# Mining Patterns from large Star Schemas based on Streaming Algorithms

Andreia Silva and Cláudia Antunes

**Abstract** A growing challenge in data mining is the ability to deal with complex, voluminous and dynamic data. In many real world applications, complex data is not only organized in multiple database tables, but it is also continuously and endlessly arriving in the form of streams. Although there are some algorithms for mining multiple relations, as well as a lot more algorithms to mine data streams, very few combine the multi-relational case with the data streams case. In this paper we describe a new algorithm, *Star FP-Stream*, for finding frequent patterns in multi-relational data streams following a star schema. Experiments in the emphAdventureWorks data warehouse show that Star FP-Stream is accurate and performs better than the equivalent algorithm, FP-Streaming, for mining patterns in a single data stream.

## 1 Introduction

In many real world applications, complex data is not only organized in multiple database tables, but it is also continuously and endlessly arriving in the form of streams. For example, in a real-time sales analysis problem, we need to handle a fast stream of sales transactions, along with a much slower stream of customers (news and recurring ones), and a mostly static stream of products (for long periods of time). Consequently, processing and learning from multiple data streams have become a very important data mining task.

Although there are some algorithms for mining multiple relations, as well as a lot more algorithms to mine data streams, only few combine the multi-relational case with the data streams case [8]. Multi-relational data stream mining, MRDSM,

Department of Informatics Engineering
IST – Technical University of Lisbon
Lisbon, Portugal 2900
E-mail: {andreia.silva, claudia.antunes} @ ist.utl.pt

is an emerging and inter-disciplinary area of data mining that encompasses both problems.

Multi-relational data mining (MRDM)[4] is a fairly recent area that aims for learning from multiple tables, related somehow by foreign keys, in its original structure, i.e. without joining all the tables in one before mining. A commonly used structure for databases is a *star schema*, which is composed of a central fact table linking a set of dimension tables. In a star schema, data is modeled as a set of facts, each describing an event or occurrence, characterized by a particular combination of dimensions. In turn, each dimension aggregates a set of attributes for a same domain property or constraint [9]. In recent years, the most common mining techniques for a single table have been extended to the multi-relational context, but there are few dedicated to star schemas ([2, 12, 14, 13]).

Frequent pattern mining over data streams [10] is also a very important area of data mining that allows us to handle large amounts of data, which may arrive continuously and endlessly. Its main ideas are to avoid multiple scans of the entire datasets, optimize memory usage, and use a small constant time per record. Existing techniques are able to avoid large amounts of data at a time, keeping only the needed information in some summary data structure. Most of the existing algorithms for mining data streams are designed for a single data table ([11, 6, 10]).

In this paper we describe a MRDSM method, *Star FP-Stream*, for finding frequent patterns in multi-relational data streams modeled as a star schema (star streams). The algorithm is an extension of the data streams' algorithm *FP-Streaming* [6] based on *FP-Growth* [7], and adopts the "mining before join" strategy of the MRDM algorithm *Star FP-Growth* [13].

In section 2 we define the problem of mining frequent itemsets in star streams, and then we present the related work, in section 3. The proposed method is described in section 4. Experimental results are shown and discussed in section 5 and section 6 concludes the paper with a discussion about the results achieved and some guidelines for future work.

## 2 Problem Statement

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.

Let $S$ be a tuple ($D_1$, $D_2$, ..., $D_n$, $FT$) representing a data warehouse modeled as a star schema, with $D_i$ corresponding to each dimension table and $FT$ to the fact table.

Also, let $I = \{i_1, i_2, \ldots, i_m\}$ 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*), and a subset of items is denoted as an *itemset*. $T = (tid, X)$ is a tuple where $tid$ is a tuple-

id (corresponding to a primary key) and $X$ is an itemset in $I$. Each dimension table in $S$, is a set of these tuples. Transactions on the fact table, from now on called *facts*, are sets of $n$ $tids$: tuples of the form $(tid_{D_1}, tid_{D_2}, ... tid_{D_n})$.

