Intersecting Data to Closed Sets with Constraints

Taneli Mielikäinen
HIIT Basic Research Unit
Department of Computer Science
University of Helsinki, Finland
Taneli.Mielikainen@cs.Helsinki.FI

Abstract

We describe a method for computing closed sets with data-dependent constraints. Especially, we show how the method can be adapted to find frequent closed sets in a given data set. The current preliminary implementation of the method is quite inefficient but more powerful pruning techniques could be used. Also, the method can be easily applied to wide variety of constraints. Regardless of the potential practical usefulness of the method, we hope that the sketched approach can shed some additional light to frequent closed set mining.

1 Introduction

Much of the research in data mining has concentrated on finding from some given (finite) set R all subsets that satisfy some condition. (For the rest of the paper we assume, w.l.o.g., that R is a finite subset of $\mathbb{N}$.)

The most prominent example of this task is probably the task of finding all subsets $X \subseteq R$ that are contained at least $minsupp$ times in the sets of a given sequence $d = d_1 \dots d_n$ of subsets $d_i \subseteq R$, i.e., to find the collection

$$\mathcal{F}(minsupp, d) = \{X \subseteq R : supp(X, d) \geq minsupp\}$$

where

$$supp(X, d) = |\{i : X \subseteq d_i, 1 \leq i \leq n\}|.$$

The collection $\mathcal{F}(minsupp, d)$ is known as the collection of frequent sets. (We could have defined the collection of frequent sets by by the frequency of sets which is a normalized version of supports: $fr(X, d) = supp(X, d)/n$.)

Recently one particular subclass of frequent sets, frequent closed sets, has received quite much attention. A set X is closed in d if $supp(X, d) > supp(Y, d)$ for all proper supersets Y of X. The collection of closed sets (in d) is denoted by

$$\begin{aligned}\mathcal{C}(d) &= \{X \subseteq R : Y \subseteq R, Y \supset X \\ &\qquad \Rightarrow supp(X, d) > supp(Y.d)\}\end{aligned}$$

The collection of frequent closed sets consists of the sets that are frequent and closed, i.e.,

$$\mathcal{FC}(minsupp, d) = \mathcal{F}(minsupp, d) \cap \mathcal{C}(d).$$

Most of the closed set mining algorithms [3, 12, 13, 14, 16, 19, 20] are based on backtracking [10]. In this paper we describe an alternative approach based on alternating between closed set generation by intersections and pruning heuristics. The method can be adapted to many kinds of constraints and needs only few passes over the data.

The paper is organized as follows. In Section 2 we sketch the method, in Section 3 we adapt the method for finding closed sets with frequency constraints, in Section 4 we describe some implementations details the method, and in Section 5 we experimentally study the properties of the method. Section 6 concludes the work and suggests some improvements to the work.

2 The Method

Let us assume that $R = \bigcup_{i \in \{1,\dots,n\}} d_i$ as sometimes R is not known explicitly. Furthermore, we shall use shorthand $d_{i,j}$ for the subsequence $d_i \dots d_j$, $1 \leq i \leq j \leq n$. The elements of R are sometimes called *items* and the sets d_i *transactions*.

As noted in the previous section, a set $X \subseteq R$ is closed in d if and only if $supp(X, d) > supp(Y, d)$ for all proper supersets Y of X. However, the closed sets can be defined also as intersection of the transactions (see e.g. [11]):

Definition 1 *A set $X \subseteq R$ is closed in d if and only if there is $I \subseteq \{1, \dots, n\}$ such that $X = \bigcap_{i \in I} d_i$. (By convention, $\bigcap_{i \in \emptyset} d_i = R$.)*

A straightforward implementation of Definition 1

$$\mathcal{C}(d) = \left\{ \bigcap_{i \in I} d_i : I \subseteq \{1, \ldots, n\} \right\}$$

leads to quite inefficient method for computing all closed sets:[1]

BRUTE-FORCE(d)
1 $R \leftarrow \bigcup_{i \in \{1,\ldots,n\}} d_i$
2 $supp(R) \leftarrow 0$
3 **for each** $I \subseteq \{1, \ldots, n\}, I \neq \emptyset$
4 **do** $X \leftarrow \bigcap_{i \in I} d_i$
5 **if** $supp(X) < |I|$
6 **then** $supp(X) \leftarrow |I|$
7 **return** $(supp : \mathcal{C} \rightarrow \mathbb{N})$

A more efficient solution can be found by the following recursive definition of closed sets:

$$\begin{aligned} \mathcal{C}(d_1) &= \{R, d_1\} \\ \mathcal{C}(d_{1,i+1}) &= \mathcal{C}(d_{1,i}) \cup \{X \cap d_{i+1} : X \in \mathcal{C}(d_{1,i})\} \end{aligned}$$

