Mining semi-structured data, theoretical and experimental aspects of pattern evaluation

Graaf, E.H. de

Mining Semi-Structured Data

Theoretical and Experimental Aspects of Pattern Evaluation

E.H. de Graaf

Mining Semi-Structured Data

Theoretical and Experimental Aspects of Pattern Evaluation

proefschrift

ter verkrijging van
de graad van Doctor aan de Universiteit Leiden
op gezag van de Rector Magnificus prof. mr. P. F. van der Heijden,
volgens besluit van het College voor Promoties
te verdedigen op woensdag 29 oktober 2008
klokke 13:45

door

Edgar Hubert de Graaf
geboren te Schagen
in 1979

Promotiecommissie

Promotor:	Prof. Dr. J.N. Kok
Co-promotor:	Dr. W.A. Kosters
Referent:	Dr. J.M. Peña (Universidad Politécnica de Madrid)
Overige leden:	Prof. Dr. T.H.W. Bäck
	Prof. Dr. F.S. de Boer
	Prof. Dr. G. Rozenberg

This research was financed by the Netherlands Organisation for Scientific Research (NWO) in the framework of project MISTA, grant no. 612.066.304.

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming Research and Algorithmics).

Contents

1 Introduction **1**
- 1.1 Data Mining . . . 1
- 1.2 Structured and Unstructured Data . . . 5
- 1.3 Semi-Structured Data . . . 5
  - 1.3.1 Itemsets and Sequences . . . 6
  - 1.3.2 Trees and Graphs . . . 7
- 1.4 Overview of this Thesis . . . 8
- 1.5 Overview of Publications . . . 10

2 Mining with Consecutiveness **13**
- 2.1 Introduction . . . 13
- 2.2 Consecutive Support . . . 15
- 2.3 Pruning Methods . . . 19
  - 2.3.1 Parent Support Recalculation . . . 19
  - 2.3.2 Introducing α . . . 21
  - 2.3.3 Exact Depth . . . 21
  - 2.3.4 Hyperclique Patterns and h-confidence . . . 22
- 2.4 Results and Performance . . . 23
  - 2.4.1 Consecutive Support . . . 25
  - 2.4.2 Selection of ρ and σ . . . 30
  - 2.4.3 Combination with h-confidence . . . 32
- 2.5 Conclusions . . . 35

3 Patterns with a Fixed Interval In Between **37**
- 3.1 Introduction . . . 37
- 3.2 Stable Patterns . . . 38
- 3.3 Definition . . . 39
- 3.4 Algorithm . . . 41
- 3.5 Results and Performance . . . 43
- 3.6 Conclusions . . . 46

4 Mining Balanced Patterns 47
4.1 Introduction . . . 47
4.2 Definition . . . 48
4.3 Algorithm . . . 50
4.4 Results and Performance . . . 52
4.5 Conclusions . . . 57

5 The Most Discriminating Patterns and Domain Knowledge 59
5.1 Introduction . . . 59
5.2 The Maximal Discriminating Patterns . . . 61
5.3 Algorithm without Domain Knowledge . . . 64
5.4 Domain Specific Improvements . . . 65
5.5 Experimental Results . . . 69
5.6 Conclusions . . . 72

6 Visualization of Graph Patterns 73
6.1 Introduction . . . 73
6.2 Distance Measure . . . 76
6.3 Optimization: Only Frequent Subgraphs and Grouping . . . 77
6.4 Visualization . . . 79
6.5 Performance . . . 79
6.6 Conclusions and Future Work . . . 85

7 Improved Exploration of Graph Mining Results 87
7.1 Introduction . . . 87
7.2 Exploring the Lattice . . . 90
7.3 Distance Measure . . . 91
7.4 Grouping Fragments . . . 92
7.5 Experimental Results . . . 94
7.6 Conclusions . . . 95

8 Displaying Graph Pattern Co-Occurrence in Streams 97
8.1 Introduction . . . 97
8.2 Related Work . . . 98
8.3 Model Realization . . . 99
8.3.1 Support . . . 100
8.3.2 Distance . . . 101
8.3.3 Merge and Split . . . 102
8.3.4 The Algorithm . . . 104
8.4 Experiments and Discussion . . . 105

8.5 Conclusions and Future Work 109

9 Mining Web Access Data and the Interpretation of Patterns 113
9.1 The Itemset View 114
9.2 Mining with Sequences 118
9.2.1 Sequential Patterns 119
9.3 Co-Occurring Subgraph 121
9.4 Conclusions 126

Conclusions 126

A A measure of "surprisingness" 131
A.1 Root Mean Square Deviation 131
A.2 Pruning with RMSD 134

Bibliography 135

Nederlandse Samenvatting 145

Acknowledgements 147

Curriculum Vitae 149

1 Introduction

With the arrival of the internet and the rise of bioinformatics the analysis of data is more and more faced with data that is loosely structured. Examples of such loosely structured data are molecule data or XML based web pages. When data has a less rigid structure it becomes more difficult to discover interesting patterns. There are more potential patterns (candidates) since there are many ways of combining the parts. For example, in bio-chemistry one analyses molecules for interesting patterns (e.g., many molecules have three carbon molecules connected with a single bond). However, one molecule can potentially have many different atoms and a number of connection types between several of these atoms.

In this chapter a general introduction is provided about principles that are needed to better understand this thesis. First data mining will be explained. Data mining can be done on data with different level of structure; this thesis will treat the data mining of semi-structured data. We discuss different kinds of semi-structured data. Finally, we give an overview of the thesis together with a list of publications on which the thesis is based.

1.1 Data Mining

Data mining can be seen as the analysis of large quantities of data in an attempt to discover new patterns and relations in an automated fashion. There are many books written about data mining, e.g., [32, 64, 71]. Each of them gives a definition of data mining:

- In [64] Tan, Steinbach and Kumar define it as: "Data mining is a technology that blends traditional data analysis methods with sophis-

ticated algorithms for processing large volumes of data ...".

- In [71] Witten and Frank define data mining as: "The process of discovering patterns in data. The process must be automatic or (more usually) semi-automatic. The patterns discovered must be meaningful in that they lead to some advantage, usually an economic advantage. The data is invariably present in substantial quantities ...".

- In [32] Hand, Mannila and Smyth give the following definition: "Data mining is the analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner ...".

We prefer the third definition. In [32] they continue: "The relationships and summaries derived through a data mining exercise are often referred to as models or patterns. Examples include linear equations, rules, clusters, graphs, tree structures, and recurrent patterns in time series ...". In this thesis the patterns resulting from data mining are sets, sequences and graphs and the models are rules and clusters.

The data to be mined can have different levels of structure. Some data does not really have any defined structure, like text or video (unstructured data), while other sources of data are completely and rigidly defined (structured data). In this thesis we will focus on data that is between these extremes. Data will have a defined structure, however each record may have a different size or might not contain the same parts as the other records.

There is a broad range of techniques used in data mining. The most important data mining tasks used in the thesis are frequent pattern mining, competitive neural networks and clustering. We next discuss these three data mining tasks.

In *frequent pattern mining* we discover certain re-occurring substructures that occur at least *minsupp* times, where *minsupp* is the minimal frequency threshold (minimal support). The search space is potentially huge and frequent pattern mining is usually involved in pruning the search space. Commonly this is done by using anti-monotonicity, i.e., the occurrence count of a pattern is never more than the occurrence of sub-pattern (= all elements are also in the pattern, but all elements of the pattern are not necessarily in the sub-pattern).

The competitive neural network used in this thesis is a *dimensional reduction method*. Dimensions in this context basically mean the number of

values of one record. More than three dimensions are hard to visualize, since a human can only effectively understand a picture of up to three dimensions. For these reason we use a number of techniques that Tan, Steinbach and Kumar in [64] define as: "find a projection of the data to a lower-dimensional space that perserves pairwise distances as well as possible, as measured by an objective function ...". A technique is called a dimensional reduction method if it approaches the distances between instances as good as possible with (far) less dimensions. In this way the visualized data can be made understandable to the analyst with one picture.

The neurons or centroids in a *competitive neural network* compete for being selected, given an input vector. E.g., assume that we have a data set where each record has 100 values (= 100 dimensions) with a number ranging from 0 to 10 indicating a grade one person gave to 100 different movies. It is very hard for a human to quickly see groups of people that like the same movie. However, if we reduce the number of dimensions to two, two numerical values, then we can easily make a picture and the user can see groups and their approximate distance in one view. One of the most commonly used algorithms was proposed by Kohonen in [38]; this algorithm is called the *self-organizing map*. In this thesis we will use an other algorithm: the "push and pull"-algorithm as proposed in [39].

The "push and pull"-algorithm places instances in a two-dimensional or three-dimensional space adhering to the given distance between them. Basically each instance is represented by a neuron. In the case of a Kohonen network, neurons are initialized with a random weight vector equal in length to the input vector. At each iteration the neuron with the smallest Euclidean distance to the input vector is selected and they are place a little bit closer to this input. This enables us to make different kinds of visualizations, a common visualization is displayed in Figure 1.1. Each hexagon is one neuron and each side of a hexagon is colored dark if the weight vectors of neighbouring neurons are relatively distant from each other and light otherwise. Each hexagon is colored dark if the average relative distance is large and light if it is low.

The difference between the self-organizing maps, like those from the Kohonen algorithm, and the "push and pull"-algorithm is that in the former each instance is prelinked to a neuron which is moved to an optimal position using the pair-wise relative distance. For the "push and pull"-algorithm we do not need to know the input vectors.

Self-organizing maps, on the other hand, have neurons with a weight vector. These weight vectors have equal dimensions for the input vectors. For each input vector provided to the self-organizing map the best-matching

Figure 1.1: A common way of visualizing Kohonen networks.

unit (neuron) and its neighbours are pulled closer to this input. In the case of the "push and pull"-algorithm we link each neuron with an instance in the dataset. Two neurons are selected and their two-dimensional co-ordinates are slightly adjusted towards their target distance. The main difference between self-organizing maps and the "push and pull"-algorithm is this different way of using the neurons.

The advantage in comparison to SOM (and other more traditional methods) is that we can stop the iterations of the "push and pull"-algorithm at any time and immediately get a valid approximated two-dimensional (or three-dimensional) picture of the distances between all neurons when we only know the pair-wise relative distance for the instances.

Neurons that have a low Euclidean distance are put close together in the two-dimensional space and neurons with a big distance are placed far apart. Furthermore, because each instance (or multiple instances) is prelinked with a neuron you can quickly view the relative distances. In the case of self-organizing maps you will first need to locate the neuron closest to each instance and then calculate the distance between these neurons or interpret the color image. Further processing of the neurons is needed to get a two-dimensional picture of the relative distances.

The disadvantage of the "push and pull"-algorithm is that many instances will give many neurons and this either slows the algorithm down or makes the approximation less accurate. A solution would be to have some way to quickly prelink multiple instances to one neuron.

Clustering is discussed in many sources, e.g., Hopgood in [34] defines clustering as a form of unsupervised learning, i.e., the training examples consist of input vectors without the desired output. As successive input vectors are presented, they are clustered into N groups, where the integer N may be prespecified or may be allowed to grow according to the diversity of

the data. The competitive neural networks used in this thesis are primarily used as a dimension reduction and visualization method. However, as points are pulled closer, they form clusters. The difference with traditional clustering methods is that the user sees the picture and indicates the clusters. The traditional methods automatically make the clusters.

1.2 Structured and Unstructured Data

Some data sources impose a rigid structure on the data they contain and others impose no structure at all. First these extremes need to be explained before discussing semi-structured data. Structured data has the following properties:

- Data is organized into transactions.
- Similar transactions are grouped together, e.g., in tables.
- These groups all have the same attributes.
- The value of an attribute is always of a certain type and length.

Unstructured data is the opposite of structured, making mining of unstructured data difficult. Often it also difficult to explain mining results when mining unstructured data. Examples of unstructured data are plain text, video, sound and images and they share the following properties:

- Data can be of any type.
- There are no transactions, we have no information about the meaning of attributes; there is no specific format.
- The structure doesn't follow any rules, e.g., for a video there will be no rule disallowing certain colors in certain places.
- The value of an attribute have no specific type or length.

1.3 Semi-Structured Data

Semi-structured data arises when the source or the environment does not impose a rigid structure on the data and when data is combined from several heterogeneous sources [59]. E.g., bibliographic data where some books are written by one author and others by two or more. For some of these

authors only their name is known and for others their age or their specialty. Also for describing a specific paper different fields are needed than for a novel. Another example of semi-structured data is website browsing data, where the browsing behaviour of a user is one record in a dataset of graphs. Each hyperlink click a user makes is represented by one node in this graph. Some nodes might have extra attributes giving extra information about the webpage. The connections between the nodes indicate the hyperlinks a user chooses and towards which page (node) that link pointed. So also in this data there is structure, but it is not completely rigid. Semi-structured data has the following properties:

- Records do not necessarily have the same number of fields and fields can be different.
- Records or parts of a record describing a similar principle, e.g., a molecule or an atom, can be grouped together.
- Fields do not have to be in a specific order.

This thesis will discuss the evaluation of pattern occurrence in semi-structured data in the form of itemsets, sequences, trees and graphs.

1.3.1 Itemsets and Sequences

The simplest form of semi-structured data we will treat in this thesis are item sets. In the case of item sets one has a data set where each transaction is a set of items. The most common way of mining this data is *frequent subset mining* where we search for groups or sets of items that can be found in at least a user-defined number of transactions:

Definition 1.3.1 (Frequent Subsets) *Assume we have a data set $\mathcal{D}$ of sets where each transaction is a finite set I of items. A set S is a* frequent subset *for $\mathcal{D}$ if for minsupp transactions I it holds that $S \subseteq I$. The* minimal support threshold *minsupp is a pre-defined threshold. Note that $\mathcal{D}$ itself is considered as a sequence of transactions.*

Sequences are a similar type of semi-structured data, but they differ in that the order of items is important. Ten times an item `A` is different than one and a sequence of first an item `A` and then `B` is different from first an item `B` and then `A`. In this thesis we will discuss sequences in the context of *frequent sequence mining*, where sequences are frequent if they occur in at least *minsupp* sequences:

Definition 1.3.2 (Frequent Subsequences) *A sequence* $d = (d_1, d_2, \ldots, d_m)$ *is called a* super-sequence *of a sequence* $s = (s_1, s_2, \ldots, s_k)$ *if* $k \leq m$ *and for each* s_i $(1 \leq i \leq k)$ *there is a* d_{j_i} $(1 \leq j_i \leq m)$ *with* $s_i = d_{j_i}$ *and* $j_{i-1} < j_i$ $(i > 1)$*. We denote this with* $s \prec d$*. The sequence* s *is called a* sub-sequence *of* d*.*

The sequence s *is called frequent if it is a subsequence of at least minsupp sequences in the dataset* $\mathcal{D}$ *of sequences.*

1.3.2 Trees and Graphs

Both graphs and trees consist of vertices where some are connected with edges. When edges of a graph have a direction they are called *directed*, and *undirected* otherwise. We call a graph *connected* if all vertices can be reached from all other vertices by following the edges. All graphs in this thesis are undirected connected. Furthermore a graph is called *fully connected* if between all vertices their is an edge connecting them.

A tree is basically a graph without cycles. This means that edges are such that when we start following edges from a vertex, we can not end up at the same vertex. Also trees usually have a root and leaves. The *root* is the designated vertex where we can start and follow the edges until we end up at one of the last vertices, a *leaf*.

A typical example of a graph is a molecule that consists of atoms and these atoms are connected with bonds. Frequent subgraphs (see Definition 1.3.3) are harder to find in comparison with frequent subtrees because of possible cycles. This can be explained with Figure 1.2. Say one wants to see if `subgraph 1` is a subgraph of the molecule `Ribose` (where no character means the vertex is Carbon, `C`). We need to match all atoms and edges from the subgraph with the atoms and edges of the `Ribose` and this can require many attemps depending on the atom and edge that you match first.

Definition 1.3.3 (Frequent Subgraph) *Let* $G = (V, E)$ *and* $G' = (V', E')$ *be connected graphs, where* V *and* V' *are finite, non-empty sets of vertices and* E *and* E' *are non-empty sets of edges (links between pairs of vertices). The graph* G' *is a* subgraph *of* G *if* $V' \subset V$ *and* $E' \subset E$*.*

If G' *is a subgraph of at least minsupp graphs* G *in a dataset* $\mathcal{D}$ *of graphs then we call* G' *a* frequent subgraph*.*

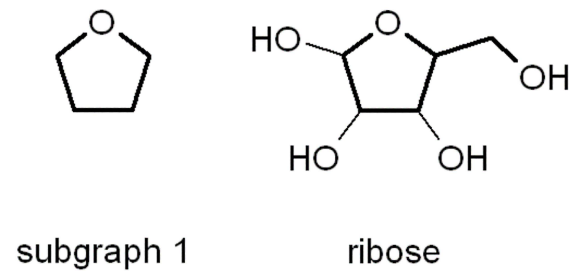

Figure 1.2: An example molecule and two subgraphs

1.4 Overview of this Thesis

In this thesis we investigate ways of mining (different types of) semi-structured data. We look how the occurrence of patterns in the data can be evaluated in different ways in order to find interesting patterns.

Representing results to the user is also important and the visualization of results from mining semi-structured data enable the user to see the patterns that are of interest to him or her.

The need for mining semi-structured data comes from the growing number of sources of semi-structured data; due to the growth of internet and advances in bio-informatics. E.g., in the case of internet we want to know how users make use of a website and how we can improve usability. Biological structures are often complex and many can be described using semi-structured data, e.g., sequences to describe protein sequences or graphs describing molecules.

Next we give an overview of the rest of the thesis.

Chapter 2 will discuss the discovery of consecutive occurring patterns. The chapter will start by defining *consecutive support*. *Consecutiveness* in this thesis is defined as the number of occurrences of patterns where we take into account the distance between transactions where the pattern occurred. With distance is meant the number of in-between transactions (each taking equal time and there are no time gaps between them) that did not contain the pattern or the amount of time between timestamps. Of course, this only makes sense if the transactions are given in some logical order.

Chapter 2 proceeds to discuss how we can prune the search space such that we can have a lower minimal consecutiveness and still get results reasonably quickly. Discovering consecutive occurring patterns can be combined

with principles used before in combination with traditional support to find interesting patterns. We show how we can combine these principles with consecutiveness to discover new patterns. We apply consecutive support on biological dataset containing anonymous patients and their deviation from normal for several spots, clones, within the chromosomes. For all these patients, with the same illness, we search the most common sets of deviating clones. Out of this we construct a picture showing which areas are most commonly deviating from normal.

Related is Chapter 3: in this chapter the data about occurrences is used to discover *stable* and *balanced patterns* occurring many times and with similar time between intervals. The basic approach for stable patterns is to take all pairs of occurrences and see if another occurrence (approximately) occurs half-way. If this happens many times, the pattern has often similar time between intervals. The approach for balanced patterns prunes the search space for patterns by deciding the number of occurrences of all distances (up to a certain maximal distance) between all pairs and prune them if this is lower than a user-defined threshold.

The pruning threshold for balanced patterns is more intuitive to the data analysist. The advantage of balanced patterns is that its parameters are easier to estimate. However its disadvantage, in comparison to stable patterns, is that there has to be a maximal distance for which we keep a count. One pattern discovered, during the experiments, was that one user of `portalexecutivo.com` visited the research and training part every seven days during one month.

Chapter 4 is about faster discovery of patterns when we know approximately where patterns are located within a sequence. This domain knowledge can be used when we search for the *most discriminating patterns*, patterns occuring many times in one set of sequences and not in another. The main application area for this technique is in the area of protein sequence analysis. Two different groups of protein sequences are assumed to be most different in the helix areas. The proposed speed-up of Chapter 4 was found to depend mainly on how exact one can estimate the position (probable time window) of the patterns within the (transaction) sequences. Furthermore it became clear that the speed up also depends on the discriminative power of the patterns, e.g., if we search for the ten best patterns and these patterns are very discriminative then we can skip the counting of many other patterns. This is because we can calculate their occurrence to be such that they will never be more discriminative.

In Chapter 5, Chapter 6 and Chapter 7 the visualization of patterns for different data is discussed.

In Chapter 5 a model is built that visualizes co-occurring patterns. In this way the user can quickly see which patterns often occur in the same transactions and which almost only occur in different ones. The grouping of patterns, as done in this chapter, improves the runtime and the readability of the co-occurrence model.

In Chapter 6 an application is presented where the user can browse the results of a frequent subgraph mining algorithm in different ways. First of all one can start with one graph and extend it or shrink it to another depending on it being frequent or not. E.g., the biologist can start with one interesting pattern. They add or remove parts and make the pattern biologically more relevant (and only a little bit less occurrence). For the second way of browsing the algorithm first constructs groups of patterns that are structural related and co-occur a lot. Then one can browse from one group to the other quickly view the more interesting co-occurring groups of structural unrelated patterns.

Finally in Chapter 7 a method of visual modelling co-occuring patterns in streams is presented. Streams are potentially infinite, making an approach that first mines for frequent patterns impossible. Our approach builds a model of patterns in a stream by shrinking and growing patterns based on their frequency and approximated co-occurrence (distance).

In Chapter 8 we apply our techniques on real data in scenarios based (weblogs). It becomes clear that not all techniques are equaly effective in the weblog scenario. E.g., discriminating pattern mining techniques give a large number of interesting patterns. We also use a measure such that we can discover patterns that are proportionally different. This allows us to find patterns that are perhaps relatively small in their occurrence, but suprising in their occurrence within the different groups (i.e., the pattern occurs much more in one group). Finally we discuss how one can prune the search space when this measure is used.

Co-occurrence modelling is less effective in this setting because most sequences (of users going from page to page) are too short. For most of these short sequences there are not many co-occurrening patterns.

1.5 Overview of Publications

Parts of this thesis are published in the form of papers. Next we give an overview of the papers on which the different chapters are based.

Chapter 2: Mining with Consecutiveness

A large part of this chapter is based on a paper in the Proceedings of the ECML/PKDD Workshop on Data and Text Mining for Integrative Biology (BIOWS'06) [29].

Chapter 3: Patterns with a Fixed Interval In Between
The content of this chapter is based on the paper published in the Proceedings of the 18th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC'06) [25].

Chapter 4: Mining Balanced Patterns
The issues in this chapter were published in the Proceedings of the International Conference on Artificial Intelligence and Applications (AIA'08) [30].

Chapter 5: The Most Discriminating Patterns and Domain Knowledge
Parts of this chapter are published in the Proceeding of the 17th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC'05) [28] and in the Proceedings of 3rd International ECML/PKDD Workshop on Mining Graphs, Trees and Sequences (MGTS'05) [27].

Chapter 6: Visualization of Graph Patterns
The content of this chapter is based on the paper published in the Proceedings of the 27th SGAI International Conference on Artificial Intelligence (AI'07) [24].

Chapter 7: Improved Exploration of Graph Mining Results
This chapter is largely based on the work as published in the Proceeding of the 4th IFIP Conference on Artificial Intelligence Applications & Innovations (AIAI'07) [23].

Chapter 8: Displaying Graph Pattern Co-Occurrence in Streams
The work in this chapter was published in the Proceedings of the 19th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC'07) [26].

2 Mining with Consecutiveness

We propose a new measure of support (the number of occurrences of a pattern) in which instances are more important if they occur with a certain frequency and close after each other in the stream of transactions. We will explain this so-called consecutive support and discuss how patterns can be found faster by pruning the search space, for instance by using "parent support recalculation". Both consecutiveness and the notion of hypercliques are incorporated simultaneously into the ECLAT algorithm. This combination results into new patterns.

Examples using artificial datasets show how interesting phenomena can be discovered. The new measure of support can be applied in many areas, ranging from bio-informatics to trade, retail, and even law enforcement. E.g., in bio-informatics it is important to find patterns contained in many individuals, where patterns close together in one chromosome are more significant.

2.1 Introduction

In earlier research de Graaf et al. explored the use of frequent itemsets to visualize deviations in chromosome data concerning people with a certain illness, genomic profiling [31]. During this exploration of the problem it became apparent that patterns are more important when the areas (transactions) in which they occur are close together. The consecutiveness of transactions containing the pattern plays an important role in certain applications. Here patterns are frequent sets of items where frequent means that their support, that can be defined in different ways, is larger than the *minsup* threshold. In a biological problem setting the items can be individuals and the trans-

actions can be "clones", pieces of the chromosome that might occur more or less often than in a healthy individual. Patterns in close transactions are better because they are close together in the chromosome and are biologically more significant than patterns that are far apart and are in different chromosomes.

Consecutive support informally is the support or the number of occurrences of patterns where we take into account the distance between transactions where the pattern occurred. With distance is meant the number of in-between transactions that did not contain the pattern. Of course, this only makes sense if the transactions are given in some logical order. This type of support can be applicable in a number of domains:

- **Retail**. E.g., big supermarkets receive large quantities of goods every day. Knowing which goods will be sold in large quantities close in time helps the supermarket decide when to refill these goods.

- **Trading**. E.g., a combination of stocks being sold once may lead to waves of these stocks being sold close after each other while other combinations might not.

- **Law enforcement**. E.g., when police officers investigate telephone calls, subjects that are discussed during a longer period might be more interesting than subjects (word combinations) that are mentioned often at separate moments.

This type of support still needs to be defined and its usefulness needs to be shown. To this end, this chapter makes the following contributions:

- **We define different variations of consecutive support, having two parameters.** We will define this support with a reward factor ρ and a punishment factor σ; and we will show how this can be implemented.

- **We will show how to speed up the search by pruning the search space.** Some methods of pruning will not give all patterns, but we can find most of the important patterns faster. Other pruning methods (i.e., "parent support recalculation") will not influence the outcome, but they require more calculation and the speed up is less.

- **We will show the usefulness of consecutive support using a motivating example.** With our experimental results we show how consecutive support, compared to the results in [31], gives new and interesting patterns when applied in a biological setting of finding patterns in chromosomes.

Our research is related to work done on the (re)definition of support, gap constraint and weighted association rule mining. The notion of support was first introduced by Agrawal et al. in [3] in 1993. Much later Steinbach et al. in [63] generalized the notion of support providing a framework for different definitions of support in the future. Our notion of consecutive support is not easily fitted in the eval-function provided there. (Next to the framework Steinbach also provides a couple of example functions.) Our work also has some relation with research done in [65] concerning weighted association rule mining where different items have different weights. Consecutive support can be seen as weighted patterns based on distances between transactions that contain them.

If we take a market basket database as an example, we will have a database where the customers (or transactions) are itemsets of products they bought. We could invert this database such that transactions correspond to the products, and are itemsets of customers that bought the product. Now we can search for patterns and with techniques like the time window constraint as defined in [43] or the gap constraint as defined in [5], we can search for customers who bought products close in time. However the combination of products that were bought will be lost. Furthermore in our case we want to know which products occur often in combinations. Work in this chapter is related to [31] where it was stated that the biological problem could profit from incorporating consecutiveness into frequent itemset mining.

Finally this work is related to some of our earlier work. In [27] we mentioned that support is just another measure of saying how good a pattern fits with the data. There we defined different variations of this measure, and consecutive support can been seen as such a variation.

2.2 Consecutive Support

The definition of association rules relies on the notion of support: the number of transactions that contain a given itemset. In this chapter we propose a more general definition, that takes the consecutiveness of the transactions into account.

Suppose items are from the set $\mathcal{I} = \{1, 2, \ldots, n\}$, where $n \geq 1$ is a fixed integer constant. A *transaction* is an *itemset*, which is a subset of $\mathcal{I}$. A *database* is an *ordered series* of m transactions, where $m \geq 1$ is a fixed integer constant. If an itemset is an element of a database, it is usually referred to as a transaction.

The *traditional support* of an itemset I with respect to a database $\mathcal{D}$, denoted by $\text{TradSupp}(I, \mathcal{D})$, is the number of transactions from $\mathcal{D}$ that contain I. Clearly, $0 \leq \text{TradSupp}(I, \mathcal{D}) \leq m$.

We now propose a more general definition. Fix two real parameters $\rho \geq 0$ and $0 \leq \sigma \leq 1$. Suppose we have an itemset I and let $O_j \in \{0, 1\}$ ($j = 1, 2, \ldots, m$) denote whether or not the j^{th} transaction in the database $\mathcal{D}$ contains I (O_j is 1 if it does contain I, and 0 otherwise; the O_j's are referred to as the *O-series*). The following algorithm computes a real value t in one linear sweep through the database and the resulting t is defined as the *consecutive support* of I with respect to $\mathcal{D}$ (denoted by $\text{Supp}(I, \mathcal{D}, \rho, \sigma)$):

$t := 0$; $j := 1$; $\mathit{reward} := 0$;
while ($j \leq m$) **do**
  if ($O_j = 1$) **then**
    $t := t + 1 + \mathit{reward}$; $\mathit{reward} := \mathit{reward} + \rho$;
  else
    $\mathit{reward} := \mathit{reward} \cdot \sigma$;
  fi
  $j := j + 1$;
od

The consecutive support t can become very large, and one could for example use $\sqrt{t}$ instead. In our examples we will not use $\sqrt{t}$, and just employ t.

Example 2.2.1 Assume that the O-series of a certain pattern I equals 101101, $\rho = 1$ and $\sigma = 0.1$. The consecutive support t will then be 5.41:

O	1	0	1	1	0	1
reward	0	1	0.1	1.1	2.1	0.21
t	1	1	2.1	4.2	4.2	5.41

Note that during the loop the value of *reward*, which "rewards" the occurrence of a 1, is always at least 0. If *reward* would never be adapted, i.e., it would remain 0 all the time, this algorithm would compute $\text{TradSupp}(I, \mathcal{D})$. We easily see that $0 \leq \text{Supp}(I, \mathcal{D}, \rho, \sigma) \leq m + m(m-1)\rho/2$. The maximum value is obtained if and only if all transactions from the database

$\mathcal{D}$ contain I, i.e., an O-series entirely consisting of 1s. Only the all 0s series gives the minimum value 0. Furthermore we have for any $0 \leq \sigma \leq 1$: $\mathrm{Supp}(I, \mathcal{D}, 0, \sigma) = \mathrm{TradSupp}(I, \mathcal{D})$. For all $\rho \geq 0$ and $0 \leq \sigma \leq 1$, $\mathrm{Supp}(I, \mathcal{D}, \rho, \sigma) \geq \mathrm{TradSupp}(I, \mathcal{D})$ holds. Finally, note that the so-called APRIORI property [3] or anti-monotonicity constraint is satisfied: for all $\rho \geq 0$ and $0 \leq \sigma \leq 1$, $\mathrm{Supp}(I, \mathcal{D}, \rho, \sigma) \geq \mathrm{Supp}(I', \mathcal{D}, \rho, \sigma)$ if the itemset I' contains the itemset I. This follows from the observation that the *reward*-values in the I'-case are never larger than those in the I-case.

It is not hard to show that for the O-series $1^{a_1}0^{b_1}1^{a_2}0^{b_2}\ldots 0^{b_{n-1}}1^{a_n}$ (a series of a_1 1s, b_1 0s, a_2 1s, b_2 0s, ..., b_{n-1} 0s, a_n 1s) consecutive support equals

$$\sum_{i=1}^{n} a_i + \rho \sum_{i=1}^{n} a_i(a_i - 1)/2 + \rho \sum_{1 \leq i < j \leq n} a_i a_j \sigma^{b_i + b_{i+1} + \cdots + b_{j-1}} =$$
$$(1 - \rho/2)S + \rho S^2/2 - \rho \sum_{1 \leq i < j \leq n} a_i a_j (1 - \sigma^{b_i + b_{i+1} + \cdots + b_{j-1}}),$$

where $S = \sum_{i=1}^{n} a_i$; here 0^0 must be interpreted as 1 (an exponent 0 can be avoided by demanding all b_i's to be non-zero; if we also demand all a_i's to be > 0 both the number n and the numbers a_i and b_i are unique, given an O-series). The formula follows from the fact that if *reward* equals ε, then the series 1^k0^ℓ changes this into $(\varepsilon + k\rho) \cdot \sigma^\ell$, meanwhile giving a contribution of $k + k\varepsilon + k(k-1)\rho/2$ to the consecutive support. An extra series 0^ℓ at the beginning or end has no influence on the consecutive support.

The second term of the equation, $\rho \sum_{i=1}^{n} a_i(a_i - 1)/2$, consists of the ρ's added for a subset of consecutive 1s in the O-serie. The last term of the equation is the addition of the rewards from the previous consecutive 1s decreased with σ because of the number of 0s between the groups of consecutive 1s. Also note that when we choose $\rho = 2$ we get $S^2 - \rho \sum_{1 \leq i < j \leq n} a_i a_j (1 - \sigma^{b_i + b_{i+1} + \cdots + b_{j-1}})$. This shows that consecutive support is at most S^2 if $\rho = 2$.

Example 2.2.2 Take $\rho = 2$. Then the O-series $1^5 0^\ell 1^4$ has consecutive support $81 - 40(1 - \sigma^\ell)$. As $\ell \to \infty$ this value approaches $41 = 5^2 + 4^2$, whereas for small ℓ and $\sigma \approx 1$ it is near $81 = (5 + 4)^2$.

It can be observed that the consecutive support as defined above only depends on the lengths of the "runs" and the lengths of the intermediate "non-runs": the a_i's and b_i's above. Here a *run* is defined as a maximal consecutive series of 1s in a 0/1 sequence. Indeed, the sum $\sum_{k=i}^{j-1} b_k$ equals the

number of 0s between run i and run j. This also implies that the definition is *symmetric*, in the sense that the support is unchanged if the order of the O-series is reversed — a property that is certainly required. (In fact, this is due to the fact that ρ is added, while we multiply by σ.)

Instead of this way of calculating consecutive support it is also possible to augment the O-series with *time stamps*. Then one is able to use the real time between two transactions in calculating the consecutive support. In the previous definition each transaction was assumed to take the same amount of time and there are no time gaps between transactions. Another improvement might be to reinitialize *reward* to 0 at suitable moments, for instance at chromosome boundaries or at "closing hours".

We now consider algorithms that find all frequent itemsets, given a database. A *frequent* itemset is an itemset with support at least equal to some pre-given threshold, the so-called *minsup*. Thanks to the APRIORI property many efficient algorithms exist. However, the really fast ones rely upon the concept of FP-TREE or analogues, which does not keep track of consecutivity. This makes these algorithms hard to adapt for consecutive support.

One fast algorithm that does not make use of FP-TREES is called ECLAT [77]. ECLAT grows patterns recursively while remembering which transactions contained the pattern, making it suitable for consecutive support. In the next recursive step only these transactions are considered when counting the occurrence of a pattern. All counting is done using a matrix and patterns are extended with new items (using the order in the matrix). This can easily be adapted to incorporate consecutiveness. The ECLAT algorithm works as follows:

1. Construct a matrix where rows are transactions and columns are the items.

2. For each 1-item itemset count their support and store in which transactions they occur.

3. Make an ordering for these items.

4. Extend a k-item itemset (parent pattern) to a $(k+1)$-item itemset (child pattern) by adding one item that comes later in the order.

5. Count the support for the $(k+1)$-item itemset and update the sets of transaction in which the pattern occurs.

6. Recursively continue until the *minsup* is reached.

2.3 Pruning Methods

The consecutive support of patterns can be much higher than the traditional support. As a consequence more patterns will be frequent or the minimal support threshold should be set much higher. Pruning is important, since a high minimal support might result in the skipping of interesting patterns. The lower we can set our minimal support, and still find a solution fast, the more flexibility the algorithm allows the user. In this chapter we propose several pruning methods. These are implemented in our version of ECLAT, which counts consecutive support. Our version of ECLAT will be called from here on CONSECLAT.

We will propose different pruning methods (to be discussed in detail in the next subsections):

- "Parent Support Recalculation", given the support of the parent pattern and the collected child pattern support, we can calculate (while counting) if *minsup* can still be reached for the child.

