

International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems  
© World Scientific Publishing Company

## MINING STARS WITH FP-GROWTH: A CASE STUDY ON BIBLIOGRAPHIC DATA

ANDREIA SILVA

*Department of Computer Science and Engineering  
Instituto Superior Técnico, Technical University of Lisbon  
Av. Rovisco Pais, 1049-001 Lisboa, Portugal  
andrea.silva@ist.utl.pt*

CLÁUDIA ANTUNES

*Department of Computer Science and Engineering  
Instituto Superior Técnico, Technical University of Lisbon  
Av. Rovisco Pais, 1049-001 Lisboa, Portugal  
claudia.antunes@ist.utl.pt*

Traditional data mining approaches look for patterns in a single table, while multi-relational data mining aims for identifying patterns that involve multiple tables. In recent years, the most common mining techniques have been extended to the multi-relational context, but there are few dedicated to deal with data stored following the multi-dimensional model, in particular the *star schema*. These schemas are composed of a central huge fact table linking a set of small dimension tables. Joining all the tables before mining may not be a feasible solution due to the usual massive number of records. This work proposes a method for mining frequent patterns on data following a star schema that does not materialize the join between the tables. As it extends the algorithm FP-Growth, it constructs an FP-Tree for each dimension and then combines them through the records in the fact table to form a super FP-Tree. This tree is then mined with FP-growth to find all frequent patterns. The paper presents a case study on bibliographic data, comparing efficiency and scalability of our algorithm against FP-Growth.

*Keywords:* Pattern mining, Multi-relational data mining, Star schema, FP-Growth

### 1. Introduction

Mining a database, this is, a set of linked tables instead of a single one, poses a challenge to data mining research, recognized as a sub problem of mining complex data.<sup>1</sup> This sub-area, usually known as *multi-relational data mining* (MRDM),<sup>2</sup> has the main goal of looking for information hidden in a set of related tables, either belonging to a relational database or a data warehouse, designed according to the multi-dimensional model.

One possible approach is to join the tables following the links among them, and then do the mining process as usual on a single flat table.<sup>3</sup> Indeed, this is what

has been done since the first times of data mining, and one of the reasons for wasting about 80% of the processing time on preparing the data lies on arranging the data into a proper table and reducing the table to an acceptable size by, for example, feature selection or sampling. In order to overcome, or at least minimize this step, it is necessary to deal with multiple tables at once. Actually, one of the great potential benefits of MRDM is the ability to automate the mining process to a significant extent. Fulfilling this potential requires solving the relevant efficiency problems that arise when attempting to do data mining directly from a database, instead of a single pre-extracted flat file.<sup>4</sup>

In recent years, the most common mining operations have been extended to the multi-relational context and MRDM now encompasses multi relational pattern mining (or association rule discovery), multi relational decision trees, multi relational distance-based methods, among others.<sup>2</sup> MRDM approaches have been successfully applied to a number of problems in a variety of areas (see <sup>5</sup>, <sup>6</sup> and <sup>7</sup> for example). From those approaches, just a few are dedicated to pattern mining on star schemas.<sup>3,8,9</sup>

In the case of a *star schema*, data is modeled as a fact describing an event or occurrence, characterized by a particular combination of dimensions. With each dimension aggregating a set of attributes for a same domain property or constraint. In these data warehouses it is common to have several small tables with a large number of columns for storing dimensions, and a central table (known as the *fact table*) that only has a set of numeric attributes for quantifying the event and a set of foreign keys for each dimension.<sup>10</sup> This fact table usually stores a massive number of records, which makes almost impossible to have all data in primary memory. Indeed, joining dimensional tables with the fact table will result in a table much larger than the individual ones, with the same number of records of the fact table and a number of columns close to the sum of the number of columns in the individual tables. In this manner, the humble gains in terms of space obtained with the normalization of tables in the star schema would be wasted, since each tuple in separate tables would appear repeatedly in the final table as many times as the number of records where they occur. This replication would imply both an increase of storage consumption and a decrease of efficiency. Definitely, it is difficult to consider a database that fits in primary memory, which naturally implies I/O operations that have a considerable negative impact on mining operations.

This work aims to find frequent patterns in a data warehouse following a star schema, without materializing the join of its tables. In order to do that, we extend the *FP-Growth* algorithm,<sup>11</sup> proposing a new one called *Star FP-growth* that adapts the process of reading the database into memory. Like its parent, Star FP-growth scans each table twice: first to count the support of each item in dimensions, and second to read data into a more compact structure. Henceforward, it mines the data to find frequent global patterns as in the previous algorithm, now including items from the different tables.

Beside the description of our algorithm, its performance is evaluated and com-

pared in a case study, centered on a database with more than 1.5 million bibliographic records.<sup>1</sup> This database is of particular interest, since it follows the standards on bibliographic data,<sup>12</sup> having more than seven hundred attributes to describe the bibliographic records, distributed in several XML records. In order to mine it looking for frequent patterns, this original data was transformed into a star schema, and then mined through Star FP-growth.

Experimental results show that our algorithm presents a very interesting performance when compared with FP-growth, on efficiency, scalability and memory consumption. However, it reveals some issues about mining frequent patterns that remain unsolved, like the explosion of discovered patterns and the lack of focus of that discovery.

The rest of this paper is organized as follows. Section 2 presents the related work on multi-relational pattern mining on star schemas. The proposed algorithm is described on section 3. Section 4 presents the case study on bibliographic data. The paper concludes with a critical analysis of experimental results and points out specific guidelines for future work.

## 2. Related Work

Frequent pattern mining aims for enumerating all frequent patterns that conceptually represent relations among discrete entities (or *items*). Depending on the complexity of these relations, different types of patterns arise, with the transactional patterns being the most common. A *transactional pattern* is just a set of items that occur together frequently. A well-known example of a transactional pattern is a market-basket, the set of items that are bought frequently in the same transaction by a customer.

There are many stand-alone algorithms to mine different types of patterns, with FP-growth<sup>11</sup> one of the most efficient. This algorithm follows a pattern-growth philosophy that adopts a divide and conquer approach to decompose both the mining tasks and the databases. Its main idea 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 set of items, usually known as *itemsets*. 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.

After reading the database into the FP-tree, in order to find the patterns it is not necessary to scan the database again, it is only needed to scan the tree already

---

<sup>1</sup>The bibliographic records were gathered from PORBASE: the Portuguese National Database. see <http://www.porbase.org/>

in memory. The algorithm is recursive, and transforms the problem of identifying long patterns in the identification of smaller patterns. It uses a *depth-first search* approach.

The pattern mining problem has been proved to correspond to the problem of enumerating all constrained bipartite cliques, which in the worst case is NP-complete. However, since in practice databases (bipartite cliques) tend to be very sparse, mining algorithms for this task are linear in the number of items and transactions.<sup>13</sup>

The work on multi-relational pattern mining over star schemas has been increasing in the last years. Pattern mining is a tool for looking for frequent co-occurrence of items, allowing for the identification of frequent behaviors or trends. In the case of a star schema, pattern mining should identify the tuples that occur across the different dimension tables in a significant number of records in the fact table. In order to deal with multiple tables, pattern mining have to join somehow the different tables, creating the tuples to be mined. Research in this area has shown that methods that follow the philosophy of *mining before joining* outperforms the methods following the *joining before mining* approach, even when the latter adopts the known fastest single-table mining algorithms.<sup>3</sup>

The first multi-relational methods have been developed by the *Inductive Logic Programming* (ILP) community about ten years ago (see <sup>14</sup> and <sup>15</sup>), but they are usually not scalable with respect to the number of relations and attributes in the database. Therefore they are inefficient for databases with complex schemas. Another drawback of ILP approaches is that they need all data in the form of prolog tables.

