UNIVERSIDADE TÉCNICA DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

Pattern Mining over Nominal Event Sequences using Constraint Relaxations

Cláudia Martins Antunes
(Mestre)

Dissertação para a obtenção do Grau de Doutor
em Engenharia Informática e de Computadores

Orientador: Doutor Arlindo Manuel Limede de Oliveira

Júri

Presidente: Reitor da Universidade Técnica de Lisboa
Vogais: Doutor Pieter Willem Adriaans
Doutor José Manuel Nunes Salvador Tribolet
Doutor João Emílio Segurado Pavão Martins
Doutor Mário Jorge Gaspar da Silva
Doutor Arlindo Manuel Limede de Oliveira
Doutor Alípio Mário Guedes Jorge
Doutor Fernando Henrique Corte-Real Mira da Silva

Janeiro 2005

Resumo

Um dos problemas da extracção de conhecimento ainda sem solução é a descoberta de padrões em dados temporais. Uma das abordagens mais usadas para explorar este tipo de dados é a descoberta de padrões sequenciais, uma vez que os dados temporais podem ser encarados como sequências de eventos. No entanto, os algoritmos existentes apresentam algumas deficiências, nomeadamente a falta de foco nas expectativas do utilizador e o elevado número de padrões que descobrem. Para além disso, a utilização de restrições, geralmente aceite como solução para o problema, pode transformar o processo num simples teste de hipóteses, especialmente quando se usam restrições mais fortes, como é o caso das linguagens regulares.

Nesta dissertação, argumenta-se que é possível usar algoritmos de descoberta de padrões sequenciais com restrições, para descobrir informação desconhecida, mantendo o processo de descoberta focado nas expectativas e conhecimento do utilizador. De forma a demonstrar a validade do argumento, é proposta uma nova metodologia para explorar sequências de eventos baseada na utilização de relaxamentos de restrições. Estes relaxamentos estabelecem condições mais fracas que as restrições impostas, tornando possível a descoberta de padrões que não são completamente aceites. Adicionalmente é proposta uma hierarquia de relaxamentos, indo desde relaxamentos conservativos até a relaxamentos aproximados e não aceites.

De modo a facilitar a aplicação da nova metodologia, são propostas várias extensões aos algoritmos de descoberta de padrões sequenciais existentes, de modo a lidar de forma eficiente com as restrições-Ω, também propostas.

Abstract

One of the main unresolved problems that arises in the data mining process is treating data that contains temporal information. A complete understanding of this phenomenon requires that the data should be viewed as a sequence of events. Sequential pattern mining is the approach most commonly used to explore nominal sequences, enabling the discovery of frequent sequential patterns. The main drawback of algorithms for this task has been their lack of focus on user expectations and the high number of discovered patterns. However, in some cases, the solution most commonly adopted, based on the use of constraints, can transform the mining process into a hypothesis-testing task. This risk is even stronger when mining sequential data, where more restrictive constraints, like regular languages, have been used.

In this dissertation, we argue that it is possible to use constrained sequential pattern mining algorithms, over nominal data, to discover unknown information, keeping the process centered on the user. In order to demonstrate the validity of this thesis, we propose a new methodology based on the use of *Ω-constraint relaxations*. A *constraint relaxation* establishes a weaker condition than the original constraint, making possible the discovery of patterns that are not completely accepted by the original constraint. We propose a new hierarchy of relaxations that ranges from conservative ones, to approximately accepted and non-accepted relaxations.

Additionally, we propose several extensions to existing sequential pattern mining algorithms, in order to deal efficiently with constraints and, in particular with gap and *Ω-constraints.*

Palavras Chave

Descoberta de padrões temporais
Descoberta de padrões sequenciais
Restrições temporais
Restrições de conteúdo
Relaxamentos de restrições

Key Words

Temporal Data Mining
Sequential Pattern Mining
Content Constraint
Temporal Constraints
Constraint Relaxation

Acknowledgements

I would like to thank those who have been present in these years, and have in some way contributed to this thesis.

To my advisor, Professor Arlindo Oliveira, for the continuous discussions along these years, and the numberless readings and comments of this dissertation.

To Professor Pavão Martins, for his support and confidence since my early days as a teaching assistant at Instituto Superior Técnico.

To PmeLink.PT staff, for the possibility to apply new mining methodologies over real-life data and for their availabity to answer all my questions.

To Professor Pedro Lourtie and Professor João Ventura, for their comments in the exploration of Mathematical Analysis data.

To Gregory Piatetsky-Shapiro, for his readiness on allowing the reproduction of an issue of the "KDnuggets news".

To my students, for their presence, cheerfulness and joviality. To Sofia Pinto, for her friendship and patience with me, for our long discussions about research and career expectations.

To my parents and sister, my special thank for their continuous support and encouragement through all these years.

Finally, to Pedro, for his presence, patience, encouragement and unlimited support.

Lisbon, May 2004

Cláudia Martins Antunes

Contents

Figures Index

Table Index

Algorithms Index

Chapter 1
Introduction

"As time passes, objects are created and destroyed, and people gain, and forget, information." [Hayes 1995]

Time has been one of the most interesting phenomenon for men. Understanding its relation to the observed changes has been a target, for thousands of years, from complex philosophical questions to the study of human behavior. Temporal data appears naturally in almost everything in nature, from engineering to medicine and finance domains, and with the improvements on the digital storage capabilities in the last decades, the interest on the discovery of hidden information, by automatic means, has exploded.

In general, we can consider two kinds of temporal data: *real-valued* and *nominal data*. The automatic analysis of real-valued data has deserved a considerable attention in the last decades, especially from the seventies until now. Examples of these are the analysis and prediction of continuous signals, like financial time series. This interest is probably due to the financial impact of such prediction and to the challenge of being able to predict a theoretically unpredictable behavior.

However, for nominal data, traditional automatic data exploration/mining techniques usually treat temporal data as unordered collections of events, ignoring its temporal information. They usually center their attention on the analysis of the data that occurs at the same instant (*intra-transactional analysis*), instead of analyzing the relations of data between different instants (*inter-transactional analysis*). Examples of such techniques are mechanisms for data classification, data clustering and mining frequent patterns.

As an answer to this problem, *Temporal Data Mining* has appeared in the last decade as an

extension to traditional data mining techniques. It provides the ability to explore the dynamic aspects of entities, instead of only exploring its static characteristics, and simultaneously performing *inter-transactional analysis*. In this manner, the classification and clustering of temporal data, as well as the discovery of temporal patterns, is currently being addressed.

1 – Problems in Pattern Mining over Temporal Data

The most usual technique to discover temporal patterns in nominal data is known as *Sequential Pattern Mining*. It was first introduced in 1995 [Srikant 1995] as an extension of the frequent pattern mining task, and, since then, several algorithms have been proposed. Nevertheless the advances on the performance of those algorithms, the large number of discovered patterns (usually thousands or hundreds of thousand), has been the main drawback of these approaches. Note that the combinatorial nature of sequential patterns is even stronger than for patterns in general, and the simple optimization of the corresponding algorithms is not sufficient to reduce the number of patterns, considerably.

Another issue on data mining techniques, in general, and in pattern mining in particular, is the lack of focus on user expectations and background knowledge. In fact, most of the time, the results are not sufficiently interesting and/or are too difficult to analyze. In order to minimize this problem, recent approaches use *constraints* to restrict the number and scope of the discovered patterns. Using constraints makes it possible to focus the mining process into areas or sub-spaces where useful information is likely to be gained. In the last years, several categories of constraints have been used, most of them related to the duration of sequences and the impact of the proximity among events (*gap constraint*), and more recently with the content of transactions (for example *item constraints* and regular languages). Despite the efforts to capture the semantics of the application domains, they have been applied seldom, in a non-integrated way. Indeed, it was only in the last few years, that the data mining community has understood the need for a theoretical framework for data mining that is able to deal with the presence of background knowledge, among other things [Mannila 2000b].

However, the existence of such ability to represent background knowledge and efficient algorithms to discover the hidden patterns does not solve the entire problem. As pointed by some authors [Hipp 2002], restricting the search too much approximates the mining process to a simple hypothesis-testing task, since the process will only find already known patterns. In fact, the use of

regular languages as constraints goes against the first goal of data mining – the discovery of unknown information. Nevertheless, it is undeniable the adequacy of these or other formal languages to represent the knowledge about sequential phenomena.

In this dissertation, we argue that it is possible to use constrained sequential pattern mining algorithms, over nominal data, to discover unknown information, keeping the process centered on the user.

2 – Main Contributions

In order to demonstrate the validity of this thesis, we propose a new methodology to mine nominal event sequences, based on the use of *constraint relaxations* to guide the sequential pattern mining process. The methodology makes use of sequential pattern mining algorithms and two concepts: the Ω-*constraint* and *constraint relaxations*.

The Ω-*constraint* aggregates a *content*, a *temporal* and an *existential* constraints in a unique object. The *content constraint* is composed of an *item constraint*, specifying the set of items relevant for the analysis, a *taxonomy* for allowing generalized patterns, and a formal language to specify the characteristics that the sequences of itemsets may have, in order to be interesting for the analysis. The *temporal constraint* is defined based on the concepts of time interval and other related concepts, defined in the "Reusable Time" ontology [Zhou 2002]. It specifies a restriction over the timestamps of transactions. It uses a *time interval* that includes the period of interest, a maximal allowed *time gap* to consider, and the *time granularity* of the data. Finally, the *existential constraint* is just the minimum support threshold, as is usual in sequential pattern mining algorithms.

While this constraint serves as a way to capture the user background knowledge, the use of a constraint relaxation enables the discovery of unknown information. A *constraint relaxation* establishes a weaker condition than the original constraint, and in this manner, makes possible the discovery of patterns that are not completely accepted by the original constraint. Relaxations of Ω-constraints include a relaxation both over the content and the temporal constraint. While the first one is based on the relaxation of the associated formal language, establishing weaker conditions to accept a sequence as valid, the last one is specified as the *maximal time error* accepted

In particular, we make the following contributions:

- We propose the use of *constraint relaxations* to guide the pattern mining process, enabling the discovery of unknown information.
- We propose a new type of constraints – the Ω-*constraints*, which provide a mean to represent background knowledge about nominal temporal data. In this manner, we contribute to the specification of a theoretical framework to deal with sequential and temporal data.
- We propose a hierarchy of constraint relaxations, from *conservative* ones, that accept patterns that are subsequences of accepted sequences, to non-conservative ones: the *Approx accepted*, which accepts sequences that dist a pre-defined error to some accepted sequence, and the *non-accepted* relaxation, which enables the discovery of completely unknown and unexpected patterns.
- We present the concretization of those relaxations when applied to Ω–constraints, in the particular case when the content constraint is based on a context-free language (represented as a pushdown automaton). In parallel, we extend pushdown automata to deal with sequences of itemsets.
- We present an extension to the *PrefixGrowth* algorithm – *GenPrefixGrowth* to efficiently deal with Ω–constraints and constraint relaxations.
- We propose an efficient algorithm – ε-*acceptsCFL*, for verifying if a sequence is approximately accepted by a specific pushdown automata.
- We analyze and explain the sequential pattern mining problem and its two main approaches: apriori-based and pattern-growth methods, stating clearly the advantages and disadvantages of each approach. We propose two new algorithms: one resulting from the generalization of pattern-growth methods to use gap constraints – *GenPrefixSpan*, and the other – *SPaRSe*, which optimizes the apriori philosophy, by constraining the search space. In this manner, we demonstrate that apriori-based algorithms are able to compete with pattern-growth methods, and outperform them in dense datasets.
- Finally, we show the results of the new methodology when applied to real-world problems.

3 – Dissertation Outline

The dissertation comprises eight chapters. The first chapter (Chapter 2) introduces the concepts and the process of knowledge discovery, paying a particular attention to the problem of temporal data

mining and its main areas.

In Chapter 3 we present a detailed description of the sequential pattern mining problem, presenting an analysis of its complexity. Additionally, we survey the proposed constraints applied to sequential pattern mining, formally introducing the concepts of deterministic finite (DFA) and pushdown automata (PDA), used in the rest of the dissertation. After the description of the two main approaches and the corresponding adaptations to deal with regular languages as constraints, we present our thesis statement, explaining the meaning of each claim in detail.

We begin Chapter 4 by presenting the generalization of *PrefixSpan* – *GenPrefixSpan*, generalizing the first algorithm in order to be able to deal with gap constraints. We finish the first section with a comparison between *GSP* and *GenPrefixSpan*, identifying the continuous reduction of the search space, used by pattern-growth methods, as the first cause for their differences on performance. In the second section, we present a new apriori-based algorithm – *SPaRSe*, which combines the simplicity of the candidate generation and test philosophy with the continuous reduction of the search space. With this new algorithm, we show that, with some specific improvements, apriori-based algorithms can compete with pattern-growth methods, showing similar performances on the generality of situations. Studies on performance and scalability in synthetic datasets are presented in the last section of this chapter.

In Chapter 5, we present the Ω–constraint, describing its aggregated constraints in detail. A particular attention is given to the definition of the temporal constraint and the concept of time interval, as defined in the "Reusable Time" ontology. Additionally, in terms of the content constraint, we present the extension of pushdown automata to deal with sequences of itemsets. We finish the chapter with the adaptation of existent algorithms to use Ω–constraints, presenting the *GenPrefixGrowth* algorithm. We demonstrate that this algorithm in conjunction with Ω–constraints outperforms *GenPrefixSpan* in the generality of situations.

The description of the new methodology for mining sequential / temporal patterns is made in Chapter 6, were a hierarchy of constraint relaxations is proposed. In particular, we propose a new algorithm – *ε-acceptsCFL*, to verify if a sequence is approximate accepted by a given context-free language, represented as a pushdown automaton. The chapter ends with a performance study of the usage of each constraint relaxation.

Finally, Chapter 7 presents the results obtained by the application of the new methodology on three real-life datasets. The thesis concludes in Chapter 8, where a summary of the dissertation and achieved results is presented. Moreover, some guidelines for future research are suggested.

Chapter 2
Temporal Data Mining: an introduction

In this chapter, the field of data mining is introduced, by presenting its core concepts and an overview of the major steps of the mining process. Following this introduction, temporal data mining is addressed, by exploring its goals, applications, techniques and related problems. A particular attention is given to pre-processing, mining and post-processing operations over temporal data.

The analysis of registered data was traditionally made by statisticians, who tried to give a precise portrait of populations. Data mining, as the natural successor of statistics, tries to go further, providing automatic means to classify and predict future behaviors.

The term *data mining* has been overused, most of the times to mean the whole process of *knowledge discovery in databases* (usually called KDD). In this context, it has been defined as "the nontrivial extraction of implicit, previously unknown and potential useful information from data" [Frawley 1992]. In the last decade, data mining techniques have gained acceptance as a viable means for finding full information in data, and its use has expanded to commercial domains, like customer relationship management, market basket analysis or credit card fraud detection, to scientific and engineering applications.

The analysis of temporal data has been used in a variety of domains, from engineering to scientific research, finance and medicine. In engineering matters, temporal data usually arises with either sensor-based monitoring, such as telecommunications control (e.g. [Das 1997], [Das 1998], [Grossman 1998]) or log-based systems monitoring (see, for instance [Mannila 1996] and [Mannila 2000a]). In scientific research it appears, for example, in spatial missions (e.g. [Keogh 1997] and [Oates 1999]).

In finance, the prediction of time series is the paradigmatic example of temporal data mining. See, for instance, the work of [Fama 1970], [Faloutsos 1994], [Refenes 1995], [Berndt 1996], [Chan 1999], [Gavrilov 2000] or [Ge 2000]. Nevertheless, over the last years the analysis of product sales or inventory consumption has also been of great importance to business planning (see for instance [Agrawal 1995c], [Srikant 1996] and [Zaki 1998a]).

In healthcare, temporal sequences are a reality for decades, with data originated by complex data acquisition systems like ECG's (see, for instance [Grossman 1998], [Ge 2000], [Dajani 2001]), or even with simple ones like the ones that measure the patient temperature or the effectiveness of treatments [Shahar 1996], [Caraça-Valente 2000]. In the last years, with the development of medical information systems, the amount of data has increased considerably, and more than ever, the need to react in real-time to any change in the patient behavior is crucial [Coiera 1994], [Antunes 2001a].

1 – Data Mining: basic concepts

In order to understand the data mining field, some basic concepts are needed, and in particular it is important to make the distinction between *data* and *information*. The definitions adopted here are the most commonly accepted by the data mining community [Kohavi 1998].

The term *data* is used to mean the recorded facts that describe the state or behavior of some entity, in accordance with a set of *attributes*, also known as *fields* or *variables*, each of which corresponds to a particular value. These values belong to specific sets – the *attribute domains*, which represent the values that can be taken by the attribute. In general attribute domains can belong to one of two types:

- *Real-valued* or *continuous*, subsets of real numbers, where there is a measurable quantity in a given range.
- *Categorical*, finite sets of discrete values. There are two types of categorical attributes:
    - *Nominal*, denote that there is no ordering between values, such as names and colors. These sets are usually called *alphabets*.
    - *Ordinal*, denote that there is an ordering among the values, such as in an attribute taking on the values low, medium, or high.

Another term commonly used is *feature*, which corresponds to a particular instantiation of an attribute. However, the term feature is usually misused as attribute, like in "feature selection".

When a set of attributes describes some property, we are in the presence of a *dimension*. Examples of common dimensions are space and time.

The terms *instance* and *record* denote a single object of the world. In general, they are represented by *feature vectors*, which correspond to the set of features that describes them at particular instants of time.

Unfortunately, the definition of *information* is not so consensual. In general, it has been used to denote the set of patterns or expectations that underlie the data [Witten 2000], and from a mathematical point of view, the generation of information can be seen as data compression [Adriaans 1996], since information can be seen as a model to represent several instances. Another way to view information is as the next step in the road of knowledge, since it corresponds to an abstraction of the already known (recorded) data.

When dealing with temporal data, additional terms are needed.

The term *transaction* was inherited from the database community, and refers to the facts recorded in a single transaction (in the sense of the operations performed on databases). In this context, an *item* is one of the transacted values, and an *itemset*, a set of items transacted in the same instant and by the same entity, i.e. the items belonging to the same transaction.

When dealing with transactions, two kinds of analysis can be performed:

- *Intra-transactional*, where the analysis is performed among the data transacted at the same time.
- *Inter-transactional*, where the analysis is performed among the data transacted at different time instants.

Note that intra-transactional analysis is particularly interesting for describing the state of some instance and to inter-relate its static characteristics. The analysis of its behavior cannot be performed with an intra-transactional analysis, but an inter-transactional one is able to describe it.

An *event* has been defined as a pair (e, t), where e is the *event type* and t a positive integer, called the *timestamp* [Bettini 1998]. In this context, the event type specifies the item transacted at the corresponding time instant. However, this notion can be extended, as proposed below.

First, timestamps may be replaced by a set of time attributes, completely specifying the time dimension. Second, in order to deal efficiently with simultaneous transactions performed by a unique instance, the event type can be replaced by the set of transacted items – the *itemset*. In this manner, the representation of simultaneous or parallel events is compacted in a natural way, and

the temporal data corresponding to the behavior of some instance may be represented by a sequence of events, called an *event sequence* in the rest of this text. If registered items are real-valued, the event sequence is usually called a *time series*, and, if they belong to a nominal domain, a *nominal event sequence*.

Related to the concept of event sequence appears the notion of subsequence. A *subsequence* of an event sequence *s* is an event sequence that contains part of the events of *s*, preserving the original order among events.

In the rest of this dissertation, we will adopt the extended version of the event notion.

2 – Data Mining: process overview

With the information discovery as a target, the knowledge discovery task receives data as input and returns information as the result. But several questions arise:

- How to deal with data from several distinct sources?
- How to represent different data types?
- How to treat inconsistent data?
- How to deal with noise, outliers and missing values?
- How to learn the information, that is, how to extract the patterns that underlie the data?
- How to deal with large amounts of data?
- How to evaluate the relevancy of discovered information?
- How to deal with user expectations?
- How to use background knowledge?
- How to avoid the discovery of already known information?

In order to deal with all of these questions, knowledge discovery has been seen as a process composed by a iterative sequence of steps, as illustrated in Figure 2.1.

These steps can be classified into three main categories: pre-processing, data mining and post-processing tasks.

In general, we can view the pre-processing stage as the preparation of data before the application of data mining tools, and the post-processing as the evaluation and presentation of the discovered information to the final user.

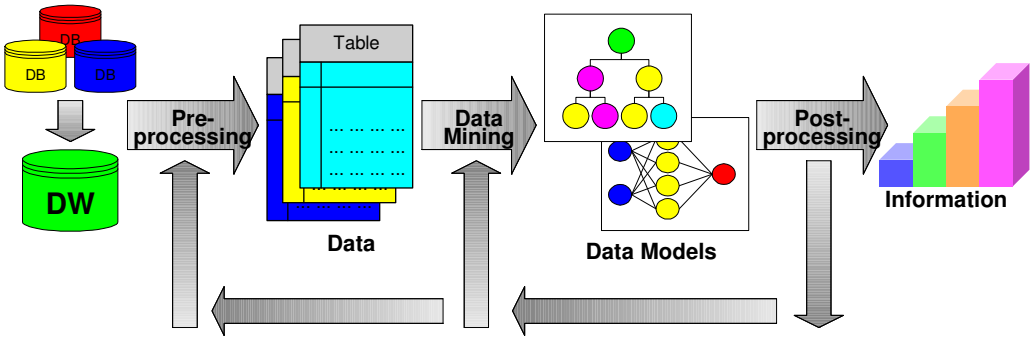


Figure 2.1 – The knowledge discovery process

Nowadays, it is usual to perform the mining process directly over the data in some data warehouse, avoiding part of the pre-processing stage. A *data warehouse* is a "subject-oriented, integrated, time-variant, and non-volatile collection of data in support of management's decision making process" [Inmon 1996]. However, and since data warehouses are mostly constructed from the integration of multiple heterogeneous data sources, their construction involves some or all the efforts required to prepare the data for the mining process – the pre-processing operations, described next.

2.1 – Pre-processing

The pre-processing stage consists of a set of operations performed over data in order to improve its quality, and, consequently, the mining results.

Recent statistics have shown that the pre-processing stage occupies about 75% of the overall time of the data mining process (In KDnuggets Past Polls: "Data preparation part in data mining projects", Sep 30 – Oct 12, 2003, reproduced in Appendix A). The time spent in this stage reveals the poor quality of the majority of the existent data, and the importance of these operations when dealing with large datasets.

The generality of pre-processing operations can be categorized as belonging to three main types of techniques: *data integration*, *data cleaning* and *data reduction* [Han 2001a].

Data Integration

Data integration operations are used to merge the data from multiple, possibly heterogeneous, data sources. The key difficulties are related to the different schema behind the different data sources and to the existence of duplicated records.

The first issue is one of the major problems in data integration, since it requires the

identification of equivalencies between the entities existent in different data sources (usually known as the *entity identification problem*).

The second challenge in data integration is the identification and removal of redundancies. This is not a trivial issue, but can be approached using correlation analysis between attributes. If the correlation between two attributes is zero, then the attributes are independent and there is no redundancy between the attributes. Otherwise, one attribute is correlated with the other, which means that we are in the presence of some redundancy. Another problem is the removal of duplicated records, which consists in identifying the existence of multiple records for the same data entry.

Data Cleaning

Once the integration of the distinct data sources is achieved, data cleaning operations aim to ensure the data quality. In general, they deal with three distinct situations: *missing values*, *outliers* or *noise* and *data inconsistencies*.

Missing Values. A frequent problem is the existence of several instances with no recorded value for a large number of their features. As several authors have noted, the vital attributes for an organization are usually correctly filled, but other attributes, less fundamental for the core business, are usually blank or incorrectly filled. In the majority of cases, this problem reflects the poverty of the tools for gathering the data.

There are several approaches to deal with this problem, which mainly consist on ignoring the records that contain missing values or on replacing the missing value with an accepted value. Since the first solution implies the exclusion of a large number of instances, it may reduce the richness of the data, impairing the results of the mining process. The replacement of missing values has been widely accepted as the solution for this problem. There are several strategies for choosing the new value, ranging from the use of a new value such as "unknown", to the use of the mean value of the attribute or the most probable value to fill in the missing value.

Outliers or Noise. Outlier values correspond to unexpected values, and are usually referred to as *noise*. This problem is mostly related with continuous attributes, where there is an order relation between values. The goal is to identify the outliers and to smooth the data, by choosing the most adequate value to replace them.

This identification can be done by grouping the values in clusters, and recognize as outliers the

values that are outside those clusters. Again, there are several strategies to choose the new value, and they mostly involve the analysis of the neighborhoods of the outlier, either by adopting the mean value between the precedent and subsequent values for the outlier (*binning* methods), by calculating the new value considering a function that better fits the data (*regression* methods) or by discretizing the values.

Inconsistent data. Data inconsistencies mostly appear as a result of the data integration operations and are easily corrected when functional dependencies among attributes and data constraints are known. When this happens, automatic or semi-automatic tools can be used, but otherwise, mainly manual processing can be applied.

Data Reduction

In general, databases contain very large amounts of data, which can result from the large number of records, the large number of attributes per record or simply from the inherent complexity of data. Since those characteristics can increase the difficulty of the mining process, the data reduction is a real need.

The simplest form of data reduction is the use of a *data sample* obtained from the original database. Given that this sample can be considerably smaller than the entire database, and that there is some danger of losing interesting data records, the sample has to be selected in order to be representative of the original data.

Another popular technique is *dimensionality reduction*, which consists in removing the attributes that seem redundant and irrelevant. In this manner, each record is transformed into a smaller feature vector. By this reason, it is commonly known as *feature selection*. One of the most popular approaches is based on the use of *Principal Components Analysis*, but others use transformations like the *Discrete Fourier* and *Wavelet Transforms* to compress data, as will be described in section 3.1.

More advanced techniques try to infer some parametric model to represent the data, for example by using linear regression to infer the line that best approximates the data points, or by using grammatical inference methods to infer some *generative model* able to generate existent data.

Other Operations

As described above, when there is a large number of attributes, it is possible to choose the most relevant ones for the task. However, sometimes, existent attributes are not able to reflect the

structure of the domain and the construction of new attributes can help to have a new insight in the inner nature of the problem. This construction is usually achieved by the combination of existent attributes, either by the Cartesian product of nominal attributes or by the conjunction of Boolean attributes.

Additionally, there are two other important techniques usually used in order to improve the quality of data: data smoothing and the use of concept hierarchies.

One of the most usual smoothing techniques is normalization. The normalization is made by scaling the possible values for some attribute, so that they fall within a specified interval, usually from *0.0* to *1.0*. In this manner, similarities may be detected, ignoring scale differences. This kind of transformation is applied to continuous values, and there are several strategies to make the transformation.

Concept hierarchies are also used to reduce the number of possible values for some attribute, by replacing low-level by higher-level concepts. When the attribute is continuous, this technique corresponds to the discretization of the original values, and concept hierarchies can be constructed automatically based on the analysis of the data distribution. When attributes are categorical, their values are already discrete, but can belong to a large set of distinct values. In this case, a concept hierarchy can be applied to abstract the values, loosing some detail but possibly improving the performance of data mining techniques. Concept hierarchies for categorical attributes are usually known as *taxonomies*.

2.2 – Data Mining

The term *data mining* also identifies the discovery step, itself, and it mainly makes use of machine learning algorithms to perform it. The great challenge of data mining is to adapt existent machine learning algorithms to use large amounts of "dirty" data in an efficient way, using all known information, to focus the discovery process in accordance with user expectations.

Several other challenges can be enumerated, but, among them, the analysis of complex data types has deserved a particular attention in the last few years, in particular, the analysis of temporal data.

The most relevant mining operations are classification/prediction, clustering and association analysis, succinctly described below.

Classification and Prediction

The aim of *classification* methods is to learn a *classifier* for predicting the value of a specific attribute, known as the *class* or *concept* attribute. In general, a classifier c can be seen as a function from a set of instances $\mathcal{I}$ to a set of class labels $\mathcal{L}$ (c: $\mathcal{I} \rightarrow \mathcal{L}$). In order to discover such functions, classification methods use a set of already classified instances, called the *training set*.

Formally, a class C_i is a subset of the domain S, consisting of all instances that satisfy the class description $desc_i$. The final goal is to discover these descriptions, which can be achieved by the discovery of the common properties among the set of training instances that belong to the same class.

Whenever those descriptions are available, it is possible to define a *classification rule* for each class. Those rules state that all instances that satisfy the description of a particular class belong to that class. A rule is said to be correct with respect to a specific training set, if its description covers all the instances of a given class (*positive instances*) and none of the instances of other classes (*negative instances*).

The distinction between classification and prediction is somehow diffuse. Most of the times the only difference is that the term classification is applied to categorical domains, while prediction is used on numerical domains. In this context, prediction methods try to discover the values for missing or unseen features, mostly by analyzing real-valued features and temporal data. While prediction methods are essentially based on the use of statistical techniques for regression, classification use several different models to discriminate existent concepts, such as decision trees, neural networks, support vector machines or Bayesian models, to name a few. Given that classification and prediction methods make use of classified instances, they are known as *supervised methods*.

A detailed description of classification and prediction methods for the analysis of temporal data is presented in section 3.2.

Clustering

Unlike classification, *clustering* is used when no labeled data exists, which means that it is an unsupervised operation. Indeed, the first goal of clustering is to partition the data in natural classes, *clusters*, by analyzing the similarities and dissimilarities between instances.

One of the main difficulties is to discover the number of classes. After this, it is necessary to

identify those classes, assign a new label to each one and discover their descriptions, like for classification methods.

In this manner, clustering is able to identify regions with different characteristics, contributing to define the overall distribution of data. Correlations among attributes may also be found, which may help in the feature selection pre-processing task.

Given that each instance should be similar to other instances in the same cluster, and dissimilar to the instances on other clusters, it is usual that clustering methods make use of a similarity measure, in order to identify the clusters. This similarity measure, also called a *distance function*, is essential to perform clustering, but can be quite hard to define, especially in the presence of complex data types. Section 3.2 describes clustering approaches to temporal sequences, and enumerates several similarity measures for this type of data, depending on the type of representation used.

Association Analysis and Frequent Itemset Mining

Association Analysis is another unsupervised task, which tries to capture existent dependencies among attributes and its values, described as association rules. An *association rule* is an implication of the form $A \Rightarrow B$, where A and B are propositions (sets of pairs attribute/value), and expresses that when A occurs, B also occurs with a certain probability. This probability is known as the rule *confidence* and is given by the conditional probability $P(B|A)$.

In basket analysis, one of the major applications of association analysis, A and B correspond to itemsets, and the rule $A \Rightarrow B$ means that if A is transacted, then B will also be transacted at the same time, with a certain probability.

This kind of discovery is usually applied to the transactions on a given database, performing an intra-transactional analysis, and, in its pure form, it is not able to perform inter-transactional analysis.

The most well known approach to discover association rules is the *apriori* algorithm [Agrawal 1994]. It acts in two steps: first, it discovers the frequent itemsets (this task is usually known as *frequent itemset mining*) and then it generates the rules. The notion of the frequency of an itemset is defined using a minimum support threshold: an itemset is frequent if its support is greater or equal to the minimum support accepted. The *support* of an itemset A is given by the probability of its occurrence in the database $P(A)$.

In order to discover frequent itemsets, *apriori* acts iteratively: first, it generates possible different itemsets (candidates) and then it chooses the frequent ones by scanning the database. It begins by trying the itemsets composed of one item, then of two items and so on, until there are no eligible candidates. The task of counting the support for each candidate is the most costly operation in apriori-based approaches, since it involves several database scans.

In the last few years, several other algorithms for association analysis have been proposed. For an overview of such methods, see for example [Zaki 1999].

The main drawback of association analysis is the huge number of discovered associations. Indeed, most of the times they are uninteresting and useless to the final user, mostly due to the lack of focus and the general inability to represent background knowledge in the discovery process.

2.3 – Post-processing

The mining operations performed in the data mining stage develop models to explain or describe the data. Nevertheless, as seen above, most of the times there are several hypotheses to describe the data, and naturally, it is necessary to choose the "best ones" to present to the final users.

The post-processing stage aims to perform the analysis of the achieved results, and to present the best ones to the final user. In this manner, there are two core tasks to perform: the evaluation of results and their presentation. Despite the fact that the presentation of the models is extremely important, the research on visualization has been almost exclusively concerned with data visualization [Fayyad 2002]. An exception to this general panorama is the system PEAR [Poças 2003], which permits the manipulation and analysis of the discovered association rules.

On the contrary, the choice of the best results to present to the users has motivated a large number of researchers. In essence, the evaluation of models is concerned with four aspects: its simplicity, its certainty, its utility and its novelty.

Simpler explanation models are usually preferred because they are easier to understand and because they are more suitable for generalizing beyond known instances. This simple principle is known as the *Occam's razor*. The difficult is on choosing between models with different certainties. The *Minimum Description Length* gives an answer to the challenge of choosing the best and simpler model, minimizing the sum of the description length of the explanation model and the description length of the data given the model [Mitchell 1997].

The certainty of a model can be stated as the measure of the trust that an user should put on the

model. In the case of classification models, this certainty is the *accuracy* or *precision*, which is the probability that the model correctly classifies unseen instances. For association rules, this certainty is given by its *confidence*, as described before.

Utility measures determine the interestingness of models. In general, it is concerned with two aspects: the usefulness and the novelty of the models. Again, utility measures are different for classification and association models.

While the *support* of an association rule (as previously defined – section 2.2) measures its usefulness, the *coverage* indicates the usefulness of a classification model. In particular, the coverage of a class description is the probability that an arbitrary instance, belonging to that class, is covered by that description.

Novelty measures define the contribution of models to increase the knowledge about the domain. An example of such measures is the *lift* of an association rule, which reflects the strength of the effect, by measuring the ratio between the confidence of the rule and the support of the basket on the consequent of the rule. Several other interestingness measures exist, for both classification models and association rules. A survey of those measures can be found in [Hilderman 1999].

3 – Temporal Data Mining

Since the great majority of phenomena occur over time, the analysis of temporal data has been one of the data mining goals, from its beginning. However, traditional data mining operations are not able to deal with their intrinsic dynamic nature, since they usually treat temporal data as unordered collections of events, ignoring temporal information. As seen before, they usually center their attention into the analysis of the data transactions (*intra-transactional analisys*), instead of analyzing the relations between different transactions (*inter-transactional analisys*).

In the last decade, the exploration of temporal data, usually called *temporal data mining*, achieved a considerable attention in the data mining community. Its main goal is to provide the ability to explore the dynamic aspects of entities, instead of only exploring its static characteristics. In particular, with this kind of analysis, it is possible to infer some cause-effect relations, allowing for the understanding of the evolution of analyzed entities [Roddick 2002].

Given that, the analysis of temporal data has been a reality for centuries, any overview of the

techniques used is necessarily incomplete. The next sections do not aim to describe all these techniques, but simply to survey the approaches mostly used by members of the data mining community in the analysis of temporal data. It is clear that many other approaches have been used, for example, in the signal processing and pattern recognition areas, but almost all of them were not used in the knowledge discovery process, and are unknown for the majority of "data miners". Since those techniques were mostly applied in the analysis of continuous signals or images, and the goal of this dissertation is to deal with nominal data, we do not survey them.

3.1 – Pre-processing of Temporal Data

The pre-processing of event sequences can be roughly seen as the selection of the most adequate representation for data. This choice is especially important when dealing with time series, since direct manipulation of continuous, high-dimensional data in an efficient way is extremely difficult.

The representation of event sequences, in general, can be addressed in several different ways. One first possibility is to use the data with only minimal transformations, either keeping it in its original form or using windowing and piecewise linear approximations to obtain manageable sub-sequences. A second possibility is to use a reduction that maps the data to a more manageable space, either continuous or discrete. Another possibility is to use a generative model, either statistical or deterministic, inferred from the data. While the three first approaches deal directly with time series, parametric models have been mostly applied to nominal event sequences.

One issue that is relevant for all the representation methods addressed is the ability to discover and represent the subsequences of a sequence. The method mostly used to find subsequences is to use a sliding window of size w and place it at every possible position of the sequence. Each such window defines a new subsequence, composed by the elements inside the window [Faloutsos 1994].

Measuring the Similarity between Temporal Sequences. After representing each sequence in a suitable form, it is, in general, necessary to define a similarity measure between sequences, in order to determine if they are closely related.

An important issue in measuring similarity between two sequences is the ability to deal with outlying points, noise in the data, amplitude differences (*scaling problems*) and the existence of gaps and other time axis distortion problems (*translation problems*). As such, similarity measures need to be sufficiently flexible to be applied to sequences with different lengths, noise levels, and time scales. There are many proposals for similarity measures, but the model used to represent

sequences clearly has a great impact on the similarity measure adopted. In this manner, after the description of each category of representation models, we present the similarity measures usually used.

Time Domain Continuous Representation

The simplest approach to represent a time series consists in using it without any pre-processing [Agrawal 1995a] [Lin 1998]. Figure 2.2 illustrates this situation.

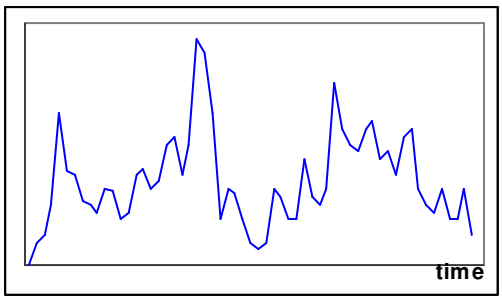


Figure 2.2 – Time series representation in time domain

However, as been said, mining operations do not deal efficiently with this kind of data. An alternative consists in finding a piecewise linear function able to approximately describe the entire sequence.

In this way, the original sequence is partitioned in several segments and each segment is represented by a linear function. Despite the fact that sequence partition can be achieved using different approaches, all of them first have to define or discover the number of segments, and then partition the original data [Das 1997], [Guralnik 1999]. The objective is to obtain a representation that can be used to detect significant changes in the sequence.

A proposal to discover the number of segments consists on segmenting a sequence by iteratively merging two similar segments. Choosing which segments are to be merged is done based on the squared error minimization criteria [Keogh 1997]. An extension to this model consists on associating a weight value to each segment, which represents the segment importance relatively to the entire sequence. In this manner, it is possible to compare sequences mainly by looking at their most important segments [Keogh 1998].

With this representation, a more effective similarity measure can be defined, and consequently, mining operations may be more easily applied. Figure 2.3 exemplifies a time series using linear segments, presenting the original with grey and the approximation with solid lines.

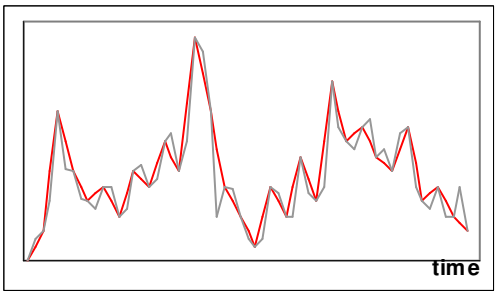


Figure 2.3 – Representing a time series using linear segments

While this is a relatively straightforward representation, not much is gained in terms of the ability to manipulate the generated representations. One possible application of this type of representations is on change-point detection, where the goal is to identify the points where a significant change in behavior takes place. One significant advantage of these approaches is the ability to reduce the impact of noise. However, problems with amplitude differences (scaling problems) and the existence of gaps or other time axis distortion are not addressed easily.

Similarity Measures. When no pre-processing is applied to the original sequence, a simple approach to measure the similarity between two sequences consists on comparing the i^{th} element of each sequence. The similarity measure most used with time-domain continuous representations is the Euclidean distance, by viewing each sub-sequence with *n* points as a point in R^n. Figure 2.4 illustrates the use of this method to compare two time series, where the illustration on the right assumes n=3.

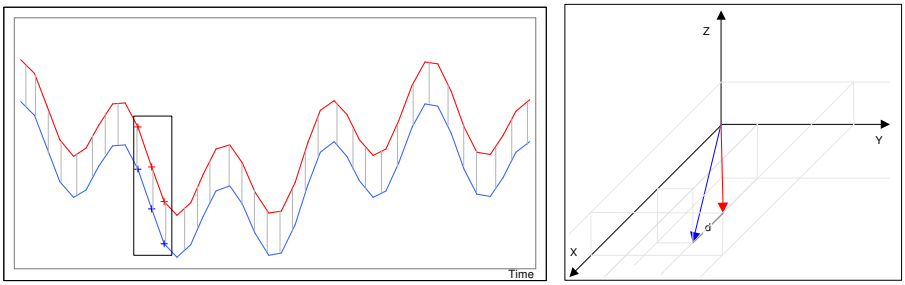


Figure 2.4 – Comparing two time series with the Euclidean distance

This simple method, however, is insufficient for many applications. In order to deal with noise, scaling and translation problems, a simple improvement consists on determining the pairs of portions of sequences that agree in both sequences after some linear transformation is applied. A concrete proposal [Agrawal 1995a] achieves this by finding one window of some fixed size from each sequence, normalizing the values in them to the [-1, 1] interval, and then comparing them to determine if they match, as illustrated in Figure 2.5.

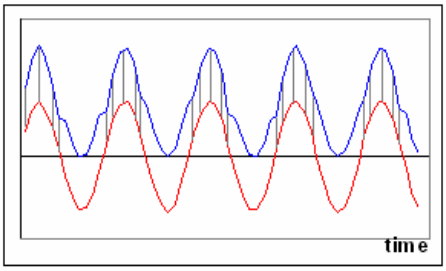


Figure 2.5 – Comparing two time series using time windows

After identifying these pairs of windows, several non-overlapping pairs can be joined, in order to find the longest match length. Normalizing the windows solves the scaling problem and searching for pairs of windows that match solves the translation problem.

A more complex approach consists on determining if there is a linear function *f*, such that a long subsequence of one sequence can be approximately mapped into a long subsequence of the other [Das 1997]. Since, in this approach, the subsequences do not necessarily consist of consecutive original elements but only have to appear in the same order, this similarity measure allows for the existence of gaps and outlying points in the sequence.

In order to surpass the high sensibility of the Euclidean distance to small distortions in the time axis, the *Dynamic Time Warping* (DWT) technique has been used [Berndt 1996]. This technique consists on aligning two sequences so that a predetermined distance measure is minimized. The goal is to create a new sequence, the *warping path*, whose elements are (*i*, *j*) points with *i* and *j* the indexes of original sequence elements that match. Algorithms for using dynamic time warping techniques have been proposed in [Keogh 1999] and [Yi 1998]. Figure 2.6 illustrates this method.

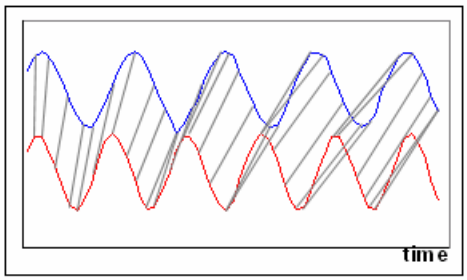


Figure 2.6 – Comparing two time series using *Dynamic Time Warping*

A different similarity measure consists on describing the distance between two sequences using a probabilistic model [Keogh 1997].

The general idea is that a sequence can be 'deformed', according to a prior probability distribution, to generate the other sequence. This is easily accomplished by composing a global shape sequence with local features, where each one is allowed some degree of deformation. In this

way, some elasticity is allowed in the global shape sequence, allowing for stretching in time and distortions in amplitude. The degree of deformation and elasticity are governed by the prior probability distributions. Figure 2.7 exemplifies the usage of this type of measure.

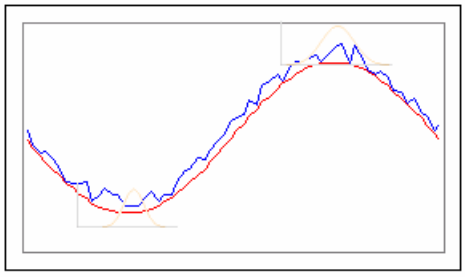


Figure 2.7 – Comparing two time series using a probabilistic measure

Compression Based Representations

The main idea of *Compression Based Representations* is to transform the time series, from time to another domain, and then to use a point in this new domain to represent each original sequence.

The simplest compression is to represent each sequence as a point in the *n*-dimensional space, with *n* being the sequence length. Another possibility is to map each subsequence to a point in a new space with coordinates that are the derivatives of each original sequence point (the derivative at each point is determined by the difference between the point and its preceding one [Gavrilov 2000]).

However, with these methods almost no reduction is performed. The most usual reduction approach is to use the *Discrete Fourier Transform* (DFT) to transform a sequence from the time domain to a point in the frequency domain [Agrawal 1993]. Choosing the *k* first frequencies, and then representing each sequence as a point in the *k*-dimensional space, achieves this goal. The DFT has the attractive property that the amplitude of the Fourier coefficients is invariant under shifts, which allows for extending the method to the problem of finding similar sequences ignoring shifts. This technique is illustrated in Figure 2.8.

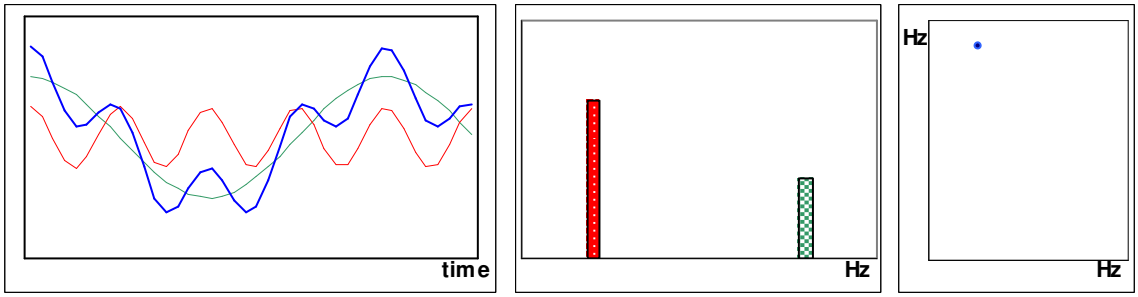


Figure 2.8 – Representing a time series using the Discrete Fourier Transform, with k=2

A more recent approach uses the *Discrete Wavelet Transform* (DWT) to translate each sequence from the time domain into the time/frequency domain [Chan 1999]. The DWT is a linear transformation, which decomposes the original sequence into different frequency components, without loosing the information about the instant of the occurrence of elements. Its features, expressed as the wavelet coefficients, represent the sequence. Again, only a few coefficients are needed to approximately represent the sequence. Figure 2.9 exemplifies the usage of wavelets for representing time series.

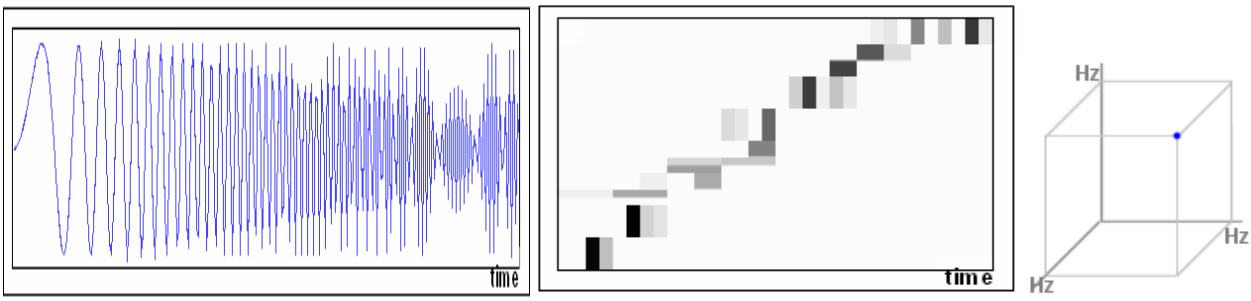


Figure 2.9 – Representing a time series using *Discrete Wavelet Transform*, with k=3

With these kinds of representations, time series became a more manageable object, leading to the definition of efficient similarity measures and an easier application of common data mining operations. However, these methods do not solve either amplitude differences or time distortions.