- "Introducing α", with α we let the user estimate the probability of a child pattern occurring. In this way the calculated maximal achievable support becomes an estimate.

- "Exact Depth", here we prune those patterns that will never reach an user-defined minimal length.

Some of the used pruning methods influence the completeness: one will not get all patterns, because we will stop counting based on probabilities.

2.3.1 Parent Support Recalculation

The first pruning method we discuss does not affect completeness. Basically this *parent support recalculation* method does the following for each transaction r:

- Calculate the consecutive support the parent had collected before considering transaction r, where the *child* is the current itemset, being the *parent* itemset generated in the previous recursive step extended with one item.
- Subtract this support from the total support of the parent.
- Add to this the support the child pattern has collected up until now. The child can still maximally achieve this consecutive support, from here on called *maximal achievable support*.
- Return a support of 0 if this is less than the minimal support.

In re-calculating the support of the parent pattern at a certain transaction we make use of the fact that we store which transactions contained the parent pattern. In ConseClat we use a list of transaction numbers that contain the pattern. With these numbers we can (re)calculate the consecutive support of the parent in the same loop through the database:

$$\begin{aligned} &\mathit{reward\_parent} := \mathit{reward\_parent} \cdot \sigma^{\mathit{diff}} \\ &\mathit{partial\_support\_parent} := \mathit{partial\_support\_parent} + 1 + \mathit{reward\_parent} \\ &\mathit{reward\_parent} := \mathit{reward\_parent} + \rho \end{aligned}$$

where *diff* is the number of transactions that did not contain the parent pattern:

$$\mathit{diff} := \mathit{current\_transaction\_number} - \mathit{last\_transaction\_number} - 1$$

Here *last_transaction_number* is the transaction number of the last transaction (before *current_transaction_number*, the current one) that contained the parent pattern. Now the maximal achievable support for the child is as follows:

$$\mathit{possible} := \mathit{parent\_total} - \mathit{partial\_support\_parent} + \mathit{support}$$

The variable *parent_total* is the support the parent pattern was able to achieve and *support* is the consecutive support that the child-pattern was able to "collect" until the current transaction.

Now the algorithm will stop counting support when it is no more possible to still achieve a support that is higher than the minimal support. The child pattern can at most get the maximal achievable support, because it can never score better than its parent on the remaining transactions.

Example 2.3.1 Assume the following "child"-pattern that is an extension of the "parent"-pattern:

O_{parent}	1	1	1	0	0	1
O_{child}	0	0	0	0	0	1

Furthermore assume $minsup = 5$, $\sigma = 0.1$ and $\rho = 1.0$; then we can stop counting support when we encounter the second zero. At that point we know that at most we can get a consecutive support of 4.03 (the consecutive support of the parent was 7.03 and a consecutive support of 3 was lost in the child).

2.3.2 Introducing α

In the parent support recalculation method we assumed that the remaining transactions will all contain the child. This is an optimistic estimate necessary for guaranteeing completeness. We could assume that the child pattern will be contained in less than all of the remaining transactions α, $0 \leq \alpha \leq 1$. We then introduce this α in the calculation of maximal achievable support:

$$possible := \alpha \cdot (parent\_total - partial\_support\_parent) + support$$

This will speed up the mining process, but we lose completeness.

2.3.3 Exact Depth

In the case of our motivating example biologists expressed the desire to visualize only long patterns, because the small patterns are so numerous that affected areas are less recognizable. This wish to only get patterns of a certain minimal length can be used for pruning. Hence we allow the user to set the minimal length η that patterns should have, and we can prune if the following holds:

$$last\_frequent\_item - item < \eta - depth$$

where the items are represented as numbers lexicographically ordered in the matrix used by ConseClat, *last_frequent_item* is the last item in that matrix that is still frequent and *item* is the current item that we are considering. The *depth* is the recursive depth, which is equal to the length of the pattern. If the inequality statement holds then the pattern will never reach the required length η and it can be pruned.

Example 2.3.2 Assume given the following database:

item numbers:	1	2	3	4
transaction 1	1	0	1	1
transaction 2	1	0	0	1
transaction 3	0	1	1	1

Assume that $\eta = 4$, the parent item set is $\{1\}$ and we are considering to extend this with $\{3\}$ to the child $\{1, 3\}$. However $4-3 < 4-2$ and so $\{1, 3\}$ and all its children are pruned.

2.3.4 Hyperclique Patterns and h-confidence

Many principles applicable to traditional support can still be used when one considers consecutive support. In the case of our working example we wanted to consider patterns with a minimal consecutive support of 25. Unfortunately there are many patterns with this support. In order to speed up the search and to filter out uninteresting patterns we can search for *hyperclique patterns* as described in [73].

Definition 2.3.1 (Hyperclique pattern) *Assume we have a* minimal confidence threshold h_c. *An itemset p is called a* hyperclique pattern *in the case that when each item of p occurs that the other items also occur in "most cases". The latter is decided by calculating confidence and using h_c.*

We explain hyperclique patterns using by means of an example:

Example 2.3.3 First a *minimal confidence threshold* h_c is defined, say $h_c = 0.6$ and then we want to know if $\{A, B, C\}$ is a hyperclique pattern. We calculate the confidence of $\{A\} \rightarrow \{B, C\}$, $\{B\} \rightarrow \{A, C\}$ and $\{C\} \rightarrow \{A, B\}$. The lowest of these confidences is the *h-confidence*, which must be higher than h_c. Assume that $conf(A \rightarrow B, C) = \text{Supp}(\{A, B, C\}, \mathcal{D}, \rho, \sigma)$ / $\text{Supp}(\{A\}, \mathcal{D}, \rho, \sigma) = 0.58$. Then $\{A, B, C\}$ is no hyperclique pattern.

When we combine the concept of consecutive support with hyperclique patterns we get patterns that occur frequent, but in the flow of transactions close after each other, and there is a *strong affinity* between items: the presence of $x \in P$, where P is an item set, in a transaction strongly implies the presence of the other items in P.

Hyperclique patterns possess the *cross-support property*. This means that we will not get *cross-support patterns* (patterns containing items of substantially different support levels).

We can easily see that hyperclique patterns possess the cross-support property. If one item has a high support and another item has a low support then the h-confidence will be low if the denominator is the item with the high support.

Example 2.3.4 Say A is an item with a consecutive support of 200 and B has a consecutive support 50. The support of $\{A, B, C\}$ will at most be 50 because of the APRIORI property (the support of the superset is always the same as or less than the support of its subsets). So the confidence $conf(A \rightarrow B, C)$ can at most be $50/200 = 0.25$. As a consequence the h-confidence of $\{A, B, C\}$ will also be at most 0.25. And if $h_c = 0.6$ then $\{A, B, C\}$ and all the patterns that are grown from it can be pruned.

The combination of hyperclique patterns and consecutive support allows us to find patterns that occur in transactions that follow each other close, yet minimal support can be relatively low. This property is especially handy for our motivating example, because a minimal consecutive support of 25 will generate many cross-support patterns, which are pruned if we search only for hyperclique patterns. Hyperclique patterns also possess the anti-monotone property, because as patterns grow the numerator of the confidence calculation stays the same or declines. The denominator stays fixed and so h-confidence will decrease or stay the same:

Example 2.3.5 Assume $conf(A \rightarrow B, C) = 0.58$. The superset $\{A, B, C, D\}$ will at most have the same consecutive support as $\{A, B, C\}$. Also the denominator $\text{Supp}(\{A\}, \mathcal{D}, \rho, \sigma)$ stays the same, so the h-confidence of the set $\{A, B, C, D\}$ can at most be 0.58.

2.4 Results and Performance

In this section we test the implementation by a number of experiments. The experiments were done for three main reasons. First of all we want to show that consecutive support can enable one to find new patterns that one does not find with the traditional support. Secondly we want to show how using the principle of h-confidence one can filter the data. Finally we want to give an indication how the reward factor ρ and punishment factor σ should be chosen.

We do not see a correlation between traditional and consecutive support other than the fact that patterns with a high support are expected to be more consecutive. Here we will not further study this correlation. However

it could be interesting to further investigate this effect since some patterns have a lower support than others but they occur more consecutive. It must be noted however that a measure incorporating this effect have not been found to be anti-monotone and as such makes pruning difficult.

All experiments were done on a Pentium 4 2.8 GHz with 512MB RAM. For our experiments we used two real datasets and four synthetic datasets. One real biological dataset, referred to as the `Nakao` *dataset*, was also used in [31]. This data set originates from Nakao et al. who used the dataset in [53]. This publicly available dataset contains normalized $\log_2$-ratios for 2,124 clones, located on chromosomes 1–22 and the X-chromosome. Each clone is a transaction with 2 to 1,020 real numbers corresponding to patients. We can look at gains and/or losses. If we consider gains, a patient is present in a transaction (clone) if his value is at least 0.225 higher than that of a healthy person (for losses at least 0.225 lower).

The second real dataset we call the `one-user` *dataset* and it contains the webpages accessed by one heavy user of the former `portalexecutivo.com` website on a single day. Some days there is no access and hence some of the 1,603 transactions are empty. Webpages are categorised resulting in a total of 185 possible items for every transaction.

Two datasets are synthetic databases, but structured like the dataset of clones. One of these datasets, the `noisy` *dataset*, contains more noise than the other, the `ideal` *dataset*. The precise structure of these datasets is described in [31]: both the noisy and the ideal dataset consist out of 3,200 clones with 150 real numbers as the items (the patients). The ideal datasets has 115 items with sufficient gains and 70 with sufficient losses. Depending on if you analyse losses or gains, "sufficient" means we have 115 frequent 1-itemsets or 70 frequent 1-itemsets. The noisy datasets all 1-itemsets are frequent (gains and losses).

The remaining datasets are synthetic datasets made to show how consecutive support can be used to find patterns that could not be found before. The third synthetic data set, referred to as the `food+drink` *dataset*, describes a cafe-restaurant where in the middle of a day a lot of people buy bread and orange juice; it has 1,000 transactions (customers) and 100 items (products). The fourth synthetic data is called the `coffee+cookie` *dataset*, where in the cafe-restaurant small bursts of people buy coffee and a cookie (during the day in the coffee breaks). This dataset also has 1,000 transactions (customers) and 100 items (products). The synthetic datasets are all available from the MISTA website `mista.liacs.nl`.

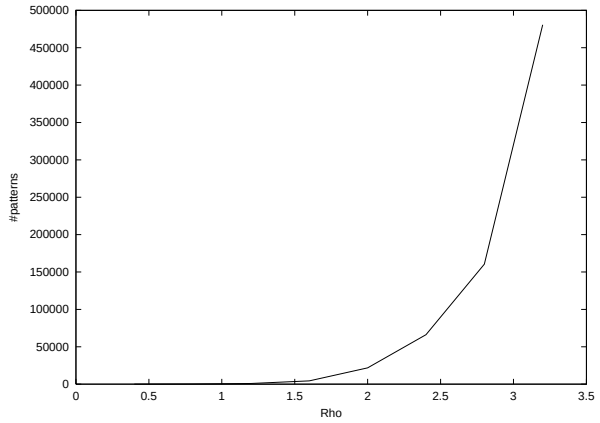

Figure 2.1: Number of patterns from the `Nakao` dataset as ρ increases (gains, $minsup = 625$, $\sigma = 0.5$)

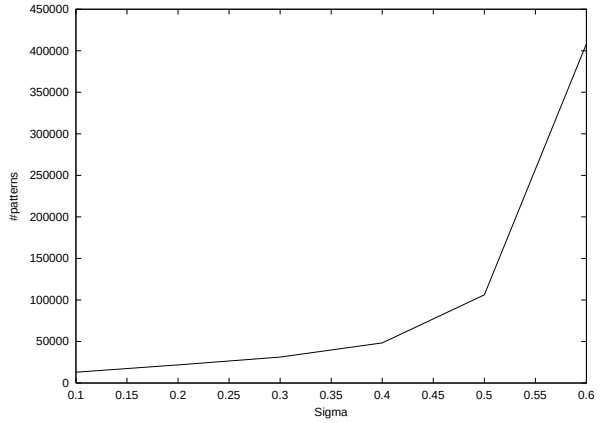

Figure 2.2: Number of patterns from the `Nakao` dataset as σ increases (gains, $minsup = 625$, $\rho = 2.0$)

2.4.1 Consecutive Support

Figure 2.1 and Figure 2.2 show how the number of patterns increases with ρ and σ. Each setting therefore requires another *minsup*. The height of the consecutive support depends on how high we choose ρ and σ. A higher reward and a weak punishment will lead to a higher consecutive support. A higher *minsup* threshold will give as an outcome more occurring patterns

that are more consecutive. Usually a user wants to know those patterns occurring a minimal number of times, but consecutive. When choosing minimal consecutive support (*minsup*) one should choose it higher than the minimal occurrence threshold and lower than the power of the minimal occurrence threshold. When choosing the latter one would only find either more occurring or extremely consecutive patterns.

In some cases it is best to select the *minsup* such that one gets a fixed number of patterns, e.g., 1,000, in order to compare the results.

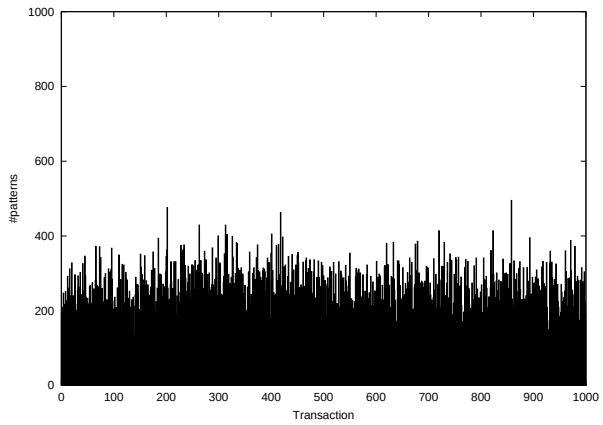

Figure 2.3: Occurrence graph of `food+drink` using traditional support ($minsup = 257$)

In the experiments of Figure 2.3, 2.4, 2.5 and 2.6 we tried to find approximately 1,000 patterns with the highest traditional or consecutive support. After this we count for each transaction how many patterns it contains, allowing us to see how active areas are. For the `Nakao` dataset more active means that many clones (gains) in the same area are present in many groups of patients.

Figure 2.3 and Figure 2.4 show where patterns occur when we use traditional support, giving results similar to those in [31]. For each transaction the number of patterns that it occurs in is plotted in a so-called *occurrence graph*. In each of these graphs we will indicate chromosome borders when the `Nakao` dataset is visualized. In the `food+drink` dataset it is very clear that consecutive support enables us to see new patterns. Figure 3.3 shows that in certain areas patterns are more consecutive. Figure 2.6 shows that certain areas are less active if we use consecutive support instead of tradi-

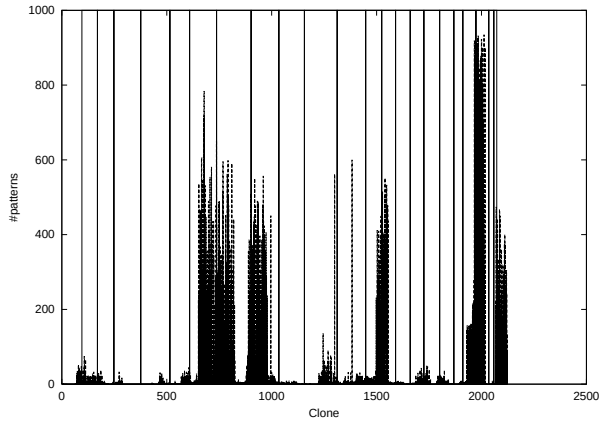

Figure 2.4: Occurrence graph of `Nakao` using traditional support (gains, $minsup = 129$)

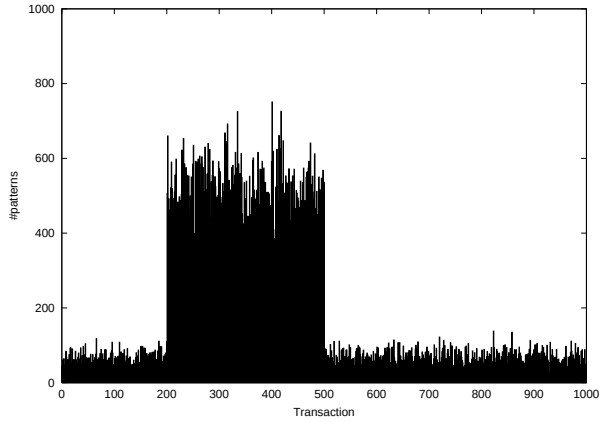

Figure 2.5: Occurrence graph of `food+drink` using consecutive support ($minsup = 467$, $\rho = 1.0$ and $\sigma = 0.5$)

tional support (chromosomes 7 and 8, near clones 600 and 800) and some areas contain more patterns (chromosome 9, near clone 1000), hence providing patterns that occur together in one part of the chromosome instead of far apart.

In order to evaluate the effect of noise on consecutive support we used the `ideal` and `noisy` dataset. These datasets are generated with properties

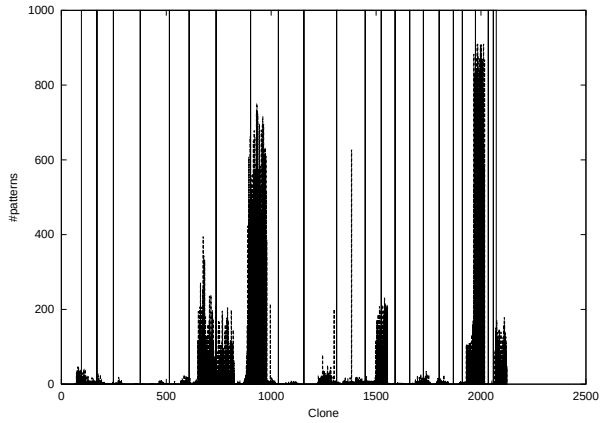

Figure 2.6: Occurrence graph of `Nakao` using consecutive support (gains, $minsup = 827$, $\rho = 1.0$ and $\sigma = 0.5$)

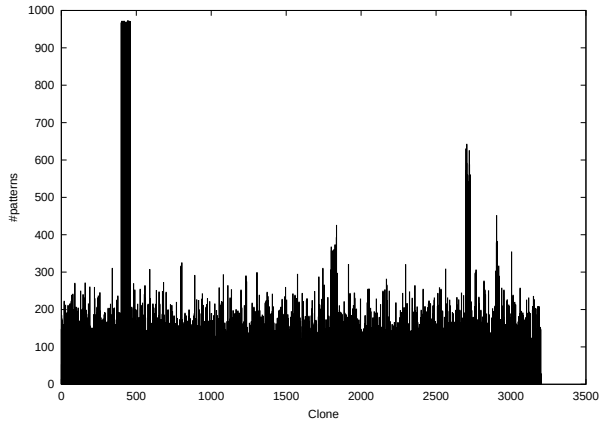

Figure 2.7: Occurrence graph of the `ideal` dataset using traditional support (gains, $minsup = 479$)

similar to the `Nakao` dataset. The results for the `ideal` dataset are plotted in Figure 2.7 and 2.8.

Figure 2.7 shows that some interesting areas are less clear when using traditional support. However they become more apparent when we apply consecutive support.

The results for the `noisy` dataset are displayed in Figure 2.9 and 2.10,

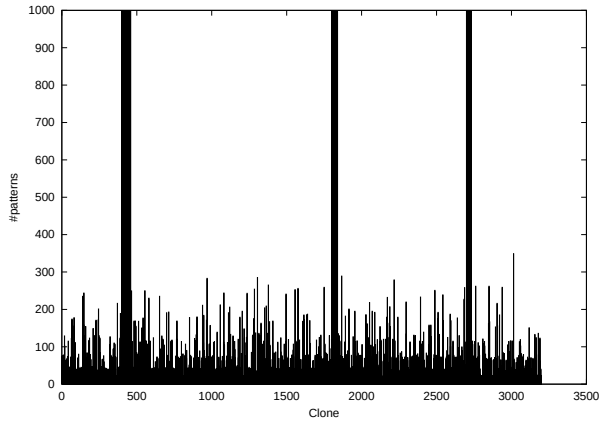

Figure 2.8: Occurrence graph of the `ideal` dataset using consecutive support (gains, $minsup = 6,180$, $\rho = 2.0$ and $\sigma = 0.7$)

because of the noise the middle peak becomes less clear. However overall the results seem hardly to be affected by noise.

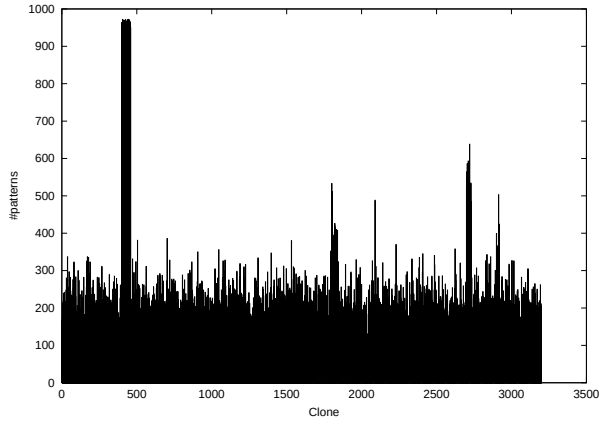

Figure 2.9: Occurrence graph of the `noisy` dataset using traditional support (gains, $minsup = 617$)

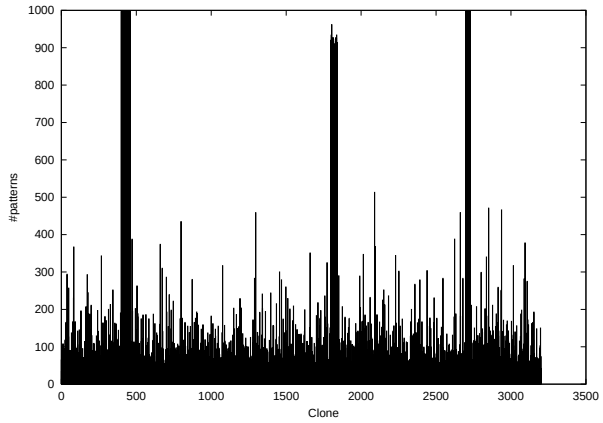

Figure 2.10: Occurrence graph of the `noisy` dataset using consecutive support (gains, $minsup = 6,039$, $\rho = 2.0$, $\sigma = 0.7$)

2.4.2 Selection of ρ and σ

The goal of the experiments in this subsection is to give some guidance in the selection of the reward factor ρ and the punishment factor σ. Each pattern has its corresponding O-series that indicates in which transactions it occurs. The right parameters should result in many patterns of which the O-series has large groups of consecutive 1s.

Figure 2.11 plots the average number of consecutive groups of 1s and 0s for all patterns. Here the value for *minsup* is heuristically chosen by linearly deriving it from *minsup* as it was empirically decided for $\rho = 2$.

The plot gives an indication of consecutiveness of patterns found using different settings of ρ and σ (less groups indicate more consecutiveness). The plot seems to stabilize around $\rho = 2$. Figure 2.12 and 2.13 show that only if we choose σ very close to 1.0 we get results more like those for traditional support. However, Figure 2.13 still shows some influence of ρ. For the `Nakao` dataset it seems that if $\rho \approx 2$, then the influence of σ is minimalized as long as σ is not too close to 1.0. Also similar experiments showed significant changes in the occurrence graph only if ρ was chosen very small.

In Figure 2.14 we see the number of groups of 1s and 0s drop in a similar fashion for the `one-user` dataset. However we also see a sharp drop if σ is exactly 1.0, i.e., no punishment for gaps. This is probably caused by the fact that the `one-user` dataset contains many very consecutive patterns with only some very large gaps between their occurrence. The users often went

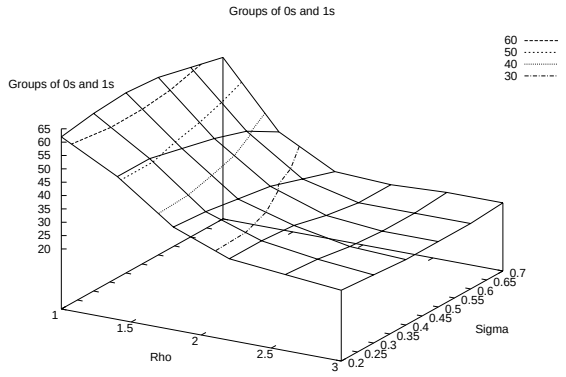

Figure 2.11: Effect of ρ and σ on the O-series for the `Nakao` dataset (gains, $minsup = 625 \cdot (\rho/2)$, heuristically chosen to guarantee a reasonable amount of patterns)

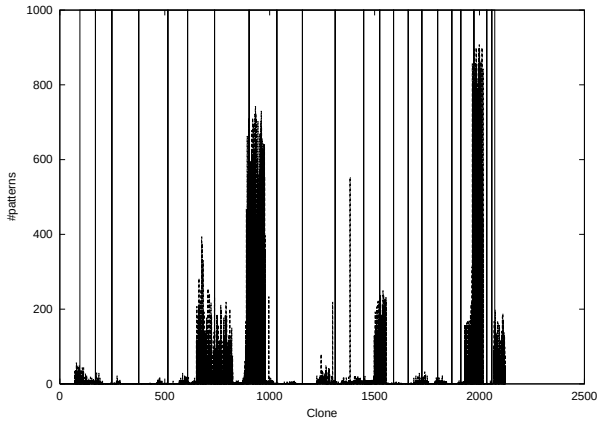

Figure 2.12: Occurrence graph of `Nakao` using consecutive support (gains, $minsup = 2,498$, $\rho = 2.0$ and $\sigma = 0.8$)

online for a couple of days and then stayed offline for a long period. It is also likely that the user did one type of task during one week and another type of task the next week (requiring different pages) and the other type again after a while. In this way one gets many patterns with large groups of 0s (non-occurrences) between groups of 1s (occurrences).

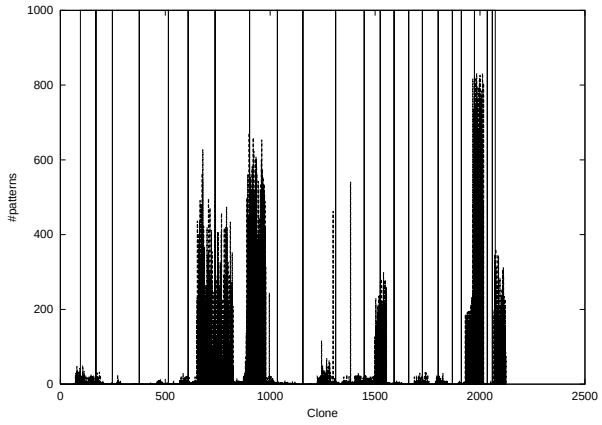

Figure 2.13: Occurrence graph of `Nakao` using consecutive support (gains, $minsup = 6,157$, $\rho = 2.0$ and $\sigma = 0.99$)

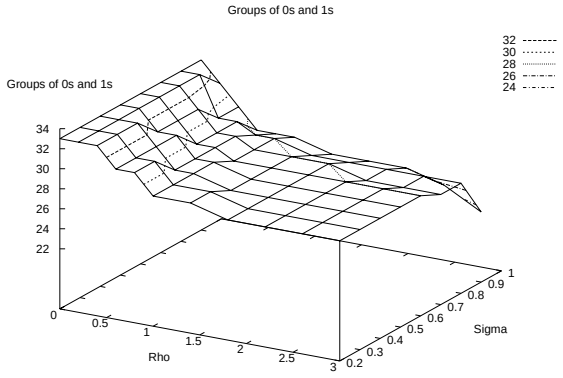

Figure 2.14: Effect of ρ and σ on the O-series for the `one-user` dataset (gains, $minsup = 25$)

2.4.3 Combination with h-confidence

In the following experiments the goal was to show that combining hyperclique patterns, see Example 2.3.5 with consecutive support enables us to see patterns occurring in bursts. In order to show this we used the `coffee+cookie` dataset, where in the cafe-restaurant small bursts of people buy coffee and a cookie (during the day in the coffee breaks). This dataset

has 1000 transactions (customers) and 100 items (products).

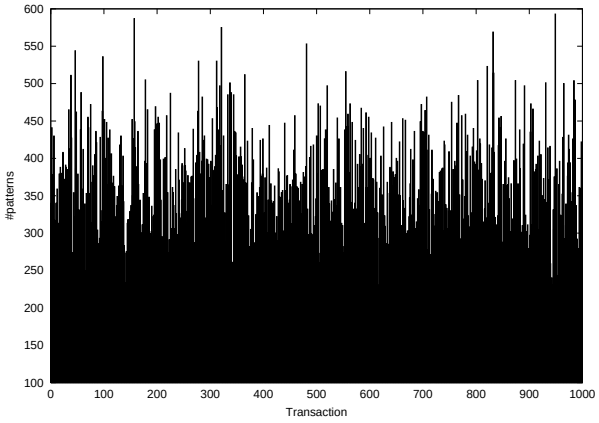

Figure 2.15: Occurrence graph of coffee+ cookie using only h-confidence ($minsup = 64$, $h_c = 0.5$)

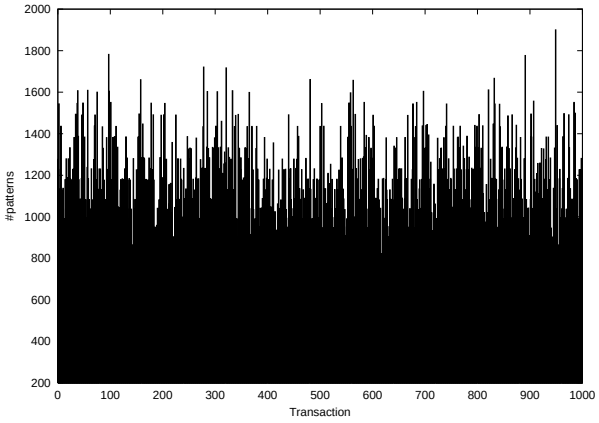

Figure 2.16: Occurrence graph of `coffee+cookie` using only consecutive support ($minsup = 225$, $\rho = 1.0$, $\sigma = 0.5$, $h_c = 0$)

Figure 2.15 does not show the small groups buying the same products: just hyperclique patterns do not reveal the bursts. Figure 2.16 shows that with only consecutive support we are also unable to discover these patterns. Figure 2.17 shows people buying the products in bursts. Consecutive support

stresses patterns that are consecutive and the principle of h-confidence filters out the noise caused by cross-support patterns.

When we apply these techniques to the `Nakao` dataset (losses), in Figure 2.19, we can see, e.g., on chromosomes 14 and 15 (near clone 1,600) that certain areas become more active compared to not using h-confidence in Figure 2.18.

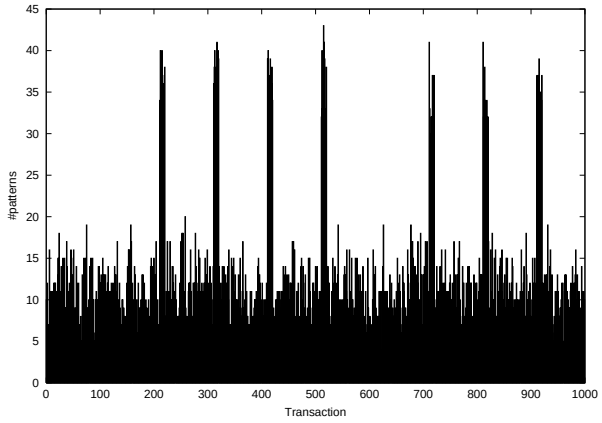

Figure 2.17: Occurrence graph of `coffee+cookie` using both consecutive support and h-confidence ($minsup = 64$, $\rho = 1.0$, $\sigma = 0.5$, $h_c = 0.31$)

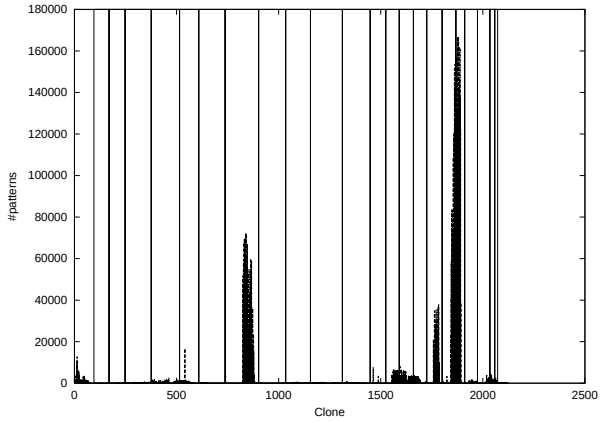

Figure 2.18: Occurrence graph of `Nakao`: consecutive support (losses, $minsup = 400$, $\rho = 1.0$, $\sigma = 0.9$, $h_c = 0$)

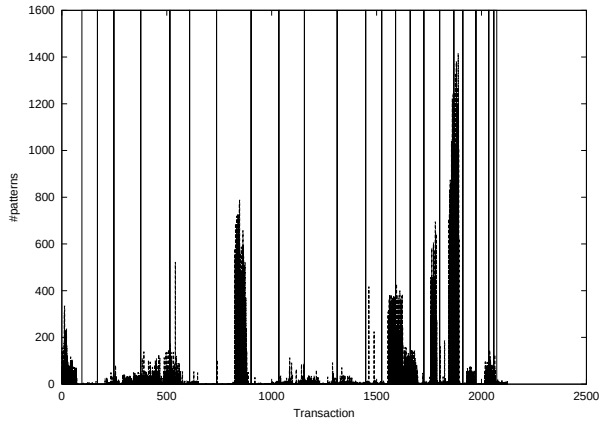

Figure 2.19: Occurrence graph of `Nakao`: consecutive support and h-confidence (losses, $minsup = 25$, $\rho = 1.0$, $\sigma = 0.9$, $h_c = 0.15$)

2.5 Conclusions

From our experimental results it follows that consecutive support enables us to find new and useful patterns compared to traditional methods. Principles applicable to traditional support can still be used with consecutive support: for instance the combination of consecutive support and the h-confidence threshold enables us to find small bursts of patterns. In this case h-confidence filters out noise and consecutive support amplifies the bursts.

Consecutive support might result in many more patterns. Because of this, pruning the search space is important. In this chapter we proposed a number of methods for pruning, where some methods do not give all patterns.

Using the distance between transactions (like it was done in this chapter) is an interesting area of research. In the future we want to examine if consecutive support enables us to visualize even more types of behavior. Also we want to see if we can speed up the search for consecutive patterns even more. Finally we want to extend consecutive support by using distance between transactions in different ways, which hopefully will give us new and interesting patterns.

3 Patterns with a Fixed Interval In Between

In this chapter we propose a new measure of support: we count the number of times a pattern occurs (nearly) in the middle between two other occurrences. We will define this measure formally and show how it can be added to the ECLAT algorithm for finding frequent patterns. We will show that if patterns occur often in the middle transaction, then the interval between their occurrences is stable.

We will use the deviation of the number of in-between non-occurrences to pick patterns with a fixed occurrence interval; we will call these "balanced". Because the anti-monotone property does not hold, we will also introduce a new way of pruning the search space.

3.1 Introduction

This research is related to work done on the (re)definition of support, using time with patterns and the incorporation of distance measured by the number of transactions between pattern occurrences. The notion of support was first introduced by Agrawal et al. in [3] in 1993. Since then many new and faster algorithms where proposed. We make use of ECLAT, developed by Zaki et al. in [77]. Steinbach et al. in [63] generalized the notion of support providing a framework for different definitions of support in the future. Our work is also related to work described in [50] where association rules are mined that only occur within a certain time interval. Furthermore there is some minor relation with mining data streams as described in [11, 20, 66], in the sense that they use time to say something about the importance of a

pattern.

