Clustering Improves the Exploration of Graph Mining Results

Edgar H. de Graaf, Joost N. Kok, and Walter A. Kosters

Leiden Institute of Advanced Computer Science,
Leiden University, The Netherlands
edegraaf@liacs.nl

Abstract. Mining frequent subgraphs is an area of research where we have a given set of graphs, and where we search for (connected) subgraphs contained in many of these graphs. Each graph can be seen as a transaction, or as a molecule — as the techniques applied in this paper are used in (bio)chemical analysis.
In this work we will discuss an application that enables the user to further explore the results from a frequent subgraph mining algorithm. Such an algorithm gives the frequent subgraphs, also referred to as fragments, in the graphs in the dataset. Next to frequent subgraphs the algorithm also provides a lattice that models sub- and supergraph relations among the fragments, which can be explored with our application. The lattice can also be used to group fragments by means of clustering algorithms, and the user can easily browse from group to group. The application can also display only a selection of groups that occur in almost the same set of molecules, or on the contrary in different molecules. This allows one to see which patterns cover different or similar parts of the dataset.

1 Introduction

Mining frequent patterns is an important area of data mining where we discover substructures that occur often in (semi-)structured data. The research in this work will be in the area of frequent subgraph mining. These *frequent subgraphs* are connected vertex- and edge-labeled graphs that are subgraphs of a given set of graphs, traditionally also referred to as *transactions*, at least *minsupp* (a user-defined threshold) times. If a subgraph occurs at different positions in a graph, it is counted only once. The example of Figure 1 shows a graph and two of its subgraphs.

In this paper we will use results from frequent subgraph mining and we will present methods for improved exploration by means of clustering, where co-occurrences in the same transactions are used in the distance measure. Grouping patterns with clustering makes it possible to browse from one pattern and its corresponding group to another group close by. Or, depending on the preference of the user, to groups occurring in a separate part of the dataset.

Please use the following format when citing this chapter:

De Graaf, E. H., Kok, J. K., Kosters, W. A., 2007, in IFIP International Federation for Information Processing, Volume 247, Artificial Intelligence and Innovations 2007: From Theory to Applications, eds. Boukis, C., Pnevmatikakis, L., Polymenakos, L., (Boston: Springer), pp. 13-20.

Before explaining what is meant by lattice information we first need to discuss *child-parent* relations in frequent subgraphs, also known as patterns. Patterns are generated by extending smaller patterns with one extra edge. The smaller pattern can be called a *parent* of the bigger pattern that it is extended to. If we would draw all these relations, the drawing would be shaped like a lattice, hence we call this data *lattice information.*

We further analyze frequent subgraphs and their corresponding lattice information with different techniques in our framework LATTICE2SAR for mining and analyzing frequent subgraph data. One of the techniques in this framework is the analysis of graphs in which frequent subgraphs occur, via competitive neural networks as presented in [1]. Another important functionality is the browsing of lattice information from parent to child and from one group of fragments to another as presented here.

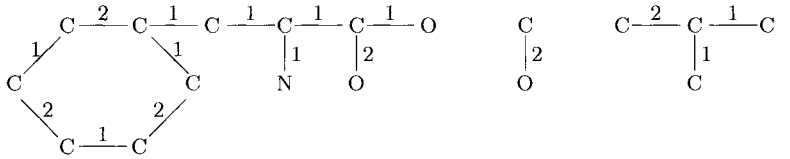

Fig. 1. An example of a possible graph (the amino acid Phenylalanine) in the molecule dataset and two of its many (connected) subgraphs, also called patterns or fragments.

Our application area is the analysis of fragments (patterns) in molecule data. The framework was originally made to handle (bio)chemical data. Obviously molecules are stored in the form of graphs, the molecules can be viewed as transactions (see Figure 1 for an example). However, the techniques presented here are not particular to molecule data (we will also not discuss any chemical or biological issues). For example one can extract user behavior from access logs of a website. This behavior can be stored in the form of graphs and can as such be analyzed with the techniques presented here.

