# Implementing and analysing dataset dimensionality reduction through frequent closed itemsets

Alan Souza
Universidade Federal do Rio Grande do Sul, Brazil
apsouza@inf.ufrgs.br

## ABSTRACT

Dimensionality reduction using frequent closed itemsets has been originally proposed by [6]. This work aims to reduce the dataset size by grouping common items together, building what is called factor-items. Hence, instead of a dataset with transactions, and each of them with its items, we have a dataset with factorized transaction, and each of them with its factor-items.
This paper aims to fully reproduce the algorithm, and compare the generated results with the experiments in the original paper. Also, this paper provides a complexity analysis over the replicated algorithm.

## Keywords

Data Mining, Frequent Closed Itemsets, Dimensionality Reduction

## 1. INTRODUCTION

Dataset is a transactional database consisting of list of transactions and each of them containing a finite set of items. Dimensionality reduction is the process of finding a set of new items (factor-items) which is considerably smaller than the original set. These factor-items aims to comprise full or nearly full information about the original elements [6].
For a dataset, a frequent itemset $P$ is an itemset included in at least a specified number of transactions, which is usually called minimum support. A frequent itemset $P$ is called *closed* if it's included in no other itemset which has the same transactions of P. A frequent itemset $P$ is called *maximal* if it's included in no other itemset [11].
In [6], Krajca et al. show that frequent closed itemset can be used to efficiently find factor-items and thus accomplishing data dimensionality reduction. Also, if only frequent closed itemsets is not enough, they propose an on-demand solution to find additional factor-items until a configurable approximation degree is achieved.
For frequent closed itemsets, there are a number of implemented algorithms [9, 10, 11, 12, 13]. The linear time Closed itemset Miner (LCM) [11] is known to be the state-of-art algorithm for mining frequent closed itemset. They won the best implementation award at FIMI'04 [4].
This paper aims to implement and analyse the algorithm proposed in [6]. Also, as Krajca et al. solution use frequent closed itemsets, the LCM algorithm [11] has also been implemented.
The remainder of this paper is organized as follows. Section 2 gives require background to understand why frequent closed itemset fits to data dimensionality reduction. Section 3 explains the LCM algorithm proposed by Uno el al. in 2004. Section 4 deeply explains the replicated algorithm originally proposed by Krajca in 2011. Section 5 reports difficulties and limitations faced during implementation time. Also, a comparison has been done over the original experiments in [6] and the one replicated in this article.

## 2. PROBLEM STATEMENT

A dataset consists of $m$ transactions $T_1, T_2, ..., T_m$ where each transaction $T_i$ $(i = 1, 2, ..., m)$ is a subset of a fixed finite set $I$ of all possible items. So, the *factorization problem* consists on finding a set $F \subset 2^I$ of non-empty subsets of $I$ and transactions $R_1, R_2, ..., R_m$ over $F$ such that for any $i = 1, 2, ..., m$ then $T_i = \bigcup R_i$. Where $\bigcup R_i$ denotes the union of all $B \in R_i$. Each $B \in F$ is called a factor-item and $R_1, R_2, ..., R_m$ is called factorized-transaction.
*Example 1.* Consider the set of items $I = \{0, 1, ..., 9\}$ and the following transactions over $I$:

$$T_1 = \{1, 2, 4, 6, 7, 8\}, T_2 = \{3, 5, 9\}, T_3 = \{1, 2, 7\},$$

$$T_4 = \{0, 1, 2, 4, 7, 9\}, T_5 = \{1, 2, 4, 6, 7, 8\}, T_6 = \{0, 3, 4, 5, 9\}$$

Building the following set $F = \{B_1, B_2, B_3, B_4\}$ of factor-items:

$$B_1 = \{0, 4, 9\}, B_2 = \{1, 2, 7\}, B_3 = \{3, 5, 9\}, B_4 = \{1, 4, 6, 8\}$$

Now, $T_1, ..., T_6$ can be converted to the following factorized transactions:

