Condensed Representation of Frequent Itemsets

Daniel Serrano

Instituto Superior Técnico – Universidade de Lisboa
Av. Rovisco Pais, 1
1049-001 Lisboa
+351 218 419 407

daniel.serrano@tecnico.ulisboa.pt

Cláudia Antunes

Instituto Superior Técnico – Universidade de Lisboa
Av. Rovisco Pais, 1
1049-001 Lisboa
+351 218 419 407

claudia.antunes@tecnico.ulisboa.pt

ABSTRACT

One of the major problems in pattern mining is still the problem of pattern explosion, i.e., the large amounts of patterns produced by the mining algorithms when analyzing a database with a predefined minimum support threshold. The approach we take to overcome this problem aims for automatically inferring variables from the patterns found, in order to generalize those patterns by representing them in a compact way. We introduce the novel concept of meta-patterns and present the RECAP algorithm. Meta-patterns can take several forms and the sets of patterns can be grouped considering different criteria. These decisions come as a trade-off between expressiveness and compaction of the patterns. The proposed solution accomplishes good results in the tested dataset, reducing to less than half the amount of patterns found.

Keywords

Frequent itemset mining, pattern explosion, compaction, summarization.

1. INTRODUCTION

Our ability to generate and collect data has been increasing rapidly [1] and the use of the World Wide Web has been one of the major catalysts for this. It is then essential to be able to extract relevant information from the huge amount of data that we have in our power. To achieve this goal, a great deal of research has been carried out in various fields of study, ranging from statistics to machine learning and database technology.

Data mining is, according to [2], the “nontrivial extraction of implicit, previously unknown, and potentially useful information from data” and thus it is a solution to the problem of analyzing and deriving meaning out of the data. Data mining comprises four demands: *high-level language*, meaning the discovered information should be presented in an easy-to-understand language for the human user; *accuracy*, as one must guarantee that the knowledge acquired by analyzing data correctly portrays the contents of the database; *interesting results*, which means the discovered knowledge should be considered interesting according to user-defined biases; and *efficiency*, making it possible for the user to discover relevant information in a reasonable amount of time.

In order to derive meaning from the data, one must find regularities in it so that it is possible to infer general cases or what

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

IDEAS'14, July 07 - 09 2014, Porto, Portugal.

Copyright ©2014 ACM 978-1-4503-2627-8/14-7 \$15.00

<http://dx.doi.org/10.1145/2628194.2628243>

is usually called patterns. A *pattern* is “an expression in some language, describing a subset of the data or a model applicable to the subset” [3]. Patterns are considered interesting when they are novel, useful and non-trivial to compute [2].

The problem of pattern mining was first proposed by [4] for market basket analysis. Consider a supermarket with a large collection of items. If the marketing team has enough information, by analyzing its customers’ previous transactions, it will be able to predict the kind of products customers buy in the same transaction. This will allow the marketing team for rearranging the items in a smarter way that will eventually lead to increases in sales.

Frequent itemset mining, or just pattern mining, strives for completeness [5], i.e., discovering all patterns that satisfy certain conditions and although this is a useful goal, it has its drawbacks. A strong drawback is that typically many patterns will fulfill the constraints at hand in a given problem. This drawback is known as *pattern explosion*: pattern mining often produces an unwieldy number of patterns, comprising strongly redundant results which become prohibitively hard to use or understand.

Since the seminal paper on pattern mining [6], a lot of research is aimed at compacting the patterns found and some encouraging results have been achieved in the meantime, proving that there are techniques to better handle the amount of information we face nowadays.

In this paper, we present RECAP (REgression CompAct Patterns), a new pattern compaction algorithm. We aim to derive relations between the different attributes that are common to a set of patterns. This type of generalization allows us to abstract a group of patterns into a new pattern, called a *meta-pattern*, which relies on a language natural to the user and more compact than the usual representation.

The rest of this paper is organized as follows. Section 2 presents relevant concepts in the area of pattern mining and provides insight to related work that in the past has tried to mitigate the problem of pattern explosion. Section 3 clearly states the goal of our proposal by formalizing the problem and explaining key concepts needed to understand the solution. Section 4 describes the algorithm that allows the creation of meta-patterns. In Section 5 we present experimental results and evaluate the overall performance and effectiveness of our approach, taking into consideration compression ratio, memory usage and computational time, among others. The paper concludes with some guidelines for future work.

