent structure in the ground graph mining algorithm.

We first inspect the horizontal merging: given a specific level of refinement i, we start with an empty latent structure l^i and aggregate siblings from low to high in the lexicographic ordering, starting with empty l^i. For each sibling s and innate l^i, it holds that either

1. s is not significant for any target class, or
2. s is significant for the same target class as l^i, i.e. $X = Y$, for s_X, l^i_Y (if empty, s initializes l^i to its class), or
3. s is significant for the other target class.

In cases 1. and 3., l^i is subjected to latent structure graph calculation and output, and a new, empty latent l^i is created. For case 3., it is additionally initialized with s. For case 2., however, s and l^i are merged, i.e. subjected to conflict resolution, aligning s and l^i, and stacking s onto l^i.

For the vertical or topdown merging, we return l^i to the calling refinement level $i - 1$, when all siblings have been processed as described above. Structures l^i and l^{i-1} are merged, if l^i is significant for the same target class as l^{i-1}, i.e.

Input : Latent structures l_1, l_2; an interval C of core node positions.
Output: Aligned and stacked version of l_1 and l_2, conflicts resolved.

```
1  repeat
2      E.clear() ; E_{l_1}.clear() ; E_{l_2}.clear() ;
3      for j = 0 to (size(C)-1) do
4          index = C[j] ;
5          I = (l_1.to[index] ∩ l_2.to[index]) ;
6          E.insert(I \ C) ;
7          E_{l_1}.insert(l_2.to[index] \ I) ;
8          E_{l_2}.insert(l_1.to[index] \ I) ;
9      end
10     if min(E_{l_1}) ≤ min(E_{l_2}) then M_1 = E_{l_1} else M_1 = E_{l_2} ;
11     if min(E) < min(M_1) then M_2 = E else M_2 = M_1 ;
12     core_new.insert(min(M_2)) ;
13     if M_1 == E_{l_1} then l_2.add_edge(min(M_1)) else l_1.add_edge(min(M_1)) ;
14 until E.size==0 ∧ E_{l_1}.size==0 ∧ E_{l_2}.size==0 ;
15 l_1 = stack(l_1, l_2) ;
16 l_1 = alignment(l_1, l_2, core_new) ;
17 return l_1;
```

Algorithm 1: Alignment Calculation

$X = Y$, for l^i_X, l^{i-1}_Y. Also, condition 1. must not be fulfilled for the current sibling on level $i-1$. Otherwise, both l^i and l^{i-1} are subjected to latent structure graph calculation and output, and a new l^{i-1} is created.

Alignment calculation (Algorithm 1) works recursively: In lines 3-9, it extracts mutually exclusive edges leaving core positions to non-core positions, i.e. there is a distinction between edges leaving the core, but are shared by $l1$ and $l2$ (conflicting edges, E), vs. edges that are unique to either l_1 or l_2 (non-conflicting edges, E_{l_1}, E_{l_2}). The overall minimum edge is remembered for the next iteration, ordered by "to"-node position (lines 11-12). The minimum edge of E_{l_1} and E_{l_2} (line 10; in case of equality, E_{l_1} takes precedence) is added to the other structure where it was missing (line 13).

The procedure can be seen as inserting pseudo-edges into the two candidate structures that were only present in the other one before, thus creating a canonical alignment. For instance, in Figure 4, exclusive edge 0-7 from $a11$ would be first inserted into $a21$, *pushing* node 7 to node 8 and edge 3-7 to edge 3-8 in $a21$. Subsequently, vice versa, exclusive edge 3-8 would be inserted into $a11$, leaving no more exclusive edges, i.e. the two structures are aligned.