Thus the closed sets can be computed by initializing $\mathcal{C} = \left\{ R = \bigcup_{i \in \{1,\ldots,n\}} d_i \right\}$ (since R is always closed), initializing $supp$ to $R \mapsto 0$, and calling the following algorithm for each d_i ($1 \leq i \leq n$):

INTERSECT($supp : \mathcal{C} \rightarrow \mathbb{N}, d_i$)
1 **for each** $X \in \mathcal{C}$
2 **do** $\mathcal{C} \leftarrow \mathcal{C} \cup \{X \cap d_i\}$
3 **if** $supp(X \cap d_i) < supp(X) + 1$
4 **then** $supp(X \cap d_i) \leftarrow supp(X) + 1$
5 **return** $(supp : \mathcal{C} \rightarrow \mathbb{N})$

Using the above algorithm the sequence d does not have to be stored as each d_i is needed just for updating the current approximation of R and intersecting the current collection $\mathcal{C}$ of closed sets.

The closed sets can be very useful way to understand data sets that consist of only few different transactions and they have been studied in the field of Formal Concept Analysis [6]. However, many times all closed sets are not of interest but only frequent closed sets are needed. The simplest way to adapt the approach described above for finding the frequent closed sets is to first compute all closed sets $\mathcal{C}(d)$ and then remove the infrequent ones:

$$\mathcal{FC}(minsupp, d) = \{X \in \mathcal{C}(d) : supp(X, d) \geq minsupp\}$$

by removing all closed sets that are not frequent.

Unfortunately the collection of closed sets can be much larger than the collection of frequent closed sets. Thus the above approach can generate huge number of closed sets that do not have to be generated.

A better approach to find the frequent closed sets is to prune the closed sets that cannot satisfy the constraints – such as the minimum support constraint – as soon as possible. If the sequence is scanned only once and nothing is known about the sequence d in advance then no pruning of infrequent closed sets can be done: the rest of the sequence can always contain each closed set at least $minsupp$ times.

If more than one pass can be afforded or something is known about the data d in advance then the pruning of closed sets that do not satisfy the constraints can be done as follows:

INTERSECTOR(d)
1 $supp \leftarrow$ INIT-CONSTRAINTS (d)
2 **for each** d_i **in** d
3 **do** $supp \leftarrow$ INTERSECT $(supp, d_i)$
4 UPDATE-CONSTRAINTS $(supp, d_i)$
5 $supp \leftarrow$ PRUNE-BY-CONSTRAINTS $(supp, d_i)$
6 **return** $(supp : \mathcal{C} \rightarrow \mathbb{N})$

The function INTERSECTOR is based on three subroutines: function INIT-CONSTRAINTS initializes the data structures used in pruning and computes the initial collection of closed sets, e.g. the the collection $\mathcal{C} = \{R\}$, function UPDATE-CONSTRAINTS updates the data structures by one transaction at a time, and function PRUNE-BY-CONSTRAINTS prunes those current closed sets that cannot satisfy the constraints.

3 Adaptation to Frequency Constraints

The actual behaviors of the functions INIT-CONSTRAINTS, UPDATE-CONSTRAINTS and PRUNE-BY-CONSTRAINTS depend on the constraints used to determine the closed sets that are interesting. We shall concentrate on implementing the minimum and the maximum support constraints, i.e., finding the closed sets $X \in \mathcal{C}(d)$ such that $minsupp \leq supp(X, d) \leq maxsupp$.

The efficiency of pruning depends crucially on how much is known about the data. For example, if only the number of transactions in the sequence is known, then all possible pruning is essentially determined by Observation 1 and Observation 2.

Observation 1 *For all* $1 \leq i \leq n$ *holds:*

$$supp(X, d_{1,i}) + n - i < minsupp \Rightarrow supp(X, d) < minsupp$$

Observation 2 *For all* $1 \leq i \leq n$ *holds:*

$$supp(X, d_{1,i}) > maxsupp \Rightarrow supp(X, d) > maxsupp$$

[1] If $supp(X)$ is not defined then its value is interpreted to be 0.

Checking the constraints induced by Observation 1 and Observation 2 can be computed very efficiently. However, the pruning based on these observations might not be very effective: all closed sets in $d_{1,n-minsupp}$ can have frequency at least $minsupp$ and all closed sets in $d_{1,maxsupp}$ can have frequency at most $maxsupp$. Thus all closed sets in $d_{1,\min\{n-minsupp,maxsupp\}}$ are generated before the observations can be used to prune anything.

To be able to do more extreme pruning we need more information about the sequence d. If we are able to know the number of transactions in the sequence, it might be possible to count the supports of items. In that case Observation 3 can be exploited.

