Algorithm MONSA for All Closed Sets Finding: basic concepts and new pruning techniques

REIN KUUSIK, GRETE LIND
Department of Informatics
Tallinn University of Technology
Raja 15, Tallinn 12618
ESTONIA
kuusik@cc.ttu.ee, grete@staff.ttu.ee

Abstract: - In this paper an algorithm named MONSA for closed sets mining is presented. It does not use such kind of techniques as in ChARM by Zaki and Hsiao. MONSA is an exact depth-first search algorithm extracting only frequent closed sets using several new very effective pruning techniques to be free from repetitive and empty patterns. MONSA does not depend on the initial order of objects. In MONSA there is active only one branch which is under construction. The purpose of this paper is to describe the approach used in MONSA and the correspondence of its basics and concepts to the approach by Zaki and Hsiao. A full example of the algorithm's work is presented. By the algorithm the intersections (closed sets) and IF…THEN rules on the subsets of source data set simultaneously are found. MONSA treats not only binary data, but a larger set of discrete values.

Key-Words: - Data mining, Frequent closed sets, Pruning techniques, Depth-first search, Monotone systems

1 Introduction

„The task of mining association rules consists of two main steps. The first involves finding the set of all frequent itemsets. The second involves testing and generating all high confidence rules among itemsets" [1]. Zaki and Hsiao prove in [1] that „It is not necessary to mine all frequent itemsets in the first step, instead it is sufficient to mine the set of closed frequent itemsets, which is much smaller than the set of all frequent itemsets" and „It is also not necessary to mine the set of all possible rules", because „any rule between itemsets is equivalent to some rule between closed itemsets. Thus many redundant rules can be eliminated" [1]. Also it is important to enumerate closed sets directly, without generating them „using Apriori-like bottom-up search methods that examine all subsets of frequent itemsets" or finding them from maximal patterns „since all subsets of the maximal itemsets would again have to be examined" [2].

As we see, for mining association rules we need to find only closed sets. We did it. But in our algorithm MONSA we use other denotations and techniques than Zaki and Hsiao [1] [2].

Algorithm MONSA [3] was created for discovering intersections with special quality (see section 2.2 (6)). Appears that the set of intersections it founds is the same as the set of all (frequent) closed sets. Additionally our algorithm enables to find rules (between closed itemsets) at the same time as closed itemsets itself. Nevertheless this is a subset of all possible rules between closed sets. In order to ensure that these rules hold with at least required confidence it is possible to prune the branches of a search tree according to the threshold given by the user.

The main goal of the paper is to describe our algorithm and the main theoretical conceptions on which it is based.

1.1 The task

MONSA is a depth-first search algorithm used for discovering intersections with special quality. The *intersection* of two (or more) sets is the set of elements, which belong to both (all) sets, simultaneously. Without used pruning techniques MONSA would perform exhaustive search and produce all permutations of all existing value combinations. The *value combination* is a set of elements, an *element* is an attribute with certain value. Taking into account the minimal frequency allowed by the user only the frequent part of the result is found.

There are several problems in intersections' finding process: we have to prevent finding 1) repetitious intersections and 2) empty intersections. Finding empty intersections is avoided by the nature of the algorithm MONSA, it does not generate (theoretical) combinations, but traverses only

through the really existing ones. In order to prevent "repetitions" we use pruning techniques called "bringing zeros down" and "backward comparison" (see 3.3). Appears that it is not enough, sometimes we have to perform one more check: 3) is the extracted intersection (i.e. closed set) really non-redundant (i.e. not extracted already) or not. This check is applied only when the current set is not closed (and addition of elements would make it closed). We always know whether the current set is closed or not. In this paper we will show the way we check it, and it differs from the way presented in [1] [2]. We also introduce the approach used in MONSA and the correspondence of its basics and concepts to the approach by Zaki and Hsiao.

1.2 Basic idea of our approach

The fundamental idea of our approach is very simple: frequencies determine a frequent set. We collect the frequencies into a frequency table, which shows how many times different values of attributes appear in data set. For example in Table 1 is given a set of two objects described by three attributes and its corresponding frequency table.

Table 1. Example X(2,3)

Object \ Attribute	*A1*	*A2*	*A3*
O1	1	2	2
O2	1	1	2

Value \ Attribute	*A1*	*A2*	*A3*
1	**2**	1	0
2	0	1	**2**

Frequent set = A1.1 AND A3.2

As we see all elements (i.e. attribute with certain value) that have a frequency equal to the number of objects belong to the frequent set. The frequent set with such properties can be found as an intersection through the table.