Based on traditional pattern mining algorithms, Jensen and Soparkar (2000)<sup>8</sup> presented an apriori-based algorithm, which first generates frequent tuples in each single table using a slightly modified version of *Apriori*,<sup>16</sup> and then looks for frequent tuples whose items belong to distinct tables via a multi-dimensional count array. It does not construct the whole joint table but processes each row as the row is formed, thus storage cost for the joint table is avoided. However, the number of candidates generated explodes as the number of dimensions, attributes, and values increase.

Ng et al. (2002)<sup>3</sup> proposed an efficient algorithm without actually performing the join operation. The algorithm performs local mining on each dimension table, and then "binds" two dimension tables at each iteration, i.e. mines all frequent tuples 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 use a vertical data format, and a prefix tree to compress the fact table and to speed up the support computation.

Xu and Xie (2006) created *MultiClose*,<sup>9</sup> which discovers frequent closed tuples without materializing the join among tables. It first converts all dimension tables to a vertical data format, and then mines each of them locally with a closed algorithm,

finding single-table frequent closed tuples. After local mining, frequent tuples are stored in two-level hash table result trees, and then by traversing those result trees frequent closed multi-table tuples are discovered for each pair of tables.

There are other algorithms for finding multi-relational frequent tuples, however they just consider one common attribute at a time. They would have to run as much times as the number of dimensions, since there is no attribute common to all the tables. Instead, in a star schema, the fact table has one attribute in common with each dimension, and the dimensions have no common attribute between them. The patterns discovered by those algorithms will not reflect the co-occurrences among dimensions. Examples are Connection<sup>17</sup> and its extension ConnectionBlock,<sup>18</sup> both based on FP-Growth and on the intersection of local patterns, and MRFP-Growth,<sup>19</sup> 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.

The proposed algorithm *Star FP-growth* also mines star schemas without computing the entire join nor materializing joint tables in memory, as described next.

### 3. Mining Stars

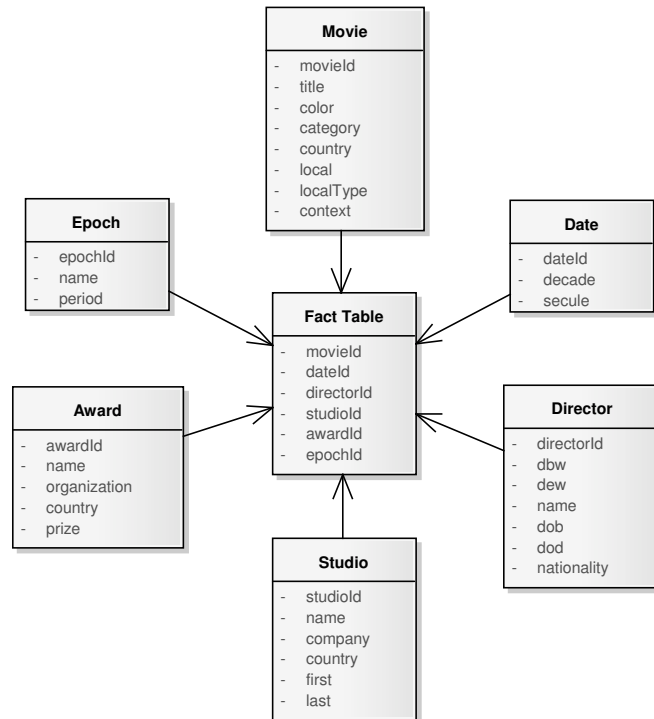


Fig. 1: Movies star schema

A *data warehouse* is a subject-oriented, integrated, time-variant, and non-volatile collection of data in support of management's decision-making process.<sup>20</sup> In terms of

data modeling, a data warehouse consists of at least one multi-dimensional model, which comprises a central fact table and a set of surrounding dimension tables, each one corresponding to a property that characterizes occurrences in the fact table. The most used multi-dimensional model is the *star schema*, which consists of multiple dimension tables that are associated only by foreign keys to a central fact table. Figure 1 illustrates a star schema with six dimensions (*Movie*, *Date*, *Director*, *Studio*, *Award* and *Epoch*) and a fact table linking all dimensions.

Let  $S$  be a tuple  $(D_1, D_2, \dots, D_n, FT)$  representing a data warehouse modeled as a star schema, with  $D_i$  corresponding to each dimension table and  $FT$  to the fact table. Each dimension table  $D_i$  only contains a primary key, denoted by transaction id ( $tid_{D_i}$ ), some other attributes and no foreign keys; on the other hand, the fact table only contains foreign keys: one  $tid$  for each dimension tables. In this manner, each record in the FT would be a tuple in the form  $(tid_{D_1}, tid_{D_2}, \dots, tid_{D_n})$ . In the case there exists some fields other than foreign keys, called *facts* or *measures*, they can be placed into an extra dimension table.

Also, let  $I = \{i_1, i_2, \dots, i_m\}$  be a set of distinct literals, called *items*. In the context of a database, an *item* corresponds to a proposition of the form (*attribute*, *value*). A subset of items is denoted as an *itemset*, which corresponds to a tuple in the database. A transaction  $T = (tid, X)$  is a tuple where  $tid$  is a transaction-id (corresponding to a primary key) and  $X$  is an itemset in  $I$ . Each table in  $S$ , is a set of transactions.

The *support* (or occurrence frequency) of an itemset  $X$ , is the number of transactions containing  $X$  in the database  $S$ .  $X$  is frequent if its support is no less than a predefined minimum support threshold,  $\sigma$ . In a database modeled as a star schema, where there are several tables, we must distinguish between the support considering just a single table versus the support considering all the database:

The *local support* of an itemset  $X$ , with items belonging to a table  $D_i$  ( $D_i.localSup(X)$ ), is the number of occurrences of  $X$  in  $D_i$ .

The *global support* (or just *support*) of an itemset  $X$  is the number of transactions in the fact table containing all the  $tids$  that contain  $X$ , as in equation 1:

$$globalSup(X) = \sum_{tid}^{tid(X)} FT.localSup(tid) \quad (1)$$

Lets consider as an example, a simplification of the star schema in figure 1, with just three dimensions:  $A$  (*Award*),  $B$  (*Studio*) and  $C$  (*Movie*) shown on the left side of Table 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 *Award.Name*, which is different from *Studio.Name* (attribute *Name* in 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 *Award.Name* = *None*. Table 2a shows the fact table.