2. BACKGROUND

The problem of pattern mining can be described as follows. Let $I = \{i_1, i_2, \dots, i_m\}$ be a set of possible items (e.g., the items for sale in a store). An *item* is then a proposition – an attribute-value pair. For example, buying milk can be translated to *milk=5* (attribute *milk* and value 5), stating that the client bought 5 units

of milk or to *milk=true* (or just *milk*) stating that the client bought some milk. Let D be a set of database transactions where each transaction T is a particular set of items such that $T \subseteq I$ (e.g., the items a client bought). A set of items is referred to as an *itemset*. The support of a given itemset $I \subseteq I$ in D , denoted by $sup(I)$ is the percentage of transactions in which I occurs; in this context, a *pattern* is a frequent itemset. The problem of pattern mining is: given the minimum support threshold, σ , determine all itemsets P such that $sup(P) \geq \sigma$.

It is clear that the threshold value given as input to any pattern mining algorithm will be crucial in regards to what patterns are found. A large support value may yield only obvious patterns in the data, whilst very small values might be the ones that reveal the most interesting patterns that are not trivially graspable. The focus of our work comes from trying to mitigate the problem known as *pattern explosion*, i.e., the large amount of patterns produced by mining algorithms when restricted to low support values. Several consequences arise from this phenomenon: high memory requirements, difficulty in understanding the reported patterns and strongly redundant results.

There is a reasonable amount of work done in the area of pattern compression and particularly on trying to reduce the number of what will be considered the set of relevant patterns. One of the first methodologies used was that of constrained sets, whereby one can filter out important patterns that describe the patterns found (typically maximal [7] and closed sets [8], [9]) or patterns that are very similar to other patterns (δ -free sets [10]). More complex constraints can be used, taking into account disjunction relations [11] or the derivability (or inference) of a larger number of patterns from a particular subset [12].

A different compaction approach uses the concept of top- k patterns, which consists in limiting the number of patterns reported to only the most k important ones. This can specifically target the top- k frequent closed itemsets with a minimum length [13], the k patterns that best approximate a collection of patterns found [14], the k most significant and redundancy-free set of patterns [15] or even organize them by relevance, considering metrics like F-score or χ^2 [16].

Another potential solution comes from the use of the minimum description length principle [17]. In the context of pattern mining it is used to find the set of frequent itemsets that best compress the database at hand. The basic premise to this is that if an itemset is to be deemed as interesting, then it must yield a good (lossless) compression of the database.

Pattern compaction has also been seen as a clustering problem by separating frequent itemsets in well-defined groups of patterns. The compression task may then create a representative pattern able to represent the common regularities in each group. A greedy algorithm with an agreed tightness measure among patterns in a cluster has previously been used [18]. Alternatively, profile-based clustering which relies on the concept of a master pattern and distribution vectors has also been proposed [19].

Additionally, there has also been work on the efficient storing of patterns, particularly using an FP-tree-like structure, the CFP-Tree or Condensed Frequent Pattern tree, in which the main idea is to store only closed frequent itemsets [20].

From a completely different line of research, inductive logic programming [21], [22] also proposed important contributions to pattern discovery, since a large number of algorithms have been

proposed in this area for allowing the inference of data from background knowledge. The Classic/cl system, influenced by various recent approaches, integrates a wide range of constraints such as minimum frequency, maximum generality, exclusive disjunctions and condensed representations [23].

All these approaches have some sort of cost: information might be lost and thus the end-user, interested in inferring the maximum amount of information he possibly can, might not find useful information in the discovered patterns; it can happen that the top- k patterns selected are not enough for the end-user to get a grasp of all information that might be obtained with more discriminative representations; the efficient structures previously mentioned are not thought of to store a form of meta-patterns, i.e., these structures normally store patterns in their simplest form, but never complex representations like the ones presented next.

3. PROBLEM STATEMENT

Consider a database and one of the commonly used pattern mining algorithms (Apriori or FP-Growth, for example). When restricted to low support values, these algorithms will report an unwieldy number of patterns, which makes them hard to understand since most of the times they are presented to the end-user as a list of patterns. Even if they are grouped using some criteria (e.g., group them by attributes), often times patterns will have regularities (i.e., redundancies between them) that might be overlooked by the user. For example, one might find that for a given set of patterns, the value of an attribute is always the same. Or the value of an attribute has a coherent relation with some other attribute (e.g., one of the attributes takes values which are always twice the value of the other attribute). For this reason, we believe a compaction scheme could be set up in which such regularities are automatically identified and new patterns, called *meta-patterns*, are generated enabling a compact representation of the patterns found.