Similarity Measures. When Compression Based Representations are used, a simple approach is to compare the points, in the new domain, which represent each sequence. For example, the similarity between two sequences may be reduced to the similarity between two points in the frequency domain, again measured using the Euclidean distance [Agrawal 1993], [Chan 1999], as Figure 2.10 illustrates.

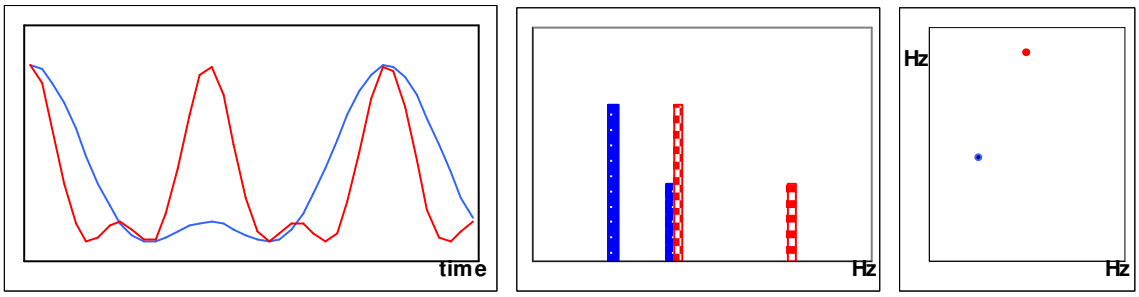


Figure 2.10 – Comparing two time series in frequency domain

In order to allow for the existence of outlying points in the sequence, the discovery of sequences similar to a given query sequence is reduced to the selection of points in a neighborhood of the query point. However, this method may allow for false hits, and after choosing the neighbor

points in the new domain, the distance in the time domain is computed to validate the similarity between the chosen sequences (for instance, using the Euclidean distance as described in the previous section).

Notice that this method implies that sequences have the same length. In order to be able to deal with different lengths, the method was improved to find relevant subsequences in the original data [Faloutsos 1994]. Another important issue is the fact that to compare different time series, the number of Fourier coefficients chosen to represent the sequences must be the same for all of them, which may not be the optimal one for each sequence.

Representations Achieved by Discretization

A third approach to deal with time series is the translation of the original sequence to a sequence composed of nominal symbols. There are two problems related with this translation: choosing the domain of the new symbols – *alphabet*, and making the translation from the real-valued elements to new symbols.

The alphabet definition is sometimes simplified by assuming that digits or characters are used. However, this decision implies that the new sequence is an artificial transformation of the original one, potentially more difficult to analyze and understand.

Automatic Conversion Methods. The simplest way to define the alphabet is to choose a set of nominal symbols (alphabet) without any special meaning, and then proceed with the translation itself. This kind of approach introduces some level of artificiality but is especially useful when there is no knowledge about the series domain.

A simple method to discretize time series is to segment a sequence by computing the *change ratio* from one time point to the following one, and representing all consecutive points with equal change ratios by a unique segment. After this partition, each segment is represented by a symbol, and the sequence is represented as a string of symbols [Huang 1999]. A particular case occurs when change ratios are classified into one of two classes: large fluctuations (symbol *1*) or small fluctuations (symbol *0*). This way, each time series is converted into a binary sequence, which can then be manipulated in a natural way by genetic algorithms [Szeto 1996], for example.

Clustering is another approach to convert a sequence into a discrete representation. First, the sequence originates a set of sub-sequences with length w, by sliding a window of width w. Then, the set of all sub-sequences is clustered, originating k clusters, and a different symbol is associated

with each cluster. The nominal version of the initial sequence is obtained by substituting each sub-sequence by the symbol associated to the cluster that it belongs to [Oates 1999].

Figure 2.11 illustrates the translation of one sequence, if we associate the symbol *a* to ascending segments (solid lines), symbol *b* to descending segments (dotted lines) and symbol *c* to flat segments (grey lines). Its final representation would be <*bcacbcacbcacbcacbca*>.

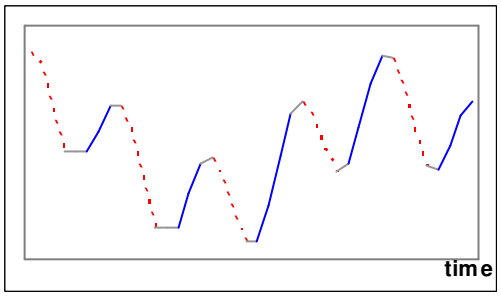


Figure 2.11 – Representing a time series using *clustering* as a discretization method

Another method to convert time series into a sequence of symbols is based on the use of *self-organizing maps* – SOM [Giles 2001], [Guimarães 2000]. This conversion consists on three steps: first, a new series composed by the differences between consecutive values of the original time series is derived; second d-sized windows over the series are given as input to the SOM, which finally outputs the node closer to the input. Each node is associated with a symbol, which means that the resulting sequence may be viewed as a sequence of nominal symbols.

The advantage of these methods is that time series are partitioned in a natural way, depending on their values. These transformed time series are more amenable to manipulation and processing than the original time series, since the number of possible different values is reduced. However, the symbols of the alphabet are usually chosen externally, which means that most of the times users impose symbols in some artificial way.

Alphabet Specification Languages. A way to define the alphabet without loosing the original meaning is to define a language able to describe all different entities relevant to the domain, and simultaneously, to provide some similarity measure between different sequences.

One such language is the *Shape Definition Language* (SDL) proposed by Agrawal [Agrawal 1995b], which is able to describe shape queries to pose to a sequence database and is able to perform 'blurry' matching. A *blurry match* is one where the important thing is the overall shape, therefore ignoring the specific details of a particular sequence. The first step in the representation process is defining the alphabet of symbols and then translating the original sequence to a sequence of nominal symbols. The translation is done by considering transitions from an instant to the

following one, and then assigning a symbol of the alphabet to each transition.

Figure 2.12 presents the new sequence (*s'*=< *zero stable Up Up Up down stable Down down disappears* >) corresponding to sequence s=< 0 0 0.5 2 4 6 5 4.5 1.5 0.5 0 >, using the alphabet Σ={*zero*, *stable*, *Up*, *down*, *Down*, *disappears*}.

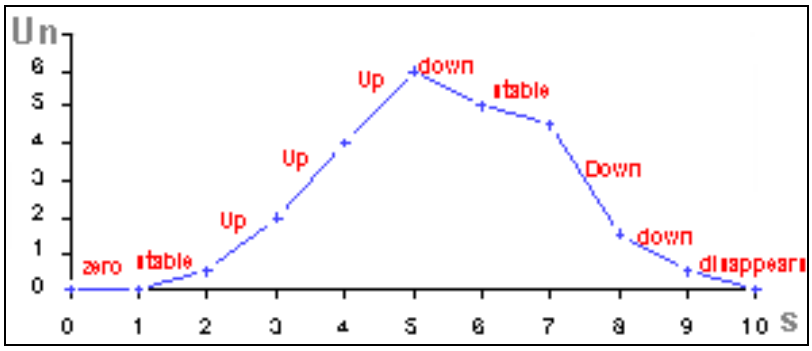


Figure 2.12 – Representing a time series using *SDL*

Another language used to specify patterns is the *Constraint-Based Pattern Specification Language* (*CAPSUL*) [Chakravarty 2000], which like *SDL* describes patterns by considering some abstraction over the values at each time interval. The key difference between these two languages is that CAPSUL uses a set of expressive constraints upon temporal objects, which allows describing more complex patterns, for instance, periodic patterns. Finally, to perform each of those abstractions the system accesses an ontology, where relations among values are defined.

Similarity Measures. When a sequence is represented as a sequence of nominal symbols, the similarity between two sequences is, in general, achieved by comparing each element of one sequence with the corresponding one in the other sequence, as proposed in the first similarity measure described above. However, if the categorical symbols are not ordered some provisions must be taken to allow for blurry matching.

One possible approach is to use the *Shape Definition Language*, where a blurry match is automatically possible. This is possible because the language defines a set of operators, such as *any*, *noless* or *nomore*, which allows describing overall shapes, solving the existence of gaps in an elegant way [Agrawal 1995b].

Figure 2.13 exemplifies two cases of similarity between two pairs of sequences. Both pairs are considered similar since they verify the query

```
(in 5 (and (noless 2 (any up Up)) (nomore 1 (any down Down))))
```

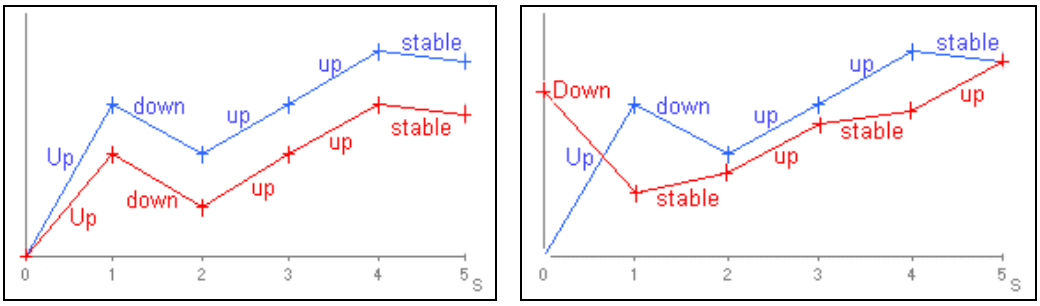


Figure 2.13 – Comparing two time series using the *blurry matching* [Agrawal 1995b]

Another possibility is to use well-known algorithms for approximate string matching [Navarro 2001] to define a distance over the space of strings of discrete symbols [Huang 1999], [Mannila 1997]. This distance, usually known as *edit distance*, reflects the amount of work needed to transform a sequence to another, and is able to deal with different sequence lengths and the existence of gaps. Insertion, deletion, replacement and transposition are the operations usually accepted to transform a string into another one. The edit distance will be formally defined in Chapter 6, section 3.

Generative Models

A significantly different approach consists on applying data reduction techniques in order to obtain a model that can be viewed as a generator for the sequences, as described in section 2.1. There are different types of models, to represent event sequences, in general, and nominal ones, in particular. These models can be classified into two main categories: *graph-based models* and *formal languages*.

Graph Based Models. One possible approach is based on the use of acyclic graphs that define partial orders or constraints between basic events – *episodes*. In this context, an event is a pair (A, t), with A a symbol to describe the event type [Mannila 1995] [Guralnik 1998]. Note that with this kind of model, it is possible to represent events that occur either in sequence or in parallel. Further developments were proposed: one for dealing with events that occur during a set of different instants [Laxman 2002] and other for building a mixture model that generates, with high probability, sequences similar to the one that one wishes to model [Mannila 2000a].

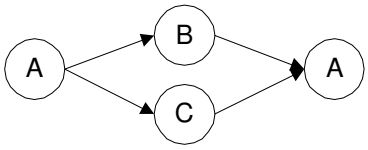


Figure 2.14 – Example of an episode

Figure 2.14 shows an example of an episode, with four events: first A occurs, second B and C occur in parallel and then A occurs again.

In order to deal with different time granularities (for example hours, days or weeks), TAGs were proposed [Bettini 1998]. A TAG (*Timed Finite Automaton with Granularities*) is essentially a standard finite automaton with a set of clocks, each of which as an associated granularity. Instead of each transition being conditioned only by the input symbol, it also depends on the values of the different clocks. In this manner, the transitions are constrained both by the content and by the temporal information contained in the sequence of events.

These models can be viewed as graph-based models, since one or more graphs that describe the relationships between basic events are used to model the observed sequences.

Formal Languages. An alternative approach to describe the relations between events is based on the use of a formal language. A *formal language* is a set of nominal sequences generated by some *grammar*.

Grammars may belong to a variety of classes [Hopcroft 1979], ranging from regular [Lang 1998], [Juillé 1998], [Oliveira 2001], to context-free [Sakakibara 1997], [Sakakibara 2000], context-sensitive and natural grammars. Given that regular grammars are equivalent to finite automata, context-free grammars to non-deterministic push-down automata, context-sensitive to linear bounded automata, and grammars for natural languages to Turing machines, formal languages can also be seen as a kind of graph-based models. We choose to distinguish them, because they have been exhaustively studied and their properties are well known.

Extensive research in grammatical inference methods has led to many interesting results [Miclet 1996], [Honavar 1998], [Oliveira 2000] but has not yet found its way into a significant number of real applications. Usually, grammatical inference methods are based on discrete search algorithms that are able to infer complex grammars. Neural net-based approaches have also been tried (see, for example [Moody 1992]) but have been much less successful when applied to complex problems [Horne 1995], [Giles 1992]. However, some recent results have shown that neural-network based inference of grammars can be used in realistic applications with some success [Giles 2001] and regular languages can be used to represent background knowledge in some mining operations [Garofalakis 1999], [Han 2001b].

When there are probabilities associated to the relations between the occurrences of events, we are in the presence of a probabilistic model. Either *Stochastic Grammars* [Carrasco 1994],

[Stolcke 1994], [Higuera 1998], [Smyth 1999] or *Hidden Markov Models* [Ge 2000], [Smyth 1997] are examples of such models.

Similarity Measures. When a generative model is obtained from a dataset, the similarity measure between sequences can be obtained by verifying how close the sequence fits one of the available models.

For deterministic models (graph-based models, regular grammars or context-free), verifying if a sequence matches a given model provides a *yes* or *no* answer, and, therefore, the distance measures are necessarily discrete, and, in many cases, only take one of two possible values.

For stochastic generative models, it is possible to obtain a number that indicates the probability that a given sequence was generated by a given model.

In both cases, this similarity measure can be used effectively in classification problems, without the need to resort to complex heuristic similarity measures between sequences.

It is clear that formal languages obtained by induction from examples can be used for prediction and classification, since it is possible to verify if the model can generate a given sequence, and what is the most likely event after the observed ones.So far, however, these methods have not been extensively applied to actual problems in data mining, mainly due to the difficulties inherent to the inference process.

3.2 – Mining Operations on Temporal Data

As has been noted, temporal data mining methods perform inter-transactional analysis, discovering information related to dynamical behavior of target entities. In this context, prediction and association analysis have deserved more attention than classification and clustering.

In particular, while prediction of event sequences allows for the identification of trends, association analysis is able to provide a set of relations among events, identifying the impact of some occurrences over others.

In this section, it is presented a small overview of the several approaches to temporal mining operations.

Classification and Prediction

In the specific domain of prediction, care must be taken with the domain where prediction is to be applied. Well-known and generally accepted results on the inherent unpredictability of many

financial time series [Fama 1970], [Weigend 1994] imply that significant gains in prediction accuracy are not possible, no matter how sophisticated the techniques used. However, financial time series forecasting has been the focus of research on prediction on the last decades. As such, several authors have presented work that aims specifically to obtain algorithms that can be used to predict the evolution of time series using genetic algorithms [Cortez 2001], [Szeto 1996], recurrent neural networks, or available background knowledge on economics and finance [Levitt 1996], [Mannila 1999].

In the vast literature on computer-assisted prediction of time series, a large fraction is based on neural networks. In order to deal with this kind of data, two main types of neural networks are used: *time delay neural networks* (TDNN) and *recurrent neural networks* (RNN).

A TDNN receives an entire input sequence as a single input, using delay units to hold the past values of the original sequence. On one hand, this kind of networks imposes that the length of the sequence cannot be variable and too long, because of its nature and the resulting unmanageable size of the TDNN, respectively. On the other hand, it only performs a static association mapping between the input and output patterns, since for each input sequence it retrieves the output pattern associated with the prototype of the most frequent input vector configuration [Berthold 1999]. A neural network is called recurrent (RNN) if cross, auto and backward connections are allowed. These networks have shown to be adequate to dynamic learning tasks and can be used to learn a mapping function from the space of input sequences to the space of output sequences. The main drawback of these networks resides on the complexity and computational costs of their training algorithms, mainly because of their complex topologies [Berthold 1999].

Applications of RN networks can be found in the financial and medical fields [Giles 2001], [Guimarães 2000]. A good example of such applications is the work by Giles et al., which combines discretization with grammar inference, and applies the resulting automaton to explicitly predict the evolution of financial time series, using recurrent neural networks.

Classification is one of the most typical operations in supervised learning, but only recently has deserved some attention in temporal data mining.

Since traditional classification algorithms are difficult to apply to sequences, mostly because there are a vast number of potentially useful features for describing each example, an interesting improvement consists on applying some data reduction technique to extract relevant features. One approach to implement this idea consists on discovering frequent sub-sequences, and then using them, as the relevant features to classify sequences with traditional methods, like *Naive Bayes* or

Winnow [Lesh 1999]. Another possibility is to use *support vector machines* (SVMs), with some improvements, that may include the incorporation of non-linear time alignments into the kernel function, extending the original method to the case of variable length sequences [Shimodaira 2001]. Applications on the analysis of electroencephalograms [Blankertz 2002] or on speech recognition [Shimodaira 2001] using SVMs have already been proposed.

Another classification method that deals specifically with nominal event sequences is based on the *merge* operator [Keogh 1998]. This operator receives two sequences and returns a sequence whose shape is a compromise between the two original ones. The basic idea is to iteratively merge a typical example of a class with each positive example, building a more general model for the class. Using negative examples is also possible, but then is necessary to emphasize the shape differences. This is accomplished by using an influence factor to control the merge operator function: a positive influence factor implies the generalization of the input sequences, and a negative reinforces the differences between the positive and negative shapes.

Classification and prediction are relatively straightforward if generative models are employed to model the data. Deterministic and probabilistic models can be applied in a straightforward way to perform classification [Lesh 1999] since they give a clear answer to the question of whether a sequence matches a given model, but have not been used widely.

Clustering

There are two central problems in clustering temporal data: choosing the number of clusters and initializing their parameters, and defining a similarity measure among event sequences.

Considering that the number of clusters (say K) is known, and that a sequence is viewed as being generated according to some probabilistic model (for example, a Markov model), clustering may be viewed as modeling the data sequences as a finite group of K sequences in the form of a finite mixture model. Using the EM algorithm [Dempster 1997] their parameters could be estimated and each group would correspond to a cluster [Smyth 1999]. Learning the value of K, if it is unknown, may be accomplished by a Monte-Carlo cross validation approach as suggested by some authors [Smyth 1997].

A different approach proposes to use a hierarchical clustering method to cluster temporal sequences [Ketterlin 1997]. The clustering algorithm used has been the COBWEB [Fisher 1987] and it works on two steps: first, it groups the elements of the sequences, and then it groups the sequences themselves. Considering a simple time series, the first step is accomplished without

difficulties, but to group the sequences is necessary to define a generalization mechanism for sequences. Such mechanism has to be able to choose the most specific description for what is common to different sequences.

Association Analysis

One possible modification to the formulation of association rules, when dealing with temporal data, consists on extending the notion of a typical rule $A \Rightarrow B$ to be a rule with a new meaning: $A \Rightarrow^{T} B$ (which states: if *A* occurs then *B* will occur within time interval *T*) [Das 1998]. Stating a rule in this new form allows for controlling the impact of the occurrence of an event in the probability of the occurrence of another event, within a specific time interval. An easy way to find temporal association rules is to adapt the algorithms for discovering sequential patterns, imposing some constraint (a *gap constraint*) on the notion of *contained in* when the support of each sequence is counted – this is usually called *sequential pattern mining*.

There are several sequential pattern mining algorithms from which GSP [Srikant 1996] is the best-well known apriori-based algorithm. Like apriori, it acts iteratively: first it generates possible different sequences and then it chooses the large ones, by scanning the database, until there are no candidates. Given that the task of counting the support for each candidate is the most costly operation, another possibility is to avoid the candidate generation step. Two examples of such proposals are the *PrefixSpan* and *SPADE* algorithms.

PrefixSpan (PREFIX projected Sequential PAtterN mining) [Pei 2001] acts recursively reducing the search space at each step, avoiding the generation of non-frequent sequences. *SPADE* (Sequential PAttern Discovery using Equivalence classes) [Zaki 1998a] scans the database only three times, by transforming the initial database into a vertical database, where each entry is an item id-list (a list of customer and transaction id pairs). This new layout allows for the restriction of the search-space like *PrefixSpan*.

A more recent method to find sequential patterns is SPAM [Ayres 2002], a depth-first algorithm which maintains the entire database in memory, represented as a bitmap. SPAM uses a lexicographic sequence tree to generate candidate sequences and uses candidate pruning techniques like GSP, to reduce the number of candidates. The use of bitmaps simplifies the support counting procedure, which becomes a simple check of the bitmap values: if all bits for the specified sequence are set, then the candidate is supported by the current database sequence. However, SPAM assumes that the entire database and all used data structures completely fit into main

memory, and the use of any constraint was not introduced.

As pointed before, one of the main problems of algorithms for mining patterns is the lack of focus or user control [Ng 1998]. An interesting family of algorithms, named *SPIRIT* (Sequential Pattern mining with Regular Expressions) [Garofalakis 1999] adapt apriori-based algorithms to use regular expressions (embedded in the algorithm as finite automata) to constrain the generation of candidates, which reduces the acceptable candidates for which support counting is needed. *PrefixGrowth* is a pattern-growth method that like *SPIRIT* uses regular languages to constrain the mining process [Pei 2002b], and *cSPADE* expands *SPADE* to use several constraints [Zaki 2000].

A similar task is the discovery of frequent episodes in a unique event sequence. The goal is essentially the same: to find all frequent episodes in the given event sequence, given a class of episodes (which can be seen as a constraint). A description of algorithms for this task can be found in [Mannila 1995], [Mannila 1996] and [Guralnik 1998].

When applied to temporal data, association analysis has two additional goals: to identify time periodicities among data (known as *periodicities analysis*) and to find valid time periods during which association rules hold [Chen 2000].

In order to discover periodicities or *cyclic rules* between temporal data, it is necessary to search for frequent patterns in a restricted portion of time, since they may occur repeatedly at specific regular time instants but on a little portion of the global time considered. A method to discover such rules is based on the application of an algorithm similar to *apriori* that, after having obtained the set of traditional rules, detects the cycles behind the rules. A more efficient approach to discover cyclic rules consists on inverting the process: first, it discovers the cyclic patterns and then generates the rules [Özden 1998]. A natural extension to this method consists on allowing the existence of different time units, such as days, weeks or months, and is achieved by defining calendar algebras to define and manipulate groups of time intervals. Rules discovered are designated *calendric association rules* [Ramaswamy 1998]. A similar idea was applied in the search of periodicities in time series [Han 1998]. The main idea is to look for the discovery of periodic segments, recognizing that only some segments have cyclic behavior.

An approach to determine time periods during which association rules hold, is also apriori-based and consists on partitioning the initial data in several time periods and then applying the pattern mining process to each partition [Lee 2001].

3.3 – Post-processing for Temporal Information

In general, the evaluation criteria for discovered temporal patterns apply to temporal data with a particular emphasis, given the complexity of this kind of data. Again, simpler models are preferred and the trade-off between accuracy and coverage remains unchanged. Furthermore, to our knowledge, there are no specific interestingness measures for discovered temporal patterns, except for the definition of support, as stated above.

4 – Open Issues in Temporal Data Mining

From the survey presented, it is clear that there are several approaches to explore temporal data. However, the number and the diversity of solutions reflect the small consensus among what the best solutions are, and shows that the key problems of dealing with real temporal data are not resolved yet.

In particular, the analysis of nominal event sequences has deserved relatively little attention, when compared with the exploration of time series. Sequential Pattern Mining is one the few approaches specifically designed to deal with nominal sequences, but even state of the art sequential pattern mining algorithms are still likely to discover large amounts of patterns, many of them useless when applied in real problems.

Summary

In this chapter, we have introduced the data mining field, presenting its fundamental concepts and techniques, and motivating the need for special procedures when dealing with temporal data.

We have presented an overview of the pre-processing, mining and post-processing techniques applied to this type of data. In terms of pre-processing, we have categorized and described the existent models to represent temporal data: time-domain continuous, and compression based representations, models achieved by discretization and generative models.

For mining temporal data, we have succinctly explained the existent methods to perform the main tasks of data mining: classification, prediction, clustering and pattern mining. The existence of so many different approaches reflects the small consensus among them.

Chapter 3
Related Work and Thesis Statement

In this chapter, we review related work, presenting the sequential pattern mining problem and the approaches to deal with it. After identifying the open issues in sequential pattern mining and its application to temporal data, we make our thesis statement, posing the research questions that have driven this work, and giving an overview of how these questions are answered.

The exploration of nominal sequences in general, and of nominal event sequences in particular, has deserved some attention only in the last decade. Among the few existent approaches to this problem, sequential pattern mining is the one that has been most explored.

In general, we can see sequential pattern mining as an approach to perform inter-transactional analysis of nominal event sequences. Indeed, sequential pattern mining was motivated by the need to perform this kind of analysis, mostly in the retailing industry, but with applications in other areas like the medical domain.

This dissertation focuses on the problem of pattern mining over nominal temporal data, presenting a new mining methodology based on the use of the sequential pattern mining approach. For this reason, a detailed review of the related work in this area is presented below.

1 – Related Work: sequential pattern mining

Sequential Pattern Mining algorithms address the problem of discovering the existent maximal frequent sequences in a given database. Algorithms for this problem are relevant when the data to

be mined has some sequential nature, i.e., when each piece of data is an ordered set of elements, like event sequences in the case of temporal information, or nucleotides and amino-acid sequences for problems in bioinformatics.

The problem was first introduced by Agrawal and Srikant, that established the basic concepts involved in pattern detection [Agrawal 1995c]. Some interesting issues were added in subsequent work [Srikant 1996], like the use of taxonomies and the introduction of time constraints. In the last years, several sequential pattern mining algorithms have been proposed, but not all of them assume the same conditions. Despite the fact that the use of constraints is somehow orthogonal to the general philosophy of pattern mining algorithms, the use of gap constraints (the simplest form of time constraints) has a great impact on their design and efficiency.

In this section, we will introduce the problem of sequential pattern mining formally, describing its adaptations to deal with nominal temporal data.

1.1 – Problem Statement and Analysis

In order to introduce the problem formally, we will begin by giving the definitions of sequence and its related concepts.

Definition 1 - An *item* is an element from a totally ordered set Σ, called the *item collection*.

Items represent the objects under analysis, e.g., the objects manipulated in a given transaction. They correspond to the atomic entities in sequential pattern mining.

An *itemset*, also called a *basket*, represents the set of items that occur together. If the data is time dependent, an itemset corresponds to the set of items transacted at a particular instant by a particular entity. Formally,

Definition 2 - An *itemset* is an ordered set of non-repeated items.

The itemset composed of items a and b is denoted by (a,b). The i^{th} item of the itemset a_n is denoted by a_n^i.

The assumption that the items in an itemset are in a specific order facilitates the design of sequential pattern mining algorithms, avoiding the repetition of some operations (such as the generation of repeated sequences). This assumption also enables the definition of two predicates over itemsets.

Definition 3 - Given two itemsets $a=(a_1,\ldots,a_n)$ and $b=(b_1,\ldots,b_m)$, with n<m: a is a *prefix-itemset* of b iff for all $1\leq i\leq n$ a_i is equal to b_i.

Definition 4 - Given two itemsets $a=(a_1,\ldots,a_n)$ and $b=(b_1,\ldots,b_m)$, with n<m: a is a *suffix-itemset* of b iff for all $1\leq i\leq n$ a_i is equal to b_{m-n+i}.

Moreover, we can define the notion of *contained in* for itemsets:

Definition 5 - Given two itemsets $a=(a_1,\ldots,a_n)$ and $b=(b_1,\ldots,b_m)$, a is contained in b (denoted by $a\subseteq b$) iff there exist integers $1\leq i_1<i_2<\ldots<i_n\leq m$ such that $a_1= b_{i_1}$, $a_2= b_{i_2}$, …, $a_n=b_{i_n}$.

Finally, we are able to define the concept of sequence.

Definition 6 - A *sequence* is an ordered set of itemsets. A sequence is *maximal*, with respect to a set of sequences, if it is not contained in any other sequence of the set.

A sequence with k items is called a k-sequence. The number of elements (itemsets) in a sequence s is the *length* of the sequence and is denoted by $|s|$. The i^{th} itemset in the sequence is represented by s_i and <> denotes the empty sequence. The result of the concatenation of two sequences x and y is a new sequence denoted by xy.

The set of considered sequences is usually designated by *database* or *dataset* (DB), and the number of sequences by *database size* (|DB|).

Definition 7 - A sequence $a=<a_1a_2...a_n>$ is *contained* in another sequence $b=<b_1b_2...b_m>$, or a is a *subsequence* of b, if there exist integers $1\leq i_1<i_2<\ldots<i_n\leq m$ such that $a_1\subseteq b_{i_1}$, $a_2\subseteq b_{i_2}$, …, $a_n\subseteq b_{i_n}$.

If s' is a subsequence of s is, this relationship is denoted by $s'\subseteq s$, and by $s'\subset s$ if s' is a *proper subsequence* of s, i.e. if s' is a subsequence of s but is not equal to s.

Prefixes and *suffixes* are subsequences with specific meanings.

Definition 8 - A sequence $a=<a_1a_2...a_n>$ is a *prefix* of another sequence $b=<b_1b_2...b_m>$ if $n\leq m$ and $a_i=b_i$ for $i<n$ and a_n is a prefix-itemset of b_n. The prefix is said to be a *proper maximal prefix* if it is equal to the original sequence without the last item on the last itemset.

Definition 9 - A sequence $a=<a_1a_2...a_n>$ is a *suffix* of another sequence $b=<b_1b_2...b_m>$ if $n \leq m$, and $a_i=b_{m-n+i}$ for $2 \leq i \leq n$ and a_1 is a suffix-itemset of b_{m-n+1}. The suffix is said to be a proper *maximal suffix* if it is equal to the original sequence without the first item of the first itemset.

In order to define the sequential pattern mining problem, two other notions are needed: frequent sequences and sequential patterns.

Definition 10 - Given a database D of sequences and some user-specified minimum support threshold σ, a sequence is *frequent* in the database D, if it is contained in at least σ sequences of D. A sequence is called a *sequential pattern* in a database D, if it is a maximal sequence with respect to the set of frequent sequences.

Finally, the sequential pattern mining problem may be stated in its entirety.

Definition 11 - Given a database D of sequences and some user-specified minimum support threshold, the sequential pattern mining process aims to discover the set of sequential patterns in D.

Sequential Pattern Mining over Nominal Temporal Data

When applied to nominal temporal data, the problem of sequential pattern mining can be restated, by redefining some of the previous definitions and introducing the formal definition of event, according to the notion introduced in Chapter 2 , section 1.

Definition 12 - An *event* is a pair $e=(a, t)$, where a is an itemset and t a positive integer, called the *timestamp*.

As before, the itemset represents the set of items transacted at a particular instant, in this case at instant t. The itemset of an event e is denoted by $e.set$ and the timestamp by $e.time$. Additionally, a sequence is now composed of events.

Definition 13 - An *event sequence*, or just a sequence, $s=<e_1e_2\ldots e_n>$ is an ordered list of events, whose timestamps respect the following order:

$$\forall i \in \mathbb{N}: 1 \leq i < n \Rightarrow e.time_i < e.time_{i+1}$$

The rest of the definitions remain valid when applied to temporal data.

Problem Analysis

What makes the problem of sequential pattern mining more challenging than *frequent itemset mining*? It is clear that frequent itemset mining is just a particular case of sequential pattern mining, since frequent itemsets are a particular case of sequential patterns – *1-sequential patterns*. Sequential pattern mining requires, in addition to the discovery of frequent itemsets, the arrangement of those itemsets in sequences and the discovery of which of those are frequent.

To understand why there exists a significant increase in the number of potential patterns, assume that there is a database to be mined with the minimum support threshold (the minimum number of sequences in the database that have to contain the pattern) set to σ and with $n = |\Sigma|$ different items in the item collection Σ. The goal of frequent itemset mining is to find which itemsets are frequent from the $|I|$ different possible existent itemsets, where I is the powerset of Σ (excluding the empty set), and its value is given by equation (1).

$$|I| = \sum_{j=1}^{n} \binom{n}{j} - 1 = 2^n - 1 \tag{1}$$

To understand the sequential pattern mining problem, let's begin by considering that the database has sequences with at most m itemsets and each itemset has at most one item. In these conditions, there would be n^m possible different sequences with m itemsets and

$$\sum_{k=1}^{m} n^k = \frac{n^{m+1}-1}{n-1} - 1 = \frac{n^{m+1}-n}{n-1} \tag{2}$$

different arbitrary length sequences.

Similarly, if each itemset has an arbitrary number of items, there would be S_m possible frequent sequences with m itemsets, with the value of S_m given by equation (3).

$$S_m = |I|^m = (2^n - 1)^m \tag{3}$$

In this case, there would be S sequences in general (equation 4).

$$S = \sum_{k=1}^{m} (2^n - 1)^k = \frac{(2^n-1)^{m+1}-1}{2^n-2} - 1 = \frac{(2^n-1)^{m+1}-2^n+1}{2^n-2} = \Theta(2^{nm}) \tag{4}$$

The number of different items and the average length of frequent sequences are tightly connected: a large number of items in a short sequence may imply a reduced number of frequent patterns, since the probability of the generality of items has to be small. In this way, algorithms will run efficiently. The opposite situation is a large sequence with a reduced number of items, where the probability of each element to occur in a large number of sequences is high. This leads to the

existence of many patterns, and consequently a large amount of processing time. The concept of database density, proposed here, quantifies this relationship.

Definition 14 - (*Density*) The *database density* (ρ) is the ratio between the number of existing frequent sequences (F) and the number of possible sequences (S).

$$\rho = \frac{F}{S} \approx \frac{F}{2^{nm}} \tag{5}$$

The database density depends on the support considered, since the number of frequent sequences (*F*) depends on the minimum support threshold accepted. The density is higher when the minimum support thresholds are low, since a larger number of frequent sequences exist.

One parameter that has impact on the density is the number of different items existent in the database (*n*). On one side, a large number of items allows for a potentially large number of different sequences. Since with many distinct sequences, their probability to be frequent is low, there will exist (all other conditions being equal) a smaller number of patterns and the database density will be low. On the other side, more reduced number of items generates a smaller number of potential sequences, which will be more frequent in the database, increasing the number of patterns and consequently the density of the database.

Despite the potentially large number of sequences (given by expression (4)), only a small fraction will be, in general, supported by the database. In particular, at most there only exist $|DB|/\sigma$ sequences of length *m* that are frequent.

Given the discrepancy between the number of different sequences and frequent ones, the difficulty of the data mining process resides in figuring out what sequences to try and then efficiently finding out which of those are frequent [Srikant 1996].

Like in frequent itemset mining, one of the most common techniques used to minimize the number of sequences that should be tried is the exploration of the anti-monotonicity property [Ng 1998].

Definition 15 - (*Anti-Monotonicity*) Given any set S, a constraint C is *anti-monotone* if and only if

$$\forall S': (S' \subseteq S \wedge S \text{ satisfies } C \Rightarrow S' \text{ satisfies } C).$$

In particular, the frequency of a sequence is an anti-monotone constraint, which means that a sequence cannot be frequent unless all its subsequences are frequent. The generality of sequential

pattern algorithms use this property to avoid testing the majority of possible sequences, minimizing their processing times. In general, they discover the 1-frequent patterns, then the 2-sequence patterns and so on, until there are no *k*-sequence patterns.

Interestingness Measures for Sequential Patterns

As noted before, to our knowledge and to date, no interestingness measure was proposed specifically for event sequences, except the notion of support, which can be stated by equation (6).

$$\sup(s) = \frac{|\{s' \in DB : s \subseteq s'\}|}{|DB|} \tag{6}$$

Other interestingness measures used in pattern mining are applied to association rules and not to the patterns themselves. In this manner, in order to use them in sequential pattern mining, a definition of a sequential rule is needed.

Definition 16 - A *sequential rule* is an association rule $S \Rightarrow T$, where S and T are sequences.

In this manner, a sequential rule is similar to an association rule, and the adaptation of the existent interestingness measures is possible. The *confidence* of the sequential rule $S \Rightarrow T$ can be defined as the conditional probability of the sequence T given S as in equation (7).

$$conf(S \Rightarrow T) = \frac{P(ST)}{P(S)} = \frac{\sup(ST)}{\sup(S)} \tag{7}$$

Similarly, the *lift* can be defined by equation (8).

$$lift(S \Rightarrow T) = \frac{P(ST)}{P(S) \times P(T)} = \frac{conf(S \Rightarrow T)}{\sup(T)} \tag{8}$$

As happens with association rules, the discovery of sequential rules can be achieved by discovering the sequential patterns and generating all possible sequential rules for each sequential pattern. For example, from the sequential pattern $S_1S_2S_3S_4$ it is possible to generate the sequential rules $S_1S_2S_3 \Rightarrow S_4$, $S_1S_2 \Rightarrow S_3S_4$ and $S_1 \Rightarrow S_2S_3S_4$.

1.2 – Constraints for Sequential Pattern Mining

Despite the reasonable efficiency of pattern mining algorithms, the lack of focus and user control have been factors that have impaired their generalized use, since the usually large number of

discovered patterns makes the analysis of novel information a difficult task.

In order to solve this problem, several authors proposed the use of constraints [Pei 2002a].

Definition 17 - A *constraint* is a predicate on the set of finite sequences.

In this manner,

Definition 18 - Given a database D of sequences, and some user-specified minimum support threshold σ and constraint C, a sequence is *frequent* if it is contained in at least σ sequences in the database and satisfies the constraint C.

This approach has been widely accepted by the data mining community, since it gives the user the possibility to control the mining process, either by introducing his background knowledge deeply into the process or by narrowing the scope of the discovered patterns. The use of constraints also reduces the search space, which contributes significantly to achieve better performance and scalability levels [Srikant 1997], [Pei 2002a], [Garofalakis 1999].

The simplest constraint is to impose that only some items are of interest – *item constraints*, permitting the reduction of the discovered patterns. Examples of such constraints are Boolean expressions over the presence or absence of items [Srikant 1997] and the naïve relaxation [Garofalakis 1999], as explained in section 1.4.

Another constraint involves the specification of the maximum distance between consecutive elements – *gap constraints*, either in temporal or non-temporal data. It consists on imposing a limit in the maximum distance between two consecutive elements in the sequence. This simple constraint is very useful to reflect the impact of some itemset on another one, in particular, when each transaction occurs at a particular instant of time. In this manner, it is possible to specify that an event has greater impact on near events than on distant ones. When using gap constraints, the notion of *contained in* has to be adapted.

Definition 19 - A sequence $a=<a_1a_2...a_n>$ is a δ-*distance subsequence* of $b=<b_1b_2...b_m>$ if there exist integers $1\leq i_1<i_2<\ldots<i_n\leq m$ such that $a_1\subseteq b_{i_1}$, $a_2\subseteq b_{i_2}$, …, $a_n\subseteq b_{i_n}$ and $i_k-i_{k-1}\leq\delta$. The sequence a is a *contiguous subsequence* of b if a is a 1-distance subsequence of b, i.e., the items of a can be mapped to a contiguous segment of b.

Note that a contiguous subsequence is a particular case of δ-distance subsequence and is the most restrictive notion of subsequence. A δ-distance subsequence s' of s is denoted by $s' \subseteq_{\delta} s$. Using δ=1 eliminates the possibility of having gaps between consecutive items. In the rest of this text this is designated by gap=0. When applied to event sequences, the gap may reflect the time interval allowed between consecutive events

More recently, regular languages have been proposed [Garofalakis 1999] and used to constrain the mining process [Zaki 2000], [Pei 2002b], reducing the number of discovered patterns.

Next, the concepts related to regular and context-free languages are introduced.

Regular Languages

In order to define the term regular language we make use of the notion of finite automaton, among others.

Definition 20 - A *Finite Automaton* is a tuple $\mathcal{A}=(Q, \Sigma, \delta, q_0, \mathcal{F})$, where: Q is a finite set of states; Σ is a finite set of symbols, called the *alphabet*; $q_0 \in Q$, designates the initial state; $\mathcal{F} \subseteq Q$, the set of *accepting* or *final states*; and δ is the *transition function* mapping $Q \times \Sigma$ to Q.

An automaton can be viewed as composed by a set of states and a set of transitions from state to state that occur on symbols chosen from an alphabet [Hopcroft 1979]. Finite automata are usually represented as directed graphs, where the vertices correspond to states and arcs to transitions, as illustrated in Figure 3.1.

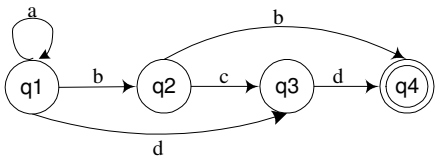


Figure 3.1 – A deterministic finite automaton

We can consider two kinds of finite automata: *Deterministic Finite Automata* (DFA), where there is one and only one transition for each pair (state, symbol), and *Nondeterministic Finite Automata* (NDFA), where it is possible to exist none, one or more than one transition for each pair (state, symbol). Any language accepted by any NDFA can also be accepted by an equivalent DFA.

A *language* is a set of finite sequences of symbols from some alphabet; those sequences are

usually called *strings*. A language is *accepted* by an automaton if it corresponds to the set of strings accepted by the automaton. A string is *accepted* by an automaton if its symbols define a path in the graph that represents the automaton, beginning in the initial state and reaching an accepting state. Examples of strings accepted by the DFA represented in Figure 3.1 are *bb* (beginning on state q_1 and going to state q_2 with the first *b* and then achieving state q_4 with the second *b*), *abb*, *add* and *aaabcd*; *a*, *aa* and *abca* are examples of non-accepted strings.

Definition 21 - A language is *regular* if it is accepted by a DFA.

When applied to sequential pattern mining, strings are replaced by sequences (as defined in the previous section), and symbols are replaced by itemsets. In this new context, the transition function maps $Q \times \mathcal{P}(\sum)$ to Q, with $\mathcal{P}(\sum)$ representing the powerset of $\sum$.

Context-Free Languages

Despite the fact that regular languages provide a simple and natural way to model sequential patterns, there are interesting patterns that these languages are not powerful enough to describe. Consider, for example, the following problem: a company wants to find out typical billing and payment patterns of its customers, but wants to restrict the search to customers, who have paid all their bills. This is equivalent to the requirement that an equal number of invoices and payments are registered. Additionally, it is known that an invoice always precedes a payment. In a semi-formal way, if an invoice is represented by an *a* and a payment by a *b*, the model has to be able to describe sequences that have the same number of *a*'s and *b*'s, and the number of *b*'s that have occurred at a given instant never exceeds the number of *a*'s that have occurred until that moment. Therefore, it has to be able to accept sequences like *abab* as well as *aabbab*, and reject strings like *aabbb* or *baab*.

Note that no finite automata can be used to model this constraint, because they have finite memory. Since regular languages are not expressive enough to describe some interesting constraints the simplest approach to this problem is to climb one-step up on the Chomsky hierarchy [Chomsky 1956].

Context-free languages (CFLs) are at the next level in this hierarchy, and are strictly more powerful than regular languages. This type of languages represents a formalism of great importance, since it is powerful enough to describe the structure of many interesting problems while remaining simple enough to allow for the construction of efficient parsers [Allen 1995].

CFLs have been widely used to represent programming languages and, more recently, to model biological sequences [Searls 1992] [Searls 1995].

In this thesis, we propose the use of context-free languages instead of regular languages for constraining the mining process, and for this reason, they are introduced next.

Unlike regular languages, context-free languages are not equivalent to the languages generated by deterministic finite automata. However, DFAs can be extended with a stack memory in order to recognize these new languages. These automata are known as *Pushdown Automata* (PDA).

Definition 22 - A *pushdown automaton* is a tuple $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \mathcal{F})$, where: Q is a finite *set of states*; Σ is an alphabet called the *input alphabet*; Γ is an alphabet called the *stack alphabet*; δ is a mapping from $Q \times \Sigma \cup \{\varepsilon\} \times \Gamma$ to finite subsets of $Q \times \Gamma^*$; $q_0 \in Q$ is the *initial state*; $Z_0 \in \Gamma$ is a particular stack symbol called the *start symbol*, and $\mathcal{F} \subseteq Q$ is the *set of final states* [Hopcroft 1979].

The language accepted by a pushdown automaton is the set of all inputs for which some sequence of moves causes the pushdown automaton to empty its stack and achieve a final state.

As an example, the following PDA accepts the language exemplified above:

$$M = (\{q_1, q_2\}, \{a, b\}, \{S, X\}, \delta, q_1, S, \{q_2\})$$

$$\delta(q_1, a, S) = \{(q_2, XS)\}, \delta(q_2, a, S) = \{(q_2, XS)\},$$

$$\delta(q_2, a, X) = \{(q_2, XX)\}, \delta(q_2, b, X) = \{(q_2, \varepsilon)\} \text{ and } \delta(q_2, \varepsilon, S) = \{(q_2, \varepsilon)\}$$

This automaton is represented in Figure 3.2.

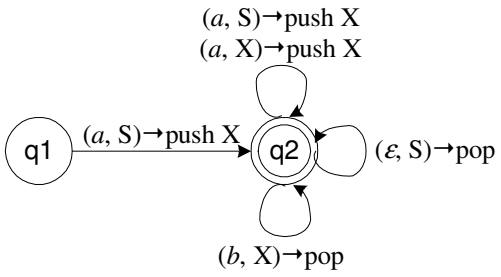


Figure 3.2 – A pushdown automaton

As an example, the "(*a*, S)→*push* X" transition label is used to indicate that when the input symbol is *a* and the top stack symbol is S, then X is pushed into the stack. When applied to sequences instead of strings, pushdown automata have to be redefined. In Chapter 5, section 3.1

these adaptations are described in detail.

Generalized Patterns

A complementary proposal to the use of constraints was based on the idea of generalized patterns. This proposal makes use of taxonomies, which capture the existent *is-a* relation between items, as introduced in Chapter 2, section 2.1. Like constraints, taxonomies facilitate the introduction of background knowledge into the mining process, by representing known relations between items (for example, the knowledge that two items belong to the same category).

Generalized patterns include items across different levels of the taxonomy, and their goal is to find rules about infrequent items at low abstraction levels, but that are frequent at higher-levels.

Since algorithms that find generalized patterns discover more patterns, their final step involves filtering redundant patterns. This is achieved by pruning all the patterns deemed non-interesting using an interestingness measure, which considers a pattern as redundant if it does not convey any more information that another pattern, and is less general than the second pattern. A detailed description of an algorithm to discover generalized patterns can be found in [Srikant 1995].

1.3 – Unconstrained Algorithms

There are two main approaches to the sequential pattern mining problem: *apriori-based* and *pattern-growth* methods. We are going to study these two different philosophies by analyzing their best-known implementations, *GSP* [Srikant 1996] and *PrefixSpan* [Pei 2001], respectively. There are several implementations of apriori-based methods. However, most of them assume some specific conditions (for example that the entire dataset fits in memory), allowing for significant improvements. Although *GSP* considers time constraints and uses taxonomies, if no taxonomy is provided, and time constraints are not set (min-gap assumes 0, max-gap assumes ∞ and the sliding window size is 0) both algorithms have similar goals: *to discover sequential patterns, without considering any constraints and without any database restrictions*.

Apriori-based Methods

Like the original approach to sequential pattern mining (*AprioriAll* [Agrawal 1995c]), *GSP* follows the candidate generation and test philosophy, used by the *apriori* algorithm (as described in Chapter 2 , section 2.2). It begins with the discovery of frequent *1*-sequences (k=1), and then generates the set of potentially frequent (k+1)-sequences from the set of frequent k-sequences, as

presented in Algorithm 1. Potentially frequent sequences are usually called *candidates*.