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 $D_i$, is the number of occurrences of $X$ in $D_i$ ($D_i$.localSup($X$)).

The *global support* (or just *support*) of an itemset $X$ is the number of facts that contain the $tid$s of the dimensional tuples that contain $X$ (given by $tid(X)$), as in equation 1:

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

The problem of multi-relational frequent pattern mining over star schemas is to mine all itemsets whose global support is greater or equal than $\sigma \times |FT|$, where $|FT|$ is the number of facts and $\sigma \in ]0,1]$ the user defined *minimum support threshold*.

The previous definitions consider that the database is mined all together. Let us now assume that the tables are data streams, where new transactions arrive sequentially in the form of continuous streams. In this *star stream*, only the fact stream connects the other tables / streams. Let the *fact stream* $FS = B_1 \cup B_2 \cup ...B_k$ be a sequence of batches, where $B_k$ is the current batch, $B_1$ the oldest one, and each batch is a set of facts. Additionaly, let $N$ be the current length of the stream, i.e. the number of facts seen so far. In this streaming case, we should only see one fact and one batch at a time, and only once. Therefore, we have to store information that allow us to keep track of all patterns, taking into account that current infrequent itemsets may become frequent later, as well as current frequent ones may no longer be, after mining the following batches. As it is unrealistic to hold all streaming data in the limited main memory, and if we assume a *deterministic* approach (with no probability of failure), data streaming algorithms have to sacrifice the correctness of their results by allowing some counting errors. These errors are bounded by a user defined *maximum error threshold*, $\varepsilon \in [0,1]$, such that $\varepsilon \ll \sigma$. Thus, the support calculated for each item is an approximate value, which at most has an error of $\varepsilon N$.

The problem of multi-relational frequent pattern mining over star streams consists in finding all itemsets whose estimated support is greater or equal to $(\sigma - \varepsilon) \times N$.

## 3 Related Work

There are many stand-alone algorithms to mine different types of patterns in traditional databases (with no streaming data and only one table), with *FP-growth* [7]

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. The algorithm represents the data into a compact tree structure, called *FP-tree*, to facilitate counting the support of each set of items and to avoid expensive, repeated database scans. It then uses a *depth-first search* approach to traverse the tree and find the patterns.

Some of the traditional algorithms have been extended to be able to mine several tables, as well as to deal with data streams. In this work we will focus on the task of frequent pattern mining (PM).

### *3.1 Multi-Relational PM over Stars*

The work on multi-relational pattern mining over star schemas has been increasing in the last years. In order to deal with multiple tables, pattern mining has to join somehow the different tables, creating the tuples to be mined. An option that allows the use of the existing single-table algorithms, is to join all the tables in one before mining (a step also known as propositionalization or denormalization). However, there are several disadvantages, like the possible explosion of attributes and null values, and the difficulty of this task when dealing with complex relations between the tables. 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 [12].

The first multi-relational methods have been developed by the *Inductive Logic Programming* (ILP) community about ten years ago (WARMR [3]) , 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 large schemas. Another drawback of ILP approaches is that they need all data in the form of prolog tables.

Few approaches were designed for frequent pattern mining over star schemas: an apriori-based [1] algorithm is introduced in [2], wich first generates frequent tuples in each single table, and then looks for frequent tuples whose items belong to different tables via a multi-dimensional count array; [12] proposed an efficient algorithm that mines first each table separately, and then two tables at a time to find patterns from multiple tables; [14] presented *MultiClose*, that first converts all dimension tables to a vertical data format, and then mines each of them locally, with a closed algorithm. The patterns are stored in two-level hash trees, which are then traversed in pairs to find multi-table patterns; StarFP-Growth, proposed in [13], is a pattern-growth method, based on FP-Growth [7], that avoids the candidate generation processing. Its main idea is to construct an FP-Tree for each dimension (*DimFP-Tree*), accordingly to the global support of its items. These trees are compact representations of the patterns of the respective dimension. Then, the algorithm builds a Super FP-Tree, combining the FP-Trees of each dimension, accordingly to the facts, and at last calls the original FP-Growth with this tree to find multi-relational patterns.

