A HYBRID DATA MINING METHOD: EXPLORING SEQUENTIAL INDICATORS OVER ASSOCIATION RULES

Heitor M. Gomes[1], Deborah R. Carvalho[2]

[1] Universidade Tuiuti do Paraná (UTP). Curitiba, PR, Brazil. E-mail: heitor_murilo_gomes@yahoo.com.br

[2] Pontifícia Universidade Católica do Paraná (PUC-PR). Curitiba, PR, Brazil. E-mail: deborah@ipardes.pr.gov.br

Abstract. This work presents a hybrid data mining method which was designed to combine knowledge from an Association Rules discovery process along with chronological order evaluation. In order to evaluate Association Rules chronological order the Chrono Assoc algorithm was developed. To illustrate our methodology we present guidelines regarding database preparation, algorithm execution and post-processing. We also demonstrate performance experiments using synthetic data.

Keywords. Data Mining. Sequential data mining. Knowledge discovery in databases.

1. Introduction

Nowadays, companies have enough technology to store and maintain a great volume of data regarding their operations. As a consequence, there is an increasing demand for tools capable of discovering interesting knowledge within these databases. In order to achieve that, several data mining techniques were developed, such as the Apriori algorithm [1], which is capable of finding Association Rules among transactional data. Even though these algorithms represent an impressive evolution in how information is discovered in databases, very often they do not take into account an important dimension of transactional data, the time. On the other hand, algorithms which consider temporal aspects of data, such as GSP [3], ignore non-temporal attributes.

To illustrate the relevance of both aspects of data, consider the following example. Many patients whose treatment included an experimental drug have symptoms of the same disease in their medical records. This may indicate that the drug “caused” the disease, like a counter-effect. But what if 70% of the patients had shown the symptoms before they took the drug? So the pattern “drug then disease” is not feasible anymore. But, what if those 30%, where the pattern holds, were all women? Once again the pattern seem interesting, but it was necessary to include both interpretations, the chronological order evaluation and the non-temporal attribute 'gender' association with the drug and the disease.

In this work, we present a hybrid data mining method which is designed to allow the evaluation of rules in both aspects, i.e. temporal and non-temporal, simultaneously. We use the Apriori [1] algorithm to obtain Association Rules and then execute our algorithm - Chrono Assoc - to evaluate these rules from a chronological perspective. Chrono Assoc is based on the assertion that all the antecedents of a rule should occur before the consequent in the transactions (in this work we refer to it as “sequences”) where the rule holds (from an Association Rules perspective), otherwise it is not a cause-effect association and thus this rule is less interesting from a chronological standpoint. “Ordered events form sequences and the cause of an event always occurs before its result” [10].

The remainder of this paper is organized as follows. Section 2 presents an overview about temporal data mining and related work. Section 3 contains the terminology used in this work. Section 4 describes our data mining algorithm – Chrono Assoc. Section 5 is dedicated to presenting guidelines about performing our data mining method. Section 6 provides information regarding algorithm performance. The last section comprises conclusions and guidance for future work.

2. Temporal Data Mining

Data mining can be defined as: “the nontrivial extraction of implicit, previously unknown and potential useful information from data” [9], and temporal data mining can be seen as a subset of it, which focuses on “finding relations between sequences and sub-sequences of events” [4]. The task of discovering relations between sequences of events is denominated sequence mining and is a very common application of temporal data mining. A sequence can either contain nominal or continuous attributes, in the first case it is named temporal sequence and in the other, time series.

According to [5] the discovery of relations between sequences involves three steps:

- Representation and modeling of the data sequence in a suitable form;
- Definition of similarity measures between sequences;
- Application of models and representations to the actual mining problems.

2.1. Related work

The Apriori Algorithm [1] discovers Association Rules between sets of items in a transactional database. The algorithm iterates through the input database looking for items which occur within the same transaction to determine the “frequent itemsets”, which are the set of items that surpass the Minimum Support and Minimum Confidence thresholds given by the user (min_sup and min_conf respectively). The rules have to obey to the form: **antecedent(s) then consequent**. The algorithm comprehends two measure indicators for the rules, known as Support and Confidence. Support is the ratio of transactions that contains the set of all items that appear in the rule to the number of total transactions. Confidence is the ratio of transactions that contains the set of all items that appear in the rule to the number of transactions where only the set of items from the antecedent appear. Our data mining algorithm relies on the execution of Apriori [1] to obtain Association Rules, since we measure the chronological order of a rule based on the disposition of items within a rule. In our work we use the Apriori implementation derived from [6].