Table 2. Intersection through X(2,3)

Object \ Attribute	*A1*	*A2*	*A3*
O1	1	2	2
O2	1	1	2
Intersection	**1**	*	**2**

$IntSec_X$ = A1.1 AND A3.2

As we see from Table 1 and Table 2 an intersection is also findable through the frequencies: every value which has frequency equal to the number of objects in the table belongs to the intersection. But we need special techniques to form tables with this property. Then for every table we have formed we find an intersection over the frequency table as we did in Table 1.

We start the description of our approach with definitions in 2. Terms used in closed set mining are given in 2.1 and definitions used explaining MONSA and the relations between terms of two approaches are shown in 2.2. Section 3 describes MONSA, including the algorithm in 3.1, an example of the working process in 3.2, and the explanation of used pruning techniques in 3.3. Section 4 concludes this paper.

2 Definitions

2.1 Definitions of closed set mining

Definitions used in closed set mining are presented here as in [1] [2].

Typically the database is arranged as a set of transactions, where each transaction contains a set of items. Let $I = \{1,2,...,m\}$ be a set of items, and let $T = \{1,2,...,n\}$ be a set of transaction identifiers or *tids*.

A set $X \subseteq I$ is also called an *itemset*, and a set $Y \subseteq T$ is called *tidset*.

$t(X)$ denotes a tidset that corresponds to an itemset X, i.e. the set of all tids that contain X as a subset: $t(X) = \cap_{x \in X} t(x)$.

$i(Y)$ denotes an itemset that corresponds to a tidset Y, i.e. the set of items common to all the tids in Y: $i(Y) = \cap_{y \in Y} i(y)$.

The *support* of an itemset X, denoted $\sigma(X)$, is the number of transactions in which it occurs as a subset.

An itemset is *frequent* if its support is more than or equal to a user-specified *minimum support* (*min_sup*) value, i.e. if $\sigma(X) \geq min\_sup$.

A frequent itemset X is called *closed* if there exists no proper superset $Y \supset X$ with $\sigma(X) = \sigma(Y)$.

Closed sets are found using closure operation.

A *closure* of an itemset X, denoted $c(X)$, is defined as the smallest closed set that contains X. An itemset X is closed if and only if $X = c(X)$.

The closure of an itemset X is found as: $c(X) = i(t(X))$.

The support of an itemset X is also equal to the support of its closure, i.e. $\sigma(X) = \sigma(c(X))$.

2.2 Definitions used in MONSA

The first version of MONSA was presented in 1993 [3]. Here we present denotations and definitions used

for MONSA as in [3]. We also give comments how these notions and concepts relate to the ones used for closed sets (see 2.1).

(1) X - a set X = {Xi}, i = 1,2,...,N,
where each object Xi is a conjunction of M attribute values: $Xi = \overset{M}{\underset{j=1}{\&}} h_j$.

X is a set of N objects (records) that are described by M attributes. Set X has not to be a binary dataset, every attribute j can acquire integer values in the interval $h_j = 0,1,2,...,K_j\text{-}1$. X can be a transaction database also.

(2) H - a value combination (VC) of certain attributes $H = \underset{q\in D}{\&} h_q$, D = {$j_e$}, e = 1,...,$E_H$ (number of elements h_q in H), $1 \le E_H \le M$, $1 \le j_e \le M$, $j_f, j_t \in D$, $j_f \# j_t$, f # t, $H \subseteq Xi$.

A value combination can contain 1 to M attributes (with certain values), each only once (i.e. only with one value); it is a subset of some object or is a whole object.

(3) Each value combination H defines on the set X a subset of objects X_H = {Xp}, p = 1,2,...,N_H, $1 \le N_H \le N$,
{Xp} are all objects Xi ∈ X that contain H: X_H = {Xi∈X | Xi ⊇ H}.

The subset of objects defined by the value combination is similar to the tidset that corresponds to some itemset (*t(X)*).

(4) Each value combination H defines on set X a subset of elements $X^H \subseteq X$: X^H = {Xij ∈ X | Xij ∈ H, i = 1,2,...,N, j = 1,2,...,M}.

An attribute with certain value is called element. 'Element' corresponds to 'item'. The difference is that each attribute produces as many elements as many different values it has. Definition (4) says that 'value combination' is the same as 'itemset' (with extension that values need not be binary).

(5) Intersection over a set Y = {Yt}, t = 1,2,...,T, $Yt = \&h_j$ is a set of such elements h_q which belong simultaneously to all Yt: $\cap Y = \overset{T}{\underset{t=1}{\cap}} Yt = \underset{q}{\&} h_q = H$.