The distance between patterns can be measured by calculating in how many graphs (or molecules) only one of the two patterns occurs. If this never happens then these patterns are very close to each other. If this is always the case, their distance is very large. In both cases the user is interested to know the reason. In our application the chemist might want to know which different patterns seem to occur in the same subgroup of effective medicines or on the other hand which patterns occur in different subgroups of effective medicines. In this paper we will present an approach to solve this problem that uses clustering. Furthermore all occurrences for the frequent subgraphs will be discovered by a graph mining algorithm and this occurrence information will be highly compressed before storage. Because of this, requesting these occurrences will be costly.

We will define our techniques for browsing the lattice of fragments. To this end, this paper makes the following contributions:

— An **application** will be introduced that **integrates techniques that facilitate browsing** of the lattice as provided by the frequent subgraph miner

(Section 2).
— We will use a **distance measure based on the co-occurrence of fragments to browse** from one fragment group to another (Section 3 and Section 4).
— We will give an **algorithm for grouping** very similar subgraphs using hierarchical cluster methods and lattice information (Section 4).
— Finally through experiments we will take a **closer look at runtime performance** of the grouping algorithm and discuss it (Section 5).

The algorithm for grouping was also used in [1], both papers discuss a component of the same framework. However in this work groups are used differently, for fragment suggestion during browsing.

This research is related to research on clustering, in particular of molecules. Also our work is related to frequent subgraph mining and frequent pattern mining when lattices are discussed. In [8] Zaki et al. discuss different ways for searching through the lattice and they propose the ECLAT algorithm.

Clustering in the area of biology is important because of the improved overview it provides the user with. E.g., [4] Samsonova et al. discuss the use of Self-Organizing Maps (SOMs) for clustering protein data. SOMs have been used in a biological context many times, for example in [2, 3]. There is also a relation with work done on hierarchical clustering in the biological context, e.g., as presented in [5]. In some cases molecules are clustered via numeric data describing each molecule; in [6] clustering such data is investigated.

Our package of mining techniques for molecules makes use of a graph miner called GSPAN, introduced in [7] by Yan and Han. This implementation generates the patterns organized as a lattice and a separate compressed file of occurrences of the patterns in the graph set (molecules).

2 Exploring the Lattice

We propose a *fragment exploration tool* to explore fragments in a dataset of molecules, the whole process is visualized in Figure 2. The application requires both fragment and lattice information from the frequent subgraph miner. This information is already extracted from the dataset when the application starts. All this data is first read and an in-memory lattice structure is built, where each node is a fragment. Occurrences are kept in a compressed format since the user wants to view this data when required. Also this data is needed by our distance measure which will be explained in Section 3; to make a distance matrix for all fragments will probably cost too much memory. First we make groups using information from the lattice only. Then we fill a matrix storing the distances between groups, which is possible if we assume to have far less groups of similar fragments.

After this process it is possible for the user to browse from fragment to fragment by adding or removing possible edges, where an edge is possible if it leads to a child or parent fragment. Figure 3 shows the current fragment in the

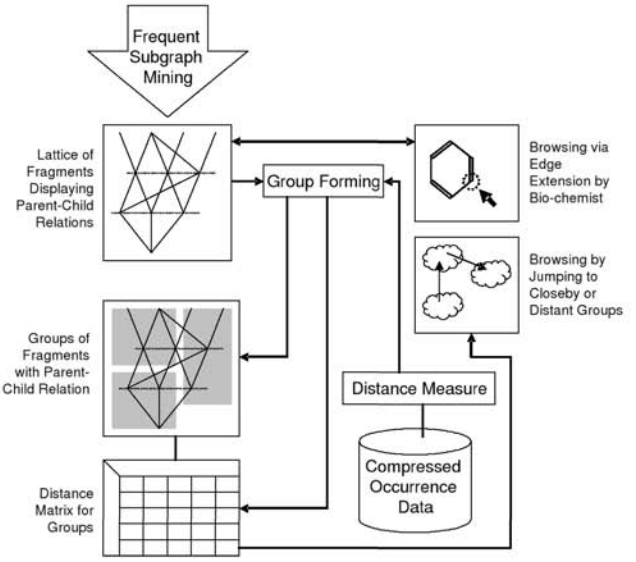

Fig. 2. The process of exploring the fragment lattice.