The generation of potentially frequent *k*-sequences (*k*-candidates) uses the frequent (*k*-1)-sequences discovered in the previous step, which may reduce significantly the number of sequences to consider at each moment, as shown in `candidateGeneration` pseudocode.

```
GSP (DB, min_sup, δ)
    L1 ← {frequent 1-sequences}
    for (k←2;Lk-1≠∅;k++) do
        Ck←candidateGeneration(Lk-1,k)
        Ck←candidatePruning(Lk-1,Ck,k,δ)
        Lk←supportPruning(Ck,DB,min_sup,δ)
return ∪k Lk
```

```
candidatePruning(Lk-1, Ck, k, δ)
    for each s∈Ck do
        if not isPossibleFrequent(s, Lk-1, k, δ)
            Ck←Ck\{s}
return Ck
```

```
isPossibleFrequent(s, Lk-1, k, δ)
return (~∃t⊂δs∧|t|=k-1 ∧ t∉Lk-1)
```

```
candidateGeneration(Lk-1, k)
    for each a∈Lk-1 do
        for each b∈Lk-1 do
            Ck←Ck∪join(a, b)
return Ck
```

```
supportPruning(Ck, DB, min_sup, δ)
    for each s∈DB do
        for each c∈Ck do
            if c⊂δ s then
                c.support ++
    Lk←{c∈Ck: c.support≥min_sup}
return L
```

Algorithm 1 – *GSP* pseudocode (without taxonomies and only with max_gap=δ)

In this procedure, any two frequent (*k-1*)-sequences, such that the proper maximal prefix of one is equal to the proper maximal suffix of the other, are combined to create a *k*-candidate by joining the last item of the second sequence to the first sequence, in accordance to the procedure `join` (Algorithm 2). In this pseudocode, the following notation was used: a_n represents the n^{th} itemset of the sequence *a* and a_i^n represents the n^{th} item of a_i; the symbol "\" denotes the difference operation among sets.

```
join (s, t)
    n ← |s|
    m ← |t|
    if n=1 ∧ m=1 ∧ (∀2≤i≤n: si=ti-1)
        u ← s1...snsm
    if n>1 ∧ m=1 ∧ (∀1≤i≤n-1: si=ti) ∧ s1\{s1^1}=t1 ∧ sn=tm\{tm^z}
        u ← s1...snsm
    if n>1 ∧ m>1 ∧ (∀2≤i≤n-1: si=ti) ∧ s1\{s1^1}=t1 ∧ z=|tm|∧sn=tm\{tm^z}
        u ← s1...sn-1tm
    if n=1 ∧ m>1 ∧ (∀2≤i≤n: si=ti-1) ∧ z=|tm| ∧ sn=tm\{tm^z}
        u ← s1t1...tm
return u
```

Algorithm 2 – Pseudocode for the *join* procedure

In order to illustrate the joining procedure consider three sequences: *x*=(*a*,*b*)*cd*, *y*=*bc*(*d*,*e*) and *z*=*c*(*d*,*e*,*f*). Joining x and y will create the sequence (*a*,*b*)*c*(*d*,*e*), and joining y and z will generate

c(*d*,*e*,*f*). Note that x and z cannot be joined.

Note that to decide if one sequence *s* is frequent or not, it is necessary to scan the entire database, verifying if *s* is contained in each sequence in the database. This check is performed by the procedure `supportPruning`.

In order to reduce the time needed to scan the database and count the support for each candidate, *GSP* adopts a particular scheme: it maintains all candidates in a hash-tree and for each sequence *s* in the database, it increments the counter of the candidates contained in *s*. In this manner, *GSP* only scans the database once per step.

Knowing that the frequency of a sequence is an anti-monotone property, *GSP* adopts two other optimizations. The first is that it only creates a new *k*-candidate when there are two frequent (*k*-1)-sequences with the prefix of one equal to the suffix of the other (`join` procedure). The second one is that, before counting their support, it eliminates all candidates that have some non-frequent maximal subsequence (`candidatePruning` procedure), in accordance with the anti-monotonicity property (verified by `isPossibleFrequent` procedure). By using these strategies to reduce the number of candidates, *GSP* reduces the time spent in scanning the database, increasing its general performance.

In general, apriori-based methods can be seen as breath-first traversal algorithms, since they construct all *k*-patterns simultaneously. Consider a database that is composed of sequences with items belonging to the set Σ={*a*, *b*}. If all possible arrangements of these two elements are frequent, *GSP* works as illustrated in Figure 3.3.

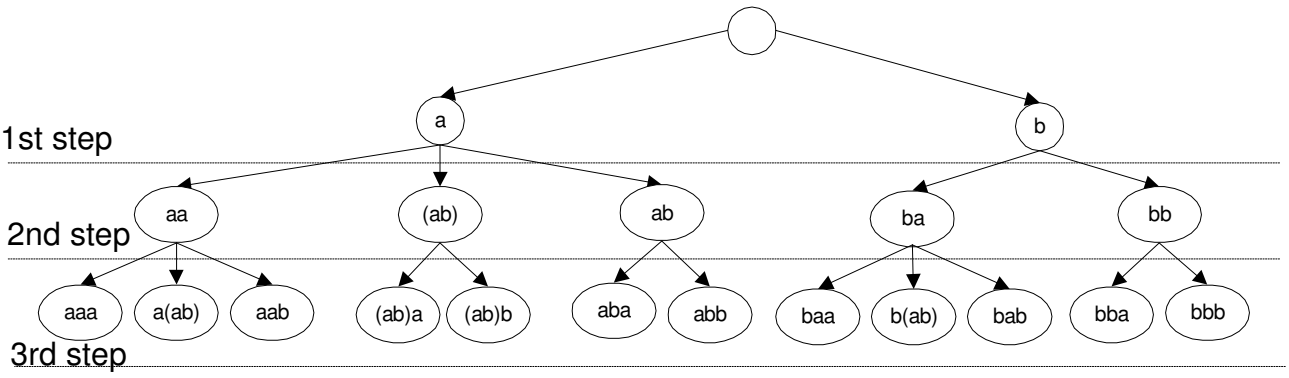


Figure 3.3 – Illustration of *GSP* behavior

At the end of the first iteration, *GSP* would have found *a* and *b* as frequent 1-sequences. At the end of the second step, it would have found {*aa*, (*ab*), *ab*, *ba*, *bb*}. Finally, at the end of the third iteration, *GSP* would have found all arrangements of 2 items taken 3 at a time, including the basket

(*ab*): {*aaa*, *a*(*ab*), *aab*, (*ab*)*a*, (*ab*)*b*, *aba*, *abb*, *baa*, *b*(*ab*), *bab*, *bba*, *bbb*}.

Note that, at each step *GSP* only maintains in memory the already discovered patterns and the *k*-candidates.

Pattern-growth Methods

Pattern-growth methods are a more recent approach to deal with the sequential pattern mining problem. The key idea is to avoid the candidate generation step altogether, and to focus the search on a restricted portion of the initial database.

PrefixSpan [Pei 2001] is the most representative of the pattern-growth methods and is based on the recursive construction of the patterns, while simultaneously restricting the search space to projected databases, as shown in Algorithm 3.

```
PrefixSpan (DB, σ)
return run(<>,0, σ, DB)

run (α, k,  σ, DB)
    f_list ← discoverFList(α,σ, DB);
    for each b∈f_list do
        α' ← αb;
        L← L ∪ α'
        L←L∪run(α',k+1, σ, createProjDB(α',DB))
return L
```

```
createProjDB (α, DB)
    for each s∈DB do
        if α ⊆ s
            n←s.firstOccurrenceAfter(α, 0)
            β←s. suffix (α, n)
            α–projDB← α–projDB ∪ {β}
return α–projDB
```

Algorithm 3 – PrefixSpan pseudocode

Its main loop consists on recursively constructing each sequential pattern by joining another frequent item to the already known frequent prefixes, as shown in procedure `run` (the `discoverFList` procedure finds the items that frequently appear after α).

An α-projected database is the set of subsequences in the database, which are suffixes of the sequences that have prefix α. Each α–projected database is constructed by registering which sequences on the database contain α (`createProjDB` procedure). At each step, the algorithm looks for the frequent sequences with prefix α, in the corresponding projected database. In this way, the search space is reduced at each step, allowing for better performance in the presence of small support thresholds, since the support counting operation is performed in specific portions of the original database.

In general, pattern-growth methods can be seen as depth-first traversal algorithms, since they construct each pattern separately, in a recursive way. If we consider the same database as previously, *PrefixSpan* would work as illustrated in Figure 3.4.

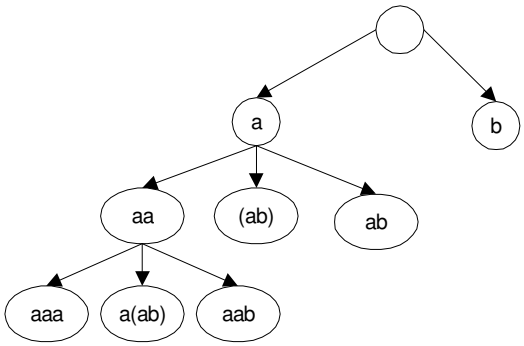


Figure 3.4 – Illustration of *PrefixSpan* behavior

At the first recursion step, *PrefixSpan*, like *GSP*, would have found *a* and *b* as frequent 1-sequences. Before encountering the first frequent sequences with three items, it would have only found {*aa*, (*ab*), *ab*} as frequent 2-sequences, since it expands only one branch at a time. Then, *PrefixSpan* would find {*aaa*, *a*(*ab*), *aab*}. Only after it has discovered the maximal patterns with *aa* as prefix, would it expand the (*ab*) branch in order to discover the patterns with (*ab*) as prefix. In this manner, *PrefixSpan* is able to deal efficiently with projected databases, since at each step it has only a few projected databases in memory.

PrefixSpan outperforms *GSP* in the generality of situations, with much better performances in the presence of very low minimum support thresholds. The reasons for this difference are not well understood and will be studied in Chapter 4.

1.4 – Algorithms for Sequential Pattern Mining with Constraints

From the constraints used in sequential pattern mining, the use of regular languages to restrict the discovery process is the one that has deserved more attention among the community. Again, there are essentially two approaches to this problem: an apriori-based – S*PIRIT*, and a pattern-growth method – *PrefixGrowth*.

SPIRIT

SPIRIT [Garofalakis 1999] is a family of apriori-based algorithms that uses a regular language (expressed as a finite automaton) and a gap constraint to restrict the mining process. Given that most of the interesting regular languages do not represent anti-monotone constraints, the algorithm uses constraint relaxations (weaker constraints) to manage the candidate generation and pruning procedures, as described in Algorithm 4.

```
SPIRIT (DB, min_sup, δ, C)
    C' ← relaxation( C)
    L← {frequent 1-sequences satisfying C '}
    k←2
    repeat
        Ck←candidateGeneration(L, k, C ')
        Ck←candidatePruning(L, Ck, k, C', δ)
        Lk←supportPruning(Ck, DB, min_sup)
        L ← L ∪Lk:
        k ← k+1
    until holdTerminatingCond(L, C ')
return {s∈L ∧ s satisfies C }
```

```
candidateGeneration(L, k, C ')
    for each a∈L do
        for each b∈L do
            s ← join(a, b)
            if (s satisfies C ')
                Ck←Ck∪s
return Ck
```

```
candidatePruning(L, Ck, k, C', δ)
    for each s∈Ck do
        if not isPossibleFrequent(s, L, k, C', δ)
            Ck←Ck\{s}
return Ck
```

```
isPossibleFrequent(s, L, k, C', δ)
return (~∃t⊂δs ∧ |t|=k-1 ∧ t satisfies C' ∧ t∉L)
```

```
supportPruning(Ck, DB, min_sup, δ)
    for each s∈DB do
        for each c∈Ck do
            if c⊂δ s then
                c.support ++
    Lk←{c∈Ck: c.support≥min_sup}
return L
```

Algorithm 4 – *SPIRIT* pseudocode

In this manner, the great differences between *SPIRIT* and *GSP* reside on the generation of candidates that have to be accepted by the constraint relaxation (`candidateGeneration` procedure) and on the candidate pruning step, which only accepts sequences that have all maximal accepted subsequences frequent (`isPossibleFrequent` procedure). (The `join` procedure remains identical to the one presented on Algorithm 2).

There are two other differences: the terminating condition on the main loop (`holdTerminatingCond`) and the use of the set of all frequent sequences (`L`) instead of the set of frequent (*k*-1)-sequences (L_{k-1}) as parameters to the candidate generation. Both differences are due to the fact that even the proposed constraint relaxations may not be anti-monotone, and different relaxations impose different terminating conditions and candidate generation procedures.

Four relaxations, representing a natural progression on imposing stronger relaxations in the mining process and the corresponding SPIRIT implementations, have been proposed: *naïve* – SPIRIT(N); *legal* – SPIRIT(L); *valid* – SPIRIT(V); and *the regular language itself* – SPIRIT(R).

Naïve relaxation only prunes candidate sequences having elements that do not belong to the constraint alphabet (the alphabet of the finite automaton), which simply corresponds to an item constraint. Since naïve relaxation is anti-monotonic, the terminating condition and the candidate generation procedures are equal to the corresponding ones in *GSP*. The process ends when there is no frequent (*k*-1)-sequence and the candidate generation procedure uses L_{k-1} elements to generate the *k*-candidates (C_k).

The *legal* relaxation requires that every pattern is legal with respect to some state of the automaton. A sequence is said to *be legal with respect to* q (with q being a state of the automaton) if there is a path in the automaton, which begins in state q and is composed by the sequence elements. The process ends when there are no legal k-sequences with respect to the initial state, and the candidate generation step remains similar to the corresponding one in *GSP*.

The *valid* relaxation only accepts sequences that are valid with respect to any state of the automaton. A sequence is said to *be valid with r*espect to q if it is legal with respect to q and reaches an accepting state. Given that this relaxation does not impose an anti-monotone constraint, the candidate generation step has to be different: each frequent *k*-sequence is extended (at the beginning) with a frequent element (an element that belongs to L_1). The terminating condition remains equal to the corresponding one in *GSP*. Note that the term *valid* was somewhat misused, since the relaxation considers suffixes of accepted sequences. In the rest of this dissertation, this relaxation is called *valid-suffix*.

Finally, the regular language itself imposes the most different terminating condition and candidate generation procedure. Consider, for example, the automaton in Figure 3.1. Despite the fact that *a*, *ab* and *abc* are not accepted, *abcd* is accepted. This show that it is possible to have accepted sequences whose prefixes are not accepted. In order to deal with this situation, the process only ends when it achieves the n^{th} iteration (where *n* is the number of states in the automaton) and there is no frequent sequence (all L_k are empty). Additionally, at each step, the candidate generation procedure creates each candidate by enumerating all possible k-sequences accepted by the automaton.

PrefixGrowth

Like *SPIRIT*, *PrefixGrowth* [Pei 2002b] uses regular languages, expressed as finite automata, to constrain the mining process.

```
PrefixGrowth (DB, σ, C)
return run(<>,0, σ, DB, C)

createProjDB (α, DB)
    for each s∈DB do
        if α ⊆δ s then
            n←s.firstOccurrenceAfter(α, 0)
            β←s.suffix (α, n)
            α-projDB← α-projDB ∪ {β}
return α-projDB
```

```
run (α, k, σ, DB, C)
    f_list ← discoverFList(α, σ, DB, C)
    for each b∈f_list do
        α' ← αb;
        if α' potentially satisfies C
            if α' satisfies C
                L← L ∪ α'
            L←L∪run(α',k+1,σ,createProjDB(α',DB),C)
return L
```

Algorithm 5 – *PrefixGrowth* pseudocode

It is a pattern-growth method, similar to *PrefixSpan,* but it only generates sequences that potentially satisfy the constraint, this is, it only extends sequences that are prefixes of accepted sequences. Algorithm 5 presents the pseudocode for *PrefixGrowth*. Another difference to *PrefixSpan* is that the `discoverFList` procedure in *PrefixGrowth* also removes the sequences that do not contain any accepted subsequence from the database.

Note that *PrefixGrowth* deals with non-monotonic constraints in an easier way than SPIRIT. Indeed, it only requires that relaxed constraints are prefix-monotone. A constraint is *prefix-monotone* if it is either prefix anti-monotonic or prefix-monotonic.

Definition 23 - (*Prefix anti-monotonicity*) A constraint is *prefix anti-monotonic* if for each sequence α satisfying the constraint, so does every prefix of α[Pei 2002b].

C is *prefix anti-monotonic* $\Leftrightarrow \forall \beta$: β is prefix of $\alpha \wedge \alpha$ satisfies C $\Rightarrow \beta$ satisfies C)

Definition 24 - (*Prefix monotonicity*) A constraint is *prefix monotonic* if for each sequence α satisfying the constraint, so does every sequence having α as a prefix [Pei 2002b].

C is *prefix-monotonic* $\Leftrightarrow (\forall \alpha$: α satisfies *C* $\Rightarrow (\forall \beta$: α is prefix of $\beta \Rightarrow \beta$ satisfies *C*))

An interesting issue is to note that neither PrefixSpan nor *PrefixGrowth* are able to deal with gap constraints without some adaptations. Moreover, the differences on the performance of *PrefixSpan* and *GSP* may be explained by this difference on goals. In order to clarify this issue, in the next chapter we will propose a new pattern-growth method able to deal with gap constraints (*GenPrefixSpan*), and we will compare the performance of this new algorithm with the performance of *GSP*.

1.5 – Review of Related Work

The recent interest in performing inter-transactional analysis has lead to the definition of the sequential pattern mining problem and the development of new algorithms to address it. However, like for pattern mining, those algorithms "produce" a large number of patterns, most of them useless and uninteresting for the final user.

In order to focus the discovery on user expectations, some authors have proposed the use of constraints. On one hand, they introduce some difficulties in the mining process, and, as been shown, it is necessary to use some constraint relaxations (with specific properties) to maintain the efficiency of the algorithms. On the other hand, constraints have rarely been applied and have not

been able to represent the user background knowledge, in an integrated form.

Another important issue related with the use of constraints is the inability to discover unknown information. Some authors have pointed out that when used incautiously, constrained pattern mining may reduce the data mining process to a hypothesis-testing task. Note that, when blindly applied on the first step of the process, it prevents the discovery of unknown and unexpected patterns, which is the first and foremost goal of data mining [Hipp 2002].

2 – Thesis Statement

We may formulate the central statement of the present thesis as:

> **It is possible to efficiently use constrained sequential pattern mining algorithms over nominal temporal data to discover unknown information, keeping the process centered on the user.**

An analysis of the thesis statement leads to six questions that need further discussion:

- *What does "unknown information" mean?* Information is used as a synonymous of pattern and the term unknown designates both the novel information in the reference frame of the information system or of the user himself [Frawley 1992].
- *What does "to center the process in the user" mean?* Centering the process in the user has essentially two aspects: the management of user expectations and the use of user background knowledge in the mining process [Brachman 1996]. By expectation management, we mean that the results of the process have to be in accordance with user expectations. This management is done by constraining the discovery process using his background knowledge.
- *How is the discovery performed?* The discovery is performed by sequential pattern mining algorithms (described in detail in Chapter 4 and Chapter 5).
- *Which constraints may be used?* As described in the above section, the problem of pattern mining implicitly uses an existential constraint – the minimum support threshold, to identify which are the frequent patterns. Additionally, other kinds of constraints have been used to focus the discovery process, such as temporal constraints, taxonomies [Srikant 1995] and formal grammars [Garofalakis 1999]. Despite the more recent efforts to study these constraints, they have been used in a non-integrated way. This thesis proposes a new set of constraints – Ω-constraints, which combine the most interesting currently existent constraints

with new ones, in an integrated way. These constraints are defined in Chapter 5, in detail.

- *How is the discovery of unknown patterns possible?* The use of constraints helps on focusing the search in accordance with user expectations, but prevents the discovery of unknown patterns. In this thesis, we propose the use of constraint relaxations, instead of constraints by themselves, for guiding the mining process. This approach will be proposed in Chapter 6, where several relaxations are defined.
- *What does efficiently mean*? Sequential pattern mining algorithms show an acceptable performance exploring large sparse datasets. However, in the presence of dense datasets or very low support thresholds, their performance suffers a considerable degradation. In this work, we argue that the use of constraints and constraint relaxations improve the performance of those processes. In this manner, *efficiently* means that in the majority of situations, constrained sequential pattern mining can be performed in less time than unconstrained sequential pattern mining.

The validation of this thesis will be performed by comparing the number and interest of discovered patterns, using unconstrained and constrained sequential pattern mining, with constraints and with constraint relaxations. These evaluations will be conducted using synthetic data (presented at the end of each chapter) and two real-life datasets, presented in Chapter 7.

Summary

In this chapter, we have presented a detailed description of sequential pattern mining and the proposed solutions to it. Additionally, we have presented our thesis statement, which corresponds to a new data mining methodology. This methodology relies on the notion of constraint relaxation, and aims to keep the focus on user expectations allowing for the expression of existing background knowledge. This is done without compromising the main goal of data mining: the discovery of unknown information. With the conjunction of a new mining process and the use of constraint relaxations, the user is able to choose the level of knowledge that he wishes to incorporate on the process.

Chapter 4
Sequential Pattern Mining: new algorithms

In this chapter, PrefixSpan is generalized in order to deal with gap constraints – GenPrefixSpan. A comparison between GSP and GenPrefixSpan is performed and the results presented. Section 3 presents a new apriori-based algorithm – SPaRSe, which combines the simplicity of apriori-based algorithms with the efficiency of pattern-growth methods. With this new algorithm, we show that with some specific improvements, apriori-based algorithms can compete with pattern-growth methods. Studies on performance, scalability and memory requirements in synthetic datasets are presented in the last section of this chapter.

The rapid growth of the amount of stored digital data and the recent developments in data mining techniques, have lead to an increased interest in methods for the exploration of data, creating a set of new data mining problems and solutions. *Frequent Structured Mining* is one of these problems, whose target is to discover hidden structured patterns in large databases. Sequences are the simplest form of structured patterns, while trees and graphs are other examples of structured patterns.

The main approaches to sequential pattern mining, namely apriori-based and pattern-growth methods, are being used as the basis for other structured pattern mining algorithms. However, and despite the fact that pattern-growth algorithms have shown better performance in the generality of situations, its advantages over apriori-based methods are not sufficiently understood.

In this chapter, we present a comprehensive comparison of these approaches to sequential pattern mining, in order to explain the main reasons why pattern-growth methods outperform apriori-based approaches. However, a fair evaluation of the methods requires that they have exactly the same goals, which is not true for the best-known algorithms, *GSP* and *PrefixSpan*. In order to

accomplish our goal, we present a generalization of *PrefixSpan* (*GenPrefixSpan*) that deals with gap constraints, and maintains the pattern-growth philosophy. Finally, we analyze the conditions under which apriori-based methods become as efficient as pattern-growth methods, presenting a new apriori-based algorithm – *SPaRSe* (Sequential PAttern mining with Restricted SEarch).

1 – Pattern-Growth Methods with Gap Constraints

In Chapter 3 , section 1, we have described the two well-known algorithms for sequential pattern mining. In this section, we will study the effectiveness of these algorithms when dealing with gap constraints.

As referred, *GSP* is a generalization of *AprioriAll* that deals with gap constraints. However when gap constraints are used, the *PrefixSpan* algorithm (and also PrefixGrowth) cannot be applied directly. To illustrate this limitation, consider for example the data in Table 1 and a minimum support threshold of 40%, which means, in this case, that a pattern has to occur at least twice in the database. Additionally, assume that the gap constraint is set to zero, which means that only contiguous sequences are allowed.

Table 1 – Database example

Database				
pumpu	*acjcde*	*ababa*	*achcde*	*nozrs*

PrefixSpan finds *a*, *c*, *d* and *e* as frequent items, which constitute the *f_list*. Then, it recursively calls the main procedure (`run`) with α=*a* and an α-projected database equal to {*cjcde*, *baba*, *chcde*}. Next it recursively proceeds with α=*ac* and an α-projected database equal to {*jcde*, *hcde*}, and it finishes this branch. Similarly for element *c*: `run` is called with α=*c* and an α-projected database equal to {*jcde*, *hcde*}. Since there is no frequent element at distance 1, the search stops and *cde* is not discovered. This happens because the α-projected database only maintains the suffix after the first occurrence of α (procedure `createProjDB` in Algorithm 3).

1.1 – *GenPrefixSpan*

The generalization we propose for *PrefixSpan* (*GenPrefixSpan*) is based on the redefinition of the method used to construct projected databases. Instead of looking only for the first occurrence of α, every occurrence is considered. For instance, again using the example in Table 1, the creation of the *c*-projected database would give as result {*jcde*, *de*, *hcde, de*} instead of {*jcde*, *hcde*} as before.

It is important to note that, including all suffixes after the occurrence of an element may change the number of times that each pattern appears, thus modifying the database size. In order to be able to properly compute the support for each pattern, an *id* is associated with each original sequence. In this manner, the support counting procedure ensures that each original sequence counts at most once for each pattern support. For instance, for the same example the *a*-projected database would be {1-*cjcde*, 2-*baba*, 2-*ba*, 3-*chcde*}, but the support for *ab* would remain one, since *baba* and *ba* correspond to the same original sequence.

Given that this new method requires that the α-projected database considers all the occurrences of α, the database considered at each step may be considerably larger than the previous one. This means that the search space is no longer smaller at each step, as in the *PrefixSpan* algorithm [Pei 2001].

```
GenPrefixSpan (DB, σ, δ)
  f_list ← discoverFrequentItems (σ, DB)
  for each b∈f_list do
    L← L ∪ {b}
    L←L∪run(b,δ,σ,genProjDB(b,δ,DB,true))
return L
```

```
run (α, k, σ, δ, DB)
  f_list ← discoverFrequentItems (σ, δ, DB)
  for each b∈f_list do
    α' ← αb;
    L← L ∪ α'
    L←L∪run(α',k+1,σ,δ,genProjDB(α',δ,DB,false))
return L
```

```
genProjDB(α, δ, DB, firstStep)
    for each s∈DB do
        i←0
        repeat
            i← s.firstOccurrenceAfter(α, i)
            β←s.suffix(α, i)
            α-projDB←α-projDB ∪ {β}
        until (i + δ≥|s|) ∧ (firstStep ∨ i≥δ)
return α-projDB
```

Algorithm 6 – *GenPrefixSpan* pseudocode

Fortunately, the projected database creation can be improved for the subsequent steps. In fact, for the remaining steps, in order to create αb-projected databases (with b a frequent item after α) it is only necessary to look for the occurrences of b, whose distance to α is less than the maximum distance allowed (δ+1), ensuring that αb is frequent in accordance with the established gap constraint (as shown in the new `createProjDB` procedure in Algorithm 6). Therefore, the design of *GenPrefixSpan* remains similar to the original *PrefixSpan*, as shown in Algorithm 6, and the generation of the projected databases can be done with any of the proposed methods. All the presented results on the final section use the pseudo-projection to generate the projected databases.

The difference between the use of `createProjDB` in the first and remaining steps resides in the `firstStep` flag, which ensures that in the first step all occurrences of α are considered and that, for the remaining steps, only the occurrences of α at a legal distance are considered. Note that when there is no gap constraint, the creation of projected databases (`createProjDB`(b,∞,DB,true)) is similar to the corresponding procedure defined in original *PrefixSpan*, since it only generates the projection relative to the first occurrence of α. In this manner, the performance of *GenPrefixSpan* and *PrefixSpan* are similar in the absence of gap constraints.

Another interesting issue is that when only contiguous patterns are allowed, the procedure `createProjDB` only looks for the first itemset after α, which increases its performance.

2 – Comparison between *GSP* and *GenPrefixSpan*

In order to understand and identify what are the most time consuming operations of each algorithm, we have performed a profiling study, recording the total time spent by the main steps of each algorithm. Both *GSP* and *GenPrefixSpan* were executed in a set of synthetic datasets (described in detail in section 4) with several different values for minimum support threshold.

As other pattern-growth methods, *GenPrefixSpan* generally outperforms *GSP*, and shows much better results for low minimum support threshold values. In order to understand why this happens, let us analyze the time spent in each step of *GSP* when using low minimum support values.

We have considered the two main steps of *GSP*: candidate generation and candidate test. Candidate generation includes `generateL1`, which corresponds to the initial step, where frequent 1-sequences are discovered; `candidateGeneration` – the procedure that defines the sequences potentially frequent and `candidatePruning` – the procedure that eliminates some of the candidates, in accordance to the anti-monotonicity property.

The step of candidate testing corresponds to `supportPruning`, the procedure that counts the support for each potentially frequent sequence, by performing the database scan.

Table 2 shows that the support-based pruning procedure consumes almost the totality of processing time. For *GenPrefixSpan*, the relative results are quite different: the processing time spent in scanning the database is approximately 50% (Table 2), and uses much less time than *GSP* for low minimum support values.

Since both methods spend a large percentage of time scanning the database, what makes *GenPrefixSpan* much faster than *GSP*? The answer lies in the reduction of the search space. In fact, at each recursion step, *GenPrefixSpan* usually scans a smaller database, since the α-projected database has more sequences than the αb-projected database.

Table 2 – Processing Times for *GSP* and *GenPrefixSpan*

	GSP					GenPrefixSpan				
sup	CandidateGeneration		Candidate Test		Total	Find Elements		Create Proj DB		Total
50%	0,01s	0%	3,19s	100%	3,19s	0,88s	50%	0,89s	50%	1,78s
40%	0,01s	0%	4,90s	100%	4,91s	1,33s	53%	1,16s	47%	2,49s
33%	0,02s	0%	9,08s	100%	9,10s	2,01s	55%	1,67s	45%	3,68s
25%	0,02s	0%	17,32s	100%	17,34s	3,40s	55%	2,74s	45%	6,15s
10%	1,26s	1%	157,63s	99%	158,89s	18,71s	58%	13,41s	42%	32,13s

3 – *SPaRSe* – Sequential Pattern Mining with Restricted Search

The results of this analysis lead us to analyze the possibility of applying a search restriction to apriori-based methods. In this section, we present a new algorithm, which combines the candidate generation and test philosophy with the restriction of the search space obtained from the use of projected databases.

SPaRSe (*Sequential Pattern mining with Restricted Search*) is a new algorithm obtained by adapting *GSP* to use restricted databases. It acts iteratively like apriori-based algorithms, in that after discovering the frequent elements, it looks for patterns with growing length at each step. It finishes when there are no more potential frequent patterns to search. The key idea is to maintain a list of supporting sequences for each candidate, and to verify the existence of quorum only in the subset of sequences that support both generating candidates.

Algorithm 7 describes the main procedure of *SPaRSe*. Note that its main procedure is identical to the main procedure of *GSP*, since the `patternDiscovery` procedure aggregates the functionalities of `candidateGeneration`, `candidatePruning` and `supportPruning` in *GSP*.

The difference to *GSP* is the fact that *SPaRSe* generates and tests each candidate separately. Procedure `satisfies` counts the support for one candidate and returns `true` if it is frequent and `false` otherwise. This behavior is similar to the behavior of `supportPruning` in *GSP*. (The `join` procedure remains identical to the one presented on Algorithm 2).

```
SPaRSe (DB, min_sup, δ)
   L1 ← {frequent 1-sequences}
   for (k←2;Lk-1≠∅;k++) do
      patternDiscovery(Lk-1,DB,min_sup,δ)
      k ← k+1
return ∪k Lk
```

```
patternDiscovery(Lk-1,DB,min_sup,δ)
   Lk ← ∅
   for each s∈Lk-1 do
      for each t∈Lk-1 do
         c←createNewCandidate(s, t)
         if (satisfies(c,min_sup,δ) ∧
            isPossibleFrequent(c.sequence,
                                 Lk-1,δ))
            Lk←Lk∪{c}
return Lk
```

```
createNewCandidate(a, b)
   c←join(a.sequence, b.sequence)
   c.supDB←c1.supDB ∩ c2.supDB
return c
```

```
isPossibleFrequent(s, Lk-1, k, δ)
return (~∃t⊆δs∧|t|=k-1∧t∉Lk-1)
```

```
satisfies(c, min_sup, δ)
   s← c.sequence
   nr←0
   for each t∈c.supDB do
      if (s⊆δt nr←nr+1)
return (nr≥sup)
```

Algorithm 7 – *SPaRSe* pseudocode

3.1 – Support-Based Pruning

However, what makes *SPaRSe* more than a variant of *GSP*, is the restriction of the search space in a way similar to *PrefixSpan*: it associates each frequent discovered pattern with the set of sequences where it appears. This set is called the *support database*. In this manner, it is possible to count the support of a new candidate, only in the intersection of the support databases of its parents. Note that the anti-monotonicity property implies that if a sequence does not support a pattern, then it could not support any of its super-patterns. When the support of a candidate is counted, only the potential support sequences are scanned.

In *SPaRSe* a pattern is not only a sequence in itself, but it contains the information that lists the sequences where it occurs, which correspond to its support database. This simple modification justifies the new procedure for generating candidates – `createNewCandidate`. This simple inclusion allows for constraining the search considerably, improving the global average performance. However, maintaining support databases for each discovered pattern, and for every candidate of length *k*, may be prohibitive in terms of memory.

A simple contribution to minimize this problem is to use an array of bits to represent the support database. Note that, in *GenPrefixSpan* an α-projected database uses more memory (since it also keeps the sequence identification and the index of the α occurrence). However,

GenPrefixSpan is a depth-first traversal algorithm, which avoids having all projected databases in memory at the same time.

In the case of breath-first traversal algorithms, as *SPaRSe*, the solution may be to redesign the candidate generation and test procedure: instead of generating all candidates at once and then testing them, it is possible to generate and test each one alone, minimizing the memory requirements, which explains the design of the `patternDiscovery` procedure.

Note that with this change, it does not make sense to use sophisticated data structures, as hash-trees, to count the support for each candidate. Usually, apriori-based algorithms use hash-trees to store all candidates, and scan the database once to count the support for all candidates. Generating and testing each candidate alone invalidates this strategy.

However, for very low support thresholds *SPaRSe* does not work better than *GenPrefixSpan*, spending long times in the candidate generation and pruning steps (as shown in Figure 4.1).

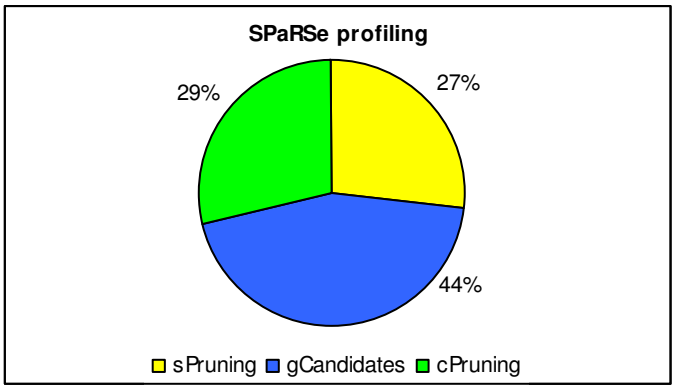


Figure 4.1 – *SPaRSe* profiling for very low support thresholds

3.2 – Candidate Generation

Remember that apriori-based methods generate *k*-sequence candidates by joining two (*k*-1)-patterns, when the prefix of one is equal to the suffix of the other (`join` procedure). This operation may consume a considerable amount of time when there are many frequent patterns. This happens, since for every pattern it is necessary to verify which patterns have a prefix equal to its suffix.

To improve the generation of *k*-candidates, *SPaRSe* stores all (*k*-1)-patterns in a hash-tree. Figure 4.2 exemplifies this data structure when storing the different combinations of two elements.

When the number of items is large and the database is sparse it is useful to use the same leave to store sequences with different prefixes, justifying the use of a hash-tree instead of a suffix-tree.

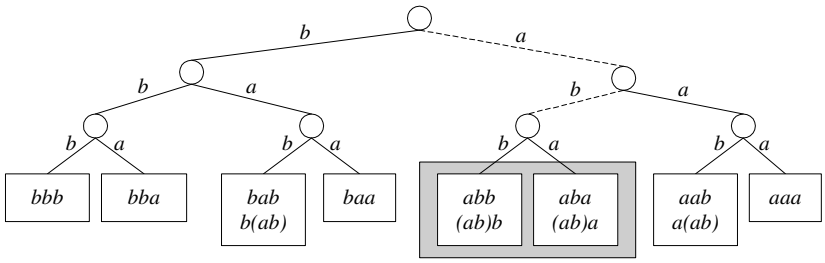


Figure 4.2 – Hash-tree used in candidate generation

In order to generate a new candidate with length *k*, *SPaRSe* does not need to look for every other candidate. The algorithm takes the suffix of the candidate and follows its path in the hash-tree. The reached sub-tree contains the sequences that may match with *s* to generate a new candidate. Now it is only necessary to verify if they really match with *s*, and then generate new candidates.

Consider for example the sequence *a*(*ab*): following the path of its suffix (*ab*) in the hash-tree, we discover the sequences that may match with it – *abb*, (*ab*)*b*, *aba* and (*ab*)*a*. Note that only (*ab*)*b* and (*ab*)*a* are really appropriate to join with it and generate two new candidates: *a*(*ab*)*b* and *a*(*ab*)*a*. Figure 4.2 shows the followed path with a doted line, and possible matching sequences in a shadowed box.

By avoiding testing if any two patterns match, *SPaRSe* improves its performance by about 50%, for low support thresholds.

3.3 – Candidate Pruning

We have also considered another improvement, the use of a hash-tree to implement candidate pruning.

The key idea of candidate pruning is to eliminate candidates that cannot be frequent, as stated before. However, verifying if every maximal subsequence is frequent for every candidate may be prohibitive, especially when low support thresholds are used.

Like candidate generation, this procedure may use a hash-tree to identify the potential frequent patterns. Consider the hash-tree in Figure 4.3 and the candidate (*ab*)*aa.* It has three maximal subsequences *aaa*, *baa* and (*ab*)*a*. Although the first and second ones are frequent patterns (presented in shadowed boxes), the third one is not, since it is not stored in the hash-tree. Looking for the path of each subsequence in the hash-tree reduces significantly the time needed to make this discovery.

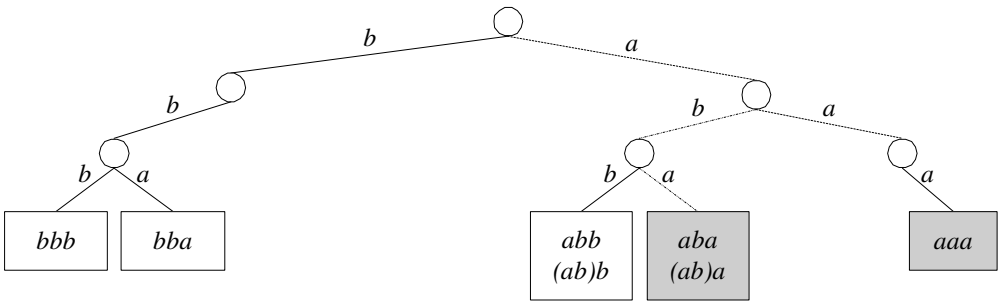


Figure 4.3 – Candidate pruning example

In summary, *SPaRSe* is an apriori-based algorithm, which follows the candidate generation and test philosophy. It has three fundamental differences to *GSP*:

- it generates and tests one candidate at a time;
- it uses *support databases* to count the support for each candidate;
- it uses a hash-tree to store frequent patterns.

These improvements directly contribute to accelerate the candidate generation and pruning procedures. In the next section, it is shown how *SPaRSe* and *GenPrefixSpan* algorithms deal better with datasets of different characteristics, and that either one of them may represent the best choice for a particular application.

4 – Experimental Results

The comparison of sequential pattern mining algorithms over a large range of data characteristics, such as different support thresholds, dataset sizes and sequence lengths, has been done by several authors (see for instance [Agrawal 1995c], [Srikant 1996], [Zaki 1998a], [Pei 2001] or [Ayres 2002]). However, as stated in Chapter 3, section 1.1, the results depend on the dataset density, and to our best knowledge, there has been no study about the performance of sequential pattern mining algorithms in dense datasets.

Our goal in this section is to understand the impact of those characteristics in the performance of the algorithms. In order to do that, we compare the performance of *GenPrefixSpan*, *SPaRSe*, and *GSP*, over several distinct synthetic datasets, considering all of the enumerated characteristics. The performance of *GSP* only serves as a reference line to the performance of the other two algorithms, since the execution times are generally much larger. Neither *PrefixSpan* nor *SPAM* [Ayres 2002] could be used, since they do not deal with gap constraints.

The basic methods used by the different algorithms were the same (basic operations on

sequences, support verification, etc) which means that the comparisons are meaningful and repeatable. The *GSP* algorithm follows the original description [Srikant 1996], and *GenPrefixSpan* and *SPaRSe* the descriptions in this chapter. The datasets were maintained in main memory during the processing, avoiding hard disk accesses.

To perform the study over a large range of different characteristics, we used the standard synthetic data set generator from IBM Almaden (described in Appendix B). The datasets used in these experiments were generated maintaining all, except one, of the parameters fixed, and exploring different values for the remaining parameter. In general, the datasets contain 10.000 sequences (Parameter D of the generator set to 10), with 10 transactions each on the average (C=10). Each transaction has on the average 2 items (T=2). The average length of maximal patterns is set to 4 (S=4) and maximal frequent transactions set to 2 (I=2). These values were chosen in order to follow closely the parameters usually chosen in other studies. The values for different sequential patterns (Ns) and transactional patterns (Ni) were also chosen similarly, set to 5.000 and 10.000, respectively.

The next subsections present the performance results achieved using datasets with different densities, followed by the studies on different support thresholds and different gap values. The section finishes with the scalability studies.

Performance with different densities

The behavior of both algorithms is somehow different for different levels of density. As can be observed in Figure 4.4, *GenPrefixSpan* achieves better results for sparse datasets, but shows performances similar to the ones shown by *SPaRSe* for dense datasets.

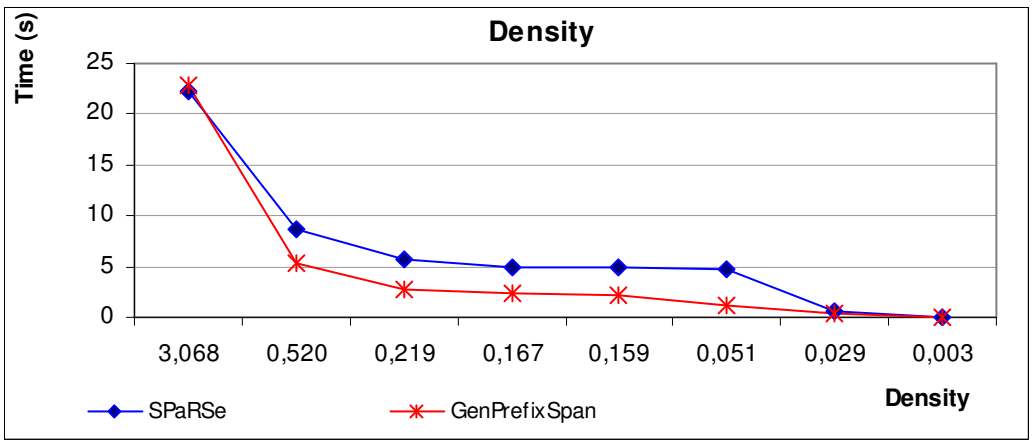


Figure 4.4 – Performance with different dataset densities (support set to 10%)

The main reason for this difference is that *SPaRSe* does not waste so much time generating

infrequent candidates for dense datasets. Since there are more patterns, both algorithms have to generate a similar number of sequences, reducing the difference between their processing times.

The different values for density were achieved by varying the number of different items in the dataset from ten to one thousand (N∈{10, 20, 30, 40, 50, 100, 500 and 1.000}).

We have also studied the memory requirements. As stated before, GenPrefixSpan requires more memory than SPaRSe, since it has to maintain multiple indexes for the same sequence and the corresponding pattern position for each occurrence.

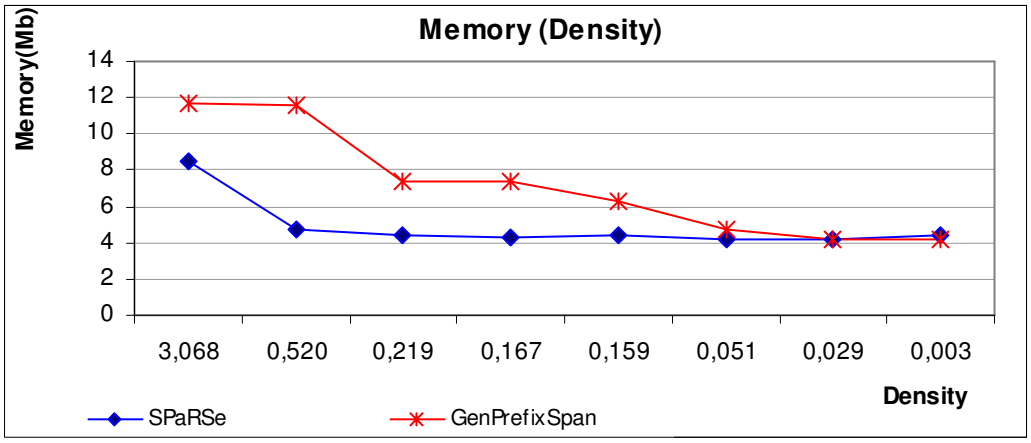


Figure 4.5 – Memory requirements in the presence of different dataset densities

Figure 4.5 shows that both algorithms require more memory when processing dense datasets, since the number of patterns is higher.

Performance with different support thresholds

For different minimum support thresholds, the results are consistent. *SPaRSe* is comparable to *GenPrefixSpan* in dense situations (Figure 4.6),

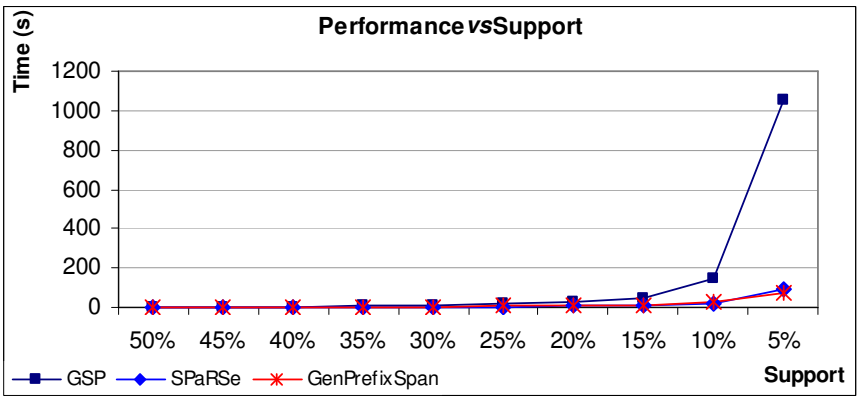


Figure 4.6 – Performance with different support thresholds in dense datasets

and shows worst results for sparse datasets (Figure 4.7).