Award				A		
<i>tid</i>	Name	Organization	Country	<i>tid<sub>A</sub></i>	Itemsets	Support
<i>a</i> <sub>1</sub>	None			<i>a</i> <sub>1</sub>	<i>x</i> <sub>1</sub>	3
<i>a</i> <sub>2</sub>	AAN	Hollywood Academy	USA	<i>a</i> <sub>2</sub>	<i>x</i> <sub>2</sub> <i>x</i> <sub>3</sub> <i>x</i> <sub>4</sub>	1
<i>a</i> <sub>3</sub>	H****	Halliwell's Film Guide	GB	<i>a</i> <sub>3</sub>	<i>x</i> <sub>5</sub> <i>x</i> <sub>6</sub> <i>x</i> <sub>7</sub>	5
<i>a</i> <sub>4</sub>	H	Halliwell's Film Guide	GB	<i>a</i> <sub>4</sub>	<i>x</i> <sub>8</sub> <i>x</i> <sub>6</sub> <i>x</i> <sub>7</sub>	1

Studio				B		
<i>tid</i>	Name	Country	Last	<i>tid<sub>B</sub></i>	Itemsets	Support
<i>b</i> <sub>1</sub>	Unknown			<i>b</i> <sub>1</sub>	<i>y</i> <sub>1</sub>	2
<i>b</i> <sub>2</sub>	U.A.	USA	1983	<i>b</i> <sub>2</sub>	<i>y</i> <sub>2</sub> <i>y</i> <sub>3</sub> <i>y</i> <sub>4</sub>	1
<i>b</i> <sub>3</sub>	Fox	USA	1994	<i>b</i> <sub>3</sub>	<i>y</i> <sub>5</sub> <i>y</i> <sub>3</sub> <i>y</i> <sub>6</sub>	1
<i>b</i> <sub>4</sub>	Paramount	USA	1994	<i>b</i> <sub>4</sub>	<i>y</i> <sub>7</sub> <i>y</i> <sub>3</sub> <i>y</i> <sub>6</sub>	2
<i>b</i> <sub>5</sub>	Shamley	USA		<i>b</i> <sub>5</sub>	<i>y</i> <sub>8</sub> <i>y</i> <sub>3</sub>	1
<i>b</i> <sub>6</sub>	Warners	USA	1994	<i>b</i> <sub>6</sub>	<i>y</i> <sub>9</sub> <i>y</i> <sub>3</sub> <i>y</i> <sub>6</sub>	2
<i>b</i> <sub>7</sub>	Universal	Japan	1994	<i>b</i> <sub>7</sub>	<i>y</i> <sub>10</sub> <i>y</i> <sub>11</sub> <i>y</i> <sub>6</sub>	1

Movie			C		
<i>tid</i>	Title	Color	<i>tid<sub>C</sub></i>	Itemsets	Support
<i>c</i> <sub>1</sub>	Stagecoach	bnw	<i>c</i> <sub>1</sub>	<i>z</i> <sub>1</sub> <i>z</i> <sub>2</sub>	1
<i>c</i> <sub>2</sub>	RearWindow	tcol	<i>c</i> <sub>2</sub>	<i>z</i> <sub>3</sub> <i>z</i> <sub>4</sub>	1
<i>c</i> <sub>3</sub>	The Searchers	tcol	<i>c</i> <sub>3</sub>	<i>z</i> <sub>5</sub> <i>z</i> <sub>4</sub>	1
<i>c</i> <sub>4</sub>	Vertigo	tcol	<i>c</i> <sub>4</sub>	<i>z</i> <sub>6</sub> <i>z</i> <sub>4</sub>	1
<i>c</i> <sub>5</sub>	Rio Bravo		<i>c</i> <sub>5</sub>	<i>z</i> <sub>7</sub>	1
<i>c</i> <sub>6</sub>	Psycho	bnw	<i>c</i> <sub>6</sub>	<i>z</i> <sub>8</sub> <i>z</i> <sub>2</sub>	1
<i>c</i> <sub>7</sub>	The Birds	tcol	<i>c</i> <sub>7</sub>	<i>z</i> <sub>9</sub> <i>z</i> <sub>4</sub>	1
<i>c</i> <sub>8</sub>	El Dourado	col	<i>c</i> <sub>8</sub>	<i>z</i> <sub>10</sub> <i>z</i> <sub>11</sub>	1
<i>c</i> <sub>9</sub>	Star Wars	tcol	<i>c</i> <sub>9</sub>	<i>z</i> <sub>12</sub> <i>z</i> <sub>4</sub>	1
<i>c</i> <sub>10</sub>	Superman	tcol	<i>c</i> <sub>10</sub>	<i>z</i> <sub>13</sub> <i>z</i> <sub>4</sub>	1

Table 1: Dimension Tables A (Award), B (Studio) and C (Movie)

For example on table *Award* some local support would be:

$$Award.localSup(Award.Name = None) = 1$$

$$Award.localSup(Award.Organization = "Halliwell's Film Guide") = 2$$

With respect to the entire database (Table 2a) the global support of the same itemsets would be:

$$globalSup(Award.Name = None) = FT.localSup(a_1) = 3$$

$$globalSup(Award.Organization = "Halliwell's Film Guide") = FT.localSup(a_3) + FT.localSup(a_4) = 5 + 1 = 6$$

This example will be used to show how the proposed algorithm works, with a minimum support equals to 40% of the database, which means that an itemset is frequent if its support is no less than (40% \* 10 transactions) four transactions.

$tid_A$	$tid_B$	$tid_C$	$Itemsets_A$	$Itemsets_B$	$Itemsets_C$
$a_3$	$b_2$	$c_1$	$x_6x_7x_5$	$y_3$	–
$a_3$	$b_4$	$c_2$	$x_6x_7x_5$	$y_3y_6$	$z_4$
$a_3$	$b_6$	$c_3$	$x_6x_7x_5$	$y_3y_6$	$z_4$
$a_3$	$b_4$	$c_4$	$x_6x_7x_5$	$y_3y_6$	$z_4$
$a_1$	$b_1$	$c_5$	–	–	–
$a_1$	$b_5$	$c_6$	–	$y_3$	–
$a_3$	$b_7$	$c_7$	$x_6x_7x_5$	$y_6$	$z_4$
$a_1$	$b_1$	$c_8$	–	–	–
$a_2$	$b_3$	$c_9$	–	$y_3y_6$	$z_4$
$a_4$	$b_6$	$c_{10}$	$x_6x_7$	$y_3y_6$	$z_4$

(a) Fact Table FT

(b) Denormalized Fact

Table 2: Fact table and the frequent itemsets corresponding to each  $tid$ 

### 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 joint table, but without materializing it.

After constructing an FP-tree for each dimension, the corresponding tables are discarded. The trees already take into account the minimum frequency and incorporate transaction ids. The super FP-tree that represents the whole star is then constructed, by aggregating the dimension trees all together, based on the fact table. Finally, this tree is mined using the known pattern growth method, FP-Growth.<sup>11</sup>

It is based on FP-Growth,<sup>11</sup> and the main idea is to construct a Super FP-Tree, combining the FP-Trees of each dimension, so that the original FP-Growth can run and 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. In the example, the  $tid$  support is shown in the third column of each dimension table (Table 1, on the right side).

**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 DimFP-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,<sup>11</sup> 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.

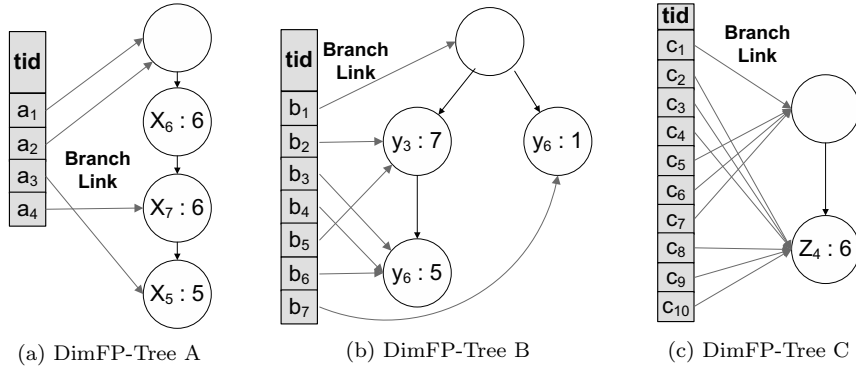


Fig. 2: DimFP-Trees for each dimension table

### 3.1.1. Constructing the DimFP-Tree

A DimFP-Tree is very similar to an FP-Tree.<sup>11</sup> There are two major 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 the entire database, 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( $T$ ). For example,  $b_4$  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 another structure, the branch table, that keeps track of the path correspondent to each  $tid$ . It stores the last node of that path for each  $tid$ . 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.

