Mining Patterns on Bibliographic Data

Andreia Liliana Perdigão da Silva

Dissertação para obtenção do Grau de Mestre em Engenharia Informática e de Computadores

Júri

Presidente: Joaquim Armando Pires Jorge
Orientadora: Cláudia Martins Antunes
Vogal: Arlindo Manuel Límede de Oliveira
Vogal: José Luís Brinquete Borbinha

November 2009
Acknowledgements

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

To my advisor, Professora Cláudia Antunes, for the support and confidence.

To Professor José Borbinha, for providing the data and the resources I needed.

To my friends, for their presence and support.

To my parents, thanks for the continuous support and encouragement through all these year.

And finally, to Filipe, for his presence, patience, encouragement and support.

Lisbon, November 25, 2009
Andreia Liliana Perdigão da Silva
“le difficile n’est pas de monter, mais en montant, de rester soi.”
- Jules Michelet – Le Peuple
Rezumo

A necessidade para o intercâmbio de informação de forma padronizada, para reduzir os custos da catalogação e aumentar a qualidade dos resultados, levou à criação de múltiplas normas de catalogação. O mundo bibliográfico tem sido guiado por formatos normalizados para descrever os seus registos, Universal MARC (UNIMARC) e MARC21. Estes formatos têm sido eficazes mas denotam algumas deficiências relacionadas com o paradigma actual da Internet. O Functional Requirements for Bibliographic Records (FRBR) é um novo e revolucionário modelo que oferece uma nova perspectiva sobre a estrutura e as relações dos registos bibliográficos e de autoridade, e pretende ser independente de qualquer código de catalogação ou implementação.

O Data Mining tem sido usado poucas vezes para este objectivo, e por isso, este trabalho tem como objectivo usar técnicas de exploração de dados para identificar padrões escondidos nos dados UNIMARC da Biblioteca Nacional de Portugal. Estas relações escondidas entre os vários elementos nesses formatos Machine Readable Cataloguing (MARC) podem vir a ajudar o mapeamento estre estas e os elementos FRBR. E por isso, facilitar a mudança para o paradigma FRBR.

No entanto, estes formatos não são tabulares e apresentam desafios na aplicação de técnicas de Data Mining. Este trabalho propõe assim uma modelação dos dados num esquema em estrela, que resolve alguns dos problemas que advêm dos dados e facilitam a sua exploração. Em conjunto com esse modelo, este trabalho propõe um algoritmo capaz de minar esquemas em estrela baseado no FP-Growth, sem que seja preciso juntar os dados numa única tabela.
Abstract

The need to interchange information in a standardized way, to reduce cataloguing costs and to increase quality of results, led to the creation of multiple cataloguing standards. The bibliographic world has been guided by normalized formats to describe its records, UNIMARC and MARC21. These formats have been effective, but they denote some inadequacy relating to actual Internet paradigm. FRBR is a new and revolutionary model that offers a fresh perspective on the structure and relationships of bibliographic and authority records and intends to be independent of any cataloguing code or implementation.

Data mining had been used fewer times for this purpose, therefore, this work aims to use data mining techniques to identify the hidden patterns in UNIMARC data from National Library of Portugal. These hidden relationships between elements in those MARC formats are likely to help map them into FRBR elements. And therefore, facilitate the change to FRBR paradigm.

However, these formats are not tabular and present some challenges in the application of Data mining techniques. Therefore, this work proposes modelling the bibliographic data in a star schema, which resolves some of the problems that arise from the characteristics of data and facilitate data mining. In conjunction with this model, this paper proposes an algorithm capable of mining star schemas based on FP-Growth, without having to join the data into a single table.
Keywords

Bibliographic Mining
Pattern Mining
Muti Relations
Star Schema
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents</td>
<td>ix</td>
</tr>
<tr>
<td>Acronyms</td>
<td>ix</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2 Contextualization</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Problem Statement</td>
<td>5</td>
</tr>
<tr>
<td>2.1.1 Bibliographic Formats</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Data Usage Statistics</td>
<td>12</td>
</tr>
<tr>
<td>3 Literature Review</td>
<td>17</td>
</tr>
<tr>
<td>3.1 Pattern Mining</td>
<td>19</td>
</tr>
<tr>
<td>3.1.1 Common Notation</td>
<td>21</td>
</tr>
<tr>
<td>3.1.2 Transactional mining</td>
<td>21</td>
</tr>
<tr>
<td>3.1.3 Sequence mining</td>
<td>23</td>
</tr>
<tr>
<td>3.1.4 Tree mining</td>
<td>25</td>
</tr>
<tr>
<td>3.1.5 Graph mining</td>
<td>27</td>
</tr>
<tr>
<td>3.2 Data Mining over Bibliographic Data</td>
<td>29</td>
</tr>
<tr>
<td>3.3 Challenges</td>
<td>32</td>
</tr>
<tr>
<td>3.4 Solution</td>
<td>32</td>
</tr>
<tr>
<td>4 Multi-Relational Pattern Mining</td>
<td>37</td>
</tr>
<tr>
<td>4.1 Introduction</td>
<td>37</td>
</tr>
<tr>
<td>4.2 Related Work</td>
<td>38</td>
</tr>
<tr>
<td>4.3 Mining Stars</td>
<td>39</td>
</tr>
<tr>
<td>4.3.1 The Algorithm</td>
<td>41</td>
</tr>
<tr>
<td>4.4 Experimental Results</td>
<td>50</td>
</tr>
<tr>
<td>4.4.1 Comparing Super FP-Trees built with different orderings</td>
<td>50</td>
</tr>
<tr>
<td>4.4.2 Time spent in each step</td>
<td>51</td>
</tr>
<tr>
<td>4.5 Conclusions</td>
<td>52</td>
</tr>
<tr>
<td>5 Case Study</td>
<td>55</td>
</tr>
<tr>
<td>5.1 Dataset Description</td>
<td>55</td>
</tr>
<tr>
<td>5.2 Experimental Results</td>
<td>55</td>
</tr>
<tr>
<td>5.2.1 Tree’s Evaluation</td>
<td>55</td>
</tr>
<tr>
<td>5.2.2 Performances</td>
<td>57</td>
</tr>
<tr>
<td>5.3 Patterns Evaluation</td>
<td>58</td>
</tr>
<tr>
<td>5.3.1 Quantitative Analysis</td>
<td>58</td>
</tr>
<tr>
<td>5.3.2 Qualitative Analysis</td>
<td>59</td>
</tr>
</tbody>
</table>
List of Figures

2.1 An example of FRBR levels. ................................................. 10
2.2 Group 1 entities and primary relationships. ............................. 12
3.1 Pattern Mining Algorithms ................................................. 20
3.2 Pattern Mining Algorithms characteristics ................................ 20
3.3 UNIMARC star schema ..................................................... 33
4.1 A Movies data warehouse schema [Wie89] ................................. 37
4.2 Construction of the DimFP-Tree of dimension B ......................... 44
4.3 DimFP-Trees of dimensions A and C respectively ....................... 44
4.4 FP-tree constructed based on frequency descending ordering may not always be minimal . 46
4.5 Facts and Super FP-Tree resulting from a support descending order of items . 47
4.6 Facts and Super FP-Tree resulting from a support descending order of dimensions . 48
4.7 Joining two trees ................................................................ 49
4.8 Facts and Super FP-Tree resulting from a path ascending order of dimensions . 49
4.9 Average size of the Super FP-Tree .......................................... 51
4.10 Average Number of Paths of the Super FP-Tree ......................... 51
4.11 Average time on Super FP-Tree construction ............................... 51
4.12 Average Time mining with FPGrowth .................................... 52
4.13 Average time of mining with Star FP-Growth ............................ 52
5.1 Number of entries in each table ............................................ 56
5.2 Size of Attribute’s DimFP-Tree ............................................ 56
5.3 Size of the Super FP-Tree .................................................. 57
5.4 Time spent in the algorithm, for 50% of minimum support .............. 58
5.5 Time spent in Local Mining, for 50% of minimum support ............. 58
5.6 Number of patterns found ................................................ 59
5.7 Number of patterns found for 90% of minimum support ................ 59
List of Tables

2.1 UNIMARC functional blocks. ................................................................. 8
2.2 MARC-21 functional blocks. ................................................................. 9
2.3 Number of existing Fields and Subfields ............................................ 13
2.4 Number and Percentage of records of each type in PORBASE ............. 13
2.5 Fields usage in PORBASE, per Block .................................................. 14
2.6 Fields usage in PORBASE, by Type of Record ..................................... 14
2.7 Detailed fields usage in PORBASE, by Type of Record (Note: rows in which every value was less than 10% were removed) ........................................... 16
3.1 Characters that influence the type of a record ...................................... 34
4.1 Dimension Tables A (Award), B (Studio) and C (Movie) ..................... 40
4.2 Fact table and the frequent itemsets corresponding to each tid .......... 41
4.3 Characteristics of the movies database ............................................... 50
5.1 Intra-Dimension Patterns ................................................................. 60
5.2 Fields Description .............................................................................. 60
5.3 Inter-Dimension Patterns .................................................................. 61
List of Algorithms

1 Apriori Pseudocode .................................................. 21
2 Apriori candidates generation ........................................... 22
3 FP-tree construction .................................................... 23
4 DimFP-tree construction ................................................ 43
5 Super FP-tree construction ............................................. 45
6 Denormalize a fact ....................................................... 46
# List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AACR</td>
<td>Anglo-American Cataloguing Rules</td>
</tr>
<tr>
<td>BFS</td>
<td>Breath-First Search</td>
</tr>
<tr>
<td>BNP</td>
<td>National Library of Portugal</td>
</tr>
<tr>
<td>DBmS</td>
<td>Database Management System</td>
</tr>
<tr>
<td>DFS</td>
<td>Depth-First Search</td>
</tr>
<tr>
<td>ER</td>
<td>Entity-Relationship</td>
</tr>
<tr>
<td>FPM</td>
<td>Frequent Pattern Mining</td>
</tr>
<tr>
<td>FRBR</td>
<td>Functional Requirements for Bibliographic Records</td>
</tr>
<tr>
<td>IFLA</td>
<td>International Federation of Library Associations</td>
</tr>
<tr>
<td>ISBD</td>
<td>International Standard Bibliographic Description</td>
</tr>
<tr>
<td>MARC</td>
<td>Machine Readable Cataloguing</td>
</tr>
<tr>
<td>MRDM</td>
<td>Multi-Relational Data Mining</td>
</tr>
<tr>
<td>OCLC</td>
<td>Online Computer Library Center</td>
</tr>
<tr>
<td>OLAP</td>
<td>On-line Analytical Processing</td>
</tr>
<tr>
<td>OPAC</td>
<td>Online Public Access Catalog</td>
</tr>
<tr>
<td>TEL</td>
<td>The European Library</td>
</tr>
<tr>
<td>UNIMARC</td>
<td>Universal MARC</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The collection of written knowledge in some sort of repository is a practice as old as civilization itself [KK01]. The library of Alexandria, often championed, has been a survivor throughout its long history and serves as a testament to the thirst for knowledge. Physically, books were not what we think of today, but rather scrolls, mostly made of papyrus or leather. They were kept in pigeonholes with titles written on wooden tags.

Later, Europe began to look to the Greek and Roman artistic and literary classics for inspiration. Innovation in the 1400s revolutionized bookmaking. Printed books replaced handwritten manuscripts and were placed on open shelves. Throughout the 1600s and 1700s, libraries surged in popularity. Universities developed and national, state-supported collections began to appear, leading to the grow of libraries. Many of these became national libraries. British Library, the largest library in Britain, was founded in 1759 as part of the British Museum\(^1\). National Library of Portugal in Lisbon was created in 1796\(^2\). The Library of Congress in U.S. was established in 1800 and serves as the research arm of Congress. It is the largest library in the world\(^3\). Libraries may have changed over the years but the need for a repository of knowledge remains.

It is remarkable to observe that the widespread Internet use started in the 1990’s. The emergence of the Internet has led to the adoption of electronic catalog databases (often referred to as “webcats” or as OPACs, for “online public access catalog”). Libraries are now understood as extending beyond the physical walls of a building, by including material accessible by electronic means, and by providing the assistance of librarians in navigating and analysing tremendous amounts of information with a variety of digital tools and allowing users to search the library’s holdings from any location with Internet access.

The bibliographic world has been guided by normalized formats to describe its records, such as UNIMARC and MARC21. These formats have been effective, but they denote some inadequacy relating to actual Internet paradigm, digital content and the need of interoperability between different libraries and other entities. The introduction and development of automated systems for the creation and processing of bibliographic data and the growth of large-scale databases, both national and international, containing records supplied and used by thousands of libraries, contributed to change the environment in which cataloguing principals and standards operate. These changes are increasing the need to reduce cataloguing costs by minimizing duplicate cataloguing effort, to simplify the cataloguing process and to adapt practices. Equally important has been a recognized need to respond more effectively to an increasingly broad range of user expectations and needs.

There are some problems with bibliographic data and the way it is catalogued. Usually, different

\(^1\)The British Library: http://www.bl.uk
\(^2\)BNP: http://www.bnportugal.pt
\(^3\)LoC: http://www.loc.gov
libraries have different catalogues, formats and practices. This leads to data duplication and increases the chance of little misspells and errors (e.g.: two or more records for the same work). Besides that, the more records there are, the harder it is to identify the correct relationships among them. Some information can also be missing or even ill-catalogued, in different or non-searchable fields. If we want to share data around the bibliographic world, it’s imperative to create more specific and uniform rules and, most importantly, really apply those rules.

New models are arising, and Functional Requirements for Bibliographic Records (FRBR) is the most revolutionary case in the bibliographic community [IFL98]. FRBR is a conceptual model of the bibliographic universe, approved by International Federation of Library Associations and Institutions (International Federation of Library Associations (IFLA)) in 1997 and published in 1998. The catalogue is not seen anymore as a sequence of bibliographic records and a replica of the traditional card catalogue, but rather as a network of connected entities, enabling the user to perform seamlessly all the necessary functions.

Some relationships between UNIMARC or MARC-21 and FRBR have already been identified, but there are hidden relationships that human beings cannot identify, and it is also hard to do it using computer programs, simply based on queries. Data mining is a set of techniques that allow the discovery of those hidden relationships between data. It helps on getting appropriate, accurate and useful information which cannot be found with simple queries. These hidden relationships are likely to help on mapping elements in those MARC formats into FRBR elements. And therefore, on facilitating the change to FRBR paradigm.

Despite the UNIMARC define a common structure for the exchange of bibliographic records, in a machine-readable form, this format is not suitable for data mining, since it is not tabular. The data cannot easily be arranged in a table or in a systematic arrangement by columns and rows. And traditional data mining algorithms deal with a table. Also, some data is not categoric neither numerical (the title of a record, for example), and data mining techniques only deal with these kinds of data.

This has several challenges, like how to deal with the high amount of bibliographic records and of possible fields and subfields. Further, several fields are specific for each type of record (e.g. the field 128 – Music performances and scores, will only have values for music records), therefore, the transactional table of this data will be sparse (it will have many empty cells). There are also many fields that are not mandatory, and thus, similar records may pass unnoticed, since they may not have the same fields. Another challenge is the fact that some fields are the composition of several elements. If we consider these elements in the table, each of them will result in another column (on top of already high number of columns). If we do not consider them, we will miss some possible patterns within these composed fields.

All these challenges make almost infeasible the application of most of the common pattern mining algorithms. In particular, existing commercial frameworks cannot deal with non categoric data and the high number of attributes at the same time.

In order to achieve our goal of finding patterns in bibliographic data, we propose a new model to represent this data for analysis and a new algorithm for mining this model.

The model is based on a star schema, a known and widely used data warehouse model, which consists of multiple dimension tables that are associated by foreign keys to a central fact table. This model enables us to split the data into several tables, which helps dealing with the high number of attributes and records. The fact table deals with the sparsity of data, keeping only the existing associations between fields and values. The dimension tables deal with the non categoric data, once each possible value is stored there. The model is also easy to extend. We can add new dimensions and attributes to take into account the relevant aspects for the analysis. Finally, this model also contributes to the automation of the On-line Analytical Processing (OLAP) studies of bibliographic data.

To mine this star model, we propose Star FP-Growth, based on a well known pattern mining algorithm,
the FP-Growth. The main idea is to mine each table separately, keeping only what is frequent in each in a compact tree structure, and then use the fact table to combine those trees and to construct the final result.

To prove this algorithm, we show the results of applying it to a simpler but real movies dataset, and then to the bibliographic records from Porbase, the Portugal Bibliographic Database, with more than 1500000 records in UNIMARC from more than 170 Portuguese libraries.

Chapter 2 describes the problem and the bibliographic data that will be analysed.

Chapter 3 presents the work related to data mining projects in bibliographic data and the state of the art of pattern mining. It also presents the challenges of applying those algorithms in the data in question, as well as the solutions adopted to overcome them.

Chapter 4 describes the approaches followed and the algorithms developed, based on multi-relational pattern mining.

Chapter 5 shows the results of mining the Porbase’s bibliographic data.

Finally, chapter 6 concludes this work and proposes some future work in this area.
Chapter 2

Contextualization

To contextualize this work and better understand the bibliographic data in question, section 2.1 presents the problem statement and bibliographic formats and section 2.2 shows some statistics on data usage.

2.1 Problem Statement

This study is integrated into the TELplus project, more precisely into the working package 3, task 3.3 – FRBR aggregation, search and browsing. This task intends to develop solutions to support alternative services of searching and browsing in The European Library (TEL) according to the FRBR paradigm. It will provide the proof of concept, and will deliver software solutions so that the TEL Office will be able to integrate them in the TEL service in an effective way.

The data from TEL is in the UNIMARC format, stored in eXtensible Markup Language (XML) files. UNIMARC format places several challenges due to the structure it imposes to the records. The hierarchy of each record is not very deep but it is wide, once records can have many fields, and each field can also have many subfields. Further, several fields are specific for each type of record (e.g. the field 128 – Music performances and scores, will only have values for music records), therefore, the transactional table of this data will be sparse (it will have many empty cells). There are also many fields that are not mandatory, and thus, similar records may pass unnoticed, since they may not have the same fields.

The translation between UNIMARC and FRBR paradigms face some limitations. It is difficult to validate the rules already defined to make that translation, and which are the actual librarians practices, i.e. the rules they use to create the records.

This exploitation aims to extract patterns from several UNIMARC records. These patterns can then be used to validate librarian practices and to serve as a basis for the establishment of the translation rules between UNIMARC and FRBR.

Then, for a better understanding of the problem and to gain domain knowledge, the major bibliographic formats will be described, as well as some technical challenges of applying data mining techniques to this data.