In [2] the problem of mining sequential patterns was introduced along with three algorithms to solve it. Despite the difference between the three approaches they aim for the same goal, discovering frequent sequences. To measure the discovered sequences a slightly different support definition to that in [1] was proposed, which reflects the fraction of transactions where the sequence holds, which means that the order of the itemsets in the transactions needs to be considered. At a later date the same authors [2] presented the GSP Algorithm in [3], which performed better than the previous algorithms and allow the user to specify constrains to the discovered sequences, such as the minimum and maximum gap between adjacent itemsets.

In [12] the SPADE (Sequential Pattern Discovery using Equivalence classes) algorithm for discovering frequent sequences was presented. It diverges from previous work as it does not inherit from Association Rules discovery techniques, instead the authors use a Lattice-based approach along with efficient search methods to discover and count the occurrence of sequences.

The works of [2], [3] and [12] discovered frequent sequences within transactions. These sequences are formed by events ordered in time axis, a sequence is frequent if it surpasses a minimum support threshold. Transactions where the sequence is found and its events happen obeying its order increase the support. Providing measures based on Association Rules chronological order can be viewed as a sequence mining task. Although our algorithm is not meant to discover sequences, it uses the Association Rules outputted by Apriori [1] and analyzes the order between the antecedent(s) and the consequent, e.g. it checks if all the antecedent(s) occurred before the consequent.

3. Terminology

The purpose of this section is to clarify the terms used in this work. It is especially important to emphasize what we mean when we use the terms ‘event’ and ‘sequence’, since we employ these terms differently from other works.

- **Static Attribute.** The static attribute contains no temporal information associated with it, for example: gender, city, etc. Where temporal information is implicit or not relevant for the problem scope, for example: birth date, can also be considered to be a static attribute.
- **Sequential Attribute.** This is where data has temporal information associated, for example: medical records, items bought, etc. We highlight that these attributes may not represent any complex multidimensional temporal information, since Chrono Assoc is only meant to work with one temporal dimension (it usually represents the event creation or something similar).
- **Event.** Instantiation of a Sequential Attribute. This means that an event is formed by a 2-tuple in the format $Evt = (V, t)$, where V is the event's value and t represents its timestamp. Since we only care about the chronological order, in the rest of this work when referring to an event the timestamp information is neglected.
- **Sequence.** This term has a different meaning in our work than it has in other works. Here it does not stands for the discovered patterns (named as ‘sequences’). In our scope a sequence represents all the events of a registry. It can be seen as a union of all transactions of the same registry in crescent chronological order. Basically, when

referring to the algorithm perspective a sequence in our work represents the same as a transaction in other works.

- **Sequential Database.** A database in which transactions happen to be sequences (contain a sequential attribute).

4. The *Chrono Assoc* Algorithm

Chrono Assoc is meant to explore the sequential patterns behind standard Association Rules. The focus is on "validating" the cause-effect that a rule indicates. The rules obtained after the execution of Apriori [1] are used as input for Chrono Assoc, which check their chronological order fidelity to cause-effect assertive (e.g. antecedent(s) before consequent). This check yields the measures of Chronological Support and Chronological Confidence. Chronological Support is the fraction of the database where the antecedent(s) and consequent of a rule occurs together respecting the order "antecedent(s) then consequent" relative to the total number of sequences. Chronological Confidence is very similar, although it is relative to the fraction of the database where the components of the rule existed concomitantly.

Formally: Let D be a sequential database, T be the set of Association Rules obtained from D, D_t a subset of D which contains the transactions where the Association Rule t exists and D_{tk} a subset of D_t which contains the transactions where the chronological order (antecedent then consequent) holds. The Chronological Support and Confidence are calculated by:

$$1. ChronoSupport(t) = \frac{|D_{tk}|}{|D|}$$

$$2. ChronoConfidence(t) = \frac{|D_{tk}|}{|D_t|}$$

The order between the antecedent(s) and consequent is considered valid even for sequences where other events occur in the middle of them. For example: consider the rule "R = X, Y → Z (100%, 100%)", where the two percentages at the end represent its Support and Consequent respectively, and Table 1 represents the sequences which produced rule R. The order will be valid for sequences: 1, 2 and 3. Note that the rule R has Chronological Support = 3 / 5 (60%) and Chronological Confidence = 3 / 5 (60%).

Table 1. Sequential database example

ID	Sequence
1	X K B Y Z
2	B X Y Z
3	X C Y Z
4	Z Y B X
5	X Z Y B

This example can be used to highlight one bias of our approach, which happens to affect sequence mining algorithms as well. It is the possibility of associating events with a great distance in the time axis between them. As we ignore the timestamp of the events and only look for the chronological order, we assume that the input database period is meaningful for the given problem, in other words, that it has been pre-processed adequately and there are no great gaps between events. Selecting the best distance between events is problem-dependent and thus is not covered by this work.

