Displaying Co-occurrences of Patterns in Streams for Website Usage Analysis

Edgar H. de Graaf Joost N. Kok Walter A. Kosters

Leiden Institute of Advanced Computer Science
Leiden University, The Netherlands
edegraaf@liacs.nl

Abstract

One way of getting a better view of data is by using frequent patterns. In this paper frequent patterns are (sub)sets that occur a minimal number of times in a stream of itemsets. However, the discovery of frequent patterns in streams has always been problematic. Because streams are potentially endless it is harder to say if a pattern is frequent or not. Furthermore, the number of patterns can be huge and a good overview of the structure of the stream is lost quickly. The proposed approach will use competitive neural network methods to online model pattern co-occurrence in a stream of itemsets.

A model of the co-occurrence of patterns will give the user an improved view on the structure of the stream. Some patterns might occur so often together that they should form a combined pattern. In this way the patterns in the clustering will approximate the largest frequent patterns: maximal frequent patterns. The number of (approximated) maximal frequent patterns is much smaller and combined with methods of visualization using competitive neural networks these patterns provide a good view on the structure of the stream.

1 Introduction

Effectively mining streams of data with *frequent patterns*, i.e., patterns occurring at least a minimal number of times, has always been a hard problem to tackle. The difficulty lies in the potential endlessness of the stream; frequent patterns can suddenly become infrequent; and standard ways of pruning the search space are harder to use. In this work *patterns* are sets of items occurring in a record (also called transaction or *itemset*) at a certain moment in time.

This work is motivated by a wish to view pages accessed together by users helping website analysts to improve the website. To this end we will propose a method of modeling co-occurring patterns in a stream of itemsets. Knowing how much patterns co-occur can provide interesting structural information about the stream in an online way. Note that the model is an approximation and due to this the frequent subsets are also approximately maximal.

We will define our method of displaying co-occurring patterns in a stream of itemsets and show its usefulness. Our algorithm keeps track of a condensed and approximate representation of interesting patterns within the stream, and allows for online visualization. This paper makes the following contributions:
— We use a **dynamic support estimation** to determine the support of those itemsets we need, and do this in an online way (Section 2.1).
— It will be **explained how the distance between patterns is approximated** by placing patterns closer (pulling) or further away (pushing) depending on their co-occurrence. If this distance is large, patterns occur almost never together, and otherwise they do have many common occurrences (Section 2.2).
— We will define when patterns **can be merged and when they should be split** to form smaller patterns and how this could be done (Section 2.3).
— Finally through experiments **the effectiveness of our method is shown** and efficiency is discussed (Section 3).

We first mention related work, and in the next section we discuss the algorithm in full detail. Finally we describe experiments and discuss these.

This research is related to work done on visualization of patterns in streams and visualization of website usage using patterns as done in [4]. Also our work is related to (maximal) frequent pattern mining in streams

and large datasets. *Maximal frequent itemsets* are sets of items occurring often in the stream while there is no frequently occurring bigger set of items containing these same items.

There are many algorithms for mining maximal frequent patterns, in "normal" datasets, in different ways. We mention GENMAX discussed in [9] and MAFIA presented in [3]. Large datasets are different from streams in that there is an end to the dataset. One approach to mining large datasets was proposed in [7], where an extremely large dataset is mined for maximal frequent patterns by proceeding in parallel. Furthermore clustering on large datasets was done in [12]. Much work has been performed on mining frequent patterns in (online) data streams, e.g., in [5]. In [6] frequent patterns are mined by using sliding window methods. However it must be said that our work is more concerned with co-occurrence and frequent patterns are approximately maximal. Our work has little overlap with work done on maximal pattern-based clustering as discussed in [13] and [14] where objects basically are clustered by linking attribute groups with object groups when attributes have a minimal similarity. Related research has been done on clustering on streams in [1], where a study on clustering evolving data streams, (fast) changing data streams, is done. Clustering categorical data was also done in [8] where also co-occurrence is used, but only for attribute values; the authors propose a visualization where the x-axis is the column position and the y-axis the distance based on co-occurrence of values.