### *3.2 Data Streams*

In many real world applications, data appears in the form of continuous data streams, as opposed to traditional static datasets. A data stream is an ordered sequence of instances that are constantly being generated and collected. The nature of this streaming data makes the mining process different from traditional data mining in several aspects: (1) each element should be examined at most once and as fast as possible; (2) memory usage should be limited, even though new data elements are continuously arriving; (3) the results generated should be always available and updated; (4) frequency errors on results should be as small as possible. This implies the creation and maintenance of a memory-resident summary data structure (also called *synopsis data structure*), that stores only the information that is strictly necessary to avoid loosing patterns [10]. Hence, data stream mining algorithms have to sacrifice the correctness of their results by allowing some counting errors. Existing approaches are deterministic or probabilistic: an algorithm is *deterministic* if it only allows an error in the frequency counts, and is *probabilistic* if it also allows a probability of failure. In this work we will focus on deterministic algorithms, so that we can have the guarantee that all existing patterns are found and returned.

The first algorithm for mining all frequent itemsets over all streaming data was *Lossy Counting* [11]. It divides the data stream into batches with width $\lceil 1/\varepsilon \rceil$, so that the batch id exactly refers to the threshold $\varepsilon N$ (the maximum error). Each itemset is associated with its estimated frequency and its maximum error. At each batch boundary, all itemsets which frequency plus its maximum error is less than the current batch are discarded. In the end, itemsets with frequency of at least $(\sigma - \varepsilon)N$ are returned, and all true patterns are guaranteed to be reported.

Reference [6] presented a novel algorithm, *FP-Streaming*, that adapts FP-Growth [7] to mine frequent itemsets in time sensitive data streams. The stream is divided into batches and the arriving transactions are stored in a new FP-tree structure. At each batch boundary, the frequent patterns are extracted from that FP-tree by means of the FP-Growth, and the pattern-tree structure (called *FP-stream*) is updated and pruned. Each node in this tree represents a pattern (from the root to the node) and its frequency is stored in the node, in the form of a *tilted-time window table*, which keeps frequencies for several time intervals. The tilted-time windows give a logarithmic overview on the frequency history of each pattern, allowing the algorithm to address queries requesting frequent itemsets over arbitrary time intervals, rather than only over the entire stream (called a *landmark model*). FP-Streaming returns all patterns with an estimated frequency larger than $(\sigma - \varepsilon)N$, and guarantees, like Lossy Counting, that all frequent itemsets in $N$ are delivered.

Some other algorithms were proposed to mine frequent itemsets in data streams (see [10] for a more exhaustive survey), but most of them are adaptations of the strategies applied in the algorithms above.

### *3.3 MRDM over Data Streams*

To the best of our knowledge, there are only two works on multi-relational frequent pattern mining over data streams, both based on ILP.

In [5] *SWARM*, a Sliding Window Algorithm for Relational Pattern Mining over data streams, is proposed. SWARM is a deterministic approach for data streams, that iteratively generates and evaluates the candidates for each slide. It keeps the patterns in a *Set Enumerated tree* that enumerates all possible patterns, and stores a sliding vector in each node with the support of the respective patterns for each slide. New patterns lead to the expansion or update of the tree. When a new slide flows, the support vector is shifted to remove the expired support and the tree is pruned to eliminate unknown patterns. For dealing with multi-relational databases it is based on WARMR [3], that needs all data in prolog form, composed with predicates of variables and constants (atoms), each representing the relations in the database.

Finally, [8] presented *RFPS* (Relational Frequent Patterns in Streams), a probabilistic approach based on period sampling, for finding relational patterns over a sliding time window of a relational data stream. Since it is also based on WARMR [3], it needs the database in prolog form. RFPS is an apriori-based algorithm[1] that first generates and tests candidates with the help of a *Patterns Joint Tree* (with the possible refinements of atoms), and then maintains frequent patterns in a *virtual stream tree*, based on a periodical sampling probability.

