March 2014

Universiteit Leiden
Opleiding Informatica

Time-interval sequential patterns

in insurance data

Piet van Hekke

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Time-interval sequential patterns in insurance data

Piet van Hekke
pvanhekke@gmail.com
LIACS, Leiden University

Supervisors
Dr. W.A. Kosters (LIACS)
Dr. S.G.R. Nijssen (LIACS)
Drs. R. Guitink (Quinity)
M. van Zetten MSc (Quinity)

March 28, 2014

Abstract

Insurance data provides a rich environment for data mining, with large real world data sets. The master thesis is the result of a study on customer behaviour in a real world insurance data set, focussed on associations between products, and sequential product purchase patterns. We tried to get insight into the data set and to extract rules describing the characteristics of the insurance products, for which we used the Apriori and Generalized Sequential Pattern algorithms. As an extension to the purchase orders we took the interval between the purchase of multiple products into account. To do so, we defined two extensions. First, we used fixed length intervals as extra constraints to the patterns. Second, we used a variety of weighting functions to calculate a weighting of each customer based on the length of the purchase intervals.

Contents

1 Introduction

Data mining is a very broad field of study, with a lot of possible applications in a large number of areas. The recent renewed interest in "Big Data" — which is actually just data mining and data management of very large data sets and has been around for years — has made companies look into their production data and archives with renewed interest. One field for which data mining has a lot to offer is the insurance business. The data on customers, policies and claims is the very heart of an insurance company. Digging up new information, such as customer behaviour, claim frequency and fraud patterns, from this data, can be promising for marketing purposes.

This thesis aims to be a small part of this field of study. Given a real world data set with insurance data, our goal is to find and interpret patterns in the data, with a focus on customer-product relations. To do so, we will search for association rules and sequential patterns in the purchase history of insurance products. With these results, we will look into ways to integrate the time aspect into the evaluation, by introducing time intervals and support weighting.

This report is the final result of a Master Thesis project for the Master Computer Science at Leiden University. The project is done as an internship with Quinity, a Utrecht based manufacturer of insurance administration systems. The data set is kindly provided by one of Quinity's customers.

The remainder of this thesis is structured as follows. Section 2 gives a short overview of the ways data mining can be used in insurance. Section 3 gives the formal definitions of associations, patterns and transactions, defines quality evaluation metrics, explains the theory and pseudocode of the algorithms used and introduces some extensions of the algorithms to handle time intervals. Section 4 starts with an introduction to the data set and discusses the results of the application of the algorithms on the data set. Finally, Section 5 will draw some conclusions and give directions for further research.

2 Insurance data mining

A lot of research has been done on the application of data mining techniques in insurance. Insurance data provides a rich environment for data mining and has many real world applications. For example, insurance data can be used in market basket analysis, fraud detection, customer clustering and insurance risk modelling. This section will give a quick overview of the most important fields of study in insurance data mining.

2.1 Claim prediction

For insurance firms, it is very important to have an accurate prediction of the claim rate on its policies, in order to determine reasonable prices. It is equally important to have an estimate of the risk on new policies. These risks can be modelled using, for example, classification algorithms or regression (see [8] for more examples).

Classification algorithms are used to divide the data instances, presumably customers or policies, into two classes, positive or negative, depending on profit or loss for the instance. Input can be everything that is known about the customers. Many classification algorithms exist, using various approaches. Decision trees use a tree structure to classify instances, where each node tests for a specific attribute. Naïve Bayes classifiers calculate a claim profit probability assuming independent variables, using Bayes' theorem. Clustering algorithms group similar instances together, so the risk on new instances can be predicted from its neighbourhood, for example in [21]. Neural Networks use a self adapting network to classify instances into groups. The quality of this classification depends on the training of the network. There are various types of neural networks, such as back propagation networks and Self-Organizing Maps, that are useful for claim rate prediction. See [22] for examples.

Regression algorithms aim to estimate the value of a continuous output variable based on a number of independent input variables by using a function to map the input variables to an output. The function can be a simple linear function for linear regression or a more sophisticated one, for example a logistic function. The weights of the function are obtained using a training set. This resulting function can be used to make accurate predictions of claim rates or claim cost.

2.2 Fraud detection

Detecting fraud in insurance data is a challenging topic. Fraud patterns are rare and they are not easy to find. A number of approaches has been suggested, mainly focussing on the development of models to recognize possibly fraudulent claims, each having advantages and disadvantages. The predictive models of logistic regression, a C4.5 decision tree, k-nearest neighbour clustering, a Bayesian learning neural network, a support vector machine, Naïve Bayes and tree augmented Naïve Bayes are discussed and compared in [19]. Brockett [6] suggests Kohonen's Self-Organizing Feature Map to detect automobile claim fraud and automated fraud detection with Bayesian learning neural networks is proposed in [18].

2.3 Marketing

Data mining techniques are also being used extensively for marketing purposes, for example to find patterns in customer purchases, or to identify target groups for ad campaigns. In the insurance industry, data mining can help companies to respond to customer behaviour, acquire new customers and develop new insurance products, as suggested in [9]. To model sales return, customer segmentation has been used, for example in [3, 17]. Segmentation tries to divide the customer database into groups of somehow related customers, which can then be addressed with focussed ad campaigns. This can be done based on predefined criteria or by clustering based on similarities and analysing the resulting segments afterwards [11]. Commonly used for market segmentation are clustering algorithms such as K-means [10].

Another much applied technique for marketing is Market Basket Analysis. This technique looks into sets of products that are purchased together and tries to find relations. These relations can be frequent itemsets (products that are often sold together), association rules (relations describing products that often lead to the purchase of other products) or sequential patterns, describing frequently appearing orders in which products are sold. Sequential patterns can be expanded with time interval information.

In this thesis we will focus on the application of association rule mining on the purchase of insurance products. We want to find a way to describe temporal patterns in purchase behavior. It is straightforward to find associations between products, but we get much more information from a sequential order between these products, i.e., knowing that a certain product is generally bought after another certain product. It would be even more useful to have some quantitative expression of the time between these purchases. The goal of this thesis is to find a way to derive these quantitative temporal patterns from the dataset.

3 Market Basket Analysis in insurance data

The most general and most common systematical approach to Market Basket Analysis is the search for association rules, which describe strong relations between products, hidden in large datasets. A (fictional) example of an association rule is

$$\langle \text{iPad} \Rightarrow \text{iPad case} \rangle$$

This rule suggest a relationship between the sale of iPads and the sale of iPad cases, because many customers who buy an iPad will also buy a protective case. It states, with a certain probability, that the action of buying an iPad implicates the buying of a case. Marketing divisions can use this kind of rules for the placement of products in stores, or for targeted marketing campaigns. Note that this relation is unidirectional only, the example above does not imply that buying an iPad case leads to buying an iPad. However, this connection might even be more likely. Also note that the order of the purchases is not an issue here.

Although association rules can be very useful, they contain limited information on the relation between items. To increase their usefulness, associations can be expanded into sequential patterns. Sequential patterns describe the chronological order between items, for example

$$\langle (\text{iPad}), (\text{iPad case}), (\text{iPad charger}) \rangle$$

This pattern states that it is a common purchase order to first buy an iPad, then buy a case for it, and after some time buy a new charger. With this knowledge, a marketing division can, for example, present iPad buyers with a combination discount on cases and chargers, thereby maximizing the number of customers who behave like this pattern.

The process of finding association rules and sequential patterns in a dataset is commonly done in two steps [1]:

1. **Frequent itemset generation**, aiming at finding all itemsets (groups of items) that are found frequent enough to satisfy a minimum support threshold.

2. **Rule or pattern generation**, which aims at extracting from the frequent itemsets all rules or patterns with a confidence that is above a confidence threshold.

In this chapter, the algorithms to generate associations and patterns are discussed in Section 3.3 and 3.4, but before that we will first discuss the formal definitions and evaluation metrics in Section 3.1 and 3.2.

3.1 Definitions

Both association rules and sequential patterns are defined over items and transactions. Let $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$ be a set of items, ordered by $i_1 < i_2 < \ldots < i_n$. An *itemset* is a non-empty subset of $\mathcal{I}$. Let $\mathcal{D} = \{t_1, t_2, \ldots, t_m\}$ be a set of transactions. A transaction t contains a unique identifier, a number of items and one or more transaction dates.

A rule is an implication of the form $X \Rightarrow Y$, with itemsets $X, Y \subseteq \mathcal{I}$ and $X \cap Y = \emptyset$. They are defined over $\mathcal{I}$. Each rule consists of an antecedent X and a

consequent Y, and reads as "if X is true, then Y must also be true". In Market Basket Analysis, a proposition being true usually means the purchase of one or more products. Both X and Y can consist of multiple items. If $\mathcal{I}$ consists of tens or hundreds of items, a large number of rules can be constructed. To identify interesting or strong rules, we can calculate measurements of the quality of rules, as defined in Section 3.2 [1].

We define a pattern or sequence $s = (s_1, s_2, \ldots, s_n)$ as an ordered set of itemsets. Each itemset s_j, also called an *element*, contains a non-zero number of items. An item can occur only once in an element, but can occur multiple times in different elements. A sequence reads as "first the items of s_1 are bought, then the items of s_2, then the items of s_3, etc." In the initial setup, sequential patterns only say something about the order of purchases. However, it is straightforward to come up with all kinds of time constraints, such as a minimum or maximum gap between transactions [2].

3.2 Quality evaluation

With most association rule algorithms, it is quite easy to generate a large number of valid associations or patterns. Therefore we need a method to distinguish the useful rules from useless results. To do this in a measurable way, a set of metrics is necessary. In this section we will discuss *support*, *confidence* and *lift* as quantitative measurements of the quality of the results. We define [1]:

$$\text{Support}: \quad s(X \Rightarrow Y) = \frac{\sigma(X \cup Y)}{N}$$

$$\text{Frequency count}: \quad \sigma(Y) = \sum_{t \in D, Y \subseteq t} 1$$

In these definitions, $\sigma(Y)$ means the frequency count of Y. The frequency count is the number of transactions $t \in D$ where $Y \subseteq t$, i.e., where all items of Y appear in t and N is the total number of transactions in the dataset. Thus, $s(X \Rightarrow Y)$ is defined as the proportion of transactions in D which contain $X \cup Y$. Note that $s(X \Rightarrow Y) = s(Y \Rightarrow X)$, so for the support metric, it is sufficient to see a rule as a union of two itemsets. We define:

$$\text{Confidence}: \quad c(X \Rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}$$

The confidence $c(X \Rightarrow Y)$ is defined as the frequency count of $X \cup Y$ divided by the frequency count of X, thereby determining how frequently items in Y appear in transactions which contain X. Finally we define [20]:

$$\text{Lift}: \quad \ell(X \Rightarrow Y) = \frac{s(X \Rightarrow Y)}{s(X) \times s(Y)} = \frac{c(X \Rightarrow Y)}{s(Y)}$$

Both support and confidence are biased towards very frequent itemsets. If a certain itemset occurs much more than every other itemset, it will appear in almost every generated association rule. Lift gives a measure of how much a rule deviates from the expected confidence. The expected confidence is the confidence calculated with the assumption that X and Y are statistically independent, as denoted by $s(X) \times s(Y)$. Note that $\ell(X \Rightarrow Y) = \ell(Y \Rightarrow X)$. The lift of a rule

expresses the deviation from this expected confidence. Values can be within the range $[0, \infty)$ and can be divided into three ranges:

- $\ell(X \Rightarrow Y) = 1$ indicates that the confidence is equal to the expected confidence and that the occurrence of X has no effect on the occurrence of Y.
- $\ell(X \Rightarrow Y) > 1$ indicates that the occurrence of X has a positive effect on the occurrence of Y.
- $\ell(X \Rightarrow Y) < 1$ indicates that the occurrence of X has a negative effect on the occurrence of Y, they are mutually exclusive.