In Y for H there exists always a corresponding subset of objects Y_H = {Yp}, p = 1,...,N_H, $N_H \le N$.

If N = 1, then intersection over Y is an object itself.

If there exist no objects Yt ∈ Y for which H ⊂ Yt, then $Y_H = \varnothing$.

Definition (5) says that intersection over a set of objects is a set of common elements, this is the same as itemset that corresponds to some tidset (*i(Y)*).

(6) Elementary conjunction (EC) on X_H is such an intersection over the set X_H, where $\cap X_H = A(\supseteq H)$, $X_A = X_H$, $N_H \le N$.

In the case of $A \supset H$, $X_A = X_H$, A is an EC.

We have a set of elements H and its corresponding set of objects X_H (i.e. *t(H)*) and find an intersection over it: $\cap X_H$ (i.e. *i(t(H))*). The operation *i(t(H))* means finding the closure of H. Therefore the resultant set A (of elements) called elementary conjunction is a closed set. From the viewpoint of the algorithm it is essential to find a technique that guarantees the finding of such subsets X_H only for which $\cap X_H = A(\supseteq H)$ (i.e. finding of closed sets only).

(7) Maximal EC on X_H is such an intersection over X_H in case of which for a VC $H = \underset{q}{\&} h_q$ is a valid relation
$\cap X_H = A(= \underset{e}{\&} h_e) \supset H$, $1 \le q < e \le M$, $X_A = X_H$.

By definition, VC H is EC if $\cap X_H = H$.

H is a maximal EC if it is EC and contains at least one VC $Ht \subset H$ such, that $|X_{Ht}| = |X_H|$ on X. That means that the set of objects $X_H \subseteq X$ is defined unique.

Definition (7) says that maximal EC is EC that has at least one (non-closed) subset with the same frequency (support) i.e. we can remove at least one element without changing in frequency. Our maximal elementary conjunction here is not the same as maximal (closed) set[1].

Additionally, our '(absolute) frequency' is the same as 'support'.

For rules we use also 'relative frequency' which corresponds to 'confidence'.

3 MONSA

MONSA is a depth-first search algorithm. Without pruning techniques it would traverse the complete search tree and find all permutations of all existing value combinations. Empty (i.e. non-existing) combinations are avoided by nature of algorithm, they are not generated (and checked for existence) at all.

[1] "A frequent itemset X is called maximal if it is not a subset of any other frequent itemset." [2]

From the initial data set MONSA finds a result as a set of intersections (closed sets) and/or a set of trees (forest), they are listed in the order they are found, that order does not depend on the initial order of objects (records). The frequencies of nodes (intersections) decrease strictly along branches of a tree(s). The decrease makes allowable to prune the branches according to the minimal allowed frequency (support). This is similar to the other tree-based algorithms, including ChARM [2]. The fact that the decrease is strict gives a high potential to the intersection (combination from root to the certain node) to be a closed set. At any level the descendants of a common parent-node are found in a weakly decreasing order of their frequencies, the roots also are found in a weakly descending order. The order of nodes with equal frequency depends on the searching principle (usually by columns or by rows of frequency table).

MONSA uses frequency tables (introduced in section 1.2). For every extract of objects its corresponding frequency table is formed. Zero in the initial frequency table means that the corresponding element does not exist. During the work the elements that are exhaustively analyzed are zerofilled in frequency tables. Such prohibited (eliminated) elements are not included into the intersections any more, only the elements with frequency over zero (or some higher threshold) are considered.

3.1 Description of the algorithm

MONSA finds intersections in given set X(N,M), where N is the number of objects (for example transactions), M is the number of attributes and each attribute j has an integer value h_j=0,1,2,…,K-1. A pair of attribute and its certain value is called element.

By essence MONSA is a recursive algorithm. Here its backtracking version is presented.

In this algorithm the following notation is used:

t	the number of the step (or level) of the recursion
FT_t	frequency table for a set X_t
$IntSec_t$	vector of elements over set X_t (intersection)
Init	activity for initial evaluation

The algorithm is given in Fig. 1.

It has been proved that if a finite discrete data matrix X(N,M) is given, where $N=K^M$, then the complexity of algorithm MONSA to find all $(K+1)^M$ elementary conjunctions (intersections) as existing value combinations is $O(N^2)$ operations [3].

By our estimation in practice the upper bound of the number of value combinations (with minimal frequency allowed = 1) is

$$L_{UP} \approx N(1+1/K)^M, \quad (1)$$

but usually it is less.