Observation 3 *If there exists $A \in X$ such that $supp(X, d_{1,i}) + supp(\{A\}, d_{i+1,n}) < minsupp$ then $supp(X, d) < minsupp$.*

Also, if we know the frequencies of some sets then we can make the following observation:

Observation 4 *If there exists $Y \subseteq X$ such that $supp(X, d_{1,i}) + supp(Y, d_{i+1,n}) < minsupp$ then $supp(X, d) < minsupp$.*

Note that these observations do not mean that we could remove the infrequent closed sets from the collection since an intersection of an infrequent closed set with some transaction might still be frequent in the sequence d. However, we can do some pruning based on the observations as shown in Proposition 1.

Proposition 1 *Let Z be the largest subset of $X \in \mathcal{C}$ such that for all $A \in Z$ hold $supp(X, d_{1,i}) + supp(\{A\}, d_{i+1,n}) \geq minsupp$. Then $X \in \mathcal{C}$ can be removed from $\mathcal{C}$ if there is a $W \subseteq Y \in \mathcal{C}, Z \subset W$, such that $supp(Y, d_{1,i}) \geq supp(X, d_{1,i})$ and for all $A \in W$ hold $supp(Y, d_{1,i}) + supp(\{A\}, d_{i+1,n}) \geq minsupp$, and replaced by Z otherwise.*

Proof. All frequent subsets of X are contained in Z. If there is a proper superset $W \subseteq Y \in \mathcal{C}$ of Z such that $supp(Y, d_{1,i}) + supp(\{A\}, d_{i+1,n}) \geq minsupp$ then all frequent subsets of X are contained in W and thus X can be removed. Otherwise Z is the largest subset of X that can be frequent and there is no superset of Z that could be frequent. If Z is not closed, then its support is equal to some of its proper supersets' supports. If Z is added to $\mathcal{C}$ then none of proper supersets is frequent and thus also Z is infrequent. □

This idea of replacing infrequent sets based on the supports items can be generalized to the case where we know supports for some collection $\mathcal{S}$ of sets.

Proposition 2 *Let $\mathcal{S}$ be the collection of sets such that $supp(Y, d_{1,i})$ and $supp(Y, d_{1,i})$ are known for all $Y \in \mathcal{S}$, and let $\mathcal{S}'$ consist of sets $Y \in \mathcal{S}, Y \subseteq X$, such that $supp(X, d_{1,i}) + supp(Y, d_{i+1,n}) < minsupp$. Then all frequent subsets of $X \in \mathcal{C}$ are the collection $\mathcal{S}''$ of subsets $Z \subseteq X$ such that $Z \not\subseteq Y$ for all $Y \in \mathcal{S}'$, no $W \subset Z \in \mathcal{S}''$ is contained in $\mathcal{S}''$.*

Proof. If $Z \subseteq X$ is frequent then there is a set in $\mathcal{S}''$ containing Z, or Z is contained in some set in $\mathcal{S}'$ but there is another set $Y \in \mathcal{C}$ such that $supp(Y, d_{1,i}) > supp(X, d_{1,i})$. □

Proposition 3 *Let $\mathcal{S}$, $\mathcal{S}'$ and $\mathcal{S}''$ be as in Proposition 2. Then $X \in \mathcal{C}$ can be replaced by the collection $\mathcal{S}'''$ consisting of sets in $\mathcal{S}''$ such that $supp(Y, d_{1,i}) + supp(W, d_{i+1,n}) < minsupp$ for some $W \subseteq Z \subseteq Y$ with $Y \in \mathcal{C}$ and $W \in \mathcal{S}$.*

Proof. If $Z \subseteq X$ is frequent then it is subset of some set in $\mathcal{S}''$ or there is $Y \in \mathcal{C}, Z \subseteq Y$, such that $supp(Y, d_{1,i}) + supp(W, d_{i+1,n}) < minsupp$ for all $W \in \mathcal{S}, W \subseteq Z$.

If $Z \in \mathcal{S}''$ is not closed then it is infrequent since none of its supersets is frequent. □

The efficiency of pruning depends crucially also on the ordering of the transactions. In Section 5 we experimentally evaluate some orderings with different data sets.

4 The Organization of the Implementation

A preliminary adaptation of the algorithm INTERSECTOR of Section 2 to minimum support and maximum support constraints is implemented as a program `intersector`. The main components of the implementation are classes `Itemarray`, `ItemarrayInput` and `ItemarrayMap`.

The class `Itemarray` is a straightforward implementation consisting of `int n` expressing the number of items in the set and `int* items` that is a length (at least) `n` array of items (that are assumed to be nonnegative integers) in ascending order. One of the reasons why this very simple representation of a set is used is that `Itemarray`s are used also in the data sources, and although some more sophisticated data structures would enable to do some operations more efficiently, we believe that `Itemarray` reflects better what an arbitrary source of transactions could give.