This process is repeated until no more edges are found, resulting in the alignment of l_1 and l_2. Line 15 then calls the stacking routine, a set-insertion of l_2's node and edge labels into l_1's and the addition of l_2's edge weights to l_1's, and line 16 repeats the process for the next block of core ids. Due to the definition of node and edge lists, the following invariant holds in each iteration: For the node list, core components are always enumerated in a contiguous block, and for each edge e, the core components are always enumerated at the beginning of the partition of the edge list that corresponds to e. For horizontal (vertical)

merging, we call Algorithm 1 with $l_1 := l^i$, $l_2 := s$ ($l_1 := l^{i-1}$, $l_2 := l^i$). This ensures that l_1 comprises only ground features lower in the canonical ordering than l_2. Thus, Algorithm 1 correctly calculates the alignments (we omit a formal proof due to space constraints).

4.1 Complexity

We modified the graph miner Gaston by Nijssen and Kok [9] to support latent structure pattern mining[4]. It is especially well-suited for our purposes: First, Gaston uses a highly efficient canonical representation for graphs. Specifically, no refinement is enumerated twice. Second, Gaston employs a canonical depth sequence formulation that induces a partial order among trees (we do not consider cycle-closing structures due the complexity of the isomorphism problem for general graphs). Siblings in the partial order can be compared lexicographically.

LAST-PM allows the use of anti-monotonic constraints for pruning the search in the forward direction, such as minimum frequency or upper bounds for convex functions, e.g χ^2. The former is integrated in Gaston. For the latter, we implemented statistical metric pruning using χ^2 upper bound as described in [8]. Obviously, the additional complexity incurred by LAST-PM depends on conflict resolution, alignments, and stacking (see Algorithm 1), as well as weighting (SVD).

- Algorithm 1 for latent structures $l1$, $l2$ takes at most $|l1| + |l2|$ insert operations, i.e. is linear in the number of edges (including conflict resolution).
- For each patch, a SVD of the $m \times n$ latent structure graph is required ($mn^2 - n^3/3$ multiplications).

Thus, the overhead compared to the underlying Gaston algorithm is rather small (see Section 5).

5 Experiments

In the following, we present our experimental results on three chemical datasets with binary class labels from the study by Rückert and Kramer [10]. The nctrer dataset deals with the binding activity of small molecules at the estrogen receptor, the Yoshida dataset classifies molecules according to their bioavailability, and the bloodbarr dataset deals with the degree to which a molecule can cross the blood-brain barrier. For the bloodbarr/ nctrer/ yoshida datasets, the percentage of active molecules is 66.8/ 59.9/ 60.0. For efficiency reasons, we only consider the core chemical structure without hydrogen atoms. Hydrogens attached to fragments can be inferred from matching the fragments back to the training structures. Program code, datasets and examples are provided on the supporting website `http://last-pm.maunz.de`.

[4] Version 1.1 (with embedding lists), see `http://www.liacs.nl/~snijssen/gaston/`.

5.1 Methodology

Given the output XML file of LAST-PM, SMARTS patterns for instantiation are created by parsing patterns depth-first (directed). Focusing on a node, all outgoing edges have weights according to Section 3.4. This forms weight levels of branches with the same weight. We may choose to make some branches optional, based on size of weight levels, or demand all branches to be attached:

- *nop*: demand all (**n**o **op**tional) branches.
- *msa*: demand number of branches equal to **m**aximum **s**ize of **a**ll levels
- *nls*: demand number of branches equal to highest (**n**ext) **l**evel **s**ize

For example, *nop* would simply disregard weights and require all of the three bonds leaving the arrow-marked atom of Figure 2 (right), while *nls* (here also *msa*) would require any two of the three branches to be attached. With *msa* and *nls*, we hope to better capture combinations of important branches. The two methods allow, besides simple disjunctions of atomic node and edge labels such as in Figure 1, for (nested) optional parts of the structure [5].