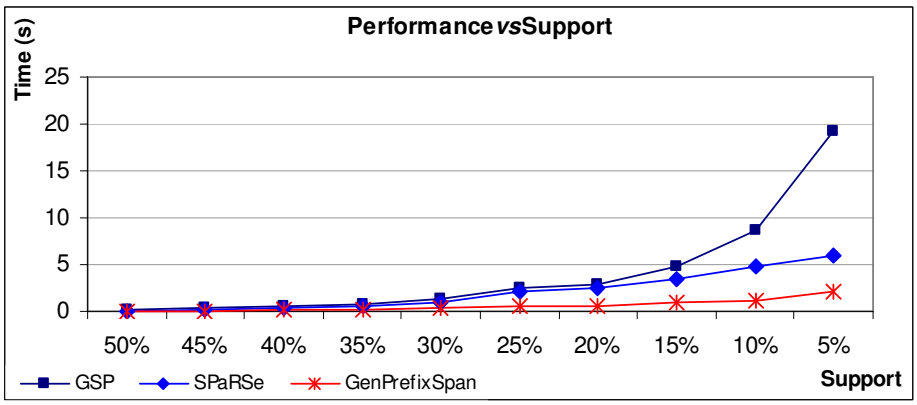


Figure 4.7 – Performance with different support thresholds for sparse datasets

It is interesting to note that the execution times in sparse datasets are about ten times faster than in dense datasets, for all the algorithms.

These results show that a great part of the efficiency of *GenPrefixSpan* is due to its memory usage. In several other experiments, conducted in machines with less available memory, the results were slightly different, with *GenPrefixSpan* showing worst results than *SPaRSe* for dense datasets. However, in the presence of machines with less memory (say 250Mb) the results are again worst for *SPaRSe*. In fact, since *GenPrefixSpan* works in a depth-first manner, it is able to manage hard-disks access in a more efficient way.

Performance with different gap constraints

When comparing the algorithms for different gap constraints, the results are considerably different (Figure 4.8). *GenPrefixSpan* is considerably worst than *SPaRSe* for less restrictive gaps.

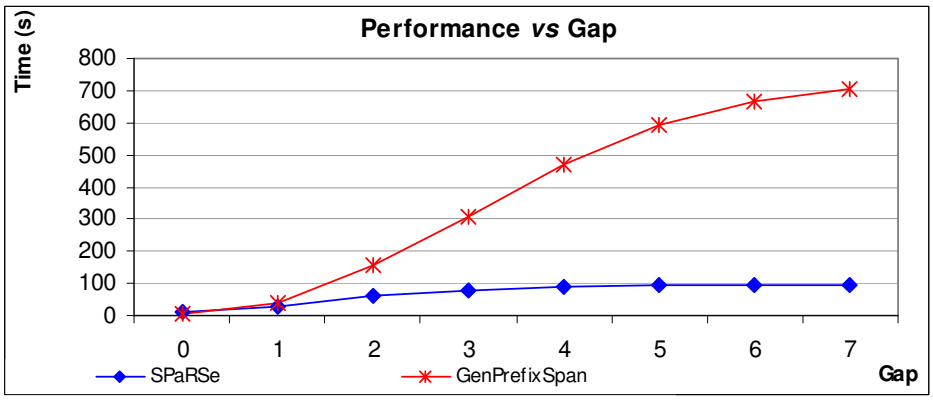


Figure 4.8 – Performance with different gap constraints (support set to 10%)

In fact, while *SPaRSe* must scan the entire sequences for finding a pattern (even if it is not

present in the sequence), *GenPrefixSpan* only has to look for the itemsets in the positions near the already discovered pattern prefix. When gap is set to zero, *GenPrefixSpan* only has to look at the next position, reducing the amount of time needed in scanning the dataset. Furthermore, for longer gaps, the number of sequences in the projected database increases, which also contributes to reduce its performance.

Scalability

Since the most time consuming operation is scanning the database, the results achieved by algorithms for bigger datasets are not surprising. All algorithms present worst behaviors for large datasets, but with slightly different patterns of growth (Figure 4.9).

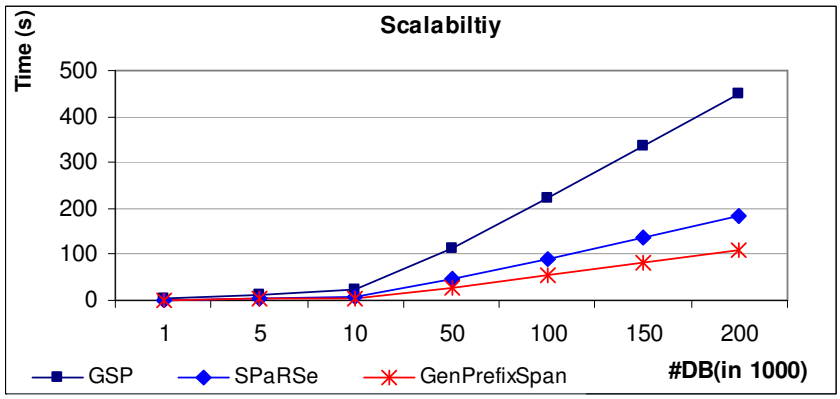


Figure 4.9 – Performance with different dataset sizes (support set to 10%)

The results show that *SPaRSe* and *GenPrefixSpan* present a considerably better performance for very large databases (larger than 10 thousands of sequences) than *GSP*.

It is interesting to see that *GenPrefixSpan* requires much more memory than apriori-based algorithms (see Figure 4.10).

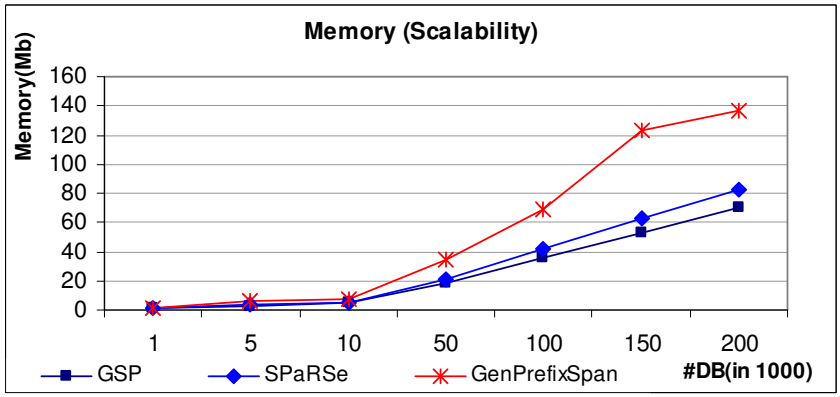


Figure 4.10 – Memory requirements in the presence of different dataset sizes

This difference in memory requirements is due to the creation of projected large databases,

since *GenPrefixSpan* has to maintain multiple indexes for the same sequence and needs to store the pattern position for each occurrence (see section 1.1 for details), wasting more memory than *SPaRSe*. This is clearly most notorious for larger datasets.

Scale-up for different average sequence length

Another important factor in the performance of sequential pattern mining algorithms is the average length of sequences. In order to evaluate different situations, the generated datasets include sequences with different numbers of transactions. Indeed, the sequence length influences the time spent when looking for each frequent candidate.

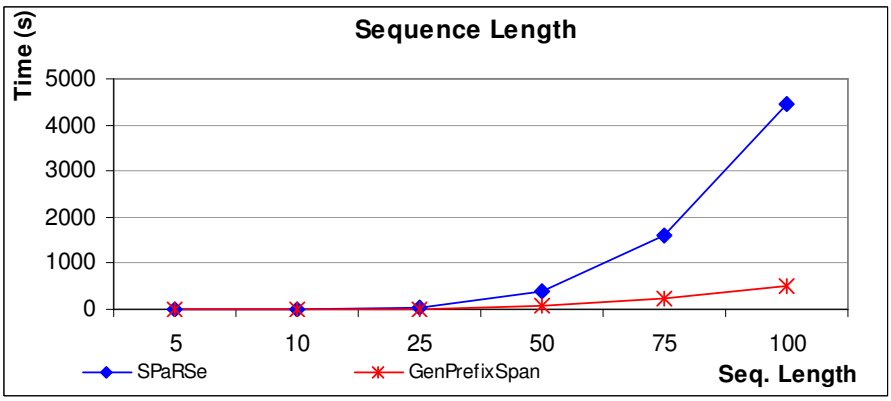


Figure 4.11 – Performance with different average sequence length (support set to 33%)

For long sequences (more than 25 itemsets), the probability of supporting every element is very high. In this manner, *SPaRSe* is not able to reduce the search space (since the support databases approximately maintain the original size) and its candidate pruning does not eliminate a significant number of candidates. On the other side, *GenPrefixSpan* only has to look for the next position, efficiently dealing with long sequences (Figure 4.11).

In summary, the synthetic experiments reveal essentially four aspects:

- *GenPrefixSpan* clearly outperforms *GSP*, maintaining the characteristics of pattern-growth methods;
- *SPaRSe* and *GenPrefixSpan* show similar performances on dense datasets;
- *GenPrefixSpan* outperforms *SPaRSe* in sparse datasets, mainly due to the time spent on candidate generation by *SPaRSe*;
- *GenPrefixSpan* requires much memory than *SPaRSe* and *GSP*.

Summary

In this chapter, we have presented and compared two new algorithms: GenPrefixSpan and SPaRSe.

GenPrefixSpan is a generalization of the PrefixSpan algorithm to deal with gap constraints. In order to achieve that goal, we have proposed a new method to generate projected databases that store the subsequences of all occurrences of each frequent prefix. It is shown that this new method keeps its performance advantages relatively to apriori-based algorithms in the more difficult situation of low support thresholds.

The SPaRSe algorithm is another contribution of this thesis. In general, it can be viewed as an apriori-based algorithm that matches the execution times of GenPrefixSpan, in most situations. This algorithm conjugates the simplicity of candidate generation and test philosophy, with the benefits that come from the restriction of the search space, used by pattern-growth methods. Like apriori-based algorithms, SPaRSe deals naturally with constraints, in particularly with gap constraints.

This chapter also presents a detailed discussion of advantages and disadvantages of both approaches (apriori-based and pattern-growth methods) by comparing the performance and memory requirements of SPaRSe and GenPrefixSpan in a diversity of situations. Identifying the situations where each algorithm presents better results.

Chapter 5
Constraints for Mining Event Sequences

In this chapter, we propose a new theoretical framework for finding patterns over nominal event sequences. This is achieved by the definition of a set of new constraints, which permits the introduction of user background knowledge into the mining process. The chapter finishes with a description of the extensions applied to sequential pattern mining algorithms to deal with those constraints and some experimental results.

Data mining algorithms are usually unable to produce optimal results with respect to all the trade-offs that they account for: usefulness *versus* certainty, concise and understandable models *versus* highly accurate black boxes, sample size *versus* error rate, or simply model expressiveness *versus* computation time. As pointed by Bayardo [Bayardo 2002], constraints play a critical role in solving those problems by focusing "the algorithm on regions of the trade-off curves (or space) known (or believed) to be most promising". By using constraints, the user assumes the responsibility of choosing which of those aspects are most important for the current task.

When considering the discovery of patterns among nominal event sequences, the trade-offs remain challenging and are mostly related to the inverse relation between the amount of discovered patterns and the human ability to deal with those quantities. In order to deal with this problem several categories of constraints have been proposed: existential, item, length, model-based, aggregate, regular expression, and duration and gap constraints (as described previously in Chapter 3, section 1.2).

1 – Ω-constraints

Despite the efforts to capture application semantics using constraints, they have been used in a non-integrated way, being seldom applied to the sequential pattern mining process. An exception is the existential constraint, which is inherent to the pattern mining process, since it defines the notion of frequency for each task.

Definition 25 - An *existential constraint* specifies the minimum threshold support allowed in the mining process.

Although existential constraints play a fundamental role on pattern mining, they only capture the knowledge about the number of entities that are significant for the specific business. As such, they are not able to represent any other knowledge about the business domain.

However, the existence of this background knowledge is a reality. For example, taxonomies defined over the items are often available, since they represent part of the business domain – usually an accepted categorization of the items relevant to it. Another usually known issue is related to the lifespan of items or just to the time interval that is interesting to the analysis.

The other constraints used in pattern mining, can be generally classified into two categories: constraints over *content* and constraints over *temporal* aspects. Indeed, item, aggregate, model-based constraints and regular expressions are concerned with content, since they specify which entities are of interest to the mining task, just by enumerating which ones may be considered or by enumerating interesting relations between different entities. Examples of constraints concerned with temporal issues are duration, length and gap constraints, since they establish temporal relations between the elements of the sequences.

The inexistence of a framework able to register these types of knowledge has hampered the expansion of the use of pattern mining in general, and of pattern mining over nominal temporal data in particular. In order to provide this integrated framework, we propose a new class of constraints – the Ω-constraint.

Definition 26 - An Ω-*constraint* is a triple $\Omega = (\varphi, \theta, \sigma)$, where φ is a content constraint, θ is a temporal constraint and σ is an existential constraint.

The goal of this class of constraints is to allow an easier representation of background knowledge, including content and temporal knowledge. This is achieved by the aggregation of

three categories of constraints: existential constraints that are the core of pattern mining algorithms; content and temporal constraints that are able to represent the existing knowledge about the business domain structure and rules.

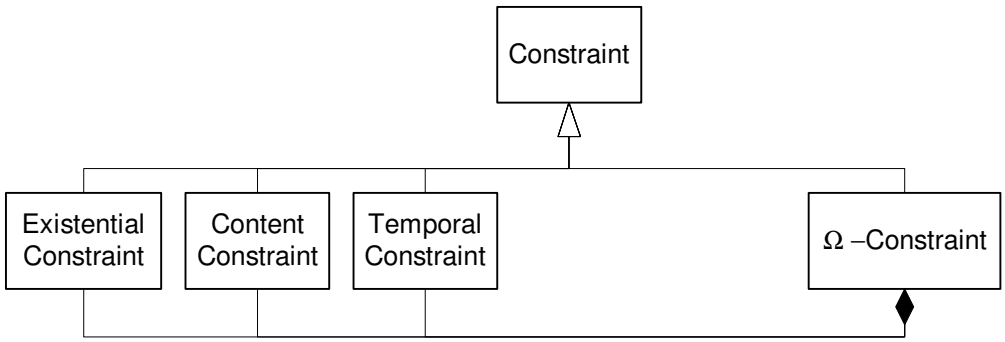


Figure 5.1 – Constraint's hierarchy

Figure 5.1 illustrates the relations between the four classes of constraints. The diagram is represented in UML [Alhir 1998], where △ represents the inheritance and ♦ represents the aggregation. Content and temporal constraints are defined next.

2 – Temporal Constraints

In general, temporal issues have been discarded from the sequential pattern mining process. For example the interesting approaches to explore existent temporal relations, as the ones used by Özden and Ramaswamy for discovering *cyclic* [Özden 1998] and *calendric* rules [Ramaswamy 1998], did not have often been applied to sequential pattern mining. Actually, most sequential pattern mining algorithms do not use any temporal constraint (see for instance [Agrawal 1995c], [Zaki 1998a], [Pei 2001] and [Ayres 2002]). The easy matching of the position of each sequence itemset with a time instant has lead to a simplification of the problem. Nominal event sequences are usually represented without the timestamp of each event. In this manner, algorithms do not have to concern themselves with temporal relations beside the sequential nature of the data.

Like items, timestamps are used in accordance with the business domain. However, while items are specific for each domain, the possible values for timestamps usually belong to a pre-defined set of time-concepts. Since these concepts are common to most business domains and can be represented in a similar way in the generality of applications, several representations have been proposed.

As described in Chapter 2, section 3.2, the most usual representation for time concepts is based on the use of calendars that are formally defined as structured collections of time intervals. However, calendars make use of a set of time concepts that have to be defined precisely, in order to make possible the clear definition of time constraints.

Fortunately, the recent developments in the area of knowledge representation make possible the creation, reuse and sharing of ontologies. An ontology is an explicit specification of a conceptualization [Gruber 1998], which means that it is a specification of an abstract, simplified view of the domain.

Since the representation of time is fundamental to any knowledge base that includes representations of change and action, in the last years some Time Ontologies have been proposed (*Simple Time* and *Reusable Time* [Zhou 2002], *DAML-Time* [Hobbs 2003], etc.). Like any other ontology, those present a description of the concepts and relationships that can exist in a given domain. In the case of Time Ontologies, several approaches can be followed, but a large number of them follow the definitions proposed in [Allen 1983], where time points and time intervals are the central concepts.

2.1 – Reusable Time Ontology

In concrete, the *Reusable Time Ontology* [Zhou 2002] is based on the notion of a time line, with time being continuous and linear.

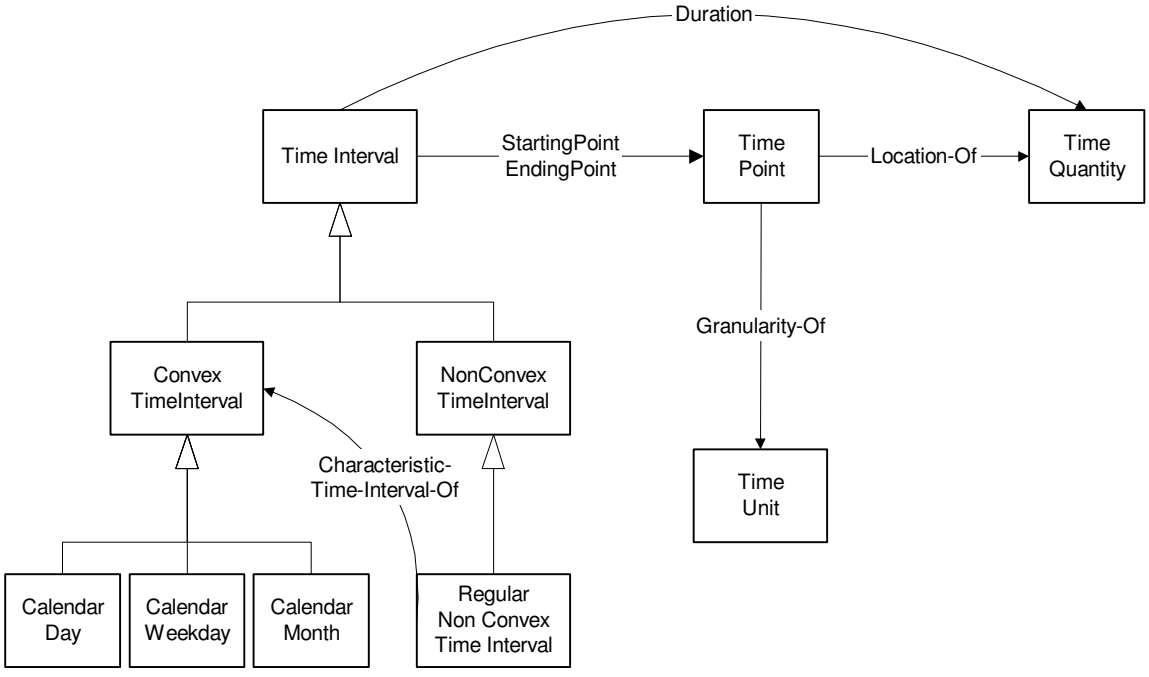


Figure 5.2 – Reusable Time Ontology

The core concepts are `TimePoint` and `TimeInterval`, but include the concepts of `TimeQuantity`, `TimeUnit` and `TimeGranularity`, as shown in Figure 5.2, again represented in UML, and with the relations labeled with their names.

A `TimePoint` represents a specific time position on the timeline, which can be viewed as a particular moment. A `TimeInterval` corresponds to a time-period that occurs between two time points (the `StartingPoint` and the `EndingPoint`). Another way to see time intervals is to consider them as approximations to time points [Hayes 1995], accepting some uncertainty on their value. This means that a time interval may represent partial information about the location of a time point. Time intervals are the most central concept for temporal pattern mining, since they can represent both the lifespan of items and periods of interest. A `TimeQuantity` is an amount of time that is represented by a real number and a `TimeUnit`. Time units correspond to the granularities of time, such as year, month, day, hour, and so on.

This ontology defines two classes of time intervals, *convex* and *non-convex* ones. The first class corresponds to connected intervals in the time line and the second is a disjoint and complete decomposition of time intervals, which correspond to non-connected time intervals, i.e. with "holes" in them. A special kind of non-convex time interval is the `RegularNonConvexTimeInterval`, which is composed of several convex time intervals for representing regularly recurring events.

As can be seen on Figure 5.2, the granularity is an attribute of a time point, which means that it occurs anywhere in a certain time interval, with some uncertainty. The ontology also provides a set of predefined `ConvexTimeIntervals`, for different days, weekdays and months – `CalendarDay`, `CalendarWeekday` and `CalendarMonth`, respectively.

The *Reusable Time Ontology* also distinguishes between open and closed time intervals. However, those concepts are not needed to define temporal constraints, and consequently are not addressed in this text.

2.2 – The Hierarchy of Temporal Constraints

Using the concepts defined in the *Reusable Time Ontology*, the definition of temporal constraints is considerably easier.

Definition 27 - A *temporal constraint* is a triple $\theta = (\Delta, \delta, \gamma)$, which specifies some restriction over the timestamps of transactions. It is composed of:

i) a *time interval* (Δ);

ii) a time quantity to represent the allowed *time gap* (δ);

iii) a *time granularity* (γ).

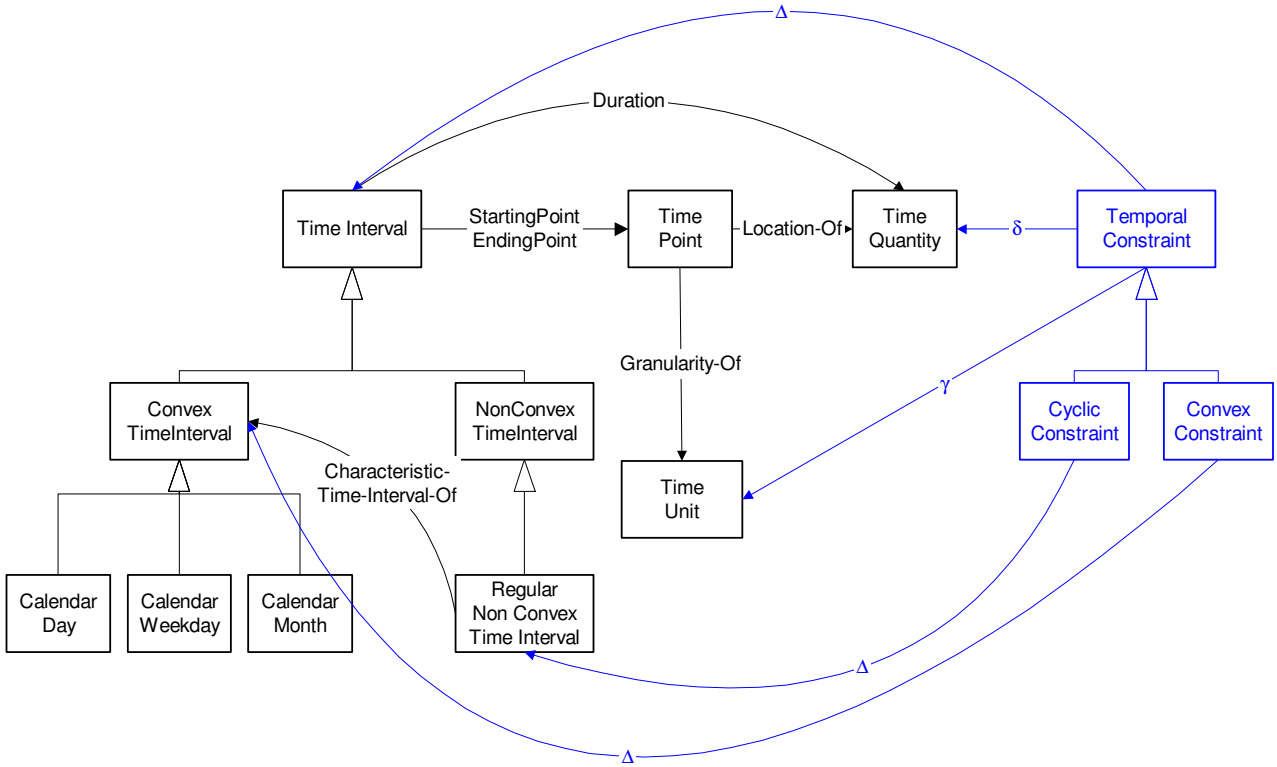


Figure 5.3 – Temporal Constraints: regular and convex constraints

As in time intervals, we can distinguish two classes of temporal constraints: *Convex Temporal Constraints* and *Cyclic Constraints* (Figure 5.3).

Definition 28 - A *convex constraint* is a temporal constraint, whose time interval is a convex time interval.

These constraints are adequate to specify contiguous amounts of time, like the lifespan of some item, as defined in Chapter 3, section 1.2. When using a convex time interval to verify if a time instant belongs to the interval, it is only necessary to compare the corresponding time point with the starting and ending points of the time interval.

Definition 29 - A *cyclic constraint* is a temporal constraint, whose time interval is a regular non-convex time interval.

Cyclic constraints are ideal to deal with cyclic patterns, since they permit the definition of non-contiguous time intervals, but impose a determined periodicity. The verification of the satisfiability of these constraints is more complex than for convex constraints, but still feasible: a time instant belongs to a regular non-convex time interval, if and only if it belongs to any of the convex intervals of the main interval.

The other two attributes of temporal constraints complement the role of Δ (the time interval), since γ (the time granularity) specifies the granularity of interest, and δ specifies the time gap allowed between consecutive events.

3 – Content Constraints

Content has been the main focus of pattern mining and the use of constraints concerning content has been widely studied by the data mining community, probably due to its generality. Examples of this kind of constraints are item constraints and regular languages, as described in Chapter 3, section 1.2.

Content constraints provide an integrated framework to represent the knowledge about the items relevant in the business domain and registered along time. In order to answer these challenges, we propose a new *content constraint*.

Definition 30 - A *content constraint* is a tuple $\varphi=(\Sigma, \tau, \lambda, \pi)$, which is defined over the set of items, specifying the characteristics that sequences of itemsets may have. It consists of:

i) the *alphabet* or set of items (Σ) to consider;

ii) a *taxonomy* (τ) defined over the set of items and the *abstraction level* (λ) to be used in the mining process;

iii) a *formal language* (π) defined over the alphabet, for specifying the accepted path.

The alphabet corresponds to a general item constraint, specifying which are the items to be considered relevant; the taxonomy (as defined in Chapter 2, section 2.1) is used to represent the background knowledge about the items and existent *is-a* relations among them. The first difference to existing approaches is the use of a specific abstraction level (λ). As noted in Chapter 3, section 1.2, the discovery of generalized patterns introduces the need to measure the interestingness of

each discovered pattern, increasing the complexity of algorithms. In order to conjugate the benefits of being able to represent background knowledge and the reduction of the number of discovered rules, we propose the specification of the abstraction level that would be used in the pattern mining process. In this manner, the discovered patterns would be as general as desired by the final user, according to his expectations and knowledge domain. Simultaneously, it is possible to reduce the number of patterns, agglomerating several possible sequences on more abstract ones, avoiding the assessment of the interest of patterns.

Finally, the formal language specifies which sequential patterns are accepted as interesting to the user. As pointed out in Chapter 2, section 3.1, there are several kinds of formal languages. In this work, we propose the use of context-free languages to define the parameter π of the content constraint. The next section explains how context-free languages can be applied to constrain the sequential pattern mining process.

3.1 – Content Constraints with Context-free Languages

As pointed before (in Chapter 3), regular languages are unable to represent some of the most interesting patterns. Since context-free languages are more general than regular ones and are able to deal with part of those patterns, it is natural that we try to use them to guide the mining process.

The first problem that should be faced is related with the use of context-free languages with itemsets. In fact, a language is a set of finite sequences of symbols, but sequential pattern mining deals with sequences of itemsets. Considering that an itemset is not a symbol, the use of any formal language to constrain the mining process is not directly possible. However, as shown in related work (Chapter 3, section 1.4) the use of languages can be very useful.

A simple way to deal with this problem is to consider that each itemset corresponds to a different symbol in a new alphabet that corresponds to the powerset of items in the original alphabet (excluding the empty set).

For example, if there are three different items x, y and z, the new alphabet would be composed of a, b, c, d, e, f and g, with the symbols a, b and c corresponding to the items x, y and z, respectively, and the symbols d, e, f and g corresponding to the itemsets (x,y), (x,z), (y,z) and (x,y,z), respectively.

The second problem that should be solved is the use of pushdown automata instead of finite automata, since context-free languages are generated by (non)deterministic pushdown automata, while regular languages are generated by deterministic finite automata. In general, the use of this

class of languages in sequential pattern mining introduces two new challenges:

- to manipulate the pushdown stack effectively;
- and to deal with the non-determinism.

In fact, the non-determinism of some of the pushdown automata has influence on the efficiency of mining algorithms, since the verification of the acceptability of each sequence can be much slower.

However, the manipulation of the stack introduces additional difficulties when in the presence of sequences of itemsets, instead of sequences of items (strings). A similar difficulty was also noted by SPIRIT authors, which had to define some intermediate predicates in order to deal with itemsets.

Context-free Languages with Itemsets

However, in order to solve these difficulties, the syntax and semantics of pushdown automata have to be extended.

The problem is related to the fact that sequential pattern mining algorithms manipulate one item per iteration, instead of an entire itemset. In this manner, we need to perform partial transitions, corresponding to the item involved at the specific step iteration. To illustrate this situation consider the pushdown automaton represented in the Figure 5.4.

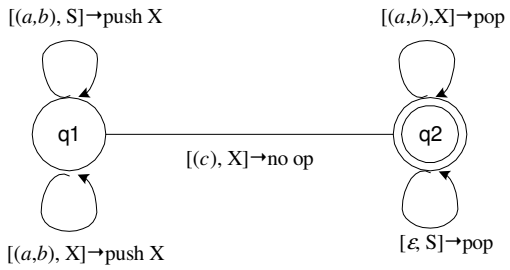


Figure 5.4 – Pushdown automaton with itemsets

The automaton generates sequences with the same number of baskets (a,b) on the left and right side of c, which means that it generates sequences like $(a,b)c(a,b)$ or $(a,b)(a,b)c(a,b)(a,b)$.

Consider for example that algorithm *PrefixGrowth* is applied to find patterns that satisfy this restriction, and it finds a, b and c as frequent. Then it has to proceed to discover which items are frequent after a. At this point, there is already one problem: given that it has found a, which operation should it perform over the stack? If it applies the push X, then c will be accepted after a. However, if the push operation is applied after finding b, then it will accept as "potentially

accepted" sequences like *aaa*, *aaaaa* and so on, since S remains on the top of the stack.

In order to deal efficiently with itemsets, we have to extend the notion of PDA.

Definition 31 - An *extended pushdown automaton* (ePDA) is a tuple $\mathcal{E}=(Q, \Sigma, \Gamma, \delta, q_0, w_0, \mathcal{F})$, with Q, Σ, Γ, q_0 and $\mathcal{F}$ defined as for pushdown automata; $w_0=Z_0\$$ with Z_0 the start symbol as in pushdown automata, and δ defined as a mapping function from $Q\times\mathcal{P}(\Sigma)\cup\{\varepsilon\}\times\Phi$ to finite subsets of $Q\times\Phi^*$, with $\mathcal{P}(\Sigma)$ representing the powerset of Σ and Φ defined as follows

$$\Phi = \{w \in \Gamma \cup \{\$\}: w = x^*\$ \wedge x \in \Gamma\}$$

The difference to pushdown automata is the transition function, which manipulates itemsets and strings of stack elements instead of items and stack elements, respectively. Figure 5.5 illustrates an extension to the PDA illustrated in Figure 5.4.

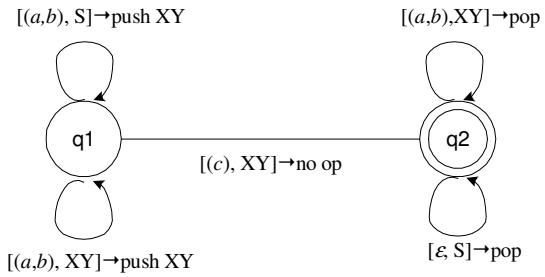


Figure 5.5 – Extended pushdown automaton equivalent to the PDA in Figure 5.4

Note that the pop operation removes the top of the stack, which means that it removes the entire queue XY.

Implementation Issues

With the proposed extension, it is now possible to represent more interesting constraints over sequences of itemsets than with finite automata.

However, in order to explore sequential data with existing sequential pattern mining algorithms, it is necessary to adapt those algorithms to use ePDAs instead of DFAs. In particular, given that algorithms manipulate one item at each step (instead of entire itemsets), the implementations of the algorithms have to be able to manipulate the stack elements in a partial form. In this manner, at the implementation level, we need two new operations: a partial push – `addAtLast`, which introduces a new symbol on the element on the top of the stack, and a partial

pop – `removeFirstOnLast`, which removes the first element on the top of the stack.

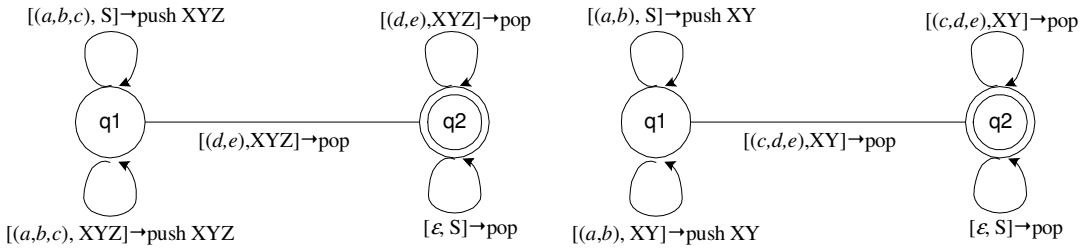


Figure 5.6 – Two other extended pushdown automata

Now consider the automata in Figure 5.6. The number of items in both baskets is not equal, but pop operations can be performed in different manners. When there are more elements in the top of the stack than the items on the basket (exemplified on the automaton on the left), the pop operation is performed by several `removeFirstOnLast` operations, one per each step: it removes the first element on the top of the stack, until it has found the last item, then it removes the top of the stack. In the presence of the other situation (illustrated by the automaton on the right), the pop operation will remove X when it finds *c*, removes Y when it finds *d* and removes the top of the stack when it finds *e*. The choice of each kind of pop operation can be determined by comparing the basket size with the number of elements in the top of the stack.

Next, we will discuss how sequential pattern mining algorithms can be modified to use efficiently temporal pattern mining constraints.

4 – Algorithms for Ω–constraints

As shown before, the use of constraints may have a great impact on the design of sequential pattern mining algorithms. Fortunately, the use of Context-Free Languages, instead of Regular languages, does not imply any adaptation to existent algorithms, but only to the implementation of pushdown automata (as explained in the last section). Moreover, those algorithms can be extended to use gap constraints as well without any additional change.

In order to provide an algorithm that deals with Ω–constraints, we propose an extension of *GenPrefixSpan – GenPrefixGrowth*, which follows the proposals of *PrefixGrowth* closely. In general, the structure of this new algorithm remains the same, as shown in Algorithm 8.

```
GenPrefixGrowth (Database DB, Ω–Constraint Ω)
    Σ←DB.alphabet
    projDBs←new array[|Σ|]
    f_list←discoverL1(DB, Ω, Σ, projDBs)
    for each i∈{0,1,...,|f_list|} do
        b←f_list[i]
        if Ω.prefixAccepts(b) then
            if Ω.accepts(b) then
                L←L∪{b}
            L←L∪run (projDB[i], b, f_list, Ω)
return L
```

```
run (ProjDB DB, EventSequence α, Alphabet Σ,
                                  Ω–Constraint Ω)
    projDBs←new array[|Σ|]
    f_list←discoverFList(DB, α, Ω, Σ, projDBs)
    for each i∈{0,1,...,|f_list|} do
        b←f_list[i]
        α'←αb
        if Ω.accepts(α') then
            L←L ∪ α'
        L←L∪run (projDB[i], α', Σ, Ω)
return L
```

Algorithm 8 – Pseudocode for the main procedures of *GenPrefixGrowth* algorithm

The great difference to *GenPrefixSpan* resides on the verification that α is a valid prefix in accordance to the Ω–constraint, as used by *PrefixGrowth*, and the inexistence of a specific procedure to generate the projected databases.

In fact, the introduction of several constraints simultaneously, implies the verification of each potential pattern by several different filters (content, temporal and existential ones). In order to deal efficiently with this aggregation of constraints, the algorithm should avoid the multiple test of each potential pattern.

For example, the first step of both algorithms consists on the discovery of frequent elements in the database (call it `discoverL1`). In particular, `discoverL1` in *GenPrefixGrowth* also has to identify the sequences that have events belonging to the specified time interval. In this manner, the first time that `discoverL1` is called, it determines the minimum number of times that a sequence has to occur, to be considered frequent, as illustrated in Algorithm 9.

```
discoverL1(Database DB, Ω–Constraint Ω, Alphabet Σ, ProjDB[] projDBs)
    for each i∈{0,1,...,|DB|} do
        s←DB.sequence(i)
        found←Ω.findSerialEvents(α, s, Σ, projsDBs, true)
        if found then
            support++
    Ω.setSupport(support)
return {x∈Σ: x is frequent}
```

Algorithm 9 – Pseudocode for the `discoverL1` procedure in *GenPrefixGrowth*

In the following steps, the difference is the same: *GenPrefixGrowth* has to verify the

acceptability of α in accordance to the imposed constraint. The new `discoverFList` procedure performs this task, creating the projected databases simultaneously (Algorithm 10). Like in *GenPrefixSpan*, the procedure has to discover the parallel and serial valid events, a task performed by `findParallelEvents` and `findSerialEvents`, respectively.

```
discoverFList(ProjDB DB, EventSequence α, Ω-Constraint Ω, Alphabet Σ, ProjDB[] projDBs)
    Ω.simulate(α)
    for each i∈{0,1,...,|DB|} do
        s←DB.sequence(i)
        k←DB.position(i)
        Ω.findParallelEvents(α, s_k, Σ, projsDBs)
        Ω.findSerialEvents(α, s, Σ, projsDBs, false)
return {x∈Σ: x is frequent}
```

Algorithm 10 – Pseudocode for the `discoverFList` procedure in *GenPrefixGrowth*

The great difference between the versions of `discoverFList`, is the simulation of α in the automaton responsible for generating the formal language used in the content constraint (Ω.*simulate*(α)). When the automaton is deterministic (either a finite or a pushdown automaton), the simulation of α can improve the overall performance. The goal is to avoid the multiple verification of the acceptability of α. After this simulation, the discovery of the items accepted after α, is reduced to the verification that the item in consideration can be appended to α.

The easiest way to integrate this new constraint in the algorithm is to represent the constraint as an object, able to perform each test. In this manner, the algorithm just has to receive it as an argument, and let the constraint discover which sequences are accepted and frequent. This object-based approach is very useful, since it allows the substitution of the constraint with any other that is able to perform the same task. In this manner, there is no difference in algorithms that deal with regular (DFA) or context-free languages (PDA).

Since the constraint encapsulates the methods to decide which elements are accepted, we are no more in the presence of a simple algorithm able to deal with a specific constraint, but in the presence of an algorithm able to deal with a class of constraints. As stated by other authors, these constraints only have to be anti-monotonic or prefix-monotonic, if used by apriori-based or pattern-growth methods, respectively.

In the next chapter, we will use the same algorithm with constraint relaxations, without adding any change. In order to discover relaxed constrained patterns, it is only needed to define and implement constraint relaxations, with `accepts`, `acceptsPrefix`, `findParallelEvents`,

`findSerialEvents` and `simulate` methods.

5 – Experimental Results

The use of constraints has been one of the main issues in sequential pattern mining. On one hand, the discovered patterns correspond to the expected ones and on the other hand, processing times are considerably smaller for constrained processes.

In this section we will show that these claims remain true when applying Ω-constraints, and using *GenPrefixGrowth* to incorporate them into the mining process.

In order to support our claim we will show that:

- *GenPrefixGrowth* outperforms *GenPrefixSpan* whenever the Ω-constraint filters a considerable number of patterns;
- the incorporation of Ω–constraints during the mining process is more efficient then filtering the patterns in a post-processing step;
- the use of Context-Free Languages, as opposed to regular languages, does not invalidate previous claims;

Section 5.1 will support the first two points; section 5.2 will support the third point.

5.1 – Constrained *versus* Unconstrained Mining

As pointed by several authors, the use of constraints inside the mining process, instead of filtering the discovered patterns in accordance with the constraint, reduces the processing time needed to discover all patterns. In this section, we will show that this claim remains valid when comparing the use of Ω–constraints in both situations, whenever the constraint is significantly restrictive.

To illustrate the use of constraints, we have used a synthetic dataset, generated by the dataset generator from IBM Almaden. The dataset contains 10.000 sequences, with 10 transactions each on the average; each transaction has on the average 2 items; the average length of maximal patterns is set to 4 and maximal frequent transactions is set to 2 (dataset D10C10T2S4I2). The values for different sequential patterns (Ns) and transactional patterns (Ni) were also chosen similarly, set to 5.000 and 10.000, respectively. However, the number of different items was set to 10 in order to increase the number of discovered patterns.

Usually, in real problems (eg. the ones addressed in Chapter 7), exists background knowledge

that can be used to constraint the mining process. Since in synthetic datasets, there is no background knowledge about its data, in order to define a Ω–constraint we have proceeded as proposed in [Antunes 2002b]: first, the unconstrained *patterns* were discovered, then we have manually defined a regular language able to represent part of the patterns, and finally this constraint was applied to find accepted patterns. The constraint does not impose any temporal restriction beside the gap constraint (also used by *GenPrefixSpan*) equal to zero (0). Table 3 presents the unconstrained patterns discovered using 50% as the minimum support threshold

Table 3 – Patterns discovered with a 50% support threshold and an unconstrained process

Unconstrained Patterns			
2	262	26	5
(2,6)	22	25	52
(2,6)2	2(2,6)	20	522
(2,5)	2(2,5)	29	0
(2,5)2	222	6	02
(2,0)	2222	62	9
(2,9)	226	622	7

and Figure 5.7 shows an automaton defined over those patterns, which accepts about 20% of the unconstrained patterns and accepts about 30% as valid prefixes.

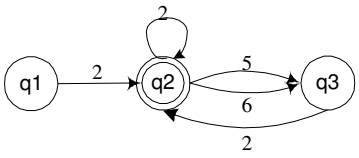


Figure 5.7 – DFA representing the content constraint for the dataset D10C10T2S4I2

The results achieved are clear, and show that the initial claim is true. In fact, the performance of constrained mining (with an Ω–constaint based on that DFA) can be extremely better than the performance of unconstrained processes.

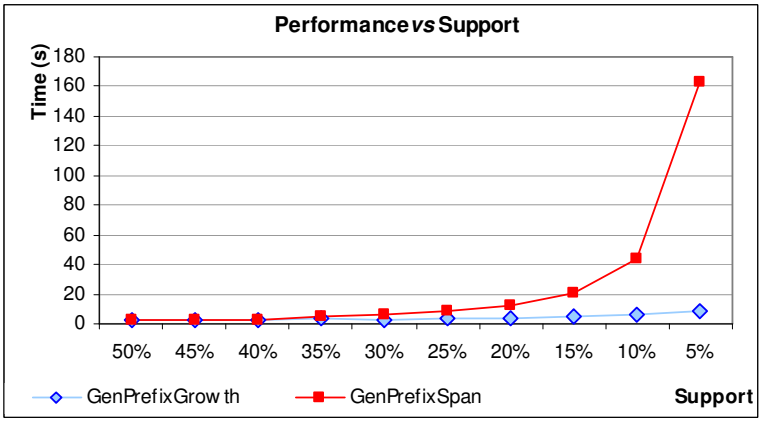


Figure 5.8 – Comparison of processing times spent by unconstrained and constrained mining

Figure 5.8 shows that for very low support thresholds constrained mining can be ten times faster than unconstrained mining. With this difference, it is clear that whenever the constraint is significantly restrictive, the use of constrained pattern mining is much more efficient than any filtering approach after unconstrained mining.

The level of restriction imposed by the constraint can be measured by the ratio between the number of discovered patterns by constrained and unconstrained mining. As shown in Figure 5.9, the constraint used accepts only 1% of the unconstrained patterns for 5% of support.

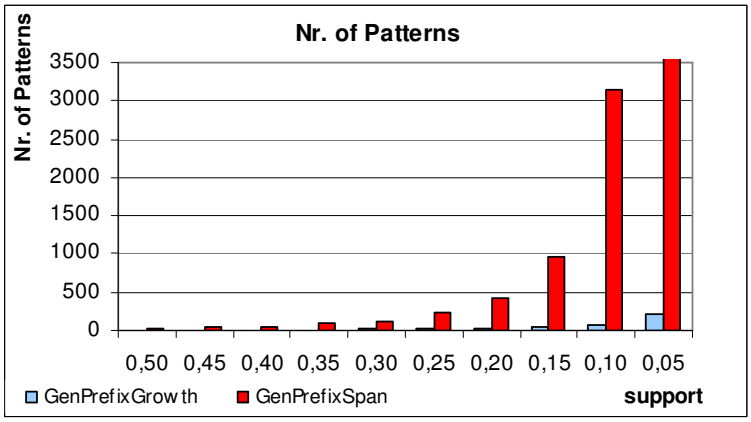


Figure 5.9 – Comparison of number of discovered patterns by unconstrained and constrained mining

Despite these results, it is undeniable that the use of constraints increases the complexity of sequential pattern mining. Indeed, the time spent to verify the acceptability of each pattern is not negligeble.

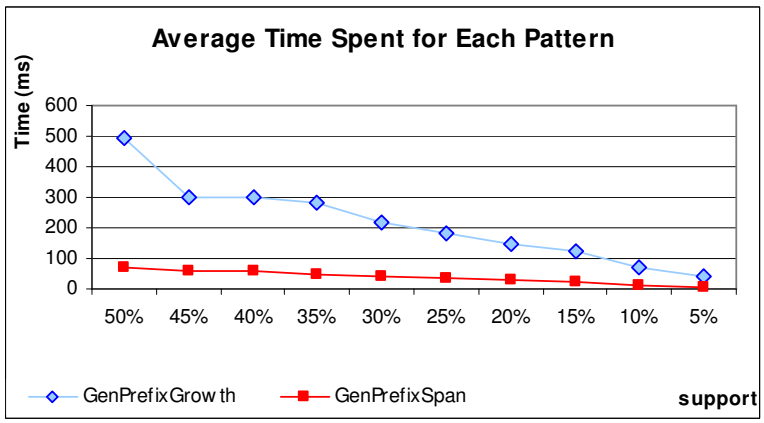


Figure 5.10 – Comparison of the average times spent per pattern by unconstrained and constrained mining

For example, when using the constraint specified above (using a regular language), the time spent for each discovered pattern is five times higher than for unconstrained mining (Figure 5.10).

5.2 – Regular *versus* Context-Free Languages

In order to evaluate the use of context-free languages, consider another two Ω-constraints similar to the previous one, but based on the pushdown automata represented in Figure 5.11.

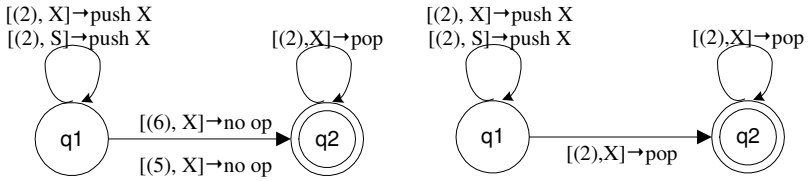


Figure 5.11 – Deterministic (left) and non-deterministic ePDAs (right) defined over unconstrained patterns

These automata impose more restrictive constraints than the DFA defined above; they only accept sequences with the same number of the item 2 before and after the item 5 and 6 (the deterministic) and sequences with an arbitrary number of the item 2 (the non-deterministic one).