$$R_1 = \{B_2, B_4\}, R_2 = \{B_3\}, R_3 = \{B_2\}$$

$$R_4 = \{B_1, B_2\}, R_5 = \{B_2, B_4\}, R_6 = \{B_1, B_3\}$$

It's possible to check a dimensionality reduction. Instead of $|I| = 10$, we have $|F| = 4$ while still comprising full information over the original dataset (40% of factor-items is enough to represent 100% of the original items). With this exact factorization we will always be able to go back to the orignal set of items because there is no data loss.
If we relax the factorization condition to $T_i \approx \bigcup R_i$, we would still be able to represent the original dataset, but with some approximation factor.
Consider the above example, replacing the factor items $F$ to $F = \{B_1, B_2, B_4\}$. So, we can approximately express $T_1, ..., T_6$ by the following factorized transactions:

$$R_1 = \{B_2, B_4\}, R_2 = \{\}, R_3 = \{B_2\}$$

$$R_4 = \{B_1, B_2\}, R_5 = \{B_2, B_4\}, R_6 = \{B_1\}$$

The reduction here is bigger (30%) but we some data loss. This is called *approximate factorization*. Depending on the user's needs it may be acceptable loose some information depending on the approximation factor. So, it's important to have a formula to calculate the *approximation degree*, which is defined as bellow:

$$\frac{\sum_{i=1}^{m} |\bigcup R_i|}{\sum_{i=1}^{m} |T_i|}$$

In the above example, the approximation degree would be $\frac{24}{29} \approx 0.83$. We would read this as "with 3 factor-items we are able to approximate 83% of the original 10 items."
The remaining work here is to actually formally define the process of finding these factor-items, which is the main contribution of [6].
An $m \times n$ Boolean matrix $A$ is any real-valued matrix such that $A_{i,j} \in \{0, 1\}$ for all $i = 1, 2, ..., m$ and $j = 1, 2, ..., n$. A Boolean matrix multiplication is defined as $(A \circ B)_{ik} = 1$, where $B$ is an $n \times p$ Boolean matrix and $k = 1, ..., p$, iff there is such $j$ that $A_{ij} = 1$ and $B_{jk} = 1$.
An $m \times p$ Boolean matrix $C$ can be seen as representing relationship between object corresponding to rows and attributes corresponding to columns. If $C_{ij} = 1$ represents that the object given by row $i$ has attribute $j$. A decomposion [3] of C can be represented as Boolean matrix multiplication as $C = A \circ B$. With $n$ as small as possible ($n$ in our case are the factor-items). The purpose of the decomposition is to express the object-attribute relationship represented by $C$ by a relationship between object and factors ($A$) and factors and attributes ($B$).
**Theorem 1.** There is $F = \{B_1, ..., B_n\} \in 2^I$ and transactions $R_1, R_2, ..., R_m$ iff there is an $m \times n$ matrix $R$ and an $n \times p$ matrix $F$ such that $T = R \circ F$. The corresponding Boolean Matrix representation from Example 1 is given bellow:

$$\begin{pmatrix} 0&1&1&0&1&0&1&1&1&0 \\ 0&0&0&1&0&1&0&0&0&1 \\ 0&1&1&0&0&0&0&1&0&0 \\ 1&1&1&0&1&0&0&1&0&1 \\ 0&1&1&0&1&0&1&1&1&0 \\ 1&0&0&1&1&1&0&0&0&1 \end{pmatrix} = \begin{pmatrix} 0&1&0&1 \\ 0&0&1&0 \\ 0&1&0&0 \\ 1&1&0&0 \\ 0&1&0&1 \\ 1&0&1&0 \end{pmatrix} \circ \begin{pmatrix} 1&0&0&0&1&0&0&0&0&1 \\ 0&1&1&0&0&0&0&1&0&0 \\ 0&0&0&1&0&1&0&0&0&1 \\ 0&1&0&0&1&0&1&0&1&0 \end{pmatrix}$$