2.1.1 Bibliographic Formats

The need to interchange information in a standardized way, to reduce cataloguing costs and to increase quality of results, led to the creation of multiple cataloguing standards[Per07]. These standards are very important because, as well as decreasing costs and facilitating communication between different libraries and institutions, also allow better management, monitoring and recovery of bibliographic data. This has a great impact in both the librarians who see their work simplified, and the users, who see more rapid
and effective responses to their needs and expectations. The sharing of catalogs and even the process of cataloging itself has become simpler, faster and more structured.

The bibliographic information is usually structured by the International Standard Bibliographic Description (ISBD) (International Standard Bibliographic Description) standards derivatives[Byr00]. These standards do not define patterns of information itself, but merely guidelines that can be seen as requirements, according to the type of material to describe. They intend to regularize the form and content of bibliographic descriptions. These standards were initially designed for paper catalogues, first in 1969, and they are now nearly universally applied.

In addition to these standards should be set rules for cataloguing, being the Anglo-American Cataloguing Rules (Anglo-American Cataloguing Rules (AACR)) the most relevant1. The rules cover the description of, and the provision of access points for all library materials commonly collected at the present time. This set of standards and rules define as the basic requirements for the modeling of any bibliographic catalogue.

The generic Machine Readable Cataloguing (MARC) format appeared in the 1960s with the purpose of normalize the data structures, and therefore, to facilitate the transfer of bibliographic data between different systems[IFL99]. MARC is neither a kind of catalogue nor a method of cataloguing. In fact, it is a short and convenient term for assigning labels to each part of a catalogue record so that it can be handled by computers. This model gives rise to various other formats, local or national, such as CATMARC, IBERMARC, DANMARC, LCMARC, UKMARC, etc. Some of these formats still exist, but have been converging to two international formats, UNIMARC and MARC21 (Sometimes one can see the name MARC referring to MARC21).

From 1992-1995 the IFLA Study Group on Functional Requirements for Bibliographic Records (FRBR) developed an entity relationship model as a generalized view of the bibliographic universe, intended to be independent of any cataloguing code or implementation[IFL98]. Their purpose was to formulate recommendations for a basic level national bibliographic record. FRBR offers a fresh perspective on the structure and relationships of bibliographic and authority records, and also a more precise vocabulary to help future cataloguing rule makers and system designers in meeting user needs.

UNIMARC is described below, since our bibliographic data is in this format. MARC-21 is also briefly described, as it may be necessary for the subsequent analysis of data from other libraries as well as FRBR, as a format on the rise.

UNIMARC

Since the early 1970s, an extended family of more than 20 MARC formats has grown up, each with different national cataloguing practices and requirements. Thus, in order to exchange data, bibliographic agencies had to convert it before. One solution to the problem of incompatibility was to create Universal MARC (UNIMARC), an international format which accepts records created in any MARC format. The intention was that each national agency would need to write only two programs - one to convert into UNIMARC and one to convert from it - instead of one program for each other MARC format.

So in 1977 IFLA published UNIMARC: Universal MARC format, stating that “The primary purpose of UNIMARC is to facilitate the international exchange of data in machine-readable form between national bibliographic agencies”. It may also be used as a model for the development of new machine-readable bibliographic formats[IFL94].

The IFLA UNIMARC Core Activity was established in 2003 with the responsibility for the maintenance and development of the UNIMARC format through the Permanent UNIMARC Committee, and

1Anglo-American Cataloguing Rules: http://www.aacr2.org
has been hosted by the National Library of Portugal since then\(^2\). It is a format in constant evolution and expansion.

The scope of UNIMARC is to specify the content designators (tags, indicators and subfield codes) to be assigned to bibliographic records in machine-readable form and to specify the logical and physical format of the records. It does not stipulate the form, content, or record structure of the data within individual systems. It does provide recommendations on the form and content of data when it is to be exchanged. It covers monographs, serials, cartographic materials, music, sound recordings, graphics, projected and video materials, rare books and electronic resources.

UNIMARC is a specific implementation of the standard: Format for bibliographic information interchange on magnetic tape (ISO 2709: 1981). It specifies that every bibliographic record prepared for exchange conforming to the standard must consist of:

<table>
<thead>
<tr>
<th>RECORD LABEL</th>
<th>DIRECTORY</th>
<th>DATA FIELDS</th>
<th>R/T</th>
</tr>
</thead>
<tbody>
<tr>
<td>R/T = Record Terminator</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

a A Record Label consisting of 24 characters that contains data relating to the structure of the record and several implementation-defined data elements like the type of record and the degree of completeness. The data elements in the Record Label are required primarily to process the record and are intended only indirectly for use in identifying the bibliographic item itself;

b A Directory consisting of a 3-digit tag of each data field, along with a 4-digit number indicating its length and a 5-digit number indicating its starting character position relative to the first data field. It ends with a field terminator;

c Data Fields of variable length, each separated by a field separator. Data in fields may be preceded by two indicators (except those with 00- tag) and subdivided into subfields, each with its subfield identifier and delimiter (ISO 646);

d Each record must end with a Record Terminator character to separate it from the next.

The terms defined below are those used in a special sense in UNIMARC. Terms used in their usual bibliographic sense are not defined and definitions of ISBD data elements can be found in the ISBD documents[IFLA07].

**Data Element** The smallest unit of information that is explicitly identified. Data can be coded data or bibliographic data. Coded data is used to represent such items as control numbers, publication type, and main language of text. Bibliographic data is defined by reference to the ISBD for that type of material.

**Field** A defined character string, identified by a tag, which contains one or more subfields. Consecutive fields are separated by a Field Separator.

**Fill Character** A character used in specified character positions to indicate that no data is available to supply the appropriate value in that position, although that character position is applicable.

**Indicator** A character (numeric or alphabetic) associated with a field which supplies additional information about the contents of the field, about the relationship between the field and other fields in the record, or about the action required in certain data manipulation processes.

**Subfield** A defined unit of information within a field (see also Data Element).

\(^2\)IFLA UNIMARC Core Activity: [http://www.ifla.org/VI/8/up.htm](http://www.ifla.org/VI/8/up.htm)
**Subfield Identifier** A code consisting of two characters identifying individual subfields within a variable field. The first character, the delimiter, is always the same unique character specified in ISO 2709 and the second character, the subfield code, is either numeric or alphabetic.

**Tag** A series of three numeric characters used as a label of its associated fields.

The fields are arranged in functional blocks. These blocks organize the data according to its function in a traditional catalogue record. In the table below, fields 0-- – 1-- hold the coded data while fields 2-- – 8-- contain the bibliographic data:

<table>
<thead>
<tr>
<th>Block</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-- Identification block</td>
<td>010 International Standard Book Number</td>
</tr>
<tr>
<td>1-- Coded information block</td>
<td>101 Language of the work</td>
</tr>
<tr>
<td>2-- Descriptive information block</td>
<td>205 Edition statement</td>
</tr>
<tr>
<td>3-- Notes block</td>
<td>336 Type of computer file note</td>
</tr>
<tr>
<td>4-- Linking entry block</td>
<td>452 Edition in a different medium</td>
</tr>
<tr>
<td>5-- Related title block</td>
<td>516 Spine title</td>
</tr>
<tr>
<td>6-- Subject analysis block</td>
<td>676 Dewey Decimal Classification</td>
</tr>
<tr>
<td>7-- Intellectual responsibility block</td>
<td>700 Personal name - primary intellectual responsibility</td>
</tr>
<tr>
<td>8-- International use block</td>
<td>801 Originating source</td>
</tr>
<tr>
<td>9-- Reserved for local use</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1: UNIMARC functional blocks.

Despite the UNIMARC define a common structure for the exchange of bibliographic records, in a machine-readable form, this format is not suitable for data mining, since it is not tabular. The data cannot easily be arranged in a table or in a systematic arrangement by columns and rows. And traditional data mining algorithms deal with a table. Fundamentally, they analyze the set of instances in the table, and construct a model to explain that data. This limitation implies an extra step of pre-processing of data in UNIMARC, to convert it to a table.

Even so, UNIMARC structure is highly hierarchical, and flattening out the data into a set results in loss of structural information. Therefore, the use of traditional data mining techniques is likely to be ineffective. Algorithms that deal with semi-structured data, that will be described later, take into account some structural information, and therefore are more likely to provide more effective results.

**MARC-21**

MARC 21 is the result of the combination of the United States and Canadian MARC formats (USMARC and CAN/MARC, respectively). MARC 21 was designed to redefine the original MARC record format for the 21st century and to make it more accessible to the international community. MARC 21 has formats for the following five types of data: Bibliographic Format, Authority Format, Holdings Format, Community Format, and Classification Data Format. Currently MARC 21 has been implemented successfully by The British Library, the European Institutions and the major library institutions in the United States, and Canada[Tay03].

MARC 21 is an implementation of the American national standard, Information Interchange Format (ANSI Z39.2) and its international counterpart, Format for Information Exchange (ISO 2709)[LN00]. These standards allow users of different software products to communicate with each other and also specify the requirements for a generalized interchange format that will accommodate data describing all forms of materials susceptible to bibliographic description and related information. As communication formats, they do not mandate internal storage or display formats to be used by individual systems.
Like UNIMARC, and because MARC-21 is an implementation of ISO 2709, its bibliographic records have three main sections: a Record Label, a Directory and Data Fields. The fields are also arranged in functional blocks, but in a different way, as shown in the table below:

<table>
<thead>
<tr>
<th>Block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Control information, numbers, and codes</td>
</tr>
<tr>
<td>1</td>
<td>Main entry</td>
</tr>
<tr>
<td>2</td>
<td>Titles and title paragraph (title, edition, imprint)</td>
</tr>
<tr>
<td>3</td>
<td>Physical description</td>
</tr>
<tr>
<td>4</td>
<td>Series statements</td>
</tr>
<tr>
<td>5</td>
<td>Notes</td>
</tr>
<tr>
<td>6</td>
<td>Subject access fields</td>
</tr>
<tr>
<td>7</td>
<td>Added entries other than subject or series; linking fields</td>
</tr>
<tr>
<td>8</td>
<td>Series added entries; location, and alternate graphics</td>
</tr>
<tr>
<td>9</td>
<td>Reserved for local use</td>
</tr>
</tbody>
</table>

Table 2.2: MARC-21 functional blocks.

FRBR

Why create a new and widespread vision of the bibliographic world? Not just to address the need identified at the 1990 Stockholm Seminar on Bibliographic Records for a core level standard that would allow national bibliographic agencies to reduce their cataloguing costs through the creation, as necessary, of less-than-full-level records, but at the same time ensure that all records produced by national bibliographic agencies met essential user needs.

Before FRBR our cataloguing rules tended to be very unclear about using the words “work”, “edition”, or “item”. This was not initially very problematic, because the systems are primarily intended to describe examples of traditional genres, printed or handwritten. The issue gained importance with the emergence of new genres, especially the digital, and with the growing need to support more records with different characteristics[Per07].

Therefore, this model considers the diversity of users (library clients, staff, publishers, distributors, retailers, and the providers and users of information services outside traditional library settings), of materials (textual, music, cartographic, audio-visual, graphic and three-dimensional), of a full range of physical media described in bibliographic records (paper, film, magnetic tape, optical storage media, etc.) and of all formats (books, sheets, discs, cassettes, cartridges, etc.). They also reflect all modes of recording information (analogue, acoustic, electric, digital, optical, etc.).

FRBR is defined in relation to the following generic tasks that are performed by users when searching and making use of national bibliographies and library catalogues:

- **Find**: using the data to find materials that correspond to the user’s stated search criteria (e.g., in the context of a search for all documents on a given subject, or a search for a recording issued under a particular title);

- **Identify**: using the data retrieved to identify an entity (e.g., to confirm that the document described in a record corresponds to the document sought by the user, or to distinguish between two texts or recordings that have the same title);

- **Select**: using the data to select an entity that is appropriate to the user’s needs (e.g., to select a text in a language the user understands, or to choose a version of a computer program that is compatible with the hardware and operating system available to the user);
• Obtain: using the data in order to acquire or obtain access to the entity described (e.g., to place a purchase order for a publication, to submit a request for the loan of a copy of a book in a library’s collection, or to access online an electronic document stored on a remote computer).

The Entity-Relationship (Entity-Relationship (ER)) model, originally proposed by Peter in 1976, is seen as a way to unify the network and relational database views[Che76]. Simply stated the ER model is a conceptual data model that views the real world as entities (e.g. “customer”, “product”) and relationships (“buys”, “pays for”). In this model, an entity is an object that exists and is distinguishable from all other objects. It may be concrete (a person or a book, for example) or abstract (like a holiday or a concept). In turn, attributes describe the entity of which they are associated. They can uniquely identify an instance of an entity, or describe a non-unique characteristic of the instance. A relationship is an association between two or more entities.

To develop FRBR model, it was used the ER model to the analysis of entities, attributes, and relationships and as the framework for assessing the relevance of each attribute and relationship to the tasks performed by users of bibliographic data. Each attribute and relationship is mapped to the four generic user tasks, and relative values are assigned to each attribute and relationship with specific reference to the task performed and the entity that is the object of the user’s interest.

Entities:
In FRBR Model, the entities have been divided into three groups:

• Group 1: The products of intellectual or artistic endeavor that are named or described in bibliographic records:
  – Work: A distinct intellectual or artistic creation;
  – Expression: The intellectual or artistic realization of a work;
  – Manifestation: The physical embodiment of an expression of a work;
  – Item: A single exemplar of a manifestation.

![Diagram of FRBR levels](image)

Figure 2.1: An example of FRBR levels.

Figure 2.1 shows an example of the various FRBR levels.

An expression is any realization of a work: a revision, an update, a translation, or other variant text. Even abridgments or enlargements of an existing text, or the addition of parts or an accompaniment to a musical composition are considered to be different expressions of the same work. However,
when the modification of a work involves a significant degree of independent intellectual or artistic effort, the result is viewed as a new work (paraphrases, rewritings, adaptations for children, parodies, musical variations, etc.). The boundaries of the entity expression are defined so as to exclude aspects of physical form. Defining expression as an entity in the model gives us a means of reflecting the distinctions in intellectual or artistic content that may exist between one realization and another of the same work.

The third entity defined in the model is manifestation: the physical embodiment of an expression of a work. As an entity, manifestation represents all the physical objects that bear the same characteristics, in respect to both intellectual content and physical form. This entity encompasses a wide range of materials, including manuscripts, books, periodicals, maps, posters, sound recordings, films, video recordings, CD-ROMs, multimedia kits, etc. That physical embodiment (paper, audio tape, video tape, canvas, plaster, etc.) constitutes a manifestation of the work. The entity manifestation serves to describe the shared characteristics of copies of a particular publication, edition, release, etc., as well as to describe unique productions such as manuscripts, etc.

An item is a single exemplar of a manifestation. The entity defined as item is a concrete entity. It can be a single physical object (e.g., a copy of a one-volume monograph, a single audio cassette, etc.), or it can comprise more than one physical object (e.g., a monograph issued as two separately bound volumes, a recording issued on three separate compact discs, etc.). Defining item as an entity enables us to separately identify individual copies of a manifestation, and to describe those characteristics that are unique to that particular copy and that pertain to transactions such as circulation, etc. involving that copy.

- **Group 2:** Entities responsible for the intellectual or artistic content, the physical production and dissemination, or the custodianship of such products:
  - Person: An individual;
  - Corporate body: An organization or group of individuals and/or organizations.

- **Group 3:** Entities that serve as the subjects of intellectual or artistic endeavor:
  - Concept: An abstract notion or idea;
  - Object: A material thing;
  - Event: An action or occurrence;
  - Place: A location.

**Relationships:**

In the context of the model, relationships serve as the vehicle for depicting the link between one entity and another, and thus as the means of assisting the user to “navigate” the universe that is represented in a bibliography, catalogue, or bibliographic database.

The first relationships represented in figure 2.2 indicate that a work is “realized through” expression. Viewed from the reverse direction, the relationship indicates that an expression “is a realization of” a work, which is in fact how expression is defined as an entity (“the intellectual or artistic realization of a work”). The logical connection between work and expression serves as the basis both for identifying the work represented by an individual expression and for ensuring that all expressions of a work are linked to the work. Similarly, the relationship connecting expression with manifestation, indicating that an expression is “embodied in” a manifestation, or conversely that a manifestation is the embodiment of an expression, reflects the definition of manifestation (“the physical embodiment of the expression”). In this
case the logical connection serves as the basis both for identifying the expression of a work embodied in an individual manifestation and for ensuring that all manifestations of the same expression are linked back to that expression. The same holds true for the “exemplified by” relationship that connects manifestation with item. Again, this is a unique relationship that is integral to the definition of item ("a single exemplar of a manifestation"). The logical connection serves as the basis both for identifying the manifestation exemplified by an individual item and for ensuring that all copies (i.e., items) of the same manifestation are linked to that manifestation.

2.2 Data Usage Statistics

This section intends to analyse the frequency of use of the diversity of UNIMARC’s fields on all the records in PORBASE.

Analysis carried out by Texas Center for Digital Knowledge [MB03], over a sample of 400000 MARC21 records from Online Computer Library Center (OCLC), state that more than 50% of the fields or subfields are not used in any record, and only 4% of the fields or subfields focus 80% of all MARC21 usage. More recent analysis performed by National Library of Portugal (BNP) [MML07] over more than 1000000 UNIMARC records from PORBASE, state that the same conclusions can be made in UNIMARC format.

UNIMARC offers a wide range of fields and subfields, a lot of them specific for certain types of records. Those works helped understanding that most cataloguers do not take advantage of all the potential of the format, as well as the final users. Although this format can make a richer description, the analysis, design, development and maintenance is very complex.

The analysis of these UNIMARC fields considering their usage in every type of record may be an important complement to these studies. We cannot decide if a certain field is important or not by only look for its global frequency (the number of time it appears, independently of the type of records). Fields that are specific of a particular type of record will have a relatively high frequency for that type but a low global frequency. Unused fields and those that have a low frequency on every type can be further analysed to understand the reasons why their values are so low (for example, some fields are discarded when they are integrated in PORBASE).

The results presented in this section reflect the bibliographic records of PORBASE on June, 2008.

The number of records analysed is 1561098, and the version of UNIMARC is the fourth revision, second edition [IFL02], as in the BNP’s work. Table 2.3 shows the number of fields and subfields existing in this version of UNIMARC. Fields from block 9-- were ignored, once they are local use and definition
To analyse the frequency of the fields and subfields, according to the type of the record, they were grouped into two wider types: languages and others, and then grouped into the types in table 2.4. The criteria used to choose these categories were the two characters that represent the type (type of record and bibliographic level), and the studies above.