Finally this issue is related to some of our earlier work. Results from [31] indicated that the biological problem could profit from incorporating consecutiveness into frequent itemset mining, which was elaborated in Chapter 2. In the case of stable patterns we also make use of the transactions and the distance between them. Secondly in Chapter 5 it was mentioned that support is just another measure of saying how good a pattern fits with the data. There we defined different variations of this measure, and stability can been seen as one such variation.

3.2 Stable Patterns

Mining frequent patterns is an important area of data mining where we discover substructures that occur often in (semi-)structured data. In this chapter we will further investigate one of the simplest structures: itemsets. Much research has been done in the area of frequent itemset mining. We will propose an algorithm that discovers patterns that occur at regular moments, or rather in regular intervals. This will enable us to mine for events that occur, e.g., every Friday. The technique can be extended to more complicated structures like sequences.

Stable patterns are patterns that occur frequent and with certain intervals. The *interval* is the number of transactions *without* the pattern in between two transactions (events) *with* the pattern. Instead of discovering stable patterns one could consider to add an extra item, for example the day of the week. However by discovering stable patterns we hope to find more unexpected intervals, e.g., every three hours a pattern occurs.

We will define this type of support and show its usefulness. To this end, this chapter makes the following contributions, with emphasis on the first and second:

— We will **define stable patterns and show that they possess the** Apriori **property**. This means that if pattern p' is contained in pattern p, then the stability value of p is at most equal to that of p'. The property makes an efficient implementation possible.

— Furthermore we will **propose an algorithm** for the discovery of stable patterns and discuss its efficiency.

— Finally we will show that **this enables us to find new and interesting patterns** via explorative experiments on real and synthetic datasets.

Our working example is the mining of an access log from the Computer Science Institute of Leiden University. This access log will first be converted

to sets of properties in, e.g., pages visited every hour. From here on we call this dataset the `website` dataset.

The formal definitions concerning stable patterns and an algorithm are given in Section 3.3. In Section 3.5 we present experimental results.

3.3 Definition

In this section we will define stable patterns. In particular, patterns that occur at regular intervals (e.g., at equidistant time stamps) will be called stable. In order to judge this property, we will determine how often events occur "in the middle" between two other events.

In this chapter a dataset consists of transactions that take zero time. Each transaction is an itemset, i.e., a subset of $\{1, 2, 3, \ldots, max\}$ for some fixed integer max. The transactions can have time stamps; if so, we assume that the transactions take place at different moments. We choose some notion of *distance* between transactions; examples include: (1) the distance is the time between the two transactions and (2) the distance is the number of transactions (in the *original* dataset) strictly in between the two transactions. We will define $Trans(p)$ as the series of transactions that contain pattern (i.e., itemset) p; the *support* of a pattern p is the number of elements in this ordered series.

We now define *w-stable patterns* as itemsets that occur frequent (support $\geq$ *minsup*) in the dataset and that have *stability value* $\geq$ *minstable*, where the values *minsup* and *minstable* are user defined thresholds. A *w-good triple* (L, M, R) consists of three transactions L, M and R, occurring in this order, such that $|distance(L, M) - distance(M, R)| \leq 2 \cdot w$; here w is a pregiven small constant, e.g., $w = 0$. The stability value of a pattern p is the number of w-good triples in $Trans(p)$, plus the number of transactions in $Trans(p)$ that occur as left endpoint in a w-good triple, plus the number of transactions in $Trans(p)$ that occur as right endpoint in a w-good triple.

Note that the stability value of a pattern p' with $p' \subseteq p$ is at least equal to that of p: the so-called Apriori or anti-monotone property. Also note that the stability value remains the same if we consider the dataset in reverse order.

We now show that equidistant events are "very" stable (in case $w = 0$):

Theorem Suppose that $Trans(p)$ has n elements, so p has support n. If $Trans(p)$ satisfies:

1. $n - 2$ elements occur as the left endpoint of a 0-good triple,

2. $n-2$ elements occur as the right endpoint of a 0-good triple, and

3. the number of 0-good triples equals $\lfloor n/2 \rfloor (\lceil n/2 \rceil - 1)$
   i.e., for even n: $n/2\,(n/2-1)$; for odd n: $((n-1)/2)^2$

then the transactions in *Trans*(p) are *equidistant*. The values in 1, 2 and 3 are maximal, as is their sum.

Proof We proceed from the right (formally by induction). The end of the sequence *Trans*$(p) = (T_1, T_2, \ldots, T_n)$ looks like:

$\cdots$	$L\&R$	$L\&R$	$L\&R$	$L\&R$	$L\&R$	R	R
$\cdots$	T_{n-6}	T_{n-5}	T_{n-4}	T_{n-3}	T_{n-2}	T_{n-1}	T_n
$\cdots$	6	5	4	3	2	1	0

Here L/R denotes: this T_i is a left/right endpoint in a 0-good triple; the numbers beneath the T_i's indicate the number of times T_i is the middle of a 0-good triple.

First observe T_{n-2}, T_{n-1} and T_n, where T_{n-2} is a left endpoint of a 0-good triple; this implies that *distance*(T_{n-2}, T_{n-1}) = *distance*$(T_{n-1}, T_n) = a$ for some a.

Now suppose we have the following situation: $T_i = L$ (with $i \geq \lfloor n/2 \rfloor$) is the left endpoint of a 0-good triple (L, M, R), for some $M = T_j$ with $j > i$; furthermore $a =$ *distance*$(T_\ell, T_{\ell+1})$ for all $\ell > i$. Now T_j occurs $n-j$ times as middle of 0-good triples, whose right endpoints are the consecutive $T_{j+1}, \ldots, T_n$. We can conclude that *distance*$(T_i, T_{i+1}) = a$. So we have *distance*$(T_\ell, T_{\ell+1}) = a$ for $\ell = \lfloor n/2 \rfloor, \lfloor n/2 \rfloor + 1, \ldots, n$.

Similarly, using the right endpoints, one can show that *distance*$(T_\ell, T_{\ell+1}) = b$ for some b ($\ell = 1, 2, \ldots, \lceil n/2 \rceil$). Using $\ell = \lfloor n/2 \rfloor$ we see that $a = b$. □

Example 1 Assume we have the following itemsets in our dataset:

transaction 1: {A, B, C}
transaction 2: {D, C}
transaction 3: {A, B, E}
transaction 4: {E, F}
transaction 5: {A, B, F}
transaction 6: {E, F}
transaction 7: {A, B, F}
transaction 8: {E, F}
transaction 9: {A, B, C}

As distance we take the number of intermediate transactions. The stability value (with $w = 0$) of $\{A,B\}$ is $4 + 3 + 3 = 10$, the maximal value possible. There are 4 0-good triples; we have 3 transactions that are left (right) endpoint of a 0-good triple (see picture below, left). If we insert two transactions $\{E, F\}$ between transaction 1 and 2, and also two between 8 and 9, we still have 4 0-good triples, but now we only have 2 transactions that are left (right) endpoint of a good 0-triple (see picture below, right), leading to stability value $4+2+2 = 8 < 10$. This example shows that condition *3* from the Theorem is in itself not sufficient yet in order to guarantee equidistance.

3.4 Algorithm

We now consider algorithms that find all stable patterns, given a dataset. Thanks to the APRIORI property many efficient algorithms exist. However, the really fast ones rely upon the concept of FP-TREE or something similar, which does not keep the order of transactions. This makes these algorithms hard to adapt for discovering stable patterns.

ECLAT [77] is a fast algorithm that does not make use of FP-TREEs; it grows patterns recursively while remembering which transactions contained the pattern, making it very suitable for our purpose. In a recursive step only these transactions are considered when counting the occurrence of a pattern. All counting is done by using a matrix and patterns are extended with new items using the order in the matrix. This can easily be adapted to incorporate stability.

Now suppose that $Trans(p) = T^{parent}$, with $n = |T^{parent}|$, is the ordered series of transactions (augmented with their index numbers from the *original* dataset) that contain itemset p. The algorithm below (Algorithm 1) will calculate the stability value when adding a new *item* to p. The algorithm will also calculate the support and the new series of transactions T^{child} that will be considered in the next step of a frequent pattern mining algorithm: the ECLAT algorithm is extended to STABLECLAT. The *child* is the *parent* itemset p extended with the new *item*. Note that *Left* and *Right* are *sets*. In line (9) we add the *index numbers* of the transactions. The function $contains(trans, item)$ checks if the transaction *trans* contains the item *item*, the function $has(T^{parent}, index)$ verifies that transaction *index* is in T^{parent}; T_{index} is the transaction as retrieved from the original dataset. The *mindepth* threshold defines from which depth the stability should be calculated —

otherwise, for small itemsets with large supports the computation would become cumbersome. The *depth* is the recursive depth that is equal to the size of the child pattern that we are considering.

Algorithm 1 Stability Value

1: $support := 0$, $stable := 0$, $Left := \emptyset$, $Right := \emptyset$, $T^{child} :=$ empty series, $i := 1$
2: **while** $i \leq n$ **do**
3: **if** $contains(T_i^{parent}, item)$ **then**
4: $T^{child} := T^{child}$ with T_i^{parent} appended, $support := support + 1$
5: **if** $depth \geq mindepth$ **then**
6: $j := i + 2$
7: **while** $j \leq n$ **do**
8: **if** $contains(T_j^{parent}, item)$ **then**
9: $middle := (T_i^{parent} + T_j^{parent}) \bmod 2$, $index := (T_i^{parent} + T_j^{parent}) / 2$
10: **if** $middle = 0$ **and** $has(T^{parent}, index)$ **and** $contains(T_{index}, item)$ **then**
11: $stable := stable + 1$
12: $Left := Left \cup \{T_i^{parent}\}$, $Right := Right \cup \{T_j^{parent}\}$
13: **end if**
14: **end if**
15: $j := j + 1$
16: **end while**
17: **end if**
18: **end if**
19: $i := i + 1$
20: **end while**
21: $stable := stable + |Left| + |Right|$

This algorithm will only increase *stable* if the pattern is *exactly* in the center of two transactions containing the pattern (so $w = 0$); this can be easily generalized. It is possible that the pattern does not occur in the center transaction but in a transaction that is very near. This can be recognized when $w > 0$, and should give a better score in that case. One possibility is to also count patterns almost in the center. For which threshold w can be specified. Suppose T_i is the outer left transaction and T_j is the outer right transaction, then we consider every T_ℓ, where $i < (i + j)/2 - w \leq \ell \leq (i + j)/2 + w < j$. Now our algorithm needs to check if the pattern occurs

in one of these transactions.

Example 2 Suppose we have the same 9 transactions as in the previous example.
If $w = 1$ then the stability value of $\{A,B\}$ will be $10 + 2 = 12$: transactions 1 and 7 (and 3 and 9) are now also endpoints of a 2-good triple.

The maximal stability value depends on the size of the database. This makes setting the minimal stability threshold *minstable* somewhat difficult. To make it easier one only needs to give a number *dist*, where $dist > 0$. With *dist* we calculate the *stable* value if the distance between all transactions containing the pattern is precisely *dist*. In this calculation we disregard the count of the left and right endpoints. This *dist* can now be used to propose a reasonable value for *minstable*, where D is the original dataset:

$$minstable = \binom{|D|/dist}{2}$$

Most frequent itemsets or patterns have a high stability value because it is more likely that a center transaction contains the pattern. However these patterns will not necessarily make a stable pattern more apparent. Furthermore it might also be contained in many transactions that do not form a stable interval. In order to solve this problem we can divide the stability value by the square of *support* and let $newstable = stable/support^2$.

However we will loose the anti-monotone property, so this will only be useful as a post-processing step. We choose to divide by $support^2$ because stable can maximally become

$$\binom{support}{2} + 2 \cdot (support - 2) < support^2$$

In such a way we remove the influence of a high support on stability.

3.5 Results and Performance

The experiments were done for three main reasons. First of all by using the synthetic dataset we *show that patterns with a stable interval will be found.* Secondly with the `website` dataset we *show that the algorithm also finds good stable patterns for real problems.* And finally with a synthetic dataset and with the `website` dataset we *want to examine the efficiency of the algorithm compared to normal* Eclat. Of course the normal Eclat algorithm only finds frequent itemsets and not stable patterns. However the goal is to

show the influence of the search for stable patterns on speed. Our implementation of ECLAT that discovers stable patterns is called STABLECLAT. All experiments were done on a Pentium 4 2.8 GHz with 512MB RAM.

The synthetic datasets can be seen as a supermarket that sells newspapers and credits for cell phones. The combination of the two is sold every day in the morning at least x times. The first dataset contains 1,000 transactions and 110 items. Of these 110 items 10 occur every 4 transactions. Also each item has a support of 200. From here on we call this dataset `news&credit small`. The second dataset contains 5,000 transactions and 110 items. Of these 110 items 10 occur every 10 transactions. Also each item has a support of 1000. From here on we call this dataset `news&credit large`. The STABLECLAT algorithm was also tested on a real dataset. This dataset is based on an access log of the website of the Computer Science department of Leiden University, as said before. It contains all 1,991 items of the webpages that were visited, grouped in one hour blocks, so each of the 744 transactions contains the pages visited during one hour. This dataset will be called `website` dataset.

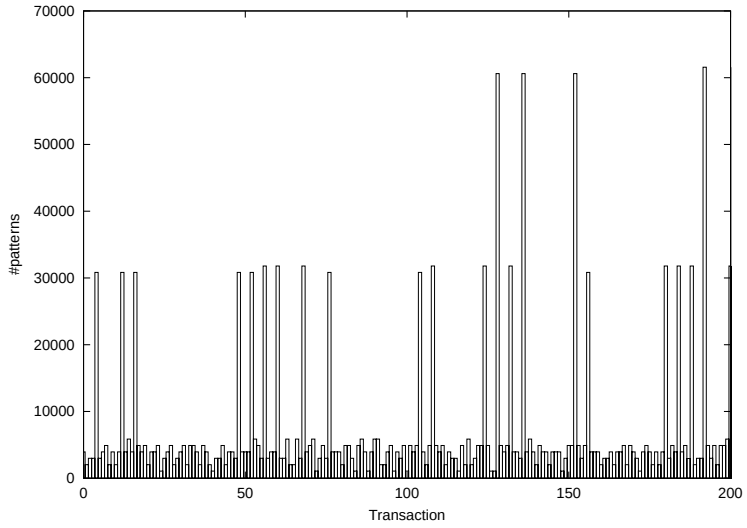

Figure 3.1: Occurrence graph of `news&credit small` dataset using traditional support ($minsup = 25$)

Figure 3.1 and Figure 3.2 show an *occurrence graph*. For each transaction the number of frequent patterns contained by it are counted and plotted. The use of a minimal stability threshold will give less patterns because it

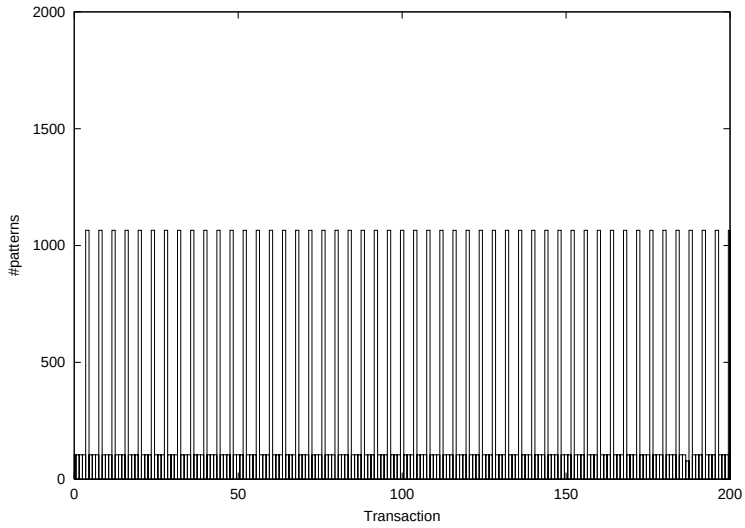

Figure 3.2: Occurrence graph of `news&credit small` dataset using traditional support *and* stability ($minsup = 25$, $dist = 20$)

filters out non-stable patterns. The items in the `news&credit small` dataset all occur 200 times, so their support is 200. Usually transactions are made up out of 20 randomly selected items. However every 4 transactions we randomly select 10 items and the remaining 10 items are the items that will be in the stable pattern. In this way there are several stable patterns, but also many unstable ones. Figure 3.1 shows that with only support we will not discover these stable patterns easily. When we use a minimal stability threshold (with the *dist* parameter from Section 3.3) we are able to see these patterns as we show with Figure 3.2. In all experiments we let $mindepth = 2$.

Figure 3.3 shows the occurrence of one pattern in the first 100 transactions from the `website` dataset. First we searched the patterns with minimal stability $\binom{744/7}{2}$. Then we selected one pattern where the fraction $stable/support^2$ is maximal. The figure shows a regularity in the occurrence of the selected pattern.

Table 3.1 gives an indication of the influence of stability calculation on the speed of the algorithm. The `news&credit large` dataset shows a large slowdown because of more frequent patterns. The occurrence of many frequent patterns means that more combinations have to be checked for a pattern occurring in the center transaction.

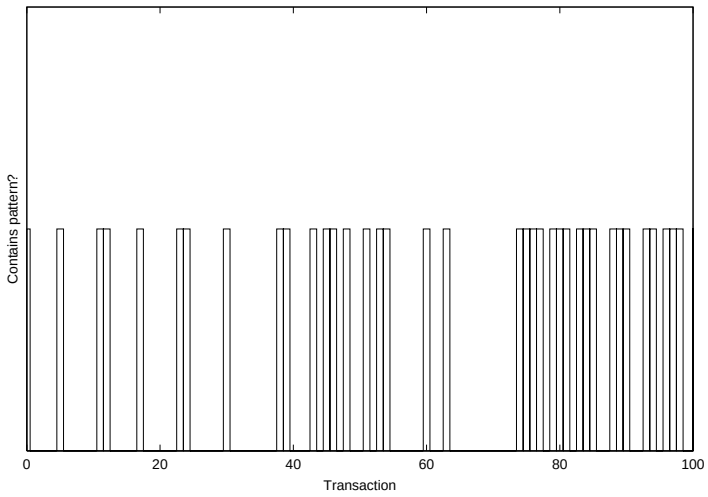

Figure 3.3: The occurrence of one stable pattern for `website` dataset ($minsup = 100$, $dist = 7$)

	`news&credit small`	`news&credit large`	`website`
support only	1	36	179
with stability	2	140	251

Table 3.1: Time in seconds needed to mine each of the three datasets ($minsup = 100$, $dist = 4$)

3.6 Conclusions

When we use stability in our search for patterns, we are able find patterns that occur with some regular interval. The measure we proposed in this chapter still enables us to prune using anti-monotonicity.

Using the distance between transactions like it is done in this chapter is an interesting area of research. In the future we want to examine new measures that would enable us to visualize other types of behavior. Also we want to see if we can speed up the search for stable patterns, e.g., by using heuristics or by improving the stability measure. Furthermore, we would like to compare our approach with post-processing methods.

4 Mining Balanced Patterns

In many applications it will be useful to know those patterns that occur with a balanced interval, e.g., a certain combination of phone numbers are called almost every Friday or a group of products are sold a lot on Tuesday and Thursday.

4.1 Introduction

In the previous chapter we proposed a new measure of support (the number of occurrences of a pattern in a dataset), where we count the number of times a pattern occurs (nearly) in the middle between two other occurrences. If the number of non-occurrences between two occurrences of a pattern stays almost the same then we call the pattern stable.

It was noticed that some very frequent patterns obviously also occur with a stable interval, meaning in every transaction. However more interesting patterns might occur, e.g., every three transactions. Here we discuss a method for finding such patterns using standard deviation and average. Furthermore we propose a simpler approach for pruning patterns with a balanced interval, making estimating the pruning threshold more intuitive.

In this chapter we will further investigate one of the simplest structures: itemsets. However the principles of balanced patterns are easily extended to sequential pattern mining, tree and graph mining. In Chapter 3 we proposed an algorithm that discovers *stable patterns* that occur at regular moments, or rather in regular intervals, enabling us to mine for events that occur, e.g., every Friday. In this chapter we will introduce a new approach to mining for patterns with a stable interval. Note that the transactions in this chapter

have an order. In order to distinguish it from stable patterns we will call these new patterns *balanced patterns*. With this new approach we will offer solutions for problems in our work done in [25]:

- Patterns occurring in every transaction made it hard to discover patterns with a more interesting intermediate interval.
- The threshold for pruning was a certain value that a measure for stability needed to achieve. Even though a formula was given to estimate this value, an easily understandable value was lacking.

We will define our approach to mining balanced patterns and show its usefulness. To this end, this chapter makes the following contributions:
— We will **define balanced patterns and show their use**. These balanced patterns will enable the user to better filter uninteresting patterns (Section 4.2).
— Furthermore we will **propose an algorithm** that will enable us to mine balanced patterns (Section 4.3).
— Finally we will empirically show that **the algorithm can find interesting patterns** efficiently (Section 4.4).

Again a typical example is the mining of an access log from the Computer Science institute of Leiden University. In this chapter `website` dataset will have transactions of half-hour blocks as opposed to hour blocks.

4.2 Definition

In this section we will define balanced patterns. We first discuss several problems and possibilities, and finally give the proper definition. We call the occurrences balanced if between two successive occurrences there is (almost) always the same amount of transactions.

The problem with patterns with balanced occurrences is that an itemset may occur less balanced than a superset of this itemset. Patterns occurring with a balanced interval do not have the *anti-monotone property*, where the subset is either equally good or better than the superset. In the balanced pattern case: the subset is not always more (or equally) balanced than the superset.

Example 4.2.1 Say that item A occurs in transactions 1, 4, 7 and 10 and item B occurs in transaction 4, 7, 10 and 13 then the itemset $\{A, B\}$ will occur in transaction 4, 7 and 10. Both A and B have three times two transactions between occurrences (successive and non-successive). However $\{A, B\}$

has only two times two transactions between occurrences because an occurrence can only become a non-occurrence and not the other way around.

For our definition of balanced patterns we first notice that all balanced occurrences (successive and non-successive) should have at least one intermediate distance a minimal number of times. Furthermore if you count the distances *between all occurrences* then this count is anti-monotone: a superset never has more of one particular distance. This is obvious because the number of occurrences will never increase for a superset and as a consequence the count of one particular distance will never increase. This property is also anti-monotone if we limit the distances we count, e.g., we count a distance only if it is smaller than 10 in-between transactions.

Example 4.2.2 The following table, where we only count upto 4 in-between transactions, is an example of counting the distances:

In-between Transactions (Distance)	Count
0	0
1	5
2	200
3	30
4	199

The *balanced value* for the pattern with these counts will be 200, the highest count in the table.

Still if we only look at the distance count we will not find the balanced patterns we want, since patterns that occur with very unbalanced intervals might still have a minimum amount of one particular distance. We filter those patterns by keeping the distance between occurrences that immediately succeed each other (instead of taking all distances). If a pattern is balanced then these distances should approach the average of all these distances. Their standard deviation will be near 0, since one distance should occur the most. Note that in calculating the standard deviation we do not limit the distances we consider. This can be done because the number of possible distances is far less for successive occurrences.

Now we can find all balanced patterns, however we will still find many patterns that are occurring every transaction. Their distance is almost always 0 and although they are well balanced they are often not interesting.

These patterns can be filtered if we demand a certain average distance, e.g., if the user-defined threshold *minavg* is set to 1 then all these patterns will be filtered out, since their average distance approaches 0.

The definition of balanced patterns should be the following: A pattern is called a *balanced pattern* if among all occurrence pairs there is a distance that occurs at least a user-defined number of times (*minnumber*) and the distance between successive occurrences have maximally a user-defined standard deviation (*maxstdev*) and minimally a user-defined average (*minavg*).

4.3 Algorithm

We now consider algorithms that find all frequent itemsets, given a database. A *frequent* itemset is an itemset with support at least equal to some pregiven threshold, the so-called *minsup*. Thanks to the APRIORI property many efficient algorithms exist. However, the really fast ones rely upon the concept of FP-TREE or something similar, which does not keep track of in-between distances. This makes these algorithms hard to adapt for use in balanced patterns.

One fast algorithm that does not make use of FP-TREEs is called ECLAT [77]. ECLAT grows patterns recursively while remembering which transactions contained the pattern, making it very suitable for balanced patterns. In the next recursive step only these transactions are considered when counting the occurrence of a pattern. All counting is done by using a matrix and patterns are extended with new items using the order in the matrix. This can easily be adapted to incorporate balance counting.

Our algorithm BALANCECLAT will use the ECLAT algorithm. However instead of counting support we count the different distances between all occurrences, e.g., pattern A has 10 times 3 transactions, without the pattern A, between occurrences. We will prune on this value instead of pruning on the minimal support threshold. In this case the user-defined threshold will be the minimal number of times at least one of $\ell + 1$ distances $\{0, 1, 2, \ldots, \ell\}$ is seen. For balanced patterns we consider this threshold to be the *minnumber* threshold. As said before, we can only find balanced patterns if we also demand a maximal standard deviation for distances between occurrences. This will be done by introducing the *maxstdev* threshold. Finally we are not interested in patterns occurring in every transaction. We introduce a third user-defined threshold that demands a minimal average distance: *minavg*. For *maxstdev* and *minavg* we only use distances between successive occurrences and for *minnumber* all distances $\leq \ell$.

We now propose a more general definition. Suppose we have an itemset I and let $O_j \in \{0,1\}$ $(j = 1, 2, \ldots, r)$ denote whether or not the j^{th} transaction in some subset $\mathcal{S}$ of the database $\mathcal{D}$ contains I (O_j is 1 if it does contain I, and 0 otherwise; the O's are referred to as the *O-series*), $r = |\mathcal{S}|$. The function $\varphi : N \to N$ is a translation from the index j for the j-th transaction in $\mathcal{S}$ to the index k giving the position of the same transaction in $\mathcal{D}$.

The main adaptation to ECLAT is replacing support with a *balance value* denoted with t. Also it calculates the standard deviation (*stdev*) and average distance (*avgdist*) for the successive occurrences:

$j := 2$, $h := -1$
succdists := sequence of distance counts between *successive* occurrences
alldists := sequence of distance ($\leq \ell$) counts between *all* occurrences
while ($j \leq r$) **do**
  if ($O_j = 1$) **then**
    $i := 1$
    while ($i < j$) **do**
      if ($O_i = 1$ **and** $\varphi(j) - \varphi(i) - 1 \leq \ell$) **then**
        $alldists_{\varphi(j)-\varphi(i)-1} := alldists_{\varphi(j)-\varphi(i)-1} + 1$
      fi
      $i := i + 1$
    od
    if ($h \neq -1$) **then**
      $succdists_{\varphi(j)-\varphi(h)-1} := succdists_{\varphi(j)-\varphi(h)-1} + 1$
    fi
    $h := j$
  fi
  $j := j + 1$
od
$t := max(alldists)$, the largest count in the sequence
stdev := standard deviation for *succdists*
avgdist := average for *succdists*, also denoted with $avg(succdists)$

The standard deviation for *succdists* can simply be calculated in the following way:

$$\sqrt{\textstyle\sum_i (avg(succdists) - i)^2 \cdot succdists_i \;/\; \sum_i succdists_i} \qquad (4.1)$$

ECLAT can now prune using the balance value t (if $t < minnumber$) and patterns are only displayed if their standard deviation and average distance

are sufficient. These are straightforward adaptations that will not be given in detail.

Standard deviation changes if patterns occur less balanced in a certain small number of successive transactions, small periods. In some cases it might be preferable to remove the influence of these periods. One possible approach is to calculate average distance and the standard deviation for *frequent distances* (for successive occurrence) only. The value for filtering with standard deviation for the sequence $\mathcal{Q} = \langle y | y = succdist_i, y \geq mindistfreq \rangle$ will be:

$$stdev = \begin{cases} \sqrt{\sum_i (avg(\mathcal{Q}) - i)^2 \cdot \mathcal{Q}_i \;/\; \sum_i \mathcal{Q}_i} & \text{if } \mathcal{Q} \text{ is } \textit{not} \text{ empty} \\ maxstdev + 1 & \text{otherwise} \end{cases} \tag{4.2}$$

Note that via the threshold *mindistfreq* the user decides when a distance is considered frequent.

4.4 Results and Performance

The experiments were done for three main reasons. First of all we want to show *known balanced patterns will be found* also in the case of noise. Secondly we want to show that *interesting balanced patterns can be found* in real datasets. Finally we want to *show runtime for real data* and *how the minnumber threshold influences runtime.*

Our implementation of the balanced pattern mining algorithm is called BalanceClat. 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.

The synthetic datasets used in our first experiment below are called `find-noise-x%` where `x` is a noise value ranging from 0 to 30. E.g., if the noise is 10%, this means there is a 10% chance for each item of the balanced pattern to not occur when it should and a 50% probability to still occur because of random chance (like the other items that are not part of the balanced pattern).

In each of these `find-noise-x%` datasets one pattern of 5 of the 200 items occur every 4 transactions (so distance = 3) and each dataset has 2,000 transactions. Furthermore the remaining items have a probability 50% to occur. If 5 items always occur balanced like this, we expect to find $\sum_{k=1}^{5} 5!/(5-k)!k! = 31$ patterns.

The first real dataset we test our algorithm on is called the `website` dataset. This dataset is based on an access log of the website of the Computer

Science department of Leiden University, as said before. It contains all 1,991 items of the web-pages that were visited, grouped in half-hour blocks, so each of the 1,488 transactions contains the pages visited during one half-hour.

The second real dataset we call the `one-user` dataset and it stores the webpages accessed by one heavy user of the `portalexecutivo.com` website. Each day is one transaction of pages accessed. Some days there is no access and some of the 1,603 transactions are empty. Webpages are categorised resulting in a total of 185 possible items for every transaction.

First the BalanceClat algorithm is executed with *maxstdev* = 2.5, *minavg* = 2.0 and *minnumber* = 150. Figure 4.1 displays the number of expected patterns that were found by the algorithm. We see that the algorithm detects most patterns up to a noise level of 15%. Due to the way we generate noise, long patterns become less likely as the noise level increases. With a high noise level we only find the patterns of 1 item in length. This can be improved if we change our settings for *maxstdev* and *minavg*, but we kept them fixed for comparison reasons.

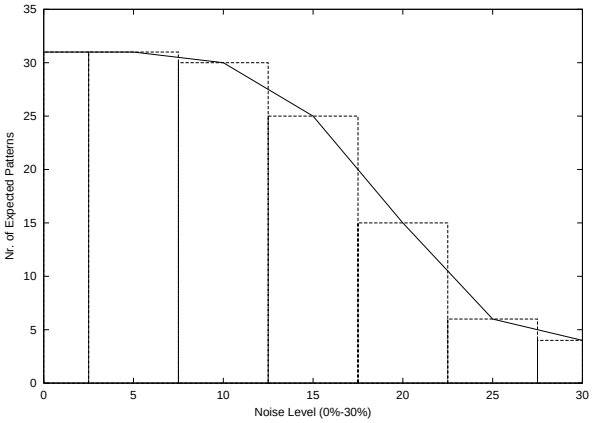

Figure 4.1: The effect of noise on the algorithm using the `find-noise-x%` datasets.

We can use the *mindistfreq* threshold to decrease the influence of small noisy periods on the balanced occurrences. Figure 8.3 shows how the effect of noise becomes less if we set a *mindistfreq* of 50. Now one also finds more of the other patterns that happen to occur reasonably balanced, however we can filter them by lowering *maxstdev*.

With our next experiment we want to show the effect of dataset size on the algorithm, scalability. To this end we measured runtime for different

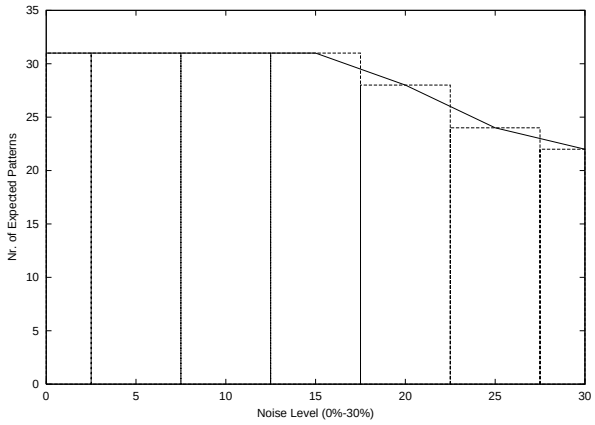

Figure 4.2: The effect of noise on the algorithm using the `find-noise-x%` datasets with *mindistfreq* = 50.

size of a dataset where each transaction can contain up to 200 items where 5 items occur every 4 transactions and the remaining items have a probability 50% to occur. In Figure 4.3 first the runtime drops; this is because many patterns have distances occurring only a few times. E.g., when the dataset size is 100 then $minnumber = 0.1 \cdot 100 = 10$. Many patterns have distances that occur at least 10 times. As this effect becomes less, runtime increases and eventually it becomes nearly linear.

Figure 4.4 shows how the runtime for the `website` dataset drops fast as *minnumber* increases. Figure 4.5 also shows a drop of runtime for the `one-user` dataset.

Many patterns in the `one-user` dataset occur mostly unstable and only some occur stable in such a way that the standard deviation of the interval does not suffer too much (becomes more than *minavg*). One pattern that was found was the access of research and training part of the website on the same day every seven days, see Figure 4.6. Also this pattern lasted for more than one month.

Table 4.1 shows the count for distances between successive occurrences. It shows that this particular pattern, consisting of the websites of two professors of the same group and the main page, occurs often with a successive distance of 0, 1 or 2. This pattern probably is caused by students having courses from both professors and some of these students access both pages nearly every half an hour.

Finally we also applied the BalanceClat algorithm to the `Nakao` dataset

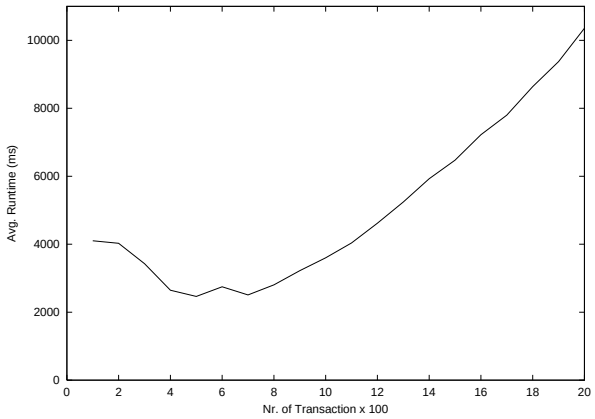

Figure 4.3: Runtime in ms for different dataset sizes; *minnumber* is 10% of the dataset size ($maxstdev = 1.0$, $minavg = 2.0$, $\ell = 10$).

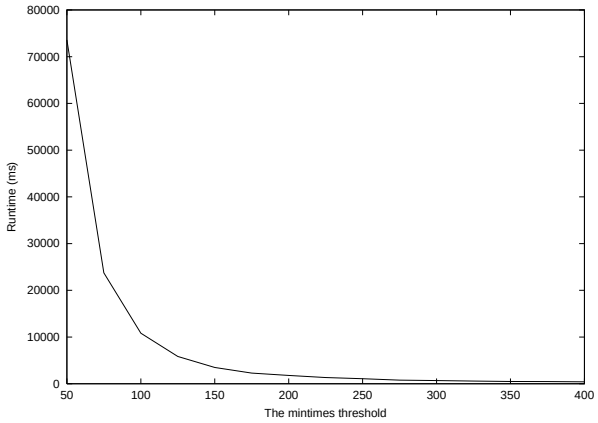

Figure 4.4: Runtime in ms for different values of *minnumber* for the `website` dataset ($maxstdev = 1.0$, $minavg = 2.0$, $\ell = 10$).

used in [29]. In this dataset each of the 2,124 transactions is a clone located on the human chromosomes. The items are the numbers of patients with a higher than normal value for this clone (≥ 0.225). The specifics of the dataset can be found in [53]. The parameter *minavg* was set 0.0, because the interesting patterns are expected to occur very close to each other. Also $mindistfreq = 10$ because patterns where expected to have small periods of

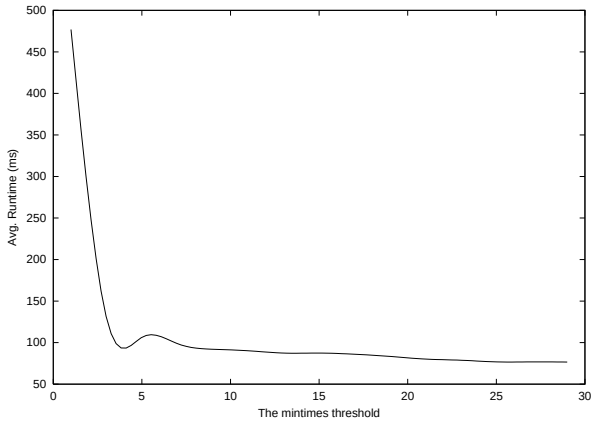

Figure 4.5: Runtime in ms for different values of *minnumber* for the `one-user` dataset ($maxstdev = 5.0$, $minavg = 2.0$, $\ell = 10$).