In this work a method of pushing and pulling points in accordance with a distance measure is used. This technique was used before in [2] to cluster criminal careers and in [10] to cluster association rules. This method of clustering was chosen because it enables us to limit the number of iterations in order to improve online performance while still having results. Furthermore we only know the distance between two patterns, where a low distance means frequent co-occurrence.

2 Model Realization

Our goal is to produce an algorithm that is capable of accepting a stream of records, each record being an unordered finite set of items, meanwhile building a model of patterns and their co-occurrence. We **first** optimize this model by restricting the patterns to frequent patterns, simply because we will have too many otherwise. Our **second** optimization is to restrict patterns to maximal frequent patterns. If we do not use maximal frequent patterns then the model might have too many frequent patterns for a reasonable online performance because all potentially frequent patterns need to be kept.

The algorithm we propose, called DISTANCEMERGESPLIT, starts with randomly positioning n points in a 2-dimensional space, e.g., in the unit square. Note that the axes have no clear meaning. Here n is the number of different items that appear in the dataset. Each of these n points represents one size 1 itemset, where the size of an itemset is of course defined as the number of items it contains. These n points remain present during the whole process, though their coordinates may change. While the records from the data stream pass by, new points are created (by merging or splitting) and others disappear (by merging, or by other reasons). Together these points constitute the evolving model $\mathcal{P}$, where points correspond with frequent itemsets.

We will first explain how we use the stream of records to update the supports of the elements of $\mathcal{P}$, next we describe how the coordinates of the elements change in accordance with the corresponding supports, and finally mention our method of growing and shrinking the number of sets present in $\mathcal{P}$: the merge and split part of the algorithm.

2.1 Support

The algorithm will receive a possibly infinite stream of itemsets, the records: $r_1, r_2, r_3, \ldots$ Each time an itemset corresponding to a point in the space is a subset of a record, we observe an occurrence of this itemset. We count the occurrences in the t records we have seen so far (and that can also be considered as the last t records), and define support:

$$support\ (p,t) = \sum_{i=1}^{t} occurrence\ (p, r_i) \tag{1}$$

$$occurrence\ (p,r) = \begin{cases} 1 & \text{if } p \subseteq r \\ 0 & \text{otherwise} \end{cases}$$

Here p is the pattern, the itemset, for which support is computed, and r is a record. If a new record arrives the support needs to be adapted accordingly. Rather than using the full support for all records, we will make use of a *sliding window* of size $\ell \geq 1$, and we will not keep all data about the occurrences of the patterns in the transactions of this window. Though this is not essential for our algorithm, it has a beneficial influence on the runtime, which is especially interesting for an online algorithm. If we have seen less than ℓ transactions ($t < \ell$) then we *do* use the previous formula to calculate support, in such case a pattern is called "young". This method will also be used when we later create new patterns online, and is referred to as "direct computation". In the other case ($t \geq \ell$) a pattern is called "old" and we give an estimate $support_t(p)$ for the support during the last ℓ records in the following way. When the itemset p is *not* a subset of the current record r_t we adapt the support as follows:

$$\begin{aligned} support_t(p) &= support_{t-1}(p)/\ell \cdot (support_{t-1}(p) - 1) \\ &\quad + (1 - support_{t-1}(p)/\ell) \cdot support_{t-1}(p) \\ &= (1 - 1/\ell) \cdot support_{t-1}(p) \leq support_{t-1}(p) \end{aligned} \tag{2}$$