The class `ItemarrayInput` implements an interface to the data set d. The class handles the pruning of infrequent items from the input and maintaining the numbers of remaining occurrences of each item occurring in the data set. The data set d is accessed by a function `pair<Itemarray*,int>*`

`getItemarray()` which returns a pointer to next `pair<Itemarray*,int>`. The returned pointer is `NULL` if the previous pair were the last one in the data set d. The main difference to the reference implementation given at the home page of Workshop on Frequent Itemset Mining Implementations[2] is that `pair<Itemarray*,int>*` is returned instead of `Itemarray*`. This change were made partly to reflect the attempt to have the closure property of inductive databases [9] but also because in some cases the data set is readily available in that format (or can be easily transformed into that format). The interface `ItemarrayInput` is currently implemented in two classes `ItemarrayFileInput` and `ItemarrayMemoryInput`. Both of the classes read the data set d from a file consisting of rows of integers with possible count in brackets. Multiple occurrences of same item in one row are taken into account only once. For example, the input file

```
1 2 4 3 2 5 (54)
1 1 1 1
```

is transformed into pairs $\langle (1,2,3,4,5), 54\rangle$ and $\langle (1), 1\rangle$.

The class `ItemarrayFileInput` maintains in the main memory only the item statistics (such as the number of remaining occurrences of each item) thus possibly reading the data set several times. The class `ItemarrayMemoryInput` reads the whole data set d into main memory. The latter one can be much faster since it can also reorder the data set and replace all transactions $d_i, 1 \leq i \leq n$, with same frequent items by one pair with appropriate count. The implementations of these classes are currently quite slow which might be seen as imitating the performance of real databases quite faithfully.

The class `ItemarrayMap` represents a mapping from `Itemarray`s to supports. The class consists of a mapping `map<Itemarray*,int,CardLex> itemarrays` that maps the sets to supports, and a set `set<Itemarray*,CardLex> forbidden` consisting of the sets that are known to be infrequent or too frequent. The set `set<Itemarray*,CardLex> forbidden` is needed mainly because of the maximum frequency constraint. The class consists two methods:

- The method `intersect(const pair<Itemarray*,int>*)` intersects the current collection sets represented by the mapping `map<Itemarray*,int,CardLex> itemarrays` by the given set `pair<Itemarray*,int>*` and updates the supports appropriately.
- The method `prune(ItemarrayInput&)` prunes the sets that are already known to be infrequent or too frequent based on the statistics maintained by the implementation of the interface `ItemarrayInput`. (The class `CardLex` defines a total ordering of sets (of integers) by their cardinality and lexicographically within of each group with same sizes.) The pruning rules used in the current implementation of the method `prune(ItemarrayInput&)` are Observation 1, Observation 2, and Observation 3.

[2] `http://fimi.cs.helsinki.fi/`

5 The Experiments

name	# of rows	total # of items
`T10I4D100K`	100000	1010228
`T40I10D100K`	100000	3960507
`chess`	3196	118252
`connect`	67557	2904951
`internet`	10104	300985
`kosarak`	990002	8019015
`mushroom`	8124	186852
`pumsb`	49046	3629404
`pumsb*`	49046	2475947

Table 1. The data sets

We tested the efficiency and behavior of the implementation by the data sets listed in Table 1. All data sets except `internet` were provided by the Workshop on Frequent Itemset Mining Implementations. The data set `internet` is the Internet Usage data from UCI KDD Repository[3].

If the data sequence is read to main memory then it can be easily reordered. Also, even if this is not the case, there exist efficient external memory sorting algorithms that can be used to reorder the data [18]. The ordering of the data can affect the performance significantly.

We experimented especially with two orderings: ordering in ascending cardinality and ordering in descending cardinality. The results are shown in Figures 1–9. Each point $(|\mathcal{C}|, i)$ in the figures corresponds to the number $|\mathcal{C}|$ of closed sets in the sequence $d_{1,i}$ that could be frequent in the whole sequence d. Note that the reason why there is no point for each number i ($1 \leq i \leq n$) of seen transactions is that same set of items can occur several times in the sequence d.

There is no clear winner within the ascending and descending orderings: with data sets `T10I4D100K`, `T40I10D100K`, `internet`, `kosarak`, and `mushroom` the ascending order is better whereas the descending order seems to be better with data sets `chess`, `connect`, and `pumsb`. However, it is not clear whether this is due to the chosen minimum support thresholds.