The result is the same as if we have the table with the transactions and their frequent sorted items. 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.<sup>21</sup> If we keep the table instead of the tree, and if two transactions have the same frequent items, those items will be repeated two times. With the tree, there is just one path corresponding to the two 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.

Lets consider the construction of the DimFP-Tree (Fig. 2b) of table B:

Step 2 starts with a first scan to the table B to calculate the support of each

item. Only  $y_3$  and  $y_6$  are frequent, i.e. have a support no less than four transactions ( $sup(y_3) = 7$  and  $sup(y_6) = 6$ ). Therefore, only the itemsets containing just  $y_3$  and/or  $y_6$  would be frequent, according to the anti-monotone property.<sup>16</sup> For each transaction  $b_1, b_2, \dots, b_7$ , frequent items are selected and sorted according to the support descending order, and then inserted in the tree. At the same time, the branch table is constructed, linking each  $tid$  to the respective node in the tree.  $b_1$ , for example, does not have any frequent item, therefore its branch link links to the root node.  $b_3$  corresponds to the first path of the tree, therefore its branch link points to the last node in that path. The other transactions follow the same reasoning. Fig. 2 shows the results for all dimensions.

### 3.1.2. 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.<sup>11</sup> Despite this, there are three differences:

First, it is not necessary the first scan to any table to calculate the supports. They are already calculated and stored in each dimension tree.

Second, a fact is a set of  $tids$ , therefore the denormalization of each fact is necessary before ordering the items or inserting them in the tree. A denormalized fact is an itemset with the items corresponding to its  $tids$ . Through the branch table of each DimFP-Tree we can get the path corresponding to the itemset of each  $tid$ . Furthermore, according to the anti-monotone property, if an itemset is not frequent, no other itemset containing it will be. 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. Table 2b shows the result of denormalizing each fact.

And third, the ordering of items in a transaction is not a support descending order of items. This ordering enhances the compactness of the FP-tree structure. However, this does not mean that the tree so constructed always achieves the maximal compactness.<sup>21</sup> Still, it is a promising order for the dimensions, since dimensions with higher support are more likely to be shared and thus arranged closer to the top of the FP-tree. We call it a *Support descending order of dimensions*.

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 support are ordered

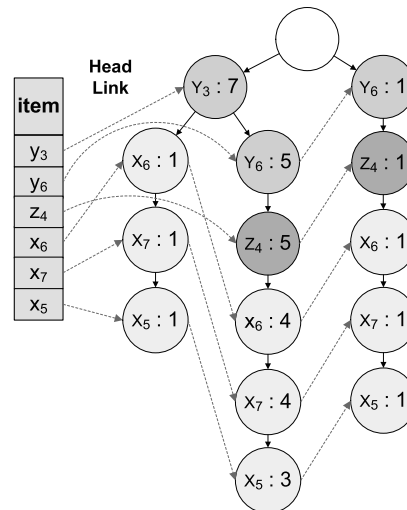


Fig. 3: Super FP-Tree with a support descending order of dimensions

alphabetically, and items of a dimension are ordered in a support descending order.

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.

In our example, the lowest support in dimensions is five in dimension A, and six in B and C. The support descending order of these dimensions is  $B \rightarrow C \rightarrow A$  ( $(y_i s) \rightarrow (z_i s) \rightarrow (x_i s)$ ). Fig. 3 presents the resulting Super FP-Tree. The branch table shows the items according to this ordering.

### 3.1.3. Complexity

Let  $m$  be the number of facts and  $n$  the number of dimensions. The size of the fact table is given by  $|FT| = m \times n$ . The size of each dimension  $D_i$  is given by  $|D_i| = t_{di} \times c_{di}$  with  $t_{di}$  and  $c_{di}$  the number of transactions and columns in each dimension  $D_i$ , respectively. Then the size of the star is  $|FT| + \sum_{i=1}^n |D_i|$ . If we needed to join the tables before mining, it would result in a much larger table, whose size would be the number of facts times the sum of all columns in all dimensions (with the exception of the *tids*). This would have a negative impact on the memory needed, as well as on the time, not just on the extra pre-processing step for the creation of this table, but also on all steps involving the scans to the database.

The first step of *Star FP-Growth* – global support – is  $O(|FT| \times \sum_i^n \log(t_{di}))$ . It consists in one scan to the fact table, to count the occurrences of each *tid*, therefore it depends on its size and on the number of foreign keys. In terms of memory, it needs to keep the support of each *tid*, which depends on the number of transactions in each dimension.

In step two – local mining – the algorithm needs to scan each dimension table twice. First to count the support of each item:  $O(|D_i| \times \log(I_{di}))$ , with  $I_{di}$  the set of possible items in  $D_i$ . Second to construct the DimFP-Tree:  $O(|D_i| \times c_{di}^2 \log(c_{di}))$ . Sorting and inserting a transaction in the tree will depend on the number of frequent items in it. In the worst case, all items are frequent, and its sorting and insertion is bounded by the number of columns in  $D_i$ . Star FP-Growth has great advantages, because: as the dimensions are smaller than the joint table, the counting of support and the construction of the tree is faster; it immediately discards the dimension tables after constructing the corresponding DimFP-Tree; and usually there are a lot of shared nodes and the size of this tree is much smaller than the size of the table. However, in the worst case, each transaction corresponds to one path in the tree and its size is identical to the table.

When constructing the Super FP-Tree, Star FP-Growth needs to scan the fact table one more time, to go to each DimFP-Tree and get the frequent items ( $c_{di}$  in the worst case) of each *tid* ( $\log(t_{di})$  to look for it in the branch table). These frequent items are then inserted in the Super FP-Tree. This step is thus limited by  $O(|FT| \times \sum_i^n (\log(t_{di}) \times c_{di}))$ .

The final step runs the algorithm FP-Growth as it is (for more details, see <sup>21</sup>).

#### 4. Case Study

There is untold value in bibliographic information, but it is largely untapped. The bibliographic world has been guided by normalized formats to describe its works, such as UNIMARC<sup>22</sup> and MARC21.<sup>12</sup> These formats have been effective, but they denote some inadequacy relating to the actual Internet paradigm, digital content and the need of interoperability between different libraries and other entities.

The need to reduce cataloguing costs by minimizing duplicate cataloguing effort, to simplify the cataloguing process and to adapt practices is increasing. To achieve that, new models are arising, and Functional Requirements for Bibliographic Records (FRBR) is the most revolutionary case in the bibliographic community.<sup>23</sup> FRBR is a conceptual model of the bibliographic universe, that no longer sees the catalogue 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.

There are several data mining works in this bibliographic area (see for example <sup>24</sup>, <sup>25</sup> and OCLC research<sup>2</sup>), but few are about finding co-relations or patterns in the bibliographic data. The goal of this case study is to extract patterns from Porbase, the Portugal Bibliographic Database, that contains more than 1.5 millions of works from more than 170 Portuguese libraries. The data in Porbase is in the UNIMARC format, and the patterns found 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.

##### 4.1. *Data in Porbase and its challenges*

The data in the Porbase is in the UNIMARC format,<sup>22</sup> stored in eXtensible Markup Language (XML) files.