One can see that 94% of the records in the PORBASE are related to languages, specially monographs, that correspond to 89% of the languages.

It is also important to notice that there are several records where there is no information about their type of record (Non-catalogued) or that information is not one of the possible types defined by UNIMARC (Unknown) (about 2%, 34092 records).

The usage of the fields in PORBASE, analysed by Block, is shown in table 2.5.

Like stated in [MML07], in block 4--, no field is used in more than 1% of the records. This may be due to the fact that, in PORBASE, analytic records (or component parts) are not considered. Therefore, and because fields in block 4 are responsible for linking the parts of the records, many of these fields will not be used. Block 6-- has also a lower frequency count because the content of its fields is discarded when the records are integrated in PORBASE. The field that is used more times is 675 (in 74% of the records), the only one that is never eliminated.

Blocks 1-- and 8-- have good frequencies, but several fields are automatically filled or corrected, which
may not reflect the behaviour of the cataloguers.

Blocks 2– and 7–, with 60% and 45% of fields being used more than 1%, respectively, may be those that best reflect the actual practice of cataloguing.

Table 2.5: Fields usage in PORBASE, per Block

Table 2.6: Fields usage in PORBASE, by Type of Record

Table 2.6 presents the frequency of fields by type of record.

There are 18 fields that are never used, in any type of record:

- 014, 016, 072, 073
- 131, 140
- 334
- 462
- 608, 615, 626, 660, 670, 680
- 716, 730
- 850, 886

Overall, the most used fields (above 60% of the records) are:

- 001 (100% usage), 005
- 100 (100% usage), 101 (100% usage), 102
• 200 (100% usage), 210 (100% usage), 215
• 675
• 700
• 801

Some of those fields are mandatory (like fields of block 0--) and automatically filled, and the others are the basic structure of a bibliographic record (author, title, publisher, description and classification). Field 801 is the result of a correction task.

Looking at the table above, we find that the number of fields used in less than 1% of the records is 136 (77.3% of all fields). This number is very similar for both types of record. Languages have more than 137 fields with less than 1% usage. However, if we look at each type of language, that value increases to 140 (for serials) and 142 (for monographs). This indicates that distinct types of languages use different fields with different frequencies. This is specially seen in the others type: the mean value of fields being used less than 1% across subtypes is around 137. If the subtypes use the same fields with the same frequency, there would be 137 fields used less than 1% in the records of all other, but there are 128. This means that, while some subtypes use more determined fields and less others, other subtypes will use more the other fields and less the first ones.

If we analyse the detailed fields usage according to their type of record (see table 2.7), we can state that, field 126, for example, has less than 1% usage in every type of record except on sound records (it appears in 32% of these records). And there are a lot of other cases like this. The most significant one is the field 206, which appears in 85% of cartographies and less than 1% on all the other records. This happens because 206 is a field specific for cartographies: material specific area: cartographic materials - mathematical data, as well as 126 a specific field for sound records: sound recordings - physical attributes. Therefore, we can see these specificities just by looking to the table.

Other interesting analysis we can make with this table is to compare the percentages of non-catalogued records with the percentages of the other types, to try to understand to which type they may or not belong. For example, field 005 appears 100% of the times in non-catalogued records, but it only appears 28% in music scores. Since the number of music scores is almost a third of non-catalogued, we might presume that those records are not musics. Furthermore, field 206 has a frequency of 85% for cartographies (it appears in 5130 records), against 0% in non-catalogued records (does not appear in 33670 records). It might indicate that those records are not cartographies.

By finding these relations, the librarians can try to better understand what is the problem that causes the lack of the type of record.

Similar studies can be conducted over the unknown type of records, however the number of records that are not well catalogued is very low comparing to other types, and therefore, the results might not be significant.

To make the results more reliable and significant to the definition of UNIMARC fields, the results of applying these statistics to several different UNIMARC databases should be compared. However, results over PORBASE can be used to detect possible weaknesses, areas of improvement and where to take advantage of the potentialities of the format.
Table 2.7: Detailed fields usage in PORBASE, by Type of Record (Note: rows in which every value was less than 10% were removed)
Chapter 3

Literature Review

Data mining deals with discovering hidden data and unexpected patterns and rules in large databases [AZ97]. It can bring significant gains to organizations, especially if they have extensive databases. We can say that data mining is a set of techniques that help getting appropriate, accurate and useful information, which we cannot find with standard query tools.

While query languages only help to find data under constraints that are already known, data mining algorithms can find interesting regularities in a database, even when we do not know exactly what we’re looking for. Still, query tools and data mining tools are complementary, and they do not replace each other.

It is very difficult to introduce data mining into a whole organization. Data mining projects face some problems like: missing or incorrect data; timing, distance and interpretation problems; duplication of data; legal or privacy restrictions; etc. That’s why 75% of the whole process of extraction of knowledge from data is about cleaning and preparing data and only the remaining 25% is about mining.

The data mining methodology is an ongoing process, by which information and understanding of the data improves and deepens all the time. Organizations should continually work on their data, constantly identifying new information needs and trying to improve the data to make it better match the goals.

Fundamentally, traditional data mining is the analysis of a table with data, i.e. a set of instances, and the construction of a model explaining these data. Since this discovery is made based on data already known, this kind of learning is called inductive or empirical learning. The model discovered is then evaluated, being confronted with the expectations of the user, measuring essentially model’s capability of explaining, whether data already known, as yet unknown.

Essentially, there are three data mining operations: Clustering, Association and Classification. Choosing the right pattern recognition technique for each data is not an easy task. The first two operations are unsupervised, since they just receive training data and intend to find patterns in it, without any other external source of information, i.e. there is no already known result to guide these techniques. On the other side, classification is a supervised operation, because it compares calculated values with the known results, and its goal is to find a model capable of classifying the instances of the problem in question.

These operations consist of:

**Clustering** Clustering analysis identifies clusters embedded in the data. A cluster is a collection of data objects that are similar in some sense to one another. A good clustering method produces high-quality clusters to ensure that the inter-cluster similarity is low and the intra-cluster similarity is high. In other words, members of a cluster are more like each other than they are like members of a different cluster [Ora05]. Clustering is based on a divide and conquer methodology, which states that, by decomposing a large system in smaller components, its modeling and implementation
becomes easier. The greatest difficulty of these methods lies in finding the number of clusters and in determining the function of distance that measures the similarity between two elements.

Most common applications of clustering include customer segmentation and targeted marketing.

**Association** Given a set of records, where each transaction is a set of objects (called items), an association rule is an expression of the form $X \Rightarrow Y$, where $X$ and $Y$ are sets of items [Sri96]. The intuitive meaning of such a rule is that database records which contain $X$ tend to contain $Y$. An example of an association rule is: “30% of records that contain the title proper (field 200 $a$) also contain a note pertaining to it (field 304 $a$); 2% of all transactions contain both of these items”. Here 30% is called the confidence of the rule, and 2% the support of the rule. The problem is to find all association rules that satisfy user-specified minimum support ($min\_sup$) and minimum confidence ($min\_conf$) constraints.

Association rules are generated according to the frequent itemsets, also called patterns. Therefore, association problems may be seen as two independent problems: pattern mining and rule construction.

In this manner, association rule mining can be divided into two steps. First, frequent patterns with respect to support threshold $min\_sup$ are mined. Second, association rules are generated with respect to confidence threshold $min\_conf$. As shown in many studies (e.g., [AS94]), the first step, mining frequent patterns, is significantly more costly in terms of time than the rule generation step [Pei02].

Besides simple transactional patterns, we can also mine sequential, tree and graph patterns, which are inter-transaction associations, unlike intra-transaction association rules.

**Classification** The input data for classification, also called the training set, consists of multiple examples (records), each having multiple attributes or features. Additionally, each example is tagged with a special class label. Classification consists of dividing the items of a collection into categories or classes, and it can be used both to understand the existing data and to predict how new instances will behave. While clustering is a way to segment data into groups that are not previously defined, classification is a way to segment data by assigning it to groups that are already defined. Classification models are created by examining already classified data, gathered from the training set, and inductively finding a predictive pattern/rule. This model is then used to accurately predict the target class label for each record in new data, which is not in the training data and for which the class labels are unknown.

For instance, consider a credit card company with data about its cardholders. Assume that the cardholders have been divided into two classes, good and bad customers, based on their credit history. The company wants to develop a profile for each customer class that can be used to accept/reject future credit card applicants. This problem can be solved using classification. First, a classifier is given the customer data along with the assigned classes as input. The output of the classifier is a description of each class (good/bad) which can then be used to process future card applicants [Sri96].

Similar applications of classification include target marketing, medical diagnosis, treatment effectiveness and store location.

Some data mining problems may require several techniques. For example, a classification algorithm may be run on the results of clustering to understand the results. Visualization techniques provide tools to allow human guidance of the rule discovery process and they are often useful both before the mining (to look at the data) and after the mining (to understand the output).
Since the aim of this study is to verify whether the rules defined for the inclusion of bibliographic data are actually being used, only pattern mining will be used, to find the patterns embedded in data and to compare them with those that are defined.

First, it is made a survey of pattern mining algorithms, focused on its application to XML databases. Then the related work is presented.

### 3.1 Pattern Mining

Frequent pattern mining (Frequent Pattern Mining (FPM)) is an important data mining paradigm that helps to discover patterns that conceptually represent relations among discrete entities (or items). Depending on the complexity of these relations, different types of patterns arise [CHSZ08].

The most common types of patterns are sets, where the relation is the co-occurrence of items. A well-known example of a set pattern is a market-basket, the set of items that are bought together by a customer, at a supermarket. Next, there are sequence patterns, where we require an ordering (temporal or positional) between items. Examples include time-series data in financial markets, genome sequence data in bioinformatics, etc. Data mining researchers also work with tree and graph patterns. In tree patterns the item relationship takes a hierarchical form, and in graph patterns the relationship is mostly arbitrary. Mining web log data, XML, or semi-structured data are examples of tree mining, and mining chemical compounds for drug discovery, or web communities in a web graph, are examples of graph mining. It is thus clear that different applications require the ability to define and mine different types of patterns.

All of these scenarios require efficient and flexible FPM algorithms and support data/index structures, which can be reused in a variety of domains.

It is worth noting that there exist open source data mining suites such as Weka [WF99] and Parmol, which include commonly used data mining methods for association rules, clustering, and classification. For the specific case of itemset mining, there also exist repositories of separate methods, such as Frequent Itemset Mining Implementations. However, no unified framework for various FPM tasks currently exists.

There are many stand-alone algorithms to mine different types of patterns. On closer examination, certain common themes and common algorithmic paradigms permeate all of the existing methods. The number of candidates generated or the way they are generated (adding a node or adding an edge) differ for one pattern type versus another. There are three approaches for this generation: A breadth-first approach (Breadth-First Search (BFS)), where the search is performed level-wise. First all itemsets of size 1 (1-itemsets) are generated and counted (e.g. A, B and C), then from the frequent 1-itemsets candidate 2-itemsets are constructed (AB, AC, BC), and so on. A depth-first approach (Depth-First Search (DFS)), in which first, itemset A is generated and counted, then, if it is frequent, itemset AB is generated and counted, and so on. The third approach is to use a combination of depth-first and breadth-first traversal. First, all itemsets of size 1 are generated and counted. Then, A is joined with other frequent 1-itemsets and the obtained set of 2-itemsets (AB and AC) are recursively investigated further. Only after all itemsets containing item A have been investigated, itemsets which do not contain A are considered.

The most important pattern mining algorithms are shown in figure 3.1.

They can be divided in terms of the data they deal. Transactional data consists of any data stored in an unstructured format at an atomic level. That is, in the unstructured content, there is no conceptual definition and no data type definition. On the contrary, structured data is anything that has an enforced composition to the atomic data types. This data is managed by technology that allows for querying and reporting against predetermined data types and understood relationships.

19
Figure 3.1: Pattern Mining Algorithms

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Authors</th>
<th>Year</th>
<th>Extends</th>
<th>Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apriori</td>
<td>Agrawal &amp; Srikant</td>
<td>1994</td>
<td>-</td>
<td>With candidates generation</td>
</tr>
<tr>
<td>FP-growth</td>
<td>Han &amp; Pei</td>
<td>2000</td>
<td>-</td>
<td>No candidates generation</td>
</tr>
<tr>
<td>GSP</td>
<td>Agrawal &amp; Srikant</td>
<td>1996</td>
<td>Apriori</td>
<td>Apriori-based</td>
</tr>
<tr>
<td>PrefixSpan</td>
<td>Han &amp; Pei</td>
<td>2001</td>
<td>FreeSpan</td>
<td>Pattern-growth based</td>
</tr>
<tr>
<td>SPADE</td>
<td>Zaki</td>
<td>2001</td>
<td>-</td>
<td>Lattice-based</td>
</tr>
<tr>
<td>Spam</td>
<td>Ayres &amp; Gehrke</td>
<td>2002</td>
<td>-</td>
<td>Pattern-growth based</td>
</tr>
<tr>
<td>Tree Miner</td>
<td>Zaki</td>
<td>2002</td>
<td>Pattern Matcher</td>
<td>Lattice-based</td>
</tr>
<tr>
<td>Xspanner</td>
<td>Wang &amp; Pei et al.</td>
<td>2004</td>
<td>PrefixSpan</td>
<td>Pattern-growth based</td>
</tr>
<tr>
<td>X3Miner</td>
<td>Tan et al.</td>
<td>2005</td>
<td>TreeMiner</td>
<td>Apriori-based</td>
</tr>
<tr>
<td>AGM</td>
<td>Inokuchi et al.</td>
<td>2000</td>
<td>-</td>
<td>Apriori-based</td>
</tr>
<tr>
<td>FSG</td>
<td>Kuramochi et al.</td>
<td>2001</td>
<td>AGM</td>
<td>Apriori-based</td>
</tr>
<tr>
<td>gspan</td>
<td>Yan &amp; Han</td>
<td>2002</td>
<td>PrefixSpan</td>
<td>Pattern-growth based</td>
</tr>
<tr>
<td>MoFa</td>
<td>Borgelt &amp; Berthold</td>
<td>2002</td>
<td>-</td>
<td>Pattern-growth based</td>
</tr>
<tr>
<td>FFSM</td>
<td>Huan et al.</td>
<td>2003</td>
<td>gspan</td>
<td>Pattern-growth based</td>
</tr>
<tr>
<td>Gaston</td>
<td>Niessen et al.</td>
<td>2004</td>
<td>-</td>
<td>Lattice-based</td>
</tr>
<tr>
<td>ADI-Mine</td>
<td>Wang &amp; Pei et al.</td>
<td>2004</td>
<td>gspan</td>
<td>Pattern-growth based</td>
</tr>
</tbody>
</table>

Figure 3.2: Pattern Mining Algorithms characteristics
Some characteristics of these algorithms are listed in table 3.2. Next sections will give a brief description of them. To learn more about each algorithm or about other related algorithms, see their references.

3.1.1 Common Notation

Let \( I = \{i_1, i_2, \ldots, i_m\} \) be a set of literals, called items. Let \( D \) be a set of transactions, where each transaction \( T \) is a set of items such that \( T \subseteq I \). Associated with each transaction is a unique identifier, called its \( TID \). We say that a transaction \( T \) contains \( X \), with \( X \) a set of some items in \( I \), if \( X \subseteq T \).

3.1.2 Transactional mining

A transactional mining (also called itemset mining) problem is to discover frequently co-occurring sets of items (or attributes) in simple transactions. The sequence of transactions and the structure/taxonomy/hierarchy of data is not taken into account. Therefore, applying these algorithms to semi-structured data, like XML, may not give the more effective results. In other words, they probably will miss possible interesting patterns and discover many useless ones.

Most known (and used) methods are Apriori and FP-growth. They are the basis of many other algorithms:

**Apriori** [AS94, Sri96] Apriori is the most known method for pattern discovery. It uses an approach based on BFS (also called, a candidate generation approach).

Algorithm 1 gives the Apriori algorithm.

```plaintext
L_1 \leftarrow \{\text{frequent 1-itemset}\}

\text{for} \ k := 2; L_{k-1} \neq \emptyset; k := k + 1 \text{ do}

\quad C_k \leftarrow \text{New candidates of size } k \text{ generated from } L_{k-1};

\quad \text{for all transaction } T \in D \text{ do}

\quad \quad \text{Increment the count of all candidates in } C_k \text{ that are contained in } T;\n
\quad L_k \leftarrow \text{All candidates in } C_k \text{ with minimum support}\n
\text{return } \bigcup_k L_k
```

Algorithm 1: Apriori Pseudocode.

In the first step, the algorithm simple counts the support of individual items and determines which of them are frequent, i.e. have minimum support. In each subsequent step, it starts with a seed set of itemsets found to be frequent in the previous step. It uses this seed set for generating new potentially frequent itemsets, called candidate itemsets, and counts the actual support for these candidate itemsets during the pass over the data. At the end of the pass, it determines which of the candidate itemsets are actually frequent, and they become the seed for the next step. This process continues until no new frequent itemsets are found.

Although the number of items is usually very high, the most expensive operation is counting the support of each candidate. Indeed, each iteration \( k \) involves a passage through all transactions, and for every transaction, verify if each candidate of size \( k \) is contained therein. Therefore, to optimize the process, Apriori explores a lexicographical order of the items and the anti-monotonic property, in which an itemset is not frequent if any of its subsets is not frequent. Algorithm 2 shows how the candidates are generated, and how these properties are used.

In the **join** step, candidates generation is made by crossing two frequent itemsets (\( L_{k-1} \) with itself). Taking advantage of the lexicographical order of the items, a new candidate of size \( k \) only derives
Algorithm 2: Apriori candidates generation.

from the crossing of two frequent itemsets of size $k-1$ if they share the maximum prefix, i.e. the first $k-2$ items (in other words, if the two itemsets have the same sequence when the last element is removed), and if the last item from the first set is smaller (lexicographically) than the last item from the second set. The generated candidate has the maximum prefix shared, followed by the last item from the first and second sets, respectively. For example, if $abc$ and $abd$ are two frequent itemsets (i.e. belong to $L_3$), $ab$ is common to both and $c$ is smaller than $d$, according to alphabetical order, and then, $abcd$ will be a candidate with size 4 (this prevents from generating $abdc$).

In the prune step, by the anti-monotonic property, all candidates in $C_k$ that have any subset not frequent, i.e. not belonging to $L_{k-1}$, are deleted. In the previous example, $abcd$ would be deleted if any of its subsets $abc$, $abd$, $acd$ or $bcd$ were not frequent.