The previously defined quality metrics can be used for both associations and patterns. Confidence and lift can still be useful for patterns of two elements, but are not very relevant for higher order patterns. Support needs to be defined carefully to handle the order of a pattern. Therefore, we define the frequency count as:

$$\text{Frequency count}: \quad \sigma(Y) = \sum_{t \in D, Y \subseteq t} 1$$

where a pattern sequence $Y = (y_1, y_2, \ldots, y_n)$ is a subset of a transaction $t = (t_1, t_2, \ldots, t_k)$, denoted by $Y \subseteq t$, if there exist integers $i_1 < i_2 < \ldots < i_n$ such that $y_1 \subseteq t_{i_1}, y_2 \subseteq t_{i_2}, \ldots, y_n \subseteq t_{i_n}$. Notice that we use the same notation as on the previous page. For example, the sequence $(\{3\}, \{4,5\}, \{8\}) \subseteq (\{7\}, \{3,8\}, \{9\}, \{4,5,6\}, \{8\})$, because $\{3\} \subseteq \{3,8\}, \{4,5\} \subseteq \{4,5,6\}$ and $\{8\} \subseteq \{8\}$. Sequence $(\{3\}, \{5\})$ however, is not a subsequence of $(\{3,5\})$. See [2] and [15] for further explanation.

3.3 Association rules

The generation of association rules can be done with a brute force approach, by enumerating all possible combinations of items in the antecedent and consequent and evaluating the resulting rules. However, for an itemset of k items, there are $2^k - 1$ possible rules, so this quickly becomes infeasible. The computational complexity of association rule generation can be reduced by limiting the number of candidate itemsets. An effective way of doing this is using the Apriori principle [1]:

> **Apriori principle**. If an itemset is frequent, then all of its subsets must also be frequent.

This principle states that if, for example, itemset $\{a, b, c\}$ is frequent, the subsets $\{a, b\}, \{b, c\}, \{a, c\}, \{a\}, \{b\}, \{c\}$ are frequent too. Conversely, if an itemset is infrequent, all its supersets will be infrequent. So, if $\{c\}$ is infrequent, all itemsets containing c will be infrequent, as shown in Figure 1. This property, known as *anti-monotonicity* [14], gives us the possibility to discard large parts of the search space and is the main principle of the Apriori algorithm.

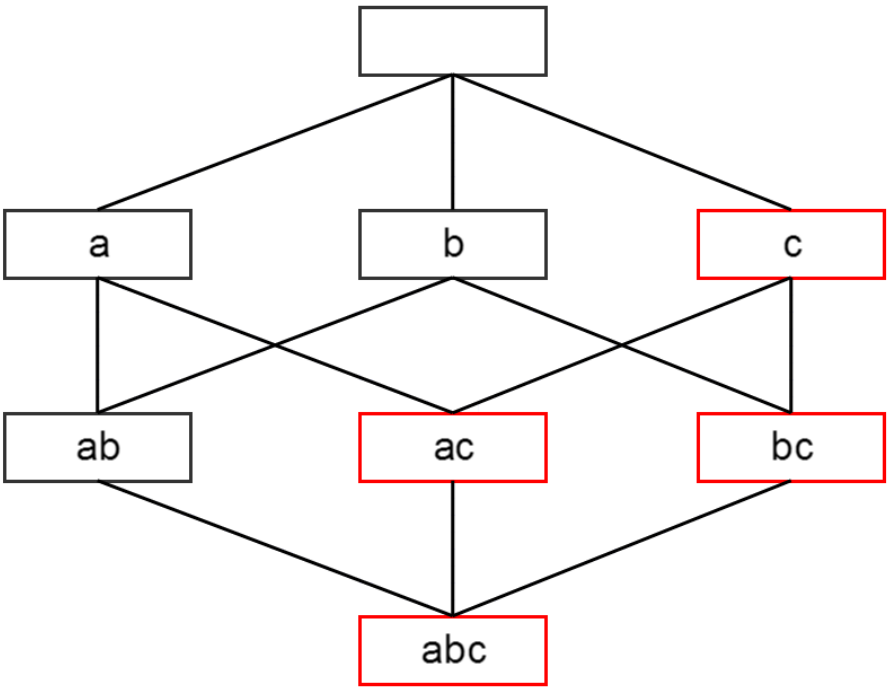

Figure 1: Anti-monotonicity example.

3.3.1 Frequent itemsets

The Apriori algorithm uses the Apriori principle by creating frequent itemsets in an iterative way, as shown in Algorithm 1. A frequent itemset f is an itemset for which the support count in the set of transactions D is above a certain treshold. The set of frequent itemsets with k items is denoted by F_k. The algorithm starts by making a single pass over the dataset to count the support for each individual item. This is done in step 2 of the algorithm. With this set of 1-item frequent itemsets F_1, a new set of 2-item candidate itemsets C_2 is generated with the AprioriGenerate function, as descried in Algorithm 2 and explained later on. For each candidate itemset, the support is calculated. The brute force approach is to make a pass over the dataset for each candidate itemset. There is, however, a better approach using a hashmap of the set of candidate itemsets. With this approach, the support of all candidates can be calculated in a single pass over the dataset (see [16, Chapter 6]). At the end of the iteration, the candidates with enough support are stored as frequent itemsets and used as input for the next iteration. In each iteration, the size of the candidate itemsets is increased with one. The algorithm halts when the resulting set of frequent itemsets is $\emptyset$, that is, when there are no frequent itemsets of size k, where k is defined as the number of items in the itemset. The set of frequent itemsets F now contains all frequent itemsets from sizes 1 to k, and F is used as input to create association rules, as explained in Section 3.3.2.

The AprioriGenerate function is used to generate candidate itemsets with k items for an iteration of the Apriori algorithm. As input it needs the set of

Algorithm 1 APRIORI algorithm

```
1: k ← 1
2: F_k ← {i | i ∈ I ∧ σ({i}) ≥ N × minimumSupport}
3: while F_k ≠ ∅ do
4:    k ← k + 1
5:    C_k ← AprioriGenerate(F_{k−1})
6:    for each t ∈ D do
7:       for each f ∈ C_k do
8:          if f ⊆ t then
9:             σ(f) ← σ(f) + 1
10:         end if
11:      end for
12:   end for
13:   F_k ← {c | c ∈ C_k ∧ σ(c) ≥ N × minimumSupport}
14: end while
```

frequent itemsets obtained in the last iteration, F_{k-1}. The output of the function is a set of candidate frequent itemsets with k items. Naturally, this set should be as small as possible, to prevent unnecessary evaluations. There is a number of methods to generate candidates. The simplest one is a brute force method, which, given a set of (frequent) items, generates all possible subsets of size k. That approach generates a large number of candidates, where most of them can be pruned due to the anti-monotonicity property. A smarter approach is to generate all combinations of F_{k-1} and F_1. This gives a shorter list of candidates, but it still contains itemsets that could be left out due to anti-monotonicity, for every subset of F_k must be frequent, not only F_{k-1} and F_k. For example, with $\{a, b\}$ and $\{c\}$, subset $\{a, c\}$ must also be frequent. The smallest complete candidate set can be reached by combining F_{k-1} with F_{k-1} itself, as explained in Algorithm 2. This algorithm generates candidates F_k by joining F_{k-1} with F_{k-1} where all items except the last one are equal. For each candidate f_{new} of size k, all subsets with $k-1$ itemsets are checked and only if all subsets are frequent itemsets, f_{new} is accepted as a candidate. We define the function LAST(Y) to return the last element in the ordered set Y, where $\emptyset \neq Y \subseteq I$.

3.3.2 Association rules

With the frequent itemsets generated with Algorithm 1, association rules can be generated quite easily. For each itemset, the algorithm generates all possible combinations of two non-empty disjoint sets that contain all items from the itemset. One of the sets serves as antecedent, the other as consequent. This is done recursively, starting with all 1-item antecedents. For these antecedents, the consequent is known, because it must contain the remainder of the itemset. For each candidate rule, the confidence is calculated using the already computed support values. If the confidence is above the threshold, the rule is stored. If all 1-item antecedents are evaluated, all possible 2-item antecedents are generated and evaluated, etc. The pseudocode of the algorithm is given in Algorithms 3 and 4. See [16, Chapter 6] for a detailled explanation.

Algorithm 2 AprioriGenerate(F_k) algorithm

```
1: C ← ∅
2: for each f_1 ∈ F_k and f_2 ∈ F_k do
3:    if f_1\{Last(f_1)} = f_2\{Last(f_2) and Last(f_1) < Last(f_2) then
4:       f_new ← f_1 ∪ f_2
5:       validCandidate ← true
6:       for each k-subset s of f_new do
7:          if s ∉ F_k then
8:             validCandidate ← false
9:          end if
10:      end for
11:      if validCandidate then
12:         C ← C ∪ {f_new}
13:      end if
14:   end if
15: end for
16: return C
```

Algorithm 3 AprioriAssociations algorithm

```
1: for each f_k ∈ F_k, k ≥ 2 do
2:    H_1 ← {i | i ∈ f_k}
3:    AprioriGenerateRules(f_k, H_1, k, m)
4: end for
```

Algorithm 4 AprioriGenerateRules(f_k, H_1, k, m) algorithm

```
1: if k > m + 1 then
2:    H_{m+1} ← AprioriGenerate(H_m)
3:    for each h ∈ H_{m+1} do
4:       c ← σ(f_k) / σ(f_k\{h})
5:       if c ≥ minimumConfidence then
6:          output f_k\{h} ⇒ {h}
7:       else
8:          H_{m+1} ← H_{m+1}\{h}
9:       end if
10:   end for
11:   AprioriGenerateRules(f_k, H_{m+1}, k, m + 1)
12: end if
```

3.4 Sequential patterns

The generation of sequential patterns is an extension of the process of generating association rules. A common algorithm to do this is the Generalized Sequential Pattern (GSP) algorithm, as proposed in [2] and [15]. The GSP algorithm has the same structure as the Apriori algorithm. They both start with the counting of frequent items in the dataset, which form the basis for the rest of the algorithm. As described in Algorithm 5, the GSP algorithm takes the same iterative approach as Apriori, where in each step a number of candidate patterns is generated and evaluated. The generation of candidate patterns is almost identical to the generation of frequent itemsets and is explained in Algorithm 6. The only difference lies in the join of f_1 and f_2, where we denote the elements of a pattern as $f = (f^1, f^2, \ldots, f^k)$. The new candidate f_{new} is created by appending the last item of f_2 to f_1. If the last element of f_2 contains more than one item, the item is added to the last itemset of f_1. If the last element of f_2 has only one item, it is added to f_1 as a new element. In the special case where two frequent items are joined into a candidate with two items, three versions of the candidate are generated, one with both items in one element and two with each item in a separate element, in both possible orders. So for example, the patterns $(\{a\})$ and $(\{b\})$ will result in three candidates $(\{a, b\})$, $(\{a\}, \{b\})$ and $(\{b\}, \{a\})$.

With the resulting candidate set, support for all candidates is calculated by making a single pass over the transaction database. For each transaction, all candidates that are subsets of the transaction have their support count increased with the GspSubset procedure. At the end of each iteration of the main loop, the candidate patterns with enough support are stored and used as a seed set for the next iteration.