From a method perspective, our approach is very straightforward, as it simply checks the order of the events within a sequence based on Association Rules. However, technically it is hard to do that in a scalable way. If we use a naive approach such as keeping both inputs (rules and the database) in a secondary storage (i.e. hard drive) then perform $M * N$ scan, where N stands for the rules and M for the sequences, for larger inputs it would take too long to finish, as it would demand too much access to slow storage devices. On the other hand, it is not possible to load both files (rules and sequences) in the main memory, usually not even one of them entirely, as they are usually large and might not fit. There is also another problem: the “order check” implies looking into every sequence and scanning it for all the antecedents and the consequent of a rule, which represents many comparisons of strings (the $M * N$ scan).

To cope with these drawbacks we elaborated some strategies to decrease the access to secondary storage and reduce the time spent in checking sequences. The first important performance issue is connected to the way that sequences are represented. We choose to represent sequences as ordered double-linked lists which expressively lower the time when checking for the occurrence of an event within the sequence. Furthermore, this representation of a sequence has its events summarized, which allows less time to compare events. Section 4.1 details the sequence data structure. Another crucial matter is an efficient “chronological order check” function, since it is invoked for every rule against every sequence. Also, to allow constant time comparison between events, they are mapped to integers. New versions of the input files (rules and database) are created using the events' codes (integers) to represent them. It is necessary to keep the association between events and their codes and to do that a dictionary file is also created

To diminish the access to the secondary storage the input database is split into chunks of sequences which fit in the main memory. To determine how many parts the database must be split into, the algorithm inspects the main memory for free space. It does not consider other processes that will eventually consume memory; it assumes that the current memory state will remain as it is.

The algorithm execution occurs as described in Figure 1. The ellipses numbered from 1 to 6, in Figure 1, represent the order in which the operations take place. Operations 1 and 2 occur concurrently. Operation 3 uses the output from 2. Operations 4 and 5 are iterative, i.e., they repeat until all the sequences have been loaded and checked against every rule, and the last one is 6, which creates the output file.

In Algorithm 1 the basic functionality of Chrono Assoc is shown and in the subsequent sections the most relevant parts – Data structures, Chronological Support Counting, Output, etc. are discussed in more detail, along with explanations of the types of rules Chrono Assoc is able to outline.

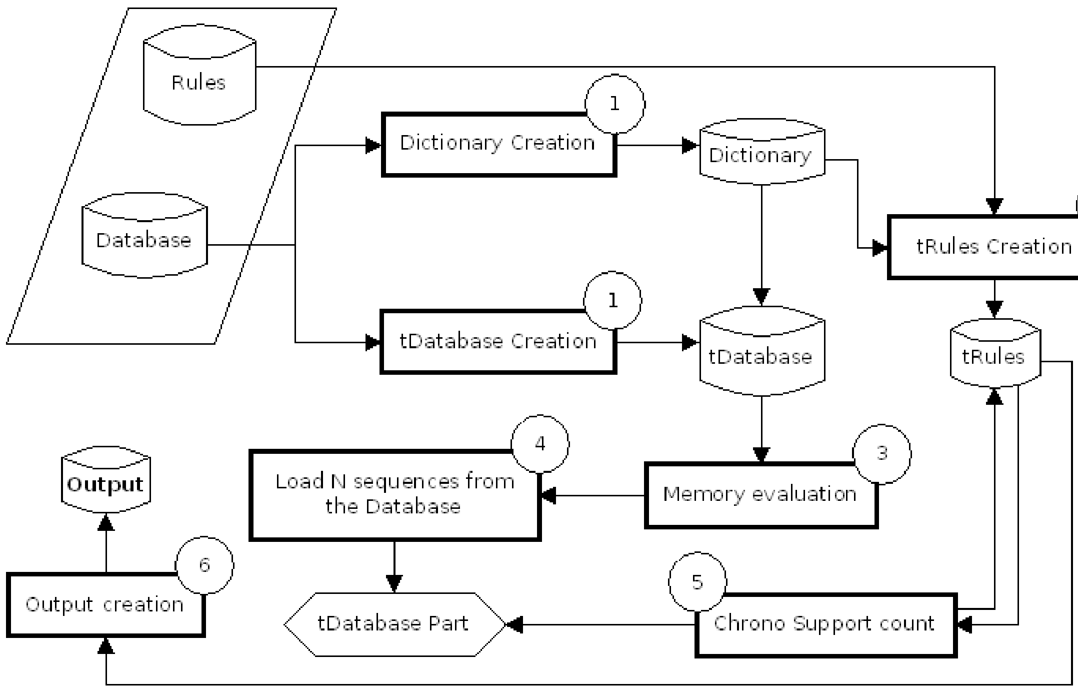


Figure 1. Chrono Assoc execution[1]