UNIMARC defines a common structure for the exchange of bibliographic works, in a machine-readable form and specifies the logical and physical format of the works. It specifies that every bibliographic work is composed of a record label, a set of data fields and a set of control fields. It defines about 175 possible data fields, which can be subdivided in subfields (in a total of 1150 combinations “field –

<sup>2</sup>Online Computer Library Center (OCLC) Data Mining Projects: <http://www.oclc.org/research/projects/mining>

subfield”). There are also other possible combinations between fields, subfields and control information.<sup>25</sup> An example of a bibliographic work in porbase can be seen in fig. 4.

```
<mx:record xmlns:mx="info:lc/xmlns/marcxchange-v1"
  format="Unimarc" type="bibliographic">
  <mx:leader>00800cam 02200229 04500</mx:leader>
  <mx:controlfield tag="001">1505</mx:controlfield>
  <mx:datafield tag="200" ind1="1" ind2="">
    <mx:subfield code="a"><Os >Lusiadas</mx:subfield>
    <mx:subfield code="e">poema épico</mx:subfield>
    <mx:subfield code="f">Luís de Camões</mx:subfield>
  </mx:datafield>
  <mx:datafield tag="205" ind1="" ind2="">
    <mx:subfield code="a">Nova edição</mx:subfield>
    <mx:subfield code="f">contendo uma introdução do
      Dr. Lopes de Almeida</mx:subfield>
  </mx:datafield>
  <mx:datafield tag="210" ind1="" ind2="">
    <mx:subfield code="a">Porto</mx:subfield>
    <mx:subfield code="c">Lello & Irmão</mx:subfield>
    <mx:subfield code="d">1980</mx:subfield>
  </mx:datafield>
  <mx:datafield tag="700" ind1="" ind2="1">
    <mx:subfield code="a">Camões</mx:subfield>
    <mx:subfield code="b">Luís de</mx:subfield>
    <mx:subfield code="f">1524?-1580</mx:subfield>
    <mx:subfield code="3">10642</mx:subfield>
  </mx:datafield>
</mx:record>
```

Fig. 4: Example of a work in Porbase: “*Os Lusíadas*”, a well known Portuguese epic poem (“*The Lusiads*”), by *Luís Vaz de Camões*.

In datafield 200 – title and responsibility, we can see the title, subtitle and first author. Datafield 205 shows information about the edition and datafield 210 about publication. More information about the authors can be found in datafield 700, like their names and dates of birth and death.

How to deal with the high amount of bibliographic works, and of possible fields and subfields, is a challenge and makes this data not suitable for data mining, since the format is not tabular.

Another 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).

Further, several fields are specific for each type of work and many are not mandatory, therefore, the transactional table of this data would be very sparse (only 6% of all subfields are used in more than 50% of the works). Also, some data is textual (the title, for example).

To overcome these challenges, a star-based schema was created for this data. This model enables us to split the data into several tables, which helps dealing with the high number of attributes and works. The fact table deals with the sparsity of data, keeping only the existing associations between fields and values. The dimension tables deal with the textual data, since each possible value is stored there once, and there are no repetitions, which saves space. The model is also easy to extend. We can add new dimensions and attributes to take into account the relevant aspects for the analysis. We can also add records from different libraries without changing the way to deal with them, which would allow us to discover patterns between libraries.

The star in analysis is described next, as well as the results of the application of Star FP-Growth to it, and the comparison with FP-Growth.

#### 4.2. *Star description*

The bibliographic data in this study was stored in a star schema (see fig. 5), with five dimensions and a fact table, described below:

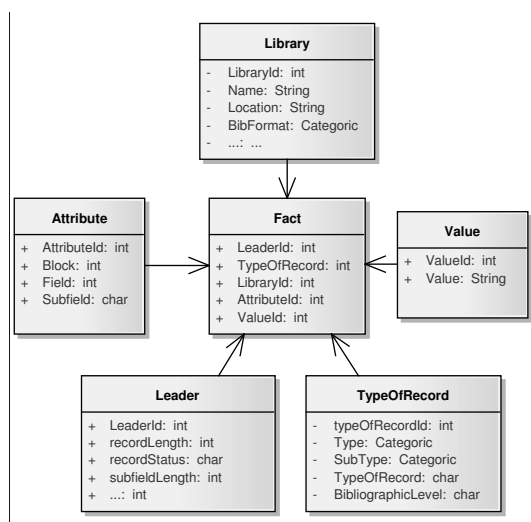


Fig. 5: UNIMARC star schema

**Dimension *Attribute*:** Holds every field and possible subfields of a record. It has an identifier (*AttributeId*) and other three fields:

**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 it is for local use and definition only).

**Field:** The number that represents the field or controlfield. It ranges from 000 to 899. (the left most digit corresponds to the block of the field).

**Subfield:** One character that references the subfield identifier. It can be a letter or a digit.

Its hierarchy is Block >> Field >> Subfield;

As example, “100:a” corresponds to Block = 1; Field = 100; Subfield = a.

**Dimension *Value*:** Keeps the values of every field; It only has the identifier (*ValueId*) and other field:

**Value:** The string corresponding to the value of a field.

**Dimension *Leader*:** Identifies the work 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 work has only one leader, and there are no two different works with the same leader;

**Dimension *TypeOfRecord*:** To keep information about each possible type of work. The UNIMARC defines two characters to represent the types of works: one to identify the major type (e.g. language materials, music scores or multimedia), and other for the bibliographic level (e.g. component or collection).

**Dimension *Library*** With information from libraries and formats. This dimension in this star schema allow us to represent and analyze data from different libraries and data in different formats. We can group data from each library and find patterns relating to them.

**Fact table:** Links all dimensions. Each row is a tuple (leaderId, attributeId, valueId, typeOfRecordId, libraryId), representing the value that the record (with that leader), of a specific type and library, has in the respective attribute.

The data used in this tests reflect the bibliographic works of Porbase on June, 2008. The version of UNIMARC is the fourth revision, second edition.<sup>22</sup> For a first analysis, only monographs were considered (that correspond to 89% of all records), and therefore, dimension *TypeOfRecord* was discarded. Only dimensions *Attribute*, *Value* and *Leader* were used in these experiments.

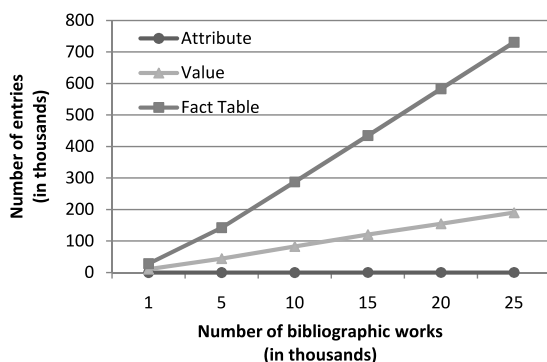


Fig. 6: Number of entries in each table

The fact table is a multi-transaction fact table,<sup>10</sup> which means that there are several transactions (or facts) in that table to describe the same bibliographic work. Each fact describes one property of the work (in this case, an association “attribute – value”). By modeling data in this star, we deal with the variable number of fields or subfields for each work, only storing the existing associations. As can be seen in figure 6, the size of the fact table grows significantly: the number of entries is almost 30 times more than the number of works. This is explained by the fact that each work has an average of 30 fields or subfields, which means that each new work

The *Leader* table has as many entries as the number of bibliographic works, once it is unique. Figure 6 shows the size of other dimensions and the fact table. The *Attribute* 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. Entries of *value*'s table are always increasing, once values are mostly textual.

introduces in the fact table an average of 30 associations “attribute – value” (30 entries). Therefore, to find an attribute or a value that occurs in every work, its frequency must be, at least, approximately 3% (one in thirty).

As FP-Growth receives as input a flat table, we have to join these dimensions and the fact table before running it.

### 4.3. *Experimental results*

First, the time and memory of the pre-processing step is studied. Then, the generated trees are analyzed, compared and evaluated. The performance of the algorithms is also 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 number of bibliographic works and supports.