All experimental results were obtained from repeated ten-fold stratified crossvalidation (two times with different folds) in the following way: We used edge-induced subgraphs as ground features. For each training set in a crossvalidation, descriptors were calculated using 6% minimum frequency and 95% χ^2-significance on ground features. This ensures that features are selected ignorant of test sets. Atoms were not attributed with aromatic information but only labeled by their atomic number. Edges were attributed as single, double and triple, or as aromatic bond, as inferred from the molecular structure. Features were converted to SMARTS according to the variants *msa*, *nls*, and *nop*, and matched onto training and test instances, yielding instantiation tables. We employed unoptimized linear SVM models and a constant parameter $C = 1$ for each pair of training and test set. The statistics in the tables were derived from pooling the twenty test set results into a global table first. Due to the skewed target class distributions in the datasets (see above), it is easy to obtain relatively high predictive accuracies by predicting the majority class. Thus, the evaluation of a model's performance should be based primarily on a measure that is insensitive to skew. We chose AUROC for that purpose. A 20% SVD compression (percentage of sum of singular value squares) is reported for the LAST-PM features, since this gave the best AUROC values of 10, 15, and 20% in preliminary trials in two out of three times. Significance is determined by the 95% confidence interval.

5.2 Validation Results

We compare the performance of LAST-PM descriptors in Table 1 with

1. ALL ground features from which LAST-PM descriptors were obtained (baseline comparison).

[5] Figure 1 is an actual pattern found by LAST-PM in the bloodbarr dataset. See supporting website at `http://last-pm.maunz.de` for the implementation in SMARTS.

Dataset	LAST-PM			ALL	BBRC	MOSS	SLS
	Variant	%Train	%Test	%Test	%Test	%Test	%Test
bloodbarr	nls+nls	84.19	72.20	70.49[a]	68.50[a]	67.49[a]	70.4[b]
nctrer	nls+msa	88.01	80.22	79.13	80.22	77.17[a]	78.4[b]
yoshida	nop+msa	82.43	69.81	65.19[a]	65.96[a]	66.46[a]	63.8[b]

[a] significant difference to LAST-PM.

[b] result from the literature, no significance testing possible

Table 1. Comparative analysis (repeated 10-fold crossvalidation).

2. BBRC features by Maunz, Helma, and Kramer [8] to relate to structurally diverse and class-correlated ground features.
3. MOSS features by Borgelt and Berthold [3] to see the performance of another type of abstract patterns.
4. SLS features by Rückert and Kramer [10] to see the performance of ground features compressed according to the so-called dispersion score.

For ALL and BBRC, a minimum frequency of 6% and a significance level of 95% were used. For the MOSS approach, we obtained features with MoSS [3]. This involves cyclic fragments and special labels for aromatic nodes. In order to generalize from ground patterns, ring bonds were distinguished from other bonds. Otherwise (including minimum frequency) default settings were used, yielding only the most specific patterns with the same support (closed features). For SLS, we report the overall best figures for the dispersion score and the SVM model from Table 1 in their paper. As can be seen from Table 1, using the given variants for the first and second fold, respectively, LAST-PM outperforms ALL, BBRC and MOSS significantly for the bloodbarr and yoshida dataset (paired corrected t-test, $n = 20$), as well as MOSS for the nctrer dataset (seven out of nine times in total).

Table 2 relates feature count and runtime of LAST-PM and ALL (median of 20 folds). FCR is the feature count ratio, RTR the runtime ratio between LAST-PM and ALL, as measured for descriptor calculation on our 2.4 GHz Intel Xeon test system with 16GB of RAM, running Linux 2.6. Since $1/FCR$ always exceeds RTR, we conclude that the additional computational effort is justified. Note that nctrer seems to be an especially dense dataset. Profiling showed, that most CPU time is spent on alignment calculation, while SVD can be neglected.

In their original paper [12], Yoshida and Topliss report on the prediction on an external test set of 40 compounds with physico-chemical descriptors, in

Dataset	LAST-PM	ALL	FCR/RTR
bloodbarr	249 (1.23s)	1613 (0.36s)	0.15 /3.41
nctrer	193 (12.49s)	22942 (0.13s)	0.0084 /96.0769
yoshida	124 (0.28s)	462 (0.09s)	0.27 /3.11

Table 2. Analysis of feature count and runtime.

which they achieved a false negative count of 2 and false positive count of 7. We obtained the test set and could reproduce their exact accuracy with 1 false negative and 8 false positives, using LAST-PM features.