Algorithm 1. ChronoAssoc (database, rules)

```
1.  /* until the dictionary is removed from memory by
2.     Create_tRules it exists in both (memory and sec. storage) */
3.  dictionary.createFile()
4.  Create_tDatabase_&_dictionary ( database, dictionary)
5.  Create_tRules(rules, dictionary)
6.  /* set values for globals such as slice, part, etc. */
7.  checkMemory()
8.  from = 0
9.  to = slice - 1
10.  For (i = 0 ; i < parts ; i = i + 1)
11.      database.loadToMemory(from, to)
12.      Foreach rule In Rules
13.         database.Count_#ChronoSupport (rule)
14.      from = from + slice
15.      to = to + slice
16.  RulesOutput( rules, database.nSequences, dictionary )
```

1 tDatabase and tRules stands for translated Database and translated rules, respectively.

4.1. Data structures, file translation and the in-memory database

The algorithm foundations are the data structures used to represent the rules and sequences, the creation of translated versions of both input files (database and rules) and the manner in which it is able to dynamically load sequences to main memory.

The data structures are utterly important since their design influences how the rest of the algorithm performs. The two main data structures are Sequences and Rules followed by Events and REvents.

Sequences encompasses the number of different Events which it contains (size) and two references to an Event's structure. The Event's structure is instantiated as an ordered double-linked list. The references in a Sequence are for the beginning and end of its Event list. The Event contains three attributes (apart from its references to previous and next member, since it is a double-linked list), the item which it represents, and its first and last occurrence within the sequence. For example, the sequence "X-Y-A-Y-X-X-A" will be represented as shown in Figure 2. Note that the item is in reality represented as an integer value, although for illustrative purposes it is shown as a character in this example.

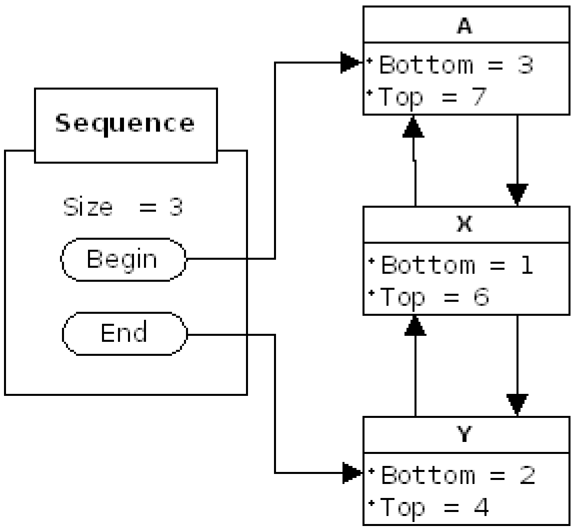


Figure 2. Representation of the sequence "X-Y-A-Y-X-X-A"

The Rules structure contains five attributes, the number of events within the rule, rule's support, rule's confidence, chronological support counter and a reference to the structure that holds the rule's components. The support and confidence are set by Apriori [1] and both are represented as percentages. The chronological support counter is the number of sequences in which the order antecedent(s) then consequent holds for this rule, and the last one is a reference to a REvent structure. The latest happens to be a linked list with the first member always being the rule's consequent followed by its antecedents. For example, consider the sample rule we used to explain the chronological valid rules (section 5.):

"R = X, Y → Z (100%, 100%)", which respected the chronological order in three sequences (see Table 1) and which is represented as shown in Figure 3.

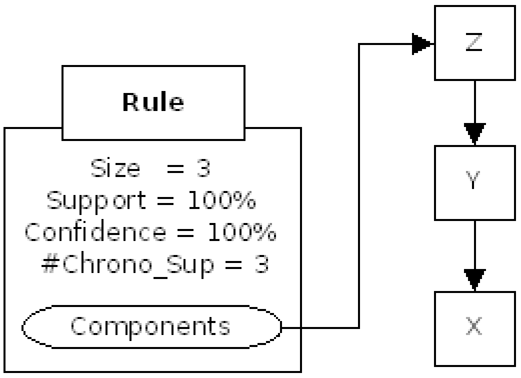


Figure 3. Representation of rule "R = X, Y → Z"

The second concept is how the translated files are created and their purpose. To perform the translation a dictionary structure is used which is kept in secondary storage when not in use, and loaded to memory as a map, which is an associative container formed by key value and mapped value. The key value is unique and holds the string version of the Event value and the mapped value is the code used to represent the Event. For more information regarding the maps container refer to [11].