The precise formula for the number of intersections is as follows:

$$L = \sum_{f=1}^{F} \sum_{p=1}^{(M*K-u)} N_p \ , \quad (2)$$

where F is the number of formatted frequency tables on set X, u is the number of empty cells in the frequency table FT_f, N_p is the absolute number in a cell of the frequency table (frequency of an attribute value).

The most important advantages of the algorithm are:

- The use of new original and very effective pruning techniques
- The possibility to output the tree immediately during the finding closed sets
- The frequency is known at the moment the new node is found
- The ability to find nodes consisting of more than one element, this reduces the number of nodes and the size of the tree
- In order to prevent repetitions we do not look through the already found result and therefore we do not need additional data structures
- Attributes can have more values than only 0/1

An example in the following chapter explains how MONSA works.

```
Algorithm MONSA

Init
t←0, IntSec0←{}
Find a table of frequencies FT0 for all attributes in X0
DO WHILE there exists FTs#Ø in {FTs}, s≤t
  FOR EACH element hf∈FTt with frequency V=max FTt(hf)#0 DO
    IF pruning is needed (hf has to be pruned) THEN GOTO BACK
    Separate submatrix Xt+1⊂Xt such that Xt+1={Xij∈Xt | X.f=hf}
    Find a table of frequencies on Xt+1 FTt+1
    ZeroesDown(t+1)
    CheckUniqueness(t+1)
    IF new intersection is unique THEN
      Add elements j with FTt+1(j)=V into vector IntSect+1
      BackwardComparison(t+1)
      Output of IntSect+1
      IF there exist attributes to analyse THEN t←t+1
    ENDIF
  NEXT
  BACK: t←t-1
  IntSect+1←IntSect
ENDDO
All intersections are found
END: end of algorithm
```

Elimination (pruning) activities:

```
1)
ZeroesDown(t+1)
    FOR EACH element hu∈FTt DO
      IF FTt(hu)=0 THEN FTt+1(hu)←0
    NEXT

2)
BackwardComparison(t+1)
    FOR EACH element hu∈FTt+1 with frequency #0 DO
      IF FTt+1(hu)=FTt(hu) THEN FTt(hu)←0
    NEXT

3)
CheckUniqueness(t+1)
    IF there exists on Xt+1 hu, 1≤u≤M such, that
    [hu∈IntSect+1 AND FTt+1(hu)=0 AND frequency of hu in Xt+1=V]
    THEN
      Intersection is not unique
    ELSE
      Intersection is unique
    ENDIF
```

Fig. 1. Algorithm MONSA

3.2 Steps of MONSA

In order to explain the work of MONSA we will demonstrate it using the initial data set given in Table 3.

In order to find intersections MONSA uses frequency tables. A frequency table contains the counts of occurrences of all existing values for each attribute. Each attribute can have a different number of different discrete values.

The frequency table corresponding to the initial data from Table 3 is given in Table 4.

Table 3. Initial data

Object \ Attribute	*A1*	*A2*	*A3*
O1	1	0	3
O2	2	2	1
O3	2	3	0
O4	2	0	2
O5	0	1	3
O6	0	1	3
O7	1	1	2
O8	1	0	3
O9	2	3	0

Table 4. Frequency table (FT)

Value \ Attribute	*A1*	*A2*	*A3*
0	2	3	2
1	3	3	1
2	4	1	2
3	0	2	4

MONSA is a depth-first search algorithm which backtracks when the current branch is exhausted or has to be pruned. Inside the backtracking algorithm the main steps at each level are as follows:

S1: Choose a "leading" element – the first element with maximal frequency (over zero; at least with threshold frequency specified by the user), add it into the (potential) intersection and zerofill the corresponding cell in the frequency table

S2: Calculate the next frequency table for objects containing that leading element

S3: If there exist element(s) with frequency equal to the leading one check the intersection's uniqueness => if it is unique, add these elements (with frequency equal to the leading frequency) into the intersection; otherwise backtrack

S4: Output intersection

S5: "Bring down" zeroes from the frequency table of the previous level i.e. at the current level zerofill all elements that have been zerofilled at the previous level

S6: "Backward comparison": elements that have equal frequencies at both levels are zerofilled at the previous level

Next we will show how the algorithm works. For better understanding we give a frequency table (FT) for each level, the number in subscript shows the level of recursion (starting from the level 0). The row above the table header shows the current potential intersection, where '*' means that certain value for that attribute is not determined yet. The number after "=" is the frequency of shown intersection. We also use a non-positional representation of intersections, for example: A2.0=3 – the intersection consists of one element – attribute A2 with value 0 and its frequency is 3.