Definition 1. A *meta-pattern* is a compact representation of a set of patterns.

Consider the set of patterns in Table 1. In this example, we initially have 4 patterns but these can be compacted in 2 meta-patterns, capable of representing all the original patterns. To understand the process that leads to this outcome, we introduce the concept of meta-item.

Definition 2. A meta-pattern is composed of one or more meta-items. A meta-item is a compact representation of a set of items. In some manner, a meta-item plays the same role as a variable.

As an example, consider again Table 1. The first meta-pattern is composed by two meta-items: one describing the values taken by attribute *Diapers* and another one describing the values taken by attribute *Beers*. Different types of meta-items can be inferred, and

Table 1. Patterns and corresponding meta patterns

Patterns	Meta-patterns
Diapers = 1, Beers = 2 Diapers = 2, Beers = 4	Diapers = {1, 2}, Beers = 2Diapers
Chips = 2, Sodas = 4 Chips = 2, Sodas = 5	Chips = 2, Sodas = {4, 5}

we define three particular types of meta-items: constant meta-items, disjoint meta-items and regression meta-items.

Definition 3. A constant meta-item is a meta-item that describes a set of items sharing the same attribute-value pair.

In Table 1, the set of patterns that share attributes *Chips* and *Sodas* both contain item *Chips* = 2, which can lead us to the definition of a meta-item *Chips* = 2. The constant meta-item can be seen as an abstraction of a set of items that are equal among them.

Definition 4. A disjoint meta-item is a meta-item that describes a set of items sharing the same attribute but different values over the set of items.

Continuing from the previous example, one can see that *Sodas* takes values 4 and 5. We thus can infer a disjoint meta-item given by *Sodas* = {4, 5}. It will guarantee that only one expression is used to represent this set of items, instead of two or more. In particular, this type of meta-item is necessary when considering that we can infer a more complex type of meta-item, the regression meta-item, which is also the most expressive one.

Definition 5. A regression meta-item is a meta-item that describes a set of items sharing the same attribute and different values that are dependent on another attribute through a simple linear regression model. A regression meta-item describes the values of a given attribute y (dependent one) using a single explanatory attribute x (target one), a slope m and an intercept b , such that $y = mx + b$.

The first meta-pattern in Table 1 contains a case of such a type of meta-item, in which the values of attribute *Beers* are dependent on the values of attribute *Diapers* with a slope of 2 and an intercept of 0 ($Beers = 2Diapers + 0 = 2Diapers$).

Our problem is then to find regularities among a set of patterns and be able to extract expressions capable of describing relations among attributes. The goal is to decrease the amount of information presented to the user, while increasing its expressiveness and understanding.

3.1 Requirements

From the description of the formulated problem, we can understand that a meta-pattern is able to represent a set of patterns that share some regularity. However, it should also be clear that there are sets of patterns that are not passible of being represented by a unique meta-pattern.

Indeed, in order of being able to compact the entire set of patterns discovered on a given dataset, the patterns reported by the mining algorithm have to be reorganized in such a way that patterns are grouped together considering regularities shared among attributes. Otherwise, if the patterns were randomly distributed, we would have a combinatorial explosion in the number of possibilities to distribute patterns in sets that share the same characteristics (whichever they are; constant items between those patterns, disjoint relations between items in that group of patterns, or

simple linear regression models). It is then important to reorganize the patterns in such a way that the task of finding generalizations between the sets of patterns is eased.

1st Requirement. Patterns must be organized in a way that reduces the amount of effort needed to perform generalizations. Patterns should be formed in groups so that the search for regularities is limited to each group in which it makes sense to find those regularities.

When discussing regression meta-patterns, it is clear that the definition at hand only makes sense if we are in the presence of numerical data. In particular, two numeric attributes must be present in order for this type of meta-patterns to be found (we must have at least two dependent attributes, both having to be numeric).

2nd Requirement. Simple linear regression requires the existence of at least two numeric attributes.