The dictionary is used to create the tDatabase and the tRules files. These are binary representations based on the Sequences and Rules data structures mentioned earlier in this section. Both contain the same data as the original files but are smaller and thus allow efficient data loads to memory (tDatabase) and reliable temporary data storage for retrieval and update (tRules). Algorithms 2 and 3 express how the tDatabase, tRules and dictionary files are created.

Algorithm 2. Create_tDatabase_&_dictionary (database, dictionary)

```
1.  tDatabase.createFile()
2.  Foreach seq In database
3.    Foreach evt In seq
4.      rawSequence.push ( dictionary.insert( evt ) )
5.    tDatabase.insert ( rawSequence )
```

Algorithm 3. Create_tRules(rules, dictionary)

```
1. tRules.createFile()
2. Foreach rule In rules
3.   Foreach comp In rule
4.     rawRule.push ( dictionary.getCode(comp) )
5.     tRule.insert( rule.sup, rule.conf, rawRule)
6.   /* dictionary returns to memory when
7.      it is time to create the Output */
8. dictionary.removeFromMemory()
```

The tRules file facilitates the retrieval and update of a rule's chronological support counter. It is necessary to store the chronological support counter within the rule structure since rules are not retained in main memory.

Efficient data load to memory refers to the last foundation mentioned at the beginning of this section. To overcome the problem of slow storage access, the database is loaded to main memory while the algorithm is performing the chronological support counting, but as the database can be very large (and available memory very short) there is a need to split it in N parts which alone fit in the available memory. As the sequences within tDatabase are represented in a very compressed format the number of sequences which can be loaded at the same time is improved. There are eight important terms regarding loading part of the tDatabase in the main memory:

1. **Available Memory.** Physical memory available
2. **Maximum Sequence Size.** The maximum sequence size found in the database.
3. **Slice.** Number of sequences that can be loaded at once in memory.
4. **Portion.** The part of the database loaded to memory. It is represented as a vector [11] of sequences, whose size is determined by Slice.
5. **Sequence magnitude.** Maximum sequence size plus sequence's metadata size.

6. **Part.** Number of parts in which the database must be split.
7. **Total Number of Sequences.** Total number of sequences within the database.
8. **Database Estimated Size.** The total number of sequences multiplied by the sequence magnitude.

The main issue while dynamically loading portions of the database to memory is that sequences do not have the same size (number of events). Therefore it is possible that the sequences being loaded surpass the available space in main memory. One way to prevent this from occurring is by precisely sizing every sequence before actually loading it, although it is not actually feasible to do so, since it overloads the "load to memory" function and compromises the execution time with no great rewards. To avoid trying to load more than the memory is capable of sustaining, it is assumed that every sequence has the same size as the sequence with the greatest size (Maximum Sequence Size).

Another issue is that a sequence demands more space to be kept in memory than in second storage because while in memory it is necessary to maintain references to other sequences along with other metadata (i.e. reference to its Events). Formula 3 determines the sequence magnitude and Formula 4 guarantees that there will be enough space for the database portion to be loaded. Even if it ends up not taking complete advantage of the available memory, it is better than eventually causing the system to enter a "thrashing" state [7].

$$3. Sequence_{magnitude} = sizeof(Sequence) + Sequence_{\max Size} * sizeof(Event)$$

$$4. Database_{estimatedSize} = Database_{nSequences} * Sequence_{magnitude}$$

Algorithm 4. Demonstrates how the memory is evaluated in order to allocate part of the database.

Algorithm 4. MemoryEvaluation ()

```
1.  /* currently available main memory, it is a system call */
2.  availableMemo = memoryStatus()
3.  /* total number of Sequences within database */
4.  nSequences = database.getNSequences()
5.  /* the largest sequences size */
6.  maxSequenceSize = database.getMaxSequenceSize()
7.  /* hypothetical size in bytes of any Sequence. */
8.  sequenceMagnitude = sizeof(Sequence) + maxSequenceSize * sizeof(Event)
9.  /* hypothetical size in bytes of the database (while in memory) */
10. dbMemo = nSequences * sequenceMagnitude
11. /* number of parts the database must be split to fit in memory, default = 1 */
12. parts = 1;
13. If(availableMemo < dbMemo)
14.    parts = dbMemo % availableMemo > 0 ? (dbMemo/availableMemo)+1 : dbMemo/availableMemo;
15. /* maximum number of sequences in memory at once */
16. slice = nSequences % parts > 0 ? (nSequences / parts)+1 : nSequences/parts;
```

Notice in Algorithm 4 how the Sequence structure benefits the search for an event within a sequence, setting where it starts searching (lines 12 from begin or line 18 from end) or ignoring sequences without even needing to iterate over its events (line 7).