Maximum rectangles in $T$ represent particular submatrices of T which are full of 1s and cannot be enlarged.
**Theorem 2.** There is a number $q \leq n$ and maximum rectangles $D_1, ..., D_q$ in $T$ such that $T = R \circ F$ holds for $m \times q$ and $q \times p$ Boolean matrices R and F where $R_{*j} \circ F_{j*} = D_j$ for all $j = 1, ..., q$.
According to Theorem 2, the task of finding a factorization with $n$ as small as possible reduces to the problem of finding a minimal set of maximal rectangles in $T$. This turns out to be the solution for the exact factorization problem.
The corresponding maximal rectangles in $T$ for the Example 1 are the following:

$$D_1 = \begin{pmatrix} 0&\mathbf{1}&\mathbf{1}&0&1&0&1&\mathbf{1}&1&0 \\ 0&0&0&1&0&1&0&0&0&1 \\ 0&\mathbf{1}&\mathbf{1}&0&0&0&0&\mathbf{1}&0&0 \\ 1&\mathbf{1}&\mathbf{1}&0&1&0&0&\mathbf{1}&0&1 \\ 0&\mathbf{1}&\mathbf{1}&0&1&0&1&\mathbf{1}&1&0 \\ 1&0&0&1&1&1&0&0&0&1 \end{pmatrix}, D_2 = \begin{pmatrix} 0&1&1&0&1&0&1&1&1&0 \\ 0&0&0&1&0&1&0&0&0&1 \\ 0&1&1&0&0&0&0&1&0&0 \\ \mathbf{1}&1&1&0&\mathbf{1}&0&0&1&0&\mathbf{1} \\ 0&1&1&0&1&0&1&1&1&0 \\ \mathbf{1}&0&0&1&\mathbf{1}&1&0&0&0&\mathbf{1} \end{pmatrix}$$

$$D_3 = \begin{pmatrix} 0&1&1&0&1&0&1&1&1&0 \\ 0&0&0&\mathbf{1}&0&\mathbf{1}&0&0&0&\mathbf{1} \\ 0&1&1&0&0&0&0&1&0&0 \\ 1&1&1&0&1&0&0&1&0&1 \\ 0&1&1&0&1&0&1&1&1&0 \\ 1&0&0&\mathbf{1}&1&\mathbf{1}&0&0&0&\mathbf{1} \end{pmatrix}, D_4 = \begin{pmatrix} 0&\mathbf{1}&\mathbf{1}&0&\mathbf{1}&0&\mathbf{1}&\mathbf{1}&\mathbf{1}&0 \\ 0&0&0&1&0&1&0&0&0&1 \\ 0&1&1&0&0&0&0&1&0&0 \\ 1&1&1&0&1&0&0&1&0&1 \\ 0&\mathbf{1}&\mathbf{1}&0&\mathbf{1}&0&\mathbf{1}&\mathbf{1}&\mathbf{1}&0 \\ 1&0&0&1&1&1&0&0&0&1 \end{pmatrix}$$

Maximal rectangles are in a one-to-one correspondence with closed itemsets. The corresponding closed itemsets in this example are:

$$B_1 = \{1, 2, 7\}, B_2 = \{0, 4, 9\}$$

$$B_3 = \{3, 5, 9\}, B_4 = \{1, 2, 4, 6, 7, 8\}$$

# 3. LCM ALGORITHM

LCM is an abbreviation of Linear time Closed itemset Miner. In [11], Uno proposes three algorithms LCM, LCMfreq an LCMmax, for enumerating closed, all and maximal frequent itemsets, respectively. This paper replicates the LCM implementation for mining frequent closed itemsets which is going to be used in Section 4.
Existing algorithms for closed itemset mining usually enumerate all frequent itemsets and, as a post-processing pruning method, eliminate the non-closed ones. This is known by *apriori* or *level-by-level* technique [2, 1]. It's expected the number of closed itemsets to be much smaller than the number of all frequent ones. Hence, the main problem of this technique is high memory consuption for data which is not going to be part of the final solution. Apriori algorithms have advantages for frequency counting [11].
Other approaches for mining itemsets from large databases use a technique known as backtracking, which is based on recursive calls. An iteration of a backtracking algorithm inputs a frequent itemset $P$, and generates another possible frequent itemsets by adding every items to P. Then, for each itemset being frequent among these new generated ones, the iteration generates recursive calls with respect to it [5, 9, 10]. An execution of the backtracking algorithm gives a three structure such that the verticies are iterations (frequent itemsets), and edges connects two iterations if one of the iterations call the other, we say that iteration $I$ is the parent of $I'$, and $I'$ is a child of $I$. Backtracking algorithms use less memory as it only stores the current solution in the memory.
LCM algorithms are based on backtracking, and use efficient techniques for frequency counting which are occurrence deliver and anytime database reduction. Frequency counting is the most heavy part of the frequent itemset mining, which is to count the number of transactions including a newly generated itemset [11]. For an itemset $P$, we call the item with the maximum index in $P$ the tail of $P$, and denote by $tail(P)$. Also, the transactions containing itemset $P$ is called $T(P)$.
Occurrence deliver [9, 10] computes $T(P \cup \{e\})$ for $e = tail(P) + 1, ..., |\chi|$, where $\chi$ is the set of all items in a transaction database, only by tracing $T(P)$. In other words, for every new itemset $P \cup \{e\}$, its frequency counting is not over the whole database. Instead, it's only iterate through $T(P)$. This uses the concept that $T(P \cup \{e\}) \subseteq T(P)$ and is clear that it holds. Hence, occurrence deliver is used to remove non-necessary transactions.
Anytime database reduction speeds up the frequency counting by iteratively removing items smaller than $tail(P)$. Suppose that an iteration $I$ of a backtracking algorithm receive a frequent itemset $P$ from its parent. Then, in any descendant iteration of $I$, no item of indices smaller than $tail(P)$ is added. Hence, anytime database reduction can be used to remove those items for the subsequent recursive calls.
Algorithm 1 shows the detailed LCM implementation replicated in this article. The algorithm first outputs the current itemset (in the very first stage it is going to be empty).

**Algorithm 1** $lcm(itemset, transactions, frequency)$

```
output(itemset);
int tail = tail(itemset, frequency);
List freqItems = freqItems(itemset, tail);
for (e : freqItems)
    intersect = intersect(transactions, e);
    if(ppcExt(itemset, e, intersect))
      p = fci(item, e, intersect);
      databaseReduction(intersect);
      freq = freqCount(p, e, transactions, frequency);
      lcm(p, intersect, freq);
    end if
end for
```

The $tail(itemset, frequency)$ procedure, starts from the last frequency count until it finds an item with a different frequency. The new tail is the item with different frequency count. If all frequency counts are the same, then the first item will be the tail.
The $freqItems(itemset, tail)$ retrieves all the frequent items $i > tail$ which is not already present in $itemset$.
The $intersect(transactions, e)$ procedure gets all the common transactions between the current itemset and the new item $e$. The $ppcExt(itemset, e, intersect)$ procedure actually makes the verification if the $P \cup e$ is closed. This uses the concept of prefix preserving closure extension (details can be found at [9, 10]). Basically, this process works as follows: for each item $i < e$, in the original set of items, check if this item $i$ is present in the current itemset and if the item $i$ is present in all intersect transacations. If true, then the union $P \cup e$ is not closed and the procedure returns false.
The $fci$ procedure actually retrieves the frequent closed itemset which is composed by every item $i < e$ from the original items, the $e$ item itself, and for each item $j > e$, it belongs to the closed itemset only if it's present in every intersect transacations. The $databaseReduction(intersect)$ builds the occurrence delivery only for the intersecting transactions and, consequently, the new items will be only the ones which the intersect transactions have.
The $freqCount(p, e, transactions, frequency)$ efficiently counts, for the remaining items, its frequency. It returns a list of integers containing the frequency for each item $e \in itemset$.
The recursive call $lcm(p, intersect, freq)$ is only executed if $P \cup e$ is a ppc extension.
*Complexity Analysis*: To get all the frequent items it would take, in the worst-case, $O(|\chi|)$, where $\chi$ is all database items. To retrieve the intersect transactions it would take, in the worst-case, $O(|T(P)|)$, where $T(P)$ is the transactions of the current itemset $P$, which is considerably smaller than the whole database transactions. To check whether $P \cup e$ is a ppc-extension it would take $O(|T(P)| \times |\chi|)$. To build the frequent closed itemset it requires $O(|T(P)| \times |\chi(P)|)$, where $\chi(P)$ are the items from itemset $P$. To rebuild the occurrence delivery with reduced database, it would take $O(|T(P)|)$. To build the new frequency count list it would take $O(|T(P)| \times |\chi(P)|)$. As a result, the overall cost to enumerate frequent closed itemsets with LCM is $O(|T(P)| \times |\chi|)$. In terms of space, it would require $O(|T|)$ space for storing all factorized transactions, where $T$ is total number of transacations.
The time and space complexity of the existing algorithms [7, 8, 14] are $O(|T| \times |F|)$ and $O(|T| + |C| + |\chi|)$, respectively. $F$ represents all the frequent patterns, $C$ is the number of frequent closed itemsets. LCM approach is cleary more efficient both in time and space. Also, as results in [10] show, LCM performance is possibly exponentially better than the existing algoritms for some instances.