If only half the attributes map to numeric data, it can happen the case in which the user knows that although a particular set of attributes corresponds to numeric data, it is not beneficial (given the specific domain of the problem) to try to find regularities or expressions that describe an attribute in terms of another attribute in that set of patterns.

3rd Requirement. We must have a way to inform the system about which attributes shall be used when considering generalization using simple linear regression.

Considering these requirements and the overall proposal presented before, we will explain the algorithm that makes it possible to find the meta-patterns previously described.

4. RECAP ALGORITHM

The problem we aim at solving is that of automatic inference of the minimal set of meta-patterns that describe an entire set of patterns. The first step for this to happen is presented next, and involves solving what we previously identified as the 1st Requirement. We must group the patterns in such a way that we only try to generalize sets of patterns that make sense to be grouped together under a singular explanatory expression – a unique meta-pattern.

After that process, we introduce the meta-pattern generation algorithm, which will use the groups of patterns previously identified in order to infer the relationships between attributes in the identified groups.

4.1 Preparation phase

The reasonable heuristic way of performing the grouping of patterns is by placing patterns that share a set of attributes in the same group. Consider once again Table 1. The first set of original patterns only makes sense to be grouped together because they both share attributes *Diapers* and *Beers*. It would not make sense to try to infer the type of expression we defined in the previous section if we tried to group retrieved patterns that do not share the same set of attributes. For example, a generalization between *Diapers* = 1, *Beers* = 2 and *Chips* = 2, *Sodas* = 4 would not be successful.

```

1 function preparation_phase(patterns):
2 begin
3   for each pattern in patterns
4     uniq_key = generate_uniq_key(pattern)
5     groups = grouped_patterns[uniq_key]
6     group = find_closest(pattern, groups)
7     if distance(pattern, group) > 0
8       group = new_group(pattern)
9     end
10    group.add(pattern)
11  end
12  return grouped_patterns
13 end

```

Figure 1. Preparation phase algorithm pseudo-code

Although this heuristic is a good starting point, it is arguable if we should group together every pattern that shares the same set of attributes. There can be cases where the support of one pattern (or more) is too different from the support of every other pattern in the group of that set of attributes. In that case, it might be better to create a separate group to consider significantly different support values between patterns sharing the same attributes. For that reason, this grouping of patterns is performed also taking into account the support of the patterns found. Furthermore, we define a distance measure as criteria to decide whether a pattern should be included in a given group of patterns. The distance measure is calculated comparing the first computed support of a given group of patterns (which will be from here on referred to as *reference support*) and the support of a pattern identified as sharing the same set of attributes of that group.

The formula for the distance is given by equation (1).

$$d = \frac{|reference - support|}{reference} \quad (1)$$

For each analyzed pattern, the distance d between a pattern and the group of patterns that shares the same set of attributes with it cannot be larger than a user-defined threshold θ , i.e., we must guarantee that $d \leq \theta$. The formula uses the reference support as the denominator so that this distance can be given as a percentage value.

The pseudo-code for the preparation phase algorithm is given in Figure 1. The grouping of patterns is done through a hash table (*grouped_patterns*), where each key corresponds to a unique identifier of the set of attributes shared by the group of patterns. Each of these identifiers contains a group of patterns that share the same attributes and have “close” supports (considering the distance measure previously defined). Figure 2 depicts the data structure described.

The patterns returned by frequent itemset mining algorithms usually give items ids as they appear in the dataset, ignoring the attributes shared, so it is possible that for example *Chips* = 2 and *Chips* = 3 have completely different ids and appear at very distinct positions in the patterns discovered. For that reason, a unique identifier for each pattern (line 4) is created by concatenating the ordered strings that represent the items. The

Figure 2. Data structure used during the preparation phase

order is important because if we did not order the strings corresponding to the attributes of a given item, we would possibly generate distinct keys (and consequently different groups) for patterns sharing the same set of attributes.

The loop beginning in line 3 is only concerned with placing the pattern in the correct group. We begin by finding the hash table corresponding to the set of attributes in the pattern (line 5). We then look for the entry that is closest in support to the new pattern (line 6). If the closest group has a support value too far from the threshold θ , then we create a new group (line 8). We then add the newly analyzed pattern to the group it fits best in (line 10).

4.2 Meta-pattern generation

After the patterns are grouped together considering the heuristic described before (sharing attributes and closeness in terms of support values), we focus on the process of generalization.