The datasets used in the tests consist of a set of 1000 to 25000 monographs, and the support was varied from 5% to 0.1%.

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, they were freed as soon as they are no longer needed.

#### 4.3.1. *Pre-Processing*

Before running the algorithms, the input data had to be loaded into memory. While Star FP-Growth only needs the star, FP-Growth needs the table resulting from the joining of all dimensions and facts. Therefore, it requires an extra step, which results in more time for pre-processing. Figure 7 illustrates this difference. We can note that time spent in pre-processing for FP-Growth is twice as much as the time for StarFP-Growth.

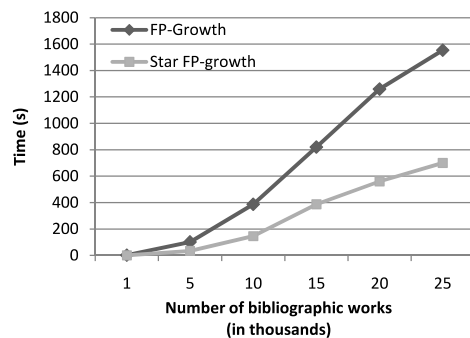


Fig. 7: Average time needed in pre-processing

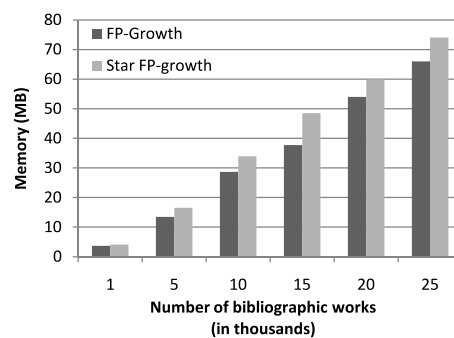


Fig. 8: Average memory used after pre-processing

Figure 8 reveals that, in the end, the star uses a little more memory than the

joint table. This can be explained by the characteristics of this bibliographic data: the values occupies much space, since they are textual, but there are very few frequent values. This means that the flat table will not have much repetitions of them. In other words, in this data, the space saved by the star by repeating only the respective foreign keys instead of the values, will not overcome of the space it needs to store all foreign keys in order to keep track of the relations between the dimensions (the flat table does not have any foreign key). Nevertheless, the difference is not significant.

#### 4.3.2. Tree's evaluation

In order to obtain a better understanding of the trees according to the characteristics of data, both DimFP-Trees and the SuperFP-Tree are analyzed. This SuperFP-Tree is also compared with the FP-Tree created by FP-Growth.

After running both algorithms for a number of bibliographic works ranging from one thousand to 25 thousand, we note that the sizes of the trees do not vary much with the increasing of works processed. This reveals that the frequent attributes and values are almost the same, independently of the works. Thereafter, next analysis only show the results for a fixed number of 10 thousand bibliographic works, but it reflects the behavior of the algorithms for other number of works.

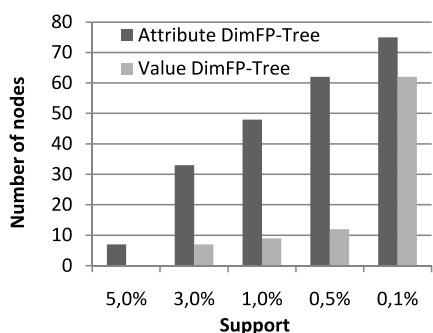


Fig. 9: Size of Attribute and Value DimFP-Trees

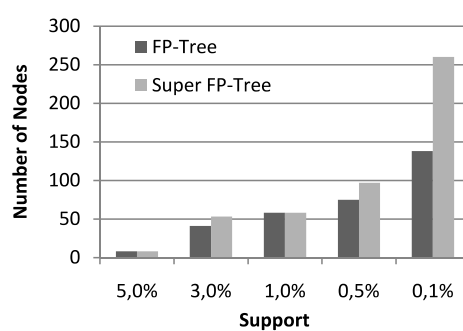


Fig. 10: Size of the FP-Tree and the SuperFP-Tree

**Attribute** The Attribute dimension has only two columns, therefore its tree 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. After the experiments (see fig. 9), we note that the number of nodes in the Attribute DimFP-Tree increases as the minimum support decreases, but just by a few nodes.

This may indicate and reflect the fact that most of the fields or subfields are infrequent. As we can see in data usage studies,<sup>25</sup> about 80% of the fields in monographs are only used in less than 1% of the works, and only

6% are used in more than 50% of the works.

**Value** Value 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 in figure 9 show that the size of this DimFP-Tree almost does not vary for supports less than 0.5%. As said before, if a value appears in every bibliographic work, its frequency has to be 3% of all facts. In the figure we can see that there are seven values that are common to every work.

Note that, for example, for 10 thousands works, if we had to keep the dimension table in memory, we would have to keep more than 82 thousand entries (size of value table). With the DimFP-Tree, we only keep what is frequent, which is much less than the size of the table.

The final tree will contain only the frequent itemsets and therefore the final patterns, i.e. the frequent co-occurrences of attributes and values within the bibliographic works. Fig. 10 compares the size of the FP-Tree, created by FP-Growth, and of the SuperFP-Tree, created by StarFP-Growth, when the support vary.

As one can see, the size of both trees increases as the minimum support decreases, once we require fewer occurrences of the same items or itemsets. By using a support descending order of dimensions instead of a support descending order of items, StarFP-Growth tries to create a more compact tree than FP-Growth. However, in this data, the SuperFP-Tree has the same or some more nodes than the FP-Tree. The difference is not much, and the algorithms will perform very similar when mining this final tree.

#### 4.3.3. Performances

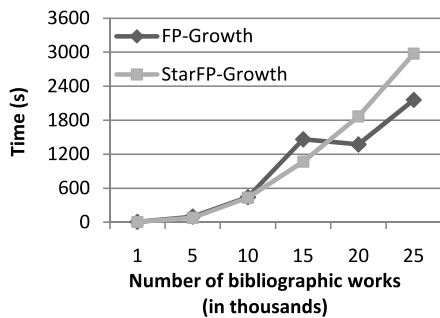


Fig. 11: Time spent in the algorithm, for 3% of minimum support

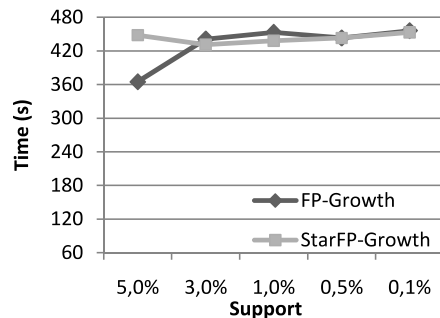


Fig. 12: Time spent in the algorithm, for 10 thousand works

In terms of temporal performance, and without considering the pre-processing step,

both algorithms behave similar, as can be seen in fig. 11 and fig. 12. This happens because, on one side, FP-Growth has to build a tree from a larger table, but on the other side, StarFP-Growth has more steps to do. In the last case, the counting of support, local mining and the global mining steps use about one third of total time, each.

As the number of bibliographic works increases, the time spent in the algorithm also increases (Fig. 11). This depends on the size of the tables (two scans to each table) and on the number of frequent itemsets (that influence the size of the trees). We stated before that, in this data, the sizes of the trees do not vary much with the increasing of works processed. Therefore, for the same support, the increase in time is mostly due to the increase of the size of the tables (shown in Fig. 6).

In other hand, as the support decreases, the time needed usually increases (Fig. 12), since there are more frequent items and, consequently, more items in the final tree to mine. In this case, the increase of trees size is not significant for time variations.