**FP-growth** [HPY00, Pei02] Pattern-growth methods adopt a divide and conquer approach to decompose both the mining tasks and the databases. Although apriori-based methods have an acceptable performance, candidate set generation is still costly, especially when there exist prolific and/or long patterns. This deterioration is due to the large number of candidates generated and the need to scan the database several times. With these methods, many interesting patterns can be mined efficiently, even patterns with some tough non-anti-monotonic constraints.

The main idea of Frequent Pattern-growth (FP-growth) is to avoid completely the costly ‘candidate-generation-and-test’ processing and avoid expensive, repeated database scans. To achieve the first goal, the algorithm represents the data into a compact tree structure, called FP-tree, to facilitate counting the support of each itemset. An FP-tree is an extended prefix-tree structure storing crucial, quantitative information about frequent patterns. Only frequent length-1 items will have nodes in the tree, and the tree nodes are arranged in such a way that more frequently occurring nodes will have better chances of sharing nodes than less frequently occurring ones. It consists of a root node labeled as "null", a set of item-prefix subtrees where each node has, in addition to a label and a reference to the next node, the support represented by the portion of the path reaching this node, and a frequent-item-header table (FP-tree header).

The method is trivial (Algorithm 3): it starts with a single node tree, and for each transaction $T$ (itemset), if it has a child $N$ such that $N$ name = $x$ name, then increment $N$’s count by 1; else create a new node $N$, with count initialized to 1, and insert it into the root of the tree. This procedure is repeated until there are no more items in $T$. Arranging the frequent itemsets by a support-descending order provides a relatively compact FP-tree structure and facilitates the counting of support.

The FP-tree header table contains two fields, one for frequent items and other for a pointer pointing to the first node in the FP-tree carrying the item. Additionally, every tree node referring to the
FP-TreeConstruction(Dataset \( D \), \( \text{min\_sup} \))
//First database scan
\( F \leftarrow \text{List of frequent items, in support-descending order} \)
\( R \leftarrow \text{the root of a new FP-tree (label = null and support = 0)} \)
//Second database scan
for all transaction \( T \in D \) do
    \( T \leftarrow \text{select frequent items in } T \text{ and order according to } F \)
    insert_tree(\( T \), \( R \))
return \( R \)

insert_tree(itemset \( T \), FP-Node \( R \))
\( x \leftarrow \text{first element of } T \)
if \( \exists \text{item in } N = x \) then
    Increment support in \( N \)
else
    \( N \leftarrow \text{new node with label = } x \text{ and support = 1} \)
    Add children \( N \) to node \( R \)
    insert_tree(\( T - \{x\} \), \( N \))
return

Algorithm 3: FP-tree construction

same item can be connected by the order of their creation. Thus, it is easy to find every pattern with the same item, just start with the corresponding node in the header table.

To find the patterns its not necessary scan the database, only need to scan the tree. This algorithm is recursive, and transforms the problem of identifying long patterns in the identification of smaller patterns. It uses a DFS approach (also called, a pattern-growth approach). The main idea is, select all transactions that contain the least frequent item (least frequent among those that are frequent) and delete this item from them. Obtain a reduced tree with those transactions and process it, calling again the algorithm, and remembering that the itemsets found in the recursion share the deleted item as a prefix. On return, remove the processed item also from all transactions of the tree and start over, i.e., process the second frequent item, etc. When there is only one single path, frequent patterns are all combinations of the items in that path, with the support set to the minimum support of the items contained in each combination.

Almost all other algorithms for pattern mining are based on the two above. Those who generate candidates are apriori-based, and those that do not generate them are pattern growth-based.

3.1.3 Sequence mining

Given a database of sequences, where each sequence is an ordered list of transactions, for example, by transaction time, and each transaction is a set of items, the problem is to discover all sequential patterns with a user-specified minimum support. A sequential pattern also consists of a list of sets of items. The support of this pattern is the percentage of data-sequences that contain the pattern.

This problem was motivated by applications in the retailing industry, including attached mailing, add-on sales, and customer satisfaction. In addition, the results can be applied to many scientific and business domains. For instance, in the medical domain, a data-sequence may correspond to the symptoms or diseases of a patient, with a transaction corresponding to the symptoms exhibited or diseases diagnosed during a visit to the doctor. The patterns discovered using this data could be used in disease research to help identify symptoms/diseases that precede certain diseases.
Sequence pattern mining may give better results than transactional pattern mining, especially for mining semi-structured data, since it considers inter-transaction associations. However, in this domain it probably won’t give more effective results, because fields and almost all subfields in UNIMARC do not have a fixed order, they may appear in any position of a record (There are some recommendations for that order in some subfields, but it is not mandatory)[IFL94]. Even when there is a required order for some subfields (e.g. numeric subfields), this order is the same for any kind of record.

Basically if, for example, the order of the subfields of a music record (e.g. $a $b $c) was always different from the order of the same subfields of a textual record (e.g. $b $a $c), transactional algorithms would consider this as a pattern (two equal itemsets) but sequential algorithms don’t (two different sequences). This means that sequential methods would give less results, but more interesting patterns (they wouldn’t consider music and textual records similar).

Sequential pattern mining is now briefly described, to a better understanding of this technique.

An itemset is a non-empty set of items. A sequence is an ordered list of itemsets. An element is denoted by $(s_1 s_2 \ldots s_n)$, where $s_j$ is an itemset (or simply an element of the sequence). An element is denoted by $(x_1, x_2, \ldots, x_m)$, where $x_j$ is an item. An item can occur only once in an element of a sequence, but can occur multiple times in different elements.

A sequence $(a_1 a_2 \ldots a_n)$ is a subsequence of another sequence $(b_1 b_2 \ldots b_m)$ if there exist integers $i_1 < i_2 < \ldots < i_n$ such that $a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, \ldots, a_n \subseteq b_{i_n}$. For example, the sequence $(3)(4)(5)(8)$ is a subsequence of $(7)(3,8)(9)(4,5,6)(8)$, since $(3) \subseteq (3,8)$, $(4,5) \subseteq (4,5,6)$ and $(8) \subseteq (8)$. However, the sequence $(3)(5))$ is not a subsequence of $(3)(5)$ (and vice versa).

So, most important sequence algorithms are:

**GSP** [SA96, Sri96] The basic structure of the Generalized Sequential Patterns (GSP) algorithm is very similar to the Apriori algorithm (Algorithm 1), except that GSP deals with sequences rather than itemsets. The differences between them are in the details of candidate generation and counting itemsets. $L_k$ and $C_k$ refer to frequent k-sequences and candidate k-sequences respectively.

GSP generates candidate sequences by joining $L_{k-1}$ with $L_{k-1}$, like Apriori. A sequence $s_1$ joins with $s_3$ if the subsequence obtained by dropping the first item of $s_1$ is the same as the subsequence obtained by dropping the last item of $s_2$. The candidate sequence generated by joining $s_1$ with $s_2$ is the sequence $s_1$ extended with the last item in $s_2$. The added item becomes a separate element if it was a separate element in $s_2$, and part of the last element of $s_1$ otherwise. For example, let $s_1 = ((1,2)(3))$, $s_2 = ((2)(3,4))$ and $s_3 = ((2)(3)(4))$. Dropping the first item of $s_3$ results in the sequence $((2)(3))$, that is the same as dropping the last element in $s_2$ and $s_3$. It will be generated two candidates, one by joining $s_1$ and $s_2$: $((1,2)(3,4))$, and other by joining $s_1$ and $s_3$: $((1,2)(3)(4))$. The prune phase is the same as Apriori, candidate sequences that have any subsequence without minimum support are deleted. In the example, first candidate would be deleted if $((3,4))$ was not frequent (the same reasoning for other candidates).

**PrefixSpan** [PHMA+01, PHMA+04] Prefix-projected Sequential pattern mining (PrefixSpan) is a pattern-growth based method and mines the complete set of patterns, greatly reducing the efforts of candidate subsequence generation.

The FP-tree structure explores maximal sharing of common prefix paths in the tree construction by reordering the items in transactions. However, the items (or subsequences) containing different orderings cannot be reordered or collapsed in sequential pattern mining. Thus the FP-tree structures so generated will be huge and cannot benefit mining.

Instead of repeatedly scanning the entire database and generating and testing large sets of candidate sequences, one can recursively project a sequence database into a set of smaller databases associated
with the set of patterns mined so far and, then, mine locally frequent patterns in each projected database. The projected database of an item is the subsequence from the position where the item is in each transaction, to the end of that transaction. In each projected database, sequential patterns are grown by exploring only local frequent patterns.

FreeSpan (i.e., Frequent pattern-projected Sequential pattern mining), proposed before PrefixSpan, works like it but, since a subsequence may be generated by any substring combination in a sequence, projection in FreeSpan has to keep the whole sequence in the original database without length reduction (if a pattern appears in each sequence of a database, its projected database does not shrink, except for the removal of some infrequent items). Moreover, since the growth of a subsequence is explored at any split point in a candidate sequence, it is costly. Therefore, PrefixSpan outperforms FreeSpan.

spade [Zak01, Zak98] Sequential PAttern Discovery using Equivalence classes (SPADE) utilizes combinatorial properties to decompose the original problem into smaller sub-problems, that can be independently solved in main-memory using efficient lattice search techniques, and using simple join operations. It uses a vertical id-list database format, where each sequence is associated to a list of objects in which it occurs, along with the time-stamps. All frequent sequences can be enumerated via simple temporal joins (or intersections) on id-lists. All sequences are discovered in only three database scans. The main steps include the computation of the frequent 1-sequences and 2-sequences, the decomposition into prefix-based parent equivalence classes, and the enumeration of all other frequent sequences via BFS or DFS search within each class (also called, a lattice-based approach).

spam [AFGY02] Sequential PAtern Mining (Spam) uses a DFS strategy for mining sequential patterns. Unlike BFS strategy, that first outputs all patterns of length one, then all patterns of length two, and so on, SPAM outputs sequential patterns of different length at the same time. It also implement various pruning mechanisms to reduce the search space. SPAM uses a vertical bitmap data layout allowing for simple and efficient counting. It traverses candidate sequences as defined by a lexicographic sequence tree, where each sequence is generated by adding a new transaction with a single item (S-step) or adding an item to the last itemset (I-step).

The algorithm is especially efficient when the sequential patterns in the database are very long.

As said before, there are also other sequence methods that consider constraints like maximum or minimum gaps, regular expressions and taxonomies, and methods for mining closed sequences (as well as for closed trees and graphs), but they will not be considered in this study, because they are not relevant for bibliographic data.

3.1.4 Tree mining

Tree mining methods[CMNK04] mine different kinds of tree patterns, such as ordered/unordered embedded trees. Its goal is to find all common subtrees in a forest (a collection of trees), or even all common sub-forests (disconnected subtrees).

Tree mining is related to tree isomorphism and tree pattern matching[Zak05]. They both deal with induced subtrees, while tree mining also deals with embedded subtrees.

A tree is an acyclic connected graph, and a forest is a collection of trees. The tree is usually rooted and it can be ordered if the children of each node have an order, and labeled if each node is associated with a label. Embedded subtrees allow not only direct parent-child branches (like induced subtrees), but also
ancestor-descendant branches. Embedded subtrees are able to extract patterns “hidden” (or embedded) deep within large trees which might be missed by the traditional definition.

XML data can easily be represented in a tree-like structure. Tree pattern mining considers the hierarchy of data, therefore, it probably will extract more interesting patterns from XML, and patterns that transactional or sequential pattern mining cannot find.

Canonical representations for labeled trees are closely related to the data structures used to store trees in memory or in disk files. Instead of the standard data structures, such as the adjacency-matrix, the adjacency-list, and the first-child-next-sibling representation, many tree mining algorithms also use other representations, like string encodings. The string encoding starts with an empty string, and then perform a Depth-First Search (DFS) starting at the root, adding the current node value to the string. Whenever a backtrack is made from a child to its parent, a unique symbol '-1' is added to the string. This format allows to represent trees with arbitrary number of children for each node. This string encoding is more space-efficient than other representations.

There are several algorithms, but most relevant are:

**Tree Miner** [Zak02] The TreeMiner algorithm developed by Zaki for mining frequent ordered embedded subtrees uses a lattice-theoretical approach (like SPADE) to decompose the original search space (lattice) into smaller pieces (sub-lattices) which can be processed independently in main memory[Zak98, Zak05]. It follows the combined depth-first/breadth-first traversal idea to discover all frequent embedded subtrees from a database of rooted ordered trees: First, all itemsets of size 1 are generated and counted. Then, first item is joined with other frequent 1-itemsets and the obtained set of 2-itemsets are recursively investigated further. Only after all itemsets containing first item have been investigated, other itemsets are considered. All candidate \((k + 1)\)-subtrees are obtained by joining two frequent embedded \(k\)-subtrees whose string encodings share the prefix up to the \((k − 1)\)th nodes (called prefix class).

For support counting, TreeMiner uses a method in which the database is rewritten in a vertical representation of scope lists. For each frequent subtree with size \(k\), the corresponding scope list records the occurrences of the subtree and, per occurrence, scope information about its last node in a pre-order walk, i.e., the so-called rightmost vertex. For each correct candidate (that is a descendant or an embedded sibling), increment the number of occurrences of the tree.

Tree Miner takes advantage of anti-monotone property (all subtrees of a frequent tree are frequent) and of another useful property of the string encodings for rooted ordered trees: removing either one of the last two nodes at the end of the string encoding of a rooted ordered tree (with correspondent adjustment to the number of backtrack symbols) will result in the string encoding of a valid embedded subtree.

PatternMatcher, also proposed by Zaki, uses an hash-tree data structure, employs a breadth-first iterative search for frequent subtrees, and its high-level structure is similar to Apriori (Algorithm 1). Tree Miner outperforms it, especially because of the number of candidates it generates.

**XSpanner** [WHP+04] This algorithm performs a DFS like TreeMiner, but in a different way. The basic idea is to recursively create the pattern by joining the new candidates to the previous found patterns. The algorithm is similar to PrefixSpan (a DFS in the search space, restricting it at each step, using its concept of projected database), except in how it manipulates the data and in how it generates frequent patterns: First, XSpanner represents a tree by its Label-Level-Sequence (\(L^2\)sequence) in a string: When performing a DFS, for each item, its label and level is added to the string. Then, there are only three patterns for each frequent item: The item is a direct descendant, a sibling or a direct descendant of a sibling of the last item found.
X3Miner [TDF+05] X3Miner algorithm is also similar to TreeMiner. But they also store a direct parent pointer (dpp) of each node in string representation ([position, scope](dpp)), called xstring, and use the xstring to generate candidates.

The algorithm can process an XML document directly taking into account the values of the nodes present in the XML tree. The frequent itemsets generated will contain node names and values in comparison to the TreeMiner approach which only generates frequent tree structures and does not process an XML document in the form that is mostly present. The way the string representation of a sub-pattern is used as a key in an hash multimap may be the cause for extra computational cost as the values and names of each node are included, and so when hashcode is calculated it is more expensive. The algorithm is still performing candidate generation efficiently on large XML documents but as frequent sub-patterns grow there will be a large increase in the computational cost.

3.1.5 Graph mining

Given a database of graph objects, the goal of graph mining[WM03] is to find all the commonly occurring sub-graph patterns.

Unlike trees, graphs allow multiple hierarchies, cycles and arbitrary relations among entities or attributes. In fact, the root of the complexity of graph algorithms is often the existence of cycles in the graph. In many cases, the number of cycles in graph instances in a database is limited, or the graphs may even be acyclic. A tree is a special case of a graph (directed acyclic graph). Therefore, graph mining techniques can also be applied to trees. However, these algorithms are more complex and less efficient than specific algorithms for trees.

XML data can easily be represented in a tree-like or graph-like structure.

A labeled graph \( G = (V, E, \Sigma, L) \) consists of a vertex set \( V \), an edge set \( E \), an alphabet \( \Sigma \) for vertex and edge labels, and a labeling function \( L : V \cup E \rightarrow \Sigma \) that assigns labels to nodes and edges. A graph is *directed* if each edge is an ordered pair of nodes; it is *undirected* if each edge is an unordered pair of nodes. A *path* is a list of nodes of the graph such that each pair of neighboring nodes in the list is an edge of the graph. The length of a path is defined by the number of edges in the path. A cycle is a path such that the first and the last nodes of the path are the same. A graph is *acyclic* if the graph contains no cycle. An undirected graph is *connected* if there exists at least one path between any pair of nodes, *disconnected* otherwise.

At the core of any frequent subgraph mining algorithm are two computationally challenging problems: 1) subgraph isomorphism: determining whether a given graph is a subgraph of another graph and 2) an efficient scheme to enumerate all frequent subgraphs.

These methods are:

**AGM** [IWNM02] The basic principle of the Apriori-based Graph Mining (AGM) is similar to the Apriori algorithm for basket analysis. Starting from frequent graphs where each graph is a single vertex, the frequent graphs having larger sizes are searched in bottom up manner by generating candidates having an extra vertex. AGM can mine various types of subgraphs including general subgraph, induced subgraph, connected subgraph, ordered subtree, unordered subtree and subpath. To distinguish a subgraph from another, it uses a canonical labeling scheme based on the adjacency matrix representation.

**FSG** [KK02] The Frequent SubGraph discovery (FSG) algorithm uses a BFS approach to discover the lattice of frequent subgraphs. The size of these subgraphs is grown by adding one-edge-at-a-time, and the frequent pattern lattice is used to prune non downward closed candidate subgraphs.
FSG employs a number of techniques to achieve high computational performance including efficient canonical labeling, efficient candidate subgraph generation algorithms, and various optimizations during frequency counting. It also encodes graphs using adjacency matrices like AGM.

**gSpan** [YH02] Graph-based Substructure pattern mining (gSpan) finds the frequently occurring subgraphs following a DFS approach. This approach also uses the idea of canonical labeling, but it uses a tree representation (into an adjacency-list) of each graph instead of the adjacency matrix to define the code of the graph. Because the code is derived in the DFS algorithm, this code is called DFS code. By applying this DFS coding and DFS search, gSpan can derive complete set of frequent subgraphs over a given \( \text{min}_\text{sup} \) in a very efficient manner in both computational time and memory consumption. In addition, gSpan does not keep the information about all previous embeddings of frequent subgraphs which saves the memory usage.

**MoFa** [BB02] The Molecular Fragment Miner (MoFa) is a chemical substructure mining algorithm that finds frequent substructures (connected subgraphs) using a DFS approach. To reduce the number of subgraph isomorphism operations, it keeps the embeddings of previously discovered subgraphs and tries to extend the embeddings by one edge. Therefore, it is a pattern-growth based algorithm. MoFa also exploits a local order of the atoms and bonds of a fragment to prune the search tree, which results in faster search and allows for a restricted depth first search algorithm. The algorithm allows us to focus on fragments that help to discriminate between different classes of molecules.