Indeed, when the first transaction of the window of size ℓ contains the pattern then support should decrease with 1. However, if the first record also does not contain p, then support remains the same. It is important to notice that the probability of a transaction containing p in a window of size ℓ is estimated with $support_t(p)/\ell$. If the new record *does* contain the itemset p then support is adapted as follows:

$$\begin{aligned} support_t(p) &= support_{t-1}(p)/\ell \cdot support_{t-1}(p) \\ &\quad + (1 - support_{t-1}(p)/\ell) \cdot (support_{t-1}(p) + 1) \\ &= (1 - 1/\ell) \cdot support_{t-1}(p) + 1 \geq support_{t-1}(p) \end{aligned} \tag{3}$$

Now when the first transaction of the window of size ℓ contains the pattern then support remains unchanged as the window shifts. However, if it does not contain the pattern p, then support will increase with 1. Both formulas *assume that occurrences are uniformly spread* over the window of size ℓ, but by using these formulas to adapt support we do not have to keep all occurrences for all patterns in the 2-dimensional space. Notice that $0 \leq support_t(p) \leq \ell$ always holds.

We have now described how the stream of records influences the supports of the itemsets that are currently being tracked, i.e., those in $\mathcal{P}$. Note that the itemsets of size 1 are always present in the model $\mathcal{P}$ of co-occurring patterns, for reasons mentioned in Section 2.4. Larger itemsets may appear and disappear as the algorithm proceeds. Also observe that the supports are estimates, due to the application of equations 2 and 3.

2.2 Distance

We now describe how the coordinates of the points change as their supports vary when the new records from the stream come in. In our model for $distance\ (p_1, p_2)$ we take the Euclidean distance between the 2-dimensional coordinates of the points corresponding with the two patterns p_1 and p_2.

These points are pulled closer to one another if they occur in the current transaction and they are pushed apart if not. Furthermore nothing is done if both do not occur. In every time step a random selection of the pairs undergoes this process.

To pull two points together we set the *goal distance* to 0 (when their corresponding patterns co-occur) and to push them apart the goal distance is $\sqrt{2}$ (when their corresponding patterns do not co-occur), which is the maximum Euclidean distance between any two points in the unit square. These distances are then used to update the coordinates (x_{p_1}, y_{p_1}) and (x_{p_2}, y_{p_2}) of the points corresponding with the itemsets p_1 and p_2:

1. $x_{p_1} \leftarrow x_{p_1} - \alpha \cdot (distance\ (p_1, p_2) - \gamma) \cdot (x_{p_1} - x_{p_2})$
2. $y_{p_1} \leftarrow y_{p_1} - \alpha \cdot (distance\ (p_1, p_2) - \gamma) \cdot (y_{p_1} - y_{p_2})$
3. $x_{p_2} \leftarrow x_{p_2} + \alpha \cdot (distance\ (p_1, p_2) - \gamma) \cdot (x_{p_1} - x_{p_2})$
4. $y_{p_2} \leftarrow y_{p_2} + \alpha \cdot (distance\ (p_1, p_2) - \gamma) \cdot (y_{p_1} - y_{p_2})$

Here α ($0 \leq \alpha \leq 1$) is the user-defined learning rate and γ ($0 \leq \gamma \leq \sqrt{2}$) is the goal distance.

Now the points in the model are moved so that their Euclidean distance corresponds to their measure of co-occurrence (where distance 0 means patterns occur always together). We not only use the distances

to place the patterns in the 2D space, but also to *decide when to merge*. Points may leave the unit square; however, when presenting the results of the experiments, such points are projected on the nearest wall of this square.

2.3 Merge and Split

Now we describe how we merge and split the itemsets of the model as time goes by. The model $\mathcal{P}$ contains points with corresponding itemsets. Two old patterns (itemsets) are assumed to occur many times together when their distance is small due to them being pulled together. In some cases one itemset can be made that represents two of them: the algorithm will try these combinations. For some combinations it is possible that they turn out to be not so good, their frequency is smaller than $minsupp$, where $minsupp$ is a user-defined threshold. This can happen when their combined frequency is lower than $minsupp$ or suddenly frequency drops below $minsupp$. In either case we need to split the size k itemset into k itemsets of size $k-1$, all being subsets of the original itemset. Later we will discuss splitting in more detail, we now first explain merging.