For each group of patterns, we perform an analysis on the group size and on the number of attributes it comprises. The two trivial cases occur when either the group only contains one pattern (no generalization can be made) or the group contains more than one pattern but only one attribute. In the first case, we generate a meta-pattern comprising a single constant meta-item, for each item present in that pattern. The second case implies generating a meta-pattern that can assume multiple values for a single attribute, which leads us to create a meta-pattern simply consisting of a disjoint meta-item on that attribute.

The most complex case happens when there are multiple patterns in a group, each of which comprising more than one attribute. The process of finding regularities in a group of patterns is as follows. For each attribute in every pattern belonging to that group, we try, in an ordered fashion, to describe that attribute in terms of another one (the best one, according to the simple linear regression used). This causes us to have to define a dependent (or class) variable. If a useful attribute is found, i.e., if we are able to describe the dependent variable in terms of another attribute, providing it is consistent with all patterns in the group, then two cases might arise. If the slope is 0, then we create a constant meta-item ($y = 0x + b = b$) to represent the dependent attribute. If the slope is different from 0, then a regression meta-item on that attribute is

```

1 function regression_meta_item(pattern_group):
2 begin
3   for i = 0 to i = num_attrs
4     if not target(i) and not independent(i)
5       regression.run(i, dependents)
6       if successful(regression)
7         if regression.slope = 0
8           meta_pattern.add(const_meta_item(i))
9         else
10          meta_pattern.add(regr_meta_item(i))
11        end
12        set_target(regression.target)
13        set_dependent(i)
14      else
15        set_independent(i)
16      end
17    end
18  end
19
20  for j = 0 to j = num_attrs
21    if target(j) or independent(j)
22      meta_pattern.add(disj_meta_item(j))
23    end
24  end
25
26  return meta_pattern
27 end

```

Figure 3. Meta-pattern discovery algorithm pseudo-code

Table 2. Indirect dependency example

Patterns	Indirect dependency case	Desired meta-pattern
Diapers = 1, Beers = 2, Lettuces = 3 Diapers = 2, Beers = 4, Lettuces = 6	Diapers = {1, 2}, Beers = 2Diapers, Lettuces = 0.66Beers	Diapers = {1, 2}, Beers = 2Diapers, Lettuces = 3Diapers

created. In both cases, a disjoint meta-item must be created in order to represent the possible values that the target attribute takes. The process of choosing the attributes to be generalized is repeated until we have tried to generalize all attributes that are not already classified as dependent or target ones.

The algorithm for this process is given in Figure 3. In the first loop (line 3), i represents the index corresponding to the attribute chosen to be the dependent variable in a regression run. If it has not yet been used as a target variable in some other regression computation, we try to generalize it (line 4). Simple linear regression is computed taking into account i and all the dependent attributes as constraints, since the algorithm needs to consider the i^{th} attribute as the dependent variable, while not trying to generalize it using any of the variables already inferred as dependent (line 5). This second constraint guarantees that no *indirectly dependent generalizations* are created, i.e., cases in which an attribute can be described in terms of another attribute which is in turn dependent of a third attribute (e.g., Table 2). In the process of *unfolding* a meta-pattern (generating the corresponding patterns that meta-pattern is summarizing), this type of dependencies would hinder the task of inferring the values a given attribute takes. For example, to infer the value of *Lettuces* in the example of Table 2, we would need to get the value of *Beers* first and only then would we be able to calculate the value of *Lettuces*. By avoiding this, the process of inferring the values of a regression meta-item becomes simply calculating $mx + b$ for each target value. Indirectly dependent generalizations may also confuse the end-user, since as it can be seen in Table 2 we would have attributes expressed in terms of different target variables. In that example, generalization could be made in terms of the same attribute (target variable *Diapers*), which simplifies the process of understanding dependencies.

After performing the regression calculations, we decide if a constant meta-item or a regression meta-item shall be created (lines 8 and 10) with the correspondent attributes being deemed target and dependent (lines 12 and 13, respectively). If some attribute was found to be non-generalizable (regression is not successful, i.e., no useful attribute was found) then it is considered independent (line 15).

The last loop in the function (line 19) exists so that disjoint meta-items are created out of the items that were found to act as target variables (on some regression meta-item) or for variables deemed independent (variables that could not be generalized because no relationship between attributes could be inferred).