**FFSM** [HWP03] Fast Frequent Subgraph Mining (FFSM) incorporates the join-based candidate generation scheme used by the horizontal algorithms into the vertical frequent subgraph mining paradigm proposed by gSpan (DFS code and search). By the combination of the candidate generation and extension, FFSM is able to prune unnecessary candidates aggressively.

**Gaston** [NK04] GrAph/Sequence/Tree extractiON (Gaston) search first for frequent paths, then frequent free trees (undirected graph that is connected and acyclic) and finally cyclic graphs. Ideally, the algorithm should behave like a specialized free tree miner when faced with free tree databases, but should also be able to deal with graph databases efficiently. It is a combined depth-first/breadth-first algorithm that uses a procedure which consists of three phases. First paths are grown and then trees are constructed from paths. Graphs are only allowed to grow from a spanning tree of that graph. Each graph is normalised and if it has already been enumerated, it is discarded. Otherwise, it is stored in a hash table.

**ADI-Mine** [WWP+04] ADI-Mine is an improvement of algorithm gSpan. At high level, the structure as well as the search strategies of ADI-Mine and gSpan are similar. The critical difference is on the storage structure for graphs: ADI-Mine uses ADjacency Index (ADI) structure and gSpan uses adjacency-list representation. In ADI-Mine, the graphs are stored in the ADI structure. The edges are indexed by their labels. Then, the graphs that contain the edges can be retrieved immediately. Moreover, all edges with the same labels are linked together. That helps the test of subgraph isomorphism substantially. Furthermore, using the index of edges by their labels, only the graphs that contain the specific edge will be loaded into main memory for further subgraph isomorphism test. Irrelevant graphs can be filtered out immediately by the index.

Most of the previous studies focus on pruning unfruitful search subspaces effectively, but few of them address the mining on large, disk-based databases, such as mining large collections of XML documents. This ADI structure is simple and effective, and was developed for large graph databases [WWZ+05].

28
3.2 Data Mining over Bibliographic Data

Libraries have made huge investments in creating and maintaining rich, structured information describing the resources in their collections. This data embodies considerable value by supporting access and inventory control. It also represents potential value in terms of:

- Knowing more about the characteristics of library collections;
- Generating interesting and innovative data displays;
- Providing intelligence to support a range of library decision-making needs, including collection development, digitization and preservation.

There is untold value in bibliographic information, but it is largely untapped. If libraries are to realize the full value of their bibliographic data, steps must be taken to release this value in innovative and useful ways, in order to create value for librarians and users.

With widespread adoption of computerized catalogs and search facilities over the past thirty years, library and information scientists have often used bibliometric methods (e.g. the discovery of patterns in authorship and citation within a field) to explore patterns in bibliographic information. During the same period, various researchers have developed and tested data mining techniques, like advanced statistical and visualization methods to locate non-trivial patterns in large data sets. The term bibliomining is sometimes associated to the use of data mining to examine library data records usage.

Forward-thinking authors in the field of library science began to explore sophisticated uses of library data some years before the concept of data mining became popularized [NS03].

1987 Nutter explored library data sources to support decision making;

1990 Johnston and Weckert developed a data-driven expert system to help select library materials and Vizine-Goetz, Weibel, & Oskins developed a system for automated cataloging based on book titles;

1996 Articles on (Mancini) extracting data to support system management decisions, (Atkins) extracting frequencies to assist in collection decision-making, and (Peters) examining transaction logs to support collection management;

1998 Banerjee focused on describing how data mining works and ways of using it to provide better access to the collection;

1999 Lawrence, Giles, and Bollacker created a system to retrieve and index citations from works in digital libraries, and Gutwin, Paynter, Witten, Nevill-Manning, and Frank used text mining to support resource discovery;

2000 Guenther discussed data sources and bibliomining applications, but focused on the problems with heterogeneous data formats. Doszkocs discussed the potential for applying neural networks to library data to uncover possible associations between documents, indexing terms, classification codes, and queries. And Liddy combined natural language processing with text mining to discover information in “digital library” collections.

---

1OCLC Data Mining Projects: http://www.oclc.org/research/projects/mining
These projects all shared a common focus on improving and automating two of the core functions of a library: acquisitions and collection management. What these projects did not discuss was the use of library data to support strategic management decisions for libraries and their host institutions. A few authors have begun to address this need by focusing on understanding library users: Schulman (1998) discussed using data mining to examine changing trends in library user behavior; Sallis, Hill, Jance, Lovettier, and Masi (1999) created a neural network that clusters digital library users; and Chau (2000) discussed the application of Web mining to personalize services in electronic reference.

More recent examples are internet giants such as Amazon and Google. They provide valuable lessons on the importance of squeezing the full value from available data. Whether in the form of book recommendations (if you like this book, you’ll also like that book), search result rankings, targeted advertising, or collection views (e.g., Google Scholar).

In fact, the basis for the use of library data mining has been finding what users need to know and how well those needs are served. Almost all data mining projects in libraries are about users and data characterization (using clustering or classification techniques and text mining), usage patterns identification (association/sequence analysis and web mining), digitization and visualization.

Recent projects belong to the Online Computer Library Center (OCLC). OCLC is a nonprofit, membership, computer library service and research organization dedicated to the public purposes of furthering access to the world’s information and reducing the rate of rise of library costs. More than 69,000 libraries in 112 countries and territories around the world use OCLC services to locate, acquire, catalog, lend and preserve library materials. OCLC and its member libraries cooperatively produce and maintain WorldCat – the OCLC Online Union Catalog. OCLC Research has a number of projects currently underway in the Data-Mining Research Area:

**Systemwide Print Book Collection** Print collections have been changing, as the distinction between local and external resources is increasingly blurred due to resource sharing. Publications are now distributed across a wide number of libraries, and any mass digitization strategy that ignores this distributional reality is likely to omit numerous works. Digitization combined with network technologies creates opportunities for one “copy” of a resource to be shared across many libraries. The focus of this project is, therefore, to analyze the size and characteristics of aggregate print book holdings, with an emphasis on implications for digitization and preservation decision-making.

**Anatomy of Aggregate Collections The Example of Google Print for Libraries**

This project offers some perspectives on the Google Print Library Project (GPLP - a Mass Digitization program) in light of what is known about library print book collections in general, and those of the Google 5 in particular, from information in OCLC’s WorldCat bibliographic database and holdings file. Its purpose is to explore a few basic questions raised by GPLP, as well as the implications of any mass digitization initiative;

**Audience Levels** The Audience Level prototype and this project seek to explore various ways to leverage intelligence from system files, and “make data work harder”. Determining a monograph’s audience level is a challenge because cataloging rules generally do not require inclusion of this information. Thus, many bibliographic records have no explicit indicator of target audience. The purpose of this project is to infer materials’ target audience, or audience level, using holdings information in WorldCat;

**“Last Copy”** The aim is to identify rare or unique materials in individual library collections. The bibliographic records and, in selected cases, the items themselves, were examined to determine characteristics and to identify the items whose content is at-risk;
WorldMap People can be inundated by an overwhelming amount of information. One method for organizing and retrieving geographically-based information is to use maps for data visualization. The OCLC WorldMap is a prototype system that provides an interactive visual tool for selecting and displaying international library holdings represented in WorldCat, and publishing, library, cultural heritage, and collection data. This data can then be used to provide information for decision making in regards to remote storage, collection management, marketing, and cooperative collection development, preservation, and digitization;

Mining for Digital Resources This project aims to identify consistent cataloging patterns in existing bibliographic records, to identify and characterize digital resources cataloged in WorldCat.

Comparative Collection Assessment It looks at collection development, assessment, and resource sharing for print- and e-book collections. Its goals are: Identify and characterize interlibrary loan (ILL) and usage patterns; Identify and characterize book-collection holdings by library type, both domestic and international; Formulate strategies for eBook collection development; And contribute to the establishment of community-wide standards for eBook collections;

Publisher Name Server This project will prototype a service that resolves ISBN prefixes to publisher name; resolves variant publisher names to a preferred form; and captures and makes available various publisher attributes (e.g., location, language, genre/format, dominant subject domain, etc. of the publisher’s output). In essence, it will support advanced collection intelligence by: facilitating the reliable clustering of collected objects based on their issuing entity, and gaining intelligence about the nature of individual publishers which can in turn be used to reveal critical collection intelligence, acquisition patterns and user behavior.

Other projects to consider are: “Data Mining MARC to find: FRBR?”[HM02]: The main idea in this project was to look at several records with similarities of an Online Public Access Catalog (OPAC) on a certain level (e.g. same author), analyze the differences, try to identify works, expressions and manifestations, and recognize problems that arise with that analysis. MARC data from both Finish and Norwegian nation bibliographies were reviewed in light of the FRBR model. They concluded that, although the information present in MARC records is enough to identify the entities (work, expression and manifestation), the accuracy and formal syntax is too simple to be done properly by software. Some of the results may be used to present better hit lists in OPAC. The project presented two suggestions for an OPAC user interface based on the ideas of the FRBR study and on the results of the project. This project is the one that most resembles this work.

Melvyl Recommender Project: This Project, from California Digital Library, had the purpose of exploring methods of closing the gap between features that library patrons want and have come to expect from information retrieval systems and what libraries are currently equipped to deliver. The project team has conducted exploratory development work in five topic areas: use of a text-based discovery system, spelling correction, user interface strategies, relevance ranking, and recommending. They used their own dynamic FRBR algorithm to search for records and, for generating recommendations, they explored two major strategies: an approach based on the mining of circulation data (i.e. “patrons who checked this out also checked out that”), and an approach based on similarities in the content of bibliographic records (“more like this”).

Multilingual Document Retrieval (Ad-Hoc), started at CLEF 2008\(^2\) and tested mono- and cross-language text retrieval. The task was organised in collaboration with TEL and searching was on collections derived from the TEL archives in English, French, and German.

\(^2\)http://www.clef-campaign.org/
As said before, many of these projects are concerning to digitization and visualization, and very few are about mining patterns in records. This work pretends to find co-relations between records in UNIMARC, to see which characteristics occur simultaneously. These patterns can then be used to validate librarian practices and to serve as a basis for the establishment of the translation rules between UNIMARC and FRBR.

3.3 Challenges

Despite the UNIMARC define a common structure for the exchange of bibliographic records, in a machine-readable form, this format is not suitable for data mining, since it is not tabular. The data cannot easily be arranged in a table or in a systematic arrangement by columns and rows. And traditional data mining algorithms deal with a table. This limitation implies an extra step of pre-processing of data in UNIMARC, to convert it to a table. This has several challenges:

1. There are more than 1000 possible fields and subfields, which means that the table would have more than 1000 columns. Besides the memory and time needed to store and process all these columns, most available commercial data mining tools impose a limit of columns below 1000.

   Other related problem is the fact that the resulting patterns are potentially very long, which leads to the need to generate too many candidates (for apriori techniques), and/or to keep in memory very long trees (for pattern growth based techniques).

2. Some data is not categoric neither numerical (the title of a record, for example), and data mining techniques only deal with these kinds of data. There are a few tools that convert data into categorical values (reading the table and storing all values taken by each attribute). However, the number of possible values increases with the number of records, as well as the memory needed to store them. And when we talk about bibliographic data, we talk about over a million of records per library, which leads us to another problem: The table may not fit in memory, nor by itself, neither during the mining process.

3. Data is very sparse, which leads to much wasted space. A record have values only for some fields and subfields, that are usually less than a half of all possible fields/subfields. This means that each row of the table would have a lot of null values.

4. Several values for some fields/subfields are the combination of other several elements (for example, the leader of a record is one string that combines implementation codes, lengths, type of record, etc.). This makes more difficult the pre-processing step. If we consider these elements in the table, each of them will result in another column (on top of already high number of columns). If we do not consider them, we will miss some possible patterns within these composed fields.

5. Other problems that difficult the pre-processing and the creation of the table, like the possibility of a record has repeated fields/subfields with the same or different values (which one to consider?), and the existence of two indicators for each field that also can have several possible values (more columns...), etc.

3.4 Solution

All these challenges make almost infeasible the application of most of the previous algorithms (section 3.1) to this bibliographic data, specially because they deal with one table.
A solution is to find a more efficient way of representing the data for analysis, without wasted space. If we split the data into several tables and mine those tables separately, many of the previous memory problems are solved. A way to do this is to create a star-based data warehouse to this data.

A data warehouse is a subject-oriented, integrated, time-variant, and non-volatile collection of data in support of management’s decision-making process [Inm96]. In terms of data modelling, a data warehouse consists of one or several dimensional models that are composed of a central fact table and a set of surrounding dimension tables, each corresponding to one of the dimensions of the fact table. The most used dimensional model is the star schema, which consists of multiple dimension tables that are associated by foreign keys to a central fact table.

This work proposes modelling the bibliographic data in a star schema (see figure 3.3), with five dimensions and a fact table, described below:

![UNIMARC star schema](image)

**Figure 3.3: UNIMARC star schema**

**Dimension Attribute:** Holds every field and possible subfields. It has four fields:

- **AttributeId:** An integer that identifies each possible field or subfield.
- **Block:** A digit that represents the block of the respective field. It ranges from 0 to 8 (Block 9 is not considered in this work, once they are for local use and definition only). The block corresponds to the first digit of the respective field.
- **Field:** The number that represents the field or controlfield. It ranges from 000 to 899.
- **Subfield:** Two characters that reference the subfield identifier. The first is the character $, and the second can be a letter or a digit. (Note that some controlfields may not have subfields)

Its hierarchy is Block >> Field >> Subfield;

As example, the subfield 100$a is represented as Block = 1; Field = 100; Subfield = a.

**Dimension Value:** To keep every value of every field; It only has two fields:
Table 3.1: Characters that influence the type of a record

| ValueId: An integer that identifies each value. |
| Value: The string corresponding to the value of a field. |
| Dimension Leader: Identifies the record and deals with control values (like lengths). The Leader has several attributes, corresponding to each element in the UNIMARC’s record leader. This is the Key dimension: each record has only one leader, and there are no two different records with the same leader; |
| Dimension TypeOfRecord: To keep information about each possible type of record. The UNIMARC characters responsible for the type of each record are: |
| bibliographicLevel: Other character that influences the type of the record (table 3.0(b)). |
| The possible types were grouped like described in section 2.2. |
| dimension Library With information from libraries and formats. This dimension in this star schema allows us to represent and analyse data from different libraries and data in different formats. We can group data from each library and find patterns relating to them. |
| Fact table: Linking all dimensions. Each row is a triple (leaderId, attributeId, valueId), representing the value that the record (with that leader) took in the respective field (or attribute). |

This star schema solves many of the problems described above. Data is split into several tables, which helps dealing with the high number of attributes and records. Taking a divide and conquer approach, i.e. mining each dimension separately, allows us to store only their frequent itemsets, and use the fact table to combine them and construct the final result. In other words, there is no need to keep all data in memory. The intermediate results are smaller, therefore, less candidates and smaller trees.

The star schema also solves the problem of non categorical data. Each possible value is stored in the respective dimension. There is no need of extra structures to store them. It also solves the problem of sparse data and the resulting wasted space. The fact table only stores the existing pairs of attribute -value, therefore there are no null values.

As said before, the leader is the combination of several elements of the record. Two of them characterize the type of the record (type of record and bibliographic level elements). Once there are some fields
specific for some type of records, this is an important aspect to analyse. Therefore, Leader dimension splits all those elements and is used, not only to identify the record, but also to aggregate the records according to their types and find the patterns within each type.

Another advantage of this star schema is that it allows the automation of the studies made above, about the fields usage in UNIMARC. OLAP techniques can be used to explore data in the star, and using the usual OLAP queries within the data warehouses, the statistics will be automatic. For example, if we make an OLAP cube from this star, we can answer questions like: What is the frequency of a field? (count in the fact table, how much times the respective attributeId appears); How much times a subfield appear in a record of a determined subtype? And in the respective type? (using grouping and measures like sum), etc.

Along with its advantages, the star schema brings other challenges, mainly because now it is necessary to mine multiple tables. Next chapter describes those challenges and the proposed algorithm.
Chapter 4

Multi-Relational Pattern Mining

4.1 Introduction

While most existing data mining approaches look for patterns in a single data table, multi-relational data mining (Multi-Relational Data Mining (MRDM)) approaches look for patterns that involve multiple tables (relations) from a relational database. In recent years, the most common types of patterns and approaches considered in data mining have been extended to the multi-relational case and MRDM now encompasses multi-relational (MR) association rule discovery, MR decision trees and MR distance-based methods, among others. MRDM approaches have been successfully applied to a number of problems in a variety of areas [D03].

From those works, just a few are dedicated to frequent itemset mining on star schemas [CJS00, CS01, NFW02, XX06, RV04, Gar08, Kan05].

This work aims to find frequent patterns in a set of tables of a data warehouse, following a star schema, without materializing the denormalization of its tables.

To simplify the work, we will consider the star of figure 4.1.

Figure 4.1: A Movies data warehouse schema [Wie89]
At first glance, it may seem easy to join the tables in a star schema, and then do the mining process on the joined result\cite{NFW02}. However, when multiple tables are joined, the resulting table will be much larger and the mining process more expensive and time consuming. There are two major problems: First, in large applications, often the join of all related tables cannot be realistically computed because of the distributed nature of data, large dimension tables and the many-to-many relationship blow up. Second, even if the join can be computed, the multifold increase in both size and dimensionality presents a huge overhead to the already expensive pattern mining process:

1. the number of columns will be close to the sum of the number of columns in the individual tables.
2. If the join result is stored on disk, the I/O cost will increase significantly for multiple scanning steps in data mining;
3. For mining frequent itemsets of small sizes, a large portion of the I/O cost is wasted on reading the full records containing irrelevant dimensions;
4. Each tuple in a dimension table will be read multiple times in one scan of the joined result. The number of times that a tuple appears in the fact table is the number of times the whole tuple will be read in the joined result.

One of the great potential benefits of MRDM is the ability to automate this process to a significant extent. Fulfilling this potential requires solving the significant efficiency problems that arise when attempting to do data mining directly from a relational database, as opposed to from a single pre-extracted flat file\cite{Dom03}.

The proposed algorithm is an adaptation of FP-Growth \cite{HPY00} to mine a star schema. The main idea is to adapt the construction of the FP-Tree, so that FP-Growth can run.