# 4. FCI-DR ALGORITHM

This algorithm is the main contribution of [6] and aims to use the frequent closed itemsets to factorize original items and achieve dataset reduction. As shown in Section 2, the frequent closed itemsets are in a one-to-one relationship with maximal rectangles. In Section 3, it's described LCM, which is an efficient algorithm to find all the frequent closed itemsets. The remaining work here is to actually select from all closed itemsets the best ones to comprise the desired approximation degree. Experiments in [6], have shown that high minimum support values when building frequent closed itemsets may turn the exact factorization impossible. In other words, with the increasing value of minimum support, frequent closed itemsets tend to lose the ability to constitute good factors. So, the algorithm takes the advantage of frequent closed itemsets but is still able to continue to achieve a preset approximation degree even if the computed frequent closed itemsets are not sufficient to do so. The algorithm combines two approaches. The basic one with frequent closed itemsets as candidates for factorization and an on-demand computation of factor-items by an incremental joining of "promissing items". Algorithm 2 shows the algorithm to the factorization problem which has been replicated in this article. The $lcm.run(minSup, dataset)$ procedure

**Algorithm 2** $factorize(approxDegree, dataset, minSup)$

```
S = lcm.run(minSup, dataset);
U = dataset.transactions;
n = itemCount(U);
while (S ≠ 0 and 1 − itemCount(U)/n < approxDegree)
    B = findBest(S);
    U = removeRectangle(U, B);
    S = updateList(S, U);
    store(B);
end while
while (1 − itemCount(U)/n < approxDegree)
    B = bestRemaining(U);
    U = removeRectangle(U, B);
    store(B);
end while
```

calls the LCM algorithm to retrieve the frequent closed itemsets and store in $S$. The $itemCount(U)$ procedure counts the number of items in the whole universe (it does not matter if it repeats). It's the total item occurrence in all transactions (universe).
The first while loop, uses the frequent closed itemset to build the factor-items. The $findBest(S)$ procedure retrieves the closed itemset which has the biggest universe cover (happens in more transactions and uses more items in one step leading to a bigger coverage). The $updateList(S, U)$ procedure removes from $S$ superfluous itemsets that no longer cover any transaction in the remaining universe.
The $removeRectangle(U, B)$ procedure removes from each transaction in the universe the items already covered by the

| data set | support | my sup | freq. itemsets | my freq. item | approx. | factors | my factors | dim. reduction | my dim. red |
|---|---|---|---|---|---|---|---|---|---|
| T10I4D100K | 0.4% | 0.4% | 1.992 | 1.068 | 95% | 631 | 620 | 63% | 62.06% |
| T40I10D100K | 0.2% | 2.5% | 1.056 | 1219 | 92% | 611 | 604 | 61% | 60.46% |
| mushroom | 6.9% | 6.9% | 8.275 | 7.816 | 94% | 78 | 44 | 65% | 36,97% |

**Table 1: Examples of dimensionality reduction with a comparison between the original implementation and the replicated solution of this paper.**