During the work we zerofill certain elements in the frequency tables for different reasons. In order to mark those elements the following notation is used:

→0 – a zerofilled leading element (the result of step S1);

! – elements with frequency equal to the leading frequency that are set to zero in case of inclusion into intersection (the result of step S3);

↓0 – zeroes "brought down" from the previous level frequency table (the result of step S5);

↑0 – frequencies equal at current and previous level (set to zero at the previous level, the result of step S6).

Now we will find all intersections with frequency at least 2 for data given in Table 3. This threshold – minimal frequency allowed (=2) – is set by the user.

Having the frequency table for the initial data set (see Table 5a), the first thing (S1) is to find an element with maximal frequency. If the search direction is by columns, then the first element with maximal frequency (which is 4) is attribute 1 (A1) with value 2. We include it into the intersection (2 * * =4; shown above Table 6). In order to prevent the repeating selection of it (A1.2) we also eliminate this element from the frequency table FT_0 by putting zero into the corresponding cell (marked with →0 in Table 5b).

Table 5. Frequency table at level 0

a)

FT_0	*A1*	*A2*	*A3*
0	2	3	2
1	3	3	1
2	4	1	2
3	0	2	4

b)

FT_0	*A1*	*A2*	*A3*
0	2	3	2 ↑0
1	3	3	1 ↑0
2	4→0	1 ↑0	2
3	0	2 ↑0	4

S2: The "leading" element A1.2 is the basis for extracting a subset of objects, i.e. only objects that

contain that certain element (A1.2) belong to the subset. As the frequency shows this subset consists of four objects, namely O2, O3, O4 and O9. We calculate a frequency table for the next level by taking into account only those (four) objects. As attribute A1 (with its value 2) was the basis of separating the subset (and it is already included into the intersection), we do not look at this attribute at the current level any more and the new frequency table FT_1 (given in Table 6a) does not contain frequencies for attribute A1.

Table 6. Frequency table for A1.2 (at level 1)

a)

2 * * =4

FT_1	*A1*	*A2*	*A3*
0		1	2
1		0	1
2		1	1
3		2	0

b)

FT_1	*A1*	*A2*	*A3*
0		1	2 ↑0
1		0	1
2		1	1
3		2→0	0

S3, S4: In the new frequency table (see Table 6a) there are no frequencies equal to the leading one (=4), therefore we can output intersection (2 * * =4 or A1.2=4) without "uniqueness check".

S5: There are no zeros to bring down from the initial frequency table (Table 5a) because the only zerofilled element is the leading element.

S6: The next thing to do is to compare the frequencies of the current level (Table 6) with the frequencies of the previous level (Table 5) and find four coincidences (A2.2, A2.3, A3.0 and A3.1). It means that all objects containing (some of) these elements belong to our current subset and will be fully analysed at the current and subsequent levels. Using those elements for extracting subsets at the previous level would cause repetitions. In order to prevent it we zerofill these four cells at the previous level (marked with ↑0 in Table 5b).

Now we will repeat all these activities at the next level.

S1: The (first) element with maximal frequency in FT_1 (Table 6a) is A2.3 with frequency 2, we remember it in the next potential intersection (2 3 * =2; shown above Table 7) and eliminate it (→0 in Table 6b). S2: Next we extract the subset of objects having this element (A2.3) and find the corresponding frequency table (see Table 7).

Table 7. Frequency table for A1.2&A2.3 (at level 2)

0 !

2 3 * =2

FT_2	*A1*	*A2*	*A3*
0			2 !
1			0
2			0
3			0

S3: This time there exists one element that has the frequency equal to the leading one (=2) – this element is A3.0 (! in Table 7). It means that all objects in the subset contain that element and we should include it into the intersection. The intersection would be (2 3 0) instead of (2 3 *). Before we can do it, we have to check whether this intersection (with A3.0) is unique (non-redundant) i.e. is not found already. The check is simple: if the frequency of that element is not zerofilled in the previous frequency table then this intersection is not found yet. The frequency of A3.0 at level 1 (Table 6a) is more than zero, so the new intersection (2 3 0) is unique and is outputted (A1.2&A2.3&A3.0=2). In order to prevent A3.0 to be a basis for extracting a new subset its frequency in the current frequency table (Table 7) will be set to zero after the "backward comparison" (S6).