It is important to note that the regression meta-item is created considering the regression itself. Indeed, it contains the values for

the slope and intercept, which are values that result from a successful run of simple linear regression. A regression run is considered to be successful if and only if the sum of mean squared errors is less than a user-specified error (domain-specific).

In the worst-case scenario, one will not be able to find any regression meta-item, and hence all the attributes will originate disjoint meta-items. In that case, we try to find patterns that have the same values between them and only vary in terms of one attribute. As an example consider the case of Table 3. The set of patterns in that group is not generalizable considering simple linear regression, but two groups can be further inferred if we split the group in regards to the “more decisive” attribute. The idea is similar to information gain in decision trees, but here it is simplified by using only the relative frequency of items in the group of patterns.

5. EXPERIMENTAL RESULTS

All the experiences were performed on a Pentium T4200 Dual-Core 2.00 GHz machine with 4GB RAM running Windows 7. We used the implementation of Apriori algorithm available in the D2PM project [24] to collect the frequent itemsets that are further post-processed by our algorithm.

Dataset. We used a private academic dataset to run the experiments. It contains the marks of students enrolled in the Computer Science and Engineering bachelor’s degree from 1990 to 2000 at Instituto Superior Técnico, Universidade de Lisboa in Portugal. Each one of the 2614 entries represents a different student and his corresponding marks in twelve courses. The grading scale goes from 10 to 20, with 10 being the minimum mark a student must achieve to be considered approved to any course. Only positive results have been used. In order to maximize the probability of generalization, a normalization was performed on the marks, in which each pair of consecutive values is given the lower bound (records with values 10 or 11 become 10, records with values 12 or 13 become 12, etc.). This is true for every pair except the last one, which also includes mark 20 (values 18, 19 and 20 become 18), since the highest mark is very unlikely to happen. This can be seen as a rough approximation to the American grading system, here translated from letters to numeric values (necessary for our algorithm).

We run the algorithm with different support values and analyze it considering four metrics. The first is compression ratio in terms of patterns found, which will be computed by comparing the number of patterns reported by our algorithm with: 1) the number of patterns found by the Apriori algorithm alone; 2) the number of closed and maximal patterns. We also perform time and memory consumption tests. We are interested in understanding how space and time will vary with support variations. Another comparison we will make to Apriori has to do with time. Being a post-processing algorithm it is desirable that the amount of time spent in compaction is the minimum possible.

The results in terms of numbers of patterns reported to the end-user (in the form of meta-patterns) from running the compaction algorithm with minimum support set to 10%, 7.5%,

Table 3. Group not generalizable using simple linear regression and resulting finer-grained groups

Diapers = 2, Beers = 3	Diapers = 2, Beers = 3
Diapers = 2, Beers = 5	Diapers = 2, Beers = 5
Diapers = 4, Beers = 7	Diapers = 4, Beers = 7

Figure 4. Comparison of reported patterns

Figure 6. Different types of meta-patterns

5% and 2.5% are shown in Figure 4. Note that we allow permissive pattern grouping in the preparation phase, making θ virtually infinite. It is clear that compaction is always successful, i.e., we can always reduce the number of patterns reported in comparison to Apriori. The figure also shows that the ratio of compaction (ratio between the number of patterns before compaction and after it) increases with the decrease of support. This happens because the amount of patterns found by the Apriori algorithm increases in an exponential rate. In this way, a broader set of patterns will be able to be grouped together into a meta-pattern capable of describing that set of patterns, because the number of similar (hence, generalizable) patterns will also increase. We also run the Eclat algorithm [25] using the openly available implementation by [26] to assess the amount of patterns reported to the end-user when using closed and maximal sets. As can be seen, the closed patterns take no effect in reducing the number of patterns found without missing values (note the overlap between *patterns* and *closed*). The maximal set is successful in reporting fewer patterns but not to the extent that meta-patterns do, with meta-patterns incurring in about half the number of patterns obtained by using the maximal set strategy.

In Figure 5, we compare the computational time spent by the compaction algorithm and by Apriori. It is clear that although being a post-processing algorithm, the amount of time spent in compaction is negligible when compared to the task of frequent itemset mining. It is again clear that with the decrease of support, the amount of time spent both during Apriori as well as in the compaction process increases (since with the decrease of support one will have more patterns to compact). However, the time spent on compaction grows much slower than the pattern mining step.