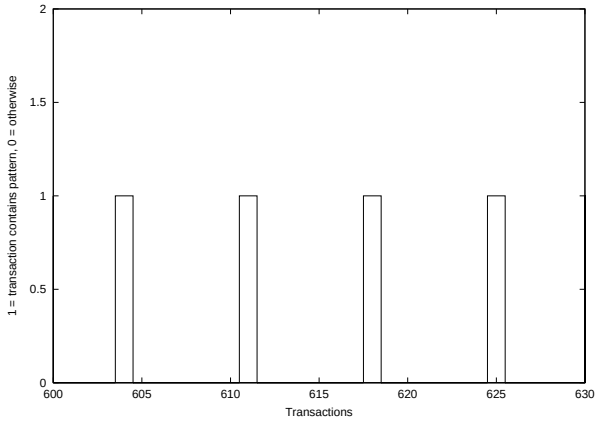

Figure 4.6: The occurrence of one pattern discovered with BalanceClat in the `one-user` dataset ($minnumber = 20$, $maxstdev = 3.0$, $minavg = 1.5$, $\ell = 7$).

transactions where they occurred unbalanced. Furthermore $maxstdev = 0.2$, $\ell = 10$ and $minnumber = 100$. Results where similar to results found with consecutive support as presented in [29] where most consecutive patterns occurred close together in chromosome 9, see Figure 4.7. However consecutive support is different in that it looks at all gap sizes between occurrences

In-between Transactions (Distance)	Count
0	385
1	171
2	78
3	25
4	23

Table 4.1: The distances (with count ≥ 20) between successive occurrences and their counts for one pattern (two professors & the main page) in the website dataset ($maxstdev = 2.0$, $minavg = 1.0$, $\ell = 10$).

and it does not have to keep a count for all possible distances.

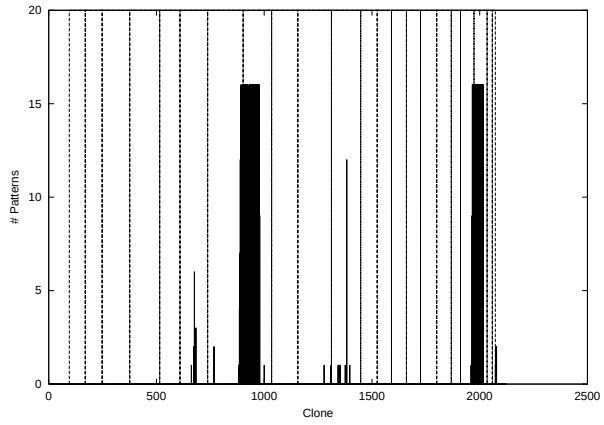

Figure 4.7: The occurrence of one pattern discovered with BalanceClat in the `Nakao` dataset ($minnumber = 20$, $maxstdev = 0.2$, $minavg = 0.0$, $mindistfreq = 10$, $\ell = 10$).

4.5 Conclusions

We have presented a new way of mining for patterns occurring with a regular interval. In comparison with stable patterns we now use a pruning threshold *minnumber* that is more intuitive to users. With it the user only indicates the number of times at least one intermediate distance should occur. Such

a distance is the number of transactions between two occurrences of the pattern (we consider only distances below a maximal distance). This has advantages for the intuitiveness of the pruning method. However the disadvantage compared to the stability measure is that we have to limit the different intermediate distances we count.

We called patterns with a regular interval balanced and we discussed an algorithm to find them efficiently. Its runtime performance and scalability has been evaluated through experimentation.

Finally in the future we plan to use balanced patterns in combination with new ways of filtering to facilitate the discovery of new patterns further. Also research will be done on effectively visualizing balanced patterns.

5 The Most Discriminating Patterns and Domain Knowledge

We investigate the discovery of sequential patterns for use in classification. We will define variations of a fitness function that enables us to tell if one pattern is "better" than another. Furthermore we will show how domain knowledge can be used for faster discovery of better sequential patterns in specific types of databases, in our case a receptor database.

5.1 Introduction

Sequence analysis has many application areas, such as the analysis of protein sequence and customer behavior. We investigate extraction of features for protein sequence classification where features are *sequential patterns*: ordered lists of items (for proteins the items are amino acids). As a motivating example, we would like to know if a protein sequence, an ordered list of amino acids, belongs to the Olfactory family or not, where the Olfactory family is a group of proteins than deals with smell. We focus on a special group of proteins called GPCRs. These G-protein-coupled receptors (GPCRs) play fundamental roles in regulating the activity of virtually every body cell [70]. Usually classification is done unsupervised using alignment, however in the case of GPCRs this turned out to be difficult. Fortunately, we know for some protein sequences whether they are of the Olfactory family or not. These sequences can thus be divided into two disjoint classes: Olfactory and Non-olfactory, and from these classes we can extract sequential patterns to be used as attributes in a classification algorithm (as is being proposed in [45]). Then question we try to answer in this chapter is which

sequential patterns are *best* used as features/patterns? And how can domain knowledge be used to improve the search for such patterns?

Classification based on sequential patterns is also applicable in many other areas. For example, in the case of customer behavior analysis, we might want to characterize groups of clients based on sequential patterns in their behavior.

Our algorithms will be based on the pattern growth approach called PrefixSpan proposed in [57]. Classification by means of patterns has been done before but not so much in the sequence domain. We first mention related work in the non-sequence domain. Apriori-C [36] constructs classification rules by extending the Apriori algorithm [3, 4]. Apriori-C discovers a large number of rules from which a fixed number of rules with the highest support are selected. Apriori-SD [37] solves the problem of selecting the right rules with *subgroup discovery*. This algorithm selects a subgroup of rules by calculating their *weighted relative accuracy*. This means that the probability of a pattern occurring in a class is compared with the probability of its occurrence outside the class. This is weighted with the probability of a class. Most class association rule mining algorithms work with unordered sets of items frequently occurring together in *item sets*. Classification with association rules is presented in [48] and [49]. Furthermore CorClass [78] describes an algorithm that also works with item sets. It introduces a new method of pruning. Specialized rules are only added if the upper bound of its correlation is higher than the minimal correlation of k rules. In our work we use a similar method of pruning. Much work has been done in the field of molecular feature mining, e.g., the MolFea algorithm described in [40]. MolFea employs a level-wise version space algorithm to discover those molecule fragments often occurring in one dataset and less often in another. Finally some other researchers try to use domain knowledge to speed up the search for frequent patterns, e.g., the Carpenter algorithm presented in [12]. In this work the authors perform row enumeration instead of the standard column enumeration done in Apriori-like algorithms. This is done because biological datasets often have many columns/items and only a few rows.

The "best" sequential patterns are discovered through a function that judges patterns. In Section 2 we will discuss different instances of this function and select one for our purposes. Section 3 adapts the PrefixSpan algorithm of [57] to deal with this function. In addition, a pruning strategy is introduced in Section 4, increasing efficiency by first searching in a certain area of the sequence, the *probable time window*. Section 4 also describes how preferring small patterns can further increase classification performance. The

effectiveness of these improvements will be shown in Section 5.

5.2 The Maximal Discriminating Patterns

One would like to select the best patterns for use as attributes in a classification algorithm. But how can we tell if one pattern is better than the other? In this section we will first explain the notion of support and why it is less useful for selecting the best pattern. Next we introduce the notion of confidence which will give more useful patterns, but it also has disadvantages. Finally we will discuss and motivate so-called maximal discriminating patterns, enabling us to have patterns specific to one class, but without the disadvantages of confidence.

Assume given a database D with $D = D_1 \cup D_2 \cup \ldots \cup D_c$, with c classes. The D_i's $(1 \leq i \leq c)$ are mutually disjoint and non empty.

Each record in the database is a non-empty finite *sequence* (i.e., an ordered list) of items from the set $\Sigma = \{\texttt{A},\texttt{B},\texttt{C},\ldots\}$, e.g., (C,B,G,A,A,A,C,B). Now fit_0 is defined as support (as used in association rule mining algorithms like Apriori [4]), because support can be seen as a measure of how well a pattern fits the data. Commonly a sequence d is said to support a *pattern* s if the pattern is contained (in the "subset" sense) in the sequence:

$$supp_0(s,d) = \begin{cases} 1 & \text{if for all } i\ (1 \leq i \leq k) \text{ there is } j\ (1 \leq j \leq \ell) \text{ with } s_i = d_j; \\ 0 & \text{otherwise,} \end{cases}$$

for $s = (s_1, s_2, \ldots, s_k)$ and $d = (d_1, d_2, \ldots, d_\ell)$. This means that s is a *subset* of d. We then can define fit_0:

$$fit_0(s, D_i) = \frac{1}{|D_i|} \sum_{d \in D_i} supp_0(s,d)$$

For $(1 \leq i \leq c)$, where s is a pattern.

We now specialize support to sequences. A sequence $d = (d_1, d_2, \ldots, d_m)$ is called a *super-sequence* of a sequence $s = (s_1, s_2, \ldots, s_k)$ if $k \leq m$ and for each s_i $(1 \leq i \leq k)$ there is a d_{j_i} $(1 \leq j_i \leq m)$ with $s_i = d_{j_i}$ and $j_{i-1} < j_i$ $(i > 1)$. We denote this with $s \prec d$. The sequence s is called a *sub-sequence* of d. This defines *sequential patterns* on sequences of items. (Another definition of sequential patterns was given by Agrawal et al. in [4], in which they define sequential patterns on sequences of item sets). We now let

$$supp_1(s,d) = \begin{cases} 1 & \text{if } s \prec d; \\ 0 & \text{otherwise,} \end{cases}$$

and define fit_1 in the same way as fit_0 was defined using supp_0.

Now fit_1 or fit_0 by itself is not useful for selection of features for classification. One of the patterns of size one will always have the highest fit and these small patterns are probably often present in more than one D_i. Thus the presence of such a pattern will not give a good distinction between classes.

The next most logical step is to use confidence to select the best patterns. The patterns x_r $(1 \leq r \leq c)$, one for each class, are then chosen to maximize *confidence*:

$$\mathit{confidence} = \frac{\mathit{fit}_1(x_r, D_r)|D_r|}{\mathit{fit}_1(x_r, D)|D|} \tag{5.1}$$

The class t of sequence s is the t $(1 \leq t \leq c)$ where $x_t \prec s$. If more than one t is possible we select based on the highest confidence. One is selected at random if more than one class t has a pattern with the highest confidence. If there is no t where $x_t \prec s$ then the sequence could be said to be "undecided".

A problem is that we only pick one pattern per class. This is plausible if a class of a sequence is only decided by one sequence of features. However, it is often the case that the class of a sequence is determined by multiple patterns. Moreover there can be constraints on the pattern. This means that the "class deciding" pattern x_t *with* the constraint is not necessarily equal to the x_t *without* the constraint. As a consequence it is usually possible to find a combination of patterns with a better classification performance. Finally it is possible that a single sequential pattern x_t is equal for two or more classes, and as a consequence a choice between the classes will be done at random. This problem will occur with a lower probability if we use multiple patterns for each class.

Another major drawback of the confidence method is that the size of the D_i's seriously influences the classification. E.g., assume we have databases D_1 and D_2. Furthermore assume D_1 contains 500 sequences and D_2 only 100. The pattern p_1 occurs 100 times in D_2 and 60 times in D_1, thus a confidence with respect to D_2 of 0.625. Another pattern p_2 occurs 70 times in D_2 and 10 times in D_1, giving a confidence of 0.875. The pattern p_2 will be used for classification if no other pattern has a higher confidence. However p_1 occurs in every sequence of D_2 and only in a small percentage of the sequences in D_1. However, one could argue that p_1 should be preferred over p_2.

Therefore we define fit_2, which we use in the sequel. For a pattern s and $1 \leq q, r \leq c$ we define $\delta(s, D_q, D_r) = \mathit{fit}_1(s, D_q) - \mathit{fit}_1(s, D_r)$, and we

let $fit_2(s, D_r) = \min\{\delta(s, D_r, D_q) \mid 1 \leq q \leq c \ \wedge \ q \neq r\}$. We then choose patterns x_r $(1 \leq r \leq c)$ with maximal $fit_2(x_r, D_r)$. We can then use them to classify sequences as before, without the drawbacks mentioned above. We will usually find those patterns that are characteristic for one class. With *characteristic* we mean that fit_1 will have a high value in D_t and a lower value in the other $D_i's$, $i \neq t$.

In [44] other measures for difference, correlation measures, where discussed. Nijssen et al. discuss the use of the $\mathcal{X}^2$ measure and it is proven how pruning is also possible with this measure with two or more classes (the databases D_r). Furthermore it is shown how pruning is not possible when we use information gain as a correlation measure. We make use of the $\delta(s, D_q, D_r)$, as defined earlier. The simplicity of $\delta(s, D_q, D_r)$ makes explanation of the use of domain knowledge to speed up the search for the most discriminating patterns (correlated patterns) easier.

Our new fit has some similarities with the concept of *emerging patterns* presented in [6] and [14]. In order to discover emerging patterns patterns are preferred where the ratio $fit_1(s, D_1)/fit_1(s, D_2)$ is the highest, where D_1 and D_2 are two databases each containing one class of sequences. Bailey et al. [6] further investigate jumping emerging patterns. These are patterns that have a support of zero in D_2 and a non-zero support in D_1. Emerging patterns can also be defined in a way similar to fit_2, but now using $fit_1(s, D_q)/fit_1(s, D_r)$ instead of $\delta(s, D_q, D_r)$. Dong et al. [14] point out that the growth rate measure used by emerging patterns does not take into account the coverage, a problem they solve with a score function. However in the case of fit_2 coverage is less of a problem, a pattern with a low $fit_1(s, D_1)$ is less likely to have a high fit_2 value. Also the fit_2 measure allows us to more easily explain and implement the pruning rules that will be discussed in the remainder of this paper.

In general terms, most of the classification algorithms perform better when dealing with small to moderate size attribute sets. In order to classify a sequence s we use a finite number of n sequential patterns $p_1^t, p_2^t, \ldots, p_n^t$ per class t, where $fit_2(p_1^t, D_t) \geq fit_2(p_2^t, D_t) \geq \ldots \geq fit_2(p_n^t, D_t)$ and p_n^t has the n-th highest fit_2 for all possible patterns. These patterns, the so-called *maximal discriminating patterns*, could be used by any classification algorithm when we first convert each sequence to a vector indicating for each pattern if it is contained in the sequence, see [45]. However it is possible that, e.g., p_1^t is supported by all or most of the sequences supporting p_2^t. Thus p_2^t might not improve classification. This problem could be solved by removing all sequences containing p_1^t from D_t. The algorithm for searching the sequence with maximal fit is then again applied to this subset of D_t in order to

find p_2^t. In this paper we do not further focus on the precise classification performance, but rather on the discovery of the discriminating patterns. Our algorithm aims at finding the set $P = P^t$ of maximal discriminating patterns.

5.3 Algorithm without Domain Knowledge

Our pattern search algorithm, coined PrefixTWEAC (Time Window Exploration And Cutting), is based on PrefixSpan. The algorithm does not generate candidates, but it grows patterns from smaller patterns. This principle makes it faster than most Apriori like algorithms [57]. PrefixSpan is a depth first algorithm, which will be explained in more detail in Section 5.4 when we adapt this algorithm to our current needs (see Table 5.1 and Table 5.2). PrefixSpan as described in [57] searches for those patterns with *support* larger than or equal to a given support threshold *minsupp*, where support is defined as fit_1. The algorithm starts with all frequent subsequences of size one. For each sub-sequence a projected database is created. These frequent sub-sequences are extended to all frequent sub-sequences of size two by only looking in the projected database. This *projected database* is a database of pointers to the first item occurring after the current pattern, also called the *prefix*. A sequence is only in the projected database if it contains the prefix. Again for each frequent sub-sequence of size two a corresponding projected database is created. This process continues recursively until no extension is frequent anymore.

PrefixTWEAC (Table 5.1) is different from PrefixSpan in that it searches for the maximal fit_2 instead of the maximal support fit_1. The function fit_2 is by definition not anti-monotone (so $fit_2(s_1, D_t) > fit_2(s_2, D_t)$ might happen, where s_1 is a super-sequence of s_2). However the anti-monotone property for fit_1 can still be used in two ways, when looking for the one pattern with maximal fit_2. First of all in PrefixTWEAC we only examine an extended pattern p if $fit_1(p, D_t) \geq minsupp$ where *minsupp* is the support threshold. Secondly p is not further examined if $fit_1(p, D_t) <$ *current n-th maximal fit*, where *current n-th maximal fit* is the current n-th best fit of all patterns found while searching. The value of $fit_2(p, D_t)$ will never become larger than the current n-th maximal fit, because it can at most become $fit_1(p, D_t)$. Note that CorClass uses similar methods to prune [78].

PrefixTWEACCore(prefix, projected_database)
1. For all items i that can extend the prefix
2. new_prefix = prefix extended with item i
3. Count $w_1 = \mathit{fit}_1$ in the projected_database$_t$ for new_prefix
4. Calculate $f_2 = \mathit{fit}_2$ for new_prefix
5. Create a projected database new_projected_database with new_prefix
6. Get δ_{min}, the lowest fit_2 in P
7. Get s_{min}, fit_1 corresponding with the lowest fit_2 in P
8. **if** $w_1 \geq \mathit{minsupp}$ **and** $|P| < n$ **then**
9. Add new_prefix to P
10. Call PrefixTWEACCore(new_prefix, new_projected_database)
11. **else if** $w_1 \geq \mathit{minsupp}$ **and** $w_1 \geq \delta_{min}$ **then**
12. **if** $f_2 > \delta_{min}$ **or**
13. ($f_2 = \delta_{min}$ **and** $w_1 > s_{min}$) **or**
14. ($f_2 = \delta_{min}$ **and** $w_1 = s_{min}$ **and** new_prefix $\prec p_n$) **then**
15. Remove p_n from P and add new_prefix to P
16. Call PrefixTWEACCore(new_prefix, new_projected_database)

Table 5.1: The PrefixTWEAC algorithm

5.4 Domain Specific Improvements

In the previous section we stated that fit_2 can be used to "prune": certain pattern extensions are not further examined because they can never lead to the maximal fit_2. The faster we get to a large fit_2 for the n-th pattern in $P = P^t$ the better, because all extensions with a lower $\mathit{fit}_1(p, D_t)$ can be pruned. The improved version of PrefixTWEAC will be explained in the sequel.

If we consider protein sequences then pattern discovery might be done faster and/or classification might improve when using certain knowledge about the sequences:

- Protein sequences are sequences of amino acids. Certain parts of such a sequence are shaped like a helix in 3D space. These helices will probably contain most of the maximal fitting sequences since parts outside the helix have more variation in size and content. Patterns (partially) outside the helix are less likely to occur in most members of the protein family.

- Small patterns are preferred. Smaller patterns are less specific and biologists prefer smaller patterns in their analysis.

PrefixTWEACExt(prefix, proj_db)

1. For all items i that can extend the prefix
2. new_prefix = prefix extended with item i
3. Count fit_1 for new_prefix:
4. $w_1 = \mathit{fit}_1$ in the proj_db$_t$ without the inclusion vector, using supp_1
5. $w_2 = \mathit{fit}_1$ in the proj_db$_t$ with the inclusion vector, using $\mathit{supp}_1^{\mathrm{PTW}}$
6. Calculate $f_2 = \mathit{fit}_2$ for new_prefix (without the inclusion vector)
7. Create new_projected_database (without using the inclusion vector)
8. Get δ_{min}, the lowest fit_2 in P
9. Get s_{min}, fit_1 of the lowest fit_2 in P
10. **if** $w_1 \geq \mathit{minsupp}$ **and** $|P| < n$ **then**
11. Add new_prefix to P
12. Call PrefixTWEACExt(new_prefix, new_projected_database)
13. **else if** $(w_1 \geq \mathit{minsupp}$ **and** $w_2 < \mathit{minsupp})$ **or**
14. $(w_2 \geq \mathit{minsupp}$ **and** $w_1 \geq \delta_{min}$ **and** $w_2 < \delta_{min})$ **then**
15. storeState(S,new_prefix, new_projected_database)
16. **else if** $w_2 \geq \mathit{minsupp}$ **and** $w_2 \geq \delta_{min}$ **then**
17. **if** $f_2 > \delta_{min}$ **or**
18. $(f_2 = \delta_{min}$ **and** $w_1 > s_{min})$ **or**
19. $(f_2 = \delta_{min}$ **and** $w_1 = s_{min}$ **and** new_prefix $\prec p_n)$ **then**
20. Replace p_n with new_prefix
21. Call PrefixTWEACExt(new_prefix, new_projected_database)

Table 5.2: PrefixTWEAC Extended: extension using the probable time window

For certain problems we know the approximate area of important features, e.g., protein sequences should have most of the discriminating patterns in the helix. Also in other problems this might be the case, for example — in the case of customer relations — customers tend to behave differently during the night. These *probable time windows* can easily be defined with an *inclusion vector*. An inclusion vector is a vector $v = (v_1, v_2, \ldots, v_n)$, $v_i \in \{0, 1\}$ $(1 \leq i \leq n)$. This vector will indicate where to search in the first phase of

the algorithm, see Table 5.2. We then let

$$supp_1^{\mathrm{PTW}}(s,d) = \begin{cases} 1 & \text{if } s \prec d/v; \\ 0 & \text{otherwise,} \end{cases}$$

where $(d/v)_i = d_i$ if $v_i = 1$ and \$ otherwise ($\$ \notin \Sigma$), so only positions with nonzero v_i are considered.

First PrefixTWEACExt (Table 5.2) is applied to the databases D_t, one at a time, each time starting with an empty $P = P^t$. After using PrefixTWEACExt with the inclusion vector we apply PrefixTWEAC (Table 5.1) without the vector to the remaining states stored in the state database S.

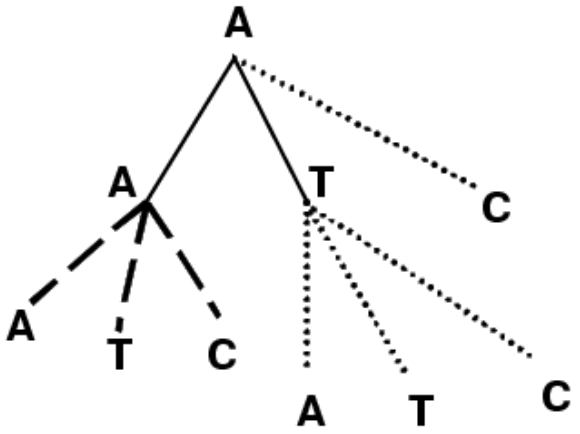

Figure 5.1: Extending the single item sequence A

Figure 5.1 shows an example of the extensions made to a sequence A. The dotted lines are extensions that do not have a high enough *fit₁* and *fit₂* inside and outside the probable time window. These extensions and their extensions are pruned. The dashed lines indicate extensions that are currently good enough with regards to the entire sequence only. Finally the solid lines are already good enough when we only count patterns inside the probable time window.

If we prefer small patterns, then we can add a new rules, using so-called *smallest maximal discriminating patterns* to improve classification:

- $fit_1(s, D_r) = 0$ for all r $(1 \leq r \leq n, r \neq t)$. Then fit_2 of the extended patterns will never increase.

- $fit_1(s, D_t) \leq fit_2(p, D_t)$ where both p and s are sequences and s is created by extending p. Then fit_2 of the extended patterns will never be better than the fit_2 of p.

These rules sacrifice some completeness for classification performance; if extensions do not improve a smaller pattern then they are not always explored further. These pruning rules will not lower classification performance because they leave out only non-improving extensions. Rather the classification is expected to improve because the set of patterns will contain less small variations of the same pattern. We will from now on abbreviate the use of these rules with SP or "small patterns".

Protein sequences usually are very long, about 300 amino acids. However these sequences are constructed out of only 20 types of amino acids. We need to use constraints to make the problem tractable. It was chosen to use the time window constraint, because the discovered patterns will be concentrated in one area. One could also mine *closed patterns*: mine for only those frequent sub-sequences for which there is no super-sequence with an equal support. With CloSpan proposed in [76] it is shown that it is possible to reduce runtime with this principle. This is interesting and it can most likely be used in combination with the time window constraint. However in any case the time window constraint is still needed since patterns spread all over the original sequence (the record or protein sequence) are biologically less interesting.

The *time window constraint* means that the distance between the first and last item of the pattern in the sequence is bounded by some constant. This is easily implemented in the algorithm used. We also considered to use the *gap constraint* [5], that allows some gaps in the matches. However this constraint would have required more memory, e.g., if we count fit_1 of (A,C,G) and we want to know whether the sequence (A,C,C,C,G) contains it. Furthermore assume the maximal gap is 1, thus in the sequence one letter is allowed between two letters of the pattern. If the algorithm only looks at the first C then the gap constraint will be broken because the gap between the C and the G is 2. An algorithm has to check two C's to match (A,C,G). PrefixSpan will have to add both projections to the projected database for at least two C's. Another reason for not using the gap constraint is that it would allow patterns to be spread all over the sequence as long as it does not break the gap constraint.

5.5 Experimental Results

The experiments are aimed at showing the effectiveness of the pruning rules we described. The protein sequences used during our experiments where extracted from the GPCRDB website [22]. The effectiveness was also tested on a `synthetic` dataset: the two classes consist of 1000 sequences of length 130, having 20 item types. First each item is chosen with a uniform probability and then we insert one of ten patterns at each starting position within the time window (position 20 to 60) of class one with 80% probability.

The results are shown in Figure 5.2 and Figure 5.3. All experiments were done on a Pentium 4 2.8 GHz with 512MB RAM. On the horizontal axis in the graphs we have the number of used sequences in the dataset. As both synthetic and protein dataset have two classes, we take one half of these sequences from the first class and the rest from the second class. In the case of the GPCRDB
`Olfactory` dataset the first class contain the Olfactory sequences and the second class the Non-olfactory sequences. Furthermore the GPCRDB `Amine` dataset contains Amine and Peptide sequences. With this data we want to show that some groups of sequences are harder to distinguish. On all the vertical axis we have the pruning effectiveness indicated by a real number between 0 and 1. This effectiveness is calculated by dividing the search time by the worst search time in the experimental results. During the experiments we searched for the 100 maximal discriminating patterns in the GPCRDB and 10 in the synthetic dataset, each with a time window of eight and a *minsupp* of zero. Note that time window and probable time window are different concepts. The experiments on the synthetic data are done to indicate that the probable time window can improve pruning efficiency. Other experiments will show the effectiveness of the method in the case of GPCRDB data.

Figure 5.2 shows the effectiveness of using probable time windows (PTW) of Table 5.2 and pruning when using "small patterns" (SP) on the GPCRDB Olfactory data. The algorithm not using PTW or SP is shown in Table 5.1. Note that SP lowers pruning effectiveness with regards to the GPCRDB Olfactory data, because less variations of the same pattern fill up the set of patterns. Some of the patterns discovered with this dataset were used for classification: these two protein families (Olfactory and Non-olfactory) could be correctly distinguished in more than 90% of the cases, depending on the chosen time window size and the classification algorithm at hand.

In the `synthetic` dataset we have most of the best patterns in the probable time window. The n-th pattern p will get a large fit_2 earlier in the

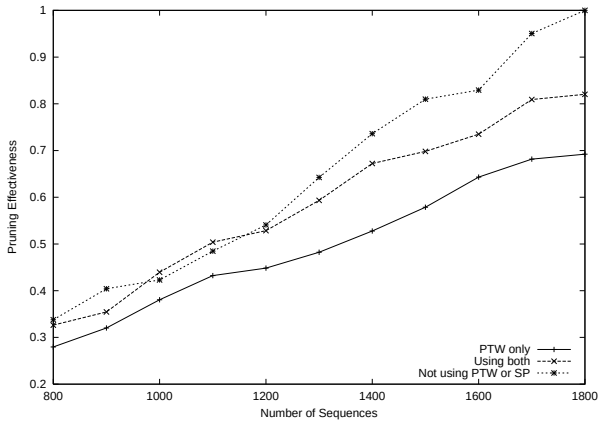

Figure 5.2: Effectiveness on the GPCRDB Olfactory/Non-olfactory data using the `Olfactory` dataset

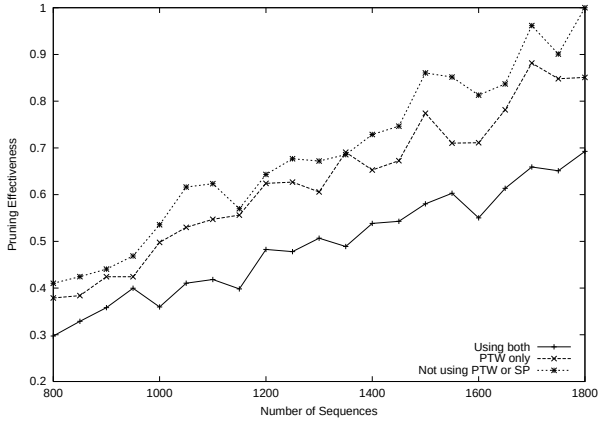

Figure 5.3: Effectiveness on the `synthetic` dataset

search, thus more extensions can be ignored. Figure 5.3 shows the effectiveness as the number of sequences in the `synthetic` dataset increases when searching for the 10 maximal discriminating patterns. The "small pattern" rules (SP) increase the effectiveness even further, because in the `synthetic` dataset many patterns are quickly non-improving.

The confusion matrices of Table 5.3 and Table 5.4 were generated using the C4.5 implementation by Weka [69] with the 10 (Olfactory) and 20

	classified as non-olfactory	classified as olfactory
non-olfactory	2015	22
olfactory	16	1909

	classified as non-olfactory	classified as olfactory
non-olfactory	2024	13
olfactory	22	1903

Table 5.3: Confusion matrices of Olfactory (GPCRDB) patterns without (upper) and with (lower) "small patterns" (SP)

	classified as amine	classified as peptide
amine	489	16
peptide	3	1091

Table 5.4: Confusion matrices of Amine/Peptide (GPCRDB) patterns

(Amine) best patterns discovered in the GPCRDB data. In Table 5.3 we get a slightly better classification in 10-fold cross-validation when using SP: 99.12% instead of 99.04%. This is as expected because the set of 10 patterns used in Table 5.3 will contain less small variations of the same pattern. The results of Table 5.4 required 20 patterns instead of 10. The Amine/Peptide problem is more difficult than the Olfactory/Non-olfactory problem and it requires more patterns. The effect of SP on classification is small, however to show that the difference in classification performance is significant a two-tailed unpaired t-test was performed. Ten-fold crossover with 1999 sequences was done 100 times with two groups of 50 Amine/Peptide patterns, with and without SP, and a time window of 4. The t-value of 6.420 with a probability of less than 0.001 of happening by chance shows that the patterns found with SP classify significantly better when using the C4.5 algorithm with these patterns as attributes.

Figure 5.4 shows less improvement of the pruning effectiveness. This is because the patterns in the probable time window of the Amine sequences are less discriminating compared to the patterns in the probable time win-

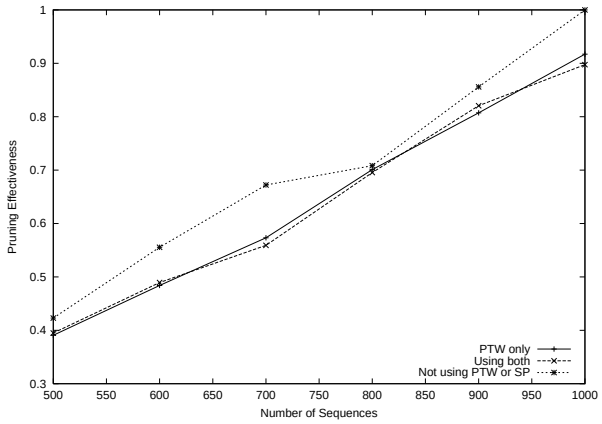

Figure 5.4: Effectiveness on the GPCRDB Amine/Peptide data using the `Amine` dataset

dow of the Olfactory sequences. We still need to evaluate many patterns if the δ_{min} stays low, even though we might find the maximal discriminating patterns quickly.

5.6 Conclusions

In this chapter we introduced and compared two sequential pattern mining algorithms. We used knowledge from the application area of protein sequence analysis. Given domain knowledge, we could improve mining for the maximal discriminating patterns. The effectiveness depends on the quality of the assumptions, e.g., how probable a discriminating pattern is within a certain time window. Our method also depends on the discriminative power of the patterns. Pruning will be less effective if this is low, even though we might find the maximal discriminating patterns quickly. It is shown that using probable time windows in protein sequences can speed up the search. Protein sequences are long but contain only a few types of items; constraints are required to make the discovery of patterns in these sequences tractable.

In future research we will further investigate methods for automatically discovering the probable time window. Furthermore we plan to use maximal discriminating patterns in other application areas like workflow analysis.

6 Visualization of Graph Patterns

Mining subgraphs is an area of data mining research where, a given set of graphs, one searches for (connected) subgraphs contained in these graphs satisfying a number of constraints. In this chapter we focus on the analysis of molecules. In the analysis of fragments one is interested in the molecules in which the patterns occur and for this reason we introduce a visualization technique. The user does not have to browse through all patterns; instead the user can directly see which subgraphs are of interest.

6.1 Introduction

Mining frequent patterns is an important area of data mining where one discovers substructures that occur often in (semi-)structured data. In this chapter we look at the area of frequent subgraph mining. The *subgraphs* are connected vertex- and edge-labeled graphs contained in another graph, the records or molecules. A subgraph is considered to be frequent if it occurs in at least *minsupp* transactions, where *minsupp* is a user-defined threshold above which patterns are considered to be frequent. A *frequent subgraph mining* algorithm will discover all these frequent subgraphs.

Figure 6.1 shows a "molecule" graph and two of its subgraphs. Our work is motivated by bio-chemists wishing to view co-occurrences of subgraphs in a dataset of molecules (graphs):

- For a bio-chemist it is interesting to know which fragments occur often together, for example in so-called active molecules. This is because frequent co-occurrence might imply that the fragments are needed simultaneously for biological activity.

- Pharmaceutical companies provide libraries of molecules. A visualization of co-occurrences in molecule libraries gives a bio-chemist insight how the libraries are constructed.

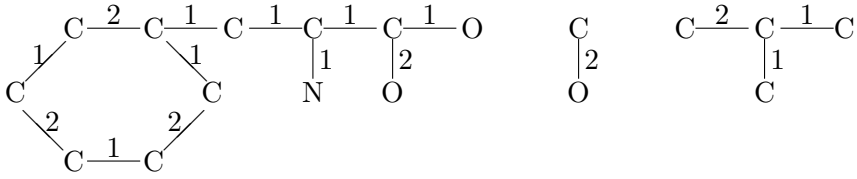

Figure 6.1: An example of a graph (the amino acid Phenylalanine) in the molecule data set and two of its many (connected) subgraphs, also called patterns or fragments.