S5: There are no zeroes to bring down in the column A3 (which is the only attribute under consideration). S6: Comparison of levels 1 and 2 (Table 6a and Table 7) sets (the same) A3.0 to zero at level 1 (↑0 in Table 6b).

Nullifying A3.0 at level 2 (Table 7) makes this frequency table empty. So we will continue at level 1 (Table 6b). Appears that there are no frequencies over the threshold (minimal frequency allowed =2).

So we backtrack to the initial level. Taking into account all zerofilling information (shown in Table 5b) the frequency table looks like in Table 8a. We find that element with maximal frequency is A3.3 (with frequency 4). S1: We include this element into the intersection (* * 3 =4; shown above Table 9) and set to zero its frequency in FT_0 (→0 in Table 8b).

Table 8. Frequency table at level 0

a)

FT_0	*A1*	*A2*	*A3*
0	2	3	0
1	3	3	0
2	0	0	2
3	0	0	4

b)

FT_0	*A1*	*A2*	*A3*
0	2 ↑0	3	0
1	3	3	0
2	0	0	2
3	0	0	4→0

The frequency table for A3.3 is given in Table 9a (S2). It contains no frequencies equal to the leading

one (=4), so the intersection A3.3=4 is outputted (S3, S4).

There are no zeros to bring down from FT_0 (Table 8a) (S5). Backward comparison (S6) sets to zero element A1.0 at level 0 (↑0 in Table 8b).

Table 9. Frequency table for A3.3 (at level 1)

a)

	*	*	3	=4
FT_1	*A1*	*A2*	*A3*	
0	2	2		
1	2	2		
2	0	0		
3	0	0		

b)

FT_1	*A1*	*A2*	*A3*
0	2→0	2	
1	2	2 ↑0	
2	0	0	
3	0	0	

We continue at level 1 (see Table 9a).

S1: The maximal frequency is 2, the first element with it is A1.0. We add it into intersection (0 * 3; shown above Table 10) and set to zero its frequency (→0 in Table 9b).

S2: We find the frequency table for the next level i.e. level 2 (see Table 10).

Table 10. Frequency table for A3.3&A1.0 (at level 2)

	0	1 ! *	3	=2
FT_2	*A1*	*A2*	*A3*	
0		0		
1		2 !		
2		0		
3		0		

S3: In FT_2 (Table 10) A2.1 has the frequency equal to the leading one (=2). Its frequency at previous level (see Table 9a) is not zeroed, so the intersection with A2.1 (0 1 3) is unique and is outputted (A3.3&A1.0&A2.1=2; S4).

S5: There are no zeros to bring down in column A2.

S6: (that same) A2.1 has equal frequencies at both levels (1 and 2; see Table 9a and Table 10), its frequency at level 1 is zerofilled (↑0 in Table 9b).

We zerofill A2.1 at level 2 also (due to the inclusion into the intersection). The frequenency table FT_2 (Table 10) becomes empty, we backtrack to level 1. After zerofilling (see Table 9b) the frequency table for A3.3 looks like in Table 11a.

Table 11. Frequency table for A3.3 (at level 1)

a)

	*	*	3	=4
FT_1	*A1*	*A2*	*A3*	
0	0	2		
1	2	0		
2	0	0		
3	0	0		

b)

FT_1	*A1*	*A2*	*A3*
0	0	2 ↑0	
1	2→0	0	
2	0	0	
3	0	0	

S1: We find that first element with maximal frequency (equal to 2) is A1.1, include it into the intersection (1 * 3; shown above Table 12) and zerofill the corresponding cell (→0 in Table 11b). S2: We calculate the frequency table for the next level (see Table 12). S3: Element A2.0 has the frequency equal to 2. As this element is not zerofilled at the level 1 (Table 11a), we include it into the intersection (1 0 3) and output (S4) the intersection (A3.3&A1.1&A2.0=2). S5: There are no zeros to bring down. S6: Element A2.0 has equal frequencies at both levels (see Table 11a and Table 12) and we zerofill it at level 1 (marked with ↑0 in Table 11b). Due to the inclusion into the intersection the element A2.0 is set to zero at level 2 also. This frequency table (Table 12) is empty now.

Table 12. Frequency table for A3.3&A1.1 (at level 2)

	1	0 ! *	3	=2
FT_2	*A1*	*A2*	*A3*	
0		2 !		
1		0		
2		0		
3		0		

The frequency table at level 1 (Table 11b) is also empty (taking into account the zerofilling information).

The frequency table at level 0 is shown in Table 13a.