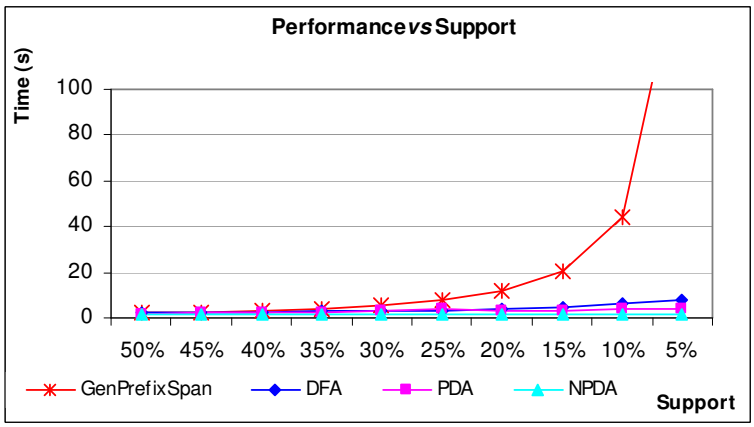


Figure 5.12 – Comparison of processing times spent by different formal languages

Figure 5.12 shows the processing times spent by unconstrained and constrained processes, when in the presence of context-free languages.

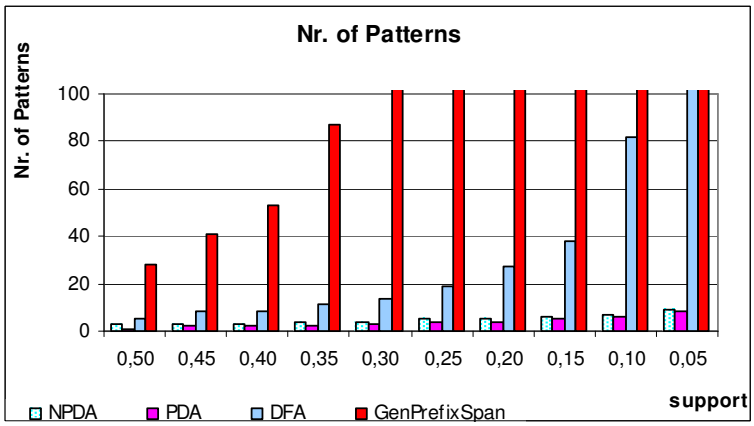


Figure 5.13 – Comparison of number of discovered patterns by different formal languages

It is interesting to note, that when compared with the use of the DFA, the performance of the algorithm using ePDAs is not impaired and is even better than the previous one. This is due to the effective reduction on the number of discovered patterns, which contributes to keep the focus on user expectations (Figure 5.13).

It is interesting to note that the use of context-free languages implies a considerable increase on the time spent for each pattern.

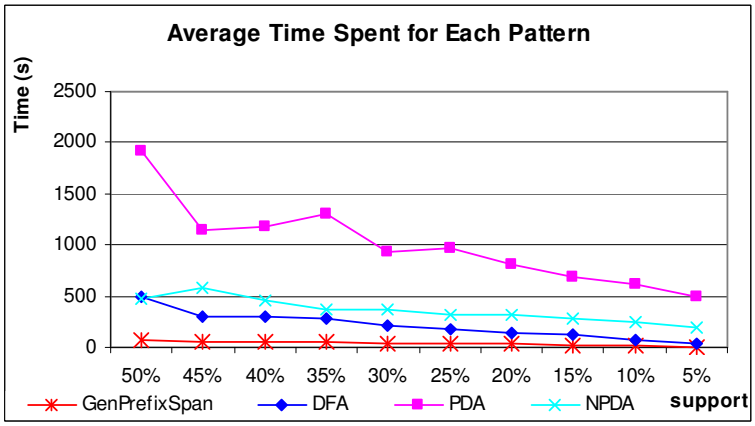


Figure 5.14 – Comparison of the average times spent per pattern by different formal languages

In fact, the time spent for each pattern may be twenty times slower than for unconstrained mining, and four times slower than for constrained mining with regular languages (see Figure 5.14). The time spent for each pattern decrease with the number of discovered patterns, which is due to the decrease of the percentage of discarded sequences, the sequences that are not frequent and accepted by the constraint.

Summary

In this chapter, we have proposed a new class of constraints – the Ω-constraints*, which aggregates content, temporal and existential constraints, in an unique object. The existential constraint is the minimum support threshold, as is usual in sequential pattern mining algorithms. A temporal constraint specifies some restriction over the timestamp of transactions, and uses a time interval, a maximal allowed time gap and the time granularity of the data in the database. Finally, a content constraint is defined over the set of items, specifying the characteristics that sequences of itemsets may have. It uses an alphabet, a taxonomy and a context-free language to specify those characteristics.*

We have extended existent sequential pattern mining algorithms to deal with this new class of constraints creating an algorithm able to deal with any prefix-monotone Ω-constraint

instance – GenPrefixGrowth.

Experimental results show that the additional expressive power of context-free languages can be used without incurring in any additional difficulties, and that post-processing methods are outperformed by constrained pattern mining.

Chapter 6
Constraint Relaxations

In this chapter, we propose the use of constraint relaxations to guide the mining process, avoiding the exclusive discovery of already known patterns caused by the use of constraints. Additionally, several relaxations over Ω-constraints are defined, and an evaluation of the proposed relaxations is presented at the end of this chapter.

The use of constraints in the pattern mining process represents a first answer to the challenge of exploring the existence of user expectations and background knowledge. However, its use may transform the mining process into a simple hypothesis-testing task [Hipp 2002]. In order to avoid this, without moving the user away from the center of the process, we propose a new general data mining methodology: the use of *constraint relaxations*, instead of constraints by themselves, to guide the mining process.

The notion of constraint relaxations has been widely used when real-life problems are addressed. In sequential pattern mining they were first introduced in [Garofalakis 1999], where a regular expression was used to constrain the mining process, and some relaxations were used to improve the performance of the algorithm.

A *constraint relaxation* can be seen as an approximation to the constraint, and when used instead of the constraint, it enables the discovery of unknown information, that will **approximately** match user expectations. If these relaxations are used to mine new patterns, instead of simply used to filter the patterns that satisfy the imposed constraint, the discovery of unknown information is possible. Given that the user may choose the level of relaxation allowed, it is possible to keep the focus and the interactivity of the process, while still allowing for the discovery of new and

unknown information.

In this manner, the objective of data mining will be achieved, while still addressing some unsolved challenges of pattern mining, namely: how to use constraints to specify background knowledge and user expectations; how to reduce the number of discovered patterns by constraining the search space, and how to reduce the amount of processing time.

When applied over Ω-constraints, relaxations are applied to both temporal and content aspects.

Definition 32 - A *constraint relaxation* specifies a weaker condition than the constraint itself. Ω-constraint relaxations have two components:

- an instance of a *content constraint relaxation*;
- the *maximum time error* allowed (ε), specified as a `TimeQuantity`.

The maximum *time error* allowed serves to verify if the time instant belongs to the specified time interval. It is assumed that this error has the same time granularity as the data.

In order to achieve those results, we propose four main classes of relaxations over Ω-constraints, as illustrated in Figure 6.1. The differences between them result from the redefinition of the acceptability notion for the formal language that defines the content constraint.

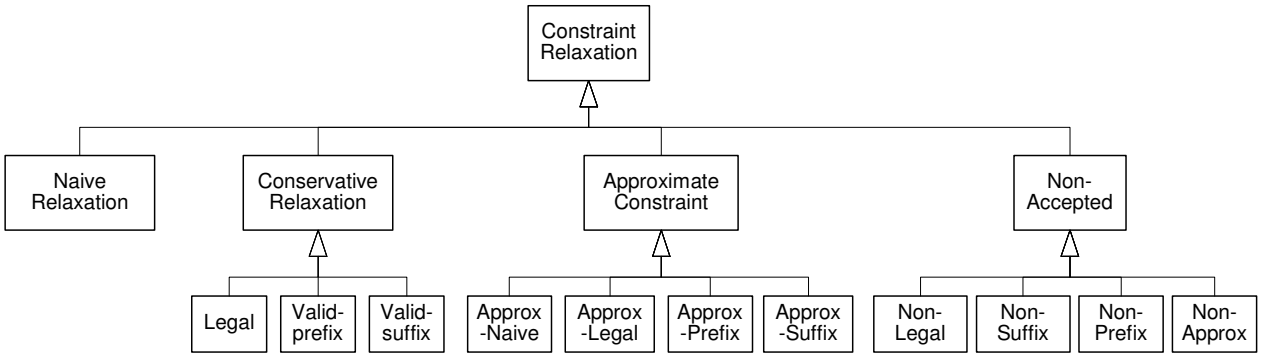


Figure 6.1 – Hierarchy of relaxations

The first class of relaxations is the Naïve relaxation, which corresponds to a simple item constraint. However, in the context of constraints expressed as formal languages, it can be seen as a relaxation that only accepts patterns containing the items that belong to the language alphabet.

Conservative relaxations group the other already known relaxations, used by SPIRIT [Garofalakis 1999], and a third one – Valid-Prefix, similar to Valid-suffix.

It is important to note that conservative relaxations are not able to discover unknown patterns, just sub-patterns of expected ones. *Approximate matching* at a lexical level has been considered an

extremely important tool to assist in the discovery of new facts, but ignored in most of the approaches to pattern mining. It considers two sequences similar if they are at an edit distance below a given threshold. An exception to this generalized frame is the *AproxMAP* [Kum 2003], which uses this distance to count the support for each potential pattern. However, to our knowledge, edit distance has not been applied to constrain the pattern mining process.

To address the need to identify approximate matching we propose a new class of relaxations – the *Approx relaxation*, which accept the patterns that are at an acceptable edit distance from some sequence accepted by the constraint.

Another important issue is related with the discovery of low frequency behaviors that are still very significant to the domain. Fraud detection is the paradigm of such task. Note that the difficulties in fraud detection are related with the explosion of discovered information when the minimum support threshold decreases.

To address the problem of discovering low frequency behaviors, we propose an additional class of relaxations – the *Non-accepted relaxation*. If $\mathcal{L}$ is the language used to constrain the mining process, Non-accepted relaxations will only accept sequences that belong to the language that is the complement of $\mathcal{L}$.

Additionally, each of these relaxations can be combined, creating compositions of relaxations, imposing particular filters. Examples of such compositions are *approx-legal* or *non-approx*.

Next, we will present each class of constraint relaxations. To illustrate the concepts, consider the extended pushdown automaton in Figure 6.2:

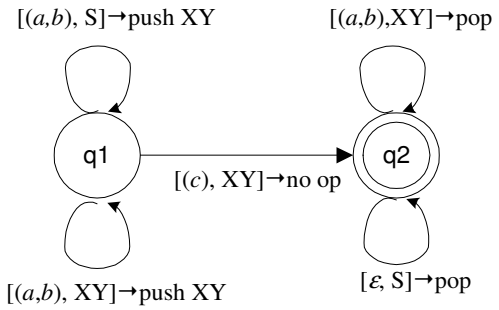


Figure 6.2 – An extended pushdown automaton

1 – Naïve Relaxation

As stated, the *Naïve* relaxation only prunes candidate sequences containing elements that do not belong to the alphabet of the language, For example, if we consider the automaton defined in Figure 6.2, only sequences with *a*'s, *b*'s and *c*'s are accepted by the Naïve relaxation.

In this manner, a sequence is accepted by the naive criterion in exactly the same conditions than for regular languages. However, this relaxation prunes a small number of candidate sequences, which implies a limited focus on the desired patterns.

Since Naïve relaxation is anti-monotonic, no change in sequential pattern mining algorithms is needed.

2 – Conservative Relaxations

Conservative relaxations impose a weaker condition than the original constraint, accepting patterns that are subsequences of accepted sequences. Among these constraint relaxations are *Legal*, *Valid-Suffix* and *Valid-Prefix*, which are similar to the ones used by SPIRIT algorithms.

When used in conjunction with context-free languages, those relaxations remain identical, but we have to redefine the related notions.

First of all consider the partial relation ψ, which maps from $Q\times\mathcal{S}\times\Gamma^*$ to $Q\times\Lambda^*$ representing the achieved state $q\in Q$ and top of the stack $\lambda\in\Lambda^*$ (with Λ equal to Γ^* as defined in Chapter 5), when in the presence of a particular sequence $s\in\mathcal{S}$ in a particular state $q\in Q$ and a string of stack symbols $w\in\Gamma^*$. Also, consider that λ.top is the operation that returns the first element on λ.

Definition 33 - $\psi(q_i, s=<s_1\ldots s_n>, w)$ is defined as follows:

i) (q_i, λ), if $|s|=0 \wedge \exists\, \lambda\in\Lambda^*: \lambda=w$

ii) (q_j, λ), if $|s|=1 \wedge \exists\, q_j\in Q;\ \lambda\in\Lambda^*: \delta(q_i,s_1,w) \supset (q_j, \lambda)$

iii) $\psi(q_j, <s_2\ldots s_n>, \lambda.\text{top})$, if $|s|>1 \wedge \exists\, q_j\in Q;\ \lambda\in\Lambda^*: \delta(q_i,s_1,w) \supset (q_j,\lambda)$

Additionally, consider that the elements on each itemset are ordered lexicographically (as assumed by sequential pattern mining algorithms).

2.1 – Legal

The *Legal* relaxation requires that every sequence is legal with respect to some state of the automaton, which specifies the constraint language. The extension of the legal relaxation to context-free languages is non-trivial, since the presence of a stack (on the automaton) makes the identification of legal sequences more difficult. However, it is possible to extend the notion of legality of a sequence with respect to any state of an extended pushdown automaton.

Definition 34 - A sequence $s=<s_1 \ldots s_n>$ is *legal with respect to state* q_i with the top of the stack w, if and only if

i) $|s|=1 \wedge \exists\, s_k \in \Sigma^*; q_j \in Q; \lambda \in \Lambda^*: \delta(q_i,s_k,w) \supset (q_j,\lambda) \wedge s_1 \subseteq s_k$

ii) $|s|=2 \wedge \exists\, s_k,s_k' \in \Sigma^*; \lambda,\lambda' \in \Lambda^*; q_j,q_j' \in Q: \delta(q_i,s_k,w) \supset (q_j',\lambda) \wedge s_1$ suffixOf s_k
$\wedge\, \delta(q_j',s_k',\lambda.top) \supset (q_j,\lambda') \wedge s_2$ prefixOf s_k'

iii) $|s|>2 \wedge \exists\, s_k,s_k' \in \Sigma^*; \lambda,\lambda',\lambda'' \in \Lambda^*; q_j,q_j',q_j'' \in Q: \delta(q_i,s_k,w) \supset (q_j',\lambda) \wedge s_1$ suffixOf s_k
$\wedge\, \psi(q_j',s_2 \ldots s_{n-1},\lambda.top) = (q_j'',\lambda')$
$\wedge\, \delta(q_j'',s_k',\lambda'.top) \supset (q_j'',\lambda''') \wedge s_n$ prefixOf s_k'

This means that any sequence with one itemset is legal with respect to an extended pushdown automaton state, if there is a transition from it, defined over a superset of the itemset (i). When the sequence is composed of two itemsets, it is legal with respect to a state, if the first itemset is a suffix of a legal transition from the current state, and the second itemset is a prefix of a legal transition from the achieved state (ii). Otherwise, the sequence is legal if the first itemset is a suffix of a legal transition from the state, and the last one is a prefix of a legal transition from the state reached with $s_2 \ldots s_{n-1}$.

Examples of legal sequences with respect to the initial state of the automaton represented in Figure 6.2 are: *a*, *b* and *c* (by rule i), *bc* and (*a*,*b*)*c* (by rule ii) and *bca* and (*a*,*b*)*ca* (by rule iii). Examples of non-legal sequences are *ac* (by ignoring rule ii) or *acb* (by ignoring rule iii).

Note that ψ is only defined for non-empty stacks. Indeed, in order to verify the legality of some sequence *s*, it is necessary to find a sequence of itemsets *t* that can be concatenated to *s*, creating a sequence *ts* accepted by the automata.

2.2 – Valid-suffix

The *Valid-Suffix* relaxation only accepts sequences that are valid suffixes with respect to some state

of the automaton. Like for the legal relaxation, some adaptations are needed when dealing with context-free languages.

Definition 35 - A sequence $s=<s_1 \ldots s_n>$ is said to be a *valid-suffix with respect to state* q_i with the top of the stack w, if and only if

i) $|s|=1 \wedge \exists\, s_k \in \Sigma^*;\ \lambda \in \Lambda^*;\ q_j \in Q:\ \delta(q_i,s_k,w) \supset (q_j,\lambda) \wedge s_1 \text{ suffixOf } s_k \wedge \lambda.top=\varepsilon$

ii) $|s|>1 \wedge \exists\, s_k',s_k'' \in \Sigma^*;\ \lambda',\lambda'' \in \Lambda^*;q_j,q_j',q_j'' \in Q:\ \delta(q_i,s_k,w) \supset (q_j',\lambda) \wedge s_1 \text{ suffixOf } s_k \wedge \psi(q_j',s_2 \ldots s_n,\lambda.top)=(q_j,\lambda') \wedge \lambda'.top=\varepsilon$

This means that a sequence is a valid-suffix with respect to a state if it is legal with respect to that state, achieves a final state and the resulting stack is empty. In particular, if the sequence only has one itemset, it has to be a suffix of a legal transition to an accepting state.

As before, consider the extended pushdown automaton defined in Figure 6.2. Examples of such sequences are *b*, (*a*,*b*), *c*(*a*,*b*) and *bc*(*a*,*b*). Negative examples are, for instance, *bca* or *bcb*. Note that, in order to generate valid-suffix sequences with respect to any state, it is easier to begin from the final states. However, this kind of generating process is one of the more difficult when dealing with pushdown automata, since it requires a reverse simulation of their stacks.

In order to avoid this difficulty, make use of prefix instead of suffix validity could represent a more useful relaxation, when dealing with context-free languages. Note that valid-suffixes are not prefix-monotone, and could not be easily used by pattern-growth methods [Han 2001b].

2.3 – Valid-prefix

The valid-prefix relaxation is the counterpart of valid-suffix, and requires that every sequence is legal with respect to the initial state.

Definition 36 - A sequence $s=<s_1 \ldots s_n>$ is said to be *prefix-valid* if and only if:

i) $|s|=1 \wedge \exists\, s_k \in \Sigma^*;\ \lambda \in \Lambda^*:\ \delta(q_0,s_k,Z_0) \supset (q_j,\lambda) \wedge s_1 \text{ prefixOf } s_k$

ii) $|s|>1 \wedge \exists\, s_k \in \Sigma^*;\ \lambda,\lambda' \in \Lambda^*;q_j,q_j' \in Q:\ \psi(q_0,s_1 \ldots s_{n-1},Z_0)=(q_j',\lambda') \wedge \delta(q_j',s_k,\lambda'.top) \supset (q_j,\lambda) \wedge s_n \text{ prefixOf } s_k$

This means that a sequence is prefix-valid if it is legal with respect to the initial state and the first itemset is a prefix of a transition from the initial state. An example of valid-prefixes according to the ePDA represented in Figure 6.2 is (*a*,*b*)*c*, while (*a*)*c* or *bc* are examples of non-valid prefixes.

Sequences with valid prefixes are not difficult to generate, since the simulation of the stack begins with the initial stack: the stack containing only the stack start symbol. The benefits from using the suffix-validity and prefix-validity are similar. When using the prefix-validity to generate the prefix-valid sequences with k elements, the frequent (k-1)-sequences are extended with the frequent 1-sequences, in accordance with the constraint.

Note that the legal relaxation accepts all the patterns accepted by valid-suffix and valid-prefix relaxations. In this manner, it is a less restrictive relaxation than the other two. Although these relaxations have a considerable restrictive power, which improves significantly the focus on user expectations, they do not allow for the existence of errors. This represents a strong limitation in real datasets, since little deviations may exclude many instances from the discovered patterns.

3 – Approx Relaxations

In order to solve this problem we propose a class of relaxations that accepts sequences that have a limited number of errors. If it is possible to correct those errors with a limited cost, then the sequence will be accepted.

A new class of relaxations, called *approx relaxations*, tries to accomplish this goal: they only accept sequences that are at a given edit distance for an accepted sequence. This edit distance is a similarity measure that reflects the cost of operations that have to be applied to a given sequence, so it would be accepted as a positive example of a given formal language. This cost of operations will be called the *generation cost*.

Definition 37 - Given a Ω-constraint C, and a real number ε which represents the maximum error allowed, a sequence $s=\langle s_1 \ldots s_n \rangle$ is said to be *approx-accepted* by C, if its generation cost $\xi(s, C)$ is less than or equal to ε.

The *generation cost* is defined as follows:

Definition 38 - Given a Ω-constraint C, and a sequence $s=\langle s_1 \ldots s_n \rangle$, the *generation cost* $\xi(s, C)$ is defined as the sum of costs of the cheapest sequence of edit operations $O=\langle o_1 \ldots o_k \rangle$ transforming the sequence s into some sequence t accepted by the language C.

The edit operations considered are the traditional in approximate string matching

[Levenshtein 1965]:

- *Insertion* – `Ins(x,i)`: adds the item *x* to the sequence at position *i*;
- *Deletion* – `Del(x,i)`: deletes the item *x* from the sequence at position *i*;
- *Replacement* – `Rep(x,y,i)`: substitutes the item *x*, that occurs at position *i*, by item *y*.

For example, considering the extended pushdown automaton defined above (Figure 6.2), *ac*(*a*,*b*) and (*a*,*b*)(*a*,*b*) are approx-accepted sequences with one error, which result from inserting a *b* on the first itemset, on the first example, and a *c* on the second position, on the second example, respectively. *c*(*a*,*b*) and *b*(*a*,*b*) are non-approximate accepted with one error, since two edit operations are needed to accept them.

The other four classes of approx relaxations are defined by replacing the acceptability by legality and validity notions. In this manner, an *Approx-Legal* relaxation accepts sequences that are approximately legal with respect to some state. *Approx-Suffix* and *Approx-Prefix* relaxations are defined in a similar way. Finally, *Approx-Naïve* accepts sequences that have ε items (with ε the maximum error allowed) that do not belong to the alphabet.

It is interesting to note that the ability of approx-relaxations to reduce the number of discovered patterns depends on the size of the alphabet; in particular, it depends on the number of frequent items. Suppose that there are *m* frequent items in a given database, and the alphabet of the formal language used as constraint has *n* items, all frequent (with $n<m$). For each pattern with one item accepted by the constraint (for example, *a*), if we are concerned with approximate patterns with one error, the relaxation may find $3\times m$ patterns with two items (for example (*ab*), *ab* and *ba*).

A solution to this proliferation is to associate an item constraint with the approx relaxation. This conjunction allows for focusing the mining process on a smaller set of the approximate accepted patterns, reducing the number of discovered patterns.

3.1 – Algorithm ε-accepts

Unfortunately, the better-known algorithms for edit distance are performed on two strings, and cannot be applied to one string and a formal language. *Agrep* [Wu 1992] is an exception to this general scenario, and it verifies if there exists a sequence in a given regular language (expressed as an automaton) at a specific edit distance from the given sequence.

However, as pointed by its authors, *agrep* does not deal well with very large alphabets, and sequential pattern mining algorithms usually deal with alphabets with thousands of symbols.

In order to deal with this situation and use context-free languages, we assume that useful automata are small in general, and we propose *ε–acceptsCFL*, a new algorithm to verify if a sequence was approximately generated by a given extended pushdown automaton (ePDA). This algorithm is illustrated in Algorithm 11.

```
boolean ε-acceptsCFL(Sequence s, int ε, ePDA A=(Q,Σ,Γ,δ,q0,Z0,F)){
  return aproxAcc(s,0,A.q0,new Stack(Z0),0,ε,A)
}

boolean aproxAcc(Sequence s, int i, Q qj, Stack λ, int acε, int ε, ePDA A){
  // Exceeds maximal error
  if (acε>ε) return False
  // Proceeds with next element
  if (i<|s|) return accTrans(s,si,i,qj,λ,acε,ε,A)
  // Test the achievement of empty stack
  else if (λ.isEmpty()) return True
  // Try to achieve a final state
  else return accTrans(s,(),i,qj,λ,acε,ε,A)
}

boolean accTrans(Sequence s,Itemset si,int i,Qqj,Stack λ,int acε,int ε, ePDA A){
  // Looks for the perfect transition
  for each q∈Q {
    if (∃ tr: tr∈{qj.getApplicableTrans(q,λ,si)})
      return aproxAcc(s,i+1,q,copy(λ).apply(tr),acε,ε,A)
  }
  for each δ(q,a)∈A.δ(qj){
    dif←max(|si|,|a|)-|a ∩ si|
    //Try a mixture of edit operations (replacement+deletion+insertion)
    if (acε+dif≤ε)
      ok←aproxAcc(s,i+1,q,copy(λ).apply(δ(q,a)),acε+dif,ε,A)
    if (ok) return True
    // If the itemset is not valid => try to ignore it (deletion)
    ok←aproxAcc(s,i+1,qj,λ,acε+|si|,ε,A)
    if (ok) return True
    // If deletion fails, try an insertion
    ok←aproxAcc(s,i,q,copy(λ).apply(δ(q,a)),acε+|a|,ε,A)
    if (ok) return True
  }
  return False
}
```

Algorithm 11 – Pseudocode for ε–acceptsCFL algorithm

The main idea behind *ε–acceptsCFL*, is to avoid the generation of potential sequences, given that this operation consumes a considerable amount of time in sequential pattern mining algorithms. Since we consider that automata are usually small, the simulation of transitions is not an expensive operation. (As several authors have noted, experts usually make use of a few simple and small rules to express their background knowledge, so the combination of those rules usually do not result in complex automata).

Instead of generating the possible sequences, considering possible errors, we verify if each itemset performs a valid transition in the automaton, beginning on the initial state and with the stack containing only the initial stack symbol. Whenever an itemset does not correspond to a valid transition, the algorithm tries to replace it (which corresponds to the application of a *Replacement* or a combination of *Deletions* and *Insertions*). If this fails, it tries to ignore it (which corresponds to a *Deletion*) and finally it tries to introduce a valid transition (which corresponds to an *Insertion*). Given that insertions and replacements only try valid transitions, instead of trying every possible itemset, the performance of ε–`acceptsCFL` does not depend on the number of different items in the database, but only on the number of states and alphabet used by the ePDA.

In general, approx relaxations can be seen as less restrictive than conservative ones. However, this relation is only valid with respect to comparable relaxations, for example: approx-legal is less restrictive than legal.

4 – Non-accepted Relaxations

Another important issue is the possibility to discover information that does not satisfy the constraint imposed by the language.

Suppose that there is a model (expressed as a context-free language) able to describe the frequent patterns existent on a large database (say for example that the minimum support allowed is 10%). If there are 3% of clients with a fraudulent behavior, it is possible that they are not discovered either by using the unconstrained mining process, or by using any of the proposed relaxations.

However, the model of non-fraudulent clients may be used to discover the fraudulent ones: the fraudulent clients are known to not satisfy the model of non-fraudulent clients.

Definition 39 - A sequence $s=<s_1 \ldots s_n>$ is said to be *non-accepted* by the language if it is not generated by that language.

In fact, this is not really a relaxation, but another constraint (in particular the constraint that only accepts sequences that belong to the language that is the complement of the initial constraint). However, since they are defined based on the initial constraint, we choose to designate them as relaxations.

The benefits from using the non-accepted relaxation are mostly related to the possibility of not rediscovering already known information, which may contribute significantly to improve the performance of sequential pattern mining algorithms. Moreover, since context-free languages are not closed under complementation [Hopcroft 1979] (which means that the complement of a context-free language is not necessarily a context-free language), the use of the complement instead of the non-accepted relaxation could be prohibitive.

Note that using this new approach, it is possible to reduce the search space, and consequently to reduce the minimum support allowed. The non-accepted relaxation will find all the patterns discovered by the rest of the introduced relaxations, representing a small improvement in the focus on user expectations. In fact, it finds all the patterns discovered by unconstrained patterns minus the ones that are accepted by the constraint. Like for *approx* relaxations, an interesting improvement is to associate a subset of the alphabet in conjunction with non-accepted relaxation. This conjunction focus the mining process over a smaller part of the data, reducing the number of discovered sequences, and contributing to achieve our goal.

As before, the sub-classes of Non-Accepted relaxations result by combining the non-acceptability philosophy with each one of the other relaxations. While non-accepted relaxation filters only a few patterns, when the constraint is very restrictive, the non-legal relaxation filters all the patterns that are non-legal with respect to the constraint. With this relaxation is possible to discover behaviors that completely deviate from the accepted ones, helping to discover the fraudulent behaviors.

Non-accepted relaxations are particularly interesting when the constraint is not very restrictive.

5 – Discussion

The discussion about the concept of novel or unknown information is one of the most difficult in pattern mining. While the concept is clear in the reference frame of a knowledge acquisition system, the same is not true in the reference frame of the final user. Indeed, several interestingness measures have been proposed for the evaluation of the discovered patterns [Hilderman 1999].

Moreover, this issue is more critical with the introduction of constraints in the mining process. In fact, in the presence of constraints the concept of novel patterns becomes unclear even in the reference frame of information systems, since they are then able to deal with that knowledge,

represented as the constraint.

In order to bring some light into the discussion, consider that,

Definition 40 - Given a model *C* as constraint, a pattern *A* is *more novel than* a pattern *B*, if the generation cost of *A* in order to *C* is larger than the generation cost of *B* in order to *C* (with the generation cost defined as above).

With this concept, it is now possible to understand the reason why non-accepted patterns can be more novel than the patterns discovered by conservative relaxations. It is now clear that, despite the differences between relaxations, all of them allow for the discovery of novel information. Indeed, the conservative relaxations are able to discover failure situations, this is, situations when for, some reason, the given model is not completely satisfied (Valid- Prefix and Valid-Suffix identify failures in the begin and end of the model, respectively, and Legal identifies problems in the middle of the model).

However, the great challenge of pattern mining is to discover novel information in accordance to user expectations. It is clear from the definition of the novel relation, that an unexpected pattern is more novel than an expected one. In fact, the challenge resides in the balance between the discovery of novel but expected patterns, and the proposed relaxations cover a wide range of this balance, giving to the user the option of which is the most relevant issue for the problem in hands: the novelty of the information or to satisfy the expectations.

6 – Simple Example

In this section, we will illustrate the use of relaxations over Ω–constraints with a simple example. Our goal is to validate our claim that they facilitate the discovery of unknown information, keeping the mining process centered on the user.

Consider again the problem of identifying typical behaviors of some company customers. Suppose that the company considers that a well-behaved customer is a customer, who always makes at least one payment after receiving one or two consecutive invoices, and has made, at the end of the period, all its payments. This constraint may be modeled by the pushdown automaton presented in Figure 6.3, where *a* corresponds to an invoice, *b* to a payment, *c* to a second copy of the invoice, *d* to a cancellation, *e* to an information request and *r* to an answer to those requests.

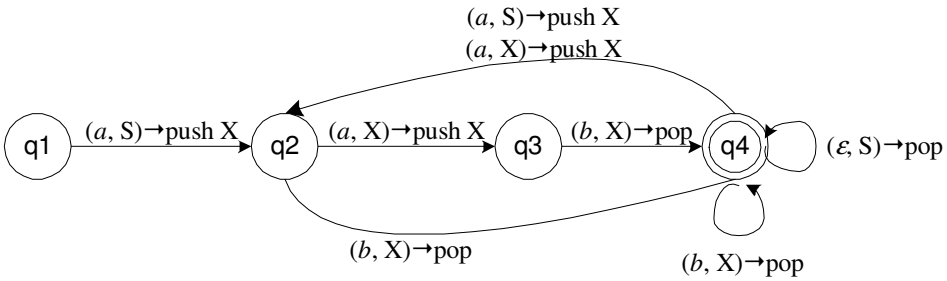


Figure 6.3 – Pushdown automaton for well-behaved customers

Suppose that the company has the dataset represented in Table 4.

Table 4 – Dataset used to exemplify the mining process

Dataset				
eababraabb	*aababbaerb*	*aaacbabab*	*aebraaaccd*	*aebaraacbe*
aabbaberab	*ababaabbab*	*aebaraaacb*	*abaaaccder*	*abaeraacbb*

Table 5 presents the patterns discovered by unconstrained and constrained algorithms, and the patterns discovered when constraint relaxations are used.

Table 5 – Comparison of the results achieved with and without constraints

Frequent	Accepted	Legal	Prefix	Approx.(ε=1)	Approx.(ε=2)	Non-Acc (w/ Σ={a,c,d,e,r}
be *baa* *era* *braa* *baba* *baer* *araa* *abab* *abaa* *raacb* *raaac* *aaacb* *aabbab* *aaaccd* *aebaraa*	*abab* *aabbab*	*baba* *abab* *abaa* *aabbab*	*abab* *abaa* *aabbab*	*abab* *aabbab* *ar*→R(*r*,*b*,2) *aeb*→D(*e*,2) *abaa*→R(*a*,*b*,4) *aacb*→R(*c*,*b*,3)	*abab* *aabbab* *abaa*→R(*a*,*b*,4) *abbab*→D(*b*,2) *be*→R(*b*,*a*,1) R(*e*,*b*,2) *br*→R(*b*,*a*,1) R(*r*,*b*,2) *cd*→R(*b*,*a*,1) R(*r*,*b*,2) *aer*→R(*e*,*b*,2) D(*r*,3) *bae*→D(*b*,1) R(*e*,*b*,3) *aacc*→R(*c*,*b*,2) R(*c*,*b*,3) *aaacb*→D(*a*,3) R(*c*,*b*,4) *aebar*→D(*e*,2) R(*r*,*b*,5) *araa*→R(*r*,*b*,2) R(*a*,*b*,4) *baba*→D(*b*,1) I(*b*,5) *raacb*→D(r,1) R(c,b,4)	*aer* *era* *raac* *araa* *raaac* *aaaccd*

In order to permit an easy comparison, only maximal patterns are shown. In this manner, some patterns appear only in some columns. In the column relative to the approx relaxation, are shown the edit operations needed to transform each pattern to a pattern belonging to the context-free language. As expected, by using the constraint itself we only discover two patterns, which satisfy the context-free language. Therefore, these results are not enough to invalidate Hipp's arguments about constraints. Nevertheless, with Legal and Valid-prefixes, it is possible to discover some other intermediate patterns, which are potentially accepted by the complete constraint.

Finally, with *approx* and *non-accepted* relaxations, it is possible to discover unexpected

patterns. Indeed, the approx relaxation shows the most interesting behavior, since it is possible to discover that it is usual that after sending the second invoice, customers pay their old bills (*aacb*). With non-accepted relaxation, it is also possible to discover interesting patterns. In this case, it is common that after three invoices without any payment, and the emission of two second invoices, the customer account is canceled (*aaaccd*).

7 – Experimental Results

As shown in the previous chapter, the use of constraints can improve either the performance or the focus on user expectations of sequential pattern mining. However, in some cases, the use of constraints may block the discovery of novel information. In this section, we will show that the initial claims remain true when applying constraint relaxations, and that their use also permits the discovery of unknown information, keeping the process centered on user expectations.

In order to support our claims we will show that whenever the Ω-constraint filters a considerable number of patterns:

- the use of constraint relaxations can be more efficient then unconstrained mining;
- the use of constraint relaxations reduces the number of discovered patterns.

The two Ω-constraints used in our experiments are the same that have been used before (in Chapter 5, section 5): they do not impose any temporal restriction beside the gap constraint (also used by *GenPrefixSpan*) equal to zero, and the content constraint is defined by a formal language.

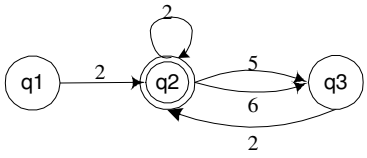


Figure 6.4 – DFA defined over unconstrained patterns

The content constraints only specify restrictions on the accepted sequence of itemsets, in accordance with the DFA represented in Figure 6.4 and with the ePDA in Figure 6.5.

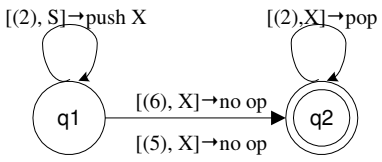


Figure 6.5 – ePDA defined over unconstrained patterns

As expected, by using the constraint itself we only discover a few patterns, which constitute already known information, since they satisfy the imposed language. However, the situation is different for the relaxations. For example, *Legal-Relaxation* begins by accepting 50% of the discovered patterns using a threshold support equal to 50%, and only accepts about 1% of the patterns for a support threshold equal to 5% (see Table 6 and Table 7).

Table 6 – Discovered patterns with several relaxations using DFA in Figure 6.4

sup	Unconstrained	Accepted		Prefix		Legal		Approx		Non-Acc	
0,50	28	5	18%	8	29%	14	50%	28	100%	23	82%
0,45	41	8	20%	13	32%	20	49%	39	95%	33	80%
0,40	53	8	15%	13	25%	20	38%	47	89%	45	85%
0,35	87	11	13%	17	20%	26	30%	72	83%	76	87%
0,30	127	14	11%	21	17%	34	27%	96	76%	113	89%
0,25	228	19	8%	30	13%	49	21%	156	68%	209	92%
0,20	422	27	6%	42	10%	67	16%	248	59%	395	94%
0,15	967	38	4%	62	6%	100	10%	449	46%	929	96%
0,10	3142	82	3%	132	4%	210	7%	1044	33%	3060	97%
0,05	24937	203	1%	334	1%	565	2%	3788	15%	24734	99%

Table 7 – Discovered patterns with several relaxations using ePDA in Figure 6.5

sup	Unconstrained	Accepted		Prefix		Legal		Approx		Non-Acc	
0,50	28	1	4%	8	29%	14	50%	8	29%	27	96%
0,45	41	2	5%	12	29%	20	49%	12	29%	39	95%
0,40	53	2	4%	12	23%	20	38%	13	25%	51	96%
0,35	87	2	2%	15	17%	25	29%	20	23%	85	98%
0,30	127	3	2%	18	14%	30	24%	27	21%	124	98%
0,25	228	4	2%	22	10%	37	16%	48	21%	224	98%
0,20	422	4	1%	27	6%	47	11%	70	17%	418	99%
0,15	967	5	1%	33	3%	56	6%	131	14%	962	99%
0,10	3142	6	0%	44	1%	76	2%	278	9%	3136	100%
0,05	24937	8	0%	63	0%	110	0%	605	2%	24929	100%

A similar behavior is shown by *Prefix-Valid* and *Approx-Accepted* relaxations, both by the regular or the context-free languages, as shown in Figure 6.6 and Figure 6.7.

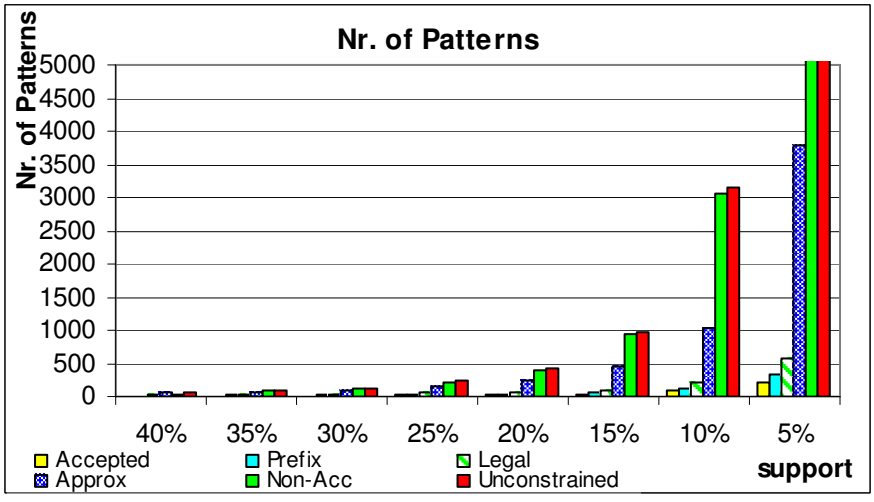


Figure 6.6 – Number of discovered patterns using the DFA in Figure 6.4

In this manner, the claim that the use of constraint relaxations can reduce the search space is validated.

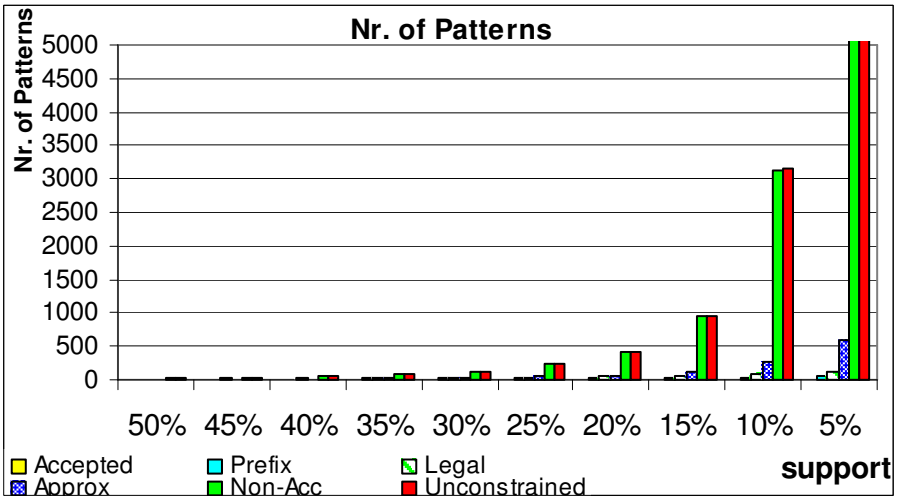


Figure 6.7 – Number of discovered patterns using the ePDA in Figure 6.5

Another interesting result is related to the processing times spent by different approaches (see Figure 6.8 and Figure 6.9), in particular in the presence of very low support thresholds.

Constrained mining is the most efficient, spending about 5% of the time spent by the unconstrained approaches. Conservative relaxations present a similar performance, justified by the low number of discovered patterns and by the similarity between its processing and the processing of the constraint itself.

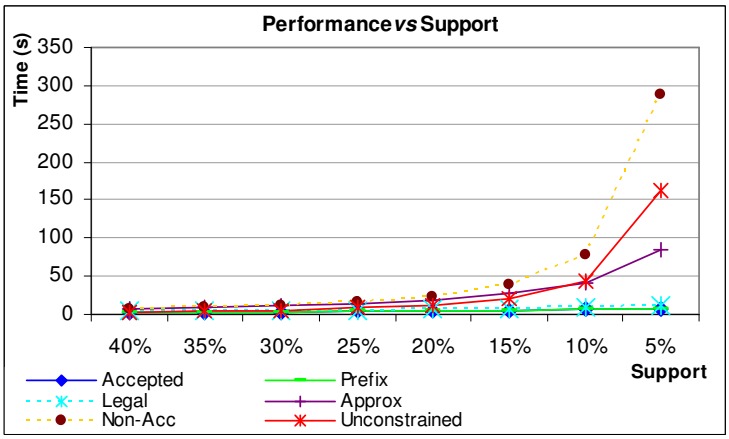


Figure 6.8 – Processing time using the DFA in Figure 6.4

The most time consuming relaxation is the *Non-Accepted*, mostly due to the high number of discovered patterns, identical to the number of unconstrained patterns.

The *Approx* relaxation also consumes a considerable amount of time, higher then the unconstrained mining in the case of the context-free language (Figure 6.9). However, unlike non-

accepted relaxation those processing times are mostly due to the verification of the acceptability of patterns.

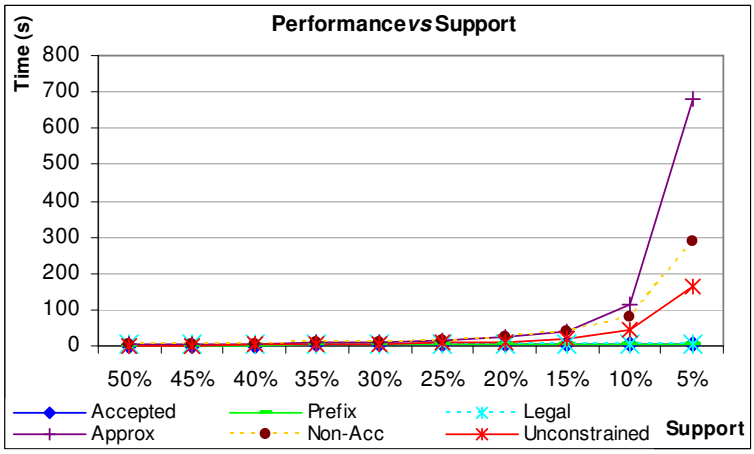


Figure 6.9 – Processing time using the ePDA in Figure 6.5

In general, relaxations spend a portion of time proportional to the number of discovered patterns. When comparing the average time spent by each relaxation per discovered pattern, the most expensive is the constrained process, followed by conservative relaxations, in the case of regular languages, and by the *Approx* relaxation, in the case of context-free languages. This difference is again due to the verification of the acceptability of patterns (see Figure 6.11).

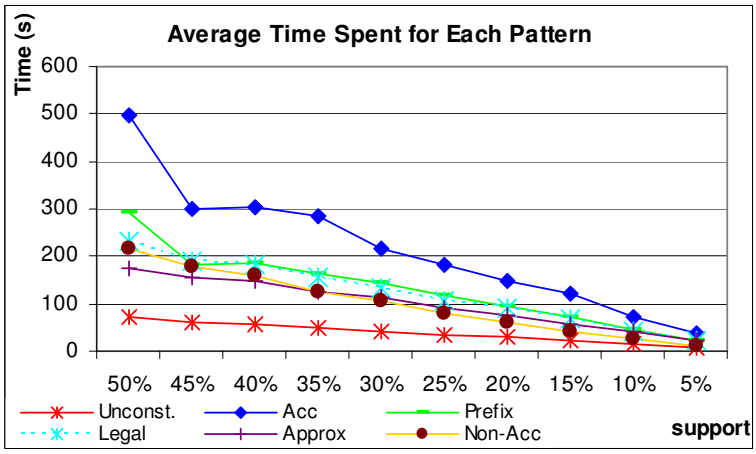


Figure 6.10 – Average time spent by pattern, using the DFA in Figure 6.4

It is important to note that, in general, the average time spent for each discovered pattern decreases when the number of patterns increases. This reduction is due to the decrease of the percentage of discarded sequences, sequences that are not frequent and are accepted by the relaxation.

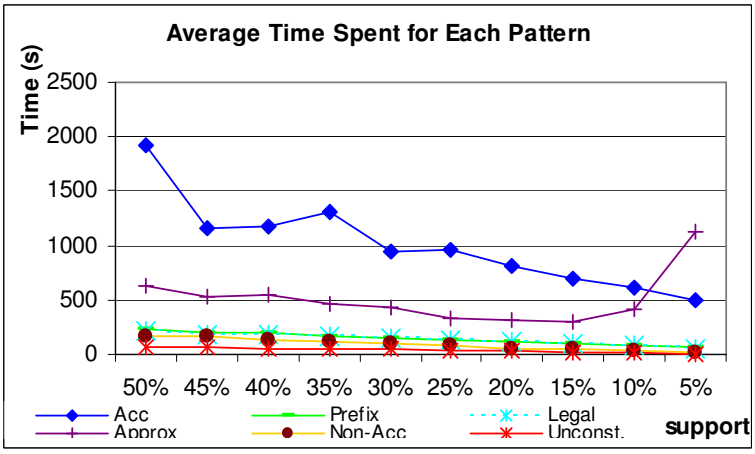


Figure 6.11 – Average time spent by pattern, using the ePDA in Figure 6.5

The exception to this general rule is the *Approx* relaxation based on a context-free language. This difference of behavior is again explained by the verification of the acceptability of patterns. Figure 6.11 shows that the time spent in that verification overwhelms the decrease of the percentage of discarded sequences.