Algorithm 5 Generalized Sequential Pattern algorithm

```
1:  F_1 ← all frequent items
2:  k ← 2
3:  while F_{k−1} ≠ 0 do
4:    C_k ← GspGenerate(F_{k−1}, k)
5:    for each t ∈ D do
6:      for each f ∈ C_k do
7:        σ(f) ← σ(f) + GspSubset(t, f)
8:      end for
9:    end for
10:   F_k ← {c | c ∈ C_k ∧ σ(c) ≥ N × minimumSupport}
11:   k ← k + 1
12: end while
```

For Algorithm 6, we define the function $\textsc{Last}(Y)$ for a pattern Y to return the last item of the last element of Y and we define:

$$\textsc{GspSubset}(t, f) = \begin{cases} 1 & \text{if } f \subseteq t \\ 0 & \text{otherwise} \end{cases}$$

The GspSubset function checks, for a given transaction t and a pattern f, if the pattern is a subset of the transaction, and if so, returns a support share for t. In the general setup, support is increased by 1 if all elements of the pattern appear

Algorithm 6 GspGenerate(F_k, k) algorithm

1: $C \leftarrow 0$
2: **for each** $f_1 \in F_k$ and $f_2 \in F_k$ **do**
3: **if** $f_1 \backslash \{\textsc{Last}(f_1)\} = f_2 \backslash \{\textsc{Last}(f_2)\}$ and $\textsc{Last}(f_1) < \textsc{Last}(f_2)$ **then**
4: **if** $k = 1$ **then**
5: $f_a \leftarrow (\{\textsc{Last}(f_1), \textsc{Last}(f_2)\})$
6: $f_b \leftarrow (\{\textsc{Last}(f_1)\}, \{\textsc{Last}(f_2)\})$
7: $f_c \leftarrow (\{\textsc{Last}(f_2)\}, \{\textsc{Last}(f_1)\})$
8: $C \leftarrow C \cup \{f_a, f_b, f_c\}$
9: **else**
10: $f_a \leftarrow (f_1^1, f_1^2, \ldots, f_1^k, \{\textsc{Last}(f_2)\})$
11: $f_b \leftarrow (f_1^1, f_1^2, \ldots, f_1^{k-1}, f_1^k \cup \{\textsc{Last}(f_2)\})$
12: $C \leftarrow C \cup \{f_a, f_b\}$
13: **if** $|f_1^k| = 1$ **then**
14: $f_c \leftarrow (f_1^1, f_1^2, \ldots, f_1^{k-1}, \{\textsc{Last}(f_2)\}, f_1^k)$
15: $C \leftarrow C \cup \{f_c\}$
16: **end if**
17: **end if**
18: **end if**
19: **end for**
20: **return** C

in the same order in the transaction. This can be narrowed down with time constraints, as discussed in Section 3.5. We define that a pattern appears in a transaction, as denoted by $f \subseteq t$ for a pattern $f = (f_1, f_2, \ldots, f_k)$ and a customer $t = (id, \langle a_1, t_1 \rangle, \langle a_2, t_2 \rangle, \ldots, \langle a_n, t_n \rangle)$, if there exist integers $i_1 < i_2 < \ldots < i_k$ such that $f_1 \subseteq a_{i_1}, f_2 \subseteq a_{i_2}, \ldots, f_k \subseteq a_{i_k}$.

3.5 Time constraints

There are many approaches to the evaluation of sequential patterns. In the simplest case, we can set a minimum support value and discard all patterns that fail to reach this level of support, or use the same approach for one of the other quality evaluation metrics defined in Section 3.2 that can be used as a minimum threshold value. However, this approach limits the results to often occurring combinations, which does not always mean the resulting patterns are interesting.

To refine the resulting set of patterns, time constraints can be used. It can be very insightful to make a distinction between patterns that occur within and outside certain time windows. For example, a maximum gap between items can be specified, such that for each two subsequent elements of itemset f found in transaction t, the time gap between them may not exceed the specified maximum. In the same manner, a minimum gap can be defined. These gaps can be extended into intervals, as discussed in Section 3.5.1. We can think of special cases such as contracting all transactions within one week like they happened on the same day. Apart from these discrete evaluations where support is either 0 or 1, a continuous evaluation can be defined, for example resulting in a support value between 0 and 1 based on the time gaps between transaction elements. This time-dependent support weighting will be discussed in Section 3.5.2.

3.5.1 Time-intervals

It is straightforward to add a number of time-intervals of predefined length to a sequential pattern algorithm, as is proposed in [7] and [23]. We can think of them as a series of extra products, such as "one month" or "80 days", that are placed between products. Every pattern consists of a number of products and a number of intervals defining the elapsed time between the purchase of two products, so intervals are defined as part of a pattern. In the initial definition of a sequential patterns, an element of a pattern can be a set of products. However, in this section we will assume that each element contains one product, because that is sufficient for our data set. It is straightforward to extend the definition to patterns with multi-item-elements. The support calculation needs to be extended to take into account time-intervals. The time-interval modification can be defined as follows.

A customer is defined as an ordered set of pairs and a unique identifier

$$c = (id, \langle a_1, t_1 \rangle, \langle a_2, t_2 \rangle, \ldots, \langle a_n, t_n \rangle)$$

where $\langle a_i, t_i \rangle$ is an item a_i with corresponding transaction date t_i, with $t_1 \leq t_2 \leq \ldots \leq t_n$. A customer can be handled as one transaction with multiple transaction dates, so we will use the terms customer and transaction interchangeably. A customer can buy the same item multiple times, as long as the transaction dates are different. Let t be the time interval between two successive items in a transaction c. For example, $t = |t_2 - t_1|$ for a customer $c = (\langle a_1, t_1 \rangle, \langle a_2, t_2 \rangle)$. We define an ordered set of points in time B as

$$B = (b_0, b_1, b_2, \ldots, b_r)$$

where $b_0 \leq b_1 \leq \ldots \leq b_r$ and $b_0 = 0$. We define a set of intervals $I = (I_1, I_2, \ldots, I_r)$ over B, where an interval $I_j = (b_{j-1}, b_j]$. Furthermore, we define a left-open interval $I_\infty = (b_r, \infty)$. Thus, we get a list of intervals that exactly contains the interval $(0, \infty)$.

- I_1 with $b_0 = 0 < t \leq b_1$
- I_2 with $b_1 < t \leq b_2$
- ...
- I_r with $b_{r-1} < t \leq b_r$
- I_∞ with $b_r < t$.

For a pattern $p = (p_1, p_2, \ldots, p_k)$, we define a sequence of intervals $I_p = (I_{p_1}, I_{p_2}, \ldots, I_{p_{k-1}})$, with $I_{p_i} \in I$ and I_{p_i} as the interval between p_i and p_{i+1}. A pattern p appears in a customer $c = (id, \langle a_1, t_1 \rangle, \langle a_2, t_2 \rangle, \ldots, \langle a_n, t_n \rangle)$, denoted as $p \subseteq c$, if there exist integers $i_1 < i_2 < \ldots < i_k$ such that $p_1 \subseteq a_{i_1}, p_2 \subseteq a_{i_2}, \ldots, p_k \subseteq a_{i_k}$ and $|t_{i_2} - t_{i_1}| \in I_{p_1}, |t_{i_3} - t_{i_2}| \in I_{p_2}, \ldots, |t_{i_k} - t_{i_{k-1}}| \in I_{p_{k-1}}$, where $t_j \in I_{p_j}$ is defined as $b_{p_{j-1}} < t_j \leq b_{p_j}$. A pattern with an interval is denoted as a pair $\langle p, I_p \rangle$.

The set of points in time B is given, so the set of available intervals I is predefined. For the generation of candidate patterns with intervals, the intervals have to be handled as an extra set of products. For every pair of subsequent

items in a pattern, an interval must be defined, so for every generated candidate pattern and for every interval, a new candidate is generated. For the combination of two items in a pattern, $r+1$ versions will be generated, each having one of the $r+1$ intervals. For example for the set of points in time $B = (0, 7, 31, 365)$ and a pattern with two items (a, b), the intervals for B are $I = (0, 7], (7, 31], (31, 365]$ and $I_\infty = (365, \infty)$. These intervals are shown in Figure 2 and the candidate generation results in 4 pairs $\langle p, I_p \rangle$:

- $\langle (a, b), (0, 7] \rangle$
- $\langle (a, b), (7, 31] \rangle$
- $\langle (a, b), (31, 365] \rangle$
- $\langle (a, b), (365, \infty) \rangle$

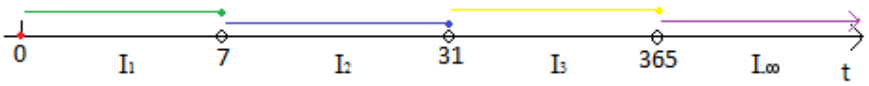

Figure 2: Time intervals example

To handle intervals, the support calculation needs an extra constraint. A pattern p is only considered to occur in transaction c if, for every successive pair of items in c to which the pattern is matched, the gap between the transaction dates lies within the interval specified for the pair in p. We define the frequency count for a pattern-interval-pair $\langle p, I_p \rangle$ and the set of customers D:

$$\text{Frequency count}: \quad \sigma(\langle p, I_p \rangle) = \sum_{c \in D, \langle p, I_p \rangle \subseteq c} 1$$

where a pattern-interval-pair $\langle p, I_p \rangle$ appears in a customer c, denoted as $\langle p, I_p \rangle \subseteq c$, if there exist integers $i_1 < i_2 < \ldots < i_k$ such that $p_1 \subseteq a_{i_1}, p_2 \subseteq a_{i_2}, \ldots, p_k \subseteq a_{i_k}$ and $|t_{i_2} - t_{i_1}| \in I_{p_1}, |t_{i_3} - t_{i_2}| \in I_{p_2}, \ldots, |t_{i_k} - t_{i_{k-1}}| \in I_{p_{k-1}}$, where $t_j \in I_{p_j}$ is defined as $b_{p_{j-1}} < t_j \leq b_{p_j}$.

To expand the example above, we consider a transaction $c = \langle (a, 15), (b, 20) \rangle$. The interval t between a and b is $|20 - 15| = 5$, so it falls within interval I_1 and counts as support for candidate $(a, 7, b)$. Note that a pair of items in a transaction can occur in exactly one interval.

3.5.2 Time-dependent support weighting

The main concept of time-dependent weighting is that the support value for transaction-pattern pairs is not limited to 0 or 1, but can have a value between these borders where the weighting value is determined by the time between transactions. We will first give a more formal definition, then we will discuss some weighting functions.

To handle variable weighting, the support function needs to be extended. Again we define a pattern as $p = (p_1, p_2, \ldots, p_k)$ where every p_i is a unique item, and a customer (or transaction) $c = (id, \langle a_1, t_1 \rangle, \langle a_2, t_2 \rangle, \ldots, \langle a_n, t_n \rangle)$. We define the support function s for a pattern p as:

$$s(p) = \frac{1}{N} \sum_{c \in D} F(c, p)$$

which is the sum of all transactions c in the set of transactions D. For each of these transactions, a weighting function $F(c, p)$ is calculated. We define:

$$F(c,p) = \begin{cases} \dfrac{1}{k-1} \displaystyle\sum_{j=1}^{k-1} f(|t_{i_{j+1}} - t_{i_j}|) & \text{if } p \subseteq c \\ 0 & \text{otherwise} \end{cases}$$

where $(i_1, i_2, \ldots, i_k)$ is the ordered set of integers as defined before and $f(t)$ is a weighting function which gives a result in the interval $[0, 1]$. Again, $p \subseteq c$ if there exist unique integers $i_1 < i_2 < \ldots < i_k$ such that $p_1 = a_{i_1}, p_2 = a_{i_2}, \ldots, p_k = a_{i_k}$.

This function sums, for all subsequent pairs of items in p and their corresponding items in c, the result of $f(t)$ for this pair. The result of the sum is divided by $k-1$, resulting in an average support value in the interval $[0, 1]$ for all subsequent pairs in p. Note that $f(t)$ is undefined for $k = 1$, because an interval can be only be defined between at least two items and has no meaning for a single product. Therefore, this support count can only be used on sequential patterns with two or more items. For the inital phase of the GSP, the counting of frequent items, we use the original support count as defined in section 3.2.