Table 13. Frequency table at level 0

a)

FT_0	*A1*	*A2*	*A3*
0	0	3	0
1	3	3	0
2	0	0	2
3	0	0	0

b)

FT_0	*A1*	*A2*	*A3*
0	0	3	0
1	3→0	3	0
2	0	0	2
3	0	0	0

S1: The maximal frequency is 3 and the first element with it is A1.1. We add it into the intersection (1 * * =3) and nullify the corresponding cell (→0 in Table 13b) S2: The frequency table for objects containing A1.1 is found (see Table 14a). S3, S4: It contains no frequencies equal to 3, so the intersection (shown above Table 14) is outputted as it is (A1.1=3).

Table 14. Frequency table for A1.1 (at level 1)

a)

1 * * =3

FT_1	*A1*	*A2*	*A3*
0		2	0
1		1	0
2		0	1
3		0	2

b)

FT_1	*A1*	*A2*	*A3*
0		2	0
1		1	0
2		0	1
3		0	2 ↓0

S5: The cell for A3.3 has been zerofilled at level 0 (Table 13) and this zero is now brought down to level 1 (↓0 in Table 14b). This zero prevents traversing A1.1&A3.3 (which is the subset of already found intersection A3.3&A1.1&A2.0=2 with the same frequency and therefore redundant) and subsequent subsets. S6: Backward comparison finds no equal frequencies.

Table 15. Frequency table for A1.1 (at level 1)

a)

1 * * =3

FT_1	*A1*	*A2*	*A3*
0		2	0
1		1	0
2		0	1
3		0	0

b)

FT_1	*A1*	*A2*	*A3*
0		2→0	0
1		1	0
2		0	1
3		0	0

S1: At current level (see Table 15a) the maximal frequency is 2 and element with it is A2.0 (zerofilling is shown in Table 15b). S2: The frequency table for objects containing A2.0 is given Table 16 and the current intersection (1 2 * =2) is above it.

Table 16. Frequency table for A1.1&A2.0 (at level 2)

1 0 * =2

FT_2	*A1*	*A2*	*A3*
0			0
1			0
2			0
3			2 !

S3: In FT_2 (Table 16) element A3.3 has frequency 2 (equal to the leading one). The "uniqueness check" (i.e. looking into the corresponding cell in the previous frequency table) gives the answer: such intersection (with A3.3) – (1 0 3) has been found already and current subset has been analysed already. In such situation we backtrack to the previous level (without outputting anything).

The frequency table at level 1 (Table 15b) contains no frequencies over the threshold (=2), therefore we backtrack again.

Table 17. Frequency table at level 0

a)

FT_0	*A1*	*A2*	*A3*
0	0	3	0
1	0	3	0
2	0	0	2
3	0	0	0

b)

FT_0	*A1*	*A2*	*A3*
0	0	3→0	0
1	0	3	0
2	0	0	2
3	0	0	0

At level 0 (Table 17a) the maximal frequency is 3 and the first element with such frequency is A2.0 (zerofilling is shown in Table 17b). S1, S2: we add it into the intersection (* 0 *) and find the next frequency table (Table 18a).

Table 18. Frequency table for A2.0 (at level 1)

a)

* 0 * =3

FT_1	*A1*	*A2*	*A3*
0	0		0
1	2		0
2	1		1
3	0		2

b)

FT_1	*A1*	*A2*	*A3*
0	0		0
1	2 ↓0		0
2	1 ↓0		1
3	0		2 ↓0

S3, S4: The current intersection is outputted as it is (A2.0=3), because there are no elements with the same frequency. S5: The comparison of frequency tables of level 1 (Table 18) and level 0 (Table 17) finds 3 zeros to bring down (for A1.1, A1.2, A3.3; shown in Table 18b). S6: Backward comparison finds no coincidences of frequencies.

The current frequency table (Table 18b) contains no frequencies over 2, therefore we go back to the level 0 (see Table 19a).

Table 19. Frequency table at level 0

a)

FT_0	*A1*	*A2*	*A3*
0	0	0	0
1	0	3	0
2	0	0	2
3	0	0	0

b)

FT_0	*A1*	*A2*	*A3*
0	0	0	0
1	0	3→0	0
2	0	0	2
3	0	0	0

S1: Include element A2.1 with frequency 3 into the intersection (* 1 * =3; shown above Table 20) and zero its frequency (→0 in Table 19b).

S2: Find the frequency table for those 3 objects (Table 20a).