## 4 Mining Star Streams

*Star FP-Stream* is a MRDM algorithm for mining multiple relational data streams. It is able to find approximate frequent relational patterns in large databases following a star schema, assuming that patterns are measured from the start of the stream up to the current moment (landmark model). *Star FP-Stream* combines the strategies of two algorithms: *Star FP-Growth* [13] (MRDM algorithm over star schemas) and FP-Streaming [6] (data streaming algorithm), both based on the traditional algorithm FP-Growth [7].

As referred in [9], dimensions are, by definition, smaller than the fact table. Therefore, we assume that all dimension tables are kept in memory, and only the fact table (the *fact stream*) is arriving in batches.

The fact stream is conceptually divided into $k$ batches of $\lceil 1/\varepsilon \rceil$ transactions each, so that the batch id ($1..k$) exactly refers to the threshold $\varepsilon N$ (the maximum error allowed is $k$, one per batch).

All items that appear more than once in a batch are frequent with respect to that batch, and potentially frequent (or *subfrequent*) with respect to the entire stream. Items that appear just once in a batch can be discarded because they are *infrequent*, and even if they reappear later in other batches and become frequent, the loss of support will not affect significantly the calculated support (error is less than $k$).

To mine the star, the algorithm uses *Star FP-Growth* [13] techniques. The first batch is processed separately with Star FP-Growth, so that we can fix the order of frequent items for all next batches. For each new batch, the idea is to build the DimFP-Trees for each dimension as new facts arrive, with the respective occurring transactions (local mining step). When a batch is completed, the DimFP-Trees are combined to form a Super FP-Tree (global mining step), which will contain the itemsets of all dimension that co-occur in the current batch. To combine the DimFP-Trees, we have to scan the facts of the batch a second time, otherwise we would not know which paths of the different DimFP-Trees occur together. However, a fact is just a set of *tids*, therefore this extra scan is not significant.

These DimFP-Trees can then discarded, and the Super FP-Tree can be mined to extract frequent itemsets. These patterns are maintained and updated in a *pattern-tree*, which is a data structure based on the FP-tree [7] that maintains crucial, quantitative information only about patterns, instead of itemsets. Since there are often a lot of sharing of frequent items among patterns, the size of the tree is usually much smaller than having them in a list or a table, and therefore, searching for an itemset in it is usually much faster.

In Star FP-Stream, each node in the pattern-tree stores a pattern (corresponding to the path from the node to the root), its estimated frequency and its maximum error (which is the batch id of its insertion in the tree, since it corresponds to the number of times it could have been discarded). This information allows the algorithm to stop mining super sets of infrequent itemsets and to prune of infrequent items from the tree.

The Super FP-Tree is mined using FP-Growth algorithm, with 3 pruning strategies: for each mined itemset, if it is in the pattern tree but its estimated frequency plus its maximum error is less than or equal to the current batch id, it is infrequent in N and it will be removed in the final pruning step, thus stop mining supersets (*type II pruning*). If it is not in the pattern tree and appears only once, do not add it to the pattern tree and stop mining its supersets (*type I pruning*), but if it appears more than once, it is frequent in this batch, therefore insert it in the pattern tree and continue mining. After mining the batch, all itemsets in the pattern-tree which estimated frequency plus its maximum error is less than or equal to the current batch id are discarded (*tail pruning*).

At any point in time, Star FP-Stream can be asked to produce a list of the frequent itemsets along with their estimated frequencies, by traversing the pattern-tree and delivering all itemsets which estimated frequency plus its maximum error is not less than $\sigma N$.

## *4.1 Strengths and Weaknesses*

As a data streaming algorithm, Star FP-Stream gives the following guarantees (like [11, 6]): All item(set)s whose true frequency exceeds $\sigma N$ are returned. There are no *false negatives*; No item(set) whose true frequency is less than $(\sigma - \varepsilon)N$ is returned;

And estimated frequencies are *less* than true frequencies by at most $\varepsilon N$. As a multi-relational algorithm for star schemas, Star FP-Stream mines the star directly, without materializing the join of the tables, and all multi-relational patterns are returned.