With this formal support weighting framework, we can create variation by defining various weighting functions $f(t)$. The general idea behind this weighting function is that it returns an output based on the gap in days between the transaction dates of two products. This function is required to give valid output in the range $[0, m]$, where m is a predefined maximum gap between two transactions. Outside of this interval, the function can be undefined and in that case, the output is handled as 0. The most trivial case is is the function f_1, which returns 1 for every input.

$$f_1(t) = 1$$

Another simple function is the linear f_2,

$$f_2(t) = 1 - \frac{t}{m}$$

which returns a value that is linearly dependent on t, where $f_2(m) = 0$. The next function, f_3, is a quadratic relationship where weighting decreases squared to the gap. As it is a part of a negative parabola, the weighting decrease starts slowly for small gaps and increases as the size of the gap reaches m.

$$f_3(t) = 1 - \left(\frac{t}{m}\right)^2$$

Then, f_4 is another quadratic function which is part of a positive parabola, so the weighting decrease starts quickly for small gaps and decreases as the size of the gap reaches m.

$$f_4(t) = \left(\frac{t}{m} - 1\right)^2$$

Another weighting function has a stepwise weighting, defined by the number of steps n

$$f_5(t) = 1 - \lfloor \frac{x \times n}{m} \rfloor \times \frac{1}{n-1}$$

All of these functions focus on shorter intervals. However, f_6 is a mirrored version of f_2 with its maximum at m.

$$f_6(t) = \frac{t}{m}$$

To get a focus on the intervals around $\frac{m}{2}$, another quadratic function f_7 is defined.

$$f_7(t) = 1 - \left(\frac{2t}{m} - 1\right)^2$$

The plots of these functions, where the maximum gap m is set to one month, can be found in Figure 3.

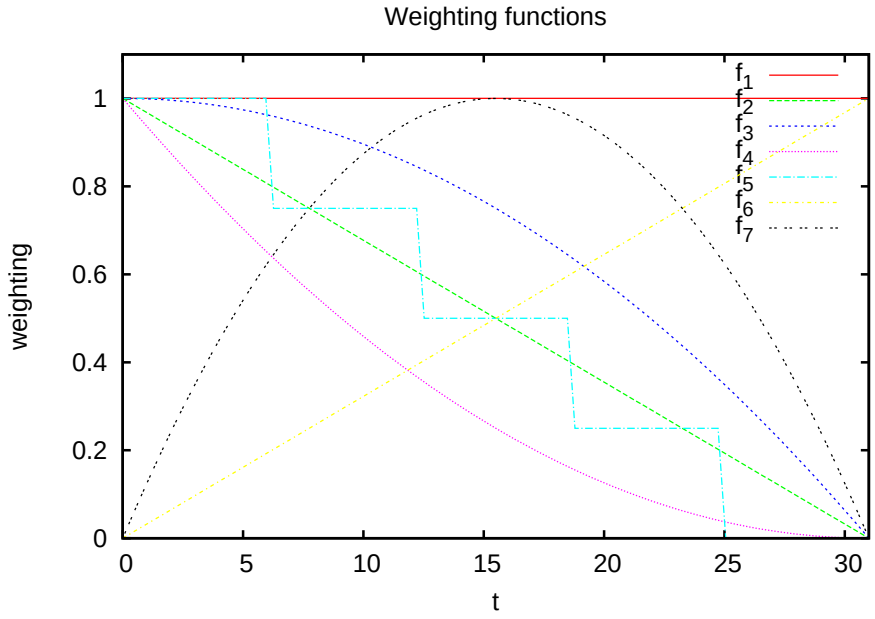

Figure 3: Weighting functions for $m = 31$; $n(f_5) = 5$.

For example we consider a pattern $p = (x, y)$ with two products x and y and two customers $c_1 = (\langle x, 1\rangle, \langle y, 5\rangle, \langle z, 10\rangle)$ and $c_2 = (\langle w, 7\rangle, \langle x, 10\rangle, \langle z, 13\rangle, \langle y, 16\rangle)$. Calculating $s(p)$, we get $s(p) = \frac{1}{2}\left(F(c_1, p) + F(c_2, p)\right)$, as both c_1 and c_2 contain p. Then, $F(c_1, p) = \frac{1}{2}(f(|5-1|))$ and $F(c_2, p) = \frac{1}{2}(f(|16-10|))$.

The final result of the support value depends on the weighting function $f(t)$ and maximum interval m we choose. For example, with $m = 15$ for f_4, we get $f_4(|5-1|) = \left(\frac{4}{15} - 1\right)^2 = \frac{121}{225} \simeq 0.538$ and $f_4(|16-10|) = \left(\frac{6}{15} - 1\right)^2 = \frac{81}{225} = 0.36$. So, for f_4, we get $s(p) = \frac{1}{2}\left(\frac{1}{2} \times 0.538 + \frac{1}{2} \times 0.36\right) = \frac{1}{4} \times 0.898 = 0.2245$.

Using one of these weighted support functions will obviously have an influence on the results. However, this is strongly dependent on the value of m. In the

case of $m \to \infty$, the weighting functions f_2 to f_5 will converge into the trivial function f_1 and f_6 and f_7 will converge to 0. A useful value for m depends on the data set. If, for example, the maximum gap in the dataset is 3 years, m should be 3 years or less. An even lower value might be better, depending on the distribution in the data set.

An important remark has to be made on the effects of the weighting function $F(c, p)$ that is defined in this section. The weighting function calculates an average over the fitness of all subsequent pairs in the sequence. This creates the possibility that, for certain cases, the support score goes up for longer sequences. This breaks the *anti-monotonicity* principle, which states that if an itemset is infrequent, all its supersets must also be infrequent. For example, consider a pattern of two items $p_1 = (a, b)$, a pattern of three items $p_2 = (a, b, c)$, weighting function f_2 and a data set where product b is usually sold after a, with a long interval between them, and c after b, with a short interval between them. As f_2 is focussed on short intervals, the support for the sequence (a, b) will be small and the support for (b, c) will be large. As the support for p_2 is an average of the support of (a, b) and (b, c), this can result in a higher support value for p_2 than for p_1. Now, if the predefined minimum support is more than the support of p_1 and less than the support of p_1, the anti-monotonicity principle no longer holds, as $(a, b) \subseteq (a, b, c)$, but (a, b, c) is frequent and (a, b) is not. Therefore, the anti-monotonicity principle can not be used for weighted patterns if the patterns contain three or more items. This means that the candidate generation phase of the GSP-algorithm can no longer assume that certain candidates can be pruned only because there is a subset which is infrequent. So instead of the AprioriGenerate candidate generation algorithm, we need a brute force candidate generation approach which generates all possible subsets of a given length for a set of frequent items. This means that all possible orderings of products of length 2 to k have to be generated, which can be very expensive, as generating all permutations of k items result in $k!$ permutations. For example, with $k = 10$ we already get 3628800 candidates of 10 items to evaluate. Therefore, this is only feasible for a small number of products. In our implementation, we used a maximum length of 3 items to limit the number of candidates, which is consistent with our data set.

3.5.3 Complexity

The algorithms and improvements that are presented are all very straightforward, with little or no focus on computational complexity. It would be easy to come up with a number of ways to improve complexity, or even to switch to a completely different, faster algorithm. However, in this study there is no reason to do so. The present data set, as is discussed in Section 4.1, has a limited size and complexity. Only 13% of the customers purchased more than one product, and less than 5% of the customers purchased more than two products. So, what we can expect is a very sparse data set, associations and patterns with small support percentages and only short patterns of two, or sometimes three products. Besides that, the data set only contains a very small number of products, so exhaustive generation of all subsets upto a given size is feasable. Therefore, there is no need to implement a more sophisticated algorithm.

Larger data sets with more products or data sets with many more relations would give more candidates and longer patterns. As the complexity of the

candidate generation phase in APRIORI algorithms is $O(n^2)$ in the number n of items for 2-itemsets and $O(n^3)$ for 3-itemsets, candidate generation quickly becomes infeasible. For longer patterns, the number of passes over the data set increases, which can lead to performance problems for large data sets. To handle this type of data sets, a more efficient algorithm should be used. A commonly used algorithm in these cases is FP-TREE, proposed by Han and Pei [12]. To omit the candidate generation phase, this algorithm generates a prefix tree by creating and extending pattern fragments. This results in a summary of the data set, the avoidance of candidate generation and reduced search space due to a divide-and-conquer approach of the evaluation phase. An extended explanation can be found in [12]. Another optimized algorithm for the generation of larger itemsets is Dynamic Itemset Counting, as suggested by Brin [5].

Another way to extract useful information for a large, rich database is to use Subgroup Discovery. This approach goes beyond finding the relations with the highest support and focusses on more detailed links within these relations. For example, a data set may contain a large number of travel insurances, mostly for customers in the age between 20 and 50. A subgroup of the owners of a travel insurance policy, might consist of elderly people in the age between 60 and 70 who happen to own an RV. These subgroups are typically overlooked by general pattern mining algorithms, but can be found with more specified subgroup discovery algorithms, for example the algorithm proposed in [13].

4 Experimental Results

In the previous chapter we have described the algorithms, quality evaluations and interval-related extensions, so now they can be tested with real world data. The Market Basket Analysis algorithms are applied to an insurance data set, as explained in Section 4.1. In the following sections, the results for each algorithm and extension are presented and discussed.

4.1 Data set

The original data set used for this thesis resides in the Quinity Data Mart, the data warehousing environment of the Quinity Insurance Solution. It contains a copy of all production data on clients, policies and claims. This data is mainly used for management reporting, giving insight in, for example, revenues per period, product or intermediary. The database has a size of about 7 TB, but most of that is not very useful for this study, as it contains a history of all changes made to customer profiles, policies and claims, logging of all automated processing of the data and large tables with atomic financial data for reporting purposes.

The complete database contains about 6.3 million customers with a total of 3.9 million policies and 630.000 claims. A policy is a connection of one customer and one product, documenting that the customer purchases this product, possibly with small changes to fit the product to the customers wishes. The database has about 150 products, but some of them are old or obsolete. Our data set uses a subset of this data, because we focus on only one of the insurance brands of the company, having about 1.6 million policies for 50 products. As the ratio between customers and policies shows, customers usually do not purchase a large number of policies. In the resulting data set, about 87% of the customers has one policy, 8.6% owns two policies and less than 5% of the customers has three or more policies. Therefore, we should expect low support and confidence rates and mainly combinations of two products.

In order to abstract a usable data set from the database, some preprocessing is required. First, an SQL query with a number of constraints gives the basic transaction data. The constraints are constructed using the definitions and requests from the insurance company, so we are only interested in users who still have at least one active policy and sparsely occurring products can be omitted. Some of these constraints are handled in a preprocessing script written in Python. This script handles the expanded products, normalization of customer-IDs and policy-IDs and the correct input format for the Java program. After preprocessing, the only remaining data is a list of normalized triples containing

$$\langle \text{customer}, \text{product}, \text{timestamp} \rangle$$

These triples, with a list of product names, are used as input for the Apriori and GSP algorithms. The results can be found in the next sections.

To protect the commercial interests of the insurance company, product names are omitted from the results and insurance products are enumerated with A, B, C, etc. However, because we deal with real world data, there are some relevant characteristics of the products that have an influence on the results.

- *Product A* appears very often in the data set, so it can be expected to occur a lot in the results.

- *Product B* appears very often in the data set, but significantly less than Product A.
- *Product C* is actually a group of products that are often sold together, so they will appear together a lot. The products of the group are defined as C1, C2, C3, etc.
- *Products D to L* do not appear in the data set very frequently, because they are either old products which are no longer actively sold or they are niche products.

Beside the commercial interests in the data set, there is also the privacy of the customers that has to be taken into account. Privacy is a general issue in data mining on real world data. To ensure the privacy of the cusomers involved, the results are completely anonymized. The only customer information extracted from the database is the customer-ID, and in the preprocessing phase, this ID is substituted for a normalized ID.

4.2 Implementation

Implementation of the Apriori and GSP algorithms was done in Java. Altough many implementations are available, we decided to create our own implementation, because of the limitations of the secure environment in which the data set was available and as a hands-on study of the algorithms. Because of the nature of the dataset, as explained in Section 4.1, a few modifications were made. According to the definition of a sequential pattern in Section 3.4, a pattern contains a series of groups of items called elements, where all items of a group have to have the same transaction date. In the implementation however, the size of an element is limited to one product, because the purchase of multiple items on the same day is very rare. In fact it almost only occurs for items of Product group C. However, items with the same purchase date are analysed in Section 4.5.