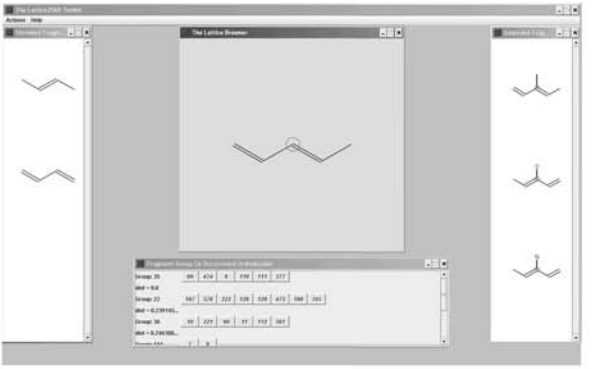

Fig. 3. Fragment exploration with the possible ways of shrinking and extending.

center window. The user must select a molecule to which an edge should be added. After an edge is selected one can select a possible extension, leading to a child, from the right window. It is also possible to shrink the current fragment towards a parent fragment, the possibilities are always shown in the left window.

The user can also jump to a fragment in a group that occurs either often in the same molecules or almost never, so fragments in close by or distant

groups. Each molecule has a group and in Figure 4 it shows its group, and the other fragments in that group, first. Then it lists all close by groups and their corresponding fragments (here close by is defined as $group\_dist \leq 0.3$, see also Section 4). For every group it shows the distance, indicated with "dist", to the group of the current fragment.

Fig. 4. Co-occurrence view for groups, showing all groups close by ($group\_dist \leq 0.3$).

3 Distance Measure

The distance measure will compute how often frequent subgraphs occur in the same graphs of the dataset. In the case of our working example it will show if different fragments (frequent subgraphs) exist in the same molecules. Formally we will define the distance measure in the following way (for graphs g_1 and g_2):

$$dist(g_1, g_2) = \frac{sup(g_1) + sup(g_2) - 2 \cdot sup(g_1 \wedge g_2)}{sup(g_1 \vee g_2)} = \frac{sup(g_1) + sup(g_2) - 2 \cdot sup(g_1 \wedge g_2)}{sup(g_1) + sup(g_2) - sup(g_1 \wedge g_2)} \quad (1)$$

Here $sup(g)$ is the number of times a (sub)graph g occurs in the set of graphs; $sup(g_1 \wedge g_2)$ gives the number of graphs (or transactions) with both subgraphs and $sup(g_1 \vee g_2)$ gives the number of graphs with at least one of these subgraphs. The numerator of the *dist* measure computes the number of times the two graphs do not occur together in one graph of the dataset. We divide by $sup(g_1 \vee g_2)$ to make the distance independent from the total occurrence, thereby normalizing it. By reformulating we remove $sup(g_1 \vee g_2)$, saving us access time for the compressed dataset.

The distance measure satisfies the usual requirements, such as the triangular inequality. Note that $0 \leq dist(g_1, g_2) \leq 1$ and $dist(g_1, g_2) = 1 \Leftrightarrow sup(g_1 \wedge g_2) = 0$, so g_1 and g_2 have no common transactions in this case. If $dist(g_1, g_2) = 0$, both subgraphs occur in the same transactions, but are not necessarily equal.

While computing the support for the graphs not all frequent subgraphs are known and not all distances can be computed while running gSpan.

4 Grouping Fragments

We will have to store the distance for all frequent subgraph combinations in order to decide fragments at an interesting distance. If we have n frequent

subgraphs then storing the support for all $n(n-1)/2$ combinations might be too much. However many frequent subgraphs often are very similar in both structure and support and often there exists a parent-child relation.