4.2. Chronological Support Counting

Chronological Support Counting is the core function of the Chrono Assoc algorithm and thus the most critical one. All the rules are checked against every sequence, looking forward to increase its Chronological Support Counter (CSC). Also, the CSC is used to calculate the Chronological Support and Confidence. The Chrono Support Counting function involves another function which is responsible for verifying the presence (or absence) of an Event within a sequence. Both functions are described in detail in Algorithms 5 and 6.

Algorithm 5. FindEvent (seq, item)

```
/* set the distance between item
   and the begin/end of the sequence */
distToBegin = item - seq.begin
distToEnd = seq.end - item
  /* if the distance to any is negative, the item
  is not compreended within the sequence. */
If(distToBegin < 0 or distToEnd < 0)
  return NULL
 /* start searching from the beginning
    since item is closer to it or Else if
    its closer to the end */
  If(distToBeg < distToEnd)
    it = seq.begin
    While(it <> NULL)
      if(it.item == item)
        return it
      it = it.next
  Else
    it = seq.end
    While(it <> NULL)
      If(it.item == item)
        return it
      it = it.prev
  return NULL
```

Algorithm 6. Count_#ChronoSupport(rule)

```
#ChronoSup = rule.getChronoSup()
Foreach seq In inMemoryDB
   count = true;
   consequent = FindEvent(seq, rule.item)
   /* consequent does not even exist in this sequence */
   If(consequent == NULL)
      continue //go to the next sequence
   comp = rule[2] //skip the first as its the consequent
   While ( i <= rule.size() )
      antecedent = FindEvent(seq, comp)
      /* antecedent does not even exists in this sequence or
         antecedent occurs after the consequent */
      If(antecedent == NULL or antecedent.bottom > consequent.top)
         count = false //prevents from counting it
         break //quit checking this sequence
      comp = rule[i]
      i = i + 1
   If( count == true )
      #ChronoSup = #ChronoSup + 1
   return #ChronoSup
```

4.3. Rules output creation

After the algorithm has completed counting chronological supports for each rule, it is time to generate the Output file, which contains all the rules with its support and confidence plus the Chronological Support and Confidence. Section 4 defines how Chronological Support and Confidence are calculated. All the necessary information to calculate both chronological indicators is encompassed within the Rule structure itself, except the total number of sequences. Chronological Support is given in Formula 5.

$$5. ChronoSupport(R) = \frac{R_{nChronoSupport}}{Database_{nSequences}}$$

The Chronological Confidence depends on the number of times where the set of antecedents and the consequent occurred together (in any order). This value (nAntecedentConsequent or simply “nAC”) can be inferred using the rules' support along with the total number of sequences, as shown in Formula 6. However, it is possible that the result obtained by Formula 6 is not accurate, since its accuracy depends on how precise the Support was set by Apriori [1]. In order to decrease this uncertainty impact, we propose a workaround based on floor() and ceiling() functions (see line 5 from Algorithm 7).

$$5.nAC(R) = \frac{R_{Support} * Database_{nSequences}}{100}$$

It is important to emphasize that using this approach to calculate the occurrence of antecedents and consequent together, allows Chrono Assoc to avoid checking all the sequences exhaustively. Algorithm 7 presents the function which calculates and outputs rules.

Algorithm 7. RulesOutput (rules, nSequences, dictionary)

```
1.  Output.create()
2.  dictionary.loadToMemory()
3.  Foreach rule In rules
4.     nAC = rule.support * nSequences
5.     EnAC = nAC % (int)nAC > 0.5 ? ceil(nAC) : floor(nAC)
6.     csup = 100*(rule.ncsup/nSequences)
7.     cconf = 100*(rule.ncsup/EnAC)
8.     Foreach comp In rules.components
9.        str_comps.push( dictionary.getString (comp) )
10.    Output.write(str_comps, rule.support, rule.confidence, csup, cconf)
```

4.4. Types of rules

We identify three different kinds of rules which Chrono Assoc is capable of identifying among Association Rules. They are:

- **Static rules.** The consequent is a static attribute. These rules always have Chronological Support and Confidence equals zero. Example: Male → Age_60
- **Temporal rules.** Both consequent and antecedent(s) are instances of the sequential attribute. The Chronological Support and Confidence is absolutely relevant for these rules. Example: Drug_X → Disease_Y
- **Misc rules.** The consequent and at least one of the antecedents happen to be sequential attributes. These are the most interesting types as they allow analysis from temporal and non-temporal aspects at the same time. Example:
  Drug_X , Woman → Disease_Y

5. Method guidelines

This section embraces the experiment guidelines which can be followed when applying our method to real data. The experiment is guided by the KDD (Knowledge Discovery in Databases) methodology [8], which is divided into three main phases: pre-processing, data mining and post processing. We have split these phases in subdivisions when necessary.