4.3 Association Rules

As a first step in mining the data set, we generate a set of association rules, using our own implementation of the Apriori algorithm presented in Section 3.3. The Apriori algorithm gives a number of associations, which are shown in Table 1.

As input for the algorithm, a selection of the data set is used, because all transactions with only one item are omitted. The resulting data set consists of 95.719 customers with two or more policies. The confidence and lift percentages are relative to this number. To limit the number of rules, support and confidence have to be set to a useful value. With minimal support set to 10% and minimal confidence set to 1%, we get a limited list of results, containing only the often occurring products A, B and C. This results in only a few associations, each consisting of two or, in some cases, three items, as shown in Table 1. Lowering the minima leads to an explosion of new rules. Since the dataset has about 100.000 customers, a support of 10% means 10.000 policies, so these results are still quite significant. A graphical representation of the associations can be found in Figure 4, where the weight of the edges is specified by the confidence

Table 1 Association rules as found with Apriori, with support, confidence and lift.

Association rule	Support	Confidence	Lift
A ⇒ B	32.082%	47.848%	0.928
B ⇒ A	32.082%	62.216%	0.928
A ⇒ C1	11.051%	16.482%	0.657
C1 ⇒ A	11.051%	44.020%	0.657
A ⇒ C2	10.529%	15.703%	0.638
C2 ⇒ A	10.529%	42.789%	0.638
B ⇒ C1	13.734%	26.633%	1.061
C1 ⇒ B	13.734%	54.707%	1.061
B ⇒ C2	13.517%	26.212%	1.065
C2 ⇒ B	13.517%	54.931%	1.065
C1 ⇒ C2	22.281%	88.752%	3.607
C2 ⇒ C1	22.281%	90.549%	3.607
C4 ⇒ C2	10.712%	96.781%	3.933
C2 ⇒ C4	10.712%	43.532%	3.933
C2 ⇒ B, C1	12.337%	50.138%	3.651
C1 ⇒ B, C2	12.337%	49.143%	3.636
B ⇒ C1, C2	12.337%	23.925%	1.074

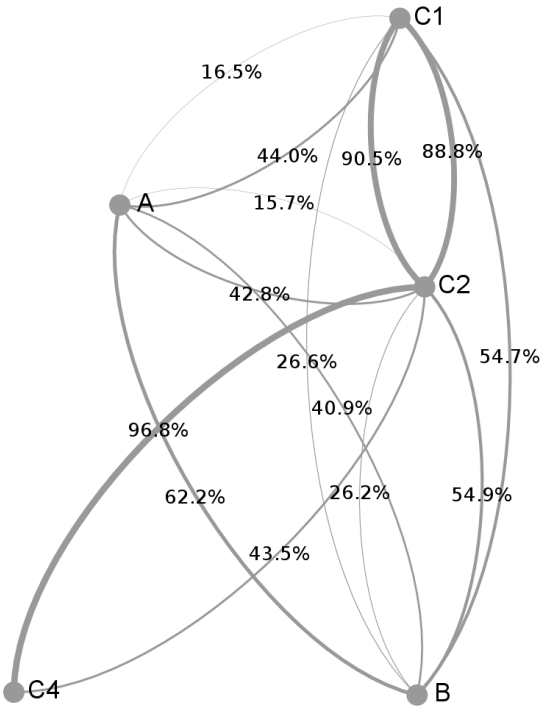

Figure 4: Two-item associationrules with confidence. Figure created with Gephi [4].

percentage. The relations have to be read clockwise, so the rule $(C2 \Rightarrow C4)$ has confidence 96.8% and the rule $(C4 \Rightarrow C2)$ has confidence 43.5%.

In this diagram, we see strong relations within the product group C, and Table 1 shows that the lift of these rules is quite high. This is no surprise, as these products are closely coupled and often sold together. There also are relations between A and B and C products, for example $\langle C1 \Rightarrow B \rangle$ and $\langle B \Rightarrow A \rangle$, but when we look at the lift of these rules (1.061 and 0.928), their values are very close to 1, so while these rules have a strong confidence, this is only because these products appear very frequent in the data set and their confidence is expected to be high.

In general we see that the association rules we have found are consistent with our knowledge of the data set. They give insight into the relations between the products, but the results are not very surprising. To gain a deeper knowledge of the data set and the relations, the purchase date of the products has to be taken into account. This is done with the GSP algorithm, whose results will be discussed in the next section.

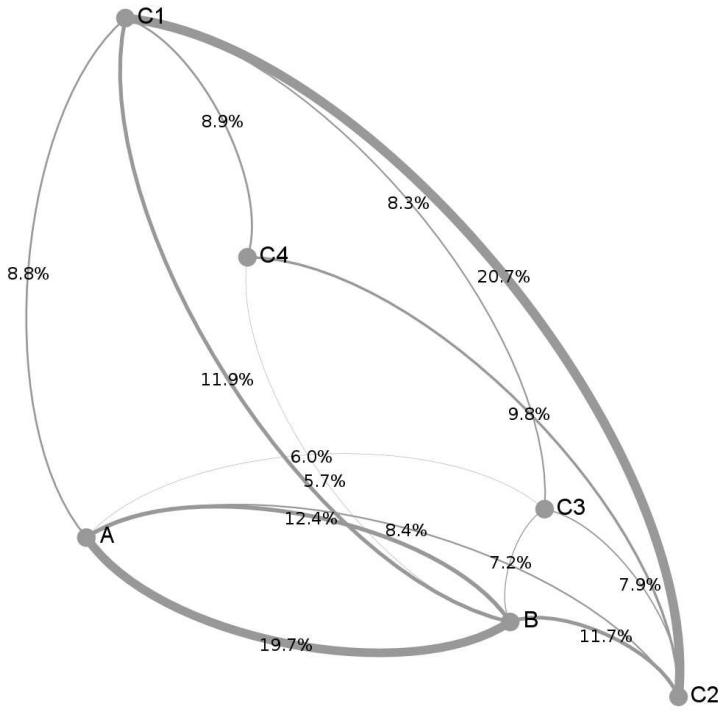

Figure 5: Two-item sequential patterns with support. Figure created with Gephi [4].

4.4 Sequential patterns

To gain a more detailed insight into the data set, sequential patterns can be very helpful, as they take into account the purchase order and the temporal relationship between products. With the GSP algorithm, as explained in Section 3.4, a set

of sequential patterns is generated. These patterns have more constraints than the association rules of the previous section, as the order of the purchases is taken into account, so a lower support percentage has to be expected. To get a significant number of patterns, the support minimum is lowered to 2%. The resulting patterns are shown in Table 2.

Table 2 Sequential pattern as found with GSP, with support $\geq 2\%$ en minimum interval 0.

Sequential pattern	Support	Sequential pattern	Support
C1 → C2	20.657%	C1 → C3 → C4 → C2	3.871%
B → A	19.650%	A → C1 → C4	3.848%
A → B	12.433%	B → I	3.741%
B → C1	11.880%	B → C1 → C3 → C2	3.701%
B → C2	11.679%	A → C1 → C4 → C2	3.512%
B → C1 → C2	9.862%	A → C1 → C3	3.424%
C4 → C2	9.833%	B → G	3.416%
C1 → C4	8.944%	A → C3 → C2	3.390%
A → C1	8.837%	C1 → C5	3.381%
A → C2	8.389%	C2 → C5	3.161%
C1 → C3	8.298%	B → A → C1	3.066%
C1 → C4 → C2	8.104%	C1 → C2 → C5	3.032%
C3 → C2	7.887%	B → A → C2	2.977%
B → C3	7.249%	A → C1 → C3 → C2	2.886%
A → C1 → C2	7.027%	B → C3 → C4	2.722%
C1 → C3 → C2	6.788%	B → A → C1 → C2	2.529%
A → C3	5.959%	B → C3 → C4 → C2	2.469%
B → C4	5.734%	C3 → C5	2.422%
A → D	5.666%	B → E	2.356%
B → C4 → C2	5.056%	G → A	2.355%
B → C1 → C4	4.791%	B → C1 → C3 → C4	2.351%
C3 → C4	4.776%	A → C3 → C4	2.302%
F → A	4.690%	B → D	2.267%
A → C4	4.640%	C1 → A	2.215%
A → E	4.470%	C1 → C3 → C5	2.215%
B → C1 → C3	4.392%	A → B → C1	2.186%
B → C1 → C4 → C2	4.368%	B → C1 → C3 → C4 → C2	2.180%
B → C3 → C2	4.363%	A → B → C2	2.154%
C3 → C4 → C2	4.331%	C2 → A	2.141%
C1 → C3 → C4	4.181%	C3 → C2 → C5	2.125%
A → C4 → C2	4.097%	A → C3 → C4 → C2	2.098%
A → G	4.073%	C1 → C3 → C2 → C5	2.040%
E → A	3.954%	A → C1 → C3 → C4	2.002%
A → H	3.886%		

A more intuitive representation of the patterns of two items with support at least 5% is shown in Figure 5, where the weight of the patterns is given by the support percentage. In this diagram we see very strong relations between C1 and C2, and B and A. Both of these relations have a significantly higher support in one direction, so apparently, the product purchase order $\langle B \to A \rangle$ appears a

lot more often than $\langle A \rightarrow B \rangle$. However, we can define the order of the product purchases in two ways. In a general approach, two products purchased on the same day are counted for the support percentage, and in a stricter approach, we require a minimum interval of one day. Note that this does not involve the time-intervals we defined previously, as their results are presented in the next section. The results in Figure 5 are generated with the less strict approach. When a set of results is generated with the stricter approach, the diagram changes, as shown in Figure 6. There are some notable differences between the two diagrams. The strongest pattern $\langle C1 \rightarrow C2 \rangle$ has disappeared and all other relations between C products are gone as well. Apparently, their support for intervals of length 1 and higher is lower than the minimum support threshold. This means that the products of group C are usually sold together at the same time, but the case where a customer with a C product purchases another C product after some time is very rare. On the other hand, the patterns with A or B and a C product are still there, so these relations are more distributed in terms of intervals. In the next sections, we will study these intervals with more detail.

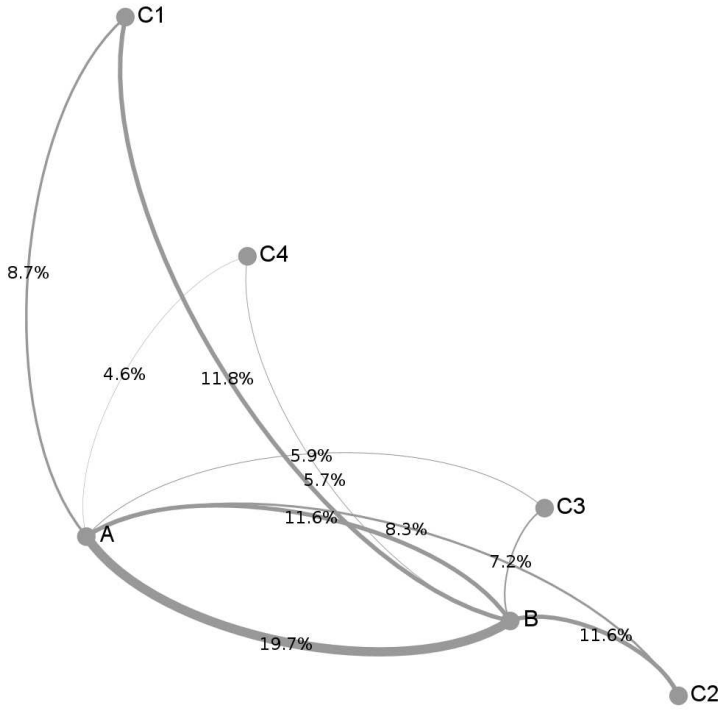

Figure 6: Two-item sequential patterns with support, with minimum interval 1. Figure created with Gephi [4].