7.1 – Constraint Relaxations combined with Item Constraints

The poor performance achieved by the *Approx* and *Non-accepted* relaxations, can be improved, if item constraints are used in conjunction with non-conservative relaxations, as proposed above.

In order to validate this idea, we have imposed an item constraint, restricting the relaxations to just accepted sequences with items belonging to the formal language alphabet.

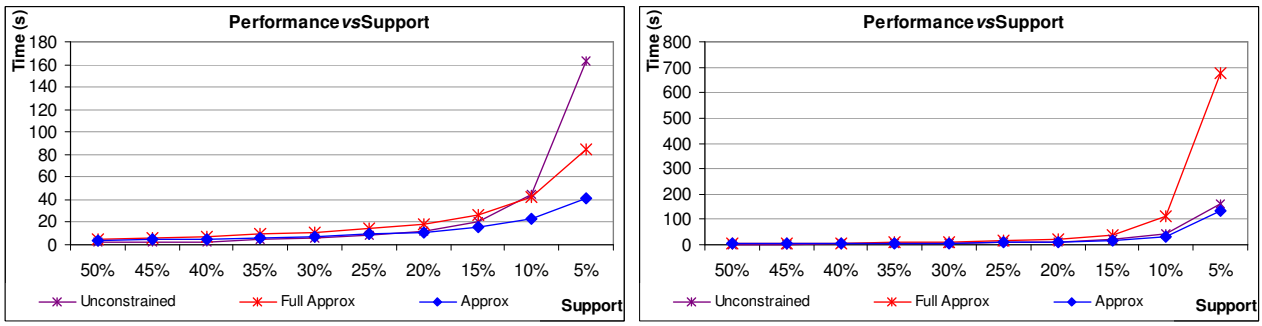


Figure 6.12 – Comparison of processing times of *Approx Relaxation* combined with item constraints with DFA in Figure 6.4 (on the left) and with the ePDA in Figure 6.5 (on the right)

Figure 6.12 shows that in conjunction with item constraints (imposing a more restricted alphabet) *Approx* relaxations are able to outperform unconstrained mining, both with regular and context-free languages. These results are naturally achieved by reducing the number of discovered patterns, as shown in Figure 6.13. In these figures, *FullApprox* refers to the *Approx* relaxation alone and *Approx* refers to the use of the relaxation with the item constraint described above.

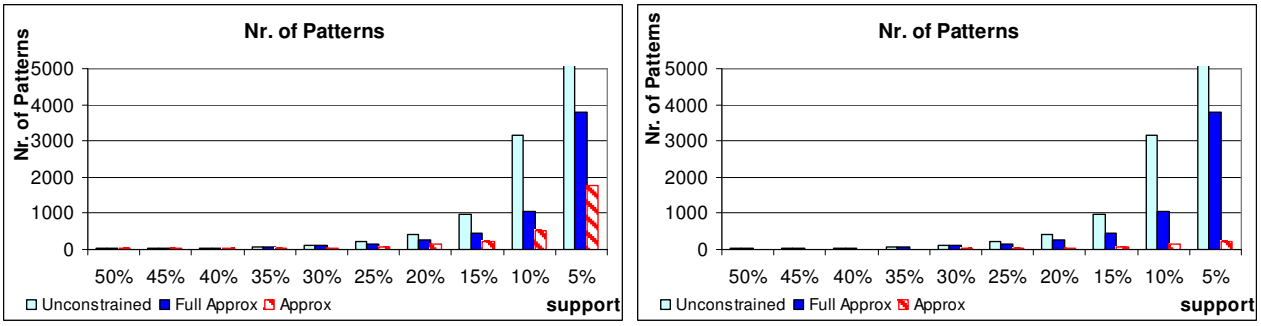


Figure 6.13 – Comparison of number of discovered patterns by *Approx Relaxation* combined with item constraints, with the DFA in Figure 6.4 (on the left) and with the ePDA in Figure 6.5 (on the right)

A comparison of the number of discovered patterns, when using variants of non-accepted relaxations, also validates our claim (see Figure 6.14).

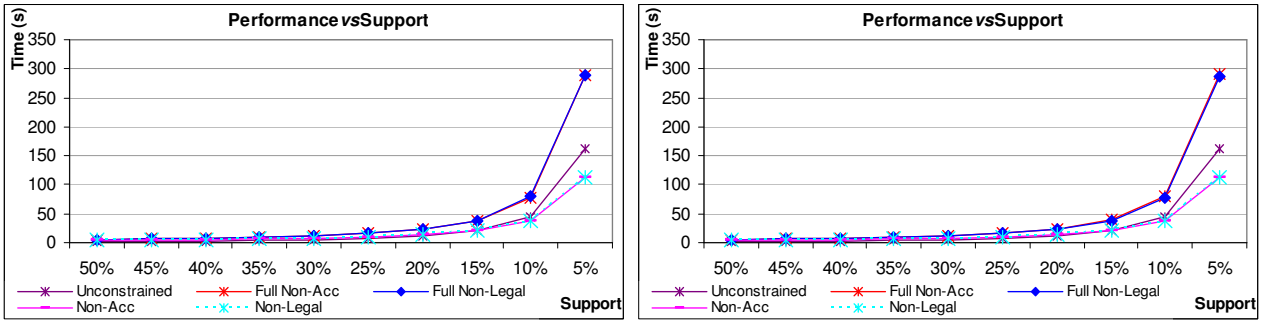


Figure 6.14 – Comparison of processing times of *Non-Accepted* variants with the DFA in Figure 6.4 (on the left) and with the ePDA in Figure 6.5 (on the right)

On one hand, the use of an item constraint in conjunction with non-accepted relaxations considerably diminishes the number of discovered patterns (between 50% and 60% in the conducted experiments – see Figure 6.15).

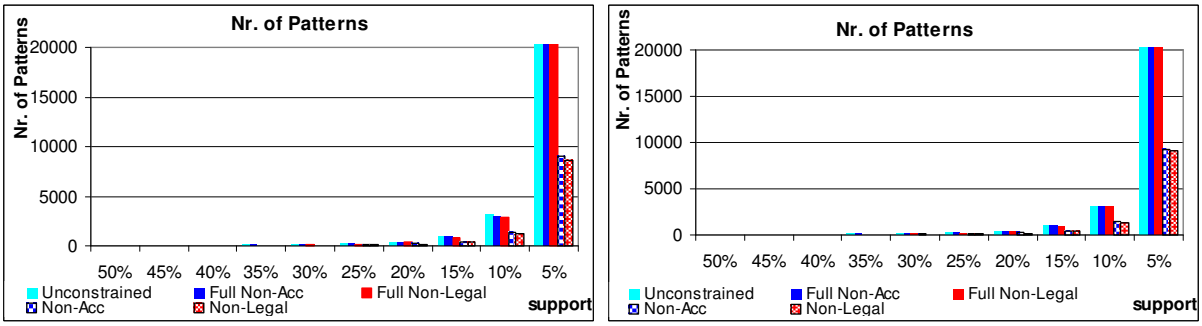


Figure 6.15 – Comparison of number of discovered patterns by *Non-Accepted* variants with the DFA in Figure 6.4 (on the left) and with the ePDA in Figure 6.5 (on the right)

On the other hand, *Non-Accepted* relaxations discover few patterns when the constraint is less restrictive. When using more restrictive constraints, the *Non-Legal* relaxation seems to be more interesting than the *Non-Accepted* one. The results also demonstrate that *Non-Accepted* relaxations are only useful when the constraint is not sufficiently restrictive.

In summary, the use of relaxations helps focusing the process on user expectations, discovering patterns that are closely related to existent background knowledge. Associated with the reduction of discovered patterns, is the reduction of processing times, more evident for conservative relaxations, but also possible for non-conservative ones, depending on the restrictive power of the constraint.

Summary

In this chapter, we have proposed a new pattern mining methodology, based on the use of constraint relaxations. A constraint relaxation imposes a weaker restriction, relaxing the constraint chosen to represent user background knowledge. We propose a relaxation for Ω*-constraints, which includes a* maximal time error *on the verification of the temporal constraint and a relaxation over the content constraint. The last one is based on the relaxation of its associated formal language, establishing weaker conditions to accept a sequence as valid. Experimental results show that the use of relaxations reduces the number of discovered patterns when compared to unconstrained processes, but allows for the discovery of unexpected patterns, when compared to constrained processes. The results also show that the processing times spent in the mining process can be reduced if constraint relaxations are used.*

Chapter 7
Case Studies

In this chapter, we present the results achieved by applying the algorithms and methodologies proposed in this dissertation, to real-life datasets. After reviewing the thesis statement, we show that these results validate our claims.

In this thesis we claim that it is possible to discover unknown information, using sequential pattern mining with constraint relaxations, without loosing the focus on user expectations, as exposed in Chapter 3. In order to validate these claims, we present the results achieved by applying this methodology to three real-life datasets: one resulting from the records of an online retail company – `PmeLink.PT`, and the other two resulting from the data collected from IST student's performance – `AMs1982-2001` and `LEIC1989-2001`.

In this manner, we evaluate our methodology by comparing:

- the performance of each mining process;
- the number of discovered patterns;
- the ability to answer some relevant questions.

1 – Applications

1.1 – PmeLink.PT

PmeLink.PT is the first Online Business Center in Portugal, supporting Portuguese small and medium enterprises (SMEs) in all of their business needs.

PmeLink.PT sells a large array of products, from furniture to small appliances, computer equipment and accessories. In order to test the applicability of the proposed approach in a real-world situation, we deployed a prototype integrated with the PmeLink.PT data warehouse. The prototype has as first goal to identify the frequent sequences of purchases, providing an insight on client's behavior and an aconselling mechanism, based on the past purchases of each client.

In this manner, it was possible to measure the performance and scalability of the approach on a real-life data (dataset `PmeLink.PT`). The experiments and data are described above.

Data Statistics

The dataset explored in this work is composed of 5383 sequences, corresponding to the purchases done by each PmeLink.PT client, in a period of two years.

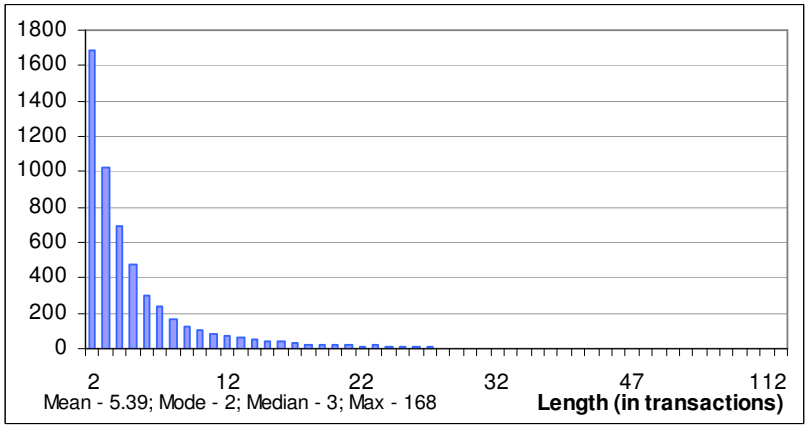


Figure 7.1 – Distribution of the length of sequences on the PmeLink.PT dataset

Unfortunately, a great portion of those sequences are very short (30% have two itemsets). Despite this, 89% of sequences have between 2 and 10 itemsets, with an average length of 5.39 itemsets, as illustrated in Figure 7.1.

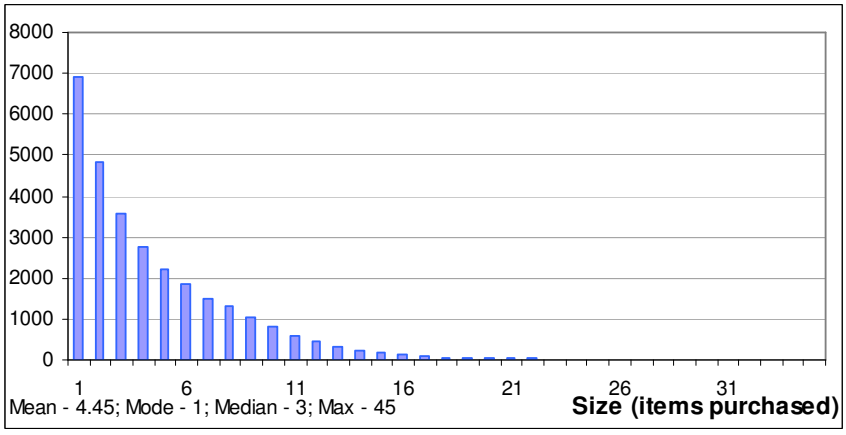


Figure 7.2 – Distribution of the size of the baskets in PmeLink.PTdataset

The average itemset has 4.45 items, with 24% of them with one item and 92% with between 1 and 10 items. There are 281 different categories of products, used as the alphabet in our exploration.

Finding purchase patterns

When explored with unconstrained sequential pattern mining, for low support thresholds, the data exploration discovers 1335 patterns with support above 5% and 219113 with support above 1%. These numbers hide any discovered information (see Table 8).

Table 8 – Number of discovered patterns with unconstrained mining

sup	$\|L_k\|$
45%	2
40%	2
35%	7
30%	12
25%	18
20%	31
15%	64
10%	203
5%	1.335
1%	219.113

However, the patterns discovered with larger support thresholds do not give any relevant information. For example, for 15% of support, 28 of the 64 discovered patterns have just one itemset, and the rest of patterns involve the purchase of `Paper A4`, which is the product most sold (present in 84% of the sequences). In this manner, the analysis of this data is very hard to perform if one doesn't use any background knowledge.

In order to gain an insight on the business of PmeLink.PT, consider the non-deterministic PDA illustrated in Figure 7.3.

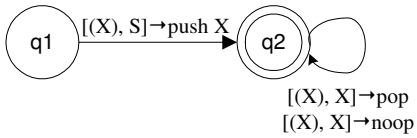


Figure 7.3 – nPDA for specifying constraints over the dataset PmeLink

This automaton accepts sequences composed of consecutive purchases of the same product. In fact, each transition on the nPDA represents several transitions, one per product. For example, the transition from q_1, represents one transition for each product, like the transitions [(staples),S]→push(staples) and [(clips),S]→push(clips), among others. We choose to use a pushdown instead of a deterministic finite automaton, since the last one must have one additional state per product, which would be somehow difficult to define and understand.

With this automaton, and constraining our exploration to Stock products, we are able to find 12 products that are consecutively purchased by at least 5% of clients, and 18 purchased by at least 1% of clients, as shown in Table 9.

Table 9 – Purchase patterns of Stock products

<(staples),(staples)>
<(clips),(clips)>
<(message-pads),(message-pads)>
<(dividers),(dividers)>
<(arch-lever files),(arch-lever files),(arch-lever files)>
<(correction-fluid),(correction-fluid)>
<(plastic covers),(plastic covers)>
<(ballpoint pens),(ballpoint pens)>
<(goods for original HP printers),(goods for original HP printers),(goods for original HP printers)>
<(disks and zips),(disks and zips)>
<(arch-lever files with box),(arch-lever files with box)>
<(goods for original Epson printers),(goods for original Epson printers)>

As can be seen, besides discovering which products are consecutively purchased, it is possible to discover the number of times that they are purchased.

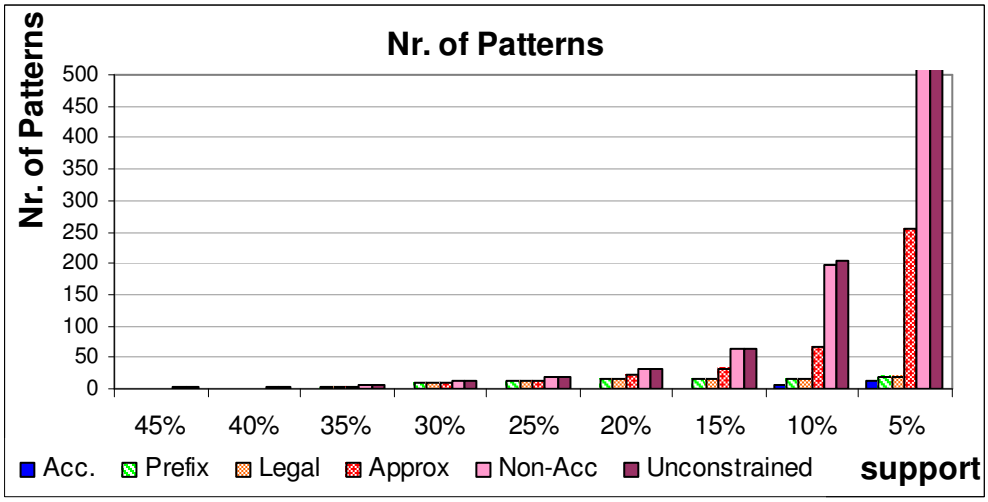


Figure 7.4 – Number of patterns discovered in the dataset PmeLink.PT

When using constraint relaxations, it is possible to go one-step further and discover other patterns of interest. Figure 7.4 shows that the use of constraint relaxations keeps the number of discovered patterns well below from the number of unconstrained patterns.

Unfortunately, this data is not rich enough to enable the discovery of more interesting patterns. Several other experiments were done, without any interesting result. Among those, experiments on analyzing different time periods (for example constraining the search to the end of the year, according to existing domain knowledge) were tried, without any other result.

1.2 – Analysis of Enforced Precedence between Subjects

The analysis of the curriculum of undergraduate programs and the corresponding student's performance is a non-trivial task. For example, the garanty that some subject is placed in the right semester is difficult to achieve. The difficulties are mainly related to the curriculum structure, which consists of a sequence of sets of simultaneous subjects, and to the number of possible curricula instantiations. Definitely, less than 50% of the students follow the established curriculum, choosing a different sequence of subjects.

In the last years, the undergraduate programs at *Instituto Superior Técnico* (*IST*) have introduced the existence of enforced precedences between subjects. An immediate consequence of this approach is the avoidance of that variety of curricula, making that a student can only attend to a subject after have concluded another preceding one.

However, despite the theoretical benefits of this approach, it disables that students create their own curriculum, in accordance to their preferences and capabilities. Indeed, a detailed analysis of the different instantiations of the same curriculum model, demonstrates that there are several different ways to achieve the same results within the same time period.

Analysis Goals

In this section, we aim to analyze the effects of the imposition of precedences among the subjects on *Mathematical Analysis* (AM) at *IST*, made in the academic year of 1994/1995.

In this manner, we will try:

- To identify the frequent sequence of results achieved in the subjects on *Mathematical Analysis*, before and after the academic year of 1994/1995.
- To identify the sequence of results achieved by students who conclude all those subjects between four and six semesters.

Data statistics

The dataset used to analyze these questions consists on the set of sequences corresponding to the sequence of results on Mathematical Analysis subjects achieved by each *IST* student, between 1982 and 2001, excluding the students that do not enrolled on those subjects at least for four semesters.

The dataset (`AMs1982-2001`) contains 17933 sequences, corresponding to 17933 students from the different undergraduate programs at *IST*. Since there are four required subjects on

Mathematical Analysis, the alphabet is composed by 8 symbols (one for each successful enrollment – X, and one for each failure – ~X). Most of the students (69%) have between 4 and 6 enrollments (they had attended classes from those subjects between 4 and 6 semesters), which leads to an average sequence length equal to 6.29 semesters (see Figure 7.5).

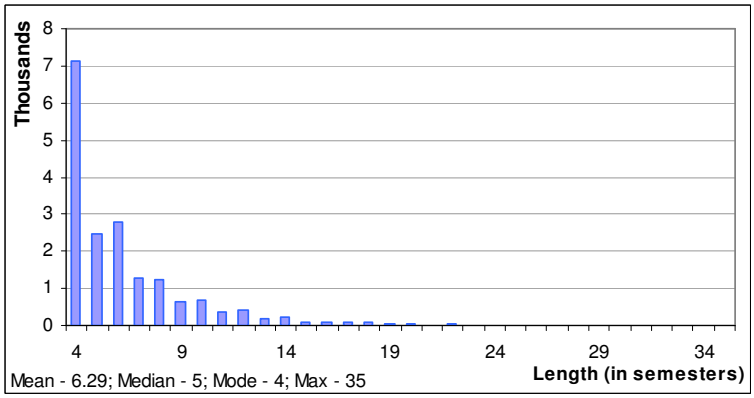


Figure 7.5 – Distribution of the length of sequences on dataset AMs1982-2001

Figure 7.6 shows that the great majority of students attended one mathematical analysis subject per semester. In this manner, the average size of itemsets is equal to 1.14.

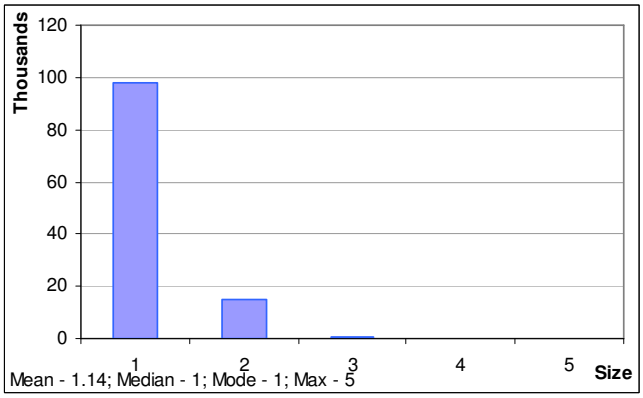


Figure 7.6 – Distribution of the number of enrollments per semester in AMs1982-2001 (size of itemsets)

Unfortunately, the data is non-equally distributed by the time intervals of interest: 66% of students attended classes before 1994/95 and only 34% after that year (Figure 7.7).

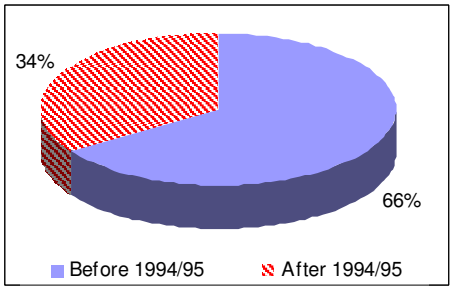


Figure 7.7 – Distribution of students per time interval

Finding Frequent Patterns of Results

A first analysis, using unconstrained sequential pattern mining, leads to the discovery of several patterns (corresponding to sub-patterns of the entire sequence of mathematical subjects), like the ones shown in Table 10.

Table 10 – Patterns discovered on dataset AMs1982-2001, for 10% of support

Discovered Patterns for 10% of support		
<(AM1),(AM2),(AM3),(AM4)>	<(AM1),(AM2),(~AM3)>	<(AM1),(~AM2),(~AM3)>
<(AM3),(~AM4)>	<(~AM2),(AM2)>	<(~AM2),(AM3)>
<(~AM2),(~AM2)>	<(~AM2),(~AM3),(~AM4)>	<(~AM3),(AM2)>
<(~AM3),(AM3)>	<(~AM3),(AM4)>	<(~AM3),(~AM2)>
<(~AM3),(~AM3)>	<(~AM3),(~AM4),(AM3)>	<(~AM3),(~AM4),(~AM3)>
<(~AM1),(AM1)>	<(~AM1),(~AM2),(AM1)>	<(~AM1),(~AM2),(~AM1),(~AM2)>
<(~AM1),(~AM1)>	<(~AM4),(AM4)>	

From these results, we are only able to say that there are several different patterns of failure, and that the patterns involving successful results are only frequent when considering very low support thresholds. Note that for lower minimum support thresholds, this technique will discover several rules (for 1% it will discover 221 patterns).

In order to discover how students conclude the four subjects on Mathematical Analysis, we propose the use of two different DFAs that represent the flow of results that students achieve in four semesters. Since we want to compare the results before and after the introduction of precedences, we are going two use two different automata.

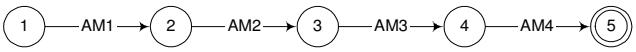


Figure 7.8 – DFA for representing the results on Mathematical Analysis subjects in 4 semesters, after 1994

Figure 7.8 shows that in the presence of precedences, there is only one way to conclude all subjects in four semesters, which correspond to not failing any subject. In this manner, the DFA only accepts the sequence `<(AM1)(AM2)(AM3)(AM4)>`.

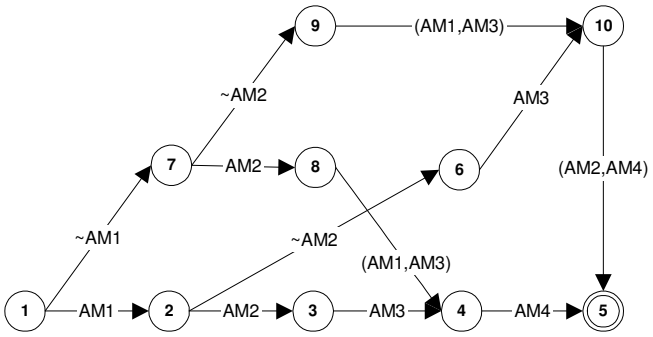


Figure 7.9 – DFA for representing the results on Mathematical Analysis subjects in 4 semesters, before 1994

Without precedences, and knowing that AM1 and AM3 function on the Fall semester and AM2 and AM4 on the Spring semester, there are four ways to conclude all subjects in four semesters:

<(AM1)(AM2)(AM3)(AM4)>, <(~AM1)(AM2)(AM1,AM3)(AM4)>, <(AM1)(~AM2)(AM3)(AM2,AM4)> and <(~AM1)(~AM2)(AM1,AM3)(AM2,AM4)>, as shown in Figure 7.9.

Using a small value for the support (say 0.01%) and two Ω–constraints, defined over the two automata above and with an adequate temporal constraint, we discover that:

- Before 1994/1995, **27.5**% of students were able to conclude all subjects without failing, and **5.5**% of students were also able to conclude all subjects in 4 semesters. This results in **33**% of students concluding all subjects on 4 semesters.
- After 1994/1995, **32**% of students are able to conclude all subjects without failing.

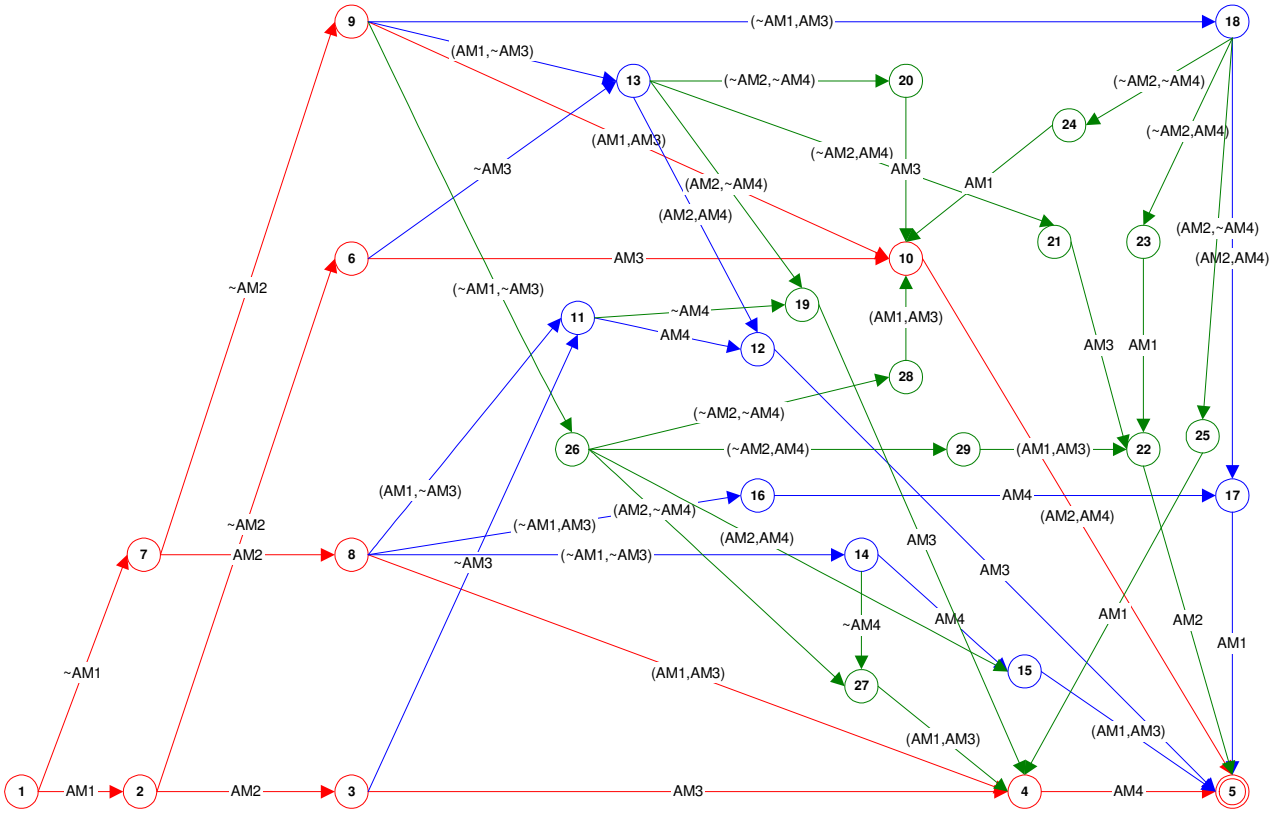


Figure 7.10 – DFA for representing results on Mathematical Analysis subjects in 6 semesters, before 1994

If we perform an identical analysis for 6 semesters (using the constraints defined over the DFA shown in Figure 7.10 and Figure 7.11) we conclude that:

- Additionally, before 1994/1995, **5**% of students were able to recover and conclude all subjects in 5 semesters, and **6**% in 6 semesters, which result on **44**% of students concluding all subjects on at least 6 semesters.
- After 1994/1995, 1**3.5**% of students are able to conclude all subjects in 5 semesters and **10**% in 6 semesters, which result on **55.5**% of students concluding all subjects on at least 6 semesters.

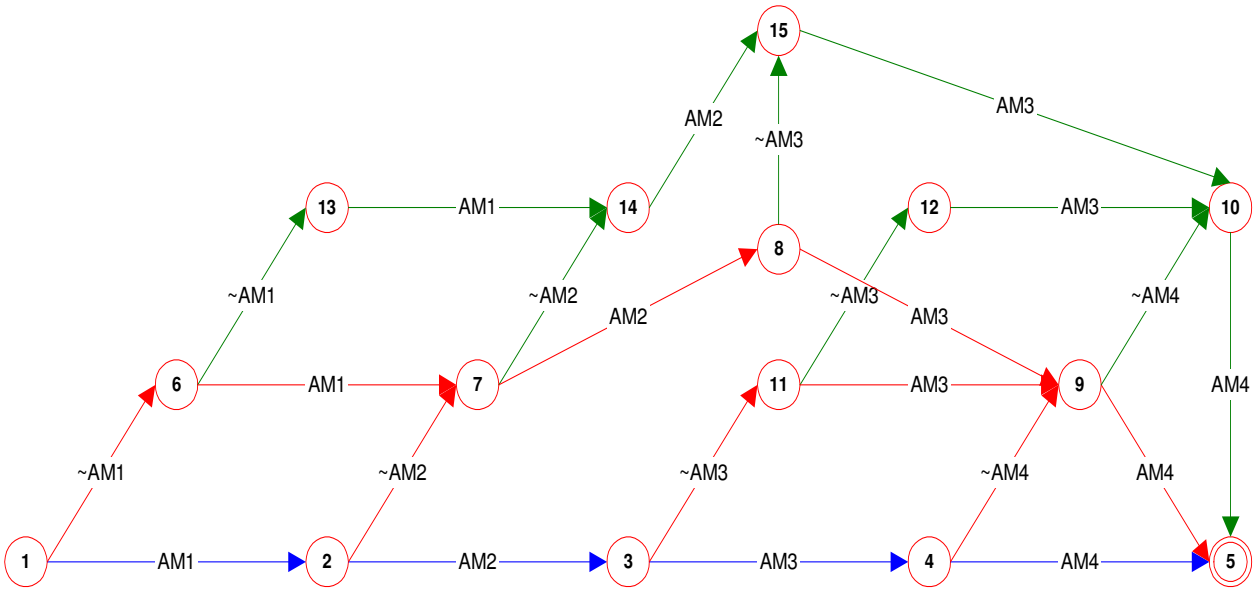


Figure 7.11 – DFA for representing the results on Mathematical Analysis subjects in 6 semesters, after 1994

These results show that, in fact, the imposition of precedences among the subjects on Mathematical Analysis introduces some alterations on students' behavior. On one side, the percentage of students that conclude all subjects in 6 semesters increase about 10%, but on the other side, the number of students that conclude on 4 semesters decreases 1%. 1% is a little difference, but students curricula do not show some frequent behaviors, like `<(AM1)(~AM2)(AM3)(AM2,AM4)>`, `<(AM1)(AM2)(~AM3)(AM4)(AM3)>` or `<(~AM1)(AM2)(AM1,AM3)(AM4)>`.

It is important to note that there is no trivial way to perform an identical analysis. Usually, the results are stored in separate records, difficulting the sequential analysis with simple queries. Remember that those queries require several natural joins, one for each constraint on the required sequence of results. In fact, the queries must define the entire automata with those operations, which is not a simple task. No other kind of queries will be able to address this problem, because without the sequence information, we are just able to discover how many students have approved or failed in each subject.

In this manner, constrained sequential pattern mining is a natural way to perform this kind of analysis, only requiring that "experts"design the possible curricula, which is trivial, since they are publicly known.

1.3 – Analysis of LEIC Students' Behaviours

Licenciatura em Engenharia Informática e de Computadores (LEIC) is an undergraduate program on information technology and computer science at *Instituto Superior Técnico*, created in the academic year of 1989/90.

The LEIC curriculum has a duration of five years (10 semesters) with 36 required subjects, a final thesis and 4 optional subjects in the last year. Since 1989, the curriculum model of LEIC has suffered two main reorganizations. Until 2003, LEIC has offered four specialty areas: PSI – Programming and Information Systems; SCO – Computer Systems; IAR – Artificial Intelligence and IIN – Information Systems for Factory Automation. In the first curriculum model, LEIC had 20 common subjects, 18 on the first three semesters and the other 2 on the following two semesters. The enrollment on a specialty area was made on the fourth semester.

The distribution of common subjects per scientific area is shown on Table 11.

Table 11 – List of common subjects, distributed by scientific area

Common subjects	
Mathematics	AL – Linear Algebra
	AM1 – Mathematical Analysis 1
	AM2 – Mathematical Analysis 2
	AM3 – Mathematical Analysis 3
	PE – Probability and Statistics
	AN – Numerical Analysis
Physics	FEX – Experimental Physics
	F1 – Physics 1
	F2 – Physics 2
Computer Science	TC – Theory of Computation
Programming Methodologies	IP – Programming Introduction
	AED – Algorithms and Data Structures
	PLF – Logical and Functional Programming
	POO – Object Oriented Programming
Architecture and Operative Systems	SD – Digital Systems
	AC – Computer Architecture
	SO – Operative Systems
Artificial Intelligence	IA – Artificial Intelligence
Information Systems	SIBD – Information Systems and Databases
Computer Graphics	CG – Computer Graphics

There are 47 other subjects, distributed by each specialty area. The deterministic finite automaton on Figure 7.12 shows the model curriculum for each specialty area.

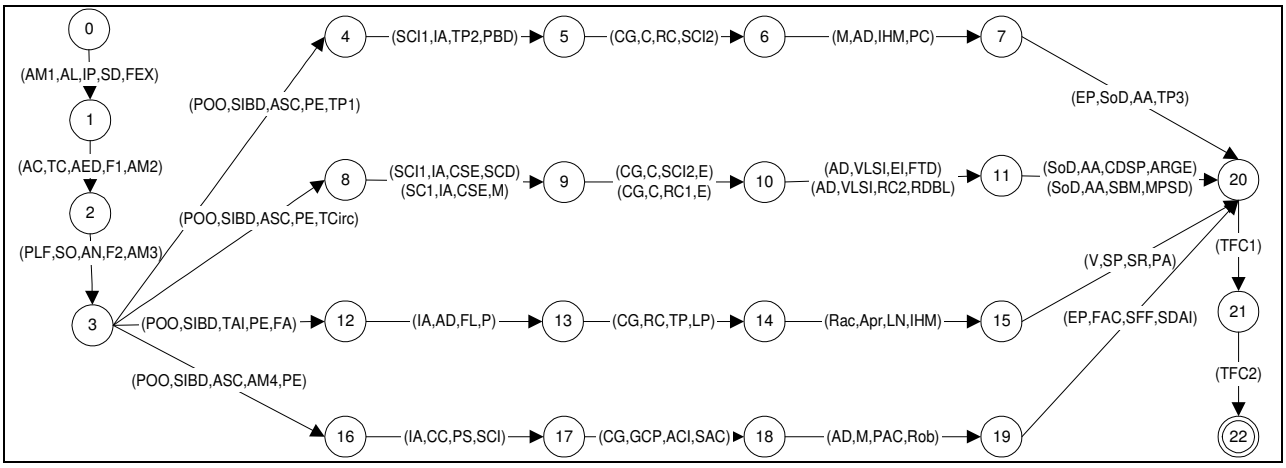


Figure 7.12 – DFA for specifying the model curriculum for LEIC specialty areas

The existence of two different transitions per semester for SCO students, are due to a minor reorganization of the SCO curriculum on 1995/1996.

Analysis Goals

The analysis of the data referring to LEIC students' performance has essentially two main reasons: to explain the low levels of success and to identify the most common profiles of LEIC students.

In this manner, we will try to answer two questions:

- 'What are the most common patterns on each scientific area, and what is the impact of some subjects on others?'
- 'What are the common curricula instantiations for each specialty area, including optional subjects?'

Data statistics

The dataset used to analyze those questions consists on the set of sequences corresponding to the curriculum followed by each LEIC student, with his first enrollment made between 1989 and 1997. From these, the students that do not enrolled in LEIC subjects more than seven semesters were excluded (this removes 260 students out of 1706), since most of them have cancelled their registration. Additionally, it is important to note that students that enrolled in 1997 have not concluded their graduation until 2001, but have been registered for at least eight semesters, and therefore may have concluded the 4th curricular year.

In this manner, the dataset (`LEIC1989-2001`) is composed of 1440 sequences, with an average sequence length equal to 11.58 semesters. Most of the students (72%) have between 8 and

12 enrollments (they had attended classes between 8 and 12 semesters) – see Figure 7.13.

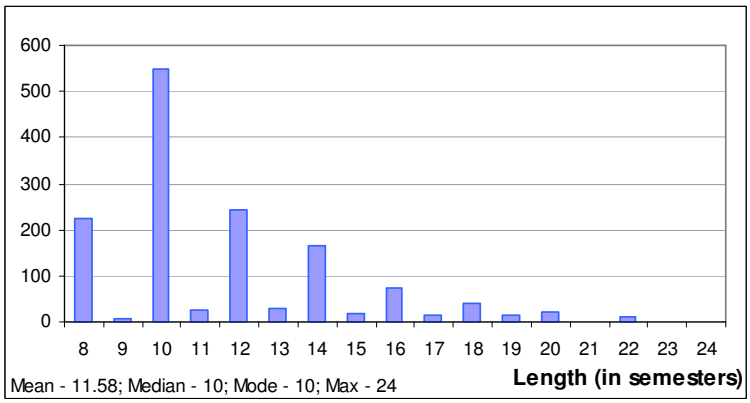


Figure 7.13 – Distribution of the length of sequences on dataset LEIC1989-2001

Naturally, the number of students with an odd sequence length is reduced, since this situation corresponds to students that have registered in only one semester on that year.

In terms of the number of enrollments per semester, its mean is 4.82 enrollments on subjects per semester, with most students (75%) enrolling on between 4 and 6 units (see Figure 7.14).

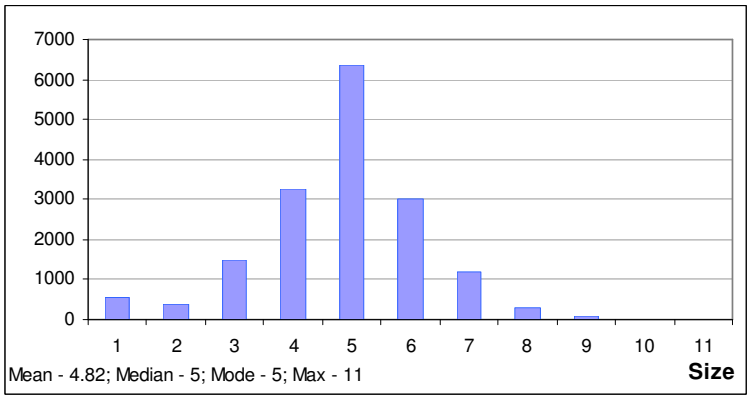


Figure 7.14 – Distribution of the number of enrollments per semester in LEIC1989-2001 (size of itemsets)

Another interesting issue is the distribution of students per specialty area, shown in Figure 7.15.

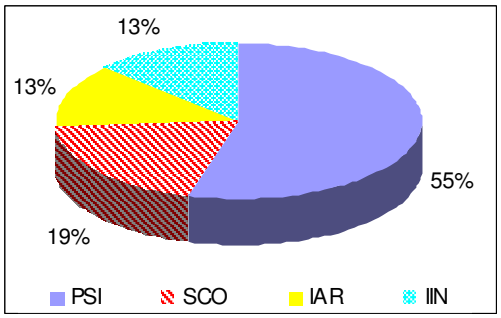


Figure 7.15 – Students per specialty area

This distribution conditions the number of enrollments per subject. For example, subjects exclusive to Artificial Intelligence and IIN have at most 13% of support (Figure 7.16, Figure 7.17).

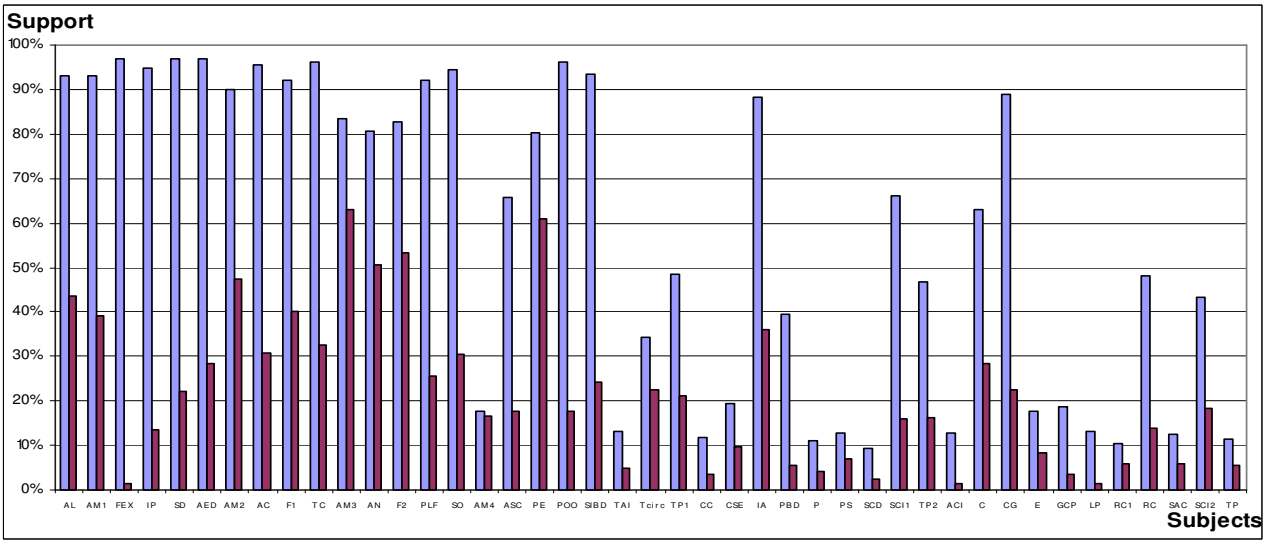


Figure 7.16 – Support for approvals and failures (subjects in the first 6 semesters of the model curriculum)

It is interesting to note that only 823 students (57%) have concluded the final work (`TFC1` and `TFC2`). Since it is usual that students only took optional subjects in parallel or after finishing the final work, the support for optional subjects is at most 57%. Since the options are chosen from a large set of choices (130 subjects), their individual support is considerably lower. Indeed the subject on Management (G) is the optional subject with more students, about 40%.

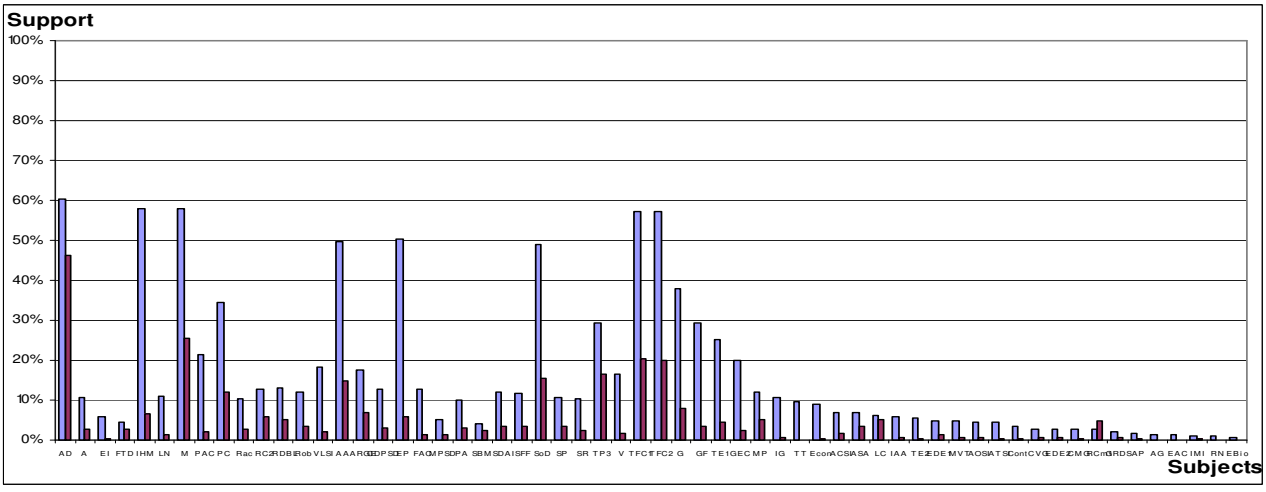


Figure 7.17 – Support for approvals and failures (subjects in the last 4 semesters of the model curriculum)

Finding frequent curricula on scientific areas

The discovery of the most common patterns on each scientific area is easily achieved if we look for students, who conclude the sequence of subjects in the same scientific area in at most four

semesters. This constraint can be specified by the pushdown automaton represented on Figure 7.18.

In this figure, each transition represents four transitions, one per scientific area. For example, `[(~X2),sa(X2)]→push(sa(X2))` represents `[(~F1),F]→push(F)`, `[(~AM2),AM]→push(AM)` `[(~AED),MTP]→push(MTP)`, `[(~AC),ASO]→push(ASO)`.