As transactions come in, some of the initial size 1 itemsets become *frequent*, meaning that the support is higher than $minsupp$. These sets can — under certain circumstances, see below — merge to itemsets of size 2, and so on: we **merge** two itemsets p_1 and p_2 if (in the algorithm in Section 2.4 the following series of conditions is referred to as "appropriate"):

- The patterns p_1 and p_2 are old enough: they exist in $\mathcal{P}$ for at least ℓ (the window size) records. (Note that the supports of these sets are currently updated through equations 2 and 3 above.)
- The two itemsets p_1 and p_2 currently are frequent, i.e., it holds that both $support_t(p_1) \geq minsupp$ and $support_t(p_2) \geq minsupp$. (Note that this condition automatically holds for all (pairs of) itemsets in $\mathcal{P}$ that have size larger than 1.)
- The itemsets are close together in the model, so they are assumed to occur often together as a subset of transactions in the stream: $distance\ (p_1, p_2) \leq mergedist$, where $mergedist$ is a user-defined upper bound for the distance for which merging p_1 and p_2 is allowed.
- The pattern p_2 has an item i_p which is not in the pattern p_1, such that $p_2 \setminus \{i_p\} \subseteq p_1$. This condition always holds if p_2 has size 1.)

First of all we merge the patterns p_1 and p_2 if they are of equal size, so we create the set $p_1 \cup p_2$ and add it to $\mathcal{Q}$, the collection of all newly formed patterns. Both original patterns are removed from the 2-dimensional space except if their size is 1.

The **second** time we merge patterns is if pattern p_1 contains more items than p_2 and $p_2 \setminus \{i_p\} \subseteq p_1$ for some $i_p \in p_2$ with $i_p \notin p_1$, then for each item $e \in p_1 \setminus p_2$ we add an itemset $p_2 \cup \{e\}$ to $\mathcal{Q}$. This enables patterns to be merged with patterns that already were merged before and disappeared from the model. The smaller pattern p_2 is removed except if it is of size 1.

Next we **split** patterns, when they contain more than one item, if they do not occur often enough and they have been in the model for at least a certain number of records (they are "old enough"). Split combinations are generated by removing each item from the original pattern once. The remaining items form one new itemset, so in this way a size k itemset will result in k combinations after splitting.

Assume we have the pattern p that is split into patterns $q_0, q_1, \ldots, q_{|p|-1}$ that are added to $\mathcal{Q}$:

$$split : p = \{i_0, i_1, \ldots, i_{|p|-1}\} \rightarrow q_0 = \{i_1, i_2, \ldots, i_{|p|-1}\},$$
$$q_1 = \{i_0, i_2, i_3, \ldots, i_{|p|-1}\}, \ldots, q_{|p|-1} = \{i_0, i_1, \ldots, i_{|p|-2}\}$$

Finally, the newly formed patterns in $\mathcal{Q}$ are united with those in $\mathcal{P}$. Of course, when patterns occur more than one time, only one copy — the oldest one — is maintained. And those patterns from $\mathcal{P}$ that are contained in a larger one in $\mathcal{P}$ are removed, unless — as stated above — they have size 1: we focus on the maximal patterns.

2.4 The Algorithm

The algorithm works with the set $\mathcal{P}$ of patterns that are currently present, represented by (coordinates of) points in 2-dimensional Euclidean space. The outline of the algorithm DISTANCEMERGESPLIT is as follows:

initialize $\mathcal{P}$ with the n itemsets of size 1
for $t \leftarrow 1$ to ∞ **do**
  $\mathcal{Q} \leftarrow \emptyset$
  for all patterns $p \in \mathcal{P}$ **do**
    compute $support_t(p)$ using the t^{th} record r_t,
    either through updating (old patterns)
    or by direct computation (young ones)
  for a random subset of pairs of patterns in $\mathcal{P}$ **do**
    update their distance according to their support
  for all "appropriate" pattern pairs in $\mathcal{P}$ **do**
    merge the pair, creating (new) pattern(s) in $\mathcal{Q}$
    mark the smallest of the pair,
    or both if their sizes are equal
  remove the marked patterns from $\mathcal{P}$
  for all patterns $p \in \mathcal{P}$ **do**
    if p is infrequent and old enough **then**
      split p into (new) patterns in $\mathcal{Q}$
      remove p from $\mathcal{P}$
  $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{Q}$, joining duplicates
  remove non-maximal frequent patterns from $\mathcal{P}$

DISTANCEMERGESPLIT

Note that itemsets of size 1 are *never* removed from $\mathcal{P}$, not even when they are infrequent. The size 1 itemsets are always present, and play a special role: besides the fact that some of them are frequent, they also serve as building blocks. In many cases they are not maximal. If they were removed, it could be impossible to re-introduce single items after having become infrequent.

Patterns that are new in $\mathcal{P}$ are called "young". When computing supports for these patterns, we use equation 1, when updating the "old" ones we use equations 2 and 3. So, each pattern present in $\mathcal{P}$ also has an *age*: patterns that have an age smaller than the window size ℓ are "young", the others are "old".

On two occasions the algorithm introduces indeterminism: first, when the support computation is done using the approximating updates for "old" patterns (saving a lot of time and memory) and second, when pushing and pulling pairs of points representing a pattern, see Section 2.2.

3 Experiments and Discussion

The experiments are organized such that we first show the method at work in a few controlled synthetic cases. Then we will use the algorithm to build a model for real datasets, showing "real life" results. The first synthetic experiment will be a stream with 10 groups of 5 items. Groups do not occur together, but all of them occur often. This dataset is called the `10-groups` dataset. The second synthetic experiment will be a stream where certain groups of items suddenly do not occur; instead another group starts occurring. We call this dataset the `sudden change` dataset.

The first real dataset comes from Internet Information Server (IIS) logs for `msnbc.com` and news-related portions of `msn.com` for the entire day of September, 28, 1999. The original dataset contained sequences of 17 possible categories viewed by a user within 24 hours and was used before in [4]. For our purpose we converted the dataset to itemsets. We removed users viewing only one or two categories to make the problem more interesting. This dataset will be called the `MSNBC` dataset with 174,042 transactions.

The second real dataset is the Large Soybean Database used for soybean disease diagnosis in [11], we call the dataset the `soybean` dataset. This dataset contains 683 records with 35 attributes. First we removed all missing values and we converted each record to a string of $n = 84$ yes/no values for each attribute value. In this research we do not deal with missing values, and each item represents an attribute value. We use this dataset to analyze the performance of our algorithm with a real dataset with more than 50 items.

All experiments were performed on an Intel Pentium 4 64-bits 3.2 Ghz machine with 3 GB memory. As operating system Debian Linux 64-bits was used with kernel 2.6.8-12-em64t-p4.

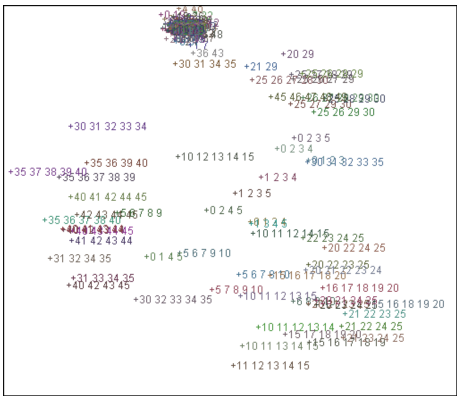

Figure 1: Model after seeing 1,200 transactions of the `10-groups` dataset ($n = 50$, $minsupp = 0.05$, $\ell = window\ size = 300$, $mergedist = 0.1$, $\alpha = 0.1$).