[3] `http://kdd.ics.uci.edu/`

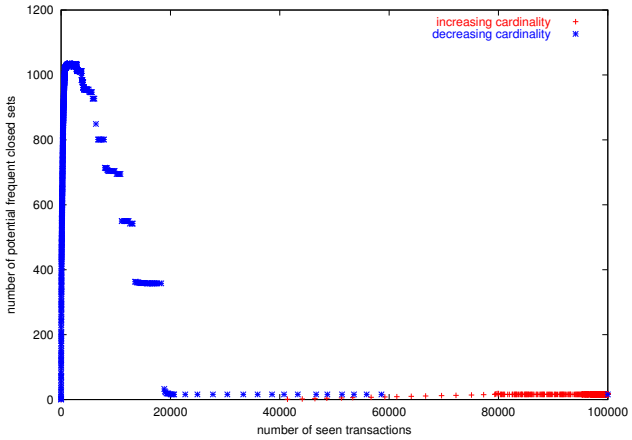

Figure 1. `T10I4D100K`, $minsupp = 4600$

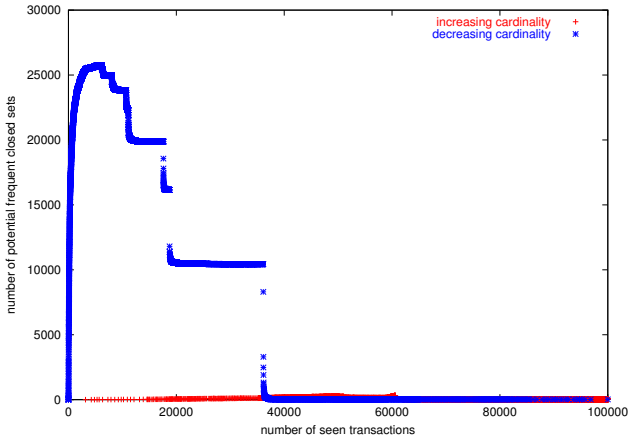

Figure 2. `T40I10D100K`, $minsupp = 16000$

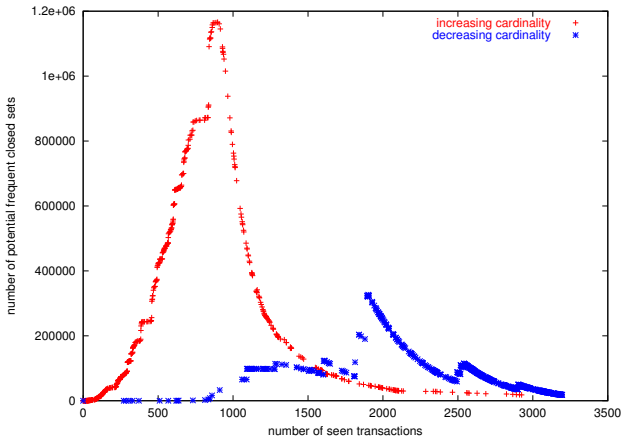

Figure 3. `chess`, $minsupp = 2300$

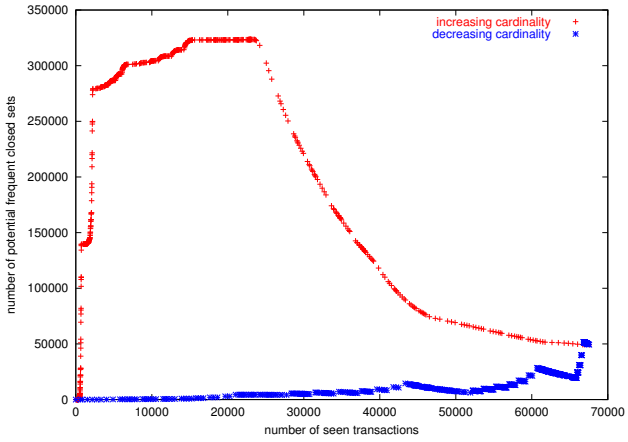

Figure 4. `connect`, $minsupp = 44000$

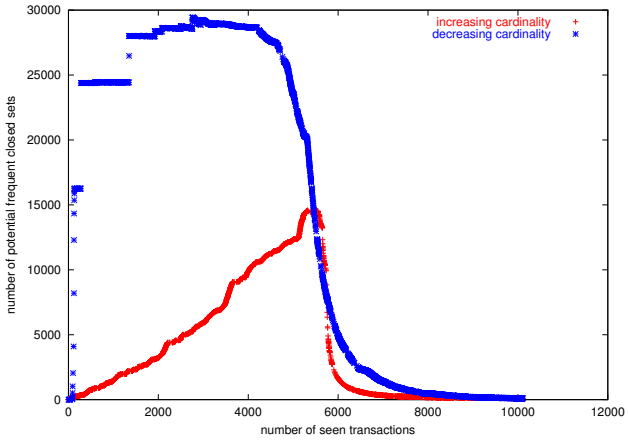

Figure 5. `internet`, $minsupp = 4200$

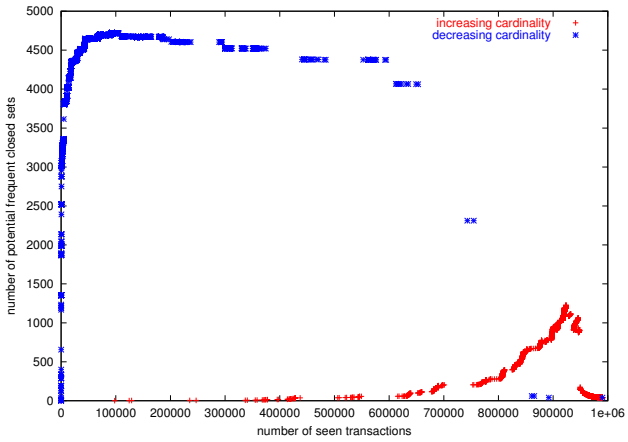

Figure 6. `kosarak`, $minsupp = 42000$

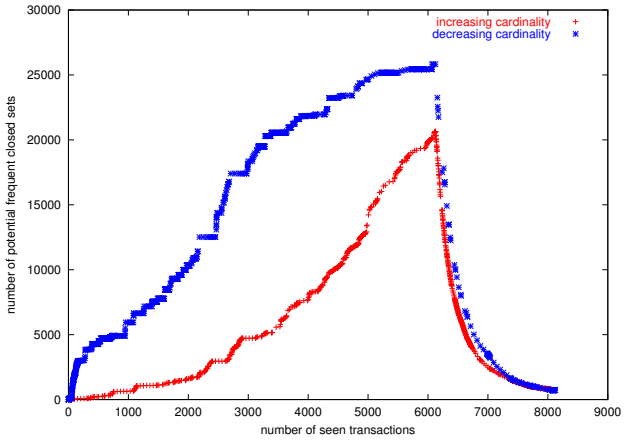

Figure 7. `mushroom`, $minsupp = 2000$

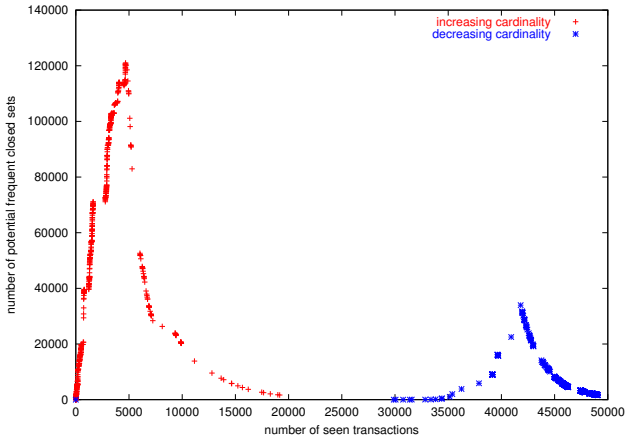

Figure 8. `pumsb`, $minsupp = 44000$

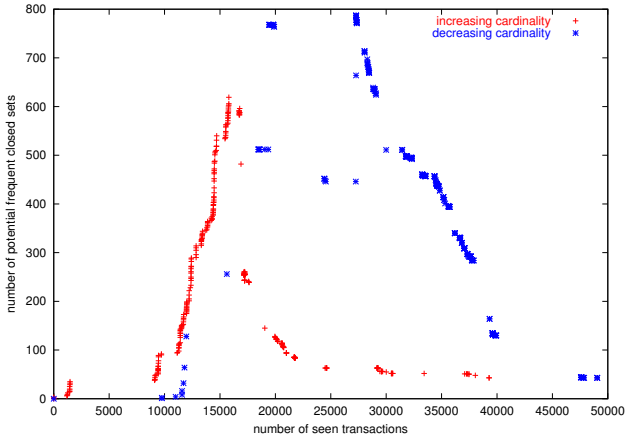

Figure 9. `pumsb*`, $minsupp = 32000$

One interpretation of the results is the following: small set d_i cannot increase the size of $\mathcal{C}$ dramatically since all new closed sets are subsets of d_i and d_i has at most $2^{|d_i|}$ closed subsets. However, the small sets do not decrease the remaining number of occurrences of items very much either. In the case of large sets d_j the situation is the opposite: each large set d_j decreases the supports $supp(\{A\}, d_{j+1,n})$ of each item $A \in d_j$ but on the other hand it can generate several new closed sets.

Also, we experimented with two data sets `internet` and `mushroom` to see how the behavior of the method changes when changing the minimum support threshold $minsupp$. The results are shown in Figure 10 and Figure 11.

The pruning seems to work satisfactory if the minimum support threshold $minsupp$ is high enough. However, it is not clear how much this is due to the pruning of infrequent items in the class `ItemarrayInput` and how much due to the pruning done by the class `ItemarrayMap`. Unfortunately, the performance rapidly collapses as the minimum support threshold decreases. It is possible that more aggressive pruning could help when the minimum support threshold $minsupp$ is low.