Like any algorithm, Star FP-Stream also has some limitations: It has to scan the facts twice, first to know which transactions of dimensions occur, and second to combine them in the end of a batch. However, a fact is just a set of *tids*, therefore the time needed for each scan and the memory needed to keep it, are not significant; The pattern-tree tends to be very large, since it has to keep all frequent and subfrequent patterns. Nevertheless, its size tends to be stable as the batches increase, and it is able to return the patterns for every minimum support $\sigma \gg \varepsilon$, anytime.

## 5 Performance Evaluation

This section presents the experiments conducted to evaluate the performance of our data streaming algorithm. Our goal is to evaluate the accuracy, time and memory usage, and to show that: (1) Star FP-Stream has a good accuracy and does not miss any real pattern; (2) mining directly the star is better than joining before mining.

We assume that we are facing a landmark model, where all patterns are equally relevant, regardless of when they appear in the data. Therefore, we test Star FP-Stream over an adaptation of FP-Streaming for landmark models, which we will call Simple FP-Stream, that stores only one counter in each node of the pattern tree (instead of one per time window). Since Simple FP-Stream does not deal with stars directly, it denormalizes each fact when it arrives (i.e. it goes to every dimension and join all the transactions corresponding to the *tid*s of the fact in question), before mining it.

Star FP-Growth was also implemented so that we can run it on all data and compare the returned patterns (the exact patterns) and evaluate the accuracy of Star FP-Stream results (approximate patterns).

Experiments were conducted varying both minimum support and maximum error thresholds ($\sigma \in \{50\%, 40\%, 30\%, 20\%, 10\%\}$ and $\varepsilon \in \{10\%, 5\%, 4\%, 3\%, 2\%, 1\%, 0.5\%\}$, respectively). A common way to define the error is $\varepsilon = 0.1\sigma$ [10]. (Additionally, we use a larger error to see how worse the results are, and a smaller error to see the improvements).

Note that the course of the mining process of streaming algorithms does not depend on the minimum support defined, only on the maximum error allowed. The support only influences the pattern extraction from the pattern-tree, which, in turn, is ready for the extraction of patterns that surpass any asked support ($\sigma \gg \varepsilon$).

We tested the algorithms with a sample of the *AdventureWorks 2008 Data Warehouse* (DW) [1], a DW created by *Microsoft*, especially to support data mining scenarios.

[1] Available at http://sqlserversamples.codeplex.com/

In this work we will analyze a sample of the star *Internet sales*, which contains information about more than 60 thousand individual customer Internet sales orders, from July 2001 to July 2004. Dimension tables were kept in memory and the fact table is read as new facts are needed. We will consider four dimension tables: *Product*, *Date*, *Customer* and *SalesTerritory*, so that we are able to relate who bought what, when and where. The fact table has only the keys of those four dimensions (other attributes were removed), and each dimension has only one primary key and other attributes (no foreign keys). Numerical attributes were excluded (except the year and semester in dimension Date) as well as translations, and other personal textual attributes, like addresses, phone numbers, emails, names and descriptions.

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 algorithms were implemented using the Java Programming language (Java Virtual Machine version 1.6.0_24).

## *5.1 Experimental Results*

The accuracy of the results is influenced by both error and support thresholds. The resulting patterns of Star FP-Stream and Simple FP-Stream are the same (the algorithms only differ in how they manipulate the data).

We know that as the minimum support decreases, the number of patterns increase, since we require fewer occurrences of an item for it to be frequent. And as the maximum error increases, the number of patterns returned also tends to increase, because although we can discard more items, we have to return more possible patterns to make sure we do not miss any real one. The *precision* measures the rate of real patterns over the patterns returned by the streaming algorithm.