newly selected factor-item (B).
The second while loop, is the on-demand solution which keeps adding factor-items till the approximation factor is achieved. This is used when the frequent closed itemsets are not enough to achieve the approximation degree (usually when the minSup is very high). The $bestRemaining(U)$ procedure finds all the remaining items in the universe and, from these remaining ones, retrieves the first with biggest occurrence (happens in many transactions). After, it iterates through the remaining items and add them to the factor-item till the first decrease in the occurrence. Before adding, it checks for the item occurrence, if it's smaller than the itemset occurrence, it stops and retrieve the current factor-item as the "best remaining".
*Complexity Analysis*: As described in Section 3, the time complexity of LCM algorithm is $O(|T(P)|\times|\chi|)$, where $T(P)$ are the transactions of the first closed itemset $P$, and $\chi$ is the total database items. To find the best closed itemset it requires $O(|F|)$, where $F$ is total number of frequent closed itemsets. To remove the superfluous items it requires $O(|T|)$, where $T$ is the number of transactions. To update the closed frequent itemset list it requires $O(|F|)$. The first while loop runs for, in the worst case, $O(|F|)$. So, the total complexity of the first loop is $O(|F|^2 \times |T|)$.
The *bestRemaining* procedure takes $O(|T| + |\chi|)$ to complete. The worst case for the second loop is if the frequent itemsets are zero (extremely high supports that no item can be found). So, the second loop complexity is $O(|\chi|^2 \times |T|)$. The total execution time for factorization algorithm is $O(|T| \times (|\chi|^2 + |F|^2))$.

# 5. CONCLUSION

In [6], the procedure $UpdateList(S)$ does not take into consideration the universe. But, if you notice when they explain the UpdateList purpose, they state that it needs the universe U. I've changed the code to actually use that, and it really makes sense to. Another difference is the $findBest$ procedure. When, I've first implemented, I decided to take into consideration the universe U to find the best closed itemset occurrence. This was consuming much time. As explained in Section 3, the frequency count is the heaviest task. So, as LCM already computes the itemset occurrence I decided to discard the universe U to find the best closed itemset. As there is just the declaration in the original paper, I assumed that the universe U was being used in findBest procedure to count the frequency for the current itemset. Also, I could not compare the timing, because they did not report the time for each factorize execution. Another difficulty I've faced, was to find all the datasets they used for testing. Despite the repository be described in the original paper, I was not able to find some of them (they might be removed from the database). They've used six datasets for testing and I was able to reproduce just three of them. Table 1 shows the comparison between the orignal results and mine. I've ommitted the dataset I was not able to find over the internet.

# 6. REFERENCES


[1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI/MIT Press, 1996.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[3] R. Belohlavek and V. Vychodil. Discovery of optimal factors in binary data via a novel method of matrix decomposition. *J. Comput. Syst. Sci.*, 76(1):3–20, Feb. 2010.

[4] FIMI. Workshop on frequent itemset mining implementations (fimi'04). http://fimi.ua.ac.be/fimi04/, Nov. 2004.

[5] R. J. B. Jr. Efficiently mining long patterns from databases. In L. M. Haas and A. Tiwary, editors, *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, pages 85–93. ACM Press, 1998.

[6] P. Krajca, J. Outrata, and V. Vychodil. Using frequent closed itemsets for data dimensionality reduction. *Data Mining, IEEE International Conference on*, 0:1128–1133, 2011.

[7] N. Pasquier, Yves, Y. Bastide, R. Taouil, and L. Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 24:25–46, 1999.

[8] J. Pei, J. Han, and R. Mao. Closet: An efficient algorithm for mining frequent closed itemsets. pages 21–30, 2000.

[9] T. Uno, T. Asai, Y. Uchida, and H. Arimura. Lcm: An efficient algorithm for enumerating frequent closed item sets. In *In Proceedings of Workshop on Frequent itemset Mining Implementations (FIMI'03)*, 2003.

[10] T. Uno, T. Asai, Y. Uchida, and H. Arimura. An efficient algorithm for enumerating closed patterns in transaction databases. In *Discovery Science*, pages 16–31, 2004.

[11] T. Uno, M. Kiyomi, and H. Arimura. Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *FIMI*, 2004.

[12] T. Uno, M. Kiyomi, and H. Arimura. Lcm ver.3: collaboration of array, bitmap and prefix tree for frequent itemset mining. In *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*, OSDM '05, pages 77–86, New York, NY, USA, 2005. ACM.

[13] S. B. Yahia, T. Hamrouni, and E. M. Nguifo. Frequent closed itemset based algorithms: a thorough structural and analytical survey. *SIGKDD Explor. Newsl.*,

8(1):93–104, June 2006.

[14] M. J. Zaki and C. jui Hsiao. Charm: An efficient algorithm for closed itemset mining. pages 457–473, 2002.