A distance between patterns, the amount of co-occurrence, can be measured by calculating in how many graphs only one of the two patterns occurs: if this never happens then these patterns are very close to each other and if this always happens then their distance is very large.

We will define a method of building a co-occurrence model and show its usefulness. To this end, this chapter makes the following contributions:
— The visualization of co-occurring graph patterns.
— We improve the clarity of the visualization by grouping.
— We will define a measure of calculating distances between patterns and show how it can be calculated (Section 6.2 and Section 6.3).
— An empirical discussion of model construction for visualizing co-occurrence (Section 6.5).

The mining techniques for molecules in this chapter make use of a graph miner called gSpan, introduced in [75] by Yan and Han.

For the visualization a method of pushing and pulling points in accordance with a distance measure is used. The main reason to choose this particular method was because it enables us to put a limit on the number iterations and still have a result. Similar techniques were used in [13] to cluster criminal careers and in [39] for clustering association rules.

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 [77] Zaki et al. discuss different ways for searching through the lattice.

Clustering is important because of the visualization that it can provide. In general our work is related to SOMs as developed by Kohonen (see [38]), in the sense that SOMs are also used to visualize data through a distance

measure and uses a pulling strategy. A Self-Organizing Map (SOM) is a type of artificial neural network that is trained to produce a low dimensional representation of the training samples. A SOM is constructed by moving the best matching point and its neighbours (within a lattice of neurons) towards the input node. SOMs have been used in a biological context many times, for example in [33, 52]. In some cases molecules are clustered via numeric data describing each molecule, in [74] such clustering data is investigated. Also our work is related to work done on the identification *structure activity relationships* (SARs) where one relates biological activity of molecules by analyzing their chemical structure [18, 35] in the sense that in our work the structure of a graph can be used to build such a model. In [17, 60, 61] a statistical analysis was done on the presence of fragment substructures in active and inactive molecules. However our work is not concerned with the discovery of SARs, but with co-occurrence of subgraphs occurring in a collection of graphs. More related is the work done by Lameijer et al. in [41]. This work is concerned with co-occurring fragments discovered with a graph splitting. Graph splitting breaks molecules at topologically interesting points. Also they use a frequency threshold to filter out some fragments after they were generated, however they do not use frequent pattern mining techniques. Furthermore they do not build a co-occurrence model or a similar visualization of co-occurrence. Figure 6.2 shows two co-occurring subgraphs (fragments) discovered by Lameijer et al. in their dataset of molecules.

In [23] we also use the current setup to cluster data; that paper discusses an application that enables the user to further explore the results from a frequent subgraph mining algorithm, by browsing the lattice of frequent graphs.

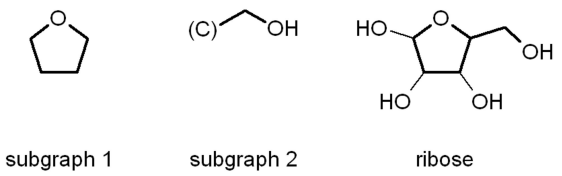

Figure 6.2: An example of co-occurring subgraphs from [41] with an example molecule.

The overview of the rest of the chapter is as follows. In Section 2 our distance measure is introduced, in Section 3 we discuss our method of group-

ing, in Section 4 we introduce the visualization and finally in Section 5 we discuss our experimental results.

6.2 Distance Measure

We are interested to know if patterns occur in the same graphs in the dataset of graphs. Patterns in this work are *connected subgraphs*.

The distance measure will compute how often subgraphs occur in the same graphs of the dataset. In the case of our working example it will show if different patterns (subgraphs) exist in the same molecules in the database. This distance measure is known as the Jaccard metric and was primarily chosen for its common use in Bio-informatics (see [72]). It is also easy to compute, given the appropriate supports; it doesn't make use of complicated graph comparisons, that would slow down the process. Formally we will define the distance measure in the following way (for graphs g_1 and g_2):

$$dist(g_1, g_2) = \frac{support(g_1) + support(g_2) - 2 \cdot support(g_1 \wedge g_2)}{support(g_1 \vee g_2)} \tag{6.1}$$

Here $support(g)$ is the number of times a (sub)graph g occurs in the set of graphs; $support(g_1 \wedge g_2)$ gives the number of graphs (or transactions) with both subgraphs g_1 and g_2 and $support(g_1 \vee g_2)$ 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 $support(g_1 \vee g_2)$ to make the distance independent from the total occurrence, thereby normalizing it. We can reformulate *dist* in the following manner:

$$dist(g_1, g_2) = \frac{support(g_1) + support(g_2) - 2 \cdot support(g_1 \wedge g_2)}{support(g_1) + support(g_2) - support(g_1 \wedge g_2)} \tag{6.2}$$

In this way we do not need to separately compute $support(g_1 \vee g_2)$ by counting the number of times subgraphs occur in the graphs in the dataset.

The measure is appropriate for our algorithm because it exactly calculates the number of transactions in which both patterns *do not* exist, hence a small distance means much co-occurrence. This measure also normalizes the exact co-occurrence, otherwise very frequent patterns can be considered mutually more distant compared to other points with the same proportional co-occurrence.

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

6.3 Optimization: Only Frequent Subgraphs and Grouping

In practice it is possible for the user to select a set of patterns for visualization. In this context we consider an optimization to be an automated selection of patterns such that the algorithm faster provides a model within reasonable time. The **first** optimization is to restrict the patterns to frequent patterns. Patterns (subgraphs) are considered to be frequent if they occur in at least *minsupp* graphs in the dataset. If we do not use frequent patterns we simply have too many patterns and, the frequent patterns give a comprehensive overview of the patterns. Efficient algorithms exist for finding frequent subgraphs, e.g., [75].

The **second** optimization is grouping: we group subgraphs and we will treat them as one point in our co-occurrence model. This will reduce the number of points. Moreover, the visualization will now show more directly the structural unrelated patterns, since related patterns are grouped. This will show to a biochemist the structural unrelated patterns that suggest to be together needed for biological activity.

The formula for the distance between supergraph g_2 and subgraph g_1 originates from Equation 6.2, where $support(g_1 \wedge g_2) = support(g_2)$:

$$\begin{aligned} dist(g_1, g_2) &= \frac{support(g_1) + support(g_2) - 2 \cdot support(g_2)}{support(g_1) + support(g_2) - support(g_2)} \\ &= \frac{support(g_1) - support(g_2)}{support(g_1)} \end{aligned}$$

The frequent pattern mining algorithm gives rise to a so-called lattice, in which the frequent subgraphs are ordered with respect to supergraphs. 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 needed by the graph mining algorithm to discover the frequent subgraphs and so the only extra calculating is done when *dist* does a search in this information.

Of course, many graphs have no parent-child relation and for this reason we define *lattice_dist* in the following way:

$$lattice\_dist(g_1, g_2) = \begin{cases} dist(g_1, g_2) & \text{if } g_2 \text{ is a supergraph of } g_1 \\ & \text{or } g_1 \text{ is a supergraph of } g_2 \\ 1 & \text{otherwise} \end{cases} \quad (6.3)$$

Note that $lattice\_dist(g_1, g_2) < 1$ if g_1 is a subgraph of g_2 and has non-zero support, or the other way around.

We will now organize "close" patterns into groups. The algorithm forms groups hierarchically, but this can be done fast because only related subgraphs are compared and also as a consequence all distances can be computed with the lattice. Now we need a distance between groups of patterns $C_1 = \{g_1, g_2, \ldots, g_n\}$ and $C_2 = \{h_1, h_2, \ldots, h_m\}$:

$$grdist(C_1, C_2) = \begin{cases} max(PG) & \text{if } PG \neq \emptyset \\ -1 & \text{otherwise} \end{cases} \quad (6.4)$$

$$PG = \{lattice\_dist(g, h) \mid g \in C_1, h \in C_2, lattice\_dist(g, h) \neq 1\}$$

Two clusters should not be merged if their graphs do not have a supergraph-subgraph relation, so we do not consider graphs where $lattice\_dist(g, h) = 1$. The value of *grdist* is -1 if no maximal distance exists, and clusters will not be merged in the algorithm.

The parameter *maxdist* is a user-defined threshold giving the largest distance allowed for two clusters to be joined. Note that grouping is efficient due to the fact that we can use the lattice information stemming from the frequent graph mining algorithm.

The outline of the algorithm 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 $\mathcal{P}$

GROUPING

6.4 Visualization

We will visualize co-occurrence by positioning all groups in a 2-dimensional area. We take the Euclidean distance $eucl\_dist(C_1, C_2)$ between the 2D coordinates of the points corresponding with the two groups (of frequent subgraphs) C_1 and C_2.

The graphs in a group occur in almost all the same transactions, hence the distance between groups is assumed to be the distance between any of the points of the two groups. We choose to define the distance between groups as the distance between a smallest graph of each of the two groups (*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 $group\_dist(C_1, C_2) = dist(g_1, g_2)$.

The coordinates (x_{C_1}, y_{C_1}) and (x_{C_2}, y_{C_2}) of the points corresponding with C_1 and C_2 are adapted by applying the following formulas:

1. $x_{C_1} \leftarrow x_{C_1} - \alpha \cdot (eucl\_dist(C_1, C_2) - group\_dist(C_1, C_2)) \cdot (x_{C_1} - x_{C_2})$
2. $y_{C_1} \leftarrow y_{C_1} - \alpha \cdot (eucl\_dist(C_1, C_2) - group\_dist(C_1, C_2)) \cdot (y_{C_1} - y_{C_2})$
3. $x_{C_2} \leftarrow x_{C_2} + \alpha \cdot (eucl\_dist(C_1, C_2) - group\_dist(C_1, C_2)) \cdot (x_{C_1} - x_{C_2})$
4. $y_{C_2} \leftarrow y_{C_2} + \alpha \cdot (eucl\_dist(C_1, C_2) - group\_dist(C_1, C_2)) \cdot (y_{C_1} - y_{C_2})$

Here α ($0 \leq \alpha \leq 1$) is the user-defined learning rate.

Starting with random coordinates for the groups, we will build a 2D model of relative positions between groups by randomly *choosing two groups r times and applying the formulas*. This is a kind of push and pull algorithm which yields a visualization in which the distances in 2D correspond to the distances in the pattern space. Note that we always have a visualization: the longer we run the algorithm, the better the Euclidean distances correspond to the distances between groups in the pattern space.

6.5 Performance

The experiments are organized such that we first show that the distances are approximated correctly. Secondly we will discuss runtime in the case of different *minsupp* settings for different datasets. Finally through experiments we analyze the speed-up due to making groups first.

One dataset we use, the `4069.no_aro` *dataset*, containing 4,069 molecules; from this we extracted a lattice containing the 1,229 most frequent subgraphs. This dataset was provided by Leiden/Amsterdam Center for Drug

Research (LACDR). Other datasets we use are datasets of the National Cancer Institute (NCI), and can be found in [55]. One of these datasets contains 32,557 2D structures (molecules, average size is 26.3 nodes) with cancer test data as of August 1999; we will call this dataset the `NCI.normal.99` *dataset*. The other NCI dataset contains 250,251 molecules and we will call this dataset the `NCI.large.99` *dataset*.

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.

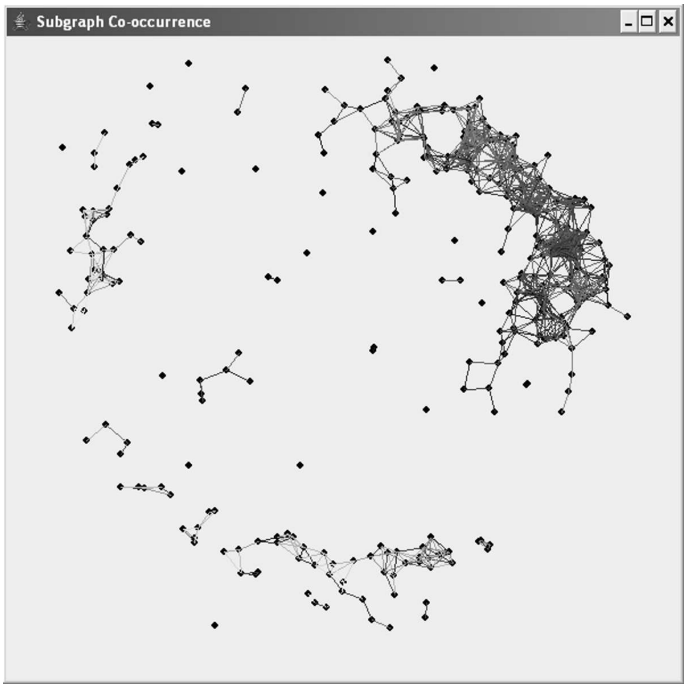

Figure 6.3: Clusters for graphs in the `4069.no_aro` dataset built in 24.5 seconds, connecting points at distance 0.05 or lower ($\alpha = 0.1$, $maxdist = 0.1$, $r = 1,000,000$).

Figure 6.3 shows how points, that represent subgraphs occurring in the same graphs (transactions) of the dataset, are close together. We draw lines between points if their Euclidean distance is ≤ 0.05. The darker these lines the lower their actual distance and in this way one can see gray clusters of close groups of subgraphs. Some groups are placed close but their actual

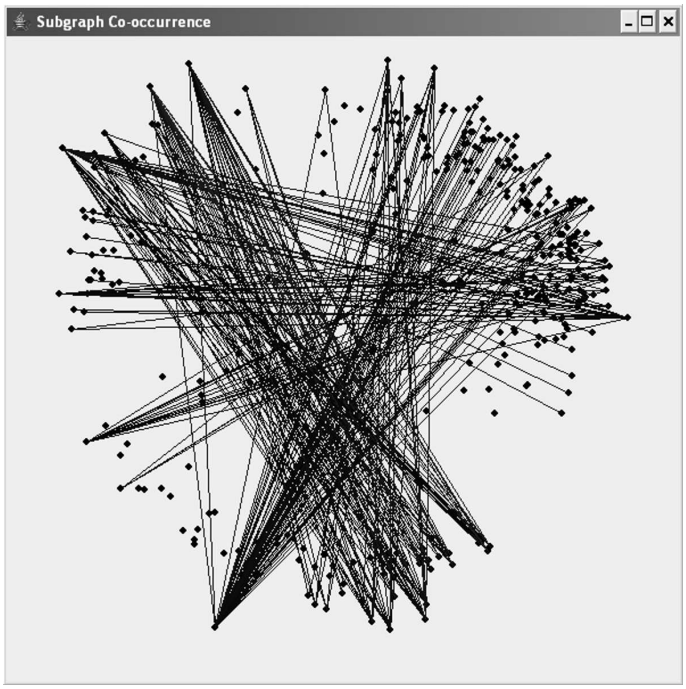

Figure 6.4: Clusters of graphs in the `4069.no_aro` dataset built in 24.5 seconds, connecting points at distance 0.95 or higher ($\alpha = 0.1$, $maxdist = 0.1$, $r = 1,000,000$).

distance is not close (they are light grey). This is probably caused by the fact that these groups do not occur together with some specific other groups, so being far away from these other ones.

In Figure 6.4 we draw lines between points with a Euclidean distance ≥ 0.95. The darker these lines the higher their actual distance. The figure shows their actual distance to be big also (the lines are black). Also Figure 6.4 shows bundles of lines going to one place. This probably is again caused by groups not occurring together with the same other groups.

The error for the cluster model for the `4069.no_aro` dataset decreases quickly, see Figure 8.5. After pushing or pulling 10,000 group pairs it becomes already hard to reduce the error further making a reduction of model building time possible.

In one experiment we assumed that the distances could not be stored in memory. In this experiment we first clustered 1,229 patterns without group-

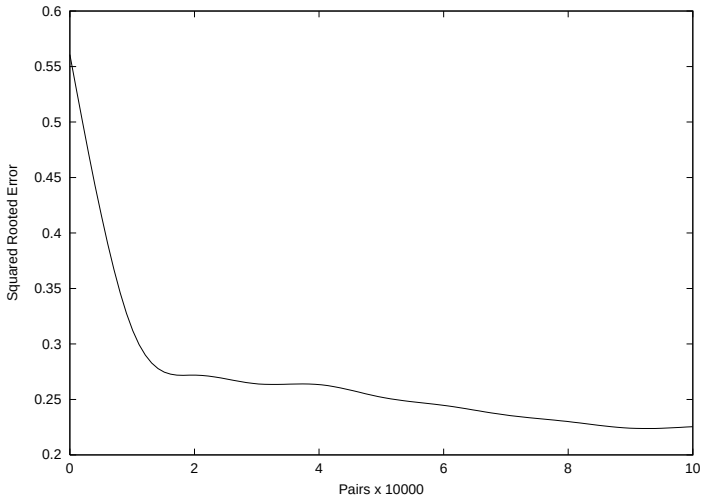

Figure 6.5: Root squared error for distance given by the cluster model for the `4069.no_aro` dataset ($\alpha = 0.1$).

ing, taking 81 seconds. However, grouping reduced the number of requests to the compressed occurrence data and because of this with grouping model construction was done in 48 seconds ($\alpha = 0.1$, $r = 1,000,000$, *maxdist* $= 0.1$, dataset is `4069.no_aro`).

Table 6.1 shows the runtime where *minsupp* varies. Obviously for a lower *minsupp* it takes longer to build the model, but for 12,734 subgraphs a model is still built within an acceptable time frame.

Table 6.2 and 6.3 show the runtime where *minsupp* varies, it is set to a percentage of the total dataset size. Results show that the algorithm is able

minsupp	average runtime (sec) $\pm$ *stdev*	number of subgraphs
200	2204.60 $\pm$ 6.36	12,734
300	335.10 $\pm$ 2.00	4,571
400	45.75 $\pm$ 0.17	2,149
500	17.95 $\pm$ 0.23	1,229

Table 6.1: Runtime performance in seconds for different *minsupp* settings for the `4069.no_aro` dataset ($\alpha = 0.1$, $r = 10,000$, *maxdist* $= 0.2$).

minsupp	average runtime (sec) ± *stdev*	number of subgraphs
5%	1495.57 ± 5.41	5,663
10%	160.82 ± 0.42	1,447
20%	17.09 ± 0.13	361
30%	4.64 ± 0.01	158

Table 6.2: Runtime performance in seconds for different *minsupp* settings for the `NCI.normal.99` dataset ($\alpha = 0.1$, $r = 10,000$, $maxdist = 0.2$).

minsupp	average runtime (sec) ± *stdev*	number of subgraphs
5%	2080.12 ± 9.40	2,391
7%	840.49 ± 11.67	1,313
10%	301.58 ± 3.57	648
15%	91.35 ± 0.59	332

Table 6.3: Runtime performance in seconds for different *minsupp* settings for the `NCI.large.99` dataset ($\alpha = 0.1$, $r = 10,000$, $maxdist = 0.2$).

to handle the `NCI.normal.99` dataset of 32,557 molecules and `NCI.large.99` dataset of 250,251 molecules, even with a low *minsupp*, within a reasonable time frame.

Our final experiments were done to show how the runtime is influenced by the *maxdist* threshold and how much the preprocessing step influences runtime. Here we assume that the distances can be stored in memory. In Figure 6.6 the influence on runtime is shown and to each line a Bézier curve is fitted (the degree is the number of datapoints). The figure displays preprocessing to proceed more or less stable.

In Figure 6.7 results show the runtime for the `NCI.normal.99` dataset with approximately an equal number of patterns. The performance for grouping is nearly the same as for the `4069.no_aro` dataset. This performance depends more on the number of patterns that are grouped. The results indicate that the total runtime depends on the size of the dataset, but that runtime can be improved strongly by better selecting the *maxdist* threshold.

The first analysis of results shows promising patterns, see Figure 6.8. The results show two frequent subgraphs (a) and (b) occurring together. This suggests that patterns (c) and (d) might also occur together, requiring

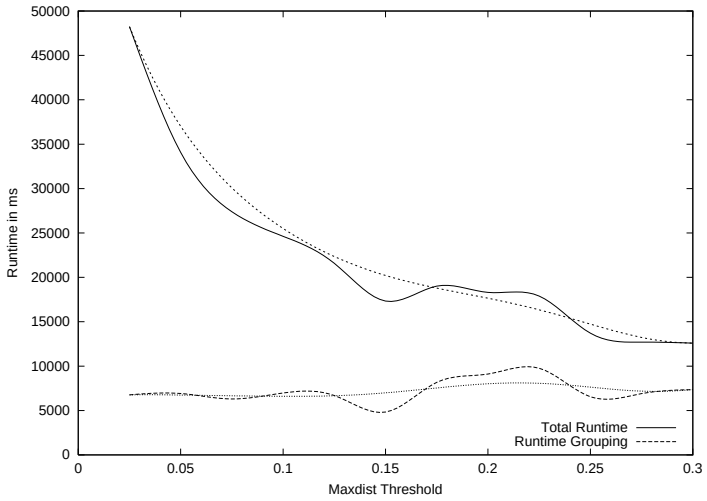

Figure 6.6: Average runtime for the `4069.no_aro` dataset with varying *maxdist* ($\alpha = 0.1$, nr. of patterns = 1,229, $r = 1,000,000$).

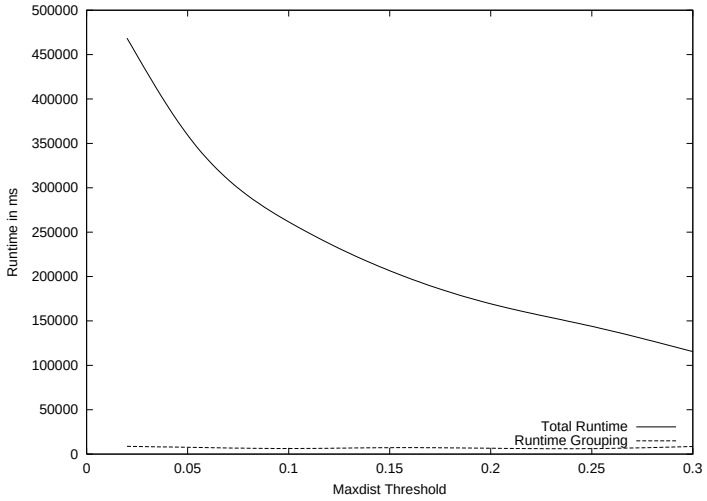

Figure 6.7: Average runtime for the `NCI.normal.99` dataset with varying *maxdist* ($\alpha = 0.1$, nr. of patterns = 1,447, $r = 1,000,000$).

further research.

Also biochemists in Leiden are actively researching the development of

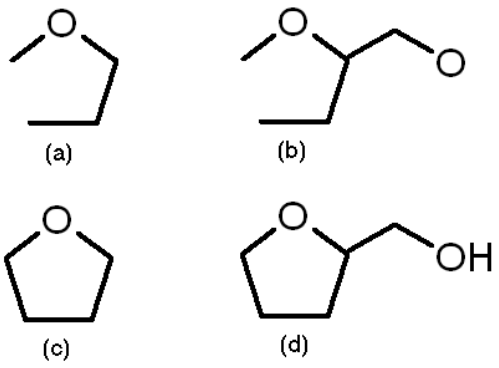

Figure 6.8: Two co-occurring frequent patterns (a) and (b), and two potentially interesting ones (c) and (d).

simple biologically active molecules consisting of fragments (subgraphs) not co-occurring frequently [42]. Modeling co-occurrence will hopefully help improve their analysis.

6.6 Conclusions and Future Work

Presenting data mining results to the user in an efficient way is important. In this chapter we proposed a visualization of a co-occurrence model for subgraphs that enables quicker exploration of occurrence data.

The forming of groups improves the visualization. The visualization enables the user to quickly select the interesting subgraphs for which the user wants to investigate the graphs in which the subgraphs occur. Additionally the model can be built faster because of the grouping of the subgraphs.

In the future we want to take a closer look at grouping where the types of vertices and edges and their corresponding weight also decide their group. Furthermore, we want to investigate how we can compress occurrence more efficiently and access it faster.

7 Improved Exploration of Graph Mining Results

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 chapter are used in (bio)chemical analysis.

In this chapter 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. Our application can display only a selection of groups that occur in almost the same set of molecules, or, if desired, in different molecules. This allows one to see which patterns cover similar c.q. different parts of the dataset.

7.1 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 7.1 shows a graph and two of its subgraphs.

In this chapter 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.

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 [24]. Another important functionality is the browsing of lattice information from parent to child and from one group of fragments to another as presented here.

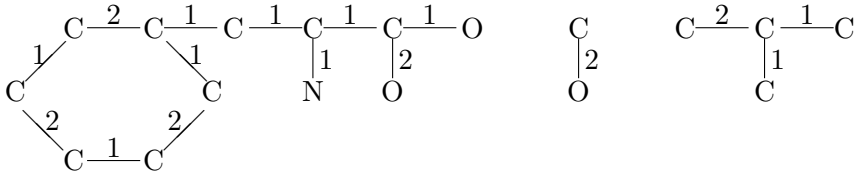

Figure 7.1: An example of a possible graph (the amino acid Phenylalanine) in the molecule dataset and two of its many (connected) subgraphs, also called patterns or fragments.

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 7.1 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 chapter 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 chapter 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 7.2).
— We will use a **distance measure based on the co-occurrence of fragments to browse** from one fragment group to another (Section 8.3.2 and Section 7.4).
— We will give an **algorithm for grouping** very similar subgraphs using hierarchical cluster methods and lattice information (Section 7.4).
— Finally through experiments we will take a **closer look at runtime performance** of the grouping algorithm and discuss it (Section 7.5).

The algorithm for grouping was also used in previous chapters, both chapters discuss a component of the same framework. However in this chapter 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 [77] 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., [62] 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 [33, 52]. There is also a relation with work done on hierarchical clustering in the biological context, e.g., as presented in [67]. In some cases molecules are clustered via numeric data describing each molecule; in [74] clustering such data is investigated.

Our package of mining techniques for molecules makes use of a graph miner called gSpan, introduced in [75] 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).

7.2 Exploring the Lattice

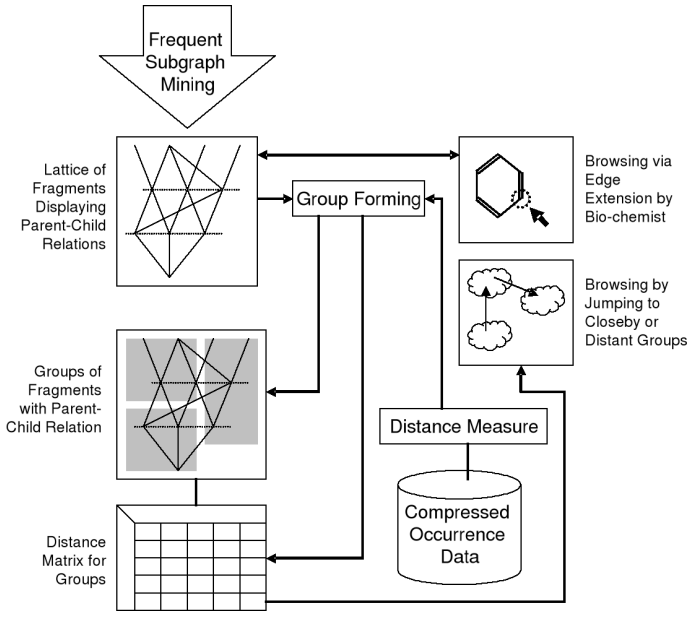

Figure 7.2: The process of exploring the fragment lattice.

We propose a *fragment exploration tool* to explore fragments in a dataset of molecules, the whole process is visualized in Figure 7.2. 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 8.3.2; 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.

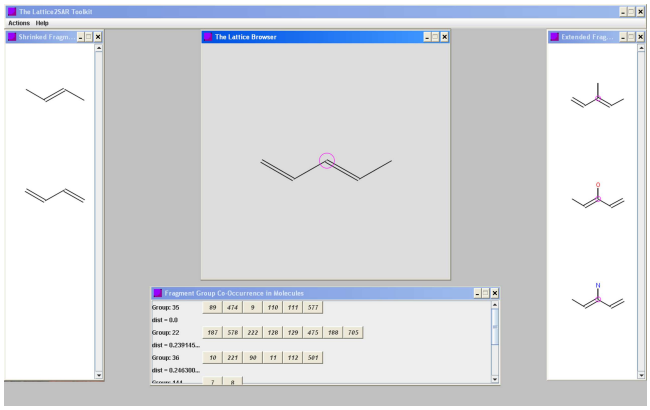

Figure 7.3: Fragment exploration with the possible ways of shrinking and extending.

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 7.3 shows the current fragment in the 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 7.4 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 7.4). For every group it shows the distance, indicated with "dist", to the group of the current fragment.

7.3 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

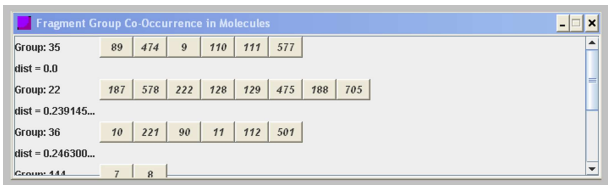

Figure 7.4: Co-occurrence view for groups, showing all groups close by ($group\_dist \leq 0.3$).

g_1 and g_2):

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

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* and *symmetry*. Note that $0 \leq dist(g_1, g_2) \leq 1$ and also $dist(g_1, g_2) = 1 \Leftrightarrow sup(g_1 \wedge g_2) = 0$, 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.

7.4 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.

7.5 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.

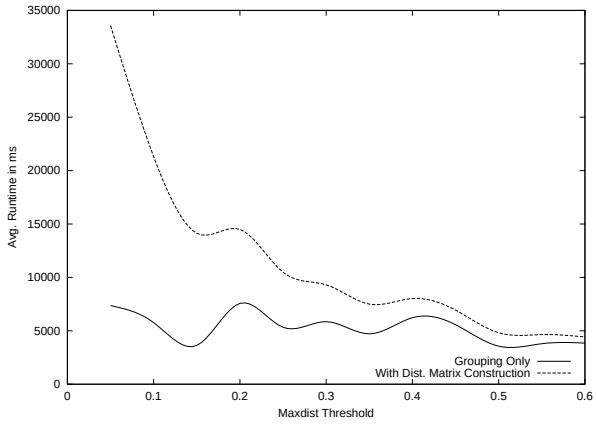

Figure 7.5: Runtime in ms for different *maxdist* settings and the influence of distance matrix construction.

Figure 7.5 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

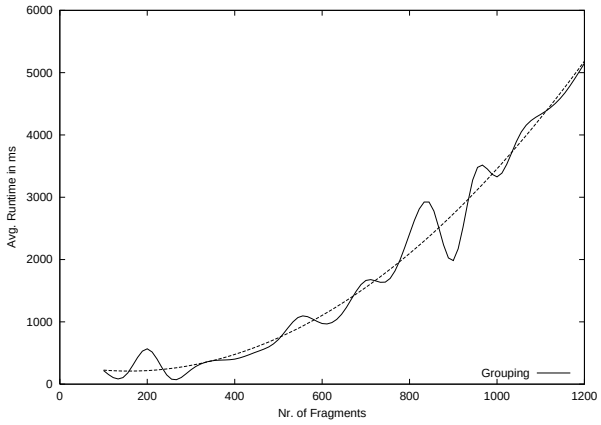

Figure 7.6: Runtime in ms for different fragment set sizes ($maxdist = 0.3$), with quadratic regression

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 7.6 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.

7.6 Conclusions

The application discussed in this chapter facilitates the exploration of fragments extracted from a data set 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 most of the 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.

8 Displaying Graph Pattern Co-Occurrence in Streams

One way of getting a better view of a dataset is by considering the frequent patterns. In this chapter we consider frequent patterns are (sub)sets that occur a minimal number of times in a stream of item sets. The discovery of frequent patterns in streams is difficult, because streams are potentially infinite and then it is harder to say if a pattern is frequent or not. Furthermore, the number of patterns can be huge and a good overview of the structure of the stream is lost quickly.

Our approach to this problem will use competitive neural network methods to online model pattern co-occurrence in a stream of itemsets. A model of the co-occurrence of patterns will give the user an improved view on the structure of the stream. Some patterns might occur so often together that they form a "combined" pattern. In this way the patterns in the clustering will approximate the largest frequent patterns: maximal frequent patterns. The number of (approximated) maximal frequent patterns is much smaller and combined with methods of visualization using competitive neural networks these patterns provide a good view on the structure of the stream.

8.1 Introduction

Effectively mining streams of data for *frequent patterns*, i.e., patterns occurring at least a minimal number of times, has always been a hard problem to tackle. The difficulty lies in the potential endlessness of the stream, frequent patterns can suddenly become infrequent and the standard ways of pruning the search space are harder to use. In this work *patterns* are sets of

items occurring in a record (also called transaction or *itemset*) at a certain moment in time.

This work is motivated by a wish to view pages accessed together by users helping website analysts improve the website. To this end we will propose a method of modeling co-occurring patterns in a stream of itemsets.

Knowing how much patterns co-occur can provide interesting structural information about the stream in an online way. Note that the model is an approximation and due to this the frequent subsets are also approximately maximal.

We will define our method of displaying co-occurring patterns in a stream of itemsets and show its usefulness. This chapter makes the following contributions:
— We use a **dynamic support estimation** to determine the support of those itemsets we need, and do this in an online way (Section 8.3.1).
— It will be **explained how the distance between patterns is approximated** by placing patterns closer (pulling) or further away (pushing) depending on their co-occurrence. If this distance is large, patterns occur almost never together, and otherwise they do have many common occurrences (Section 8.3.2).
— We will define when patterns **can be merged and when they should be split** to form smaller patterns and how this should be done (Section 8.3.3).
— Finally through experiments **the effectiveness of our method is shown** and efficiency is discussed (Section 8.4).

The rest of this chapter is structured as follows. We first mention related work, then we discuss the algorithm in full detail. Finally we describe experiments and discuss these.

8.2 Related Work

This research is related to work done on visualization of patterns in streams and visualization of website usage using patterns as done in [8]. Also our work is related to (maximal) frequent pattern mining in streams and large datasets. *Maximal frequent itemsets* are sets of items occurring often in the stream while there is no frequently occurring bigger set of items containing these same items.

There are many algorithms for mining maximal frequent patterns, in "normal" datasets, in different ways. We mention GenMax discussed in [21] and MAFIA presented in [7]. Large datasets are different from streams

in that there is an end to the dataset. One approach to mining large datasets was proposed in [15], where an extremely large dataset is mined for maximal frequent patterns by proceeding in parallel. Furthermore clustering on large datasets was done in [54]. Much work has been performed on mining frequent patterns in (online) data streams, e.g., in [9]. In [10] frequent patterns are mined by using sliding window methods. However it must be said that our work is more concerned with co-occurrence and frequent patterns are approximately maximal. Our work has little overlap with work done on maximal pattern-based clustering as discussed in [58] and [68] where objects basically are clustered by linking attribute groups with object groups when attributes have a minimal similarity. Related research has been done on clustering on streams in [1], where a study on clustering evolving data streams, (fast) changing data streams, is done. Aggarwal et al. continue their work in [2] by clustering text and categorical data in streams. Clustering categorical data was also done in [19] where also co-occurrence is used, but only for attribute values; the authors propose a visualization where the x-axis is the column position and the y-axis the distance based on co-occurrence of values. Also in [56] clustering on streams is mentioned, there the authors propose a new algorithm and compare it with K-Means (see [51]).

In this work a method of pushing and pulling points in accordance with a distance measure is used. This technique was used before in [13] to cluster criminal careers and in [39] to cluster association rules. This method of clustering was chosen because it enables us to limit the number of iterations in order to improve online performance while still having results. Furthermore we only know the distance between two patterns, where a low distance means frequent co-occurrence.