Like FP-Growth, it scans each table only twice: first to count the support of each item, and second to construct the FP-Tree. It is divided in 3 stages:

1. **Support Counting:** The fact table is scanned to count the support of each foreign key.

2. **Local Mining:** An FP-Tree is constructed for each dimension table (DimFP-Tree), with a slight modification of the original FP-Tree, taking into account the support calculated in the previous step.

3. **Global Mining:** The FP-Trees of each dimension are combined to form a Super FP-Tree, according to each fact and an established order of dimensions. This Super FP-Tree is then mined with FP-Growth, without a change, giving all the frequent patterns.

Several orders for the dimensions were studied and are presented and compared in this paper.

The rest of this chapter is organized as follows. Section 4.2 presents the related work on MRDM on data warehouses. The proposed idea is described on section 4.3. Section 4.4 gives some experimental results and section 4.5 presents the conclusions.

### 4.2 Related Work

The works related to multi-relational pattern mining are increasing.

Jensen and Soparkar (2000) \cite{CJS00} presented an algorithm based on Apriori \cite{AS94} when the database is organized in a star schema. In the first stage it generates frequent itemsets in each single table using a slightly modified version of Apriori, and then looks for frequent itemsets whose items belong to distinct tables via a multi-dimensional count array. It does not construct the whole joined table and process each row as the row is formed, thus storage cost for the joined table is avoided.

Cristofor and Simovici (2001) \cite{CS01} eliminated the explosion of candidates present in jensen’s algorithm. They also produce the local patterns existing among attributes of the same table, i.e. patterns
that are frequent with respect to their dimension table, but not with respect to the relationship (or fact) table.

Ng et al. (2002) [NFW02] proposed an efficient algorithm for a star schema without actually performing the join operation. The idea is to perform local mining on each dimension table, and then “bind” two dimensional tables at each iteration, i.e. mine all frequent itemsets with items from two different tables without joining them. After binding, those two tables are virtually combined into one, which will be “binded” to the next dimension table. They designed a special tree structure, prefix tree, which is a compressed representation of the fact table, to speed up the calculation of support values. Each node of the prefix tree involves one value of a foreign key and one corresponding counter. The tree is constructed level by level, so that nodes in the same level belong to the same dimension table. The algorithm also uses vertical data format. Experiments showed that the approach of “mining before join” outperforms the approach of “join before mining” even when the latter adopts known to be fastest single-table mining algorithm.

Xu and Xie (2006) [XX06] proposed a novel algorithm, MultiClose, which discover frequent closed itemsets in data warehouses following a star schema without materializing join tables. It first converts the dimension tables to vertical data format, and then mines each of them with a closed algorithm. After local mining, frequent closed itemsets are stored in two-level hash table result trees and the frequent closed itemset across two tables are discovered by traversing the result trees. They state that this algorithm is the first one in the literature that applies closed itemset techniques to multiple tables mining.

Several multi-relational methods to analyse data have been developed by the Inductive Logic Programming community over the recent years, being WARMR [DR97] and FARMER [NK01] the most representative ones. The ILP approaches achieve a good accuracy in data analysis. However they are usually not scalable with respect to the number of relations and the number of attributes in the database. Therefore these approaches are inefficient for databases with complex schemas. Another drawback of the ILP approaches is that they need the data in the form of prolog tables.

There are other algorithms for finding multi-relational frequent itemsets, however they just consider one common attribute at a time, and therefore, they cannot mine a star schema. They would have to run as much times as the number of dimensions, because there is no attribute common to all the tables. Instead, the fact table has one attribute in common with each dimension, and the dimensions have no common attribute between them. The patterns discovered will not reflect the relationships between the dimensions. Examples are Connection [RV04] and its extension ConnectionBlock [GV08], both based on FP-Growth and on the intersection of local patterns, and MRFP-Growth [Kan05], also based on FP-Growth, which constructs an FP-Tree with the common attribute’s keys corresponding to local patterns and mines it to find global patterns.

### 4.3 Mining Stars

Consider a relational database with a star schema. There are multiple dimension tables, which we will denote as A, B, C, ..., each containing only one primary key denoted by transaction id (tid), some other attributes and no foreign keys. In fact, what we want is to discover potentially useful relationships among the attributes, other than primary keys. (Note that relationships may exist among the attributes across distinct dimension tables.) We also assume that attributes take categorical values (Numerical values can be partitioned into ranges, and hence be transformed to categorical values [RS98]). The set of values for an attribute is called the domain of the attribute.

In order to simplify our discussion we assume the fact table, denoted as FT, only contains the tids from dimension tables as foreign keys (tidA, tidB, tidC, ...). If the fact table contains some fields other than primary keys, we can place these fields into an extra dimension table and insert a new foreign key
corresponding to it into the fact table. They are considered facts or measures, and are not considered on this paper.

Example 1: Let’s consider, as example, a sample of a real database following the star schema of figure 4.1, a movies database donated by Gio Widerhold [Wie89], which collects data about more than 10,000 movies. The sample used has three dimension tables A (Award), B (Studio) and C (Movie) showed on the left side of Table 4.1. From now on, we will refer to an attribute by concatenating the table name to it, to assure it is unique and well understood. For example, attribute “Name” of table Award corresponds to “AwardName”, which is different from “StudioName” (attribute “Name” of table Studio).

Tables on the right are a conceptual representation of tables on the left, where $a_i$, $b_i$ and $c_i$ denote the tid of dimension tables A, B and C, respectively, and $x_i$, $y_i$ and $z_i$ denote each possible item of A, B and C. Each item has the form of “attribute = value”. For example, $x_1$ corresponds to “AwardName = None”.

This example will be used to show how the proposed algorithm works, with a minimum support equals to 40% of the database. The database has 10 transactions, therefore 40% of the database corresponds to 4 transactions. This means that an itemset is frequent if its support is no less than 4 transactions.

<table>
<thead>
<tr>
<th>Award</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>tid</td>
<td>Itemsets</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$x_1$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$x_2,x_3,x_4$</td>
</tr>
<tr>
<td>$a_3$</td>
<td>$x_5,x_6,x_7$</td>
</tr>
<tr>
<td>$a_4$</td>
<td>$x_8,x_6,x_7$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Studio</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>tid</td>
<td>Itemsets</td>
</tr>
<tr>
<td>$b_1$</td>
<td>$y_1$</td>
</tr>
<tr>
<td>$b_2$</td>
<td>$y_2,y_3,y_4$</td>
</tr>
<tr>
<td>$b_3$</td>
<td>$y_5,y_3,y_6$</td>
</tr>
<tr>
<td>$b_4$</td>
<td>$y_7,y_3,y_6$</td>
</tr>
<tr>
<td>$b_5$</td>
<td>$y_8,y_3$</td>
</tr>
<tr>
<td>$b_6$</td>
<td>$y_9,y_3,y_6$</td>
</tr>
<tr>
<td>$b_7$</td>
<td>$y_{10},y_{11},y_6$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Movie</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>tid</td>
<td>Itemsets</td>
</tr>
<tr>
<td>$c_1$</td>
<td>$z_1,z_2$</td>
</tr>
<tr>
<td>$c_2$</td>
<td>$z_3,z_4$</td>
</tr>
<tr>
<td>$c_3$</td>
<td>$z_5,z_4$</td>
</tr>
<tr>
<td>$c_4$</td>
<td>$z_6,z_4$</td>
</tr>
<tr>
<td>$c_5$</td>
<td>$z_7$</td>
</tr>
<tr>
<td>$c_6$</td>
<td>$z_8,z_2$</td>
</tr>
<tr>
<td>$c_7$</td>
<td>$z_9,z_4$</td>
</tr>
<tr>
<td>$c_8$</td>
<td>$z_{10},z_1,z_1$</td>
</tr>
<tr>
<td>$c_9$</td>
<td>$z_{12},z_4$</td>
</tr>
<tr>
<td>$c_{10}$</td>
<td>$z_{13},z_4$</td>
</tr>
</tbody>
</table>

Table 4.1: Dimension Tables A (Award), B (Studio) and C (Movie)

The following definitions are some common terminology that will be used.
Table 4.2: Fact table and the frequent itemsets corresponding to each tid

### Definition 1
- Let \( I = \{i_1, i_2, \ldots, i_m\} \) be a set of distinct literals, called items. A subset of items is denoted as an itemset. A transaction \( T = (t_{id}, X) \) is a tuple where \( t_{id} \) is a transaction-id and \( X \) is an itemset in \( I \). Each table, in a relational database \( D \), is a set of transactions.

### Definition 2
- The **support** (or occurrence frequency) of an itemset \( I_T \), is the number of transactions containing \( I_T \) in the database. An itemset \( I_T \) is frequent if its support is no less than a predefined minimum support threshold, \( \sigma \).

In a database modelled as a star schema, where there are several tables, we have to make more specific definitions:

### Definition 2.1
- The **local support** of an itemset \( I_T \), with items belonging to a table \( A \) (\( A.\text{localSup}(I_T) \)), is the number of transactions containing \( I_T \) in the table \( A \). For example, on table \( A \) of table 4.1, \( x_1 \) appears in one transaction \( (tid_A = a_1) \) and \( x_6 \) in two \( (tid_A = a_3 \text{ and } a_4) \), therefore \( A.\text{localSup}(x_1) = 1 \) and \( A.\text{localSup}(x_6) = 2 \).

### Definition 2.2
- The **global support** of an itemset \( I_T \) (\( \text{globalSup}(I_T) \)), is the number of transactions of the fact table containing all the \( tid \)s that contain \( I_T \).

Let \( tid(I_T) \) be the set of \( tid \)s that contain \( I_T \). Global support can be defined as:

\[
\text{globalSup}(I_T) = \sum_{\text{tid}(I_T)} \text{FT.localSup(tid)}
\]  

Following the example above, \( \text{globalSup}(x_1) = 3 \), because \( \text{tid}(x_1) = \{a_1\} \), appears three times in the fact table (Table 4.2(a)). By the same reasoning, for \( x_6 \) (\( \text{tid}(x_6) = \{a_3, a_4\} \)):

\[
\text{globalSup}(x_6) = \text{FT.localSup}(a_3) + \text{FT.localSup}(a_4) = 5 + 1 = 6.
\]

Global support can be seen as the original support (Definition 2), once the transactions of the database are stored in the fact table, and therefore, the number of transactions containing an itemset in the database is the number of transactions of the fact table containing all the \( tid \)s that contain that itemset.

### 4.3.1 The Algorithm
Star FP-Growth mines multiple relations for frequent patterns in a database following a star schema. The result is the same as mining the denormalized table, but this algorithm does not materialize the denormalization of the table.
It is based on FP-Growth [HPY00], and the main idea is to construct a Super FP-Tree, combining the FP-Trees of each dimension, so the original FP-Growth can run to find multi-relational patterns. Like FP-Growth, it scans each table only twice: first to count the support of each item and second to construct the FP-Tree.

The overall steps are:

**Step 1: Support Counting:** The fact table is scanned to count the support of each tid of each dimension.

This step is the first scan to the fact table. The support of each tid in the fact table (i.e. FT.localSup(tid)) is stored, so that it can be used later. In example 1, the tid support is stored in the third column of each dimension table (Table 4.1, right). As one can see in the fact table 4.2(a), a1 occurs three times, therefore, the itemset corresponding to this tid has a support equals to three (FT.localSup(a1) = 3).

**Step 2: Local Mining:** An FP-Tree is constructed for each dimension table (DimFP-Tree), with a slight modification of the original FP-Tree, taking into account the support calculated in the previous step.

**Step 3: Global Mining:**

Step 3.1: Construct the Super FP-Tree: The FP-Trees of each dimension are combined to form a Super FP-Tree, according to each fact and an established order among dimensions.

Step 3.2: Mining the Super FP-Tree: Run FP-Growth [HPY00], without a change, with the Super FP-Tree and the minimum support threshold. The result of this step is a list of all patterns, not just those relating to one dimension, but also those which relate the various dimensions.

**Constructing the DimFP-Tree**

A DimFP-Tree is very similar to an FP-Tree.

The construction is performed like shown in Algorithm 4.

There are two differences between the construction of a DimFP-Tree and an FP-Tree:

First, the support used here is the global support of each item, i.e. we consider the occurrences of an item in all database, and not only in the item’s table. Therefore, a node does not start with the support equals to one, but with support = support(T), with T the tid of the transaction that originated the node. It also is not incremented by only one, but by support(T) (lines 6 and 8 of function insert).

For example, b4 has a support = 2, which means that that transaction occurs two times in the database. Adding two times the same transaction with support = 1 is the same as adding it one time with support = 2. Starting a node with support = support(T) and incrementing by support(T) avoids being repeatedly inserting the same transaction.

Second, instead of the header table, the DimFP-Tree has other structure that keeps track of the path correspondent to each tid. It stores the last node of that path for each tid (line 3 of function insert). This structure will help the global mining. If we want to know which frequent items belong to a tid, we follow the link in that table to find the last node, and then we just have to climb through its parents till we reach the root node. The items of the nodes in the path we took are the frequent items of that transaction.

Lets consider the construction of the DimFP-Tree of table B in example 1.
1: DimFP-TreeConstruction(DimensionTable DT, σ)
   //First dimension table scan
2: Calculate global support for each item in DT
3: F ← List of frequent items, in a support-descending order
4: R ← the root of a new DimFP-tree (label = null and support = 0)
   //Second dimension table scan
5: for all transaction T ∈ DT do
6:   T ← select frequent items in T and order according to F
7:   insert_tree(T, R)
8: return R

Algorithm 4: DimFP-tree construction

1. First scan to the table B will calculate the global support of each item:

   \[
   \begin{array}{cccccccc}
   y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 & \ldots & y_{11} \\
   2 & 1 & 7 & 1 & 1 & 6 & 1 & 1 & 1
   \end{array}
   \]

   Only \(y_3\) and \(y_6\) are frequent, i.e. have a global support no less than 4 transactions (the minimum support threshold). Therefore, only the itemsets containing just \(y_3\) and/or \(y_6\) would be frequent, according to the anti-monotone property defined in Apriori [AS94].

   From this scan results F, a list of the frequent items, ordered in frequent-descending order,
   \[F ← \{(y_3 : 7), (y_6 : 6)\}\] (the number after “:” indicates the support).

2. An empty tree is created, with the root node.

3. For each transaction \(b_1, b_2, \ldots, b_7\), select the frequent items and sort them according to the order of F (see figure 4.2(a)). Insert the transaction in the root of the tree.

   (a) First transaction, \(b_1\), does not have any frequent item, therefore nothing changes in the tree.
       In the branch table, \(b_1\) is linked to the root node.
       Tree: \(\langle \rangle\)

   (b) Second transaction, \(b_2\), has one frequent item, \(y_3\). It leads to the construction of the first path of the tree. A node with label = \(y_3\) and support = support\((b_2)\) = 1 is inserted as a child of the root. \(b_2\) is linked to this node.
       Tree: \(\langle (y_3 : 1) \rangle\)

   (c) Third transaction, \(b_3\), has two frequent items and support = 1, \((y_3, y_6) : 1\). It shares the prefix \(\langle y_3 \rangle\) with the existing path. The support of node in prefix is incremented by the support of \(b_2, 1\). One new node with label = \(y_6\) and support = support\((b_2)\) = 1, \((y_6 : 1)\), is created and added as a child of \((y_3 : 2)\). \(b_3\) is linked to the new node \((y_6 : 1)\).
       Tree: \(\langle (y_3 : 2), (y_6 : 1) \rangle\)
(d) Fourth transaction, \((y_3y_6) : 2\) shares the common prefix of the existing path. The support of each node along the prefix is incremented by the support of \(b_4, 2\). \(b_4\) also links to the node with \(y_6\).

Tree: \((\langle y_3 : 4 \rangle, \langle y_6 : 3 \rangle)\)

Fifth and sixth transaction follow the same reasoning, resulting the tree: \((\langle y_3 : 7 \rangle, \langle y_6 : 5 \rangle)\)

(e) The last transaction leads to the construction of another path, \((\langle y_6 : 1 \rangle)\), once it does not share the prefix of the existing path. \(b_7\) is, then, linked to the node \((y_6 : 1)\).

<table>
<thead>
<tr>
<th>tid (B)</th>
<th>Itemsets</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>(b_1)</td>
<td>(y_3)</td>
<td>1</td>
</tr>
<tr>
<td>(b_2)</td>
<td>(y_3y_6)</td>
<td>1</td>
</tr>
<tr>
<td>(b_3)</td>
<td>(y_3y_6)</td>
<td>2</td>
</tr>
<tr>
<td>(b_4)</td>
<td>(y_3)</td>
<td>1</td>
</tr>
<tr>
<td>(b_5)</td>
<td>(y_3y_6)</td>
<td>2</td>
</tr>
<tr>
<td>(b_6)</td>
<td>(y_6)</td>
<td>1</td>
</tr>
<tr>
<td>(b_7)</td>
<td>(y_6)</td>
<td></td>
</tr>
</tbody>
</table>

(a) Frequent items (sorted)

Figure 4.2: Construction of the DimFP-Tree of dimension \(B\)

The resulting tree is shown in figure 4.2(b), as well as the trees of the other dimensions are presented in figures 4.3(a) and 4.3(b) (the construction of the other DimFP-Trees is trivial, thus not described in this work).

<table>
<thead>
<tr>
<th>tid</th>
<th>Branch Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a_1)</td>
<td></td>
</tr>
<tr>
<td>(a_2)</td>
<td></td>
</tr>
<tr>
<td>(a_3)</td>
<td></td>
</tr>
<tr>
<td>(a_4)</td>
<td></td>
</tr>
</tbody>
</table>

(a) DimFP-Tree \(A\)

<table>
<thead>
<tr>
<th>tid</th>
<th>Branch Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>(c_1)</td>
<td></td>
</tr>
<tr>
<td>(c_2)</td>
<td></td>
</tr>
<tr>
<td>(c_3)</td>
<td></td>
</tr>
<tr>
<td>(c_4)</td>
<td></td>
</tr>
<tr>
<td>(c_5)</td>
<td></td>
</tr>
<tr>
<td>(c_6)</td>
<td></td>
</tr>
<tr>
<td>(c_7)</td>
<td></td>
</tr>
<tr>
<td>(c_8)</td>
<td></td>
</tr>
<tr>
<td>(c_9)</td>
<td></td>
</tr>
<tr>
<td>(c_{10})</td>
<td></td>
</tr>
</tbody>
</table>

(b) DimFP-Tree \(C\)

Figure 4.3: DimFP-Trees of dimensions \(A\) and \(C\) respectively

Concerning the branch table, this structure will give the same result as if we have the table with the
transactions and their frequent sorted items (Figure 4.2(a)). However, the size of the tree is usually much smaller than its original database, therefore, not having that table materialized in memory usually saves a lot of space and avoids duplicates [HPYM04]. In our example, transactions $b_3$, $b_4$, and $b_6$ have the same frequent items. If we keep the table instead of the tree, those frequent items will be repeated three times, one for each $tid$. With the tree, there is just one path corresponding to the three transactions. Further, shared parts can also be merged using the tree. Therefore, in a larger scale, the more transactions there are, the greater the difference.