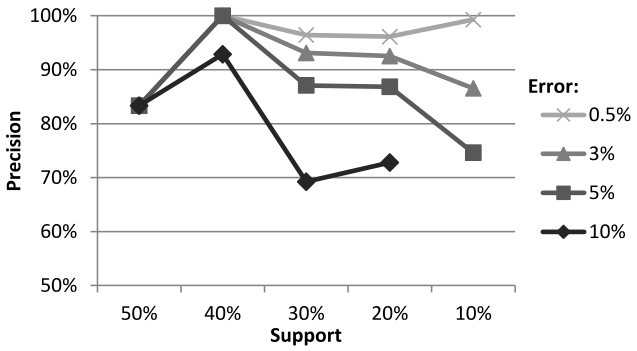


**Fig. 1** Precision variation per support

Fig. 1 presents the precision as the support varies. These results depend on the data characteristics, namely in the number of hidden patterns and in the history of

occurrences of items across the batches processed. In this case of *AdventureWorks*, we can state that for 40% of support the algorithm achieved better results (100% of precision for errors between 1% and 5%). This may mean that patterns that appear more than 40% of the times are well defined and consequently are monitored early during processing. We can also see that, as the error increases, the precision decreases, for all support thresholds. In other words, the smaller the error, fewer non real patterns are returned. The overall results show that precision is always above 69%.

The *recall* of Star FP-Stream (and Simple FP-Stream) is proved theoretically to be 100% (there are no false positives, i.e. there are no real patterns that the algorithm considers infrequent).

### 5.1.1 Time

Processing time was analyzed in terms of the time needed to process one batch (update time). It consists on the elapsed time from the reading of a transaction to the update of the pattern tree.

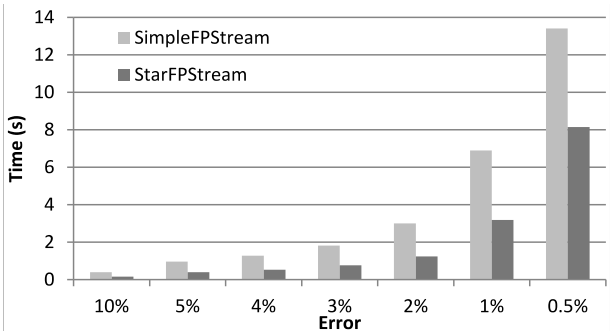


**Fig. 2** Average update time

Fig. 2 shows the average update time of both algorithms for all errors. For consistency, we do not take into account the time needed to process the first batch, since it is processed separately.

We can state that Simple FP-Stream demands, on average, more time than Star FP-Stream. This demonstrates that, for star streams, denormalize before mining takes more time than mining directly the star schema, corroborating our goal and one of the goals of MRDM.

The update time is required to be constant and not depend on the number of transactions. By analyzing in detail the time needed per batch, we verified that the update time tends to be constant and does not increase as more batches are processed. In the first batches, there are a lot of patterns in the pattern tree, and both algorithms need more time to look for and to add patterns to the tree. Around the 5500 trans-

actions, the data allows the algorithms to prune almost half of the patterns stored in the pattern tree, and then it keeps constant thereafter.

### 5.1.2 Memory

The space or memory used by the algorithms was also studied. It depends on the intermediate structures used by the algorithms, and it is strongly related with the size of the pattern tree (and therefore with the error bound). To analyze the maximum memory per batch, we measured the memory used by the algorithms for each batch, right before discarding the Super FP-Tree and doing the pruning step. We noted that the algorithms perform very similar. Star FP-Stream needs a bit more memory per batch than Simple FP-Stream, which was expected, since the first has to construct the DimFP-trees for each dimension, while the second puts the denormalized facts in just one FP-tree. Although Star FP-Stream needs, in average, a bit more memory per batch than Simple FP-Stream, it needs less memory in the worst cases. Just as in the time usage, the memory needed is higher in the first batches, and then it tends to stabilize. In the first batches, Star FP-Stream performs better in terms of memory than Simple FP-Stream, but later on, it tends to need a little more memory, for most errors. The explanation for the higher values in the first batches is the same as for the time needed.

## 6 Conclusions and Future Work

In this paper, we propose a new algorithm for mining very large data warehouses, modeled as star schema, by combining Star FP-Growth and FP-Streaming.