Now we will propose a step where we group close subgraphs to reduce both the number of distances to store and the exploration time by grouping redundant graphs. We first define a distance *grdist* (C_1, C_2) between groups (clusters) C_1 and C_2 as the maximal *dist* between parent and child graphs in the two groups. This can be calculated fast by traversing the lattice.

This distance has a special value -1 if there is no pair (g_1, g_2) with $g_1 \in C_1$ and $g_2 \in C_2$, such that they have a parent-child relation, otherwise the maximum *dist* between such elements is used.

All information used to compute these distances can be retrieved from the lattice information provided by the graph mining algorithm, when we focus on the subgraph-supergraph pairs. This information is already there to discover the frequent subgraphs, the only extra calculation is done when searching for *dist* in this information.

Now we propose the GROUPFRAGMENTS algorithm that will organize close subgraphs/supergraphs into groups. The groups will be organized in a set $\mathcal{P}$. The outline of our algorithm based on hierarchical clustering is the following:

---

initialize $\mathcal{P}$ with sets of subgraphs of size 1 from the lattice
while $\mathcal{P}$ was changed or was initialized
  Select C_1 and C_2 from $\mathcal{P}$ with minimal *grdist* $(C_1, C_2) \geq 0$
  if $grdist(C_1, C_2) \leq maxdist$ **then**
    $\mathcal{P} = \mathcal{P} \cup \{C_1 \cup C_2\}$
    Remove C_1 and C_2 from $\mathcal{P}$

GROUPFRAGMENTS

The parameter *maxdist* is a user-defined threshold giving the largest distance allowed for two clusters to be joined.

Once the clusering has been done, we redefine the distance between groups as the distance between a smallest graph of each of the two groups, representing the most essential substructure of the group (*size* gives the number of vertices): for $g_1 \in C_1$ and $g_2 \in C_2$ with $size(g_1) = min(\{size(g) \mid g \in C_1\})$ and $size(g_2) = min(\{size(g) \mid g \in C_2\})$, we let $group\_dist(C_1, C_2) = dist(g_1, g_2)$. So even if *grdist* (C_1, C_2) would give the special value -1, $group\_dist(C_1, C_2)$ will provide a reasonable distance.

Now we allow the user to define which groups are interesting. These are mostly extremes: close by or far away groups. So the set $\mathcal{P}'$ of interesting groups with a relation to group C_w will be: $\mathcal{P}' = \{C_v | group\_dist(C_w, C_v) \leq interest\_min \vee group\_dist(C_w, C_v) \geq interest\_max\}$, where *interest_max* defines the largest distance of interest and *interest_min* the smallest. The user can now browse fragments in these interesting groups.

5 Experimental Results

The experiments were done for three main reasons. First of all we want to show the development of *runtime performance as maxdist decreases*. Secondly we want to show *the effect of fragment size* on the grouping algorithm with the distance measure. Finally the *effect of using a distance matrix* for storing distances between groups will be measured.

We make use of a molecule dataset, containing 4,069 molecules; from this we extracted a lattice with the 1,229 most frequent subgraphs. All experiments were performed on an Intel Pentium 4 64-bits 3.2 GHz machine with 3 GB memory. As operating system Debian Linux 64-bits was used with kernel 2.6.8-12-em64t-p4.

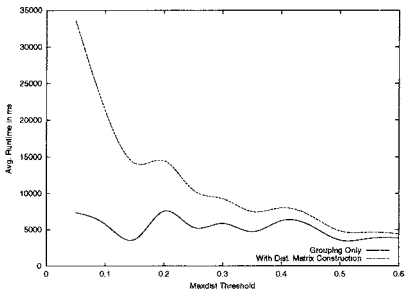

Fig. 5. Runtime in ms for different *maxdist* settings and the influence of distance matrix construction.