Hu and co-workers [7], authors of the bloodbarr dataset study, provided us with the composition of their "external" validation set, which is in fact a subset of the complete dataset, comprising 64 positive and 32 negative compounds. Their SVM model was based on carefully selected physico-chemical descriptors, and yielded only seven false positives and seven false negatives, an overall accuracy of 85.4%. Using LAST-PM features and our unoptimized polynomial kernel, we predicted only five false positives and two false negatives, an overall accuracy of 91.7%.

We conducted further experiments with another 110 molecule blood-brain barrier dataset (46 active and 64 inactive compounds) by Hou and Xu [4], that we obtained together with pre-computed physico-chemical descriptors. Here, we achieved a AUROC value of 0.78 using LAST-PM features in repeated 10-fold crossvalidation, close to the 0.80 that the authors obtained with the former. However, when combined, both descriptor types give an AUROC of 0.82. In contrast to this, AUROC could not be improved in combination with BBRC instead of LAST-PM descriptors.

6 Related Work

Latent structure pattern mining allows deriving basic motifs within the corresponding ground features that are frequent and significantly correlated with the target classes. The approach falls into the general framework of graph mining. Roughly speaking, the goal of pattern mining approaches to graph mining is to enumerate all interesting subgraphs occurring in a graph database (interestingness defined, e.g., in terms of frequency, class correlation, non-redundancy, structural diversity, ...). Since this ensemble is in general exponentially large, different techniques for selecting representative subgraphs for classification purposes have been proposed, e.g. by Yan [11]. Due to the NP-completeness of the subgraph isomorphism problem, no efficient algorithm is known for general graph mining (i.e. including cyclic structures). For a detailed introduction to the tractable case of non-cyclic graph mining, see the overview by Muntz *et al.* [1], which mostly targets methods with minimum frequency as interestingness criterion. Regarding advanced methods that go beyond the mining of ground features, we relate our method to approaches that provide or require basic motifs in the data, and/or are capable of dealing with conflicts.

Kazius *et al.* [6] created two types of (fixed) high-level molecule representations (aromatic and planar) based on expert knowledge. These representations are the basis of graph mining experiments.

Inokuchi [5] proposed a method for mining generalized subgraphs based on a user-defined taxonomy of node labels. Thus, the search extends not only due to structural specialization, but also along the node label hierarchy. The method finds the most specific (closed) patterns at any level of taxonomy and support.

Since the exact node and edge label representation is not explicitly given beforehand, the derivation of abstract patterns is semi-automatic.

Hofer, Borgelt and Berthold [3] present a pattern mining approach for ground features with class-specific minimum and maximum frequency constraints, that can be initialized with arbitrary motifs. All solution features are required to contain the seed. Moreover, their algorithm `MoSS` offers the facility to collapse ring structures into special nodes, to mark ring components with special node and edge labels, or to use wildcard atom types: Under certain conditions (such as if the atom is part of a ring), multiple atom types are allowed for a fixed position. It also mines cyclic structures at the cost of losing double-free enumeration.

All approaches have in common that the (chemical expert) user specifies high-level motifs of interest beforehand via a specific molecule representation. They integrate in different ways user-defined wildcard search into the search tree expansion process, whereas the approach presented here derives abstract patterns automatically by resolving conflicts during backtracking and weighting.

7 Conclusions

In the paper, we introduced a method for generating abstract non-ground features for large databases of molecular graphs. The approach differs from traditional graph mining approaches in several ways: Incorporating several similar features into a larger pattern reveals additional (*latent*) information, e.g., on the most frequently or infrequently incorporated parts, emphasizing a common interesting motif. It can thus be seen as graph mining on subgraphs. In traditional frequent or correlated pattern mining, sets of ground features are returned, including groups of very similar ones with only minor variations of the same interesting basic motif. It is, however, hard and error-prone (or sometimes even impossible) to appropriately select a representative from each group, such that it conveys the basic motif. Latent structure pattern mining can also be regarded as a form of abstraction, which has been shown to be useful for noise handling in many areas. It is, however, new to graph and substructure mining.