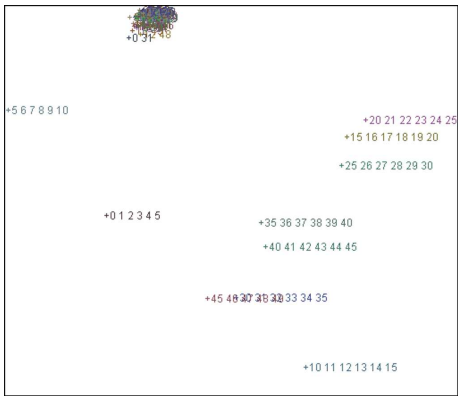

Figure 2: Model after seeing 4,500 transactions of the `10-groups` dataset ($n = 50$, $minsupp = 0.05$, $\ell = window\ size = 300$, $mergedist = 0.1$, $\alpha = 0.1$).

Figures 1 and 2 show how the cluster model changes as more transactions are coming in for the `10-groups` dataset. The first group of this dataset consists of items 0 to 5, the second has 5 to 10, etc. In Figure 2 we clearly see these patterns, where $minsupp$ is given as a percentage of the dataset size. Furthermore notice that both the second and the first group contain the item 5, so there is a slight overlap. We see these itemsets closer together because they are both close to the pattern $\{5\}$. In order to get a clear picture we did not display the size 1 itemsets. Itemsets are plotted using $+$s, accompanied by the items they contain.

The second synthetic dataset, called the `sudden change` dataset, simulates a stream that completely changes after seeing many transactions (i.e., 30,000). The results are displayed in Figure 3, where the labels above each bar reveal the size of the itemsets. First the records in the stream always contain items 1 to 5. Then after 30,000 transactions they only contain items 25 to 30. Figure 3 shows how the first pattern appears and how it slowly disappears in the middle. In the end the model contains only patterns with items 25 to 30.

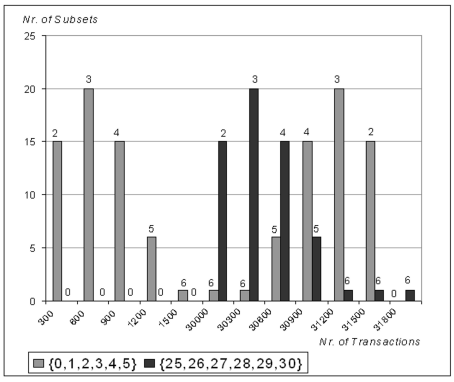

Figure 3: The `sudden change` dataset, the stream changes in the middle ($n = 50$, $minsupp = 0.05$, $\ell = window\ size = 300$, $mergedist = 0.1$, $\alpha = 0.1$).

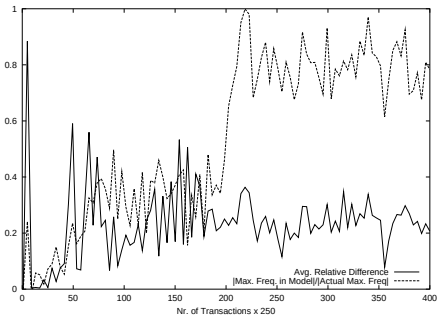

Figure 4: The model compared with the actual situation for the `MSNBC` dataset ($n = 17$, $minsupp = 0.05$, $\ell = window\ size = 1,000$, $mergedist = 0.1$, $\alpha = 0.1$).

Figure 4 was made using the formula $\frac{1}{|\mathcal{P}|} \cdot \sum_{i=1}^{|\mathcal{P}|} abs(|p_i| - |rmax(p_i)|)/|rmax(p_i)|$ for each model $\mathcal{P}$, we call this value the *average relative difference*, where the most optimal value is 0. Here $rmax$ gives the itemwise nearest maximal frequent pattern with $p_i \in \mathcal{P}$ as a subset. These maximal frequent patterns are beforehand decided with a frequent itemset miner. In short this formula calculates how itemsets in the model (itemwise) differ from the actual maximal frequent patterns. Figure 4 displays how the average relative difference stabilizes around 0.2. We also plot the number of maximal frequent patterns divided by the actual number, where 1.0 means they are equal in size. This value approaches 1.0 especially when merging and splitting is temporarily stopped after 50,000 transactions, suggesting a good fit.