4.5 Time-intervals

Now we have seen the general relations and order between the items in the data set, we can look into the purchase orders with a higher detail. For this purpose, the time interval extensions of the GSP algorithm, as defined in Section 3.5.1,

will be used. We will focus on the relations between A, B and one of the C products. The relations between C products are omitted, because they are mainly purchased on the same day. Note that, altough each pattern can have its own interval, in this section we compare various patterns for the same intervals, in order to gain insight into the temporal behavior of the patterns.

To explore the high level relation between the length of the interval and the number of matching transactions, we define a series of intervals, each with the length of one year, with the ordered set of points in time $B = (0, 365, 730, \ldots)$ which gives us a list of intervals I:

- I_1 for $0 < t \leq 1$ year
- I_2 for 1 year $< t \leq 2$ years
- I_3 for 2 years $< t \leq 3$ years
- Etc.

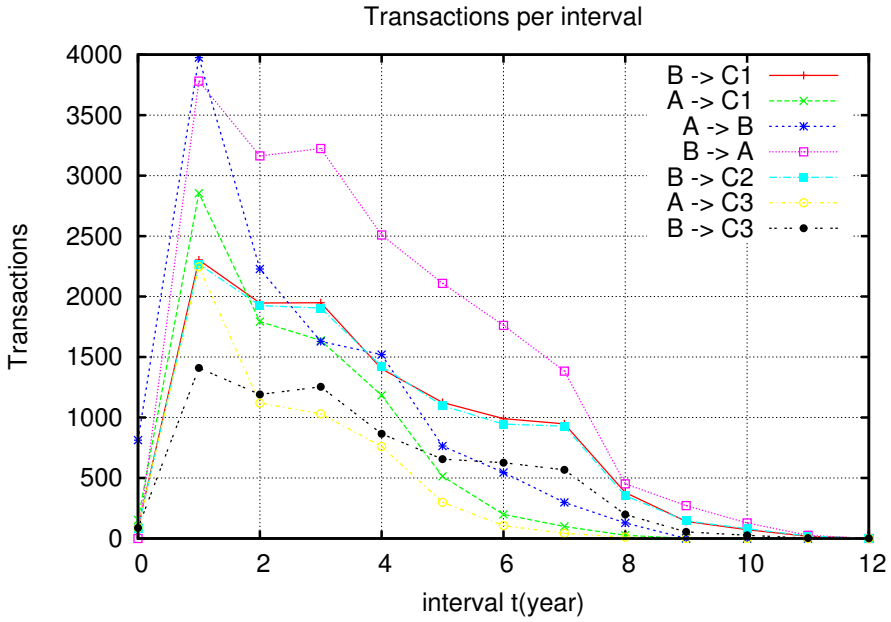

Figure 7: Number of transactions per year-long interval for the most important patterns.

When we plot the frequency of the patterns for these intervals, we get Figure 7, which plots the number of transactions per interval for all found combinations of A, B and C. In general, we see the same behaviour for all patterns. For the first interval, where $t = 0$, a small number of transactions is counted. The peak of all patterns is in the second interval, between one day and one year. The height of the peak is dependent of the frequency of the product in the data set. The number of transactions per interval decreases in the intervals 2 to 8. All patterns that start with B have a remarkable characteristic, as for all of them, the interval

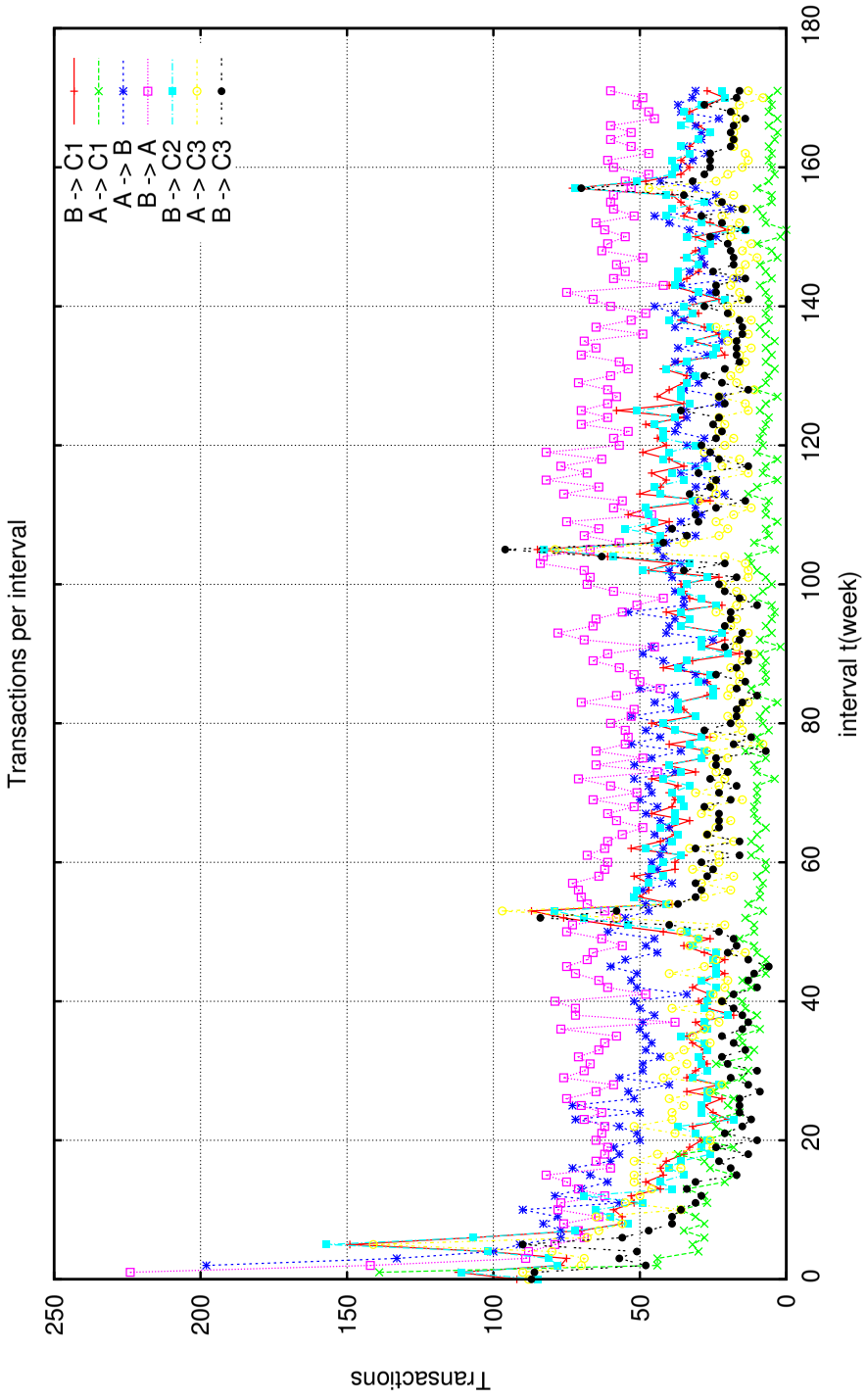

Figure 8: Number of transactions per week-long interval for the most important patterns.

3 contains more transactions than interval 2. The patterns that start with A decrease faster for longer intervals than patterns that start with B.

According to the previous diagram, the intervals from one day to three years have the most corresponding transactions for all patterns, so we will take a more detailed look at them. Figure 8 shows the number of transactions per one-week-interval for all patterns of the previous diagram. The results are scattered and vary greatly for every interval, but three peaks show up. For the patterns $\langle B \rightarrow C \rangle$, there are significantly more transactions for the intervals around $t = 52, t = 104$ and $t = 156$. When only the $\langle B \rightarrow C \rangle$ patterns are plotted, the peak becomes even clearer, as is shown in Figure 9. These results clearly indicate a correlation between the number of matching transactions and the interval being (more or less) a multiple of 52 weeks.

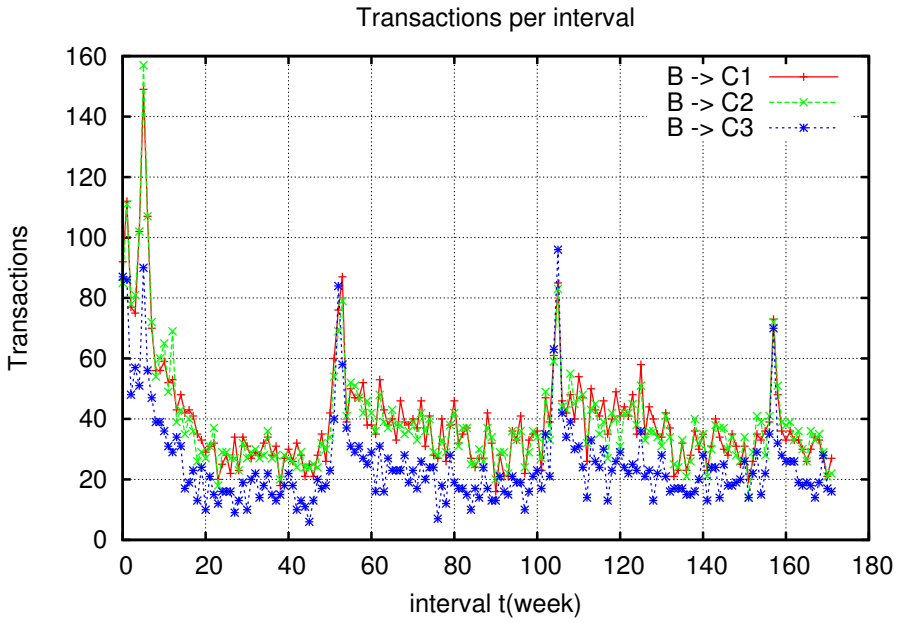

Figure 9: Number of transactions per week-long interval for the patterns $\langle A \rightarrow C \rangle$.

The time-interval extension gives a clear overview of the behaviour of the sequential patterns in different intervals and it shows some remarkable temporal patterns in the data set. However, it only shows the behaviour of the patterns, given a certain interval. The next step would be to use the temporal data in order to weigh the value of a transaction for a pattern. This will be covered in the next section.

4.6 Time-dependent support weighting

In the previous section, the intervals between purchases were used as a way to distinguish results, where the interval was a static constraint, only used to determine whether a transaction should be counted or not, so the support contribution of a transaction is either 0 or 1. This approach can be extended to a continuous support contribution value in the interval $[0, 1]$ by the time-dependent

support weighting as defined in Section 3.5.2.

With this extension of the GSP algorithm, the number and order of the results can be influenced by choosing a weighting function and maximum interval corresponding to the expected results. For example, when taking the global interval length as presented in Figure 7 into account, we can decide to focus on shorter intervals, as they contain the most transactions. In that case, support weighting functions f_2 or f_3 could be a good choice. The weighting functions are defined with respect to a maximum interval which has to be chosen as well.

An overview of the behaviour of all weighting functions is given in Figure 10. This diagram shows the support functions from Section 3.5.2 in the horizontal axis. The function f_5 has an extra parameter n, defining the number of steps, and the results for four values $\{2, 4, 6, 8\}$ are given. The vertical axis shows the order of the plotted patterns in the range $[1, 15]$. Location 1 for f_1 contains the pattern with the highest support for this weighting function, location 2 contains the pattern with the next highest support, etc. The plotted points are the set of sequential patterns for all weighting functions, where the 15 patterns with highest support for every function are shown. The plots give an insight in the relation between the weighting function and the support value for a pattern. It might have been more intuitive to use the support results for each pattern and weighting function on the vertical axis, but this would make the diagram much more difficult to read, as the support value for often occurring patterns would be a lot higher than the support value of other patterns, no matter what weighting function we would choose. By using the location on the ordered list of results, this bias is omitted.