8.3 Model Realization

Our goal is to produce an algorithm that is capable of accepting a stream of records, each record being an unordered finite set of items, meanwhile building a model of patterns and their co-occurrence. We **first** optimize this model by restricting the patterns to frequent patterns, simply because we will have too many otherwise. Our **second** optimization is to restrict patterns to maximal frequent patterns. If we do not use maximal frequent patterns then the model might have too many frequent patterns for a reasonable online performance because all potentially frequent patterns need to be kept.

The algorithm we propose, called DistanceMergeSplit, starts with

randomly positioning n points in a 2-dimensional space, e.g., in the unit square. Here n is the number of items maximally possible in an itemset. Each of these n points represents one size 1 itemset, where the size of an itemset is of course defined as the number of items it contains. These n points remain present during the whole process, though their coordinates may change. While the records from the data stream pass by, new points are created (by merging or splitting) and others disappear (by merging, or by other reasons). Together these points constitute the evolving model $\mathcal{P}$, where points correspond with frequent itemsets.

We will first explain how we use the stream of records to update the supports of the elements of $\mathcal{P}$, next we describe how the coordinates of the elements change in accordance with the corresponding supports, and finally mention our method of growing and shrinking the number of sets present in $\mathcal{P}$: the merge and split part of the algorithm.

8.3.1 Support

The algorithm will receive a possibly infinite stream of itemsets, the records: $r_1, r_2, r_3, \ldots$ Each time an itemset corresponding to a point in the space is a subset of a record, we observe an occurrence of this itemset. We count the occurrences in the t records we have seen so far (and that can also be considered as the last t records), and define support as follows:

$$support\ (p,t) = \sum_{i=1}^{t} occurrence\ (p, r_i) \tag{8.1}$$

$$occurrence\ (p,r) = \begin{cases} 1 & \text{if } p \subseteq r \\ 0 & \text{otherwise} \end{cases}$$

where p is the pattern, the itemset, for which support is computed, and r is a record. If a new record arrives the support needs to be adapted accordingly. Rather than using the full support for all records, we will make use of a *sliding window* of size $\ell \geq 1$, and we will not keep all data about the occurrences of the patterns in the transactions of this window. Though this is not essential for our algorithm, it has a beneficial influence on the runtime, which is especially interesting for an online algorithm. If we have seen less than ℓ transactions ($t < \ell$) then we *do* use the previous formula to calculate support, in such case a pattern is called "young". This method will also be used when we later create new patterns online, and is referred to as "direct computation". In the other case ($t \geq \ell$) a pattern is called "old" and we give an estimate $support_t(p)$ for the support during the last ℓ records in the

following way. When the itemset p is *not* a subset of the current record r_t we adapt the support as follows:

$$\begin{aligned} & support_t(p) \\ &= support_{t-1}(p)/\ell \cdot (support_{t-1}(p) - 1) \\ &\quad + (1 - support_{t-1}(p)/\ell) \cdot support_{t-1}(p) \\ &= (1 - 1/\ell) \cdot support_{t-1}(p) \ \leq \ support_{t-1}(p). \end{aligned} \tag{8.2}$$

Indeed, when the first transaction of the window of size ℓ contains the pattern then support should decrease with one. However, if the first record also does not contain p, then support remains the same. It is important to notice that the probability of a transaction containing p in a window of size ℓ is estimated with $support_t(p)/\ell$. If the new record *does* contain the itemset p then support is adapted as follows:

$$\begin{aligned} & support_t(p) \\ &= support_{t-1}(p)/\ell \cdot support_{t-1}(p) \\ &\quad + (1 - support_{t-1}(p)/\ell) \cdot (support_{t-1}(p) + 1) \\ &= (1 - 1/\ell) \cdot support_{t-1}(p) + 1 \ \geq \ support_{t-1}(p) \end{aligned} \tag{8.3}$$

Now when the first transaction of the window of size ℓ contains the pattern then support remains unchanged as the window shifts. However, if it does not contain the pattern p, then support will increase with 1. Both formulas *assume that occurrences are uniformly spread* over the window of size ℓ, but by using these formulas to adapt support we do not have to keep all occurrences for all patterns in the 2-dimensional space. Notice that $0 \leq support_t(p) \leq \ell$ always holds.

We have now described how the stream of records influences the supports of the itemsets that are currently being tracked, i.e., those in $\mathcal{P}$. Note that the itemsets of size one are always present in the model $\mathcal{P}$ of co-occurring patterns, for reasons mentioned in Section 8.3.4. Larger itemsets may appear and disappear as the algorithm proceeds. Also observe that the supports are estimates, due to the application of equations 8.2 and 8.3.

8.3.2 Distance

We now describe how the coordinates of the points change as their supports vary when the new records from the stream arrive. In our model for *distance* (p_1, p_2) we take the Euclidean distance between the 2-dimensional coordinates of the points corresponding with the two patterns p_1 and p_2.

These points are pulled closer to each other if they occur in the current transaction and they are pushed apart if not. Furthermore nothing is done if both do not occur. In every time step a random selection of the pairs undergoes this process.

To pull two points together we set the *goal distance* to 0. To push them apart the goal distance is $\sqrt{2}$, which is the maximum Euclidean distance between any two points in the unit square. These distances are then used to update the coordinates (x_{p_1}, y_{p_1}) and (x_{p_2}, y_{p_2}) of the points corresponding with the itemsets p_1 and p_2:

1. $x_{p_1} \leftarrow x_{p_1} - \alpha \cdot (\mathit{distance}\ (p_1, p_2) - \gamma) \cdot (x_{p_1} - x_{p_2})$
2. $y_{p_1} \leftarrow y_{p_1} - \alpha \cdot (\mathit{distance}\ (p_1, p_2) - \gamma) \cdot (y_{p_1} - y_{p_2})$
3. $x_{p_2} \leftarrow x_{p_2} + \alpha \cdot (\mathit{distance}\ (p_1, p_2) - \gamma) \cdot (x_{p_1} - x_{p_2})$
4. $y_{p_2} \leftarrow y_{p_2} + \alpha \cdot (\mathit{distance}\ (p_1, p_2) - \gamma) \cdot (y_{p_1} - y_{p_2})$

where α $(0 \leq \alpha \leq 1)$ is the user-defined learning rate and γ $(0 \leq \gamma \leq \sqrt{2})$ is the goal distance.

We not only use the distances to place the patterns in the 2D space, but also to *decide when to merge*. Points may leave the unit square; however, when presenting the results of the experiments, such points are projected on the nearest "wall" of the unit square.

8.3.3 Merge and Split

Now we describe how we merge and split the itemsets of the model as time goes by. The model $\mathcal{P}$ contains points with corresponding itemsets. Two old patterns (itemsets) are assumed to occur many times together when their distance is small due to them being pulled together. In some cases one itemset can be made that represents two of them: the algorithm will try these combinations. For some combinations it is possible that they turn out to be not so good, their frequency is smaller than *minsupp*, where *minsupp* is a user-defined threshold. This can happen when their combined frequency is lower than *minsupp* or suddenly frequency drops below *minsupp*. In either case we need to split the size k itemset into k itemsets of size $k-1$, all being subsets of the original itemset. Later we will discuss splitting in more detail, we now first explain merging.

As transactions come in, some of the initial size one itemsets become *frequent*, meaning that the support is higher than *minsupp*. These sets can — under certain circumstances, see below — merge to itemsets of size 2,

and so on: we **merge** two itemsets p_1 and p_2 if (in the algorithm in Section 8.3.4 the following series of conditions is referred to as "appropriate"):

- The patterns p_1 and p_2 are old enough: they exist in $\mathcal{P}$ for at least ℓ (the window size) records. (Note that the supports of these sets are currently updated through equations 8.2 and 8.3 above.)

- The two itemsets p_1 and p_2 currently are frequent, i.e., it holds that both $support_t(p_1) \geq minsupp$ and $support_t(p_2) \geq minsupp$. (Note that this condition automatically holds for all (pairs of) itemsets in $\mathcal{P}$ that have size larger than 1.)

- The itemsets are close together in the model, so they are assumed to occur often together as a subset of transactions in the stream: $distance\ (p_1, p_2) \leq mergedist$, where *mergedist* is a user-defined upper bound for the distance for which merging p_1 and p_2 is allowed.

- The pattern p_2 has an item i_p which is not in the pattern p_1, such that $p_2 \setminus \{i_p\} \subseteq p_1$. This condition always holds if p_2 has size 1.)

First of all we merge the patterns p_1 and p_2 if they are of equal size, so we create the set $p_1 \cup p_2$ and add it to $\mathcal{Q}$, the collection of all newly formed patterns. Both original patterns are removed from the 2-dimensional space except if their size is 1.

The **second** time we merge patterns is if pattern p_1 contains more items than p_2 and $p_2 \setminus \{i_p\} \subseteq p_1$ for some $i_p \in p_2$ with $i_p \notin p_1$, then for each item $e \in p_1 \setminus p_2$ we add an itemset $p_2 \cup \{e\}$ to $\mathcal{Q}$. This enables patterns to be merged with patterns that already were merged before and disappeared from the model. The smaller pattern p_2 is removed except if it is of size 1.

Next we **split** patterns, when they contain more than one item, if they do not occur often enough and they have been in the model for at least a certain number of records (they are "old enough"). Split combinations are generated by removing each item from the original pattern once. The remaining items form one new itemset, so in this way a size k itemset will result in k combinations after splitting.

Assume we have the pattern p that is split into patterns $q_0, q_1, \ldots, q_{|p|-1}$ that are added to $\mathcal{Q}$:

$$split : p = \{i_0, i_1, \ldots, i_{|p|-1}\} \rightarrow q_0 = \{i_1, i_2, \ldots, i_{|p|-1}\},$$
$$q_1 = \{i_0, i_2, i_3, \ldots, i_{|p|-1}\}, \ldots, q_{|p|-1} = \{i_0, i_1, \ldots, i_{|p|-2}\}$$

Finally, the newly formed patterns in $\mathcal{Q}$ are united with those in $\mathcal{P}$. Of course, when patterns occur more than once, only one copy — the oldest one — is maintained. And those patterns from $\mathcal{P}$ that are contained in a larger one in $\mathcal{P}$ are removed, unless — as stated above — they have size one: we focus on the maximal patterns.

8.3.4 The Algorithm

The algorithm works with the set $\mathcal{P}$ of patterns that are currently present, represented by (coordinates of) points in 2-dimensional Euclidean space. The outline of the algorithm DistanceMergeSplit is as follows:

---

initialize $\mathcal{P}$ with the n itemsets of size 1
for $t \leftarrow 1$ to ∞ **do**
  $\mathcal{Q} \leftarrow \emptyset$
  for all patterns $p \in \mathcal{P}$ **do**
    compute $support_t(p)$ using the t^{th} record r_t,
    either through updating (old patterns)
    or by direct computation (young ones)
  for a random subset of pairs of patterns in $\mathcal{P}$ **do**
    update their distance according to their support
  for all "appropriate" pattern pairs in $\mathcal{P}$ **do**
    merge the pair, creating (new) pattern(s) in $\mathcal{Q}$
    mark the smallest of the pair,
    or both if their sizes are equal
  remove the marked patterns from $\mathcal{P}$
  for all patterns $p \in \mathcal{P}$ **do**
    if p is infrequent and old enough **then**
      split p into (new) patterns in $\mathcal{Q}$
      remove p from $\mathcal{P}$
  $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{Q}$, joining duplicates
  remove non-maximal frequent patterns from $\mathcal{P}$

DistanceMergeSplit

Note that itemsets of size one are *never* removed from $\mathcal{P}$, not even when they are infrequent. The size one itemsets are always present, and play a special role: besides the fact that some of them are frequent, they also serve as building blocks. In many cases they are not maximal. If they were

removed, it could be impossible to re-introduce single items after having become infrequent.

Patterns that are new in $\mathcal{P}$ are called "young". When computing supports for these patterns, we use equation 8.1, when updating the "old" ones we use equations 8.2 and 8.3. So, each pattern present in $\mathcal{P}$ also has an *age*: patterns that have an age smaller than the window size ℓ are "young", the others are "old".

On two occasions the algorithm introduces indeterminism: first, when the support computation is done using the approximating updates for "old" patterns (saving a lot of time and memory) and second, when pushing and pulling pairs of points representing a pattern, see Section 8.3.2.

8.4 Experiments and Discussion

The experiments are organized such that we first show the method at work in a few controlled synthetic cases. Then we will use the algorithm to build a model for real datasets, showing "real life" results. The first synthetic experiment will be a stream with 10 groups of 5 items. Groups do not occur together, but all of them occur often. This dataset is called the `10-groups` dataset. The second synthetic experiment will be a stream where certain groups of items suddenly do not occur; instead another group starts occurring. We call this dataset the `suddenchange` dataset.

The first real dataset comes from Internet Information Server (IIS) logs for `msnbc.com` and news-related portions of `msn.com` for the entire day of September, 28, 1999. The original dataset contained sequences of 17 possible categories viewed by a user within 24 hours and was used before in [8]. For our purpose we converted the dataset to itemsets. To make the problem more interesting for our problem, we removed users viewing only one or two categories. This dataset will be called the `MSNBC` dataset and contains 174,042 transactions.

The second real dataset is the Large Soybean Database used for soybean disease diagnosis in [47], we call the dataset the `soybean` dataset. This dataset contains 683 records with 35 attributes. First we removed all missing values and we converted each record to a string of $n = 84$ yes/no values for each attribute value. In this research we do not deal with missing values, and each item represents an attribute value, we use this dataset to analyze the performance of our algorithm with a real dataset with more than 50 items.

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.

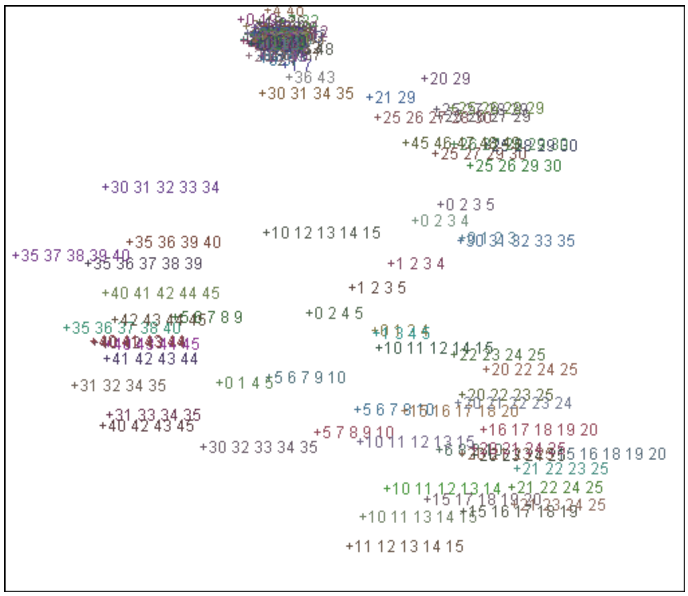

Figure 8.1: Model after seeing 1,200 transactions of the `10-groups` dataset ($n = 50$, $minsupp = 0.05$, $\ell = window\ size = 300$, $mergedist = 0.1$, $\alpha = 0.1$).

Figures 8.1 and 8.2 show how the cluster model changes as more transactions are coming in for the `10- groups` dataset. The first group of this dataset consists of items 0 to 5, the second has 5 to 10, etc. In Figure 8.2 we clearly see these patterns, where *minsupp* is given as a percentage of the dataset size. Furthermore notice that both the second and the first group contain the item 5, so there is a slight overlap. We see these itemsets closer together because they are both close to the pattern $\{5\}$. In order to get a clear picture we did not display the size 1 itemsets. Itemsets are plotted using +s, accompanied by the items they contain.

The second synthetic dataset, called the `suddenchange` dataset, simulates a stream that completely changes after seeing many transactions (i.e., 30,000). The results are displayed in Figure 8.3, where the labels above each bar reveal the size of the itemsets. First the records in the stream always contain items 1 to 5. Then after 30,000 transactions they only contain items 25 to 30. Figure 8.3 shows how the first pattern appears and how it slowly disappears in the middle. In the end the model contains only patterns with

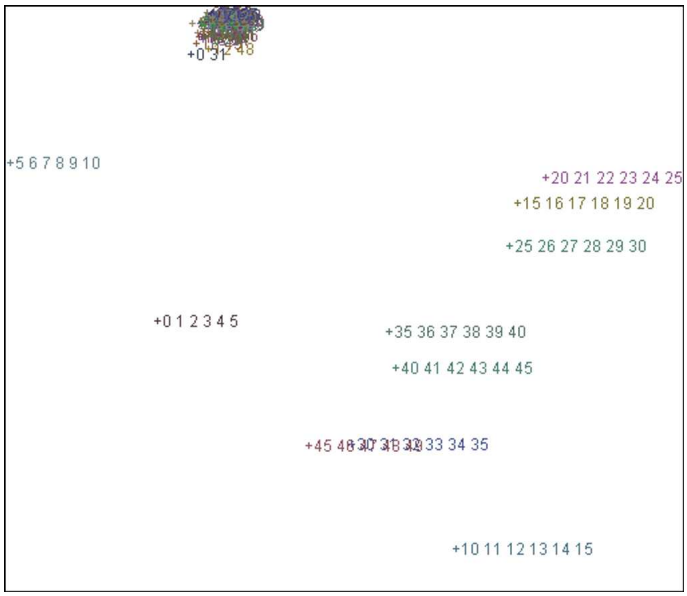

Figure 8.2: Model after seeing 4,500 transactions of the `10-groups` dataset ($n = 50$, $minsupp = 0.05$, $\ell = window\ size = 300$, $mergedist = 0.1$, $\alpha = 0.1$).

items 25 to 30.

Figure 8.4 was made using the formula $\frac{1}{|\mathcal{P}|} \cdot \sum_{i=1}^{|\mathcal{P}|} abs(|p_i| - |rmax(p_i)|)/|rmax(p_i)|$ for each model $\mathcal{P}$, we call this value the *average relative difference*. Here *rmax* gives the itemwise nearest maximal frequent pattern with $p_i \in \mathcal{P}$ as a subset. These maximal frequent patterns are beforehand decided with a frequent itemset miner. In short this formula calculates how itemsets in the model (itemwise) differ from the actual maximal frequent patterns. Figure 8.4 displays how the average relative difference stabilizes around 0.2. We also plot the number of maximal frequent patterns divided by the actual number, where 1.0 means they are equal in size. This value approaches 1.0 especially when merging and splitting is temporarily stopped after 50,000 transactions.

Approximating supports well is important in order to know which itemsets should be split. In Figure 8.5 we show for all patterns in a computed model the error between their approximated support and their real support in the time window as the transactions from the `MSNBC` dataset arrive. The root mean squared error of the supports for this model eventually approaches 0.06. The error becomes more stable after temporarily stopping

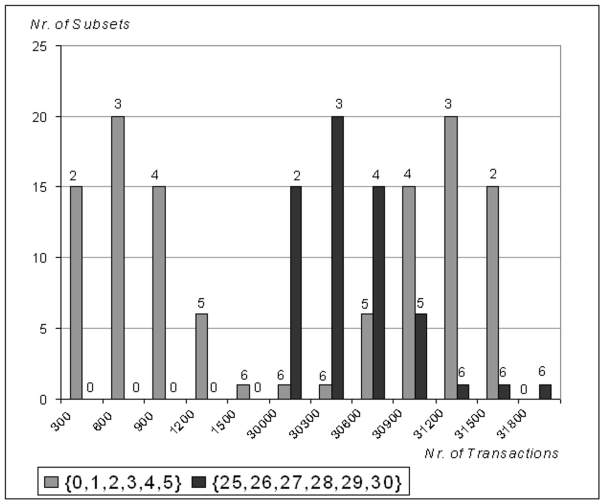

Figure 8.3: The `suddenchange` dataset, the stream changes in the middle ($n = 50$, $minsupp = 0.05$, $\ell = window\ size = 300$, $mergedist = 0.1$, $\alpha = 0.1$).

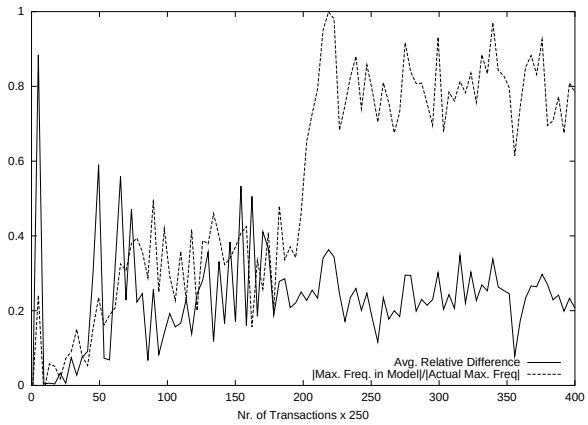

Figure 8.4: The model compared with the actual situation for the `MSNBC` dataset ($n = 17$, $minsupp = 0.05$, $\ell = window\ size = 1,000$, $mergedist = 0.1$, $\alpha = 0.1$).

itemset creation after seeing 10,000 transactions.

The processing time of the algorithm strongly depends on the support threshold *minsupp* one chooses. The lower *minsupp* is chosen the more points

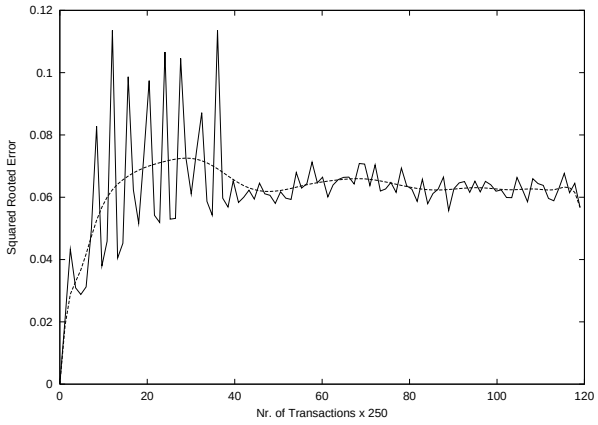

Figure 8.5: Root squared error between the real support and the approximated support for the `MSNBC` dataset ($n = 17$, $minsupp = 0.05$, $\ell = window\ size = 1,000$, $mergedist = 0.1$, $\alpha = 0.1$), with a Bézier curve.

the model will contain eventually and so processing time will get worse. Figure 8.6 shows that the average processing time for each transaction gets worse as the model contains more itemset points. However, Figure 8.7 shows that, for the `soybean` dataset, the number of points in the model eventually stabilizes. For each transaction we adapt the distances between points a number of times. In the case of the `soybean` dataset we randomly choose pairs 40,000 times in order to push or pull them, depending on their co-occurrence. Obviously one way of speeding up processing is to take a smaller number than 40,000 times or skip adapting distances from time to time.

8.5 Conclusions and Future Work

The algorithm presented in this chapter generates a co-occurrence model of (approximately) maximal frequent itemsets. This gives the user a quick overview on the patterns and how they occur in the stream.

The co-occurrence distance of patterns is computed by pushing apart or pulling together patterns in a two-dimensional space. Pushing was done when only one of the patterns occurs and pulling if they occur together. This distance is used to merge sufficiently long existing patterns together, only if support is larger than a user-defined threshold. This is because we want only maximal frequent itemsets (itemsets that are often a subset of a

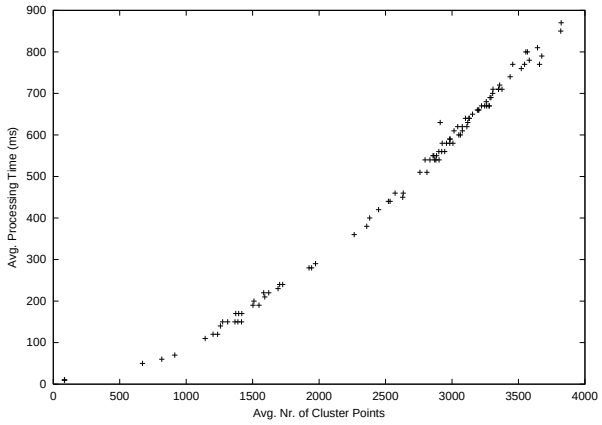

Figure 8.6: Transaction processing time in milliseconds for different model sizes for the real dataset ($n = 84$, $minsupp = 0.2$, $\ell = window\ size = 300$, $mergedist = 0.1$, $\alpha = 0.1$).

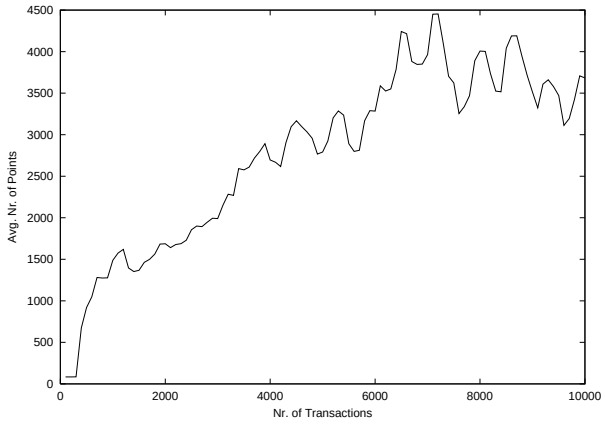

Figure 8.7: Development of model size as transactions of the real dataset are processed ($n = 84$, $minsupp = 0.2$, $\ell = window\ size = 300$, $mergedist = 0.1$, $\alpha = 0.1$).

transaction but they are never a subset of a bigger frequent itemsets) such that the model does not grow too big. Finally points are split if they happen to occur less than expected. Splitting and merging is required because the model cannot contain all patterns.

In the future we want to focus more on the applications of our algorithm and how it is best used in the analysis of streams. Furthermore we like to examine the support estimates in more detail, and see how extra parameters (e.g., to determine the threshold age for splitting) can be employed.

9 Mining Web Access Data and the Interpretation of Patterns

A common application of data mining with semi-structured data is the mining of web access data in which the browsing behaviour of users is stored. For example, one can store the information whether pages of the website are visited by an user. In this case each user will be represented by an *itemset*: the set of pages she or he has visited. However, one can also store the order of pages visited and in this way the behaviour of each user is stored as a *sequence*: the series of pages he or she has visited. Finally, one can store each page as a node and make connections between them if the user "travels" from one page to the other; in this way user behaviour is stored as a graph. In this chapter we will apply our methods to three "access log" datasets. In some cases we choose different structures for the records, e.g., a dataset of sequences is converted to a dataset of itemsets or graphs.

We hope to obtain a better understanding of the methods and the situations in which they are most useful. To this end we focus in this chapter on how the resulting patterns can be (further) interpreted. This chapter can best be seen as a case study where an expert of the website, called *the analyst*, uses a *tool suite*, consisting of all our techniques combined, to analyse the access log of her website in order to discover interesting user-behaviour. We will go through a number of patterns and try to assess whether they are useful.

We will use the following three data sets about visits to web-pages as input:

- The `liacs` dataset is based on an access log of the website of LIACS,

the Computer Science institute of Leiden University. Each of the 1,488 transactions arrive in order in half-hour blocks.

- The `one-visitor` data set stores the webpages accessed by one heavy user of the `portalexecutivo.com` website. One transaction consists of a set of pages accessed during a particular day. Some days there is no access and hence some of the 1,603 transactions are empty. Webpages are categorised resulting in a total of 185 possible items for every transaction.

- The `msnbc.com` dataset is comprised of Internet Information Server (IIS) logs for `msnbc.com` and news-related portions of `msn.com` for the entire day of September, 28, 1999. The dataset contains 989,825 sequences of 17 possible categories viewed by a user within 24 hours. This dataset was also used by Cadez et al. in [8].

In this chapter experiments concerning balanced or consecutive occurring itemsets will be done on the `liacs` and `one-visitor` datasets, because for these methods the time order must be present. For the `msnbc.com` dataset there is no time order given for the transactions, hence it is only used in experiments concerning sequences and graphs.

The rest of the chapter is structured as follows. In Section 9.1 we discuss consecutive and balanced patterns. In Section 9.2 we look at the interpretation of maximal discriminating patterns and in Section 9.3 the co-occurrence of subgraphs is discussed.

9.1 The Itemset View

We first apply our techniques for mining balanced (fixed interval in occurrence, Chapter 3) and consecutive itemset patterns (occurring very close after each other, Chapter 2) on the `liacs` dataset.

In the result section of Chapter 3 patterns found in the `liacs` dataset were presented. There only a small number of patterns was found, and even after a more intensive search with different parameters we were unable to find more patterns. This can be caused by the log not containing more balanced behaviour or because surprising patterns might be skipped due to the lack of background knowledge. Mining balanced and consecutive patterns is best combined in one tool, with an up-to-date log file, used by an expert with

inside knowledge of the website. However in the case of the `liacs` dataset many parts are stemming from many different employees.

The `liacs` dataset was not mined before using consecutive support. Mining was done and some interesting consecutive patterns were discovered. The following parameters were used:

- The minimal (consecutive) support was set to 400, indicating the minimal value for considering a pattern to be frequent.
- The ρ value was set to 1,000, rewarding the re-occurrence of a pattern.
- The σ value was set to 500, punishing the gaps between re-occurrences.

The *first pattern* that was found was a consecutive pattern where the webpage of a PhD student of the Imaging Group and the main page of the Imaging Group are accessed. An expert could use this information to see if the imaging group website is perhaps hard to reach. Possibily most browsing users access it via the website of the student; on the other hand this pattern could exist because the PhD student did an interesting discovery and many people were accessing his findings via the imaging group website. In Figure 9.1 we see this pattern. Figures like this plot the O-series of a pattern (see Chapter 2). In the displayed period we see consecutive phases and then a "big" gap. The "big" gaps mostly begin several hours after working hours and so we can probably conclude the pattern is caused by humans (as opposed to softbots) within the timezone of the university. Further knowledge of the Imaging Group website is needed to analyze this pattern further.

A *second pattern* is the MATLAB Online Reference Documentation, a popular mathematical program, that has been accessed consecutively. This can be caused by students that need to use the complex program for their assignments: they need to constantly revisit this reference manual. A website analysist could now investigate if the usability of the MATLAB documentation can be improved.

Finally we look at a *third pattern* where the website of the Imaging Group PhD is accessed together with the site of the High Performance Computing Group. Imaging and High Performance Computing often work together and one could add easy links to each others pages. Perhaps they could make an overview of their cooperation.

Some websites have a system were users login and in such a case one could focus on a single user. We will consider the `one-visitor` dataset. It stores

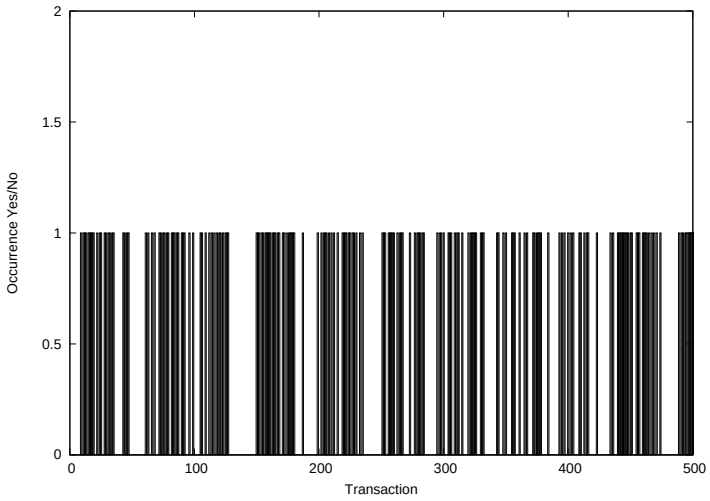

Figure 9.1: The consecutive occurrence of the imaging group website and its PhD Student.

the webpages accessed by one heavy user of the former `PortalExecutivo.com` website on a daily basis. One transaction consists of pages accessed during one day. Some days there is no access and some of the 1,603 transactions are empty. Webpages are categorised; there are 185 possible items for every transaction.

In Chapter 3 we already searched balanced patterns in this dataset. A more detailed look at the dataset revealed new patterns. General articles, research and education are accessed with a balanced interval, as seen earlier. New is that this interval is even more balanced when we include the seminaries part of the website. This is shown in Figure 9.2: for one period this pattern occurs very regular. Perhaps the user was in training in that period. The website might be able to use this information to offer seminaries particular to certain articles.

A different pattern is the "Daily News" part where the website of a particular bank is viewed for stock analysis in a balanced fashion. As an example a website analyst might suggest to add a summary of the stock analysis of a business to the "Daily News" when the news concerns this enterprise. In Figure 9.3 we see the user accessing this pattern in a stable way for a short period. This pattern has several of these short periods, see Figure 9.4.

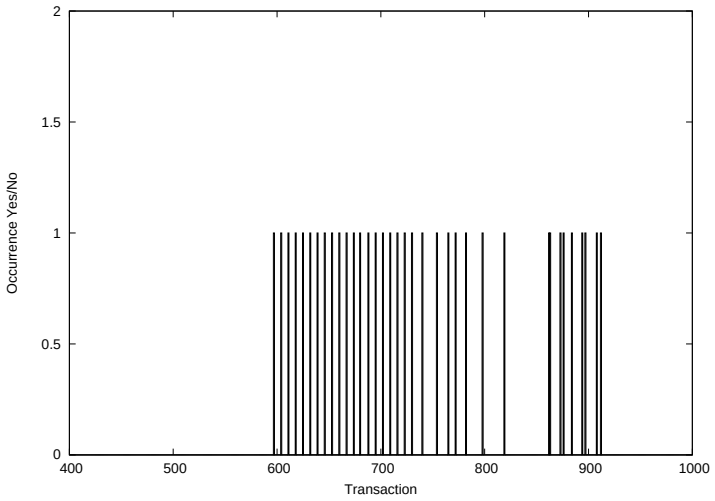

Figure 9.2: The stable occurrence of general articles, research, education and seminaries.

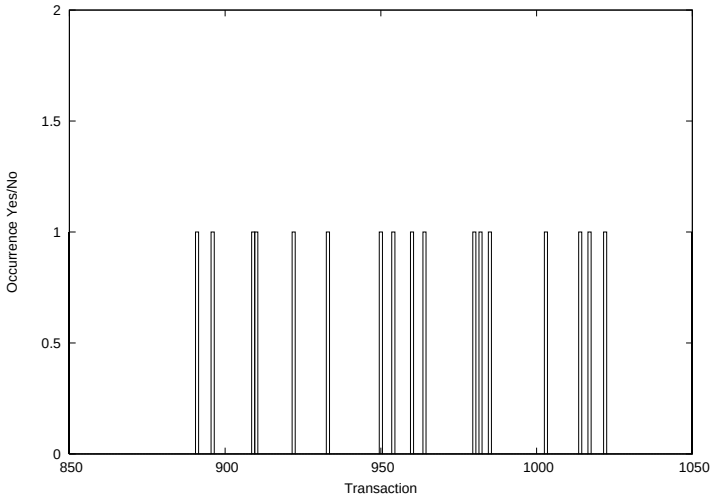

Figure 9.3: The stable occurrence of "Daily News" and stock analysis.

When we mine for consecutive patterns (with parameters equal to the ones used for the `liacs` dataset), we find some patterns similar to a stable pattern. E.g., we found the research section and general articles to be ac-

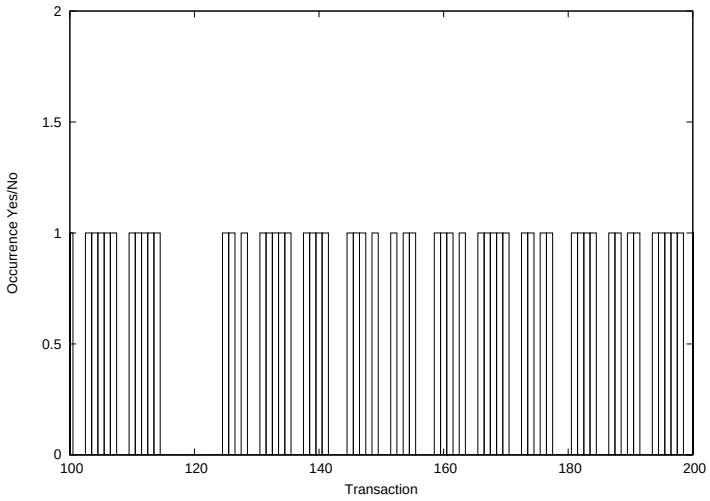

Figure 9.4: The occurrence of "Daily News" and stock analysis is also consecutive.

cessed consecutively (consecutive support: 25). Other patterns were a equal to a stable pattern, e.g., "Daily News" and the stock analysis were consecutively accessed by the user (consecutive support: 24). This is possible because the minimal standard deviation for the balanced patterns was low. Figure 9.4 shows how the "Daily News" and stock analysis occurs consecutively.