The key experimental results were obtained on blood-brain barrier (BBB), estrogen receptor binding and bioavailability data, which have been hard for substructure-based approaches so far. In the experiments, we showed that the non-ground feature sets improve over the set of all ground features from which they were derived, but also over MOSS [3], BBRC [8] and compressed [10] ground feature sets when used with SVM models. In seven out of nine cases, the improvements are statistically significant. We also found a favorable tradeoff between feature count of and runtime for computing LAST-PM descriptors compared to the complete set of frequent and correlated ground features.

We took bioavailability and blood-brain barrier data and QSAR models from the literature and showed that, on three test sets obtained from the original authors, the purely substructure-based approach is on par with or even better than their approach based on physico-chemical properties only. We also showed that LAST-PM features can enhance the performance of solely physico-chemical

properties. Therefore, latent structure patterns show some promise to make hard (Q)SAR problems amenable to graph mining approaches.

8 Acknowledgements

The research was supported by the EU seventh framework programme under contract no Health-F5-2008-200787 (OpenTox).

References

1. Yun Chi, Richard R. Muntz, Siegfried Nijssen, and Joost N. Kok. Frequent Subtree Mining - An Overview, 2001. *Fundam. Inf.*, 66(1-2):161-198, 2004.
2. Keinosuke Fukunaga. *Introduction to Statistical Pattern Recognition (2nd ed.)*. Academic Press Professional, Inc., San Diego, CA, USA, 1990.
3. Heiko Hofer, Christian Borgelt, and Michael R. Berthold. Large Scale Mining of Molecular Fragments with Wildcards. *Intell. Data Anal.*, 8(5):495–504, 2004.
4. T. J. Hou and X. J. Xu. ADME Evaluation in Drug Discovery. 3. Modeling Blood-Brain Barrier Partitioning Using Simple Molecular Descriptors. *Journal of Chemical Information and Computer Sciences*, 43(6):2137–2152, Oct 2003.
5. Akihiro Inokuchi. Mining Generalized Substructures from a Set of Labeled Graphs. *IEEE International Conference on Data Mining*, 0:415–418, 2004.
6. J. Kazius, S. Nijssen, J. Kok, T. Baeck, and A. P. Ijzerman. Substructure Mining Using Elaborate Chemical Representation. *J Chem Inf Model*, 46:597–605, 2006.
7. Hu Li, Chun Wei Yap, Choong Yong Ung, Ying Xue, Zhi Wei Cao, and Yu Zong Chen. Effect of Selection of Molecular Descriptors on the Prediction of Blood-Brain Barrier Penetrating and Nonpenetrating Agents by Statistical Learning Methods. *Journal of Chemical Information and Modeling*, 45(5):1376–1384, Aug 2005.
8. Andreas Maunz, Christoph Helma, and Stefan Kramer. Large-Scale Graph Mining Using Backbone Refinement Classes. In *KDD '09: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 617–626, New York, NY, USA, 2009. ACM.
9. Siegfried Nijssen and Joost N. Kok. A Quickstart in Frequent Structure Mining can make a Difference. In *KDD '04: Proceedings of the tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 647–652, New York, NY, USA, 2004. ACM.
10. Ulrich Rückert and Stefan Kramer. Optimizing Feature Sets for Structured Data. In *ECML '07: Proceedings of the 18th European Conference on Machine Learning*, pages 716–723, Warsaw, Poland, 2007. Springer-Verlag, Berlin-Heidelberg.
11. Xifeng Yan, Hong Cheng, Jiawei Han, and Philip S. Yu. Mining Significant Graph Patterns by Leap Search. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 433–444, New York, NY, USA, 2008. ACM.
12. Fumitaka Yoshida and John G. Topliss. QSAR Model for Drug Human Oral Bioavailability. *Journal of Medicinal Chemistry*, 43(13):2575–2585, Jun 2000.
13. Qiang Zhu, Xiaoyue Wang, Eamonn Keogh, and Sang-Hee Lee. Augmenting the Generalized Hough Transform to Enable the Mining of Petroglyphs. In *KDD '09: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1057–1066, New York, NY, USA, 2009. ACM.