Approximating supports well is important in order to know which itemsets should be split. In Figure 5

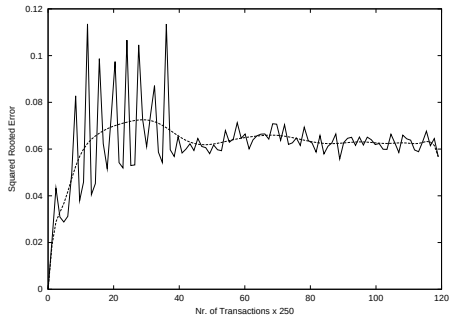

Figure 5: Root squared error between the real support and the approximated support for the MSNBC dataset ($n = 17$, $minsupp = 0.05$, $\ell = window\ size = 1,000$, $mergedist = 0.1$, $\alpha = 0.1$), with a Bézier curve.

we show for all patterns in a computed model the error between their approximated support and their real support in the time window as the transactions from the MSNBC dataset arrive. The root mean squared error of the supports for this model eventually approaches 0.06. The error becomes more stable after temporarily stopping itemset creation after seeing 10,000 transactions.

The processing time of the algorithm strongly depends on the support threshold $minsupp$ one chooses. The lower $minsupp$ is chosen the more points the model will contain eventually and so processing time will get worse. Figure 6 shows that the average processing time for each transaction gets worse as the model contains more itemset points. However, Figure 7 shows that, for the soybean dataset, the number of points in the model eventually stabilizes. For each transaction we adapt the distances between points a number of times. In the case of the soybean dataset we randomly choose pairs 40,000 times in order to push or pull them, depending on their co-occurrence. Obviously one way of speeding up processing is to make it less than 40,000 times or one can skip adapting distances sometimes.

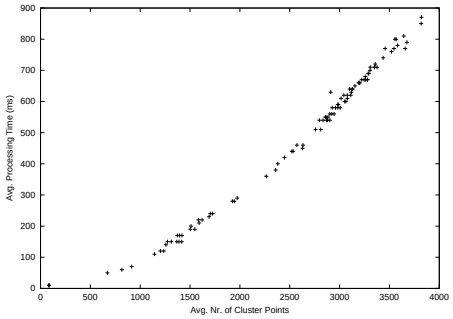

Figure 6: Transaction processing time in milliseconds for different model sizes for the real dataset ($n = 84$, $minsupp = 0.2$, $\ell = window\ size = 300$, $mergedist = 0.1$, $\alpha = 0.1$).

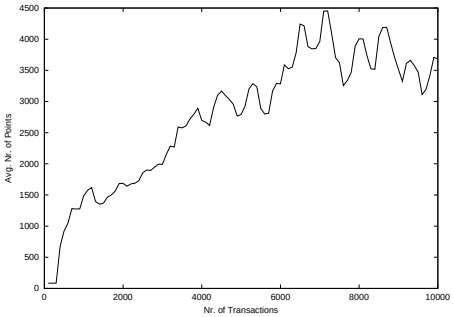

Figure 7: Development of model size as transactions of the real dataset are processed ($n = 84$, $minsupp = 0.2$, $\ell = window\ size = 300$, $mergedist = 0.1$, $\alpha = 0.1$).

4 Conclusions and Future Work

The algorithm presented in this paper will generate a co-occurrence model of approximately maximal frequent itemsets. This gives the user a quick view on the patterns, frequent subsets, in the stream and how they occur in the stream. In this way analysts can online see pages accessed together or not at all.