9.2 Mining with Sequences

In this section we will use the dataset from the Internet Information Server (IIS) logs for `msnbc.com` and news-related portions of `msn.com` for the entire day of September, 28, 1999. The original dataset contains 989,825 sequences of 17 possible categories viewed by a user within 24 hours. It was also used in [8].

The *case description* for sequence mining is the following: The proprietor of `msnbc.com` wants to know if people visiting the frontpage have a different behaviour in comparison with people that do not view the frontpage. The proprietor would like to know which parts of the frontpage could be made more clear. Also she would like to know particular behavioural patterns for news, tech, health, business and travel users. Finally she is interested in the

behavioural differences between the users of the msn news, news and local news sections and between the msn sport and sport sections.

9.2.1 Sequential Patterns

In our analysis of users that visit the frontpage we discovered (*minsup* = 5,000 visitors):

1. They visit more the news, many times for one user.
   (relative difference: 7.8%, *RMSD*: 0.19)

2. More often they first visit news and than business.
   (relative difference: 2.7%, *RMSD*: 0.55)

3. More often they first visit living and continue with news.
   (relative difference: 2.6%, *RMSD*: 0.49)

4. After seeing the news they more often see what is on-air.
   (relative difference: 2.6%, *RMSD*: 0.34)

The improved algorithm used in this subsection also gives the *RMSD*-value, indicating the "suprisingness" of the pattern. This measure is dicussed in more detail in Appendix A. Suffices to say that looking at the size proportions between datasets, it calculates how much the occurrence of a pattern deviates from this.

The case continues: with our earlier analysis we found that news is important to frontpage users, frontpage readers more often read the news. The proprietor now wants to know if news readers use the website in a different way, e.g., perhaps the user is more interested in business.

In our analysis of users who visit news, we discovered (*minsup* = 5,000 visitors):

1. News visitors more often go from the frontpage to business.
   (relative difference: 5.2%, *RMSD*: 0.18)

2. News visitors more often access the technology section, often going via the frontpage.
   (relative difference: 5.9%, *RMSD*: 0.27)

3. Sport if also more often viewed, mostly in combination with the frontpage.
   (relative difference: 3.4%, *RMSD*: 0.13)

There are two categories of sport: sport and msn sport. Also three categories of news: news, local news and MSN news.

In the comparison of sport and MSN sport visitors we discovered (*minsup* = 5,000 visitors):

1. Sport visitors visit the frontpage more and more repeatedly than MSN sport visitors.
   (relative difference: 29.4%, *RMSD*: 0.25)

2. Sport visitors more often read also the news, often going via the frontpage.
   (relative difference: 19.6%, *RMSD*: 0.23)

3. Living is more often visited by sport visitors, also often using the frontpage.
   (relative difference: 7.5%, *RMSD*: 0.29)

4. MSN sport visitors more often visit MSN news.
   (relative difference: 15.2%, *RMSD*: 0.27)

Based on this data we can improve the website. E.g., we can add a direct link to the sport section to items on the frontpage.

In the comparison of news and MSN news visitors we discovered (*minsup* = 5,000 visitors):

1. News visitors visit the frontpage more and more often than MSN news visitors.
   (relative difference: 3.2%, *RMSD*: 0.22)

2. News visitors more often visit business via the frontpage.
   (relative difference: 9.6%, *RMSD*: 0.29)

3. Health is also more often visited via the frontpage
   (relative difference: 7.8%, *RMSD*: 0.26)

4. Business, on-air and sport are visited more often.

5. The Technology section is accessed repeatedly.
   (relative difference: 3.3%, *RMSD*: 0.23)

6. MSN news visitors more often visit local news, and do so repeatedly.
   (relative difference: 4.3%, *RMSD*: 0.10)

7. MSN news visitors more often visit on-air once, but news visitors repeat their visit to on-air more.

9.3 Co-Occurring Subgraph

The graph mining technique in this thesis (see Chapter 6 and Chapter 8) focuses on the the co-occurrence of frequent subgraphs. Applying this technique to our current running example of mining web log data is difficult. First we have to convert the records of the `msnbc.com` dataset from sequences into graphs. However, for one sequence many different graphs are possible, e.g, the sequence (as described in Chapter 5) (`A`, `B`, `C`, `A`). This sequence can result in a graph of 4 nodes sequentielly connected to each item of the sequence. However it could also be described with a graph consisting of three nodes `A`, `B` and `C` and a connection from `C` back to `A`, see Figure 9.5.

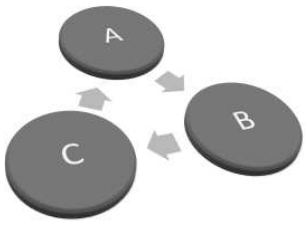

Figure 9.5: Graph consisting of three nodes.

We decided to use undirected graphs (were edges have no direction) and to have each item in the sequence as a node in the graph. Two nodes are connected if the corresponding items follow each other directly in the sequence. This way of generating graphs was chosen because we wanted to show that co-occurrence visualization can work in this "web" setting and we want to guarantee that no information is lost.

A problem with the `msnbc.com` dataset is that most sequences are only a few items (page views) long. These short sequences make co-occurrence rare and for this reason it was decided to adopt the following case: The owner of the website wants to know if certain behaviour of "long staying" *msn sports* users commonly involves certain other behaviour.

Obviously we are more interested in the co-occurrence of patterns that do not have a parent-child relation (see also Chapter 8). Furthermore, "long stay" users are defined as users that visit at least five parts of the website (which can be five times the same part).

The methods proposed in Chapter 6 are applied to build the model in Figure 9.6. We see that some patterns are placed more close together, indicating that they co-occur more than others. However, it is not very

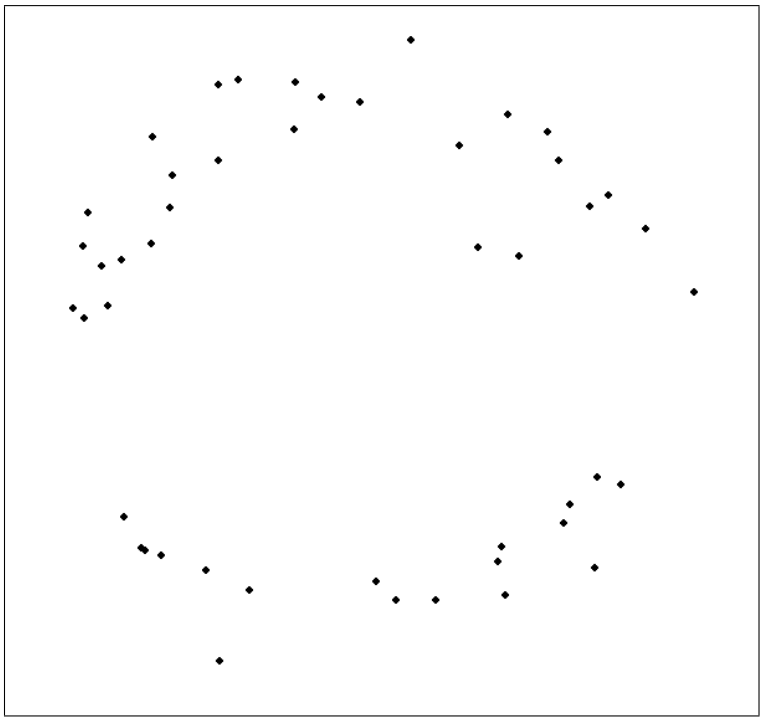

Figure 9.6: The co-occurrence model for the 47 patterns found in the msn-sports section of the `msnbc.com` dataset ($minsup = 2,000$).

apparant. Other patterns are put more far apart, this indicates they co-occur less.

In Figure 9.7 we connect those points where the corresponding patterns have a lower than 0.6 co-occurrence value (the closer this value is to 0 the more these patterns co-occur).

The different groups of co-occurring patterns are not clearly recognizable from Figure 9.6. In order to make groups more clear we wanted to transform the distance measure. We took the square root of the absolute distance between patterns while keeping the original negativity or positivity of the distance. This is made more clear with Equation 9.1, where this new distance $dist'(g_1, g_2)$ is defined. In this way close patterns are put more close and far apart patterns are put more clearly apart.

$$dist'(g_1, g_2) = sgn(dist(g_1, g_2)) \cdot \sqrt{|dist(g_1, g_2)|} \tag{9.1}$$

where $dist(g_1, g_2)$ is as defined ealier in Equation 6.1:

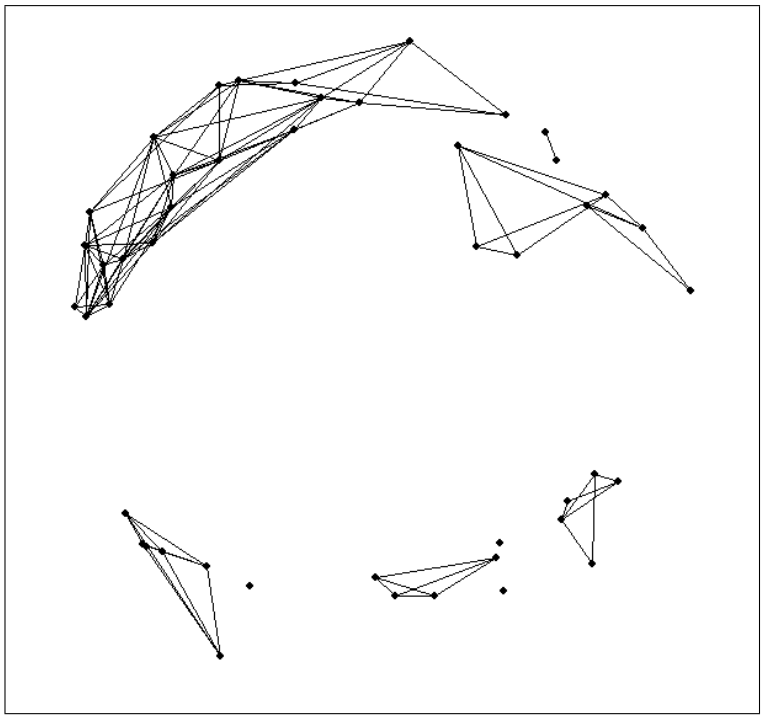

Figure 9.7: The co-occurrence model for the `msnbc.com` dataset where we connect patterns that have a minimal co-occurrence of 0.6 ($minsup = 2,000$).

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

Figure 9.8 clearly shows three clusters of patterns co-occurring. That they really co-occur more is shown in Figure 9.9. The groups have no connections between "members", the patterns co-occur too little and as a consequence are placed further apart.

In Figure 9.8 one can see three clusters of patterns: one tight cluster in the top, one cluster in the right bottom and a little bit more loose cluster on the left.

The cluster in the top contains patterns that of course have the msnsport page, but also local-news, weather and miscellaneous. These users can be seen as the "generally interested" users. Some co-occurring patterns in

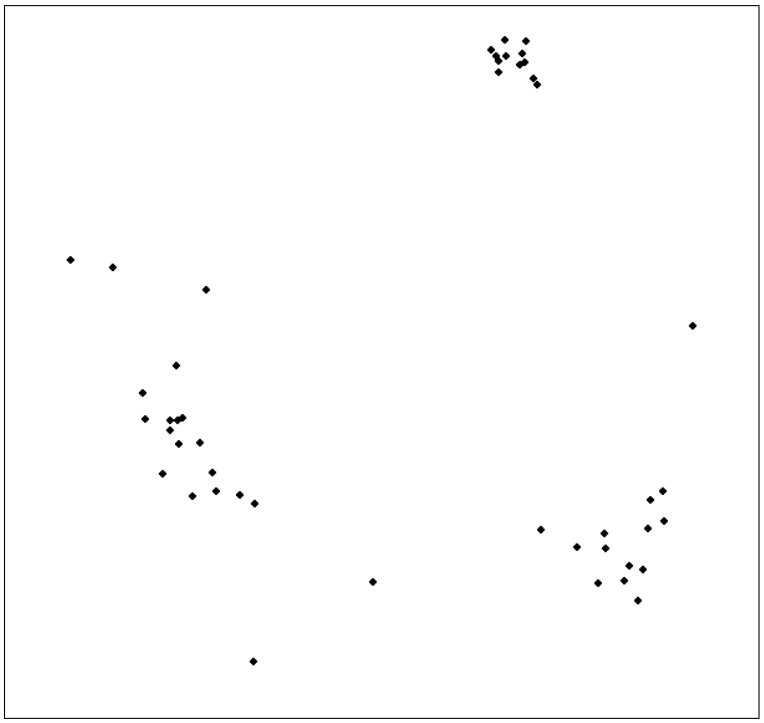

Figure 9.8: The co-occurrence model for the 47 patterns found in the msn-sports section of the `msnbc.com` dataset, using the distance function $dist'(g_1, g_2)$ ($minsup = 2,000$).

this cluster are:

- {`local-news`, `local-news`}, with a support of 2,092.
- {`msn-sport`, `weather`, `weather`}, with a support of 2,092.
- Three views of msn-sport and then some users continue with miscellaneous subjects, with a support of 2,370.

The cluster in the right-bottom contains patterns where users are viewing the msn-news section and the general sports section. These users can be seen as the 'News and sport' users. Some co-occurring patterns in this cluster are:

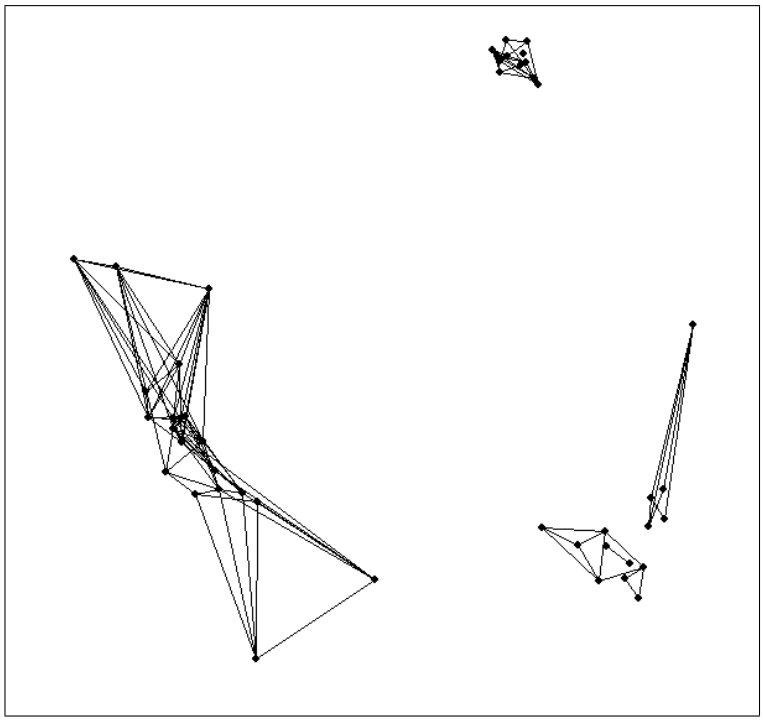

Figure 9.9: The co-occurrence model for the `msnbc.com` dataset where we connect patterns that have a minimal co-occurrence of 0.6, using the distance function $dist'(g_1, g_2)$ ($minsup = 2,000$).

- {`msn-news`, `msn-news`}, with a support of 3,300.

- {`msn-sport`, `sport`, `msn-sport`}, with a support of 2,910.

The cluster in the left-bottom mainly contains some combination of general sport and msn-sport. These users are seen as the "sport watchers". Some co-occurring patterns in this cluster are:

- {`sport`, `sport`, `msn-sport`, `msn-sport`}, with a support of 5,138.

- {`sport`, `sport`, `sport`, `sport`}, with a support of 4,343.

So we see in the example case that the building of a co-occurrence model

can be used to discover different clusters of user behaviour and in such a way recognizing different user types.

9.4 Conclusions

The different techniques proposed in this thesis can handle the different types of semi-structured data. Our technique of mining consecutive and balanced patterns are made to handle itemsets and difference mining was optimized for dealing with sequential data. Finally our method of modeling co-occurring patterns was made to analyze subgraph co-occurrence.

Applying each technique on the same dataset requires us to convert the records to the specific semi-structured data. For graphs this is difficult, because their can be many possible graphs representing a sequence.

However, we showed in this chapter that we can use balanced pattern mining and consecutive pattern mining to find interesting pattern occurrence intervals in web log data. It is important for the website analyst to keep the order of the transactions as they arrive. This could not be guaranteed for the `msnbc` dataset and we could not apply these techniques on this dataset.

With difference mining we discovered many interesting patterns in the `msnbc` dataset this shows how this technique can be applied to sequential data and what one could hope to find. We proposed a new method of evaluating the surprisingness of pattern occurrence within different datasets and showed how one can prune using this measure.

Finally we converted the sequences of the `msnbc` dataset to simple graph structures and we discovered some interesting co-occurring patterns. The co-occurrence became especially clear after we transformed the distance function using root.

Conclusions

In this thesis we started by investigating different ways of counting the occurrence of patterns, *pattern evaluation*. In some case some patterns where shown to occur many times in transactions occurring almost right after one another in the dataset, they occurred *consecutive*. In other cases patterns where occurring and then not occurring for the same amount of time before they occurred again, these patterns we call either *stable* or *balanced*. We call patterns balanced if certain time intervals occur a minimum amount of time and when the standard deviation for the complete interval of occurrences is below a maximum. In comparison with stable patterns, the pruning threshold for balanced patterns is thought to be more intuitive to users. With it the user only indicates the number of times at least one intermediate distance should occur. Such a distance is the number of transactions between two occurrences of the pattern (we consider only distances below a maximal distance).

We discussed how principles, used to get extra information with traditional support, can also be used in combination with the new way of evaluating patterns. For instance the combination of consecutive support and the h-confidence threshold enables us to find small bursts of patterns. In traditional ways of support counting one uses h-confidence to find *hyperclique patterns*. These types of patterns have a *strong affinity* between items: the presence of $x \in P$, where P is an itemset or clone, in a transaction strongly implies the presence of the other items or patients in P. For consecutive occurring patterns h-confidence filters out noise and consecutive support amplifies the bursts of patterns.

Not for all data sets consecutive, stable or balanced patterns are of interest. The transaction in the dataset at least should have an order, they should be sorted. Furthermore the time between transactions should be known or all transactions should take equal time.

For consecutive support we applied the technique on a dataset of itemsets where each transaction is a part of the chromosone. A patient occurs as

an item in each of these transactions if they deviate from normal for this part. By using consecutive support we found patterns that occurred many times and very close together in the chromsone. In the biological problem the items are individuals and the transactions are "clones", pieces of the chromosome that might occur more or less often than in a healthy individual. Patterns in close transactions are better because they are close together in the chromosome and are biologically more significant than patterns that are far apart and in different chromosomes.

For balanced and stable patterns we used the techniques to find patterns in the use of a website. We found some parts of the website to be used many times in combination and many times with equal time between occurrences.

In this thesis we also introduced and compared two sequential pattern mining algorithms by using knowledge from the application area of protein sequence analysis. Given some assumptions, we can improve mining for the maximal discriminating patterns. The effectiveness depends on the quality of the assumptions, e.g., how probable a discriminating pattern is within a certain time window. Our method also depends on the discriminative power of the patterns. Pruning will be less effective if this is low, even though we might find the maximal discriminating patterns quickly. It is shown that using probable time windows in protein sequences can speed up the search. Protein sequences are long but contain only a few types of items; constraints, e.g., a time window constraint, are required to make the discovery of patterns in these sequences tractable. The *time window constraint* means that the distance between the first and last item of the sequence is bounded by some constant. For example if all sequences in a database are equal to (`A`,`C`,`G`,`Q`), and the time window is 2, then (`A`,`Q`) is not a frequent pattern because the distance between the `A` and `Q` in the occurrences is more than 2.

Presenting data mining results to the user in an efficient way is important. In this work we proposed different ways of visually presenting frequent subgraphs (patterns). All these methods use co-occurrence of patterns in some way. Firstly proposed a tool for browsing the lattice by extending or shrinking frequent subgraphs towards bigger or smaller frequent subgraphs. This browsing application also used co-occurrence to propose other groups of graph patterns that either co-occur with the particular pattern or not at all.

Secondly we proposed a way of constructing a co-occurrence model for subgraphs that enables quicker exploration of occurrence data. The co-occurrence distance of patterns is computed by pushing apart or pulling together patterns in a 2-dimensional space (the model). Pushing was done when only one of the patterns occurs and pulling if they occur together.

Before building a co-occurrence model the algorithm first forms groups of structural related graph patterns that also co-occur many times. This reduces model building time, but more importantly it will quickly show the biologists structural unrelated groups of patterns that co-occur, which is more interesting.

Not all dataset have a clear end, in the case of streams (of itemsets) the dataset is potentially endless. In this work we introduce an algorithm that generates a co-occurrence model of approximately maximal frequent itemsets for streams of itemsets. This gives the user a quick view on the patterns, frequent subsets, in the stream and how they occur in the stream. The co-occurrence distance is used to merge sufficiently long existing patterns together if support is larger than a user-defined threshold, because we want only maximal frequent itemsets (itemsets that are often a subset of a transaction but they are never a subset of a bigger frequent itemsets) such that the model does not grow too big. Finally points are split if they happen to occur less than expected. Splitting and merging is required because the model cannot contain all patterns.

Using the distance between transactions like it is done in this thesis is an interesting area of research. In the future it should be examined if consecutive support, stable and balanced patterns enable us to visualize even more types of behavior. Next to using these existing methods to find new behavior, in the future we hope to discover new ways of counting occurrence in order to find interresting patterns. For stable patterns and balanced patterns we would like to see if we can speed-up the search, e.g., by using heuristics. And we want to see how it can be combined with principles like h-confidence. For balanced (and stable) patterns we hope research new ways of visualizing the results such that interesting patterns become more appearant. Later research hopefully also looks at the probable time window and how this time window can automatically be discovered. Finally in the future we would like to develop innovative ways of browsing the lattice and quickly select interesting patterns.

A A measure of "surprisingness"

A.1 Root Mean Square Deviation

Before we started our analysis of Chapter 9 we added an extra measure to the output provided by the difference miner: *root mean square deviation* or *RMSD*. We will first explain this measure for "surprisingness". It will allow us to find frequent patterns particular for one or a few groups within a dataset. The added value of this measure will be the independence from the strength of the frequency. Related to this work is the study of surprisingness as done by Freitas in [16].

Say we have m groups for one dataset $\mathcal{D} = \mathcal{D}_0 \cup \mathcal{D}_1 \cup \ldots \cup \mathcal{D}_{m-1}$; and a pattern p occurs in at least one group. It would be interesting if the occurrence of this pattern is mainly in one group. We let $sup(p, \mathcal{D}_i)$ be the number of occurrences of pattern p in group $\mathcal{D}_i$ $(0 \leq i < m)$. We define the total pattern occurrence as $t = \sum_{i=0}^{m-1} sup(p, \mathcal{D}_i)$. Now we can calculate the supports we would expect when there is no apparent deviation with the proportions between $\mathcal{D}_x$ and $\mathcal{D}$:

$$E[sup(p, \mathcal{D}_x)] = (|\mathcal{D}_x|/n) \cdot t \tag{A.1}$$

where $n = |\mathcal{D}_0| + \ldots + |\mathcal{D}_{m-1}|$.

The *RMSD*-value is now calculated between expected and real proportions in occurrence, see Equation A.2 and A.3:

$$RMSD(p) = \sqrt{\frac{1}{m} \sum_{i=0}^{m-1} (E[sup(p, \mathcal{D}_i)]/t - sup(p, \mathcal{D}_i)/t)^2} \tag{A.2}$$

$$= \sqrt{\frac{1}{m}\sum_{i=0}^{m-1}(|\mathcal{D}_i|/n - sup(p,\mathcal{D}_i)/t)^2} \quad \text{(A.3)}$$

The $RMSD(p)$-measure is strongly related to the sum of the individual variances, but $|\mathcal{D}_i|/n$ and $sup(p,\mathcal{D}_i)/t$ are not independent. Furthermore t makes it a measure of deviation between mean occurrence in $\mathcal{D}$ and mean occurrence in $\mathcal{D}_i$.

If $n = |\mathcal{D}_0|+|\mathcal{D}_1|$ then it can easily be proven that $(|\mathcal{D}_0|/n - sup(p,\mathcal{D}_0)/t)^2 = (|\mathcal{D}_1|/n - sup(p,\mathcal{D}_1)/t)^2$ and in that case we could also use Equation A.4:

$$RMSD(p) = \sqrt{(|\mathcal{D}_0|/n - sup(p,\mathcal{D}_0)/t)^2} \quad \text{(A.4)}$$

The value for relative difference depends strongly on the value of relative support for the pattern p in relation to the group sizes of $\mathcal{D}_0$ or $\mathcal{D}_1$. Less frequent patterns more often have a low relative difference.

In the next Example A.1.1 we will give an example of a pattern with an interesting difference in occurrence/support. However, the absolute relative difference is small because the total support is small. With the *RMSD*-value we see that there is an interesting deviation from the support we would expect. The expected support is computed as if the proportions of support were equal to the proportions of the groups.

Example A.1.1 Assume that we have pattern p occurring 100 times in $\mathcal{D}_0$ and 8 times in $\mathcal{D}_1$ and that $|\mathcal{D}_0| = |\mathcal{D}_1| = 1000$. The absolute relative difference is (only): $|100/1000 - 8/1000| = 0.09$. However, $RMSD(p) = 0.60$.

A few more occurrences in a small group have a big influence on relative difference. Example A.1.2 shows how changes in a small group have less influence on the *RMSD*-value. This can not really be called an advantage of the one or the other. In the case of *RMSD* we normalize on the total support of pattern p giving smaller groups (potentially) less influence than a bigger group. For the *RMSD*-value a change in occurrence is equal, independent from the group in which it changes. In the case of relative difference all groups have equal influence and one extra occurrence in a small group has more weight.

Example A.1.2 Assume that we have pattern p occurring 500 times in $\mathcal{D}_0$ and 80 times in $\mathcal{D}_1$. Furthermore $|\mathcal{D}_0| = 1000$ and $|\mathcal{D}_1| = 100$. The absolute relative difference will be: $|500/1000 - 80/100| = 0.3$.

However, for $RMSD(p)$ we calculate $1000/1100 \cdot 580 \approx 527.27$ and $100/1100 \cdot 580 \approx 52.72$, and so $RMSD(p) \approx 0.07$.

If we prefer equal influence, independent from group size, then we can adapt the *RMSD*-value:

$$RMSD'(p) = \sqrt{\frac{((E[sup(p,\mathcal{D}_0)] - sup(p,\mathcal{D}_0))/max(p,\mathcal{D}_0))^2 + ((E[sup(p,\mathcal{D}_1)] - sup(p,\mathcal{D}_1))/max(p,\mathcal{D}_1))^2}{2}}, \tag{A.5}$$

where $max(p,\mathcal{D}) = max(E[sup(p,\mathcal{D})], sup(p,\mathcal{D}))$.

The χ^2 measure is a good measure used for correlation mining, e.g., in the case of CorClass [78]. However, the height of the χ^2 value is influenced by the total size of the datasets and the corresponding support of the pattern relative to that size. In Example A.1.4 we notice that also with χ^2 we do not see the pattern of Example A.1.2 as interesting. This is because χ^2 also takes the proportions into account.

Example A.1.3 Assume again that we have pattern p occurring 100 times in $\mathcal{D}_0$ and 8 times in $\mathcal{D}_1$. Furthermore $|\mathcal{D}_0| = 1000$ and $|\mathcal{D}_1| = 100$.
We have the following table for frequencies:

	In $\mathcal{D}_0$	In $\mathcal{D}_1$	total
p	100	8	108
$\neg p$	900	92	992
	1000	100	1100

The expected frequency will be the following:

	In $\mathcal{D}_0$	In $\mathcal{D}_1$	total
p	98.2	9.8	108
$\neg p$	901.82	90.2	992
	1000	100	1100

Hence $\chi^2(p,\mathcal{D}_0,\mathcal{D}_1) \approx 0.40$ which indicates that the frequency is close to expected.

However χ^2 is influenced by the total number of occurrences of a pattern within the data sets:

Example A.1.4 Assume that we have pattern p occurring 200 times in $\mathcal{D}_0$ and 16 times in $\mathcal{D}_1$ and that $|\mathcal{D}_0| = |\mathcal{D}_1| = 1000$.
Now we have the following table for observed frequencies:

	In $\mathcal{D}_0$	In $\mathcal{D}_1$	total
p	100	8	108
$\neg p$	900	992	1892
	1000	1000	2000

The expected frequencies will be as follows:

	In $\mathcal{D}_0$	In $\mathcal{D}_1$	total
p	54	54	108
$\neg p$	900	992	1892
	1000	1000	2000

Now $\chi^2(p, \mathcal{D}_0, \mathcal{D}_1) \approx 82.8$ which indicates this difference is interesting.

Now let us assume that we have a pattern q occurring twice as much in both datasets, 200 times in $\mathcal{D}_0$ and 16 times in $\mathcal{D}_1$. One could say they are proportional equally interesting, but $\chi^2(q, \mathcal{D}_0, \mathcal{D}_1) \approx 175.7$ and $RMSD(q) = RMSD(p) \approx 0.60$.

With the *RMSD*-value one can see which patterns deviate strongly from the expected proportions, independent from total occurrence within the dataset. In this way only one measure (support) is influenced by the size of the dataset.

With the χ^2 measure find proportional different patterns where a larger number of occurrences is important. And *RMSD* all proportional surprising patterns and leaves the minimal size for the *minsupp* threshold to decide.

A.2 Pruning with RMSD

Assume that we have a pattern $q = p \succ \{e\}$, where p and q are sequences of items and e is a item, and $\succ$ denotes that $\{e\}$ follows p. In this case we call q an extension of p.

Before we actually go to the database to count its occurrence, we want to know if it can achieve a certain minimal *RMSD*-value. Given a user-defined $minsup_x$ threshold for minimal support in group $\mathcal{D}_x$, we can say that the squared deviation for D_x will maximally be:

$$max(\mathcal{D}_x/n - minsup_x/t, \mathcal{D}_x/n - sup(p, \mathcal{D}_x)/t) \tag{A.6}$$

This maximal *RMSD*-value can now be used to prune the search space either via a user defined minimal *RMSD*-value and/or by searching for a maximal number of patterns. In Figure A.1 a user defined minimal *RMSD*-value of 0.3 improves runtime.

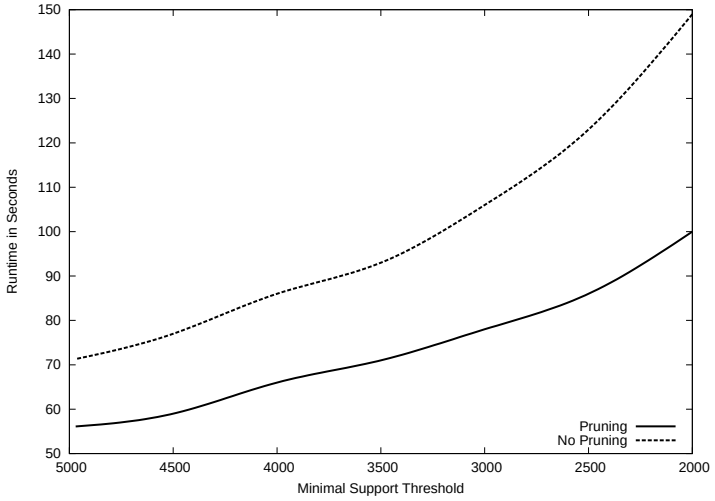

Figure A.1: Using the pruning with *RMSD* for different minimal support settings and a minimal *RMSD*-value of 0.3.

Bibliography

[1] Aggarwal, C.C., Han, J., Wang, J., and P.S. Yu.: *A framework for clustering evolving data streams*, In 29th International Conference on Very Large Data Bases (VLDB'03), 2003, pp. 81–92.

[2] Aggarwal, C.C. and P.S. Yu.: *A Framework for Clustering Massive Text and Categorical Data Streams* In SIAM Conference on Data Mining (SDM'06), 2006, pp. 477–481.

[3] Agrawal, R., Imielinski, T., and Srikant, R.: *Mining Association Rules between Sets of Items in Large Databases*, in Proceedings of ACM SIGMOD Conference on Management of Data, 1993, pp. 207–216.

[4] Agrawal, R., Srikant, R.: *Mining Sequential Patterns*, In Proceedings International Conference Data Engineering (ICDE 1995), 1995, pp. 3–14.

[5] Antunes, C., and Oliveira, A.L.: *Generalization of Pattern-Growth Methods for Sequential Pattern Mining with Gap Constraints*, in Machine Learning and Data Mining in Pattern Recognition (MLDM 2003), LNCS 2734, Springer, 2003, pp. 239–251.

[6] Bailey, J., Manoukian, T., Ramamohanarao, K.: *Fast Algorithms for Mining Emerging Patterns*, in Proceedings 6th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 2002), Lecture Notes in Artificial Intelligence 2431, Springer, 2002, pp. 39–50.

[7] Burdick, D., Calimlim, M., and Gehrke, J.: *MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases*, in 17th International Conference on Data Engineering (ICDE'01), 2001, pp. 443–453.

[8] Cadez, I.V. , Heckerman, D., Meek, C., Smyth, P., and White, S.: *Visualization of Navigation Patterns on a Website Using Model-Based Clustering*, in Knowledge Discovery and Data Mining, (KDD'02), 2000, pp. 280–284.

[9] Chang, J.H., and Lee, W.S.: *Finding Recent Frequent Itemsets Adaptively over Online Data Streams*, in 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03), 2003, pp. 487–492.

[10] Chang, J.H., and Lee, W.S.: *estWin: Online Data Stream Mining of Recent Frequent Itemsets by Sliding Window Methods*, in Journal of Information Science, 31(2), 2005, pp. 76–90.

[11] Chen, Y., Dong G., Han J., Wah, B., and Wang J.: *Multidimensional Regression Analysis of Time-series Data Streams*, in Proceedings 28th Int. Conference on Very Large Data Bases (VLDB 2002), 2002, pp. 323–334.

[12] Cong, G., Pan, F., Yang, J., and Zaki, M.J.: *CARPENTER: Finding Closed Patterns in Long Biological Datasets*, in Proceedings Conference on Knowledge Discovery in Data (SIGKDD 2003), 2003, pp. 637–642.

[13] Bruin, J.S. de, Cocx, T.K., Kosters, W.A., Laros, J.F.J., and Kok, J.N.: *Data Mining Approaches to Criminal Career Analysis*, in Proceedings 6th IEEE International Conference on Data Mining (ICDM 2006), 2006, pp. 171–177.

[14] Dong, G., Zhang, X., Wong, L., and Li, J.: *CAEP: Classification by Aggregating Emerging Patterns*, in Proceedings International Conference on Discovery Science (DS-1999), 1999, pp. 30–42.

[15] El-Hajj, M., and Zaiane, O.R.: *Parallel leap: Large-Scale Maximal Pattern Mining in a Distributed Environment*, In 12th International Conference on Parallel and Distributed Systems (ICPADS'06), 2006, pp. 135–142.

[16] Freitas, A.A.: *On Objective Measures of Rule Surprisingness*, in Proceedings of the Second European Symposium on Principles of Data Mining and Knowledge Discovery, LNCS 1510, Springer, 1998, pp. 1–9.