- **Pre-processing.** This phase encompasses data preparation for the subsequent data mining algorithms execution. It is composed of database period selection, unrepresentative attributes removal, along with data transformation (classification, stratification, or other operations that can be applied to continuous attributes).
- **Data mining.** The execution of Apriori [1] followed by Chrono Assoc. Some important factors must be considered while defining the minimum support and confidence for Apriori [1].
- **Post Processing.** A pruning of temporal and misc rules can be executed that does not hold chronologically.

5.1. Pre-processing

The first step to prepare the database is to select its period, in other words, choose which time frame is going to be used. It is a crucial step since it can jeopardize the whole process. Chrono Assoc does not take into account the exact time when an Event occurs, it only checks the event's order, this can cause rules to have meaningful Chronological support and confidence, despite the fact that its antecedents happened many years before the consequent.

The segregation between sequential and static attributes is not recognized by Chrono Assoc, so it is mandatory to arrange the sequences leaving all the static attributes at the left and the sequential attributes at the right. Furthermore, the latest must be set in ascendant chronological order. Chrono Assoc completely ignores the semantic of sequential and static attributes, in reality, it only checks the order of the items[2] within the sequence. This characteristic affects the way rules are interpreted; see section 4.4 for considerations about the types of rules.

2 In this context "item" could either be a static attribute or an event.

The second step is to stratify continuous attributes so they are more likely to appear within patterns. Thus they effectively become more representative. For example, if there is an attribute "age" with a sparse distribution, its values may be distributed over age groups to enhance its representative.

5.2. Data mining

This phase starts off with the execution of Apriori [1] in order to extract the Association Rules of the database. It is important to be attentive about the minimum support and minimum confidence threshold, since depending on the situation it may be more interesting to explore the sequential aspects of rules with very low support and confidences, which may not be true in other cases. Also, very low minimum support and confidence tends to yield too many rules, which can take too long for Chrono Assoc to process, especially if the database is also great in volume.

5.3. Post processing

After the execution of Chrono Assoc a filter can be applied to remove sequential and misc rules which have zero Chronological Support. These kinds of rules are usually not interesting because they represent situations where the consequent happened before the antecedent all the time, so it did not characterize a cause-effect, since the antecedent could not have influenced the consequent. We have not included this pruning feature build in Chrono Assoc, since there is a possibility that users want to evaluate rules with high support and confidence that do not hold chronologically.

6. Performance Evaluation

6.1. Synthetic databases

In order to provide performance evaluation we have generated four databases with different sequences quantities and sizes. All four have the same quantity of different events (900 different events). Also, we use three sets of rules for each database to perform the tests. Table 2 presents the synthetic databases while Table 3 presents the rules obtained for each database. Note that it is completely wrong to use rules extracted from one database along with another as input to Chrono Assoc, especially when using real life data. Even using synthetic data the execution time may be influenced by rules constantly ignoring sequences (see algorithm 5, line 7) since their events codes may be in different magnitudes (more or less different events between different databases).

Table 2. Synthetic databases

ID	Total Number of Sequences	Average Sequence Size	Original File Size (KBytes)	tDatabase File Size (KBytes)
1	500	217	334	102
2	1796	202	1.335	459
3	24246	220	18.015	6.242
4	73636	213	54.710	16.616

Table 3. Rules obtained through synthetic databases

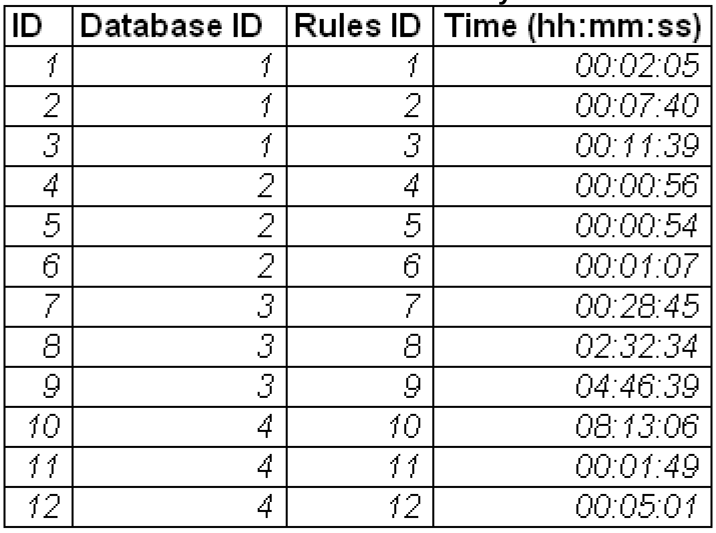
ID	Total Number of Rules	Average Rule Size	Original File Size (KBytes)	tDatabase File Size (KBytes)	Originary from Database ID
1	823935	7	89.252	22.108	1
2	1647870	5	178.540	44.069	1
3	3086920	5	215.869	65.909	1
4	4763	6	515	150	2
5	13910	6	1902	725	2
6	15839	5	2032	922	2
7	324939	6	234.869	70.576	3
8	814935	6	86.963	20.387	3
9	1545993	5	156.330	40.069	3
10	822298	5	88.963	21.997	4
11	2193	5	226	54	4
12	7539	6	703	193	4