This diagram shows a clear relation between the order of the patterns and the weighting functions. There is a large difference between f_1 to f_5 and f_6 and f_7. This is as expected, because the last two functions are the only ones not focussed on the transactions with shorter intervals. The results for f_5 with various values for n are remarkable similar to each other. The reason for this might be because all of the instances of f_5 handle the very short intervals with $t \to 0$ identical, returning the maximum support contribution. The rules which had a high support in earlier results get a large part of this support from transactions with very short intervals and a higher number of steps n for f_5 should therefore have limited influence on the support for these rules.

For the other weighting functions f_1 to f_4, there is a lot of variation in the order of the patterns, but some clear overall trends are visible. First of all, the rule with the highest support, $\langle C1 \to C2 \rangle$, is on position 1 for all these rules. On the second position for most rules is $\langle B \to A \rangle$, and it is on position 1 for the last two rules. The overall position of $\langle B \to A \rangle$, even for f_6 and f_7, shows that this pattern is a very general occurring one, regardless of the length of the interval. The only different position is for f_4, a weighting function with a very strong focus on the really short intervals, where support contribution drops sharply for longer intervals. Therefore, we see that for f_4, most high positions are for C products, who typically have a lot of intervals with $t = 0$, so for practical cases where the focus is strongly one same-day-purchases, f_4 is a good choice.

The results for the last functions f_6 and f_7 really deviate from the others, but together they are very alike, especially for the higher positions. However, both functions focus on different intervals, as f_7 focusses on the mean of the interval between 0 and the maximum, where f_6 has a strong focus on the highest intervals. Therefore, we can conclude that the rules on the first four positions for

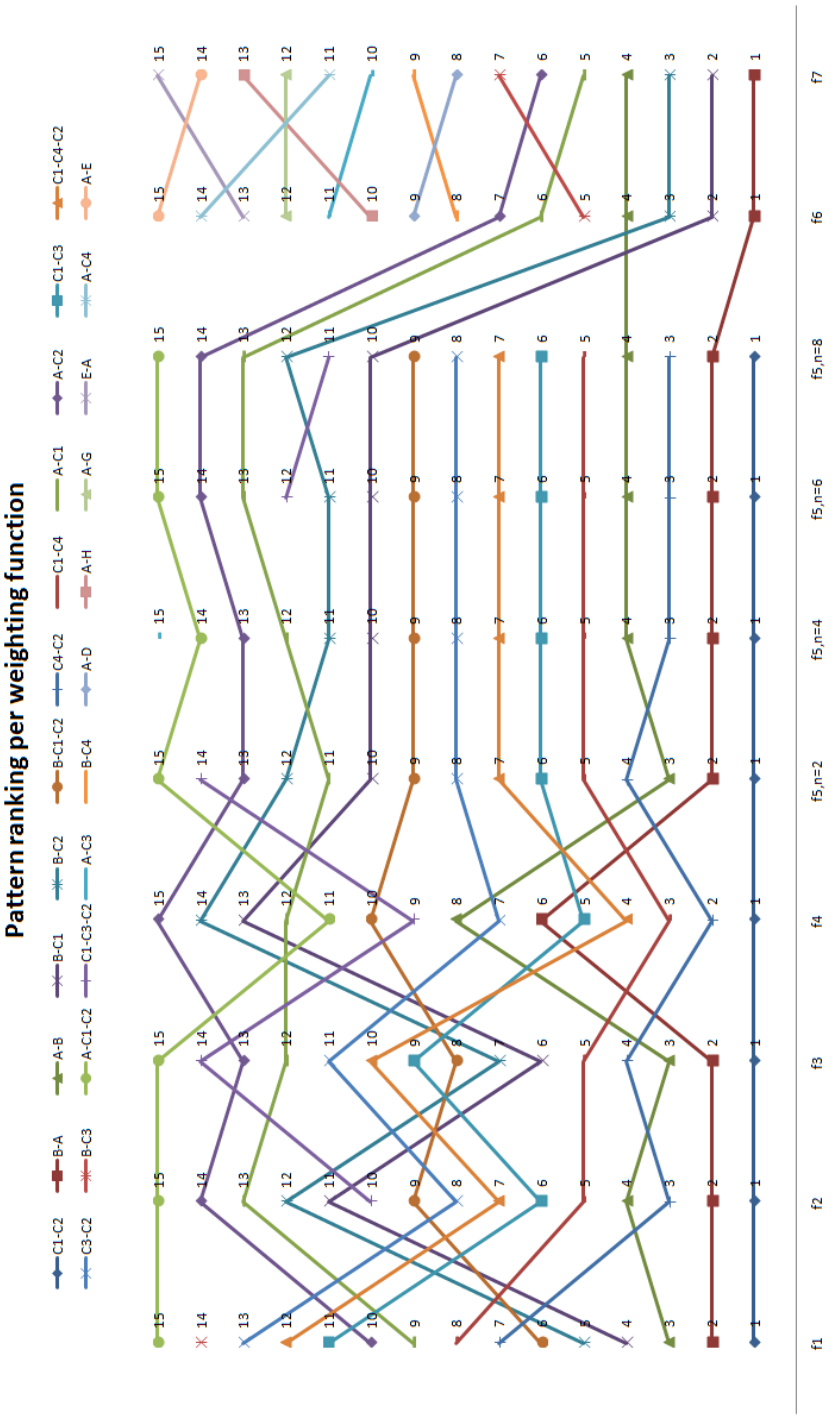

Figure 10: Patterns ordered by suppport per weighting function. Picture created with MS Excel.

f_6 and f_7 appear quite evenly distributed over all intervals. However, the results of these functions are strongly dependent on the defined maximum interval. If the maximum interval is very short, the results will be similar to f_4 or f_5, but for larger maxima, there will eventually be no results left, as there is a transaction that has the largest interval in the dataset. If the defined maximum interval is much larger than the largest found interval, the support result will close to 0. For certain practical cases, such as the discovery of outliers with very long intervals, this can be useful, but in general the maximum interval should be chosen carefully and kept as small as possible.

In order to gain a deeper insight in the results of f_7 over longer intervals, we investigated the behaviour of the function for a series of intervals, for the intervals $(0, 365]$ to $(0, 3650]$, where each interval is one year longer than the previous one, as is depicted in Figure 11.

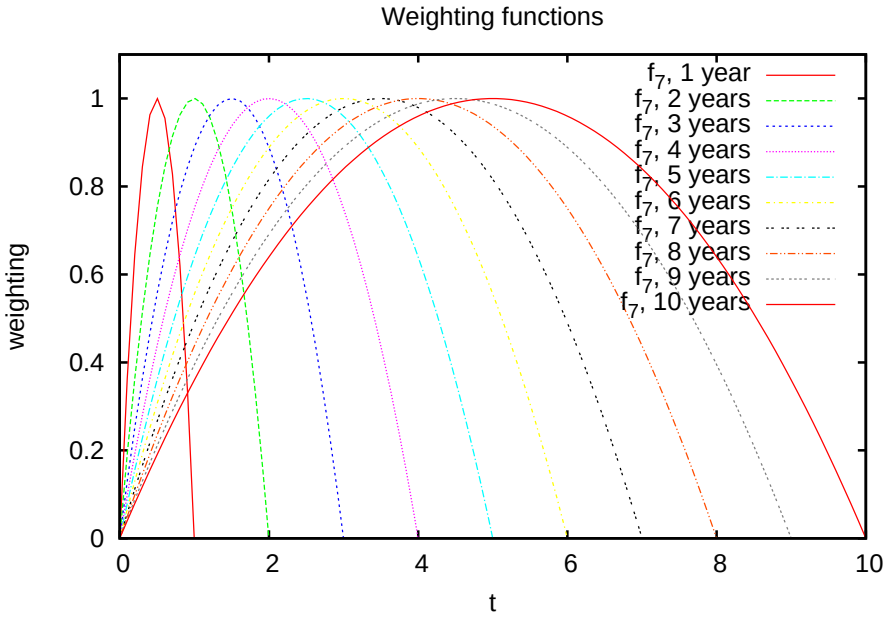

Figure 11: Behaviour of f_7 for longer intervals.

The results of this experiment are shown in Figures 12 and 13. Figure 12 shows the support values for the listed patterns for each of the intervals. As each interval is larger than the previous one, the support value for all patterns gets higher for larger intervals, as there are simply more transactions that fit within the interval. The often occurring products A and B appear in many of the intervals, and especially the pattern (A, B) has a very high support value, which is consistent with previous findings. For the largest intervals, however, we can see that the support values for most patterns have a maximum and start to decrease. Clear examples are $(B, C3)$ and $(C1, A)$ which both have their maximum for the interval $(0, 6]$ years, where the focus of the weighting function is around 3 years. This decrease is due to the fact that the data set contains a lot of transactions with purchase intervals under 4 years. Therefore, if the focus of the weighting function is on larger intervals, the effect of the short-interval-transactions will

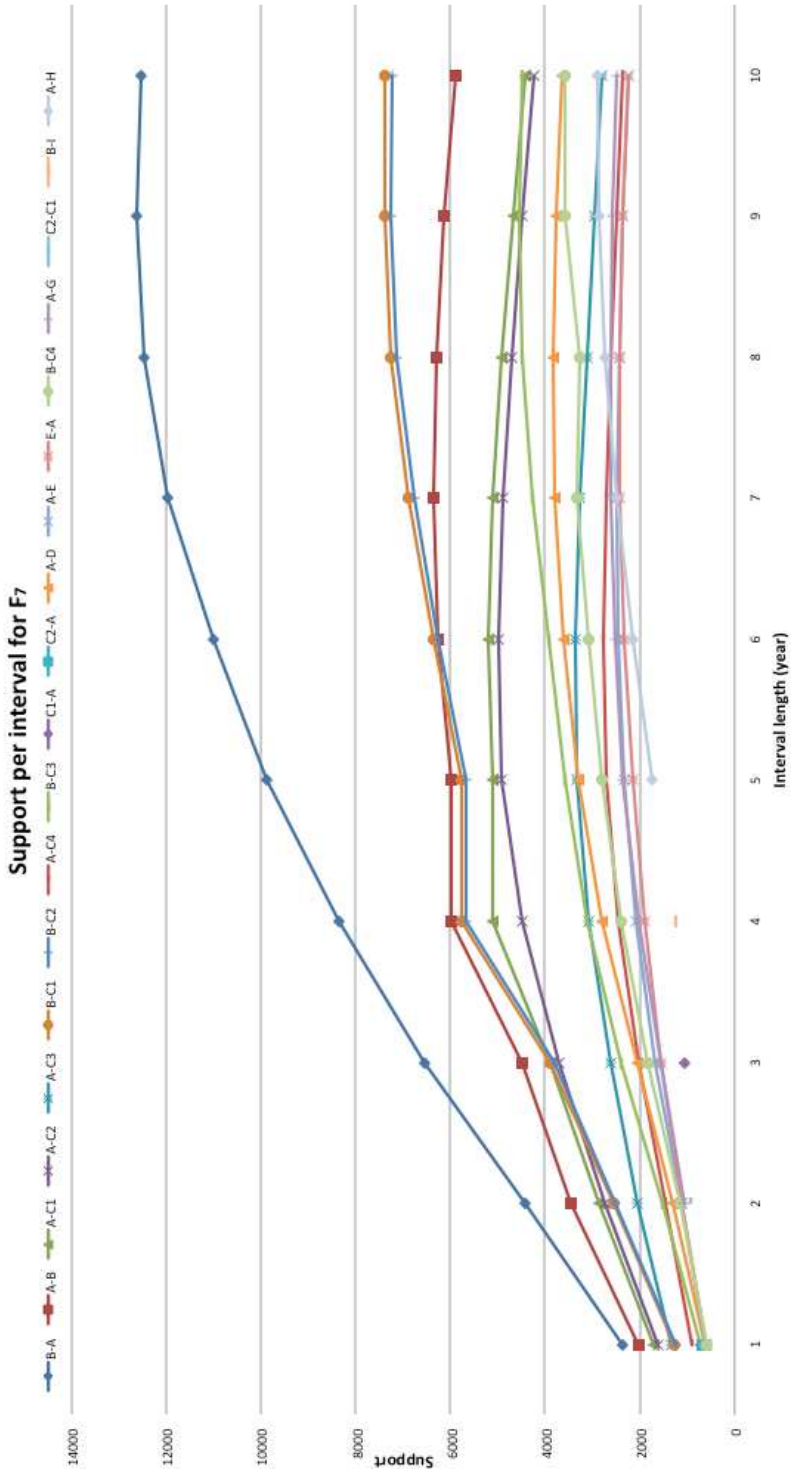

Figure 12: Patterns for f_7 per interval. Picture created with MS Excel.

be less and there be few transactions with larger purchase intervals, so overal the support value will decrease.