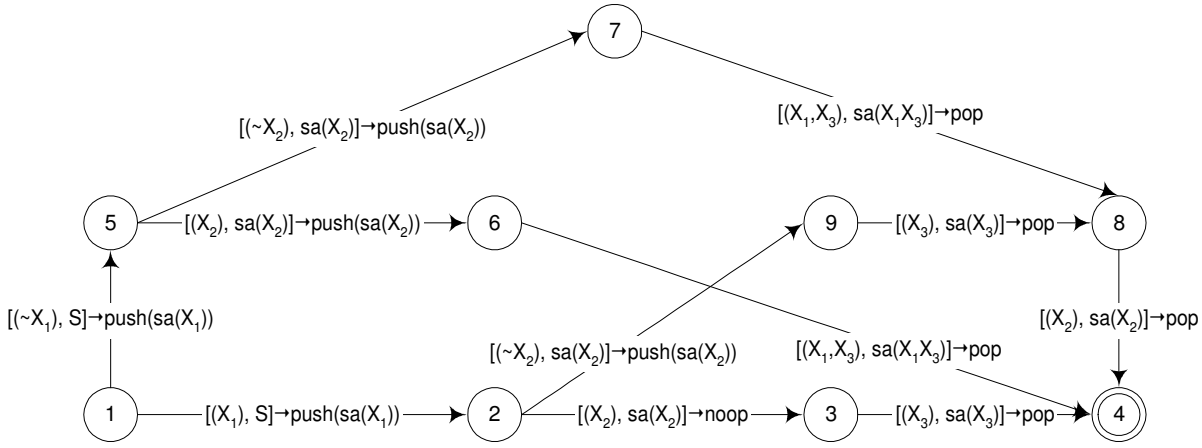


Figure 7.18 – PDA for specifying the curricula on each scientific area, where students conclude three subjects in a determined scientific area at most on 4 semesters

Also, consider that X_1, X_2 and X_3 are the first, second and third subjects on some of the following scientific areas: `MTP`, `ASO`, `Physics` and `Mathematical Analysis`. Also, consider that `sa` is a function from the set of subjects to their scientific area, for example `sa(IP)=MTP`.

The first thing that we are able to discover, using an Ω–constraint defined over the PDA on Figure 7.18, with a gap equal to zero and without any other temporal constraint, is that the majority of students are able to conclude the sequence of MTP (61%) and ASO (57%) subjects without any failure. Additionally, 6% of students are also able to conclude all but one of those subjects in four semesters (see shadowed patterns in Table 12).

Table 12 – Patterns in Scientific Areas, discovered with an Ω–constraint

Patterns	Sup
<(IP),(AED),(PLF)>	61%
<(IP),(~AED),(PLF),(AED)>	6%
<(SD),(AC),(SO)>	57%
<(SD),(~AC),(SO),(AC)>	6%
<(FEX),(F1),(F2)>	35%
<(FEX),(~F1),(F2),(F1)>	5%
<(AM1),(AM2),(AM3)>	31%

When applying an *Approx* relaxation, we are able to discover part of the patterns followed by students that are not able to conclude all subjects in four semesters, as specified in the previous automaton. For example, one of the causes of failure on the sequence of ASO subjects is failing on the subject of Operative Systems (SO). Since this subject is the third one in the sequence and it is only offered in the *Fall* semester, students in that situation are not able to conclude the three

subjects in four semesters. Similarly, students that fail on the subjects of Physics 2 (F2), Mathematical Analysis 3 (AM3) and Functional and Logic Programming (PLF) are not able to conclude the corresponding sequences on 4 semesters, as shown in Table 13.

Table 13 – Patterns in a single scientific area discovered with the Approx relaxation

Patterns	Sup
<(AM1),(~AM2),(~AM3),(AM2)>	6%
<(AM1),(~AM2),(AM2)>	6%
<(AM1),(AM2),(~AM3,AM3)>	6%
<(FEX),(F1),(~F2)>	22%
<(FEX),(~F1),(~F2),(F1)>	8%
<(FEX),(~F1),(F2),(F1)>	5%
<(~F2),(F1),(F2)>	6%
<(IP),(AED),(~PLF)>	12%
<(IP),(~AED),(PLF),(POO)>	6%
<(~IP),(~AED),(IP),(AED)>	5%
<(SD),(AC),(~SO)>	13%
<(~SD),(~AC),(SD),(AC)>	6%

Another interesting pattern found is that 6% of students fail in the first opportunity to conclude Mathematical Analysis 3 (AM3), but seize the second opportunity, concluding that subject in the first enrollment (shadowed line in Table 13).

Additionally, the use of the approx relaxation also contributes to analyze the impact of some subjects on others. For example, an *approx* relaxation with one error discovers that 49% of the students that conclude MTP subjects in 3 semesters fail on AM3 and 40% on F2. Similarly, 45% of students that conclude ASO subjects in 3 semesters fail on AM3 and 39% on F2 (shadowed patterns in Table 14).

Table 14 – Patterns in scientific areas with one error

Patterns	Sup	Patterns	Sup
<(IP,SD),(AED),(PLF)>	57%	<(IP,~SD),(AED),(PLF)>	8%
<(IP),(AED),(PLF,SO)>	53%	<(IP),(AED),(PLF,~SO)>	10%
<(IP),(AED,AC),(PLF)>	53%	<(IP),(AED,~AC),(PLF)>	8%
<(SD),(AC),(PLF,SO)>	51%	<(SD),(AC),(~PLF,SO)>	8%
<(IP,AM1),(AED),(PLF)>	50%	<(IP,~AM1),(AED),(PLF)>	15%
<(SD,AM1),(AC),(SO)>	47%	<(SD,~AM1),(AC),(SO)>	15%
<(IP),(AED,AM2),(PLF)>	45%	<(IP),(AED,~AM2),(PLF)>	16%
<(IP),(AED,F1),(PLF)>	45%	<(IP),(AED,~F1),(PLF)>	16%
<(SD),(AC,F1),(SO)>	41%	<(SD),(AC,~F1),(SO)>	16%
<(SD),(AC,AM2),(SO)>	41%	<(SD),(AC,~AM2),(SO)>	16%
<(IP),(AED),(PLF,F2)>	**36%**	**<(IP),(AED),(PLF,~F2)>**	**24%**
<(SD),(AC),(SO,F2)>	**34%**	**<(SD),(AC),(SO,~F2)>**	**23%**
<(FEX,AM1),(F1),(F2)>	31%	<(~AM1,FEX),(F1),(F2)>	6%
<(FEX),(F1),(SO,F2)>	31%	<(FEX),(F1),(~SO,F2)>	5%
<(IP),(AED),(PLF,AM3)>	**29%**	**<(IP),(AED),(PLF,~AM3)>**	**30%**
<(FEX),(F1,AM2),(F2)>	28%	<(FEX),(F1,~AM2),(F2)>	7%
<(SD),(AC),(SO,AM3)>	**27%**	**<(SD),(AC),(SO,~AM3)>**	**26%**
<(AM1),(AM2),(F2,AM3)>	23%	<(AM1),(AM2),(~F2,AM3)>	8%
<(FEX),(F1),(F2,AM3)>	21%	<(FEX),(F1),(F2,~AM3)>	14%

Finding Common Curricula Instantiations with Optional Subjects

The challenge on finding which students choose what optional subjects is a non-trivial task, especially because all non-common subjects can be chosen as optional by some student. A simple

count of each subject support does not give the expected answer, since most of the subjects are required to some percentage of students.

The other usual approach would be to query the database to count the support of each subject, knowing that students have followed some given curriculum. However, this approach is also unable to answer the question, since a considerable number of students (more than 50%) have failed one or more subjects, following a slightly different curriculum.

In order to discover the optional subjects frequently chosen by students, we have used the methodology previously proposed – the use of constraint relaxations, defining an Ω–constraint based on the deterministic finite automaton shown in Figure 7.19.

This automaton accepts sequences that represent the curricula on the fourth curricular year for each specialty area (the first for PSI students, the second one for SCO, the third for IAR and the fourth for IIN). In practice, an Ω–constraint defined over this DFA filters all patterns that do not respect the model curriculum for the last two curricular years.

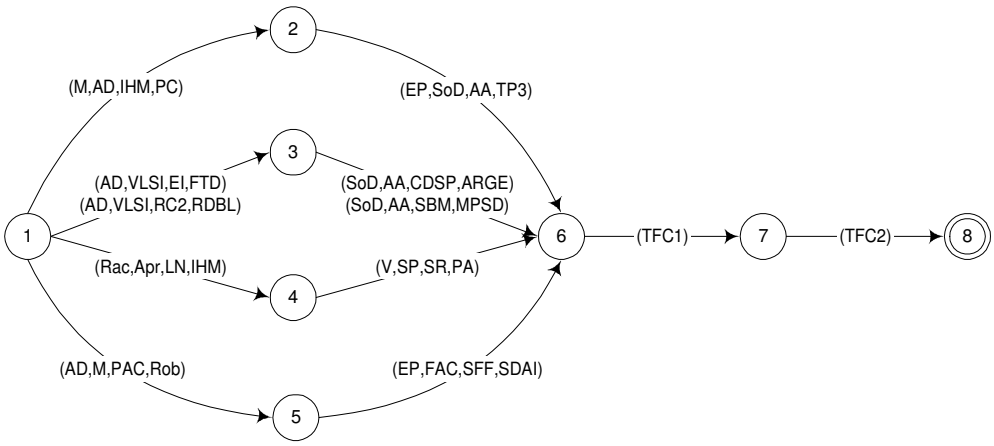


Figure 7.19 – DFA for finding optional subjects

The use of constrained sequential pattern mining (with the specified constraint) would not contribute significantly to answer the initial question, since it would only achieve results similar to the ones obtained by the query explained above.

However, the use of the *Approx-Accepted* relaxation allows for the discovery of several patterns. If the relaxation accepts at most two errors (ε=2) chosen from a restricted alphabet, composed by every non-common subject, we are able to found the frequent curricula instantiations with optional subjects. In general, students mostly attend Computer Graphics (`PAC-Computed Assisted Project; IHM-Human Machine Interfaces`) and Management subjects (`Economy-E; Economical Theory 1-TE1; Financial Management-GF; Management-G; Management`

`Introduction-IG`), as shown in Table 15.

Table 15 – Patterns with optional subjects attended by LEIC students

Specialty Area	Curricula Instantiations	LEIC support	Specialty Support
PSI	<(M, AD, IHM, PC) (AA, SoD, EP, TP3) (TFC1, **PAC**) (TFC2, **GF**)>:22	1.5%	5%
	<(M, AD, IHM, PC) (AA, SoD, EP, TP3) (TFC1, **Econ**) (TFC2, **GF**)>:33	2.5%	8%
	<(M, AD, IHM, PC) (AA, SoD, EP, TP3) (TFC1, **Econ**) (TFC2, **IG**)>:28	2%	7%
	<(M, AD, IHM, PC) (AA, SoD, EP, TP3) (TFC1) (TFC2, **GF**, **IG**)>:33	2.5%	8%
	<(M, AD, IHM, PC) (AA, SoD, EP, TP3) (TFC1, **G**, **PAC**) (TFC2)>:22	1.5%	5%
	<(M, AD, IHM, PC) (AA, SoD, EP, TP3) (TFC1, **G**) (TFC2, GF)>:32	2%	8%
	<(M, AD, IHM, PC) (AA, SoD, EP, TP3) (TFC1, **G**) (TFC2, GCP)>:19	1%	5%
	<(M, AD, IHM, PC) (AA, SoD, EP, TP3) (TFC1, **G**) (TFC2, GEC)>:23	1.5%	5%
	<(M, AD, IHM, PC) (AA, SoD, EP, TP3) (TFC1, **G**) (TFC2, ARGE)>:18	1%	4%
	<(M, AD, IHM, PC) (AA, SoD, EP, TP3) (TFC1, **G**, **TE1**) (TFC2)>:29	2%	7%
	<(M, AD, IHM, PC) (AA, SoD, EP, TP3) (TFC1, **TE1**, **Econ**) (TFC2)>:21	1.5%	5%
	<(M, AD, IHM, PC) (AA, SoD, EP, TP3) (TFC1, **TE1**) (TFC2, **GF**)>:44	3%	10%
	<(M, AD, IHM, PC) (AA, SoD, EP, TP3) (TFC1, **TE1**) (TFC2, **IG**)>:27	1,5%	6%
	<(M, AD, IHM, PC) (AA, SoD, EP, TP3) (**TE1**) (**GEC**)>:15	1%	4%
	<(M, AD, IHM, PC) (AA, SoD, EP) (TFC1) (TFC2, **TE2**)>:15	1%	4%
IIN	<(M, AD, PAC, Rob) (EP, SDAI, SFF) (TFC1) (TFC2, **IG**)>:15	1%	8%
	<(M, AD, PAC, Rob) (EP, FAC, SDAI, SFF) (TFC1, **IHM**) (TFC2, **GF**)>:18	1%	10%
	<(M, AD, PAC, Rob) (EP, FAC, SDAI, SFF) (TFC1, **Econ**) (TFC2, **GF**)>:18	1%	10%
	<(M, PAC, Rob) (EP, FAC, SDAI, SFF) (TFC1, **G**) (TFC2)>:17	1%	10%
	<(M, PAC, Rob) (EP, FAC, SDAI, SFF) (TFC1, **TE1**) (TFC2)>:18	1%	10%
IAR	<(IHM, Rac, LN) (SP, V, PA, SR) (TFC1, **TE1**) (TFC2)>:15	1%	10%
	<(IHM, A, Rac, LN) (SP, V, PA, SR) (TFC1) (TFC2, **GF**)>:17	1%	10%
SCO	<(C) (VLSI, RC2, RDBL, AD) (AA, CDPSD, ARGE, SoD) (TFC1, **G**) (TFC2)>:23	1.5%	8%
	<(Elect) (VLSI, RC2, RDBL, AD) (AA, CDPSD, ARGE, SoD) (TFC1, **G**) (TFC2)>:27	1.5%	10%
	<(RC1) (VLSI, RC2, RDBL, AD) (AA, CDPSD, ARGE, SoD) (TFC1, **G**) (TFC2)>:19	1%	7%

It is interesting to note that whenever IIN students have failed on some subject on the 4th year, they choose an optional subject in Economy (`TE1` or `IG`). The same happens for PSI and IAR students (behavior identified by shadowed rules). Note that in order to discover these rules, we have to be able to admit some errors on the sequence of subjects per specialty area, which is not easily done by specifying some query to a database.

Another interesting issue is the inexistence of frequent optional subjects among IAR and SCO students. Indeed, for the last ones there is only one frequent optional subject (Management – G).

Finding Artificial Intelligence curricula

As can be seen in previous analysis, the subjects exclusive to AI students have very low supports (about 13%). Naturally, the sequences of consecutive subjects have supports even lower. Indeed, the discovery of Artificial Intelligence (IAR) frequent curricula, like for IIN, is non-trivial, since the number of students in these specialty areas is reduced.

Given that the application of unconstrained sequential pattern mining algorithms found 5866 patterns in this dataset (for 20% of support, since we were not able to try lower supports due the memory requirements), and we want to found the sequence of subjects followed by IAR students, the use of constrained or unconstrained sequential pattern mining does not help to answer this second question.

However, if we use the proposed methodology, we have two alternatives: using a DFA specifying the IAR model curriculum and an *Approx-Accepted* relaxation as above, or using a DFA specifying PSI and SCO curricula models and a *Non-Accepted* relaxation with a restricted alphabet.

Table 16 – Artificial Intelligence frequent curricula

Patterns on Artificial Intelligence	
<(AD,FL,P)(RC,TP)(IHM,Apr,Rac,LN)(SP,PA)>	<(FA,TAI)(AD,P)(RC,LP,TP)(IHM,Apr,Rac,LN)(SP,V,SR)>
<(FA,TAI)(FL,P)(RC,LP,TP)(IHM,Apr,Rac,LN)(SP,V,PA,SR)>	<(FA,TAI)(AD,P)(RC,LP)(IHM,Apr,Rac,LN)(SP,V,PA,SR)>
<(FA,TAI)(AD,FL,P)(RC,LP,TP)(IHM,Rac,LN)(SP,V,SR)>	<(FA)(AD,FL,P)(RC,TP)(IHM,Apr,Rac,LN)(SP)>
<(FA,TAI)(AD,FL,P)(RC,LP,TP)(IHM,Apr,Rac,LN)(V,PA,SR)>	<(FA)(AD,FL,P)(RC)(IHM,Apr,Rac,LN)(SP,PA)>
<(FA,TAI)(AD,FL,P)(RC,LP)(IHM,Rac,LN)(SP,V,PA,SR)>	<(TAI)(AD,P)(RC,LP,TP)(IHM,Apr,Rac,LN)(SP,V,PA,SR)>
<(FA,TAI)(AD,FL,P)(RC,LP)(IHM,Apr,Rac,LN)(SP,V,SR)>	<(TAI)(AD,FL,P)(RC,LP,TP)(IHM,Rac,LN)(SP,V,PA)>
<(FA,TAI)(AD,FL)(RC,LP,TP)(IHM,Apr,Rac,LN)(SP,V,PA,SR)>	<(TAI)(AD,FL,P)(RC,LP,TP)(IHM,Apr,Rac,LN)(SP,V,SR)>
<(FA,TAI)(AD,P)(RC,LP,TP)(IHM,Rac,LN)(SP,V,PA,SR)>	<(TAI)(AD,FL,P)(RC,LP)(IHM,Apr,Rac,LN)(SP,V,PA,SR)>

The patterns discovered by the second alternative (using a minimum support threshold equal to 2.5%) answer the question. (Table 16 shows the discovered patterns, excluding 8 patterns that are shared by PSI students).

Note that we were not able to find the entire model curriculum for Artificial Intelligence, because of the reduced number of students that have concluded each subject on the first enrollment. This fact, explains the number of discovered patterns (24). However, it is smaller than the number of patterns discovered with an unconstrained approach, confirming our claim about constraint relaxations.

The Non-Accepted relaxation was defined using a constraint similar to the previous one with the DFA represented in Figure 7.20, which corresponds to a DFA equal to the one presented in Figure 7.12, without the model curriculum for IAR and IIN.

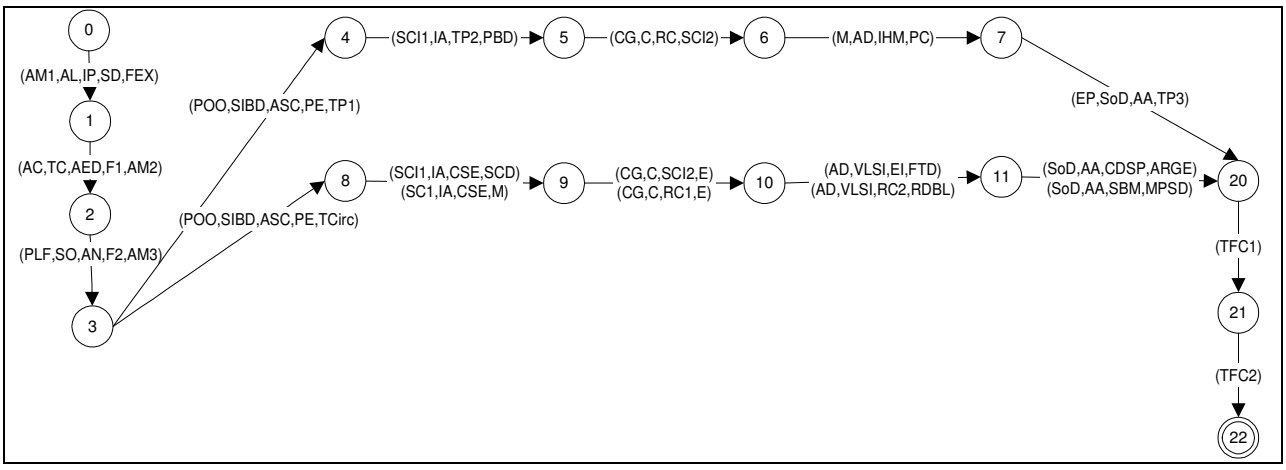


Figure 7.20 – DFA for discovering IAR frequent curricula

Additionally, the relaxation alphabet was composed of all the common subjects and the advanced subjects specific to the Artificial Intelligence specialty area (38 subjects).

2 – Efficiency Evaluation

As pointed in [Zheng 2001], real-world datasets can be drastically different from synthetic datasets. In this section, the real-world datasets `PmeLink.PT` and `LEIC1989-2001` are used to validate the efficiency studies performed in previous chapters.

First, we will compare the performance of *GenPrefixSpan* and *SPaRSe*, verifying if they show similar behaviors in dense datasets. Then, we will compare the efficiency of constrained and unconstrained sequential pattern mining. Finally, we assess the efficiency of the usage of constraint relaxations.

2.1 – Comparison between *GenPrefixSpan* and *SPaRSe*

As seen in Chapter 4, *SPaRSe* and *GenPrefixSpan* show a similar performance in the presence of dense datasets (remember Figure 4.4). The experiments on real-life datasets, previously described, confirm those results and emphasize the differences in favor of *SPaRSe*, as shown in Figure 7.21.

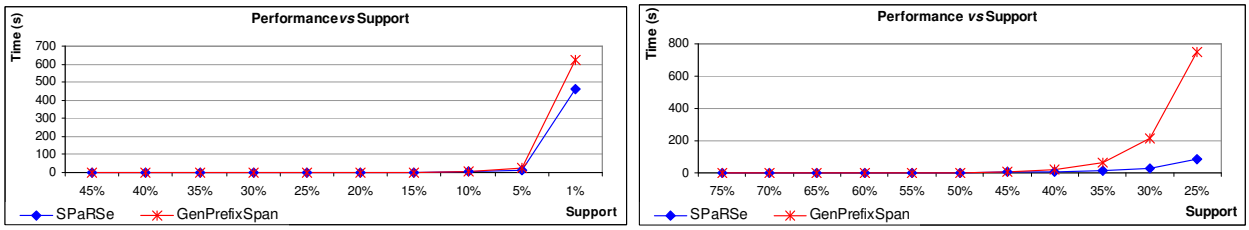


Figure 7.21 – Performance w/ variable support in dataset `PmeLink.PT` (left) and `LEIC1989-2001` (right)

In fact, `LEIC1989-2001` is more dense (ρ=5.73 for sup=25%) than `PmeLink.PT` (ρ=4.48 for sup=10%), and both more dense than the most dense synthetic dataset (ρ=3.07 for sup=10%). It was not possible to determine the density of `LEIC1989-2001` for lower supports due to memory limitations, but it is obvious that its density would increase.

In this manner, we can conclude that apriori-based algorithms can achieve the performance of pattern-growth methods in general, and outperform them on dense datasets.

2.2 – Comparison between Constrained and Unconstrained Mining

To compare the efficiency of constrained and unconstrained sequential pattern mining, we will proceed as in Chapter 5. We will compare the time spent by constrained and unconstrained mining, using the pushdown automata in Figure 7.3 and Figure 7.18 for mining the datasets `LEIC1989-2001` and `PmeLink.PT`, respectively. Simultaneously, we will evaluate the differences on the

performance when using deterministic finite automata and deterministic and non-deterministic pushdown automata.

PmeLink.PT

Consider the deterministic pushdown automaton in Figure 7.22. It accepts the same number of patterns that the non-deterministic one in Figure 7.3.

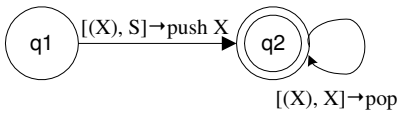


Figure 7.22 – PDA for specifying constraints over the dataset PmeLink

As argued in Chapter 5, the use of non-deterministic pushdown automata essentially presents the same behavior that deterministic ones, with a small decrease on performance. Additionally, it is clear that their use improves the efficiency of sequential pattern mining, both by reducing the number of discovered patterns, and by reducing the processing time (as shown in Figure 7.23 – left).

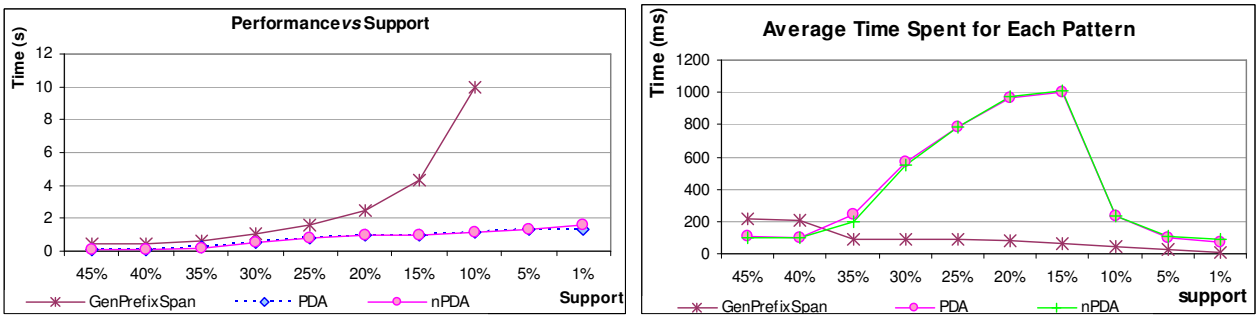


Figure 7.23 – Performance (on the left) and average time spent for each pattern (on the right), using different types of content constraints in dataset PmeLink

Like in synthetic datasets, the average time spent for each discovered pattern decreases when that number increases (Figure 7.23 – right).

LEIC1989-2001

In the LEIC1989-2001 dataset, the results are identical (see Figure 7.24). Since the number of discovered patterns is not the same for the DFA, PDA and nPDA, the charts present some fluctuations, but maintain similar the performances with any of the used automaton.

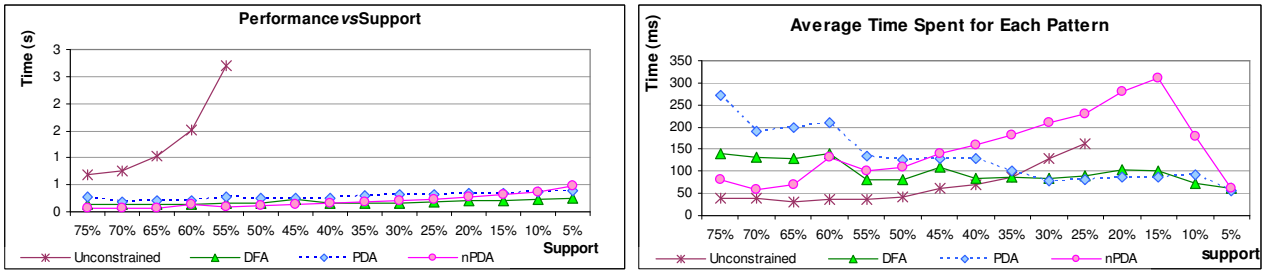


Figure 7.24 – Performance (on the left) and average time spent for each pattern (on the right), using different types of content constraints in dataset LEIC1989-2001

The DFA used in this comparison accepts the curricula on MTP and ASO scientific areas, of students that have failed at most once per subject Figure 7.25.

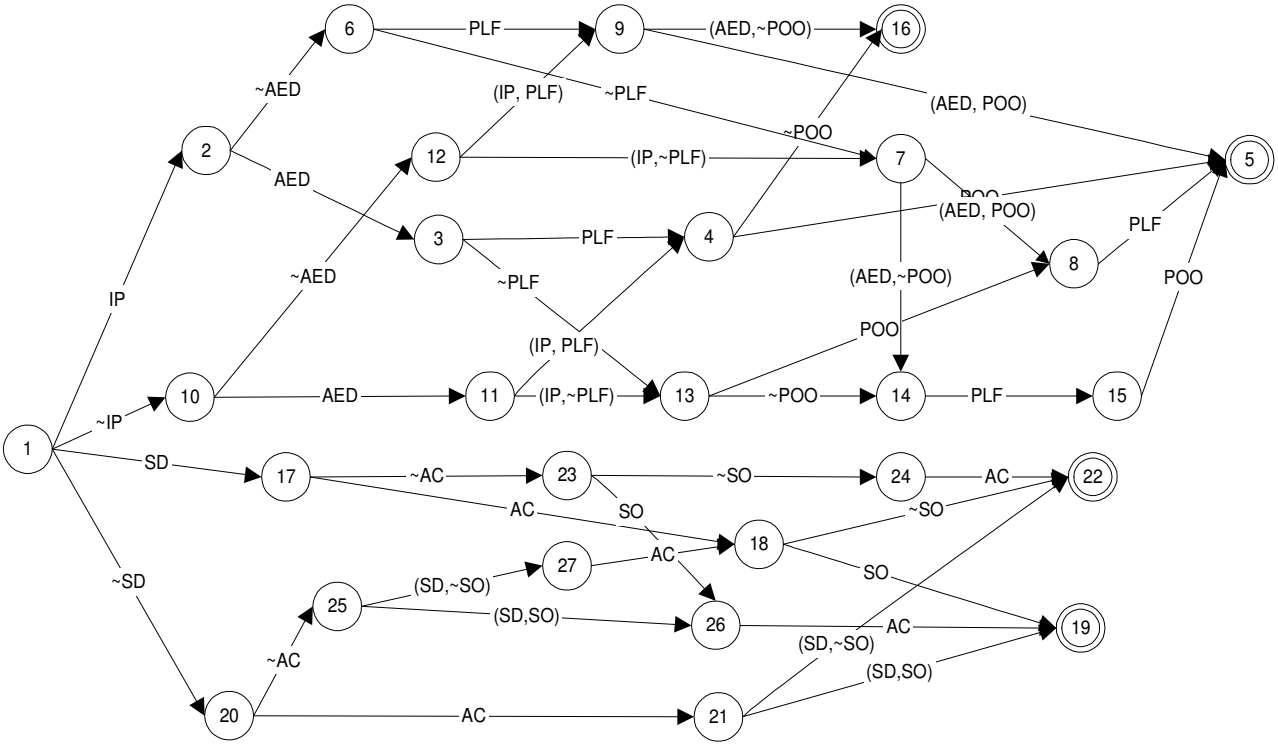


Figure 7.25 – DFA for specifying constraints over dataset LEIC1989-2001

The nPDA used is illustrated in Figure 7.26. It accepts sequences in each scientific area, that mimic the sequence of subjects attended by students, including the possibility of failure in the first two subjects of each scientific area. `(SD)(AC)(SO)`, `(~SD)(AC)(SD,SO)` and `(SD)(~AC)(SO)` are examples of accepted sequences.

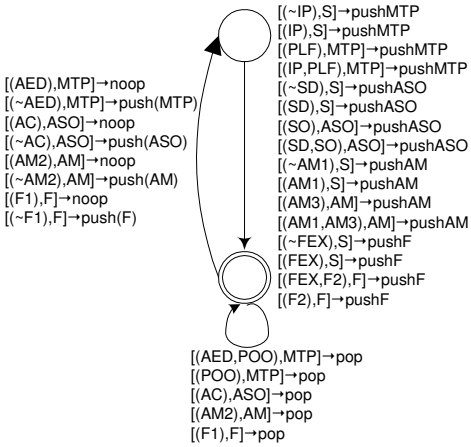


Figure 7.26 – nPDA for specifying constraints over dataset LEIC1989-2001

Figure 7.27 shows that the number of discovered patterns, using the above constraints, is not equal but is similar, which makes the comparison of their performance meaningful. These automata were chosen for this reason.

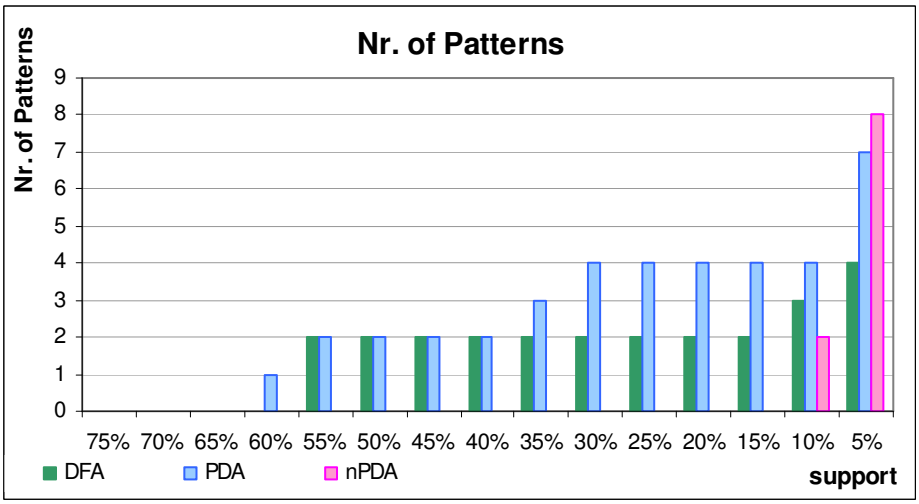


Figure 7.27 – Number of discovered patterns using different content constraints

2.3 – Evaluation of Constraint Relaxations

Finally, the performance achieved with the use of constraint relaxations is also similar to the ones achieved in synthetic datasets.

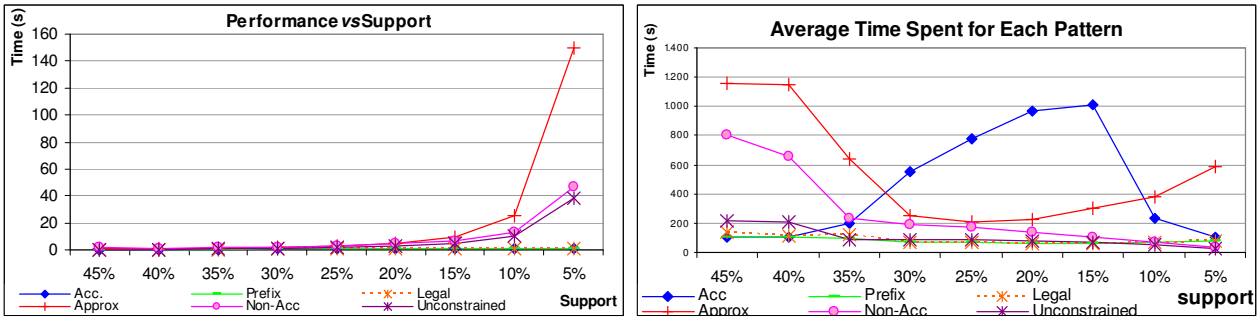


Figure 7.28 – Performance (on the left) and average time spent for each pattern (on the right), using different constraint relaxations in dataset PmeLink.PT

In general, mining with conservative relaxations is as efficient as mining with the entire constraint. However, the average time spent per discovered pattern is lower (Figure 7.28)

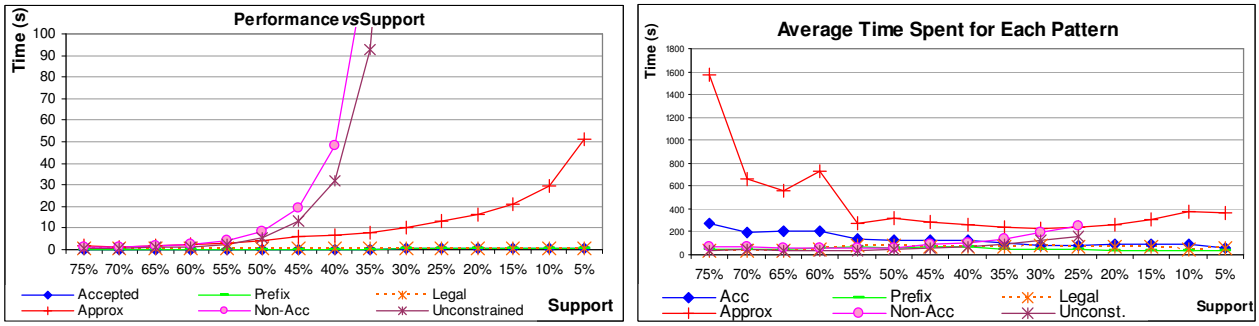


Figure 7.29 – Performance (on the left) and average time spent for each pattern (on the right), using different constraint relaxations in dataset LEIC1989-2001

Naturally, Non-Accepted and Approx relaxations spent much more time than the other relaxations, but this difference is mostly due to the number of patterns that they discover.

As in synthetic datasets, mining with *Non-accepted* and *Approx* relaxations can be less efficient than unconstrained mining. This happens when the number of patterns discovered by these relaxations is similar to the number of discovered unconstrained patterns, as is usual for Non-accepted relaxations with a very restrictive constraint (as the ones used in this chapter).

As Figure 7.29 – right shows *Approx* relaxations are the most expensive per discovered pattern. In fact, even when the number of discovered patterns is considerably lower than the number of unconstrained patterns, it is possible that *Approx* relaxations spent more time than unconstrained mining.

It is important to note that *Approx* relaxation shows a poor performance in `PmeLink.PT` dataset, because it discovers a considerable number of patterns (about 250) and the constraint is specified with a non-deterministic pushdown automata.

Approx Variants

When applied with a restricted alphabet *Approx* relaxations may present better performances, even when non-deterministic pushdown automata are used.

Figure 7.30 shows that *Approx* with a restricted alphabet is able to spend so much time as unconstrained mining, with the advantage that it finds about 20% of the patterns.

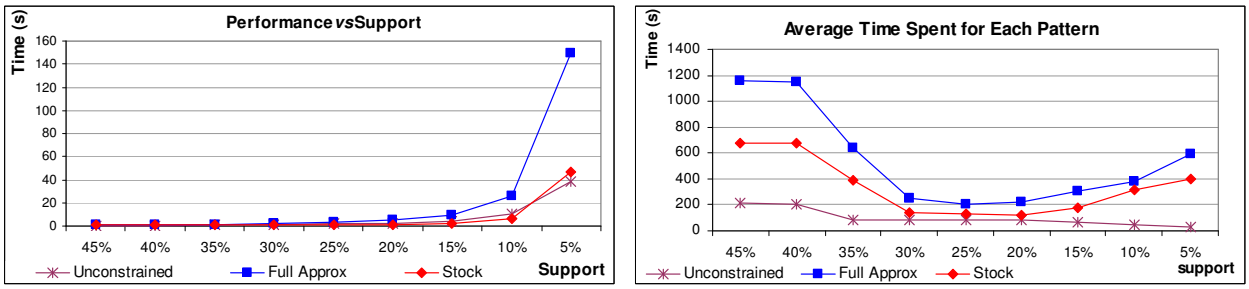


Figure 7.30 – Performance (on the left) and average time spent for each pattern (on the right), using approx variants in dataset PmeLink.PT

Fortunately, with deterministic pushdown automata the results achieved by *Approx* relaxations are much better, outperforming unconstrained mining in the generality of situations. Combining it with an item constraint (restricting its alphabet), the results are even better, as shown in Figure 7.31.

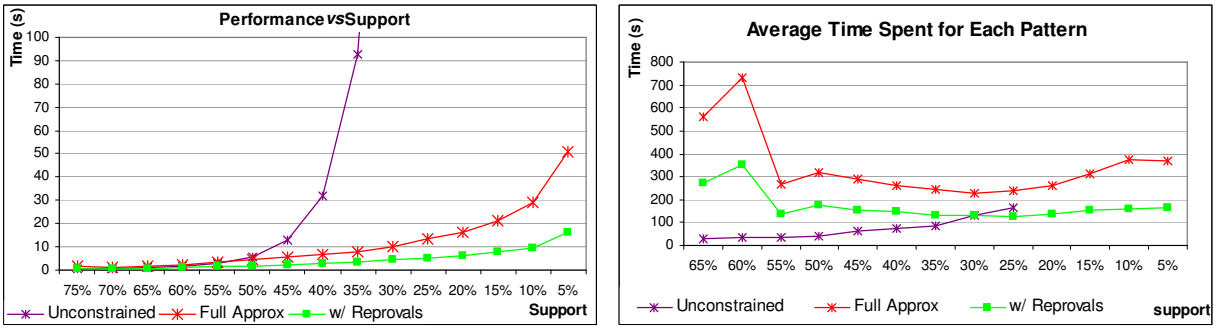


Figure 7.31 – Performance (on the left) and average time spent for each pattern (on the right), using approx variants in dataset LEIC1989-2001

Non-Accepted Variants

The results achieved with non-accepted relaxations are similar. When combined with an item constraint, it can reduce the number of patterns that it discovers. For example, due to the memory requirements of the large number of discovered patterns, it was not possible to discover unconstrained and non-accepted patterns for supports below 25%, on the `LEIC1989-2001` dataset.

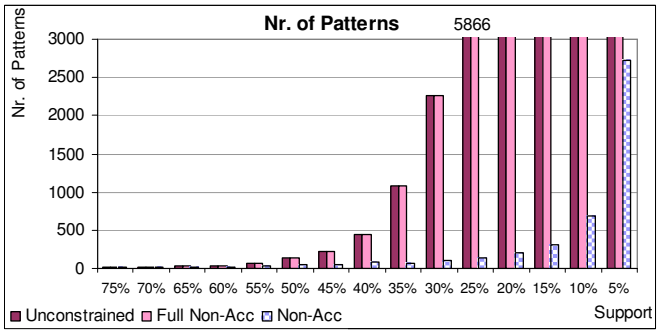


Figure 7.32 – Number of discovered patterns with Non-accepted variants on `LEIC1989-2001`

However, with items restricted to the subjects exclusive to Artificial Intelligence (as in section Finding Artificial Intelligence curricula) it is possible to discover part of the non-accepted patterns, in an acceptable time. Figure 7.32 shows the number of discovered patterns and Figure 7.33 the corresponding performance.

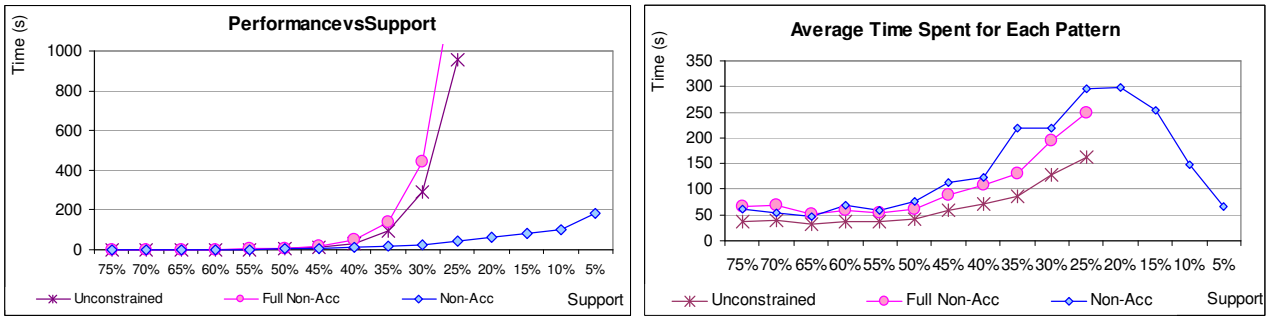


Figure 7.33 – Performance (on the left) and average time spent for each pattern (on the right), using Non-accepted variants in dataset `LEIC1989-2001`

The results are similar, if an item constraint was applied with the Non-Accepted relaxation on the PmeLink.PT dataset, as shown in Figure 7.34.

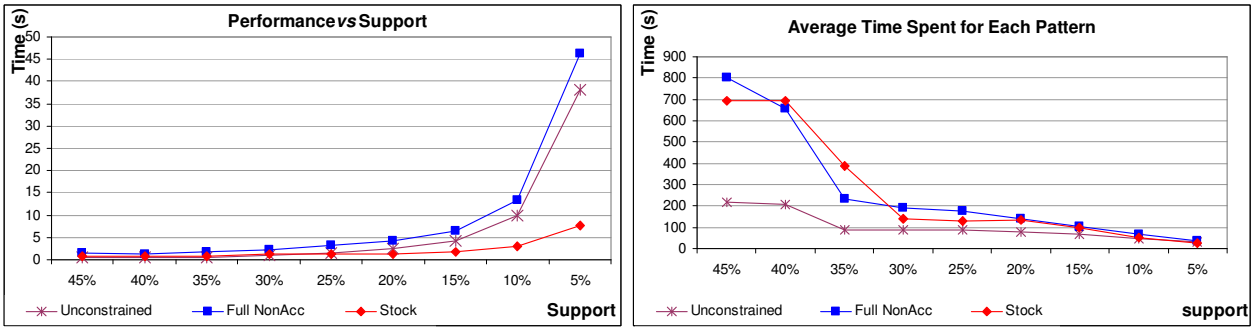


Figure 7.34 – Performance (on the left) and average time spent for each pattern (on the right), using Non-accepted variants in dataset `PmeLink.PT`

3 – Effectiveness Evaluation

In the last section, we have shown that the new methodology improves the efficiency of the mining process, mostly by reducing the number of patterns that are discovered. However, we did not discuss the quality of the discovered patterns when compared with the patterns discovered by an unconstrained mining process.

The main reason for the lack of this discussion until this point is the inexistence of a precise quality assessment of the discovered patterns. The only way to make such an assessement is to compare the discovered patterns with the existing background knowledge, which has to be good enough to distinguish between relevant and irrelevant patterns.

Analysis Goals

In order to perform such effectiveness evaluation, we considered a smaller problem whose results can be easily analised. The selected problem is to find the reasons why students abandon LEIC before concluding the 42 subjects required.

Data Statistics

The dataset used to analyze this question consists on the set of sequences corresponding to the curriculum followed by each LEIC student, with his first enrollment made between 1989 and 1997. From these, we have only considered the students that did not conclude 42 subjects and did not have any enrollment on the last year. In this manner, the dataset includes all the students that abandoned LEIC (at least temporarily). The dataset (`LEICabandons`) contains 489 sequences, with an average sequence length equal to 4 semesters.

Finding Abandon Reasons

This problem was chosen because it is easy to enumerate some resons for abandon in LEIC. By applying common sense, we can suggest two different reasons: the inability to conclude the first computer science specific subjects (‘Programming Introduction’-[IP], ‘Digital Systems’-[SD], ‘Algorithms and Data Structures’-[AED] and ‘Computer Architecture’-[AC]), and the inability to conclude the generic engineering subjects (‘Linear Algebra’-[AL] and ‘Mathematical Analysis 1 and 2’-[AM1, AM2]). This knowledge can be represented by the automaton in Figure 7.35.

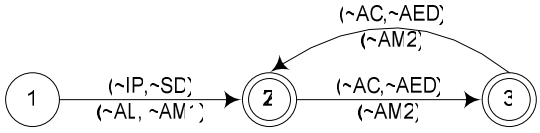


Figure 7.35 – DFA for specifying the anticipated abandon reasons

The great difficulty in determining the effectiveness of our mining process is to determine which ones are the relevant or interesting patterns. In order to choose those patterns, we applied the pattern discovery algorithms on both the `LEICabandons` and `LEIC1989-2001` datasets, and identified as relevant the sequences that are frequent in the first dataset but are not frequent in the second one. With a support of 50%, 91 sequences have been discovered. From these, 79 are relevant by this criterion (as shown in Table 17).