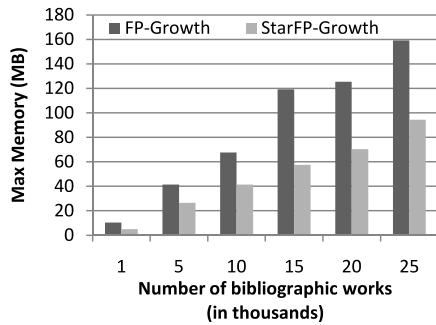


Fig. 13: Max memory used, for 3% of minimum support

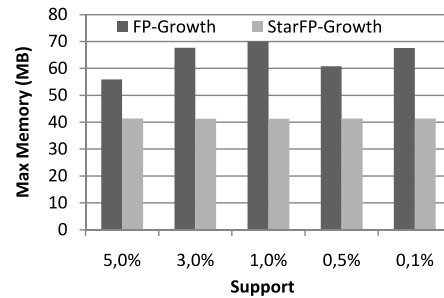


Fig. 14: Max memory used, for 10 thousand works

In terms of memory, StarFP-Growth outperforms FP-Growth (Fig. 13 and Fig. 14). In StarFP-Growth, all dimension tables are discarded after constructing the respective DimFP-Trees, and after constructing the Super FP-Tree, both the fact table and the DimFP-Trees are discarded, and we only have the Super FP-Tree in memory. For FP-Growth, before discarding the flat table right after constructing the FP-Tree, the max memory needed is more than for our algorithm.

As seen in Fig. 13, as the number of works increase, tables are larger, and therefore, the max space needed also increases. Fig. 14 shows that, as the support decreases, the max memory needed for StarFP-Growth almost does not change. This is because we start discarding dimension tables right after the construction of the first DimFP-Tree, and when we construct the SuperFP-Tree, there are no tables in memory. This max memory will change if the number of frequent items in one dimension increases a lot, or if the number of patterns explode.

#### 4.4. *Patterns evaluation*

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

##### 4.4.1. *Quantitative analysis*

Applying StarFP-Growth and FP-Growth to this bibliographic data resulted in the same patterns, since although the input data is represented in different ways, they are mining the same data. The number of patterns found for 10 thousand works can be seen in fig. 15. This number increases as the support decreases because, as has been said above, there are required fewer occurrences of the same items for them to be frequent.

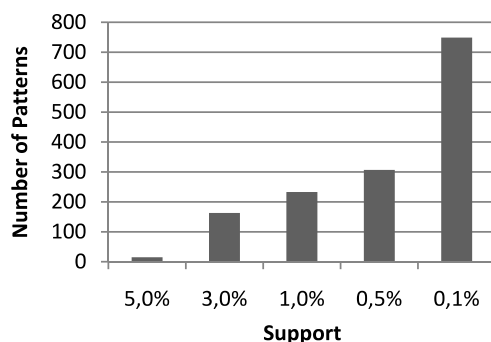


Fig. 15: Number of patterns found

Experiments on the variation of patterns for different numbers of bibliographic works show that the number of patterns found remains almost the same. For example, the number of patterns found with 3% of minimum support was always 165 (like in fig. 15), when mining 1 thousand to 25 thousand works. These results confirm that the most frequent attributes and values tend to be the same, independently of the bibliographic works being processed.

##### 4.4.2. *Qualitative analysis*

For this analysis, we will consider the patterns found in the dataset with 10 thousand bibliographic works for a minimum support of 1%. 3% is the minimum support of mandatory fields and subfields, because it indicates that it occurs at least once for each work. With 1%, we should be able to find this mandatory fields, as well as other less frequent patterns, that occur at least in one third of the total works.

Pattern	Sup.	Occurrence
(Attribute Field:Record Identifier)	3.5%	Mandatory
(Attribute Field:General Data)	3.5%	Mandatory
(Attribute Field:Language of Text)	3.5%	Mandatory
(Attribute Field:Original Source, Subfield:Agency, Subfield:Country, Subfield:Language)	3.5%	Mandatory
(Attribute Field:Statement of Resp., Subfield:Title Proper)	3.5%	Mandatory
(Attribute Field:Statement of Resp., Subfield:Subtitle)	1.4%	Optional
(Attribute Field:Version)	3.5%	Recommended
(Attribute Field:Country of Publication)	3.5%	Optional
(Attribute Field:Publication, Subfield:Publisher)	3.5%	Optional
(Attribute Field:Physical Description, Subfield:Dimensions)	3.5%	Optional
(Attribute Field:Physical Description, Subfield:Details)	1.3%	Optional
(Attribute Field:Series, Subfield:Title)	1.2%	Optional

Table 3: Patterns related with Dimension Attribute

Pattern	Sup.
(Value Language of Edition:Portuguese)	3.5%
(Value Country of Publication:Portugal)	3.5%
(Value Place of Publication:Lisbon)	1.8%
(Value Cataloguing Rules:Portuguese Cataloguing Rules)	3.5%
(Value Original Agency:National Library of Portugal)	3.5%
(Value Original Country:Portugal)	3.5%
(Value Original Language of the text:Portuguese)	3.2%
(Value Edition:med)	3.2%
(Value Other Physical Details:Illustration)	1.0%

Table 4: Patterns related with Dimension Value

Examples of patterns can be found on the tables above. For a better understanding of the patterns, the attributes' numerical values were substituted by their description and values were translated.

### Attributes

In table 3 we can analyze the occurrences of fields and subfields, and verify their real usage comparing to what is defined in UNIMARC. We can see that all five mandatory fields (first five in the table) have a support higher than 3%, meaning that they were used properly in every bibliographic work. However, all these mandatory fields are non-repeatable (one work can only have one identifier, one main title, one source, etc.), and as we can note, they are being used more than once per work in a few cases (their support is 3.5%). The field *Version* is optional, but recommended, and it is interesting to see that the recommendation is being followed and the version is being used in all works. There are also some optional fields/subfields that have a support higher than 3%, like the *Country of Publication*

and the *Publisher*. The field describing the *Dimensions* that, at first sight, does not seem very important to describe a work, is also used in all works.

By analyzing other less frequent patterns, we found that just over one third of the works have a *Subtitle* (support equals to 1.4%), to complement the mandatory *Title Proper*. Likewise, about one third of the works have the description of physical details, other than the type of material and dimensions (which all works have). We can also state that at most one third of the works belong to some series. However, we cannot say the exact number of works within a series by only looking at the pattern, since this field is repeatable if the work is in more than one.

Still about the usage of the fields, we only found 13 frequent fields for a minimum support of 1%. This indicates that only 13 fields are used in more than one third of the bibliographic works in Porbase, corroborating previous data usage studies.<sup>25</sup>

### Values

Table 4 shows the frequent values. There are seven values that occur in every record (support higher than 3%), and some refer to innate characteristics of this data in Porbase – a Portuguese database (*Country of Publication: Portugal*), catalogued using Portuguese rules (*Cataloguing Rules: Portuguese Cataloguing Rules*).

All works analyzed were Portuguese editions (*Language of Edition: Portuguese*), following a normal reference edition of Universal Decimal Classification (*Edition: med*), and that were also originally Portuguese works (*Original Language: Portuguese*), from the National Library of Portugal (*Original Agency and Country*). These characteristics may seem trivial, but it is not required that it be.

Other less frequent patterns show that about two thirds of this bibliographic works were published in Lisbon (support equals to 1.8%) and that only one third have illustrations (support of 1%).

Patterns that occur in 100% of the works (with support more than 3%), will co-occur with every other pattern. For example, we can say that “(*Value Country of publication:Portugal, Original Language of the text:Portuguese, Edition:med*)” is a patterns that occurs in every work, as well as “(*Attribute Field:Publication, Subfield:Publisher, Subfield:Place, Subfield:Date*)”.