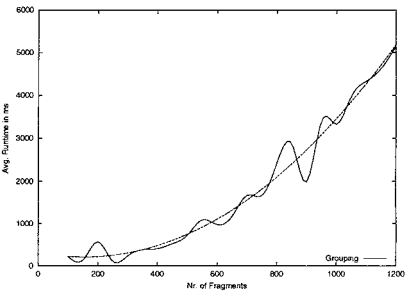

Fig. 6. Runtime in ms for different fragment set sizes (*maxdist* = 0.3), with quadratic regression

Figure 5 shows how runtime drops if we increase the *maxdist* threshold. This is mainly caused by the decrease of groups and so the size of the distance matrix. However the use of a distance matrix will provide the necessary speedup during exploration. Furthermore we also see that a low *maxdist* gives a large runtime due to a large distance matrix. This seems to show that making groups enables the application to store a distance matrix in memory and this allows the application to faster find close by groups (so faster browsing). Note that in practice we should store the distance matrix, for each dataset, on the disk and construct it only once.

In Figure 6 we see the runtime for the grouping algorithm as the number of fragments to be grouped increases. This runtime depends on the distance measure and the grouping algorithm, and runtime seems to increase polynomially.

6 Conclusions and Future Work

The application discussed in this work facilitates the exploration of fragments extracted from a dataset of molecules. With fragments we mean frequent subgraphs occurring in a dataset of graphs, the molecules.

We introduced two methods of browsing fragments. Firstly one can browse between parent and child by adding or removing edges from the fragments (only if it leads to another existing fragment). Our second method of browsing required us to first group fragments into groups of very similar fragments. We consider a fragment to be similar to another one if they have a parent-child relation and they occur in (almost) the same molecules. This allows the (bio)chemist to quickly jump to fragments that are biologically more interesting or cover a different subgroup of molecules.

Finally we discussed the runtime performance of our fragment grouping algorithm with different settings. Results showed that the construction of a distance matrix, needed for fast browsing, takes the most time. Furthermore results suggested that grouping improves the runtime, since less (redundant) distances are stored.

In the future we hope to include other innovative ways of browsing and analyzing the lattice of fragments, and we want to improve scalability where possible.

Acknowledgments: This research is carried out within the Netherlands Organization for Scientific Research (NWO) MISTA Project (grant no. 612.066.304). We thank Jeroen Kazius and Siegfried Nijssen for their help.

References

1. Graaf, E.H. de, Kok, J.N. and Kosters, W.A.: *Visualization and Grouping of Graph Patterns in Molecular Databases*, Submitted.
2. Hanke, J., Beckmann, G., Bork, P. and Reich, J.G.: *Self-Organizing Hierarchic Networks for Pattern Recognition in Protein Sequences*, Protein Science Journal 5 (1996), pp. 72–82.
3. Mahony, S., Hendrix, D., Smith, T.J. and Golden, A.: *Self-Organizing Maps of Position Weight Matrices for Motif Discovery in Biological Sequences*, Artificial Intelligence Review Journal 24 (2005), pp. 397–413.
4. Samsonova, E.V., Bäck, T., Kok, J.N. and IJzerman, A.P.: *Reliable Hierarchical Clustering with the Self-Organizing Map*, in Proc. 6th International Symposium on Intelligent Data Analysis (IDA 2005), LNCS 2810, pp. 385–396.
5. Uchiyama, I.: *Hierarchical Clustering Algorithm for Comprehensive Orthologous-Domain Classification in Multiple Genomes*, Nucleic Acids Research Vol. 34, No. 2 (2006), pp. 647–658.
6. Xu, J., Zhang, Q. and Shih, C.K.: *V-Cluster Algorithm: A New Algorithm for Clustering Molecules Based Upon Numeric Data*, Molecular Diversity 10 (2006), pp. 463–478.
7. Yan, X. and Han, J.: *gSpan: Graph-Based Substructure Pattern Mining.* In Proc. 2002 IEEE International Conference on Data Mining (ICDM 2002), pp. 721–724.
8. Zaki, M., Parthasarathy, S., Ogihara, M. and Li, W.: *New Algorithms for Fast Discovery of Association Rules*, in Proc. 3rd International Conference on Knowledge Discovery and Data Mining (KDD 1997), pp. 283–296.