Table 17 – Set of relevant sequences for abandons

Relevant Sequences			
<(~AC,~AED)>	<(~AL)(~AED,~F1,~TC)>	<(~AM1)(~AC,~AED)>	<(~IP)(~AED,~TC)>
<(~AC,~AED,~F1)>	<(~AL)(~AED,~TC)>	<(~AM1)(~AC,~AED,~F1)>	<(~IP)(~TC)>
<(~AC,~AED,~F1,~TC)>	<(~AL)(~F1,~TC)>	<(~AM1)(~AC,~AED,~F1,~TC)>	<(~IP)>
<(~AC,~AED,~TC)>	<(~AL)(~TC)>	<(~AM1)(~AC,~AED,~TC)>	<(~SD)(~AC)>
<(~AC,~F1)>	<(~AL,~AM1)(~AC)>	<(~AM1)(~AC,~F1)>	<(~SD)(~AC,~AED)>
<(~AC,~F1,~TC)>	<(~AL,~AM1)(~AC,~AED)>	<(~AM1)(~AC,~F1,~TC)>	<(~SD)(~AC,~AED,~F1)>
<(~AC,~TC)>	<(~AL,~AM1)(~AC,~AED,~TC)>	<(~AM1)(~AC,~TC)>	<(~SD)(~AC,~AED,~F1,~TC)>
<(~AED,~F1)>	<(~AL,~AM1)(~AC,~F1)>	<(~AM1)(~AED)>	<(~SD)(~AC,~AED,~TC)>
<(~AED,~F1,~TC)>	<(~AL,~AM1)(~AC,~TC)>	<(~AM1)(~AED,~F1)>	<(~SD)(~AC,~F1)>
<(~AED,~TC)>	<(~AL,~AM1)(~AED)>	<(~AM1)(~AED,~F1,~TC)>	<(~SD)(~AC,~F1,~TC)>
<(~AL)(~AC)>	<(~AL,~AM1)(~AED,~F1)>	<(~AM1)(~AED,~TC)>	<(~SD)(~AC,~TC)>
<(~AL)(~AC,~AED)>	<(~AL,~AM1)(~AED,~F1,~TC)>	<(~AM1)(~F1)>	<(~SD)(~AED)>
<(~AL)(~AC,~AED,~F1)>	<(~AL,~AM1)(~AED,~TC)>	<(~AM1)(~F1,~TC)>	<(~SD)(~AED,~F1)>
<(~AL)(~AC,~AED,~F1,~TC)>	<(~AL,~AM1)(~F1)>	<(~AM1)(~TC)>	<(~SD)(~AED,~F1,~TC)>
<(~AL)(~AC,~AED,~TC)>	<(~AL,~AM1)(~F1,~TC)>	<(~F1,~TC)>	<(~SD)(~AED,~TC)>
<(~AL)(~AC,~F1)>	<(~AL,~AM1)(~TC)>	<(~IP)(~AC)>	<(~SD)(~F1)>
<(~AL)(~AC,~F1,~TC)>	<(~AL,~AM1)>	<(~IP)(~AC,~AED)>	<(~SD)(~F1,~TC)>
<(~AL)(~AC,~TC)>	<(~AL,~SD)(~AC)>	<(~IP)(~AC,~AED,~TC)>	<(~SD)(~TC)>
<(~AL)(~AED)>	<(~AL,~SD)>	<(~IP)(~AC,~TC)>	<(~SD)>
<(~AL)(~AED,~F1)>	<(~AM1)(~AC)>	<(~IP)(~AED)>	

A simple analysis of these sequences shows that most students that cancel their registration are not able to conclude more than two subjects in the second semester. Additionaly, this analysis shows that the cancellation reasons anticipated and represented in the automaton in Figure 7.35 are close to the real reasons.

Assuming these sequences are relevant for this task, it is now possible to evaluate the effectiveness of the new methodology by comparing its results with the results reached with constraints, by counting the number of relevant sequences discovered with each relaxation. Table

18 shows the number of sequences discovered and the number of relevant ones per relaxation.

Table 18 – Precision and Recall

	Unconst.	Accepted	Legal	Approx (ε=1)	Approx (ε=2)	Approx (ε=3)	Non-Acc
Nr Total Retrieved	91	2	15	20	71	90	89
Nr Retrieved and Relevant	79	2	10	18	63	78	77
Recall	100%	3%	13%	23%	80%	99%	97%
Precision	87%	100%	67%	90%	89%	87%	87%

Considering the notions of *precision* and *recall* usually used for evaluating the effectiveness of information retrieval algorithms, it is possible to compare the use of constraints with the use of relaxations as proposed in this work.

When applied to the mining process, *precision* corresponds to the the ratio of relevant patterns retrieved by the process to all patterns retrieved by the process, and *recall* corresponds to the ratio of relevant patterns retrieved by the process to all relevant patterns in the dataset.

Applying those measures it is clear that there are significant differences among the relaxations, as shown in Figure 7.36.

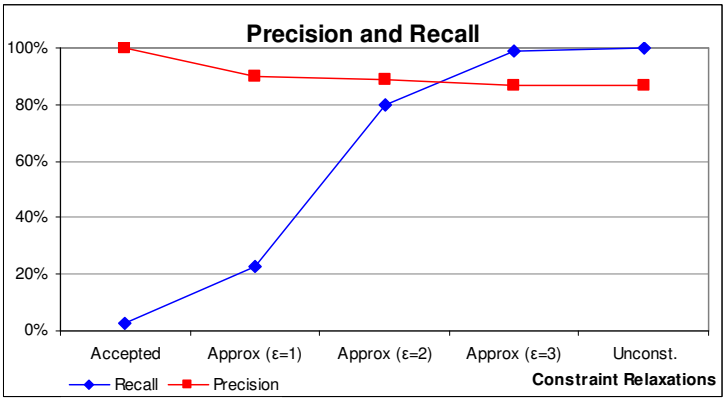


Figure 7.36 – Precision and Recall chart

Since there are only two retrieved sequences when the constraint is used, it is clear that the existing background knowledge is not complete, but is correct. In this manner, the recall of the constrained process is usually very low. The existence of more accurate background knowledge would increase the recall. On the other side, all the accepted sequences are relevant, which makes the precision of the process 100%.

The relaxation that shows a better balance between those measures and the efficiency of the process is the *approx* relaxation, because it is possible to adjust the number of patterns discovered without compromising the precision. By increasing or decreasing the number of errors allowed, it is possible to compensate the excessive selectivness of the background knowledge. Note that the

recall increases considerably when the number of allowed errors increase, but the decrease of the precision is less accentuated.

From this analysis and the other experiments presented in this work, it is clear that the great challenge of pattern mining, in general, and of sequential pattern mining, in particular, is to reach the balance between the number of discovered patterns and the user's ability to analyze those patterns with the ability to discover unknown and useful information. The use of constraint relaxations, proposed in this work, represents a first step in this direction.

Summary

In this chapter, we validate our claims by exploring real-life datasets. The experiments confirm the results achieved in synthetic datasets, both in the analysis of apriori-based algorithms and pattern-growth methods, and in the evaluation of the use of constraint relaxations.

In particular, we confirmed that apriori-based methods can present performances similar and even better than pattern-growth methods. Also confirmed, is the claim that constraint relaxations make the discovery of unknown information possible, maintaining acceptable performances and the focus on user expectations

We also studied how the use of approximate relaxations can be used to move in the precision/recall tradeoff curve, enabling the user to control the number of generated rules.

Chapter 8
Conclusions and Future Work

This chapter resumes the contributions made in this dissertation, presenting a final analysis of the results and some conclusions. At the end of the chapter, we point out some open problems and discuss possible approaches to solve them.

In this dissertation, we have proposed a new methodology to perform pattern mining over nominal event sequences, keeping the focus on the user expectations, without loosing the ability to discover unknown information.

1 – Conclusions

The new methodology is based on the use of the algorithms for sequential pattern mining and two new concepts: Ω–constraints and constraint relaxations.

The Ω–constraint is a new class of constraints that can be used to introduce existing background knowledge into the mining process. It aggregates the different issues related to nominal event sequences, namely their content and temporal aspects. While Ω–constraints provide a framework for representing knowledge about this type of data, constraint relaxations enable the discovery of unknown information.

We have shown that the use of constraint relaxations turns the mining process more efficient than unconstrained mining, making it spend much less time, but discovering patterns that are unknown, yet related to existing background knowledge.

The increase in the performance of the mining process is due to the reduction of the number of discovered patterns, which directly contributes to keep the focus on user expectations. With respect to performance concerns, we have also clarified the challenges in sequential pattern mining and revealed the advantages and disadvantages of each approach to this problem. In particular, we have shown that apriori-based methods can be as efficient as pattern-growth methods, and even better for dense datasets if some optimizations are performed. The experiments on real-life datasets, presented in the last chapter, show that advantage clearly.

Another important result of this dissertation is the definition of the Ω–constraints, which contributes to the definition of a theoretical framework for data mining, providing an integrated way to represent all the aspects related to the knowledge about sequential and nominal temporal data. This class of constraints combines the concepts precisely defined on a Time Ontology and the elegance of formal languages. In particular, we have shown that the use of more expressive languages, like context-free ones, does not impair the performance of the process. In order to deal with sequences of itemsets, we have extended the automata used to generate context-free languages (pushdown automata).

Additionally, we have defined a set of constraint relaxations, based on the relaxation of the formal language associated with the Ω–constraint. In this manner, their use provides a tool to keep the focus on user expectations, since they essentially discover patterns that, in some way, are related to the background knowledge introduced into the process. While conservative relaxations find patterns that are subsequences of patterns accepted by the constraint, non-conservative ones enable the discovery of other patterns, by accepting any sequences with a specific alphabet (*Naïve*), by allowing some errors (*Approx*) or just by considering only the sequences that are not accepted by the constraint (*Non-Accepted*).

The experiments with real-life datasets, in particular the ones performed over LEIC curricula, prove our claims. Those experiments were particularly interesting due to the existence of an accurate knowledge about the models behind the data. In fact, the success of the use of *Non-Accepted* relaxation is conditioned by the existence of this knowledge, since it represents the only way to find some specific but interesting patterns.

Those experiments have also shown that the ability to deal with errors is a real advantage, which makes possible the discovery of unknown patterns that are similar to accepted ones (with respect to some constraint), giving the user the ability to choose the level of similarity, by defining the number of errors accepted.

Unfortunately, the results with the PmeLink.PT data were not so promising, mainly due to the inexistence of such accurate knowledge about the PmeLink.PT business. In this manner, we were unabled to perform a such deep data analysis. Another important factor in these results is the nature of the products commonly transacted – most of the products are bought regularly: for example there is a clear correlation between 'paper A4' and 'goods for printers', which is easily found by association analysis. Moreover, these and most of the other products frequently transacted are just bougth everytime there an order is placed, which consequently implies the discovery of a large number of patterns.

These contributions are described in detail and validated in the main body of the dissertation. Some of them have also been published at international conferences and workhsops ([Antunes 2001b], [Antunes 2002a], [Antunes 2002b], [Antunes 2003], [Antunes 2004a], [Antunes 2004b] and [Antunes 2004c]).

2 – Future Work

This dissertation work opens several venues for future work, both with a narrow focus on temporal pattern mining and a larger focus on the design of constraints and constraint relaxations for traditional and other structured patterns.

Relaxing Temporal Constraints

Constraint relaxations are mainly based on the relaxation of content issues, and they only enable the definition of a time error. However, some recent work on ontologies has been defining fuzzy relations over time intervals [Visser 2003]. These new relations can be used in addition to the ones adopted in this work in order to relax temporal constraints.

Mining Periodicities

In terms of temporal pattern mining, over nominal data, the next step is to employ Ω–constraints to find patterns involving the temporal information, this is, the timestamp of events. Indeed, existing algorithms to find temporal patterns, for example periodicity patterns, are based on the extension of traditional apriori-based algorithms.

One of the unexplored possibilities is the extension of existing sequential pattern mining algorithms, constrained or unconstrained, to discover such periodicities.

Algorithms for Structured Pattern Mining

From the complete understanding of the exploration of sequential pattern mining and corresponding algorithms, it will be possible to face the challenge of developing new algorithms for other structured pattern mining, in a clearer context.

One of the ways to reach that understanding is to study the complexity of the existing algorithms. However, and despite the recent studies on the complexity of pattern mining algorithms (like [Koster 2003], [de Graaf 2002] and [Zaki 1998b]), no work has been done on the complexity of sequential pattern mining algorithms.

Another important task is to develop more efficient algorithms than *GenPrefixGrowth*, to deal with Ω–constraints. In fact, in order to avoid multiple databases scans to test each of the constraints (existential, temporal and content), it requires much larger amounts of memory than *PrefixGrowth*.

Integrated Constraints for Mining Association Rules

Another interesting open issue is the definition of new constraints, similar to the Ω–constraints, able to represent background knowledge for the traditional intra-transactional patterns. Such constraints could be similar to Ω–constraints, just aggregating existential, temporal and content issues. However, while the first two aspects could be the same, content constraints could not be so rich, since there is no structure behind parallel events. Beside these constraints, it could be interesting to incorporate interestingness measures into the new constraint.

Additionally, the definition of the corresponding relaxations would contribute to guide the traditional pattern mining processes, which may contribute to reduce the number of discovered patterns, and, consequently, to focus the process on user expectations.

Applications

Although those experiments were performed with success in the analysis of common curricula, we expect them to be applicable to many other types of sequential data, as for example, in the analysis of process logs. If their supporting models are known in advance, it will be possible to discover failing situations, either by using *legal* or *non-accepted* relaxations.

A similar task can be performed in web-logs, as for example, in the discovery of design errors of web sites. Based on the paths commonly followed by visitors, it is possible to understand the reason why visitors give up going further on the site or buying some product or service.

Conservative relaxations can be of great help to understand those deadlocks.

Another application of sequential pattern mining and our methodology is on fraud detection, as suggested previously. With the model of well-behavied customers and a non-accepted relaxation, it is possible to discover the behaviours of fraudulent clients, since they do not follow that model. The data collected in telecommunications companies is one of the possible applications in this area.

3 – Closing Remarks

In summary, we have presented a new methodology, which is able to solve one of the main drawbacks of sequential pattern mining processes: the lack of focus on user expectations. This methodology makes use of an integrated constraint able to represent existing background knowledge, sequential pattern mining algorithms to discover unknown patterns and constraint relaxations that enable the discovery of unknown information.

The experiments on real-life datasets, presented in this dissertation, demonstrate how this methodology can be applied, and which constraint relaxations are adequate to answer each question.

References

[Adriaans 1996] P. Adriaans and D. Zantinge. *Data Mining*. Addison-Wesley. 1996.

[Agrawal 1993] R. Agrawal, C. Faloutsos, A. Swami, "Efficient similarity search in sequence databases", in *Proc. 4th Int'l Conf. Foundations of Data Organization and Algorit*hms (FODO 93), pp. 69-84. Springer. 1993.

[Agrawal 1994] R. Agrawal, R. Srikant, "Fast Algoritms for Mining Association Rules", in *Proc. 20th Int'l Conf. Very Large Data Bases* (VLDB 94), pp. 487-499. Morgan Kaufmann. 1994.

[Agrawal 1995a] R. Agrawal et al., "Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases", in *Proc. 21st Int'l Conf. Very Large Data Bases* (VLDB 95), pp. 490-501. Morgan Kaufmann. 1995.

[Agrawal 1995b] R. Agrawal et al., "Querying Shapes of Histories", in *Proc. 21st Int'l Conf. Very Large Data Bases* (VLDB 95), pp. 502-514. Morgan Kaufmann. 1995.

[Agrawal 1995c] R. Agrawal and R. Srikant, "Mining Sequential Patterns", in *Proc. 11th Int'l. Conf. Data Engineering* (ICDE 95), pp. 3-14. IEEE Press. 1995.

[Alhir 1998] S. Alhir, *UML in a Nutshell*, O'Reilly & Associates, USA. 1998.

[Allen 1983] J.F. Allen, "Maintaining Knowledge about Temporal Intervals", in *Communications of the ACM*, vol. 26, nr. 11, pp. 832-843. ACM. 1983.

[Allen 1995] J. Allen, *Natural Languages Understanding*, 2nd edition. The Benjamin/Cummings Publishing Company, Redwood City. 1995.

[Antunes 2001a] C. Antunes, *Knowledge Acquisition System to Support Low Vision Consultation*, Master's thesis, Instituto Superior Técnico, Technical University of Lisbon. 2001.

[Antunes 2001b] C. Antunes and A. L. Oliveira. "Temporal Data Mining: an overview", in *Proc.*

1st Workshop on Temporal Data Mining – Int'l Conf. Knowledge Discovery and Data Mining (KDD 01). 2001.

[Antunes 2002a] C. Antunes and A. L. Oliveira. "Using Context-Free Grammars to Constrain Apriori-based Algorithms for Mining Temporal Association Rules", in *Proc.* 2nd *Workshop on Temporal Data Mining* – Int'l Conf. Knowledge Discovery and Data Mining (KDD 02), pp. 11-24. 2002

[Antunes 2002b] C. Antunes and A. L. Oliveira. "Inference of Sequential Association Rules Guided by Context-Free Grammars", in *Proc. 6th Int'l Conf. Grammatical Inference* (ICGI 2002), pp. 1-13. Springer. 2002.

[Antunes 2003] C. Antunes and A. L. Oliveira, "Generalization of Pattern-Growth Methods for Sequential Pattern Mining with Gap Constraints", in *Proc Int'l Conf on Machine Learning and Data Mining*, pp. 239-251. Springer. 2003.

[Antunes 2004a] C. Antunes and A. L. Oliveira, "Sequential Pattern Mining with Approximated Constraints", in *Proc. of IADIS Int'l Conf on Applied Computing*, pp. 131-138. IADIS Press. 2004.

[Antunes 2004b] C. Antunes and A. L. Oliveira. "Mining Patterns Using Relaxations of User Defined Constraints", in Proc. of *3rd Int'l Workshop on Knowledge Discovery in Inductive Databases (KDID 2004*) – *Int'l Conf. Principles and Practice of Knowledge Discovery in Databases* (PKDD 04).

[Antunes 2004c] C. Antunes and A. L. Oliveira "Sequential Pattern Mining Algorithms: Trade-offs between Speed and Memory", in Proc. of *2nd Int'l Workshop on Mining Graphs, Trees and Sequences (MGTS 2004*), – *Int'l Conf. Principles and Practice of Knowledge Discovery in Databases* (PKDD 04).

[Ayres 2002] J. Ayres, J. Gehrke, T. Yu and J. Flannick, "Sequential PAttern Mining using a Bitmap Representation", in *Proc 8th Int'l Conf Knowledge Discovery and Data Mining*,(KDD 2002), pp. 429-435. ACM Press. 2002.

[Bayardo 2002] R.J. Bayardo, The Many Roles of Constraints in Data Mining, in *SIGKDD Explorations*, vol. 4, nr. 1 pp. i-ii. ACM Press. 2002

[Berndt 1996] D. Berndt and J. Clifford, "Finding Patterns in Time Series: a Dynamic Programming Approach", in U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy (eds.), *Advances in Knowledge Discovery and Data Mining*, pp. 229-248.

AAAI Press. 1996.

[Berthold 1999] M. Berthold and D.J. Hand, *Intelligent Data Analysis: an introduction*. Springer. 1999.

[Bettini 1998] C. Bettini et al., "Discovering Frequent Event Patterns with Multiple Granularities in Time Sequences", *IEEE Transactions on Knowledge and Data Engineering*, vol. 10, no. 2 pp. 222-237. IEEE Press. 1998.

[Blankertz 2002] B. Blankertz, G. Curio, K.R. Muller, "Classifying Single Trial EEG: Towards Brain Computer Interface", in T.G. Dietterich, S. Becker, Z. Ghahramani (eds), *Advances in Neural Information Processing Systems*, vol. 14. MIT Press. 2002.

[Brachman 1996] R.J. Brachman and T. Anand, "The Process of Knowledge Discovery in Databases : A human centered approach", in U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy (eds.), *Advances in Knowledge Discovery and Data Mining*, chapter 2, pp. 37-57, AAAI/MIT Press, 1996.

[Caraça-Valente 2000] J. Caraça-Valente and I. Chavarrías, "Discovering Similar Patterns in Time Series", in *Proc. 6th Int. Conf. Knowledge Discovery and Data Mining* (KDD 2000), pp. 497-505. ACM Press. 2000.

[Carrasco 1994] R. Carrasco and J. Oncina, "Learning Stochastic Regular Grammars by means of a State Merging Method", in *Proc. 2nd Int'l Conf. Grammatical Inference* (ICGI 94), pp. 139-152. Springer. 1994.

[Chakravarty 2000] S. Chakravarty and Y. Shahar, "CAPSUL: A constraint-based specification of repeating patterns in time-oriented data", in *Annals of Mathematics and Artificial Intelligence*, vol. 30, pp. 3-22. Kluwer Academic Publishers. 2000.

[Chan 1999] K. Chan and W. Fu, "Efficient Time Series Matching by Wavelets", in *Proc. 15th Int'l Conf. Data Engineering* (ICDE 99), pp. 126-133. IEEE Press. 1999.

[Chen 2000] X. Chen, I. Petrounias, "Discovering Temporal Association Rules: Algorithms, Language and System", in *Proc. 16th Int'l Conf. Data Engineering* (ICDE 2000), pp. 306. IEEE Press. 2000.

[Chomsky 1956] N. Chomsky, "Three Models for the Description of Language", in *IRE Trans. Information Theory* 2:3, pp. 113-124. IEEE Press. 1956.

[Coiera 1994] E. Coiera, "The Role of Knowledge Based Systems in Clinical Practice", in

Knowledge and Decisions in Health Telematics – The Next Decada, pp. 199-203. IOS Press. 1994.

[Cortez 2001] P. Cortez, M. Rocha, J. Neves, "A Meta-Genetic Algorithm for Time Series Forecasting", in *Workshop Artificial Intelligence for Financial Time Series Analysis* (AIFTSA 01), pp. 21-30. 2001.

[Dajani 2001] R. Dajani et al., "Modeling of Ventricular Repolarisation Time Series by Multi-layer Perceptrons", in *Proc. 8th Conference on Artificial Intelligence in Medicine in Europe* (AIME 01), pp. 152-155. Springer. 2001.

[Das 1997] G. Das, D. Gonopulos, H. Mannila, "Finding Similar Time Series", in *Proc. 1st European Conference on Principles and Practice of Knowledge Discovery in Databases* (PKDD 97), pp. 88-100. Springer. 1997.

[Das 1998] G. Das, H. Mannila, P. Smyth, "Rule Discovery from Time Series", in *Proc. 4th Int'l Conf. Knowledge Discovery and Data Mining* (KDD 98), pp. 16-22. ACM Press. 1998.

[Dempster 1997] A.P.Dempster, N.M.Laird, D.B.Rubin: "Maximum Likelihood from Incomplete Data via the EM Algorithm" in *Journal of the Royal Statistical Society Series*, vol. 39, pp. 1-38, Blackwell Publishing. 1997.

[Denis 2002] F. Denis, A. Lemay and A. Terlutte, "Some classes of Regular Languages Identifiable in the Limit from Positive Data", in P. Adrians, H. Fernau and M. van Zaanen (eds.), *Grammatical Inference: algorithms and applications*, pp. 63-76. Springer. 2002.

[Faloutsos 1994] C. Faloutsos, M. Ranganathan, Y. Manolopoulos, "Fast Subsequence Matching in Time-Series Databases", in *Proc. Int'l Conf. on Management of Data*, pp. 419-429. ACM Press. 1994.

[Fama 1970] E. Fama, "Efficient Capital Markets: a review of theory and empirical work", in *Journal of Finance*, pp. 383-417. Blackwell Publishing. 1970.

[Fayyad 2002] U. Fayyad, G.G. Grinstein and A. Wierse (eds), *Information Visualization in Data Mining and Knowledge Discovery*. Morgan Kaufmann. 2002.

[Fisher 1987] D. Fisher, "Knowledge Acquisition via Incremental Conceptual Clustering", *Machine Learning*, vol. 2 pp. 139-172. Kluwer. 1987.

[Frawley 1992] W. Frawley, G. Piatetsky-Shapiro, C. Matheus, "Knowledge discovery in databases: an overview", in *AI Magazine*, vol. 13, no. 3, pp. 57-70. AAAI Press. 1992.

[Garofalakis 1999] M. Garofalakis, R. Rastogi, K. Shim, "SPIRIT: Sequential Pattern Mining with Regular Expression Constraint", in *Proc. Int'l Conf. Very Large Databases* (VLDB 1999), pp. 223-234. Morgan Kaufmann. 1999.

[Garofalakis 2002] M. Garofalakis, R. Rastogi, K. Shim, "Mining Sequential Patterns with Regular Expression Constraints", in *IEEE Transactions on Knowledge and Data Engineering*, pp. 530-552, vol. 14, nr. 3. IEEE Press. 2002.

[Gavrilov 2000] M. Gavrilov et al., "Mining the Stock Market: Which Measure Is Best?", in *Proc. 6th Int. Conf. Knowledge Discovery and Data Mining* (KDD 2000), pp. 487-496. ACM Press. 2000.

[Ge 2000] X. Ge and P. Smyth, "Deformable Markov Model Templates for Time Series Pattern Matching", in *Proc. 6th Int. Conf. Knowledge Discovery and Data Mining* (KDD 2000), pp. 81-90. ACM Press. 2000.

[Giles 1992] C. Giles et al., "Extracting and Learning an Unknown Grammar with Recurrent Neural Networks", in J. Moody, S. Hanson, R. Lippmann (eds), *Advances in Neural Information Processing Systems*, vol. 4, pp. 317-324. Morgan Kaufmann. 1992.

[Giles 2001] C. Giles, S. Lawrence, A.C. Tsoi, "Noisy Time Series Prediction using Recurrent Neural Networks and Grammatical Inference", in *Machine Learning*, vol. 44, pp. 161-184. Kluwer. 2001.

[de Graaf 2002] J. de Graaf, W.A. Kosters, W. Pijls and V. Popova, "A Theoretical and Pratical Comparison of Depth First and FP-growth Implementaions of Apriori", in *Proc. of 14th Belgian-Dutch Conference on Artificial Intelligence* (*BNAIC 2002*), pp. 115-122. 2002.

[Grossman 1998] R. Grossman et al., *Data Mining for Scientific and Engineering Applications*. Kluwer Academic, 1998.

[Gruber 1998] T.R. Gruber, "A Translation Approach to Portable Ontology Specifications", in *Knowledge Acquisition*, 5(2):21-66. Academic Press. 1998.

[Guimarães 2000] G. Guimarães, "The Induction of Temporal Grammatical Rules from Multivariate Time Series", in *Proc. 5th Int'l Colloquium Grammatical Inference* (ICGI 2000) 127-140. Springer. 2000.

[Gunopulos 2001] D. Gunopulos and G. Das, "Time Series Similarity Measures and Time Series Indexing", in *Proc. ACM SIGMOD Conference*, pp. 624. ACM Press. 2001.

[Guralnik 1998] V. Guralnik, D. Wijesakera, J. Srivastava, “Pattern Directed Mining of Sequence Data”, in *Proc. 4th Int'l Conf. Knowledge Discovery and Data Mining* (KDD 98), pp. 51-57. ACM Press. 1998.

[Guralnik 1999] V. Guralnik and J. Srivastava, “Event Detection from Time Series Data”, in *Proc. 5th Int'l Conf. Knowledge Discovery and Data Mining* (KDD 99), pp. 33-42. ACM Press. 1999.

[Han 1998] J. Han, W. Gong and Y. Yin, "Mining Segment-Wise Periodic Patterns in Time-Related Databases", in *Proc. Knowledge Discovery and Data Mining* (KDD 98), pp. 214-218. ACM Press. 1998.

[Han 1999] J. Han, G. Dong and Y. Yin, "Efficient Mining of Partial Periodic Patterns in Time Series Database", in *Proc. Int'l Conf. Data Engineering* (ICDE 99), pp. 106-115. IEEE Press. 1999.

[Han 2000a] J. Han, J. Pei, Y. Yin, “Mining Frequent Patterns without Candidate Generation”, in *Proc. Int'l Conf. on Management of Data*, pp. 1-12. ACM Press. 2000.

[Han 2000b] J. Han et al., “FreeSpan: Frequent Pattern-projected Sequential Pattern Mining”, in *Proc. 6th Int'l Conf. Knowledge Discovery and Data Mining* (KDD 2000), pp. 355-359. ACM Press. 2000.

[Han 2001a] J. Han and M. Kamber, *Data Mining: concepts and techniques*. Morgan Kaufmann Publishers. 2001.

[Han 2001b] J. Han and J. Pei, "Pattern-growth Methods for Sequential Pattern Mining: Principles and Extensions ", in *Proc Workshop on Temporal Data Mining* – Int'l Conf. Knowledge Discovery and Data Mining (KDD 2000). 2001.

[Hayes 1995] P. Hayes, *A Catalog of Temporal Theories*, Technical report UIUC-BI-AI-96-01, University of Illinois. 1995.

[Higuera 1998] C. Higuera, “Learning Stochastic Finite Automata from Experts”, in *Proc. 4th Int’l Conf. Grammatical Inference* (ICGI 98), pp. 79-89. Springer. 1998.

[Hilderman 1999] R. Hilderman and H. Hamilton, " Knowledge discovery and interestingness measures: a survey", *Technical Report CS 99-04*, Department of Computer Science, University of Regina. 1999.

[Hipp 2002] J. Hipp, and U. Güntzer, "Is pushing constraints deeply into the mining algorithms

really what we want?", in *SIGKDD Explorations*, vol. 4, nr. 1, pp. 50-55. ACM Press. 2002.

[Hobbs 2003] J.R. Hobbs and J. Pustejovsky, "Annotating and Reasoning about Time and Events", in *Proc. AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning*, pp. 74-82. AAAI Press. 2003.

[Honavar 1998] V. Honavar and G. Slutzki, *Grammatical Inference*. Berlin: Springer. 1998.

[Hopcroft 1979] J. Hopcroft and J. Ullman *Introduction to Automata Theory, Languages and Computation*. Addison Wesley. 1979.

[Horne 1995] B. Horne and C. Giles, "An Experimental Comparison of Recurrent Neural Networks", in G. Tesauro, D. Touretzky, T. Leen (eds), Advances in Neural Information Processing Systems, vol. 7, pp. 697-704. Cambridge: MIT Press. 1995.

[Huang 1999] Y. Huang and P. Yu, "Adaptive Query Processing for Time Series Data", in *Proc. 5th Int'l Conf. Knowledge Discovery and Data Mining* (KDD 99), pp. 282-286. ACM Press. 1999.

[Inmon 1996] W.H. Inmon, *Building the Data Warehouse*. New York: John Wiley & Sons. 1996.

[Juillé 1998] H. Juillé and J. Pollack, "A Stochastic Search Approach to Grammar Induction", in *Proc. 4th Int'l Conf. Grammatical Inference* (ICGI 98), pp. 79-89. Springer. 1998.

[Keogh 1997] E. Keogh and P. Smyth, "A probabilistic Approach to Fast Pattern Matching in Time Series Databases", in *Proc. 3rd Int'l Conf. Knowledge Discovery and Data Mining* (KDD 97), pp. 24-30. ACM Press. 1997.

[Keogh 1998] E. Keogh and M. Pazzani, "An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback", in *Proc. 4th Int'l Conf. Knowledge Discovery and Data Mining* (KDD 98), pp. 239-241. ACM Press. 1998.

[Keogh 1999] E. Keogh and M. Pazzani, "Scaling up Dynamic Time Warping to Massive Datasets", in *Proc. European Conf. Principles and Practice of Knowledge Discovery in Databases* (PKDD 99), pp. 1-11. Springer. 1999.

[Ketterlin 1997] A. Ketterlin, "Clustering Sequences of Complex Objects", in *Proc. 3rd Int'l Conf. Knowledge Discovery and Data Mining* (KDD 97), pp. 215-218. ACM Press. 1997.

[Kohavi 1998] R. Kohavi and F. Provost, "Glossary of Terms", in Spec. Issue on Apps of Machine Learning and the KDD Process, *Machine Learning Journal*, 30, pp. 271-274. Kluwer. 1998.

[Koster 2003] W.A. Kosters, W. Pijls and V. Popova, "Complexity Analysis of Depth First and FP-

growth Implementations of Apriori", in *Proc Int'l Conf on Machine Learning and Data Mining*, pp. 284-292. Springer. 2003.

[Kum 2003] H.-C. Kum, J. Pei, J., W. Wang and D. Duncan, "ApproxMAP: Approximate Mining of Consensus Sequential Patterns", in *Proc Int'l Conf on Data Mining* (SIAM 2003), pp. .311-315. ACM Press. 2003.

[Lang 1998] K. Lang, B. Pearlmutter, R. Price, "Results of the Abbadingo One DFA Learning Competition and a new Evidence-Driven State Merging Algorithm", in *Proc. 4th Int'l Conf. Grammatical Inference* (ICGI 98), pp. 79-89. Springer. 1998.

[Laxman 2002] S. Laxman, K.P. Unnikrishnan and P.S. Sastry, "Generalized Frequent Episodes in Event Sequences", in *Proc. 2nd Workshop on Temporal Data Mining* – Int'l Conf. Knowledge Discovery and Data Mining (KDD 2002), pp. 46-52. ACM Press. 2002.

[Lee 2001] H. Lee, C.R. Lin, M.S. Chen, "On Mining General Temporal Association Rules in a Publication Database", in *Proc. Int'l Conf. Data Mining* (ICDM 01), pp. 337-344. IEEE Press. 2001.

[Lesh 1999] N. Lesh, M. Zaki, M. Ogihara, "Mining Features for Sequence Classification", in *Proc. 5th Int'l Conf. Knowledge Discovery and Data Mining* (KDD 99), pp. 342-346. ACM Press. 1999.

[Levitt 1996] M.E. Levitt, "Market-Time and Short-term Forescasting of Foreign Exchange Rates", in A.S. Weigend, Y. Abu-Mostafa and A.P. Refenes (eds.), *Decision Technologies for Financial Engineering*, pp. 111-122. London: World Scientific Press. 1996.

[Levenshtein 1965] V. Levenshtein, "Binary Codes capable of correcting spurious insertions and deletions of ones" in *Problems of Information Transmission*, 1:8-17. Kluwer Academic Publishers. 1965.

[Lin 1998] L. Lin and T. Risch, "Querying Continuous Time Sequences", in *Proc. 24th Int'l Conf. Very Large Data Bases* (VLDB 98), pp. 170-181. Morgan Kaufmann. 1998.

[Mannila 1995] H. Mannila, H. Toivonen, I. Verkamo, "Discovering Frequent Episodes in Sequences", in *Proc. 1st Int'l Conf. Knowledge Discovery and Data Mining* (KDD 95), pp. 210-215. ACM Press. 1995.

[Mannila 1996] H. Mannila, H. Toivonen, "Discovering Generalized Episodes using Minimal Occurrences", in *Proc. 2nd Int'l Conf. Knowledge Discovery and Data Mining* (KDD 96),

pp. 146-151. ACM Press. 1996.

[Mannila 1997] H. Mannila and P. Ronkainen, "Similarity of Event Sequences", in *Proc. 4th Int'l Workshop Temporal Representation and Reasoning* (TIME 97), pp. 136-139. 1997.

[Mannila 1999] H. Mannila, D. Pavlov, P. Smyth, "Prediction with Local Patterns using Cross-Entropy", in *Proc. 5th Int. Conf. Knowledge Discovery and Data Mining* (KDD 99), pp. 357-361. ACM Press. 1999.

[Mannila 2000a] H. Mannila and C. Meek, "Global Partial Orders from Sequential Data", in *Proc. 6th Int'l Conf. Knowledge Discovery and Data Mining* (KDD 2000), pp. 161-168. ACM Press. 2000.

[Mannila 2000b] H. Mannila, "Theoretical Frameworks for Data Mining", in *SIGKDD Explorations*, vol. 1, nr. 2, pp. 30-32. ACM Press. 2000.

[Miclet 1996] L. Miclet and C. Higuera, *Grammatical Inference: Learning Syntax from Sentences*, Berlin: Springer. 1996.

[Mitchell 1997] T. Mitchell, *Machine Learning*. McGrawHill. 1997.

[Moody 1992] J. Moody, S. Hanson, R. Lippmann, *Advances in Neural Information Processing Systems*, vol. 4. Morgan Kaufmann. 1992.

[Navarro 2001] G. Navarro, "A guided tour to approximate string matching", in *ACM Computing Surveys*, vol. 33, nr. 1, pp. 31-88. ACM Press. 2001

[Ng 1998] R. Ng, L. Lakshmanan, J. Han, "Exploratory Mining and Pruning Optimizations of Constrained Association Rules", in *Proc. Int'l Conf. on Management of Data* (CIKM 1998), pp. 13-24. ACM Press. 1998.

[Oates 1999] T. Oates, "Identifying Distinctive Subsequences in Multivariate Time Series by Clustering", in *Proc. 5th Int'l Conf. Knowledge Discovery and Data Mining* (KDD 99), pp. 322-326. ACM Press. 1999.

[Oliveira 2000] A. Oliveira, *Grammatical Inference: Algorithms and Applications*. Berlin: Springer. 2000.

[Oliveira 2001] A. Oliveira and J.P. Silva, "Efficient Algorithms for the Inference of Minimum Size DFAs", in *Machine Learning*, vol. 44, pp. 93-119. Kluwer. 2001.

[Özden 1998] B. Özden, S. Ramaswamy, A. Silberschatz, "Cyclic association rules", in *Proc. 14th Int'l Conf. Data Engineering* (ICDE 98), pp. 412-421. IEEE Press. 1998.

[Pei 2001] J. Pei et al., “PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth”, in *Proc. 17th Int’l Conf. Data Engineering* (ICDE 2001), pp. 215-226. IEEE Press. 2001.

[Pei 2002a] J. Pei, and J. Han, "Constrained frequent pattern mining: a pattern-growth view", in *SIGKDD Explorations*, vol. 4, nr. 1, pp. 31-39. ACM Press. 2002.

[Pei 2002b] J. Pei, J. Han and W. Wang, "Mining Sequential Patterns with Constraints in Large Databases", in *Proc. Conf. Information and Knowledge Management* (CIKM), pp. 18-25. ACM Press. 2002.

[Poças 2003] J. Poças, *Um ambiente de pós-processamento para regras de associação*. MSc. Thesis, University of Oporto. 2003.

[Ramaswamy 1998] S. Ramaswamy, S. Mahajan, A. Silberschatz, “On the Discovery of Interesting Patterns in Association Rules”, in *Proc. 24th Int’l Conf. Very Large Data Bases* (VLDB 98), pp. 368-379. Morgan Kaufmann. 1998.

[Refenes 1995] A. Refenes, *Neural Networks in the Capital Markets*. New York: John Wiley and Sons. 1995.

[Roddick 2002] J. Roddick and M. Spiliopoulou, “A Survey of Temporal Knowledge Discovery Paradigms and Methods”, in *IEEE Transactions of Knowledge and Data Engineering*, vol. 14, no. 3, pp. 750-767. IEEE Press. 2002.

[Shahar 1996] Y. Shahar and M.A. Musen, “Knowledge-Based Temporal Abstractions in Clinical Domains”, in *Artificial Intelligence in Medicine*, vol. 8, pp. 267-298. Springer. 1996.

[Sakakibara 1997] Y. Sakakibara, “Recent Advances of Grammatical Inference”, in *Theoretical Computer Science*, vol. 185, pp. 15-45. Elsevier Science. 1997.

[Sakakibara 2000] Y. Sakakibara and H. Muramatsu, “Learning Context Free Grammars from Partially Structured Examples”, in *Proc. 5th Int’l Conf. Grammatical Inference* (ICGI 2000), pp. 229-240. Springer. 2000.

[Searls 1992] D.B. Searls, "The Linguistics of DNA" in *American Scientist*, 80, pp. 579-591. Sigma Xi, The Scientific Research Society. 1992.

[Searls 1994] D.B. Searls, "Genome Informatics" (invited column), in *IEEE Computers in Medicine and Biology* 12, pp. 124. IEEE Press. 1994.

[Searls 1995] D.B. Searls, "String Variable Grammar: A Logic Grammar Formalism for the

Biological Language of DNA", in *Journal of Logic Programming* 24, pp. 73-102. Springer. 1995.

[Sempere 2002] J.M. Sempere and P. García, "Learning Locally Testable Even Linear Languages from Positive Data", in P. Adriaans, H. Fernau and M. van Zaanen (eds.), *Grammatical Inference: algorithms and applications*, pp. 225-236. Berlin: Springer. 2002.

[Shimodaira 2001] H. Shimodaira et al., "Support Vector Machine with Dynamic Time-Alignment Kernel for Speech Recognition", in *Proc. 7th European Conf. Speech Communication and Technology* (Eurospeech 01). 2001.

[Shimodaira 2002] H. Shimodaira et al., "Dynamic Time-Alignment Kernel in Support Vector Machine", in T.G. Dietterich, S. Becker and Z. Ghahramani (eds), *Advances in Neural Information Processing Systems*, vol. 14. Cambridge: MIT Press, 2002.

[Smyth 1997] P. Smyth, "Clustering Sequences with Hidden Markov Models", in G. Tesauro, D. Touretzky, T. Leen (eds), *Advances in Neural Information Processing Systems*, vol. 9, pp. 648-654. MIT Press. 1997.

[Smyth 1999] P. Smyth, "Probabilistic Model-Based Clustering of Multivariate and Sequential Data", in *Proc. 7th Int'l Workshop Artificial Intelligence and Statistics*, pp. 299-304. 1999.

[Srikant 1995] R. Srikant and R Agrawal, "Mining Generalized Association Rules", in *Proc. Int'l Conf. Very Large Databases* (VLDB 1995), pp. 407-419. Morgan Kaufmann. 1995.

[Srikant 1996] R. Srikant and R. Agrawal, "Mining Sequential Patterns: Generalizations and Performance Improvements", in *Proc. 5th Int'l Conf. Extending Database Technology* (EDBT 96), pp. 3-17. Berlin: Springer. 1996.

[Srikant 1997] R. Srikant and R. Agrawal, "Mining association rules with item constraints", in *Proc. Int'l Conf Knowledge Discovery and Data Mining* (KDD 1997), pp. 67-73. ACM Press. 1997.

[Stolcke 1994] A. Stolcke and S. Omohundro, "Inducing Probabilistic Grammars by Bayesian Model Merging", in *Proc. 2nd Int'l Conf. Grammatical Inference* (ICGI 94), pp. 106-118. Springer. 1994.

[Szeto 1996] K.Y. Szeto and K.H. Cheung, "Application of Genetic Algorithms in Stock Market Prediction", in A.S. Weigend, Y. Abu-Mostafa and A.P. Refenes (eds), *Decision Technologies for Financial Engineering*, pp. 95-103. London: World Scientific Press. 1996.

[Visser 2003] U. Visser and S. Hübner, *Temporal Representation and Reasoning for the Semantic Web*, Technical Report TZI-Bericht Br. 24-2003. Bremen: TZI, Center for Computing Technologies. 2003

[Weigend 1994] A. Weigend and N. Gershenfeld, *Time Series Prediction: Forecasting the Future and Understanding the Past*. Addison-Wesley. 1994.

[Witten 2000] I.H. Witten and E. Frank, D*ata Mining Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann. 2000.

[Wu 1992] S. Wu and U. Manber, "Fast Text Searching Allowing Errors", in *Communications of ACM*, vol. 35, pp. 3-91. ACM Press. 1992

[Yi 1998] B. Yi, H. Jagadish, C. Faloutsos, "Efficient Retrieval of Similar Time Sequences Under Time Warping", in *Proc. 14th Int'l Conf. Data Engineering* (ICDE 98), pp. 201-208. IEEE Press. 1998.

[Zaki 1998a] M. Zaki, "Efficient Enumeration of Frequent Sequences", in *Proc. Int'l Conf. Information and Knowledge Management* (CIKM 98), pp. 68-75. ACM Press. 1998.

[Zaki 1998b] M. Zaki and M. Ogihara, "Theoretical Foundations of Association Rules", in *Proc. Of 3rd ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*. 1998

[Zaki 1999] M. Zaki, "Parallel and Distributed Association Mining: a survey", *IEEE Concurrency*, vol. 7, nr. 4, pp. 14-25. IEEE Press. 1999.

[Zaki 2000] M. Zaki, "Sequence Mining in Categorical Domains: Incorporating Constraints", in *Proc. Int'l Conf. Information and Knowledge Management* (CIKM 00), pp. 422-429. ACM Press. 2000.

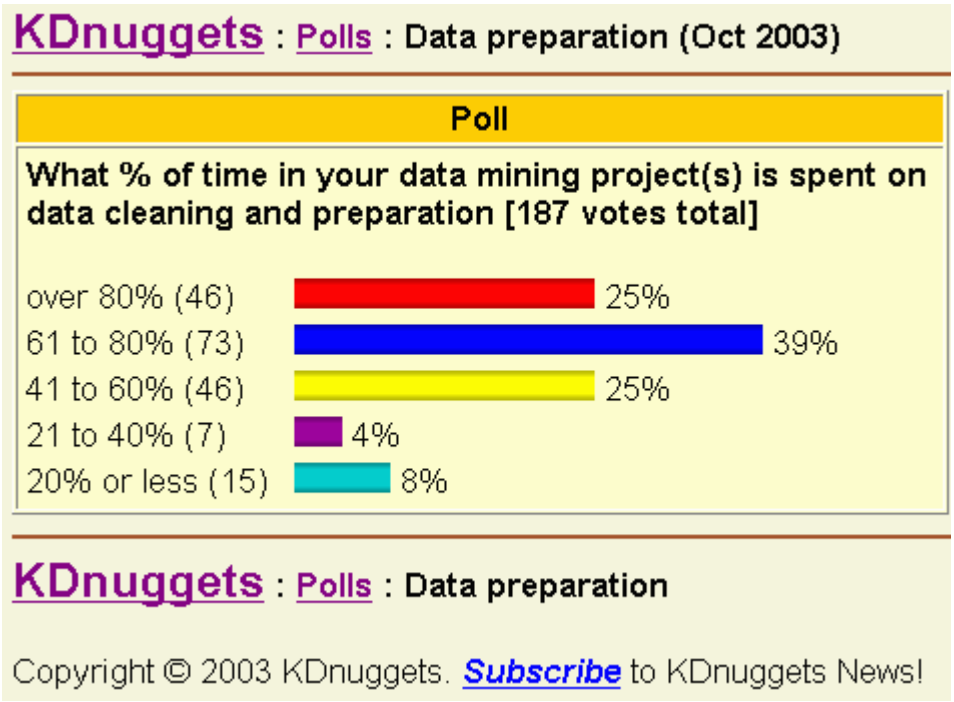
[Zheng 2001] Z. Zheng, R. Kohavi, and L. Mason, "Real World Performance of Association Rule Algorithms", in *Proc Int'l Conf. Knowledge Discovery and Data Mining* (KDD 2001), pp. 401-406. ACM Press. 2001.

[Zhou 2002] Q. Zhou and R. Fikes, "A Reusable Time Ontology", in *Semantic Web Workshop* at AAAI'02. AAAI Press. 2002.

Appendix A

KDNuggets Polls (October 2003)

KDnuggets News is a newletter for the Data Mining and Knowledge Discovery community, that among other things annunciates data mining jobs, software, courses, conferences and news. A usually issue is the "Polls page" where data miners are invited to postpone their opinion. The next figure reproduces the contents of a poll on data preparation.



The results show that the majority of voters (39%) consider that data cleaning and preparation requires between 60 and 80% of the total time.

Appendix B
Experimental Setup

The experiments conducted in this work were based in three real-life datasets and some synthetic ones generated by the QUEST generator.

The QUEST synthetic data generator from the *Intelligent Information Systems* research center at *IBM Almaden*, is a standard on the generation of synthetic sequential data, and has been used in most studies on sequential pattern mining (see [Agrawal 1995c], [Srikant 1996], [Zaki 1998a], [Pei 2001] and [Ayres 2002]).

This data generator creates datasets that try to mimic real-world transactions, where customers tend to make a sequence of transactions involving a set of items. Sequence and transaction sizes are clustered around a mean and some of them may have larger sizes. Note that a sequence may have repeated transactions, but a transaction cannot have repeated items. The data generator accepts a set of parameters to specify the dataset characteristics. Those parameters are shown and explained on Table 19.

Table 19 – Parameters for synthetic data generation

Parameter	Meaning
| D |	Number of sequences or dataset size
| C |	Average number of itemsets per sequence
| T |	Average number of items per itemset
| S |	Average length of maximal potentially frequent sequences
| I |	Average size of itemsets in maximal potentially frequent sequences
N_S	Number of maximal potentially frequent sequences
N_I	Number of maximal potentially frequent itemsets
N	Number of items

More details about the generator can be found in [Agrawal 1995c], and it can be downloaded

from http://www.almaden.ibm.com/software/quest/Resources/index.shtml.

The computer used to run all the experiments was a Pentium M 1GHz with 768MB of RAM. To make the time measurements more reliable, no other application was running on the machine while the experiments were running. The operating system used was Windows XP and the algorithms were implemented using the Java programming language (Java Virtual Machine version 1.4.2_01). The datasets were maintained in main memory during the algorithms processing, avoiding hard disk accesses.