6 Conclusions and Future Work

In this paper we have sketched an approach for finding closed sets with some constraints from data with only few passes over the data. Also, we described a preliminary implementation of the method for finding frequent but not too frequent closed sets from data. The current version of the implementation is still quite inefficient but it can hopefully shed some light to the interplay of data and closed sets.

As the current implementation of the approach is still very preliminary, there is plenty of room for improvements, e.g., the following ones:

- The ordering of input seems to play crucial role in the efficiency of the method. Thus the favorable orderings should be detected and strategies for automatically finding them should be studied.

- The pruning heuristics described in this paper are still quite simplistic. Thus, more sophisticated pruning techniques such as inclusion-exclusion [5] should be tested. Also, pruning co-operation between closed sets generation and the data source management should be tighten.

- The pruning done by the data source management could be improved. For example, the data source management could recognize consecutive redundancy in the the data source.

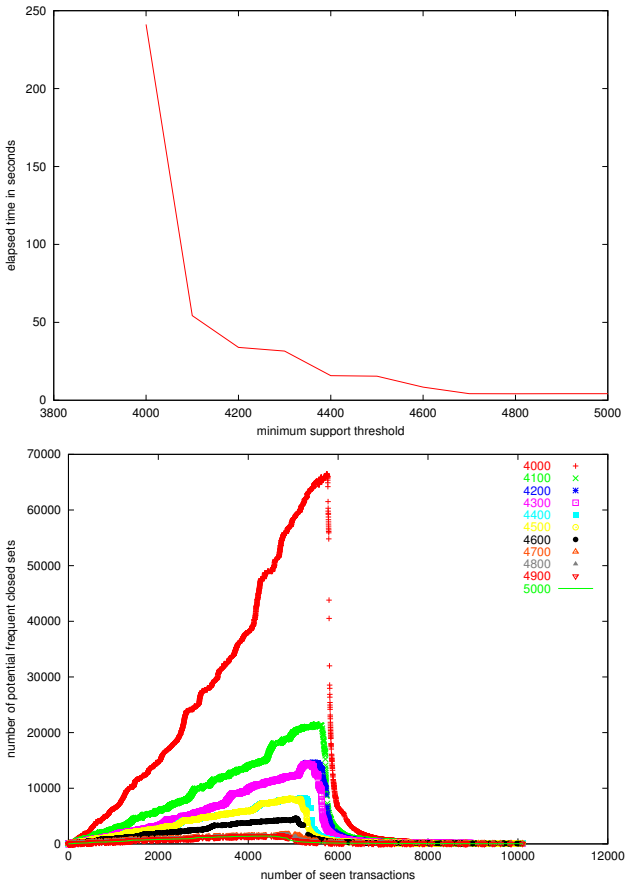

Figure 10. `internet`**, scalability**

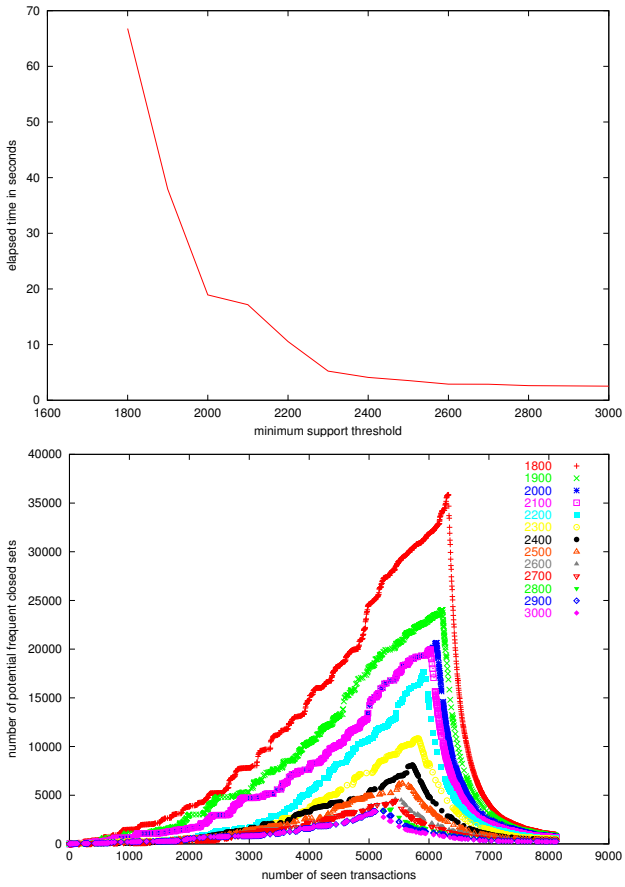

Figure 11. `mushroom`**, scalability**

- The intersection approach can be used to find all closed sets that are subsets of some given sets [11]. The method can be used to compute closed sets from the maximal sets in one pass over the data. As there exist very efficient methods for computing maximal sets [1, 2, 4, 7, 8, 15], it is possible that the performance of the combination could be quite competitive. Also, supersets of maximal frequent sets can be found with high probability from a small sample. Using these estimates one could compute supersets of frequent closed sets. This approach can be efficient if the supersets found from the sample are close enough to the actual maximal sets.