When running the experiments in the education dataset, we registered the amount of patterns that contained regression

Figure 5. Computational time per algorithm

Figure 7. Memory usage

meta-items, the number of meta-patterns involving disjoint relations and the number of patterns that remained the same even after compaction. In Figure 6 we can see that with increasing support values, the ability to generalize patterns increases (due to the reasons mentioned when describing the first chart). In particular, when setting the minimum support at 10%, we verify that no simple linear regression models could be successfully created from the set of patterns found and about half the meta-patterns reported were exactly the same as the ones reported by Apriori. The majority of meta-patterns contain disjoint generalizations, nevertheless. It is interesting to see that the largest number of meta-patterns involving simple linear regressions is found when the minimum support is set to 7.5%, which does not represent a minimum or maximum value on the supports used. This is very dependent on the data itself and consequently on the patterns found with the frequent itemset mining algorithm.

As to memory usage (Figure 7), the results follow what was expected, i.e., an exponential increase of the amount of memory used with the decrease of minimum support. For the lowest minimum support, memory used reaches approximately 50 MB, which shall be explained by the use of expensive Java data structures like hash maps, array lists and tree sets. We believe that it is possible to improve efficiency of the algorithm regarding memory consumption by using cheaper data structures (e.g., simple arrays) in detriment of more complex abstractions, or ultimately creating custom data structures.

In terms of time spent on actually running the compaction algorithm (Figure 8), besides the analysis performed on the overhead of it being a post-processing algorithm, we also report the time in seconds that the compaction task takes. When set to the lowest minimum support value the algorithm runs in less than

Figure 8. Time spent on compaction

1.2 seconds, which represents a good value but it is our understanding that performance analysis should be carried out in larger datasets to draw more accurate conclusions.

6. CONCLUSIONS

The explosion of the number of patterns discovered by frequent itemset mining algorithms, along with the difficulty on analyzing such number of patterns, have impaired the generalized use of pattern mining in data analysis.

In this paper, we proposed a method for compacting the discovered patterns into a more manageable set, creating abstractions of the original patterns discovered – the meta-patterns. Experimental results show that even considering very low levels of support, the compaction rate is high, and the time spent on this step is negligible compared to the time spent on pattern mining.

The proposal presented in this paper is a post-processing algorithm. A disadvantage of this approach is that the discovery of meta-patterns has to be performed after Apriori or any other frequent itemset mining algorithm has found frequent itemsets. If the process of generalization was done in runtime, i.e., while analyzing the itemsets in the database, we are confident better performance times could be achieved. This, together with the use of a custom data structure especially targeted at finding the types of meta-patterns we strive to find, could lead to better memory use.

Furthermore, the properties of the patterns grouped together are related to the concepts of additive and multiplicative models that are referenced in some biclustering algorithms [27] (particularly when considering simple linear regression relations). This is another point of focus that shall be further explored, especially when trying to infer regression meta-items. For now, if two meta-patterns with regression meta-items can be inferred out of one group of patterns, this will not be found by our algorithm, since it tries to generalize attributes using the simple linear regression model considering the whole set of patterns in a group. If we have knowledge of clusters of patterns that share behavior on a subset of their attributes (hence, biclusters), we would allow for more (finer-grained) meta-patterns to be found.

7. ACKNOWLEDGMENTS

This work is partially supported by FCT Fundação para a Ciência e a Tecnologia, under research project D2PM (PTDC/EIA-EIA/110074/2009).