Constructing the Super FP-Tree

The Super FP-Tree is just like an FP-Tree, since it will serve as input to FP-Growth. The construction is very similar to the construction of an FP-Tree (Algorithm 5).

1. SuperFP-TreeConstruction(FactTable $FT$, DimFP-Trees $DTs$)
2. Get the global support of each frequent item in all $DTs$
3. $F \leftarrow$ List of those frequent items
4. $R \leftarrow$ the root of a new FP-tree ($label = null$ and $support = 0$)
   //fact table scan
5. for all transaction $T \in FT$ do
6.   $T \leftarrow$ denormalise($T$)
7.   $T \leftarrow$ order($T$)
8.   insert_tree($T, R$)
9. return $R$

Algorithm 5: Super FP-tree construction

Despite this, there are three differences:

1. It is not necessary the first scan to any table to calculate the supports. They are already calculated and stored in each dimension tree. Then, this step consists only in going to each DimFP-Tree to get their frequent items and respective supports (line 2 of SuperFP-TreeConstruction).

2. The denormalisation of each fact is necessary before ordering the items or inserting them in the tree (line 7 of SuperFP-TreeConstruction). A fact is a set of $tids$ and each $tid$ corresponds to a transaction in its dimension table. Through the branch table of each DimFP-Tree we can get the path corresponding to the transactions of each $tid$. Furthermore, according to the anti-monotone property, if an itemset is not frequent, no other itemset containing it will be. This means that only frequent items are necessary and important. Thus, we only check the frequent items in the transactions of each $tid$, ensuring that the final tree has only the frequent items of each dimension. Therefore, if we check the transactions of each $tid$ in a fact, we would have a denormalised fact (algorithm 6).

On table 4.2(b), one can see the result of denormalising each fact of our example.
1: \textbf{denormalise}(Fact }T\text{)
2: \textbf{ }F \leftarrow \text{ new empty itemset}
3: \textbf{for all }tid \in T \textbf{ do}
4: \textbf{ Get frequent itemset corresponding to }tid \textbf{ in the respective DimFP-Tree}
5: \textbf{ Add it to }F
6: \textbf{return }F

Algorithm 6: Denormalize a fact

3. The ordering of items in a transaction does not have to be a frequency descending order (line 8 of SuperFP-TreeConstruction). We can explore the existing relations and characteristics of dimensions to try several different orders.

As verified in the improvement FP-Growth proposed in [HPYM04], an FP-tree constructed based on frequency descending ordering may not always be minimal.

The support descending ordering enhances the compactness of the FP-tree structure. However, this does not mean that the tree so constructed always achieves the maximal compactness. With the knowledge of particular data characteristics, it is sometimes possible to achieve even better compression.

Consider the following example. Let the set of transactions be: adef , bdef , cdef , a, a, a, b, b, b, c, c, c, and the minimum support threshold be 3. The frequent itemset associated with support count becomes a:4, b:4, c:4, d:3, e:3, f:3. Following the item frequency ordering \textit{a }→ \textit{b }→ \textit{c }→ \textit{d }→ \textit{e }→ \textit{f }, the FP-tree constructed will contain 12 nodes, as shown in figure 4.4(a). However, following another item ordering \textit{f }→ \textit{d }→ \textit{e }→ \textit{a }→ \textit{b }→ \textit{c }, it will contain only 9 nodes, as shown in figure 4.4(b).

![Figure 4.4: FP-tree constructed based on frequency descending ordering may not always be minimal](image)

There are two related and important properties of FP-tree that can be derived from the FP-tree construction process [HPYM04].

On one hand, given a transaction database \(DB\), and without considering the root, the size of an FP-tree is bounded by \(\sum_{T \in DB} |\text{freq}(T)|\).

The height of the tree is bounded by \(\max_{T \in DB}(|\text{freq}(T)|)\), where \(\text{freq}(T)\) gives the frequent items of transaction \(T\).

This means that, the number of nodes of an FP-Tree (size) is, at most, the number of frequent items in all the transactions of the database, and the number of levels (height) is, at most, the maximal number of frequent items in a transaction.

On the other hand, given a transaction database \(DB\), the number of paths in an FP-tree is bounded by \(|DB|\), i.e., it is, at most, the number of transactions in the database, if each transaction contributes to one different path of the FP-tree, with the length equal to the number of frequent items.
Ordered facts

\[ y_3 x_6 x_7 x_5 \]
\[ y_3 x_6 x_7 y_6 z_4 x_5 \]
\[ y_3 x_6 x_7 y_6 z_4 x_5 \]
\[ y_3 x_6 x_7 y_6 z_4 x_5 \]
\[ y_3 x_6 x_7 y_6 z_4 x_5 \]
\[ y_3 x_6 x_7 y_6 z_4 x_5 \]
\[ y_3 x_6 x_7 y_6 z_4 x_5 \]
\[ y_3 x_6 x_7 y_6 z_4 x_5 \]
\[ y_3 x_6 x_7 y_6 z_4 x_5 \]

Figure 4.5: Facts and Super FP-Tree resulting from a support descending order of items

With those lemmas in mind, several orders among dimensions were studied and compared in terms of the properties defined above. The three most relevant are the following:

(a) **Support descending order of items**

This ordering does not have into account any order of dimensions. After denormalising the fact, the transaction may have items from multiple dimensions. Sorting them in a support descending order may result in an itemset with items from multiple dimensions intermixed. This is the order used in the original FP-Growth. So, the tree resulting from applying ordering is the same as the tree resulting from joining the tables in one and applying directly FP-Growth to it (but in this case, the joining is not materialized).

In example 1, the support descending order of items is:

<table>
<thead>
<tr>
<th>item</th>
<th>( y_3 )</th>
<th>( x_6 )</th>
<th>( x_7 )</th>
<th>( y_6 )</th>
<th>( z_4 )</th>
<th>( x_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>support</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>

Figure 4.5 shows the ordered facts of example 1 and the resulting Super FP-Tree. This tree has 14 nodes (without root) and 4 paths.

\( size = 14 \), \( height = 6 \) and \( |paths| = 4 \).

(b) **Support descending order of dimensions**

As the support descending ordering enhances the compactness of the FP-tree structure, it is a promising order for the dimensions. Dimensions with a higher support are more likely to be shared and thus they are arranged closer to the top of the FP-tree.

Since each dimension can have multiple items with different supports, we consider that the dimension support corresponds to the support of its least frequent item. Dimensions with the same lowest support are ordered alphabetically, and the items of one dimension are ordered in a support descending order.

In our example 1, the lowest support in dimensions is five in dimension A, and six in B and C. The support descending order of these dimensions is \( B \rightarrow C \rightarrow A \) (\( (y_3 s) \rightarrow (z_4 s) \rightarrow (x_5 s) \)).
Ordered facts

\[
\begin{align*}
\text{Ordered facts} & \\
y_1 x_2 x_6 x_7 x_5 & \\
y_1 y_2 z_x y_6 x_6 x_7 x_5 & \\
y_1 y_2 z_y y_6 z_x y_6 x_7 x_5 & \\
y_1 y_2 z_y y_6 z_y y_6 x_6 x_7 x_5 & \\
y_3 y_6 & \\
y_6 z_y y_6 x_6 x_7 x_5 & \\
y_3 y_6 & \\
y_3 y_6 & \\
y_3 y_6 & \\
y_3 y_6 & \\
y_3 y_6 & \\
y_3 y_6 & \\
y_3 y_6 & \\
\end{align*}
\]

Figure 4.6: Facts and Super FP-Tree resulting from a support descending order of dimensions

Figure 4.6 shows the facts ordered according to this order and the resulting Super FP-Tree. This tree is different, but has also 14 nodes (without the root). The number of paths is less: 3 paths, instead of 4.

\[
\text{size} = 14, \text{height} = 6 \text{ and } |\text{paths}| = 3.
\]

Note that, with this ordering, items from multiple dimensions are not intermixed. Items from dimensions with higher support will always appear before those from dimensions with lower support (all items from dimension B \(y_i s\) are before items from C \(z_i s\), and these, in turn, are before A’s items \(x_i s\)).

(c) Path ascending order of dimensions

If we look at the number of paths of a tree, \(|\text{paths}|\) (or just \(P\)), we can state that, when joining two or more trees of different \(|\text{paths}|\) (i.e. adding one tree to every leaf of the other), the order in which the trees are joint influences the size of the resulting tree. Note that the \(|\text{paths}|\) of the joined tree is always the same and equals to \(\prod_{t \in TS} |P(t)|\), where \(TS\) is the set of the trees we want to join. The number of paths of a tree is the same as the number of leafs.

Its size (without the root) is given by

\[
\sum_{i=1}^{[TS]} \left( \prod_{j=1}^{i-1} P(t_j) \right) \times size(t_i) \tag{4.2}
\]

where \(P(t)\) gives the number of paths in the tree \(t\) and \(TS\) is the set of trees in the order they are joint.

The explanation is the following. A tree is inserted in each leaf of the tree immediately above, which, in turn, was also inserted in each leaf of the preceding tree.

For example, imagine we have a tree A with 1 path, and another, B, with 3 paths (Figure 4.7(a) and 4.7(b)). If we join A with B, a copy of B is inserted in each leaf of tree A. Therefore, the resulting tree will have 4 nodes (without the root), as shown in figure 4.7(c). Joining B with A will result in a tree with more nodes, 6 (figure 4.7(d)). This happens because, the more leafs the tree above had, more copies of the tree below are needed.

We can calculate with the expression above the size of the tree in figure 4.7(c):
Ordered facts
\[ x_6 \times x_7 \times x_5 \times y_3 \]
\[ x_6 \times x_7 \times x_5 \times z_4 \times y_3 \times y_6 \]
\[ x_6 \times x_7 \times z_4 \times y_3 \times y_6 \]
\[ y_3 \]
\[ x_6 \times x_7 \times z_4 \times y_6 \]
\[ z_4 \times y_3 \times y_6 \]
\[ x_6 \times x_7 \times z_4 \times y_3 \times y_6 \]

Figure 4.8: Facts and Super FP-Tree resulting from a path ascending order of dimensions

\[
\prod_{j=1}^{0} P(t_j) \times \text{size}(t_1) + \prod_{j=1}^{1} P(t_j) \times \text{size}(t_2)
\]
\[
= 1 \times \text{size}(t_1) + P(t_1) \times \text{size}(t_2)
\]
\[
= 1 \times \text{size}(A) + P(A) \times \text{size}(B)
\]
\[
= 1 + 1 \times 3 = 4
\]

As we stated, joining trees in a path ascending order, will result in the smallest tree.

However, joining two DimFP-Trees is not that linear. We may have to insert one tree not just in the leafs of the other, but also in a middle node, if that node corresponds to the last node of one transaction. For example, DimFP-Tree A (figure 4.3(a)) has just one path, but node \((x_7 : 6)\) is the last node of, at least, one transaction. Therefore, adding another DimFP-Tree to A may lead to adding more nodes at the end of node \((x_7 : 6)\), which causes the tree to have another path. Expressions above are also valid for this case, if we consider that \(P(t)\) gives the number of nodes that correspond to the last node of a transaction.

Then, this ordering consists in joining DimFP-Trees in a \(|\text{path}|\) ascending order. Dimensions with the same lowest support are ordered alphabetically, and the items of one dimension are ordered in a support descending order, like the other orderings.

The number of paths of dimensions in our example 1 is one for A and C, and two for B. Their path ascending order is \(A \rightarrow C \rightarrow B\), which means that \(x_i\)s will appear before \(z_i\)s and these before \(y_i\)s. The Super FP-Tree is shown in figure 4.8.

In fact, this ordering, in this example, does not give better results than the support descending
order. It gives: size = 15, height = 6 and |paths| = 6.

This orders have been applied to several datasets and the results are presented in section 4.4.

Note that the set of frequent items is independent of the order applied. The result is the same for every orderings.

4.4 Experimental Results

The Super FP-Tree is the main structure of this algorithm. This is the tree that will be mined with FP-Growth, and this is the tree that holds the patterns we want to find. The main purpose of the algorithm is the construction of this tree. Therefore, the Super FP-Tree is the central object of these experiments.

Our goal is to analyse the impact of different orderings on the performance of our algorithm. In order to do that, the three orderings for the construction of the Super FP-Tree are compared, varying the minimum support threshold and the time spent in each step is analysed.

The dataset is a real movies database [Wie89] following the star schema in figure 4.1, the same used to construct our example 1.

The real database has six dimensions, with different numbers of records, from about 20 to 11000, and with a fact table with about 11000 transactions (see table 4.4).

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Number of transactions</th>
<th>Number of attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Award</td>
<td>36</td>
<td>5</td>
</tr>
<tr>
<td>Date</td>
<td>120</td>
<td>3</td>
</tr>
<tr>
<td>Director</td>
<td>2961</td>
<td>7</td>
</tr>
<tr>
<td>Epoch</td>
<td>26</td>
<td>4</td>
</tr>
<tr>
<td>Movie</td>
<td>11460</td>
<td>8</td>
</tr>
<tr>
<td>Studio</td>
<td>950</td>
<td>6</td>
</tr>
<tr>
<td>Fact Table</td>
<td>11460</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 4.3: Characteristics of the movies database

Among the data, we encounter a description of the directors, producers and awards received for each film and time information about them.

To achieve reliable results, the data was split into five equal datasets. The tests were applied to each dataset and we considered the average of each local result. Therefore, we analyse about 2000 facts at each time.

The next subsections present the results achieved in these experiments. The 3 orderings for the construction of the Super FP-Tree are compared, varying the minimum support threshold. Finally, the time spent in each step is analysed.

4.4.1 Comparing Super FP-Trees built with different orderings

When comparing the size of the Super FP-Tree (figure 4.9), i.e. the number of nodes, the tree that is more compact is the one resulting from applying some order to the dimensions. In terms of compression, the support descending and path ascending orders of dimensions are very similar, and better than the support descending order of items. In this experiments, this ordering gave always the tree with more nodes. Note that the support descending order of items gives the same results as constructing an FP-Tree from the flat table (resulting from the denormalisation of the star), therefore, it serves as a reference to the other orderings. This happen because assigning an ordering for the dimensions, taking into account their
characteristics and the properties described above, increase the number of shared nodes, and therefore, the compactness of the Super FP-Tree.

![Figure 4.9: Average size of the Super FP-Tree](image)

In terms of the number of paths in the Super FP-Tree, the resulting trees are very similar (figure 4.10). Although the support descending order of items gives the less compact tree, it gives a tree with slightly less paths than the other orders.

![Figure 4.10: Average Number of Paths of the Super FP-Tree](image)

### 4.4.2 Time spent in each step

The average time spent in the mining process was also studied. On average, 98% of the time is spent in the construction of the DimFP-Trees (step 2), and counting the global support (step 1) only takes 0.20% of the time. The difference between the application of the three orderings is mostly seen in step 3. As can be seen in figure 4.11, the support descending order of items takes less time constructing the Super FP-Tree than the other orders. With the support descending and the path ascending orders of dimensions, each transaction of the fact table has to be ordered according to that ordering before it can be inserted in the tree, yielding the previous results.

![Figure 4.11: Average time on Super FP-Tree construction](image)

Even though the time for the construction of the Super FP-Tree is smaller for the support descending
order of items, the FP-Growth will take longer to execute (figure 4.12), due to resulting tree’s size (like
seen in figure 4.9, its size is bigger).

![Graph](image)

Figure 4.12: Average Time mining with FP-Growth

Therefore, in the end, the total time needed for running Star FP-Growth is very similar for all orderings
(figure 4.13).

![Graph](image)

Figure 4.13: Average time of mining with Star FP-Growth

The computer used to run the experiments was an Intel Xeon E5310 1.60GHz (Quad Core), with 2GB
of RAM. The operating system used was GNU/Linux amd64 and the algorithm was implemented using
the Java Programming language (Java Virtual Machine version 1.6.0_02). The tables were maintained
in memory, as well as all the trees. However, after Local Mining there is no need to keep the dimension
tables, therefore they were freed before Global Mining.

4.5 Conclusions

Star FP-Growth is a simple algorithm for mining patterns in a star schema. It does not perform the
denormalisation of the tables, making use of the star properties. Its main purpose is to prepare the
FP-Tree that represents the data, so that it can serve as input to FP-Growth.

Three orderings for dimensions were analysed and the results state that applying a support descending
or a path ascending order for the dimensions achieve better compression than the usual support descending
order of items. The time used in the mining process is very similar for those orders.

The proposed algorithm can be generalized to be applied to a snowflake structure, where there is a
star structure with a fact table $FT$, but a dimension table can be replaced by another fact table $FT'$,
which is connected to a set of other dimension tables. We can consider mining across dimension tables
related by $FT'$ first. Then consider the resulting Super FP-Tree as a derived DimFP-Tree and continue
processing the star structure with $FT$. This means that mining a snowflake starts from their “leaves”.

52
Using a pattern growth method and the FP-Tree gives us an important benefit: the size of an FP-tree is bounded by the size of its corresponding database because each transaction will contribute at most one path to the FP-tree, with the length equal to the number of frequent items in that transaction. Since there are often a lot of sharing of frequent items among transactions, the size of the tree is usually much smaller than its original database. Unlike the Apriori-like method which may generate an exponential number of candidates in the worst case, under no circumstances, may an FP-tree with an exponential number of nodes be generated.

If the tree cannot be maintained in main memory, several techniques can be used, whether representing and storing the tree in hard disk, or partitioning the database into a set of projected databases, and then for each projected database, constructing and mining its corresponding FP-tree [HPYM04].
Chapter 5

Case Study

As said before, the goal of this work is to mine bibliographic data, in particular, data from PORBASE, the catalogue of Portuguese libraries, which is in UNIMARC.

The goal of this case study is to prove that we can mine bibliographic records with Star FP-Growth. This data presents a lot of challenges, as presented in section 3.3, and therefore, a star schema was designed and created to represent this data in order to be able to mine it.

The already made statistics on this data (section 2.2) helped getting an insight the domain and on cataloging behaviour.

The star showed and described in section 3.4, figure 3.3, is the star used in this case study.

5.1 Dataset Description

The data used in this tests reflect the bibliographic records of PORBASE on June, 2008. The version of UNIMARC is the fourth revision, second edition [IFL02], as in section 2.2 and in the BNP’s work.