- After two passes over the data it is easy to do the third pass, or even more. Thus one could apply the intersections with several different minimum support thresholds to get refining collection of frequent closed sets in the data: the already found frequent closed sets with high frequencies could be used to prune less frequent closed sets more efficiently than e.g. the occurrence counters for frequent items.

- If it is not necessary to find the exact collection of closed sets with exact supports, then a sampling could be applied [17]. Also, if the data is generated by e.g. an i.i.d. source then one can sometimes obtain accurate bounds for the supports from relatively short prefixes $d_{1,i}$ of the sequence d.

- Other kinds of constraints than frequency thresholds should be implemented and experimented with.

References

[1] R. J. Bayardo Jr. Efficiently mining long patterns from databases. In A. T. Laura M. Haas, editor, *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 85–93. ACM, 1998.

[2] E. Boros, V. Gurvich, L. Khachiyan, and K. Makino. On the complexity of generating maximal frequent and minimal infrequent sets. In H. Alt and A. Ferreira, editors, *STACS 2002*, volume 2285 of *Lecture Notes in Computer Science*, pages 133–141. Springer-Verlag, 2002.

[3] J.-F. Boulicaut and A. Bykowski. Frequent closures as a concise representation for binary data mining. In T. Terano,

H. Liu, and A. L. P. Chen, editors, *Knowledge Discovery and Data Mining*, volume 1805 of *Lecture Notes in Artificial Intelligence*, pages 62–73. Springer-Verlag, 2000.

[4] D. Burdick, M. Calimlim, and J. Gehrke. MAFIA: A maximal frequent itemset algorithm for transactional databases. In *Proceedings of the 17th International Conference of Data Engineering (ICDE'01)*, pages 443–452, 2001.

[5] T. Calders and B. Goethals. Mining all non-derivable frequent itemsets. In T. Elomaa, H. Mannila, and H. Toivonen, editors, *Principles of Data Mining and Knowledge Discovery*, volume 2431 of *Lecture Notes in Artificial Intelligence*, pages 74–865. Springer-Verlag, 2002.

[6] B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag, 1999.

[7] K. Gouda and M. J. Zaki. Efficiently mining maximal frequent itemsets. In N. Cercone, T. Y. Lin, and X. Wu, editors, *Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 163–170. IEEE Computer Society, 2001.

[8] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, and R. S. Sharma. Discovering all most specific sentences. *ACM Transactions on Database Systems*, 28(2):140–174, 2003.

[9] T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Communications of The ACM*, 39(11):58–64, 1996.

[10] D. L. Kreher and D. R. Stinson. *Combinatorial Algorithms: Generation, Enumeration and Search*. CRC Press, 1999.

[11] T. Mielikäinen. Finding all occurring sets of interest. In J.-F. Boulicaut and S. Džeroski, editors, *2nd International Workshop on Knowledge Discovery in Inductive Databases*, pages 97–106, 2003.

[12] F. Pan, G. Cong, A. K. H. Tung, J. Yang, and M. J. Zaki. CARPENTER: Finding closed patterns in long biological datasets. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2003.

[13] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In C. Beeri and P. Buneman, editors, *Database Theory - ICDT'99*, volume 1540 of *Lecture Notes in Computer Science*, pages 398–416. Springer-Verlag, 1999.

[14] J. Pei, J. Han, and T. Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. In D. Gunopulos and R. Rastogi, editors, *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 21–30, 2000.

[15] K. Satoh and T. Uno. Enumerating maximal frequent sets using irredundant dualization. In G. Grieser, Y. Tanaka, and A. Yamamoto, editors, *Discovery Science*, volume 2843 of *Lecture Notes in Artificial Intelligence*, pages 256–268. Springer-Verlag, 2003.

[16] G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, and L. Lakhal. Computing iceberg concept lattices with TITANIC. *Data & Knowledge Engineering*, 42:189–222, 2002.

[17] H. Toivonen. Sampling large databases for association rules. In T. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, editors, *VLDB'96, Proceedings of 22nd International Conference on Very Large Data Bases*, pages 134–145. Morgan Kaufmann, 1996.

[18] J. S. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, 33(2):209–271, 2001.

[19] J. Wang, J. Han, and J. Pei. CLOSET+: Searching for the best strategies for mining frequent closed itemsets. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003.

[20] M. J. Zaki and C.-J. Hsiao. CHARM: An efficient algorithms for closed itemset mining. In R. Grossman, J. Han, V. Kumar, H. Mannila, and R. Motwani, editors, *Proceedings of the Second SIAM International Conference on Data Mining*. SIAM, 2002.