8. REFERENCES

- [1] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Second Edition, San Francisco, CA: Morgan Kaufmann Publishers, 2006.
- [2] W. J. Frawley, G. Piatetsky-Shapiro and C. J. Matheus, "Knowledge Discovery in Databases: An Overview," *AI Magazine Volume 13 Number 3*, 1992.
- [3] U. Fayyad, G. Piatetsky-Shapiro and P. Smyth, "From Data Mining to Knowledge Discovery in Databases," *American Association for Artificial Intelligence*, 1996.
- [4] R. Agrawal, T. Imieliński and A. Swami, "Mining association rules between sets of items in large databases," in *SIGMOD '93 Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, New York, New York, USA, 1993.
- [5] M. J. Zaki, "A Journey in Pattern Mining," in *Journeys to Data Mining*, Springer, 2012, p. 235.
- [6] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," in *Very Large Data Bases (VLDB) Conference*, 1994.
- [7] D.-I. Lin and Z. M. Kedem, "Pincer-Search: A New Algorithm for Discovering the Maximum Frequent Set," in *In 6th Intl. Conf. Extending Database Technology*, 1997.
- [8] N. Pasquier, Y. Bastide, R. Taouil and L. Lakhal, "Efficient mining of association rules using closed itemset lattices," *Information Systems*, 1999.
- [9] M. J. Zaki and M. Ogihara, "Theoretical foundations of association rules," *Proc. SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery DMKD'98*, 1998.
- [10] J.-F. Boulicaut, A. Bykowski and C. Rigotti, "Approximation of Frequency Queries by Means of Free-Sets," in *Principles of Data Mining and Knowledge Discovery*, Springer Berlin Heidelberg, 2000, pp. 75-85.
- [11] A. Bykowski and C. Rigotti, "A Condensed Representation to Find Frequent Patterns," in *PODS '01 Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, New York, 2001.
- [12] T. Calders and B. Goethals, "Mining All Non-Derivable Frequent Itemsets," in *6th European Conference, PKDD 2002*, Helsinki, 2002.
- [13] J. Han, J. Wang, Y. Lu and P. Tzvetkov, "Mining Top-K Frequent Closed Patterns without Minimum Support," in *IEEE International Conference on Data Mining*, 2002.
- [14] F. Afrati, A. Gionis and H. Mannila, "Approximating a Collection of Frequent Sets," in *Knowledge Discovery and Data Mining*, Seattle, Washington, USA, 2004.
- [15] D. Xin, H. Cheng, X. Yan and J. Han, "Extracting Redundancy-Aware Top-K Patterns," in *Knowledge Discovery and Data Mining*, Philadelphia, Pennsylvania, USA, 2006.

- [16] Y. Kameya and T. Sato, "RP-growth: Top-k Mining of Relevant Patterns with Minimum Support Raising," in *Society for Industrial and Applied Mathematics*, Austin, Texas, USA, 2013.
- [17] A. Siebes, J. Vreeken and M. van Leeuwen, "Item Sets That Compress," in *SIAM Conference on Data Mining*, 2006.
- [18] D. Xin, J. Han, X. Yan and H. Cheng, "Mining Compressed Frequent-Pattern Sets," in *31st VLDB Conference*, Trondheim, Norway, 2005.
- [19] X. Yan, H. Cheng, J. Han and D. Xin, "Summarizing Itemset Patterns: A Profile-Based Approach," in *Knowledge Discovery and Data Mining*, Chicago, Illinois, USA, 2005.
- [20] G. Liu, H. Lu, W. Lou and J. X. Yu, "On Computing, Storing and Querying Frequent Patterns," in *SIGKDD*, Washington, DC, USA, 2003.
- [21] S. Dzeroski, "Inductive logic programming and knowledge discovery in databases," in *Advances in Knowledge Discovery and Data Mining*, Menlo Park, California, USA, AAAI Press, 1996.
- [22] L. De Raedt, "Inductive Logic Programming," 2010. [Online]. Available: <https://lirias.kuleuven.be/bitstream/123456789/301407/1/ilp4.pdf>. [Accessed 12 October 2013].
- [23] C. Stolle, A. Karwath and L. De Raedt, "Classic'cl: An integrated ILP system," in *Discovery Science 8th International Conference*, 2005.
- [24] C. Antunes, "Project D2PM," Project funded by FCT, under the grant PTDC/EIA-EIA/110074/2009, [Online]. Available: <https://sites.google.com/site/projectd2pm/>. [Accessed 21 03 2014].
- [25] M. J. Zaki, S. Parthasarathy, M. Ogihara and W. Li, "New Algorithms for Fast Discovery of Association Rules," University of Rochester, Rochester, NY, USA, 1997.
- [26] C. Borgelt, "Efficient Implementations of Apriori and Eclat," in *Workshop of Frequent Item Set Mining Implementations*, Melbourne, FL, USA, 2003.
- [27] S. C. Madeira and A. L. Oliveira, "Biclustering algorithms for biological data analysis: a survey," *Computational Biology and Bioinformatics, IEEE/ACM*, vol. 1, no. 1, pp. 24-45, 2004.