The co-occurrence distance of patterns is computed by pushing apart or pulling together patterns in a 2-dimensional space. Pushing was done when only one of the patterns occurs and pulling if they occur together. This distance is used to merge sufficiently long existing patterns together if support is larger than a user-defined threshold, because we want only maximal frequent itemsets (itemsets that are often a subset of a transaction but they are never a subset of a bigger frequent itemsets) such that the model does not grow too big. Finally points are split if they happen to occur less than expected. Splitting and merging is required because the model cannot contain all patterns.

In the future we want to focus more on the applications of our algorithm and how it is best used in the analysis of streams. Furthermore we like to examine the support estimates in more detail, and see how extra parameters (e.g., to determine the threshold age for splitting) can be employed.

5 Acknowledgment

This research is carried out within the Netherlands Organization for Scientific Research (NWO) MISTA Project (grant no. 612.066.304).

References

[1] C.C. Aggarwal, J. Han, J. Wang, and P.S. Yu. A framework for clustering evolving data streams. In *29th International Conference on Very Large Data Bases (VLDB'03)*, pages 81–92, 2003.

[2] J.S. de Bruin, T.K. Cocx, W.A. Kosters, J.F.J. Laros, and J.N. Kok. Data mining approaches to criminal career analysis. In *6th IEEE International Conference on Data Mining Proceedings (ICDM 2006)*, pages 171–177, 2006.

[3] D. Burdick, M. Calimlim, and J. Gehrke. MAFIA: A maximal frequent itemset algorithm for transactional databases. In *17th International Conference on Data Engineering (ICDE'01)*, pages 443–453, 2001.

[4] I.V. Cadez, D. Heckerman, C. Meek, P. Smyth, and S. White. Visualization of navigation patterns on a web site using model-based clustering. In *Knowledge Discovery and Data Mining*, pages 280–284, 2000.

[5] J.H. Chang and W.S. Lee. Finding recent frequent itemsets adaptively over online data streams. In *9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)*, pages 487–492, 2003.

[6] J.H. Chang and W.S. Lee. estWin: Online data stream mining of recent frequent itemsets by sliding window methods. *Journal of Information Science*, 31(2):76–90, 2005.

[7] M. El-Hajj and O.R. Zaiane. Parallel leap: Large-scale maximal pattern mining in a distributed environment. In *12th International Conference on Parallel and Distributed Systems (ICPADS'06)*, pages 135–142, 2006.

[8] D. Gibson, J. Kleinberg, and P. Raghavan. Clustering categorical data: An approach based on dynamical systems. In *26th International Conference on Very Large Data Bases (VLDB'00)*, pages 222–236, 2000.

[9] K. Gouda and M.J. Zaki. Efficiently mining maximal frequent itemsets. In *IEEE International Conference on Data Mining (ICDM'01)*, pages 163–170, 2001.

[10] W.A. Kosters and M.C. van Wezel. Competitive neural networks for customer choice models. *E-Commerce and Intelligent Methods of Studies in Fuzziness and Soft Computing*, 105:41–60, 2002.

[11] R.S. Michalski and R.L. Chilausky. Learning by being told and learning from examples: An experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis. *International Journal of Policy Analysis and Information Systems*, 4(2):125–160, 1980.

[12] A. Nanopoulos, Y. Theodoridis, and Y. Manolopoulos. C^2P: Clustering based on closest pairs. In *27th International Conference on Very Large Data Bases (VLDB'01)*, pages 331–340, 2001.

[13] J. Pei, X. Zhang, M. Cho, H. Wang, and P.S. Yu. MaPle: A fast algorithm for maximal pattern-based clustering. In *3th IEEE International Conference on Data Mining (ICDM'03)*, pages 259–266, 2003.

[14] H. Wang, W. Wang, J. Yang, and P.S. Yu. Clustering by pattern similarity in large datasets. In *SIGMOD International Conference (SIGMOD 2002)*, pages 394–405, 2002.