Experiments on Adventure Works allowed us to analyze the behavior of our algorithm, comparing its performance for several errors and supports in terms of its accuracy, time and memory needed. Results indicate that Star FP-Stream is accurate, achieving a good precision and 100% of recall. The time and memory needed by the algorithms tend to be constant and do not depend on the total number of transactions processed so far, but only on the size of the batches and on the size of the current pattern tree, which in turn depends on the characteristics of the data.

Star FP-Stream is the algorithm that performs better. Although it needs a bit more memory per batch, it is not substantial, and it needs significantly less time to process each batch, thereby overcoming the “join before mining” approach.

From this point, there are two issues to solve. First, it is important to be able to deal with measures recorded in fact tables, for example by propagating their values for influencing support. The second issue relates to degenerated dimensions, which are just used for aggregating facts. In this case, it is necessary to redefine the notion of support, since it may be different according to distinct aggregations.

## Acknowledgment

This work is partially supported by FCT – Fundação para a Ciência e a Tecnologia, under research project educare (PTDC/EIA-EIA/110058/2009) and PhD grant SFRH/BD/64108/2009.

## References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: VLDB 94: Proc. of the 20th Intern. Conf. on Very Large Data Bases, pp. 487–499. Morgan Kaufmann, San Francisco, CA, USA (1994)
2. Crestana-Jensen, V., Soparkar, N.: Frequent itemset counting across multiple tables. In: PADKK 00: Proc. of the 4th Pacific-Asia Conf. on Knowl. Discovery and Data Mining, pp. 49–61. London (2000)
3. Dehaspe, L., Raedt, L.D.: Mining association rules in multiple relations. In: ILP 97: Proc. of the 7th Intern. Workshop on Inductive Logic Programming, pp. 125–132. London (1997)
4. Džeroski, S.: Multi-relational data mining: an introduction. SIGKDD Explor. Newsl. **5**(1), 1–16 (2003)
5. Fumarola, F., Ciampi, A., Appice, A., Malerba, D.: A sliding window algorithm for relational frequent patterns mining from data streams. In: Proc. of the 12th Intern. Conf. on Discovery Science, pp. 385–392. Springer-Verlag (2009)
6. Giannella, C., Han, J., Pei, J., Yan, X., Yu, P.S.: Mining frequent patterns in data streams at multiple time granularities: Next generation data mining (2003)
7. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: SIGMOD 00: Proc. of the 2000 ACM SIGMOD, pp. 1–12. ACM, New York, USA (2000)
8. Hou, W., Yang, B., Xie, Y., Wu, C.: Mining multi-relational frequent patterns in data streams. In: BIFE 09: Proc. of the Second Intern. Conf. on Business Intelligence and Financial Engineering, pp. 205–209 (2009)
9. Kimball, R., Ross, M.: The Data warehouse Toolkit - the complete guide to dimensional modeling, 2nd edn. John Wiley & Sons, Inc., New York, NY, USA (2002)
10. Liu, H., Lin, Y., Han, J.: Methods for mining frequent items in data streams: an overview. Knowl. Inf. Syst. **26**, 1–30 (2011)
11. Manku, G.S., Motwani, R.: Approximate frequency counts over data streams. In: VLDB 02: Proc. of the 28th Intern. Conf. on Very Large Data Bases, pp. 346–357. Morgan Kaufman, Hong Kong, China (2002)
12. Ng, E.K.K., Fu, A.W.C., Wang, K.: Mining association rules from stars. In: ICDM 02: Proc. of the 2002 IEEE Intern. Conf. on Data Mining, pp. 322–329. IEEE, Japan (2002)
13. Silva, A., Antunes, C.: Pattern mining on stars with fp-growth. In: MDAI 2010: Proc. of the 7th Intern. Conf. on Modeling Decisions for Artificial Intelligence, pp. 175–186. Springer, Perpignan, France (2010)
14. Xu, L.J., Xie, K.L.: A novel algorithm for frequent itemset mining in data warehouses. Journal of Zhejiang University - Science A **7**(2), 216–224 (2006)