6.2. Results and discussion

Chrono Assoc was executed for the databases and rules from Tables 2 and 3. The results are shown in Table 4.

It is reasonable to state that the execution time is more affected by the number of sequences than the number of rules by comparing tests 3 and 10. Note that we only used databases with large sequences; it is noticeable by analyzing column "Average Sequence Size" from Table 2. So, improvements to the algorithm can be either an even better representation of the sequence structure or start checking multiple rules concurrently, since one rule does not depend on another rule to execute and the part of the database loaded in memory is never updated (only when it is released to give place to another part).

Table 4. Performance analysis



ID	Database ID	Rules ID	Time (hh:mm:ss)
1	1	1	00:02:05
2	1	2	00:07:40
3	1	3	00:11:39
4	2	4	00:00:56
5	2	5	00:00:54
6	2	6	00:01:07
7	3	7	00:28:45
8	3	8	02:32:34
9	3	9	04:46:39
10	4	10	08:13:06
11	4	11	00:01:49
12	4	12	00:05:01

7. Conclusions

Throughout this work we have presented a hybrid data mining method which allows Association Rules evaluation from two perspectives at the same time, temporal and non-temporal. The Chrono Assoc algorithm was introduced and performed with fair performance for inputs of reasonable sizes. Despite the interesting capability of temporal and non-temporal evaluation that Chrono Assoc allows, it is straightforward to extend the algorithm to permit using it as a "post processing" algorithm, eliminating sequential and misc rules which have Chronological Support and Confidence equals zero or below a threshold informed by the user.

In terms of future work we are looking forward to research about time-distance constraints and automatic definition of those constraints. Constraints are important because they allow the use of the whole database instead of only a period. Up to now it has been necessary to define periods in order to avoid associating Events with great time gaps (as discussed in section 5.1). It is also possible to increase the algorithm's performance by exploring operations which can be paralleled. Lastly, and most importantly, we wish to apply our method to real world data and provide qualitative evaluations from specialists.

References

[1] R. Agrawal, T. Imieliński, A. Swami. (1993) Mining association rules between sets of items in large databases, Proc. ACM SIGMOD Int'l Conf. Management of Data, vol. 22, pp. 207-216.

[2] R. Agrawal, R. Srikant. (1995) Mining Sequential Patterns, Proc. 11th Int'l Conf. Data Eng., P.S. Yu and A.S.P. Chen, eds., pp. 3-14.

[3] R. Agrawal, R. Srikant. (1996) Mining Sequential Patterns: Generalizations and Performance Improvement, Proc. 5th EDBT, pp. 3-17.

[4] C. Antunes. (2007) Temporal Pattern Mining Using a Time Ontology, New Trends in Artificial Intelligence, pp. 23-34.

[5] C. Antunes, A. L. Oliveira. (2001) Temporal Data Mining: an overview. In Workshop on Temporal Data Mining (KDD2001), San Francisco.

[6] C. Borgelt. (2010) Apriori-Association Rule Induction, http://www.borgelt.net/aprio1ri.html, December.

[7] P. J. Denning. (1968) Thrashing: Its causes and prevention. In AFIPS Conference Proceedings, Vol. 33, FJCC, pp. 915-922

[8] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, R. Uthutusamy. (1996) Advances in Knowledge Discovery and Data Mining USA: American Association for artificial intelligence.

[9] G. W. Frawley, C. M. Piatetsky-Shapiro. (1992) Knowledge Discovery in databases: An overview. AI Magazine, vol. 13, n. 3, pp. 57 – 70.

[10] J. F. Roddick and M. Spiliopoulou. (1999) A bibliography of temporal, spatial and spatiotemporal data mining research, SIGKDD Explorations, 1:34-38

[11] B. Stroustrup. (1997) The C++ Programming Language. 3rd ed. Massachutts, USA, Addison-Wesley.

[12] M. J. Zaki (2001). SPADE: An Efficient Algorithm for Mining Frequent Sequences, Machine Learning, vol. 42, pp. 31-60.