For a first analysis, only monographs were considered, i.e. the records with type of record 'a' (character position 6 of leader’s value) and bibliographic level 'm' (character position 7).

Therefore, columns in leader’s dimension corresponding to the type of record and bibliographic level of a record were discarded (they are the equal for every record in analysis). The leader is the aggregator dimension which allows us to get all pairs (attribute, value) corresponding to each bibliographic record.

The datasets used in the tests consist of a set of 100 to 10000 monographs.

The support was varied from 0.9 to 0.5.

5.2 Experimental Results

First, the generated trees are analysed and evaluated, both Dim FP-Trees and Super FP-Trees. The three proposed orders for dimensions are also tested and analysed.

Then, the performance of the algorithm is studied, in terms of time and memory spent.

Finally, the patterns found are presented and evaluated in terms of their number and length, for several sizes of the datasets and supports.

5.2.1 Tree’s Evaluation

To a better understanding of the trees according to the characteristics of data, both DimFP-Trees and the SuperFP-Tree are analysed. Their size, maximum depth and number of paths are compared.

To start, the size of each dimension is presented in figure 5.1. As one can see, as the number of records
increases, the number of entries in each table also increases. The leader table has as many entries as
the number of records, once it is unique for each bibliographic record. Attribute’s dimension table is the
smaller and the one whose size increases very little. The number of entries in this table is limited by the
number of possible fields and subfields (1190 subfields, more those from block 9). Entries of value’s table
are always increasing, once values are not categoric and each new record introduces some new values.
The fact table grows a lot: the number of entries is almost 30 times more than the number of records.
This is explained by the fact that each record has an average of 30 fields or subfields, which means that
each new record introduces in the fact table an average of 30 associations attribute - value (30 entries).

Next, we will analyse how this variation reflects in the DimFP-Trees.

**DimFP-Trees**

**Attribute** Attribute’s tree has only two columns, therefore it will have, at most, depth equals to two.

The number of paths in that tree will also be, at most, the number of possible subfields. Its size is
limited by the combination of fields and subfields.

As we can see in figure 5.2, the number of nodes in the Attribute’s DimFP-Tree increases as the
minimum support decreases, but just by one or two nodes.

This may indicate and reflect the fact that most of the fields or subfields are infrequent. As we saw
in data usage studies (Section 2.2), about 80% of the fields in monographs are used in less than 1%
of the records, and only 6% in more than 50% of the records. This means that setting the minimum
support to 1% will not catch 80% of the fields.

In figure 5.2, we see that, for 10000 records, and for a minimum support of 50%, the DimFP-Tree
has only 38 nodes. While the respective table has 170 entries (figure 5.1), we only have to keep 38
nodes.
**Value** Value’s dimension only has one column, which means that the resulting DimFP-Tree will always be a flat tree (depth equals to one), and its size will correspond to the number of frequent itemsets with size 1. The same happens with Leader’s dimension.

The results show that the size of this DimFP-Tree almost does not vary. It corresponds only to 7 or 8 frequent values.

Note that, for example, for 10000 records, if we had to keep the dimension table in memory, we would have to keep more than 82000 entries. With the DimFP-Tree, we only keep what is frequent. Further, the tree is a compact structure for data that allows us to join the common nodes. In that example, we only have to keep 7 or 8 nodes, which is much less than the size of the table.

**Super FP-Tree**

The Super FP-Tree is the result of the combination of the DimFP-Trees. This is the final tree, that will contain the final patterns, i.e. the frequent co-occurrences of attributes and values within the bibliographic records. This means that each path in the Super FP-Tree can have multiple attributes and/or values. Once we have an aggregating dimension, the leader, we can keep track of all associations attribute-value of one record.

Figure 5.3 shows the size of the Super FP-Tree when the size of the database and the support vary, for a support descending order of dimensions.

![Figure 5.3: Size of the Super FP-Tree](image)

As one can see, the size of the Super FP-Tree increases as the minimum support decreases, once we require fewer occurrences of the same items or itemsets. We can also state that this tree is big, even for a very high support of 90% (about 150 nodes for 100 records and 750 for 10000). One possible explanation is the fact that some attributes are mandatory, and therefore, have a support of 100%. Another is, as stated in the data usage studies, that the fields are, or too frequent, or infrequent.

Note that the space saved by this Super FP-Tree is huge. For 50% of support, and 100 records, the fact table has more than 2900 entries, and the respective Super FP-Tree has 300 nodes. And for 10000 records, the fact table has almost 290000 entries, while the tree has less than 2000 nodes.

**5.2.2 Performances**

As the size of the datasets increase, the size of the trees also increase and, therefore, the time spent in the algorithm (figure 5.4).

This increment is specially due to local mining, which corresponds to 60% of total time (with a deviation of 15%).

Figure 5.5 shows the time spent with Local Mining. It encompasses the construction of the Dim-FP-Tree and therefore it depends on the size of the dimension tables (two scans to each table).
5.3 Patterns Evaluation

Patterns can be analysed quantitatively and qualitatively. From a quantitative point of view, we will describe the number of patterns found according to the size of the dataset and the minimum support. From a qualitative aspect, we will describe the type and meaning of them.

5.3.1 Quantitative Analysis

Applying Star FP-Growth to the bibliographic data in PORBASE resulted in a lot of patterns, as can be seen in figure 5.6. For example, for 10,000 records and 50% of support, more than 18,000,000 patterns are found. This large number of patterns makes their analysis very difficult.

Even with a high support of 90%, the number of patterns is very high (Figure 5.7). For example, for 100 records, we found more than 300 patterns. This happens because the co-occurrence matrix of items in this bibliographic domain is dense, i.e. one attribute co-occurs with almost every other attribute. As we said before, fields or subfields in this data are, or very frequent, or infrequent. Those that are very frequent (mandatory fields, for example) will co-occur with each other and, therefore, generate a lot of patterns.
5.3.2 Qualitative Analysis

We found several kinds of patterns, from intra-dimension to inter-dimension ones. An intra-dimension pattern relates items from the same dimension. In this domain they will show, for example, frequent co-occurrences of fields and subfields with each other (attribute’s dimension), or frequent co-occurrences of values (value’s dimension). Inter-dimension patterns mixture items from existing dimensions. They will relate items from attribute and value’s dimensions, and therefore, show the co-occurrences of fields or subfields and values in the Porbase.

For this analysis, we will consider the patterns found in the dataset with 100 bibliographic records for a minimum support of 90%.

Examples of intra-dimension patterns can be found on table 5.1.

For a better understanding of the patterns, the description of their fields and subfields is shown in table 5.2. Fields are represented as “fieldCode:subfieldCode”, e.g. “101:a” refers to the subfield “a” of field 101, which is responsible for storing the language of the text.

Looking to table 5.0(a), we find patterns of length one. We can verify there that, for example, every record in question were written in portuguese and published in Portugal (the first two patterns have a
### (a) Patterns of length 1

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Value {675:z:por})</td>
<td>100%</td>
</tr>
<tr>
<td>(Value {102:a:pt})</td>
<td>100%</td>
</tr>
<tr>
<td>(Value {801:g:rpc})</td>
<td>99%</td>
</tr>
<tr>
<td>(Value {801:b:bn})</td>
<td>99%</td>
</tr>
<tr>
<td>(Value {801:a:pt})</td>
<td>99%</td>
</tr>
<tr>
<td>(Value {101:a:por})</td>
<td>96%</td>
</tr>
<tr>
<td>(Value {675:v:med})</td>
<td>95%</td>
</tr>
<tr>
<td>(Attribute {Field=801})</td>
<td>99%</td>
</tr>
<tr>
<td>(Attribute {Field=801, Subfield=g})</td>
<td>99%</td>
</tr>
<tr>
<td>(Attribute {Field=801, Subfield=b})</td>
<td>99%</td>
</tr>
<tr>
<td>(Attribute {Field=801, Subfield=a})</td>
<td>99%</td>
</tr>
<tr>
<td>(Attribute {Field=200, Subfield=f})</td>
<td>95%</td>
</tr>
</tbody>
</table>

### (b) Other Patterns

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Value {101:a:por}, {801:g:rpc})</td>
<td>95%</td>
</tr>
<tr>
<td>(Value {101:a:por}, {801:a:pt})</td>
<td>95%</td>
</tr>
<tr>
<td>(Value {675:v:med}, {801:a:pt})</td>
<td>94%</td>
</tr>
<tr>
<td>(Value {675:v:med}, {801:b:bn})</td>
<td>94%</td>
</tr>
<tr>
<td>(Value {675:v:med}, {801:g:rpc})</td>
<td>94%</td>
</tr>
<tr>
<td>(Value {101:a:por}, {675:v:med})</td>
<td>92%</td>
</tr>
<tr>
<td>(Value {102:a:pt}, {101:a:por}, {675:v:med})</td>
<td>92%</td>
</tr>
<tr>
<td>(Attribute {Field=210}, [Subfield=a], [Subfield=c], [Subfield=d])</td>
<td>100%</td>
</tr>
<tr>
<td>(Attribute {Field=215}, [Subfield=a], [Subfield=d])</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 5.1: Intra-Dimension Patterns

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>Recorder Identifier</td>
<td>215</td>
<td>Physical Description</td>
</tr>
<tr>
<td>101:a</td>
<td>Language of the text</td>
<td>215:a</td>
<td>Specific Designation</td>
</tr>
<tr>
<td>102</td>
<td>Country of publication or Production</td>
<td>215:d</td>
<td>Dimensions</td>
</tr>
<tr>
<td>200:f</td>
<td>Title and first statement of responsibility</td>
<td>675:z</td>
<td>Language of Edition</td>
</tr>
<tr>
<td>210</td>
<td>Publication</td>
<td>801</td>
<td>Original Source</td>
</tr>
<tr>
<td>210:a</td>
<td>Place</td>
<td>801:a</td>
<td>Country</td>
</tr>
<tr>
<td>210:c</td>
<td>Name</td>
<td>801:b</td>
<td>Agency</td>
</tr>
<tr>
<td>210:d</td>
<td>Date</td>
<td>801:g</td>
<td>Cataloguing Rules</td>
</tr>
</tbody>
</table>

Table 5.2: Fields Description
support of 100%, which means that they occur in every record). We can also state that the original source (field 801) of 99% of the records belongs the National Library of Portugal (Value 801:b:bn and Value 801:a:pt) and use the portuguese rules for cataloguing (Value 801:g:rpc). Length-1 patterns of Attribute dimension include the occurrence of the recorder identifier (Attribute Field=001) in every record and of the first statement of responsibility (Attribute Field=200, Subfield=f) in 95% of them.

Intra-dimension patterns with more than one item (table 5.0(b)) contain only items from one dimension. An example is “(Value 101:a:por, 675:v:med)”, which relates the portuguese language of the text (with 96% of support) to a normal reference edition of Universal Decimal Classification (with 95% of support) that we know it is portuguese (Value 675:z:por has 100% of support).

As said before, patterns that occur 100% of the time, will generate a lot of patterns, once they co-occur with every other pattern. For example, “(Value 102:a:pt, 101:a:por, 675:v:med)” contains the item “(Value 102:a:pt)”, which has 100% of support. This item does not add anything new to the pattern, that symbolizes the same as the pattern described above.

Most of patterns related to attributes with 100% of support correspond to mandatory fields, like the record identifier (field 001). But there are also optional fields that have the maximum support, like the place, name and date of publication (Attribute Field=210, Subfield=a, Subfield=c, Subfield=d), or the physical description of a record, with the respective designation and dimensions (Attribute Field=215, Subfield=a, Subfield=d).

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Attribute {Field=210, Subfield=a}, Value {801:b:bn})</td>
<td>99%</td>
</tr>
<tr>
<td>(Attribute {Field=200, Subfield=f}, Value {102:a:pt}, {675:z:por})</td>
<td>95%</td>
</tr>
<tr>
<td>(Attribute {Field=200, Subfield=f}, Value {101:a:por})</td>
<td>91%</td>
</tr>
</tbody>
</table>

Table 5.3: Inter-Dimension Patterns

Examples of inter-dimension patterns can be found on table 5.3. These patterns relate items from both dimensions.

Most of them enclose those patterns with 100% of support, like first two in that table. However, the third pattern, “(Attribute Field=200, Subfield=f, Value 101:a:por)” has a support of 91%, but has items with a support of 95% and 96% respectively. It relates the first statement of responsibility with a portuguese language for the text, and it denotes that, although each of them occur alone 95% or 95% of the time, they only occur 91% of the time together.

We can also relate the patterns found in these experiments for monographs with the results of the previous data statistics and data usage studies (Section 2.2). We confirm that mandatory fields are really used in every record by finding the respective patterns with 100% of support (the recorder identifier, field 001, for example, “(Attribute Field=001)”)). We also state that only 10 fields have more than 90% of occurrences (for the dataset of 100 records), that are exactly the fields described in section 2.2 as the most used fields (except field 700, Personal Name - Primary Intellectual Responsibility).

Table 2.7 showed the detailed field usage for monographs. We can see that the results for most used fields are very similar:

<table>
<thead>
<tr>
<th>Fields</th>
<th>001</th>
<th>005</th>
<th>100</th>
<th>101</th>
<th>102</th>
<th>200</th>
<th>210</th>
<th>215</th>
<th>675</th>
<th>801</th>
</tr>
</thead>
<tbody>
<tr>
<td>Study</td>
<td>100%</td>
<td>82%</td>
<td>100%</td>
<td>100%</td>
<td>99%</td>
<td>100%</td>
<td>100%</td>
<td>96%</td>
<td>74%</td>
<td>90%</td>
</tr>
<tr>
<td>Patterns</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>99%</td>
</tr>
</tbody>
</table>

61
5.4 Conclusions

These experiments had the goal of finding patterns in bibliographic data from Porbase, and they did find them.

The domain is very hard to understand, and the characteristics imposed by the bibliographic format in use make the analysis of this data very difficult. Despite all these challenges we prove that it is possible to mine bibliographic data and find patterns in it.

From the application of Star FP-Growth, we can conclude that we save a lot of space by performing local mining first in each dimension separately. Although this step is the most costly, it allows the rapid construction of the tree that represents all data in analysis, the Super FP-Tree. However, this algorithm cannot deal well with the high number of bibliographic records. Further optimizations can be made to the algorithm to allow other exhaustive analysis.

The characteristics of data lead to the discovery of too many patterns, which difficult the analysis of those patterns, and therefore, the ability to discover unknown and useful information. Even so, we found several patterns, from intra to inter dimensional ones, and we were able to compare and corroborate previous studies on this data.
Chapter 6

Conclusions and Future Work

6.1 Conclusions

The bibliographic data in analysis presents several challenges for the mining task, not only because of the high amounts of data, but also because of the way the data is represented. UNIMARC format offers a wide range of fields and subfields in order to allow a richer description, but this makes the analysis more complex, and therefore the application of data mining techniques.

There are a lot of pattern mining techniques, able to find different kinds of patterns, from simple sets of items, to sequences, trees and graphs. These algorithms are, or apriori-based (with candidates generation), pattern-growth based or lattice-based. Pattern-growth techniques have an important advantage over the others, they do not generate candidates, which is what we want in order to deal with large amounts of data.

However, traditional algorithms cannot be applied directly to UNIMARC records, since the format is not tabular, and the data cannot easily be arranged in a table.

This work proposes a new way of representing the bibliographic data. Modelling this data in a star schema, we split the data through several dimensions, using the fact table to link them. This star schema solves some challenges imposed by the characteristics of bibliographic data, such as non categorical values and lots of sparse data. Furthermore, the star is extensible. We can add other dimensions to represent the aspects we want to model and/or analyse.

There are some algorithms to mine a star. Some of them are apriori-based and suffer from candidates explosion, others mine each dimension separately and join two tables at a time. This work purposes a new algorithm, the Star FP-Growth, for mining multi-relational patterns in a star schema, without materializing the join of the dimensions. It is based on FP-Growth and its main idea is to construct the FP-Tree corresponding to each dimension separately. And then combine them into a Super FP-Tree and mine it with FP-Growth. This way, we only consider frequent itemsets of each Dim FP-Tree for the construction of the Super.

The Star FP-Growth was applied to the star schema created for the bibliographic records from Porbase, and it proved that is possible to mine this data and find patterns in it. Although the algorithm cannot deal well with a high number of records, it is a first step for further studies and analysis.
6.2 Future Work

To deal with the high number of records and resulting patterns, the algorithm can be adapted to a database management system (Database Management System (DBmS)) to read the data directly from the database and to store the structures needed and results found. The DBmS provide facilities for controlling data access and optimized routines that allow some calculations to be made on the DBmS side (like summing, counting, sorting, grouping, cross-referencing, etc.).

Further improvements may include the use of parallelization techniques, such as Google’s MapReduce [DG04], in order to take advantage of multiple processors and lower the space requirements per processor. It is possible to parallelize FP-Growth[LWZ08], and there is already an implementation of it using MapReduce in the Apache’s toolkit called Mahout 1.

Since the bibliographic records are continuously increasing, infrequent items can become frequent later on and hence cannot be ignored. Ideas from streaming data mining can help dealing with this data [DH00]. They follow some principles like only look one time at each record, during an infinitesimal period of time, and they establish an acceptable error so that some items can be ignored.

As said in section 3.1, tree and graph mining, specially XML techniques, can give more interesting patterns, once they take into account the structure of the data. A step forward is, therefore, to apply these algorithms to bibliographic data, and compare them with the traditional ones.

An important benefit of the star schema proposed in this work is the ease of adding new dimensions to it. The only thing needed is to add a new table correspondent to the new dimension and link it in the fact table, without changing anything in the other dimensions. This is particularly important if we want to find patterns across libraries. To mine multi-language bibliographic patterns we can add an extra dimension (Bibliographic Format or Library, for example), add to the fact table the format id of each record, and mine the new star. This way, we will be able to find patterns within each format and across formats (patterns that are common to some or all formats).

Data elements and indicators can also be represented (in another dimension or integrated in the Attribute’s dimension) and explored. By doing this, we can find patterns that include them and analyse their use and impact in the bibliographic world.

---

1Mahout - http://lucene.apache.org/mahout/
Bibliography


[HPY00] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In SIGMOD’00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data, pages 1–12, New York, NY, USA, 2000. ACM.

[HPYM04] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. Data Mining and Knowledge Discovery, 8(1):53–87, 2004.


66


[YH02] Xifeng Yan and Jiawei Han. gSpan: Graph-based substructure pattern mining. In IEEE International Conference on Data Mining (ICDM), pages 721–724, 2002.