[17] Gao, H., Williams, C., Labute, P., and Bajorath, J.W.: *Binary Quantitative Structure-Activity Relationship (QSAR) Analysis of Estrogen*, in Journal of Chemical Information and Computer Sciences, vol. 39, 1999, pp. 164–168.

[18] Gedeck, P., and Willett, P.: *Visual and Computational Analysis of Structure-Activity Relationships in High-Throughput Screening Data*, in Current Opinion in Chemical Biology, vol. 5, 2001, pp. 389–395.

[19] Gibson, D., Kleinberg, J., and Raghavan, P.: *Clustering categorical data: An Approach Based on Dynamical Systems*, in 26th International Conference on Very Large Data Bases(VLDB'00), 2000, pp. 222–236.

[20] Giannella, C., Han, J., Pei J., Yan, X., and Yu, P.: *Mining Frequent Patterns in Data Streams at Multiple Time Granularities*, in Proceedings of the NSF Workshop on Next Generation Data Mining (NGDM 2002), 2002, pp.191–210.

[21] Gouda, K., and Zaki, M.J.: *Efficiently Mining Maximal Frequent Itemsets*, in Proceedings IEEE International Conference on Data Mining (ICDM'01), 2001, pp. 163–170.

[22] GPCRDB: Information System for G Protein-Coupled Receptors (GPCRs), Website `http://www.gpcr.org/7tm/`.

[23] Graaf, E.H. de, and Kosters, W.A.: *Clustering Improves the Exploration of Graph Mining Results*, in Proceedings Fourth IFIP Conference on Artificial Intelligence Applications & Innovations (AIAI'07), 2007.

[24] Graaf, E.H. de, J. Kazius, J.N. Kok, and Kosters, W.A.: *Visualization and Grouping of Graph Patterns in Molecular Databases*, in Proceedings Twenty-seventh SGAI International Conference on Artificial Intelligence (AI'07), 2007.

[25] Graaf, E.H. de, and Kosters, W.A.: *Mining For Stable Patterns: Regular Intervals Between Occurrences*, in Proceedings Eighteenth Belgium-Netherlands Conference on Artificial Intelligence (BNAIC06), 2006, pp. 149–155.

[26] Graaf, E.H. de, Kok, J.N., and Kosters, W.A.: *Displaying Co-Occurrence of Patterns in Streams for Website Usage Analysis*, in Proceedings Eighteenth Belgium-Netherlands Conference on Artificial Intelligence (BNAIC06), 2006, pp. 149–155.

[27] Graaf, E.H. de, and Kosters, W.A.: *Using a Probable Time Window for Efficient Pattern Mining in a Receptor Database*, in Proceedings of 3rd Int. ECML/PKDD Workshop on Mining Graphs, Trees and Sequences (MGTS'05), 2005, pp. 13–24.

[28] Graaf, E.H. de, and Kosters, W.A.: *Efficient Feature Detection for Sequence Classification in a Receptor Database*, in Proceedings Seventeenth Belgium-Netherlands Conference on Artificial Intelligence (BNAIC05), 2005, pp. 81–88.

[29] Graaf, E.H. de, Graaf, J.M. de, and Kosters, W.A.: *Using Consecutive Support for Genomic Profiling*, in Proceedings of the ECML/PKDD Workshop on Data and Text Mining for Integrative Biology.

[30] Graaf, E.H. de, Kok, J.N., and Kosters, W.A.: *Mining Balanced Patterns in Web Access Data*, in Proceedings of International Conference on Artificial Intelligence and Applications (AIA'08), 2006, pp. 149–155.

[31] Graaf, J.M. de, Menezes, R.X. de, Boer, and J.M., Kosters, W.A.: *Frequent Itemsets for Genomic Profiling*, in Proceedings 1st International Symposium on Computational Life Sciences (CompLife 2005), LNCS 3695, Springer, 2005, pp. 104–116.

[32] Hand, D., Mannila, H., and Smyth, P.: *Principles of Data Mining*, MIT Press, 2001, ISBN:0-262-08290-x.

[33] Hanke, J., Beckmann, G., Bork, P., and Reich, J.G.: *Self-Organizing Hierarchic Networks for Pattern Recognition in Protein Sequence*, Protein Science Journal 5, 1996, pp. 72–82.

[34] Hopgood, A.A.: *Intelligent Systems for Engineers and Scientists*, CRC Press, 2001, ISBN:0-8493-0456-3.

[35] Izrailev, S., and Agrafiotis, D.K.: *A Method for Quantifying and Visualizing the Diversity of QSAR Models*, in Journal of Molecular Graphics and Modelling, vol. 22, 2004, pp. 275–284

[36] Jovanoski, V., and Lavrač, N.: *Classification Rule Learning with APRIORI-C*, in Proceedings 10th Portuguese Conference on Artificial Intelligence (EPIA 2004), 2004, pp. 44–51.

[37] Kavsẽk, B., Lavrač, N., and Jovanoski, V.: *APRIORI-SD: Adapting Association Rule Learning to Subgroup Discovery*, in Proceedings International Symposium on Intelligent Data Analysis (IDA 2003), Lecture Notes in Computer Science 2810, Springer, 2003, pp. 230–241.

[38] Kohonen, T.: *Self Organizing Maps*, Volume 30 of Springers Series in Information Science, Springer, second edition, 1997.

[39] Kosters, W.A., and Wezel, M.C. van: *Competitive Neural Networks for Customer Choice Models*, in E-Commerce and Intelligent Methods, volume 105 of Studies in Fuzziness and Soft Computing, Physica-Verlag, Springer, 2002, pp. 41–60.

[40] Kramer, S., Raedt, L. De, and Helma, C.: *Molecular Feature Mining in HIV Data*, in Proceedings Conference on Knowledge Discovery in Data (SIGKDD 2001), pp. 136–143.

[41] Lameijer, E.W., Kok, J.N., Bäck, and Ijzerman, A.P.: *Mining a Chemical Database for Fragment Co-Occurrence: Discovery of "Chemical Clichés"*, in Journal of Chemical Information and Modelling, vol. 46, 2006, pp. 553–562.

[42] Lameijer, E.W., Tromp, R.A., Spanjersberg R.F., Brussee J., and Ijzerman, A.P.: *Designing Active Template Molecules by Combining Computational De Novo Design and Human Chemists Expertise*, in Journal of Med. Chem., vol. 50, 2007, pp. 1925–1932.

[43] Leleu, M., Rigotti, C., Boulicaut, and J.F., Euvrard, G.: *Constraint-Based Mining of Sequential Patterns over Datasets with Consecutive Repetitions*, in Proceedings 7th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 2003), LNAI 2838, Springer, 2003, pp. 303–314.

[44] Nijssen, S., and Kok, J.N.: *Multi-class Correlated Pattern Mining*, in Proceedings of the Fourth International Workshop on Knowledge Discovery in Inductive Databases, (KDID'05), pp. 165-187.

[45] Lesh, N., Zaki, M.J., and Ogihara, M.: *Mining Features for Sequence Classification*, in Proceedings International Conference Knowledge Discovery and Data Mining (KDD'99), 1999, pp. 342–346.

[46] 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.

[47] Michalski, R.S., and Chilausky, R.L.: *Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in The Context of Developing an Expert System for Soybean Disease Diagnosis*, in International Journal of Policy Analysis and Information Systems, 4(2), 1980, 125–160.

[48] Liu, B., Hsu, W., and Ma, Y.: *Integrating Classification and Association Rule Mining*, in Proceedings Conference on Knowledge Discovery in Data (SIGKDD'98), 1998, pp. 80–86.

[49] Li, W., Han, J., and Pei, J.: *CMAR: Accurate and Efficient Classification Based on Multiple Class-Association Rules*, in Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM 2001), 2001, pp. 369–376.

[50] Li, Y., Ning, P., Wang, X.S., and Jajodia, S.: *Discovering Calendar-based Temporal Association Rules*, in Proceedings of the 8th Int Symposium on Temporal Representation and Reasoning (TIME 2001), 2001, pp. 111–118.

[51] MacQueen, J.B.: *Some Methods for Classification and Analysis of Multivariate Observations*, in Proceedings 5th Berkeley Symposium on Mathematical Statistics and Probability, 1967, pp. 281–297.

[52] 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.

[53] Nakao, K., Mehta, K.R., Fridlyand, J., Moore, D.H., Jain, A.N., Lafuente, A., Wiencke, J.W., Terdiman, J.P., and Waldman, F.M.: *High-Resolution Analysis of DNA Copy Number Alterations in Colorectal Cancer by Array-Based Comparative Genomic Hybridization*, Carcinogenesis 25, 2004, pp. 1345–1357.

[54] Nanopoulos, A., Theodoridis, Y., and Manolopoulos, Y.: *C^2P: Clustering Based on Closest Pairs*, in 27th International Conference on Very Large Data Bases (VLDB'01), 2001, pp. 331–340.

[55] National Cancer Institute (NCI), DTP/2D and 3D structural information, `http://cactus.nci.nih.gov/ncidb2/download.html`.

[56] O'Callaghan, L., Mishra, N., Meyerson, A., and Guha, S: *Streaming-data Algorithm for High-Quality Clustering*, in 18th IEEE International Conference on Data Engineering (ICDE'02), 2002, pp. 685–697.

[57] Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., and Hsu, M.: *Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach*, in IEEE Trans. Knowl. Data Eng. 16(11), 2004, pp. 1424–1440.

[58] Pei, J., Zhang, X., Cho, M., Wang, H., and Yu, P.S.: *MaPle: A Fast Algorithm for Maximal Pattern-Based Clustering*, in 3th IEEE International Conference on Data Mining (ICDM'03), 2003, pp. 259–266.

[59] Project Proposal for MISTA: Mining in Semi-Structured Data, `http://www.liacs.nl/kosters/mista/mista.pdf`.

[60] Rhodes, N., Willet, P., Dunbar, J., and Humblet, C.: *Bit-String Methods for Selective Compound Acquisition*, in Journal of Chemical Information and Computer Sciences, vol. 40, 2000, pp. 210–214.

[61] Roberts, G., Myatt, G.J., Johnson, W.P., Cross, K.P., and Blower, P.E. Jr: *LeadScope: Software for Exploring Large Sets of Screening Data*, in Journal of Chemical Information and Computer Sciences, vol. 40, 2000, pp. 1302–1314.

[62] Samsonova, E.V., Bäck, T., Kok, J.N., and IJzerman, A.P.: *Reliable Hierarchical Clustering with the Self-Organizing Map*, in Proceedings 6th International Symposium on Intelligent Data Analysis (IDA'05), 2005, pp. 385–396.

[63] Steinbach, M., Tan, P., Xiong, H., and Kumar, V.: *Generalizing the Notion of Support*, in Proceedings 10th Int. Conf. on Knowledge Discovery and Data Mining (KDD'04), 2004, pp. 689–694.

[64] Tan, P., Steinbach, M. and Kumar, V.: *Introduction to Data Mining*, Pearson Addison-Wesley, 2006, ISBN: 0-321-32136-7.

[65] Tao, F., Murtagh, F., and Farid, M.: *Weighted Association Rule Mining using Weighted Support and Significance Framework*, in Proceedings 9th Int. Conf. on Knowledge Discovery and Data Mining (KDD'03), pp. 661–666.

[66] Teng, W., Chen, M., and Yu, P.S.: *A Regression-based Temporal Pattern Mining Scheme for Data Streams*, in Proceedings 29th Int. Conference on Very Large Data Bases (VLDB 2003), pp. 93–104.

[67] Uchiyama, I.: *Hierarchical Clustering Algorithm for Comprehensive Orthologous-Domain Classification in Multiple Genomes*, Nucleic Acids Research Vol. 34, No. 2, 2006, pp. 647–658.

[68] Wang, H., Wang, W., Yang, J., and Yu, P.S.: *Clustering by pattern similarity in large datasets*, in Proceedings SIGMOD International Conference (SIGMOD'02), 2002, pp. 394–405.

[69] Weka 3: Data Mining Software in Java, Website `http://www.cs.waikato.ac.nz/ml/weka/`.

[70] Wess, J.: G-Protein-Coupled Receptors: *Molecular Mechanisms Involved in Receptor Activation and Selectivity of G-Protein Recognition*, FASEB Journal 11 (5), 1997, pp. 346–354.

[71] Witten, I.H., Frank, E.: *Data Mining*, Morgan Kaufmann Publishers, 2000, ISBN: 1-55860-552-5.

[72] Willet, P., Barnad J.M., and Downs, G.M.J.: *Chemical Similarity Searching*, in Journal of Chemical Information and Computer Sciences, vol. 38, 1999, pp. 983–996.

[73] Xiong, H., Tan, P., and Kumar, V.: *Mining Strong Affinity Association Patterns in Data Sets with Skewed Support Distribution*, in Proceedings Int. Conf. on Data Mining (ICDM'03), 2003, pp. 387–394.

[74] 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.

[75] Yan, X. and Han, J.: *gSpan: Graph-Based Substructure Pattern Mining*, in Proceedings 2002 IEEE International Conference on Data Mining (ICDM 2002), pp. 721–724.

[76] Yan, X., Han, J. and Afshar R.: *CloSpan: Mining Closed Sequential Patterns in Large Datasets*, in Proceedings 2003 SIAM International Conference on Data Mining (SDM'03).

[77] Zaki, M., Parthasarathy, S., Ogihara, M., and Li, W.: *New Algorithms for Fast Discovery of Association Rules*, in Proceedings 3rd Int. Conf. on Knowledge Discovery and Data Mining (KDD '97), pp. 283–296.

[78] Zimmermann, A., and Raedt, L. De: *CorClass: Correlated Association Rule Mining for Classification*, in Proceedings International Conference on Discovery Science (DS-2004), 2004, pp. 60–72.

Nederlandse Samenvatting

Met de opkomst van het internet en de bio-informatica heeft de analyse van gegevens steeds meer te maken met een hoge maat van structurele vrijheid. Wanneer gegevens minder rigide gestructureerd zijn dan wordt het moeilijker interessante patronen te vinden door het grote aantal potentiële patronen (kandidaten). Er zijn veel mogelijke kandidaten omdat er vele manieren zijn om de onderdelen te combineren. Neem als een voorbeeld de bio-chemie, waar men moleculen analyseert in een poging interessante substructuren te vinden (bijvoorbeeld veel moleculen hebben drie koolstof atomen verbonden met een enkele verbinding). Echter één molecuul kan vele mogelijke atomen hebben en verschillende verbindingen tussen de atomen.

Semi-gestructureerde gegevens ontstaan als de bron of de omgeving geen rigide structuur voor de gegevens eist en wanneer gegevens gecombineerd worden uit verschillende heterogene bronnen. Bijvoorbeeld een bibliotheek-database waar sommige boeken geschreven zijn door één auteur en andere door verschillende. Voor sommige auteurs is alleen hun naam bekend en voor andere hun specialiteit en hun leeftijd. Ook worden vaak proceedings met andere velden omschreven dan een roman. Een ander voorbeeld van semi-gestructureerde gegevens is het bezoek aan een website en de analyse van patronen in de bezochte onderdelen. Het bezoek aan een website door een gebruiker wordt opgeslagen in de vorm van een graaf. Hier is elke selectie een verbinding die wijst naar een volgend onderdeel gerepresenteerd door een knoop in deze graaf. Sommige knopen bevatten wellicht extra informatie over het betreffende onderdeel van de website. De verbindingen tussen knopen geeft aan welke hyperlink geselecteerd is en naar welke pagina deze link leidde. Zodoende zit er in deze gegevenbron ook een structuur, maar het is niet helemaal rigide. Semi-gestructureerde gegevens hebben de volgende eigenschappen:

- Records hebben niet hetzelfde aantal velden en een veld kan ook qua type inhoud verschillen.

- Records of onderdelen daarvan die vergelijkbare principes omschrijven, bijvoorbeeld een molecuul of een atoom, zijn gegroepeerd.

- Velden hoeven niet een bepaalde volgorde te hebben.

In dit proefschrift worden verschillende manieren onderzocht om semi-gestructureerde gegevens te analyseren (data mining). Er is gekeken naar verschillende manieren om de voorkomens van een patroon te tellen om zo interessante patronen te vinden.

Het juist presenteren van de resultaten aan de gebruiker is ook van belang. Dit proefschrift behandelt de visuele weergave van resultaten van de analyse (mining) van semi-gestructureerde gegevens, zodat de gebruiker eenvoudiger interessante patronen kan vinden.

De noodzaak om semi-gestructureerde gegevens te analyseren komt voort uit het groeiende aantal bronnen van semi-gestructureerde gegevens.

Hoofdstuk 2 behandelt het zoeken van opeenvolgend voorkomende patronen. Het hoofdstuk begint met het defineren van "consecutive support" (maat opeenvolgendheid). Daarna gaat het verder met een behandeling van het snijden in de zoekruimte op zo een manier dat men nog steeds redelijk snel resultaten heeft bij een lagere opeenvolgendheid.

Het ontdekken van opeenvolgende patronen kan men combineren met technieken die eerder gebruikt zijn bij traditionele manieren van het tellen van voorkomens. We vervolgen Hoofdstuk 2 door aan te tonen dat deze technieken in combinatie met opeenvolgendheid nieuwe patronen kunnen opleveren.

In Hoofdstuk 3 worden de gegevens over het voorkomen van patronen gebruikt om patronen te ontdekken die vaak voorkomen met vrijwel gelijke tijd tussen voorkomens. De eerste benadering van Hoofdstuk 3 is om alle voorkomens te nemen en om dan te kijken of een andere voorkomen ongeveer halverwege aanwezig is. Als dit vaak voorkomt dan heeft deze reeks van voorkomens vaak gelijke tijd tussen voorkomens. De tweede benadering in Hoofdstuk 4 is om delen van de zoekruimte over te slaan als geen onderlinge afstand tussen alle paren vaak genoeg voorkomt.

In Hoofdstuk 5 wordt het sneller ontdekken van patronen besproken in het geval dat de gebruiker ongeveer weet in welke delen de meest opvallende patronen zitten. Deze domein specifieke kennis wordt gebruikt bij het zoeken naar zogenaamde 'most discriminating patterns', patronen die veel voorkomen in één verzameling van patronen en niet in een andere.

Het makkelijk kunnen overzien van resultaten is belangrijk en in Hoofdstuk 6, Hoofdstuk 7 en Hoofdstuk 8 wordt de visualisatie van patronen in

verschillende gegevensbronnen besproken. In Hoofdstuk 6 wordt er een visueel model gemaakt van het samen voorkomen van patronen. Zo kan de gebruiker snel zien welke patronen vaak samen voorkomen in dezelfde transactie en welke bijna alleen in verschillende. In Hoofdstuk 7 wordt een applicatie besproken waar men start met een graaf en deze vervolgens uitbreidt (of krimpt) naar een andere graaf wanneer deze ook veel voorkomend is. In Hoofdstuk 8 wordt een methode geintroduceerd om samen voorkomende patronen in "streams", potentieel oneindige stromen van transacties, te vinden. De potentiële oneindigheid maakt het eerst verkrijgen van veelvoorkomende patronen onmogelijk. De gekozen benadering bouwt een model van patronen in een "stream" door patronen te groeien en te krimpen afhankelijk van hun voorkomen en hun afstand (benadering van hoeveel transactie-paren samenvoorkomen).

Met Hoofdstuk 9 sluiten we het proefschrift af door alle technieken toe te passen in één setting: gegevens over webpagina gebruik. Op deze manier tonen we aan hoe alle technieken gecombineerd zouden kunnen worden in één tool voor het analyseren van het gebruik van websites.

Acknowledgements

First of all, I like to thank my parents, Wilfried and Marianne de Graaf for their endless belief in me. I also thank Florentia Mourouka, for her support and insight.

Curriculum Vitae

Edgar de Graaf was born in Schagen, North-Holland on the 29th of November of 1979. Living most of his life in Maarssen near Utrecht, he completed high-school in Maarssen. After high-school Edgar went to achieve a bachelor level education in the Hogeschool van Utrecht, with many interesting internships, e.g., at Philips and Ordina. He continued his education with a Computer Science master in Agents and Knowledge Technology at the University of Utrecht. For the last four years he has been a PhD student at Leiden University under the supervision of Dr. Walter Kosters and Prof. Dr. Joost Kok.

Titles in the IPA Dissertation Series since 2002

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects*. Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity*. Faculty of Mathematics and Computer Science, TU/e. 2002-05

M.I.A. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-06

N. van Vugt. *Models of Molecular Computing*. Faculty of Mathematics and Natural Sciences, UL. 2002-07

A. Fehnker. *Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-08

R. van Stee. *On-line Scheduling and Bin Packing*. Faculty of Mathematics and Natural Sciences, UL. 2002-09

D. Tauritz. *Adaptive Information Filtering: Concepts and Algorithms*. Faculty of Mathematics and Natural Sciences, UL. 2002-10

M.B. van der Zwaag. *Models and Logics for Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-11

J.I. den Hartog. *Probabilistic Extensions of Semantical Models*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2002-12

L. Moonen. *Exploring Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-13

J.I. van Hemert. *Applying Evolutionary Computation to Constraint Satisfaction and Data Mining*. Faculty of Mathematics and Natural Sciences, UL. 2002-14

S. Andova. *Probabilistic Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2002-15

Y.S. Usenko. *Linearization in μCRL*. Faculty of Mathematics and Computer Science, TU/e. 2002-16

J.J.D. Aerts. *Random Redundant Storage for Video on Demand*. Faculty of Mathematics and Computer Science, TU/e. 2003-01

M. de Jonge. *To Reuse or To Be Reused: Techniques for component composition and construction*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-02

J.M.W. Visser. *Generic Traversal over Typed Source Code Representations*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-03

S.M. Bohte. *Spiking Neural Networks*. Faculty of Mathematics and Natural Sciences, UL. 2003-04

T.A.C. Willemse. *Semantics and Verification in Process Algebras with Data and Timing*. Faculty of Mathematics and Computer Science, TU/e. 2003-05

S.V. Nedea. *Analysis and Simulations of Catalytic Reactions*. Faculty of Mathematics and Computer Science, TU/e. 2003-06

M.E.M. Lijding. *Real-time Scheduling of Tertiary Storage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-07

H.P. Benz. *Casual Multimedia Process Annotation – CoMPAs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-08

D. Distefano. *On Modelchecking the Dynamics of Object-based Software: a Foundational Approach*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-09

M.H. ter Beek. *Team Automata – A Formal Approach to the Modeling of Collaboration Between System Components*. Faculty of Mathematics and Natural Sciences, UL. 2003-10

D.J.P. Leijen. *The λ Abroad – A Functional Approach to Software Components*. Faculty of Mathematics and Computer Science, UU. 2003-11

W.P.A.J. Michiels. *Performance Ratios for the Differencing Method*. Faculty of Mathematics and Computer Science, TU/e. 2004-01

G.I. Jojgov. *Incomplete Proofs and Terms and Their Use in Interactive Theorem Proving*. Faculty of Mathematics and Computer Science, TU/e. 2004-02

P. Frisco. *Theory of Molecular Computing – Splicing and Membrane systems*. Faculty of Mathematics and Natural Sciences, UL. 2004-03

S. Maneth. *Models of Tree Translation*. Faculty of Mathematics and Natural Sciences, UL. 2004-04

Y. Qian. *Data Synchronization and Browsing for Home Environments*. Faculty of Mathematics and Computer Science and Faculty of Industrial Design, TU/e. 2004-05

F. Bartels. *On Generalised Coinduction and Probabilistic Specification Formats*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-06

L. Cruz-Filipe. *Constructive Real Analysis: a Type-Theoretical Formalization and Applications*. Faculty of Science, Mathematics and Computer Science, KUN. 2004-07

E.H. Gerding. *Autonomous Agents in Bargaining Games: An Evolutionary Investigation of Fundamentals, Strategies, and Business Applications*. Faculty of Technology Management, TU/e. 2004-08

N. Goga. *Control and Selection Techniques for the Automated Testing of Reactive Systems*. Faculty of Mathematics and Computer Science, TU/e. 2004-09

M. Niqui. *Formalising Exact Arithmetic: Representations, Algorithms and Proofs*. Faculty of Science, Mathematics and Computer Science, RU. 2004-10

A. Löh. *Exploring Generic Haskell*. Faculty of Mathematics and Computer Science, UU. 2004-11

I.C.M. Flinsenberg. *Route Planning Algorithms for Car Navigation*. Faculty of Mathematics and Computer Science, TU/e. 2004-12

R.J. Bril. *Real-time Scheduling for Media Processing Using Conditionally Guaranteed Budgets*. Faculty of Mathematics and Computer Science, TU/e. 2004-13

J. Pang. *Formal Verification of Distributed Systems*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-14

F. Alkemade. *Evolutionary Agent-Based Economics*. Faculty of Technology Management, TU/e. 2004-15

E.O. Dijk. *Indoor Ultrasonic Position Estimation Using a Single Base Station*. Faculty of Mathematics and Computer Science, TU/e. 2004-16

S.M. Orzan. *On Distributed Verification and Verified Distribution*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-17

M.M. Schrage. *Proxima - A Presentation-oriented Editor for Structured Documents*. Faculty of Mathematics and Computer Science, UU. 2004-18

E. Eskenazi and A. Fyukov. *Quantitative Prediction of Quality Attributes for Component-Based Software Architectures*. Faculty of Mathematics and Computer Science, TU/e. 2004-19

P.J.L. Cuijpers. *Hybrid Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2004-20

N.J.M. van den Nieuwelaar. *Supervisory Machine Control by Predictive-Reactive Scheduling*. Faculty of Mechanical Engineering, TU/e. 2004-21

E. Ábrahám. *An Assertional Proof System for Multithreaded Java -Theory and Tool Support-* . Faculty of Mathematics and Natural Sciences, UL. 2005-01

R. Ruimerman. *Modeling and Remodeling in Bone Tissue*. Faculty of Biomedical Engineering, TU/e. 2005-02

C.N. Chong. *Experiments in Rights Control - Expression and Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-03

H. Gao. *Design and Verification of Lock-free Parallel Algorithms*. Faculty of Mathematics and Computing Sciences, RUG. 2005-04

H.M.A. van Beek. *Specification and Analysis of Internet Applications*. Faculty of Mathematics and Computer Science, TU/e. 2005-05

M.T. Ionita. *Scenario-Based System Architecting - A Systematic Approach to Developing Future-Proof System Architectures*. Faculty of Mathematics and Computing Sciences, TU/e. 2005-06

G. Lenzini. *Integration of Analysis Techniques in Security and Fault-Tolerance*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-07

I. Kurtev. *Adaptability of Model Transformations*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-08

T. Wolle. *Computational Aspects of Treewidth - Lower Bounds and Network Reliability*. Faculty of Science, UU. 2005-09

O. Tveretina. *Decision Procedures for Equality Logic with Uninterpreted Functions*. Faculty of Mathematics and Computer Science, TU/e. 2005-10

A.M.L. Liekens. *Evolution of Finite Populations in Dynamic Environments*. Faculty of Biomedical Engineering, TU/e. 2005-11

J. Eggermont. *Data Mining using Genetic Programming: Classification and Symbolic Regression*. Faculty of Mathematics and Natural Sciences, UL. 2005-12

B.J. Heeren. *Top Quality Type Error Messages*. Faculty of Science, UU. 2005-13

G.F. Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations*. Faculty of Science, Mathematics and Computer Science, RU. 2005-14

M.R. Mousavi. *Structuring Structural Operational Semantics*. Fac-

ulty of Mathematics and Computer Science, TU/e. 2005-15

A. Sokolova. *Coalgebraic Analysis of Probabilistic Systems*. Faculty of Mathematics and Computer Science, TU/e. 2005-16

T. Gelsema. *Effective Models for the Structure of pi-Calculus Processes with Replication*. Faculty of Mathematics and Natural Sciences, UL. 2005-17

P. Zoeteweij. *Composing Constraint Solvers*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-18

J.J. Vinju. *Analysis and Transformation of Source Code by Parsing and Rewriting*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-19

M.Valero Espada. *Modal Abstraction and Replication of Processes with Data*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2005-20

A. Dijkstra. *Stepping through Haskell*. Faculty of Science, UU. 2005-21

Y.W. Law. *Key management and link-layer security of wireless sensor networks: energy-efficient attack and defense*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-22

E. Dolstra. *The Purely Functional Software Deployment Model*. Faculty of Science, UU. 2006-01

R.J. Corin. *Analysis Models for Security Protocols*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-02

P.R.A. Verbaan. *The Computational Complexity of Evolving Systems*. Faculty of Science, UU. 2006-03

K.L. Man and R.R.H. Schiffelers. *Formal Specification and Analysis of Hybrid Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2006-04

M. Kyas. *Verifying OCL Specifications of UML Models: Tool Support and Compositionality*. Faculty of Mathematics and Natural Sciences, UL. 2006-05

M. Hendriks. *Model Checking Timed Automata - Techniques and Applications*. Faculty of Science, Mathematics and Computer Science, RU. 2006-06

J. Ketema. *Böhm-Like Trees for Rewriting*. Faculty of Sciences, VUA. 2006-07

C.-B. Breunesse. *On JML: topics in tool-assisted verification of JML programs*. Faculty of Science, Mathematics and Computer Science, RU. 2006-08

B. Markvoort. *Towards Hybrid Molecular Simulations*. Faculty of Biomedical Engineering, TU/e. 2006-09

S.G.R. Nijssen. *Mining Structured Data*. Faculty of Mathematics and Natural Sciences, UL. 2006-10

G. Russello. *Separation and Adaptation of Concerns in a Shared Data Space*. Faculty of Mathematics and Computer Science, TU/e. 2006-11

L. Cheung. *Reconciling Nondeterministic and Probabilistic Choices*. Faculty of Science, Mathematics and Computer Science, RU. 2006-12

B. Badban. *Verification techniques for Extensions of Equality Logic*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2006-13

A.J. Mooij. *Constructive formal methods and protocol standardization*. Faculty of Mathematics and Computer Science, TU/e. 2006-14

T. Krilavicius. *Hybrid Techniques for Hybrid Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-15

M.E. Warnier. *Language Based Security for Java and JML*. Faculty of Science, Mathematics and Computer Science, RU. 2006-16

V. Sundramoorthy. *At Home In Service Discovery*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-17

B. Gebremichael. *Expressivity of Timed Automata Models*. Faculty of Science, Mathematics and Computer Science, RU. 2006-18

L.C.M. van Gool. *Formalising Interface Specifications*. Faculty of Mathematics and Computer Science, TU/e. 2006-19

C.J.F. Cremers. *Scyther - Semantics and Verification of Security Protocols*. Faculty of Mathematics and Computer Science, TU/e. 2006-20

J.V. Guillen Scholten. *Mobile Channels for Exogenous Coordination of Distributed Systems: Semantics, Implementation and Composition*. Faculty of Mathematics and Natural Sciences, UL. 2006-21

H.A. de Jong. *Flexible Heterogeneous Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-01

N.K. Kavaldjiev. *A run-time reconfigurable Network-on-Chip for streaming DSP applications*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-02

M. van Veelen. *Considerations on Modeling for Early Detection of Abnormalities in Locally Autonomous Distributed Systems*. Faculty of Mathematics and Computing Sciences, RUG. 2007-03

T.D. Vu. *Semantics and Applications of Process and Program Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-04

L. Brandán Briones. *Theories for Model-based Testing: Real-time and Coverage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-05

I. Loeb. *Natural Deduction: Sharing by Presentation*. Faculty of Science, Mathematics and Computer Science, RU. 2007-06

M.W.A. Streppel. *Multifunctional Geometric Data Structures*. Faculty of Mathematics and Computer Science, TU/e. 2007-07

N. Trčka. *Silent Steps in Transition Systems and Markov Chains*. Faculty of Mathematics and Computer Science, TU/e. 2007-08

R. Brinkman. *Searching in encrypted data*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-09

A. van Weelden. *Putting types to good use*. Faculty of Science, Mathematics and Computer Science, RU. 2007-10

J.A.R. Noppen. *Imperfect Information in Software Development Processes*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-11

R. Boumen. *Integration and Test plans for Complex Manufacturing Systems*. Faculty of Mechanical Engineering, TU/e. 2007-12

A.J. Wijs. *What to do Next?: Analysing and Optimising System Behaviour in Time*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2007-13

C.F.J. Lange. *Assessing and Improving the Quality of Modeling: A Series of Empirical Studies about the UML*. Faculty of Mathematics and Computer Science, TU/e. 2007-14

T. van der Storm. *Component-based Configuration, Integration and Delivery*. Faculty of Natural Sciences, Mathematics, and Computer Science,UvA. 2007-15

B.S. Graaf. *Model-Driven Evolution of Software Architectures*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2007-16

A.H.J. Mathijssen. *Logical Calculi for Reasoning with Binding*. Faculty of Mathematics and Computer Science, TU/e. 2007-17

D. Jarnikov. *QoS framework for Video Streaming in Home Networks*. Faculty of Mathematics and Computer Science, TU/e. 2007-18

M. A. Abam. *New Data Structures and Algorithms for Mobile Data*. Faculty of Mathematics and Computer Science, TU/e. 2007-19

W. Pieters. *La Volonté Machinale: Understanding the Electronic Voting Controversy*. Faculty of Science, Mathematics and Computer Science, RU. 2008-01

A.L. de Groot. *Practical Automaton Proofs in PVS*. Faculty of Science, Mathematics and Computer Science, RU. 2008-02

M. Bruntink. *Renovation of Idiomatic Crosscutting Concerns in Embedded Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-03

A.M. Marin. *An Integrated System to Manage Crosscutting Concerns in Source Code*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-04

N.C.W.M. Braspenning. *Model-based Integration and Testing of High-tech Multi-disciplinary Systems*. Faculty of Mechanical Engineering, TU/e. 2008-05

M. Bravenboer. *Exercises in Free Syntax: Syntax Definition, Parsing, and Assimilation of Language Conglomerates*. Faculty of Science, UU. 2008-06

M. Torabi Dashti. *Keeping Fairness Alive: Design and Formal Verification of Optimistic Fair Exchange Protocols*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2008-07

I.S.M. de Jong. *Integration and Test Strategies for Complex Manufacturing Machines*. Faculty of Mechanical Engineering, TU/e. 2008-08

I. Hasuo. *Tracing Anonymity with Coalgebras*. Faculty of Science, Mathematics and Computer Science, RU. 2008-09

L.G.W.A. Cleophas. *Tree Algorithms: Two Taxonomies and a Toolkit*. Faculty of Mathematics and Computer Science, TU/e. 2008-10

I.S. Zapreev. *Model Checking Markov Chains: Techniques and Tools*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-11

M. Farshi. *A Theoretical and Experimental Study of Geometric Networks*. Faculty of Mathematics and Computer Science, TU/e. 2008-12

G. Gulesir. *Evolvable Behavior Specifications Using Context-Sensitive Wildcards*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-13

F.D. Garcia. *Formal and Computational Cryptography: Protocols, Hashes and Commitments*. Faculty of Science, Mathematics and Computer Science, RU. 2008-14

P. E. A. Dürr. *Resource-based Verification for Robust Composition of Aspects*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-15

E.M. Bortnik. *Formal Methods in Support of SMC Design*. Faculty of Mechanical Engineering, TU/e. 2008-16

R.H. Mak. *Design and Performance Analysis of Data-Independent Stream Processing Systems*. Faculty of Mathematics and Computer Science, TU/e. 2008-17

M. van der Horst. *Scalable Block Processing Algorithms*. Faculty of Mathematics and Computer Science, TU/e. 2008-18

C.M. Gray. *Algorithms for Fat Objects: Decompositions and Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-19

J.R. Calam. *Testing Reactive Systems with Data - Enumerative Methods and Constraint Solving*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-20

E. Mumford. *Drawing Graphs for Cartographic Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-21

E.H. de Graaf. *Mining Semi-structured Data, Theoretical and Experimental Aspects of Pattern Evaluation*. Faculty of Mathematics and Natural Sciences, UL. 2008-22