Pattern	Sup.
(Attribute Field=Country of Pub., Value Country of Pub.:Portugal)	3.5%
(Attribute Field=Original Source, Subfield=Lang. of Text, Value Orig. Lang.:Portuguese)	3.2%
(Attribute Field=Physical Descr., Subfield=Details, Value Details:Illustration)	1.0%

Table 5: Patterns relating Dimensions Attribute and Value

### Attributes and Values

Examples of patterns relating items from both dimensions can be found on

table 5. Most of them enclose those patterns with 3.5% of support, like the first two in that table. For example, the country of publication was always Portugal, and therefore *Attribute Field:Country of Publication* always occurred with *Value Country of Publication:Portugal*. The same happened with the original language of the text. As for the details of physical description, the *Attribute Field:Physical Descr., Subfield:Details* has a support of 1.3%. From those, 70% have illustrations (*Value Other Details:Illustration*), corresponding to a support of 1% of all facts.

#### 4.5. Case study conclusions

These experiments had the goal of mining bibliographic data from Porbase, and finding 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 and a lot of time by mining the star directly. Further optimizations can be made to the algorithm to allow additional exhaustive analysis.

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, as well as the ability to discover unknown and useful information. The patterns found are somehow trivial because all works were Portuguese, but there can be foreign works published in Portugal, as well as Portuguese works published in other languages. As future work, the dataset can be extended to include those kinds of works, and further analysis should be made to find more interesting patterns. Despite all these challenges we prove that it is possible to mine bibliographic data and find patterns in it and we were able to compare and corroborate previous studies on this data.

## 5. Conclusions

Star FP-Growth is a simple multi-relational algorithm for mining patterns in a star schema. It does not perform the join of the tables, instead, it mines the star directly, and therefore it needs much less time in the pre-processing step to prepare the data.

Building a tree for each dimension taking into account the global support of items allows us to discard the respective table and to keep only the frequent items, gaining a lot of space. By gaining space, StarFP-Growth is also gaining in scalability because it is able to mine larger datasets.

The main purpose is to prepare the SuperFP-Tree that represents the frequent itemsets in data, combining the dimension trees, so that patterns can be easily discovered. 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.

StarFP-Growth outperforms its parent FP-Growth: it takes less time in the overall process, since it does not have to join the tables before mining and it needs much less memory. Also, like FP-Growth, it has some limitations: we cannot know in advance whether the patterns are useful or not, therefore it is needed further extensive analysis; and it needs to scan each table twice, and thus depends on the number of rows of the tables.

### 5.1. *Future work*

As future work, there are several aspects that can be improved:

- 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.<sup>21</sup>

- One important aspect of StarFP-Growth lies in the fact that local mining is independent for each dimension, which means that this step can be parallelized. The gains in time would be very significant.

- There are several possible optimizations that can help in the discovery of more useful patterns, being the addition of domain knowledge one of the most promising ones. By adding domain knowledge, in the form of ontologies or constraints, for example, the algorithms are able to filter the patterns and find only unknown information.

- In order to deal with the huge number of records, our algorithm can be adapted to the paradigm of data streams. Using data streams ideas, it would be able to mine one set of data each time, and update the patterns as new sets of data arrive.

- Finally, the proposed algorithm can also 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”.

## 6. Acknowledgments

This work is partially supported by *FCT – Fundação para a Ciência e a Tecnologia*, under research project *D2PM* (PTDC/EIA-EIA/110074/2009) and PhD grant SFRH/BD/64108/2009.

## References

1. Q. Yang and X. Wu. 10 challenging problems in data mining research. *International Journal of Information Technology and Decision Making*, 5(4):597–604, 2006.
2. S. Džeroski. Multi-relational data mining: an introduction. *SIGKDD Explor. Newsl.*, 5(1):1–16, 2003.
3. E. K. K. Ng, A. W.-C. Fu, and K. Wang. Mining association rules from stars. In *Proceedings of the 2002 IEEE Internat. Conf. on Data Mining*, pages 322–329, 2002.

4. P. Domingos. Prospects and challenges for multi-relational data mining. *SIGKDD Explor. Newsl.*, 5(1):80–83, 2003.
5. D. Page and M. Craven. Biological applications of multi-relational data mining. *SIGKDD Explor. Newsl.*, 5:69–79, July 2003.
6. L. Getoor. Link mining: a new data mining challenge. *SIGKDD Explor. Newsl.*, 5:84–89, July 2003.
7. J. F. Roddick, P. Fule, and W. J. Graco. Exploratory medical knowledge discovery: experiences and issues. *SIGKDD Explor. Newsl.*, 5:94–99, July 2003.
8. V. Crestana-Jensen and N. Soparkar. Frequent itemset counting across multiple tables. In *PADKK '00: Proc. of the 4th Pacific-Asia Conf. on Knowledge Discovery and Data Mining, Current Issues and New Applications*, pages 49–61, London, 2000. Springer.
9. L.-J. Xu and K.-L. Xie. A novel algorithm for frequent itemset mining in data warehouses. *Journal of Zhejiang University - Science A*, 7(2):216–224, 2006.
10. R. Kimball and M. Ross. *The Data warehouse Toolkit - the complete guide to dimensional modeling (2nd ed.)*. John Wiley & Sons, Inc., New York, NY, USA, 2002.
11. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *SIGMOD'00: Proc. of the 2000 ACM SIGMOD*, pages 1–12, New York, 2000. ACM.
12. Library of Congress Staff and National Library of Canada Staff. *Marc 21 Specifications for Record Structure, Character Sets and Exchange Media, 1999*. Library of Congress, Washington, DC, USA, 2000.
13. M. Zaki and M. Ogihara. Theoretical foundations of association rules. In *3rd ACM SIGMOD Workshop on Research Issues in DM and Knowledge Discovery*, Jun 1998.
14. L. Dehaspe and L. D. Raedt. Mining association rules in multiple relations. In S. Džeroski and N. Lavrač, editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297, pages 125–132. Springer-Verlag, 1997.
15. S. Nijssen and J. N. Kok. Faster association rules for multiple relations. In *IJCAI*, pages 891–896, 2001.
16. R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases*, pages 487–499, Sep 1994.
17. M. X. Ribeiro and M. T. P. Vieira. A new approach for mining association rules in data warehouses. In *FQAS*, pages 98–110, 2004.
18. E. Garcia and M. T. P. Vieira. Estudo de caso de mineração de dados multi-relacional: aplicação do algoritmo connetionblock em um problema da agroindústria. In *SBBD '08: Proceedings of the 23rd Brazilian symposium on Databases*, pages 224–237, 2008.
19. J. Kanodia. Structural advances for pattern discovery in multo-relational databases. Master's thesis, Rochester Institute of Technology, Rochester, NY, 2005.
20. W. H. Inmon. *Building the data warehouse (2nd ed.)*. John Wiley & Sons, Inc., New York, NY, USA, 1996.
21. J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, 8(1):53–87, 2004.
22. IFLA. *UNIMARC: an introduction. Understanding the UNIMARC format*, 1999.
23. IFLA. *Functional Requirements for Bibliographic Records: Final Report*. K. G. Saur, München, 1998.
24. S. Nicholson and J. Stanton. Gaining strategic advantage through bibliominig: Data mining for management decisions in corporate, special, digital, and traditional libraries. In *Organizational data mining: Leveraging enterprise data resources for optimal performance*, pages 247–262. Idea Group Publishing, 2003.
25. K. Hegna and E. Murtomaa. Data mining MARC to find: FRBR?, mar 2002.