In Figure 13 the real support values are omitted. The diagrams shows the same patterns for each of the intervals, but instead of the support values there is a ranking for the patterns. The pattern on location 1 is the pattern with the highest support, location 2 has the second highest support, etc. This diagram gives a better picture of the relative behaviour of the patterns. As we can see, the pattern (A, B) is on location 1 for all intervals, but other patterns show more interesting behaviour. Some of the patterns get lower positions for larger intervals, for example $(A, C3)$, $(A, C1)$ and (A, B). Apparently these patterns occur often in shorter intervals, but less in larger intervals. There also are patterns with high support for large intervals, who appear a lot less often in shorter intervals, for example $(B, C2)$, $(B, C3)$ and (A, H).

In conclusion, using weighted support functions for the evaluation of sequential patterns can be very useful, but it is strongly dependent on what are considered good results. Support weightings can be used to improve results when substantial knowledge of the dataset is already available from previous experiments. In these cases, a proper weighting function and maximum interval can be defined, in order to filter on important intervals, while keeping the effect of the support a pattern has in the unfiltered database.

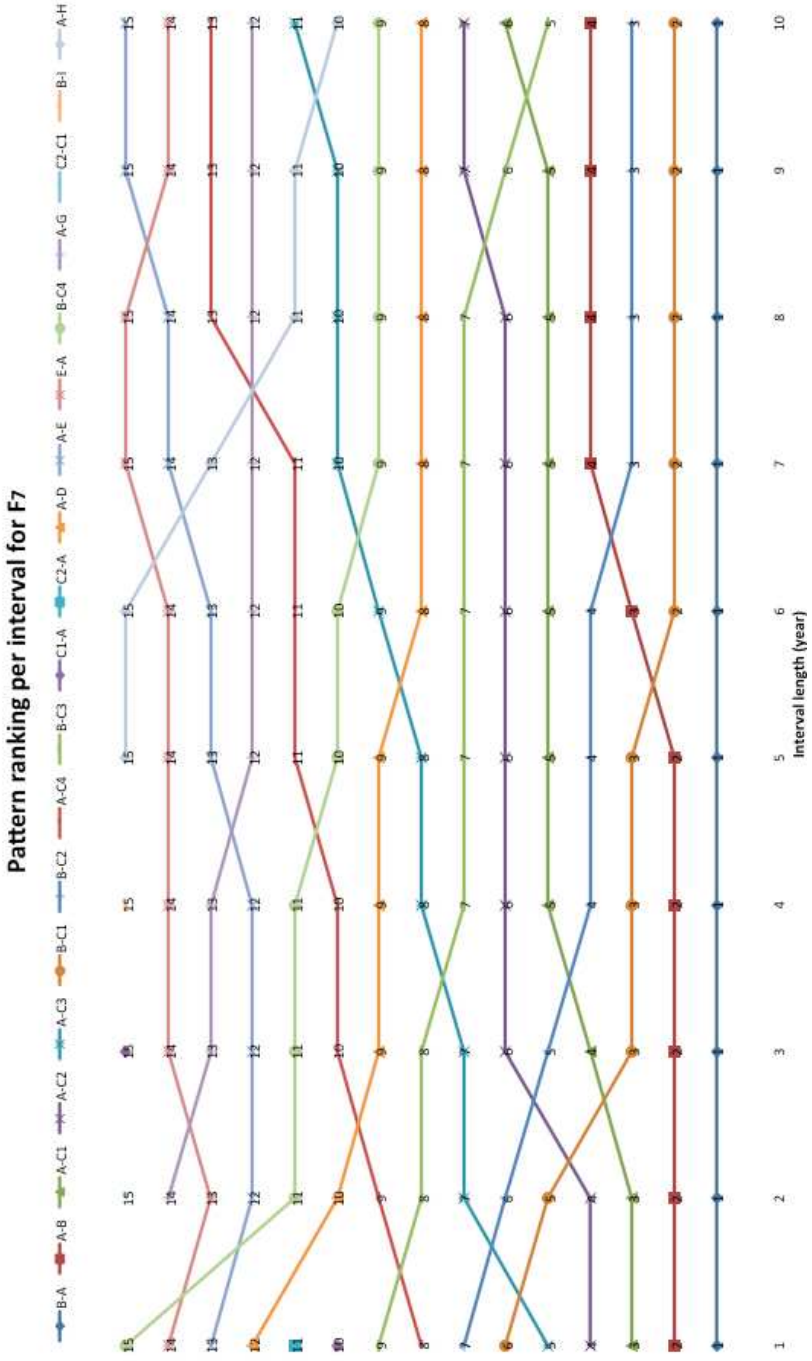

Figure 13: Patterns for f_7 ordered by suppport per interval. Picture created with MS Excel.

5 Conclusions

This report has given an overview of the application of association and pattern generating algorithms on real world insurance data. First, the Apriori-algorithm was introduced to analyse association rules in the data set, then the GSP-algorithm was presented, extending the first algorithm in order to analyse sequential patterns and gain insight into the purchase order of the products. Finally, the GSP-algorithm was extended to handle time-intervals for patterns. As first extension, fixed size time intervals were introduced, in order to expose the distribution of the patterns over time-intervals of various length. As a second extension, a weighted support calculation was introduced, in order to filter the patterns on predefined constraints, such as a focus on short or very long intervals.

The results as presented in this report give a clear impression of the data set. We have seen that working with real world data, can be challenging. Starting from the association rules and sequential patterns, we learned that although de data set is large, it still is quite sparse. The number of products is limited and some of the products are tightly coupled. This behaviour is clearly visible in the associations and patterns that were generated. The use of fixed time intervals shows the validity of the obtained patterns over intervals of various length and can give valuable insight in the behaviour of patterns over longer periods of time. Finally, the support weighting functions provide us with a method to focus the results on patterns or intervals that are considered interesting from the point of view of the user. The use of time-intervals is clearly dependent on the history that is available in the data set. For our data set, the oldest policies are about 10 years old, so that gives a clear limit on the intervals.

5.1 Future work

Beyond this thesis, a lot can be improved. A case study always is a limited experiment and behaviour might be different for other data sets. The Apriori- and GSP-algorithms are proven and trusted algorithms, but the time interval and support weighting extensions should be evaluated on other data sets. The current study only gives the results for one very sparse data set and the behaviour for data sets with other characteristics might be a very different.

Beside that, in order for the support weighting to have a practical value, a set of standard weighting functions for particular patterns or desired results should be composed. In this study, a number of candidate weighting functions is proposed and evaluated, but it is easy to come up with very different weighting functions. Therefore, a framework of weighting functions should be developed, providing links between desired result, for example a focus on short intervals, or a focus on intervals of exactly one year, and corresponding weighting functions.

Another direction for future work, from a more general Data Mining perspective, is the use of more information about the customer. In this study, the only things we know about the customer are what products he or she purchased at what time. However, usually much more information is available, for example the age and sex of the customer. It can be very useful for insurance companies who want to put the generated sequential patterns to practical use, to have a general profile of the customer and thus to know which group of customers should be targeted with ads or special offers.

As a last remark, the algorithms and extensions used in this study are not

the most efficient or sophisticated available. For our data set, performance and efficiency where no key issues, but for other data sets or real world applications, this might be a problem. Therefore, the presented extensions should be applied to other, more efficient association and pattern generation algorithms. Apart from that, the extension itself can be probably be improved in terms of performance or efficiency as well. Especially the weighted time-interval extension can be improved, as the candidate generation step is very unefficient, due to the fact that anti-monotonicity cannot be used for the pruning of candidates. Therefore, another way of efficient candidate generation should be constructed.

References

[1] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data (SIGMOD '93)*, pages 207–216, 1993.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. *Proceedings of the Eleventh International Conference on Data Engineering (ICDE '95)*, pages 3–14, 1995.

[3] C. Apte, E. Bibelnieks, R. Natarajan, E. Pednault, F. Tipu, D. Campbell, and B. Nelson. Segmentation-based modeling for advanced targeted marketing. *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '01)*, pages 408–413, 2001.

[4] M. Bastian, S. Heymann, and M. Jacomy. Gephi: An open source software for exploring and manipulating networks. *International AAAI Conference on Weblogs and Social Media*, 2009.

[5] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data (SIGMOD '97)*, pages 255–264, 1997.

[6] P. L. Brockett, X. Xia, and R. A. Derrig. Using Kohonen's self-organizing feature map to uncover automobile bodily injury claims fraud. *The Journal of Risk and Insurance*, 65(2):245, 1998.

[7] Y. Chen. Discovering time-interval sequential patterns in sequence databases. *Expert Systems with Applications*, 25(3):343–354, 2003.

[8] A. Dal Pozzolo. *Comparison of Data Mining Techniques for Insurance Claim Prediction*. PhD thesis, University of Bologna, 2010.

[9] A. B. Devale. Applications of data mining techniques in life insurance. *International Journal of Data Mining & Knowledge Management Process*, 2(4):31–40, 2012.

[10] S. Dolnicar. Using cluster analysis for market segmentation — typical misconceptions, established methodological weaknesses and some recommendations for improvement. *Australasian Journal of Marketing Research*, 11(2):5–12, 2003.

[11] P. E. Green. A new approach to market segmentation. *Business Horizons*, 20(1):61–73, 1977.

[12] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *ACM SIGMOD Record*, 29(2):1–12, 2000.

[13] M. van Leeuwen and A. Knobbe. Non-redundant subgroup discovery in large and complex data. *Proceedings of the 2011 European Conference on Machine Learning and Knowledge Discovery in Databases, LNAI 6913*, pages 459–474, 2011.

[14] J. Pei, J. Han, and L. V. S. Lakshmanan. Mining frequent itemsets with convertible constraints. *Proceedings of the 17th International Conference on Data Engineering (ICDE '01)*, pages 433–442, 2001.

[15] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. *Proceedings of the Fifth International Conference on Extending Database Technology*, pages 3–17, 1996.

[16] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley, 2007.

[17] P. C. Verhoef and B. Donkers. Predicting customer potential value — an application in the insurance industry. *Decision Support Systems*, 32(2):189–199, 2001.

[18] S. Viaene, G. Dedene, and R. A. Derrig. Auto claim fraud detection using Bayesian learning neural networks. *Expert Systems with Applications*, 29(3):653–666, 2005.

[19] S. Viaene, R. A. Derrig, B. Baesens, and G. Dedene. A comparison of state-of-the-art classification techniques for expert automobile insurance claim fraud detection. *Journal of Risk & Insurance*, 69(3):373–421, 2002.

[20] G. I. Webb. Efficient search for association rules. *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '00)*, pages 99–107, 2000.

[21] A. C. Yeo, K. A. Smith, R. J. Willis, and M. Brooks. Clustering technique for risk classification and prediction of claim costs in the automobile insurance industry. *International Journal of Intelligent Systems in Accounting, Finance & Management*, 10(1):39–50, 2001.

[22] Z. M. Yunos, S. M. Shamsuddin, N. Ismail, and R. Sallehuddin. Modeling the Malaysian motor insurance claim using artificial neural network and adaptive neurofuzzy inference system. *Proceedings of the 20th National Symposium on Mathematical Sciences*, pages 1431–1437, 2013.

[23] M. Zaki. Sequence mining in categorical domains: Incorporating constraints. *Proceedings of the 9th International Conference on Information and Knowledge Management*, pages 422–429, 2000.