Table 20. Frequency table for A2.1 (at level 1)

a)

* 1 * =3

FT_1	*A1*	*A2*	*A3*
0	2		0
1	1		0
2	0		1
3	0		2

b)

FT_1	*A1*	*A2*	*A3*
0	2 ↓0		0
1	1 ↓0		0
2	0		1
3	0		2 ↓0

S3: No need for uniqueness check.

S4: Output intersection (A2.1=3).

S5: Bring down zeros from FT_0 (Table 19) to FT_1 (marked with ↓0 in Table 20b).

S6: Backward comparison gives no consequencies.

The frequency table at level 1 (Table 20b) contains no frequencies over the threshold, so we turn back to the level 0 (see Table 21).

Table 21. Frequency table at level 0

a)

FT_0	*A1*	*A2*	*A3*
0	0	0	0
1	0	0	0
2	0	0	2
3	0	0	0

b)

FT_0	*A1*	*A2*	*A3*
0	0	0	0
1	0	0	0
2	0	0	2→0
3	0	0	0

S1: Include element A3.2 with frequency 2 into the intersection (* * 2 =2).

S2: Find the frequency table for the next level (see Table 22a).

Table 22. Frequency table for A3.2 (at level 1)

a)

* * 2 =2

FT_1	*A1*	*A2*	*A3*
0	0	1	
1	1	1	
2	1	0	
3	0	0	

b)

FT_1	*A1*	*A2*	*A3*
0	0	1 ↓0	
1	1 ↓0	1 ↓0	
2	1 ↓0	0	
3	0	0	

S3: No uniqueness check.

S4: Output intersection (A3.2=2).

S5: Bring down zeros (for A1.1, A1.2, A2.0, A2.1) – see Table 22b.

S6: Backward comparison finds no elements with equal frequencies at both levels.

The frequency table FT_1 is empty (i.e. contains only zeros). We backtrack to level 0.

The frequency table at level 0 (Table 21b) is also empty.

There are no more levels to go back, thus the algorithm has finished its work. All intersections are found.

As a result we have found 9 intersections (with minimal frequency allowed =2):

2 * *	A1.2=4
2 3 0	A1.2&A2.3&A3.0=2
* * 3	A3.3=4
0 1 3	A3.3&A1.0&A2.1=2
1 0 3	A3.3&A1.1&A2.0=2
1 * *	A1.1=4
* 0 *	A2.0=3
* 1 *	A2.1=3
* * 2	A3.2=2

They all are different (unique), and they are not empty. All three pruning techniques together prevented repetitions, both full repetitions (in different order of elements) and partial repetitions (i.e. subsets of intersections with the same frequency). Uniqueness check was applied only when there was more than one element to include into the intersection at the same step (level) and this check was made without looking through the already found intersections.

Instead of list of intersections the same result can be listed as a set of trees (immediately during the process of finding intersections):

```
(4) 0.500(2)
A1.2=>A2.3&A3.0


(4) 0.500(2)
A3.3=>A1.0&A2.1

    0.500(2)
    =>A1.1&A2.0


(3)
A1.1


(3)
A2.0


(3)
A2.1

(2)
A3.2
```

The trees are represented from left to right. This example consists of six trees, it has six root nodes (on the left). Symbols '=>' separate the nodes of a tree.

Usually a node contains one element – a pair of certain attribute and a certain value of that attribute.

Attribute is shown before period and value is after period. For example, the root node of the first tree contains attribute A1 and its value 2. A node can consist of more than one attribute-value pairs like all non-root nodes in this example, then '&' is used to connect them.

The numbers above node show frequencies.

In the parentheses node's absolute frequency is shown. The absolute frequency of node t shows how many objects have a certain attribute(s) with a certain value(s) among objects having properties (i.e. certain attributes with certain values) of all previous levels t-1,…,1. For example, the first tree shows that among four objects with A1.2 there are two objects having both A2.3 and A3.0.

Before parentheses node's relative (to the previous level) frequency is shown. The relative frequency is a ratio A/B, where A is the absolute frequency of node t and B is the absolute frequency of node t-1. For the first level the relative frequency is not calculated.

3.3 Pruning techniques used in MONSA

In order to avoid finding "repetitions" i.e. permutations of already found intersections two pruning techniques have been used in algorithm MONSA:

"bringing zeroes down" – activity that prohibits arbitrary output repetition of already separated intersection at the next (deeper) level(s);

"backward comparison" – activity that does not allow the output of the separated intersection at the same (current) level and also at previous (higher) levels (after backtracking).

Impact of these techniques is proved in [3][2].

Appears that these two activities do not prevent repetitious finding (and output) of some subsets of already found intersections.

The subset of already found intersection is redundant only if both have equal frequencies (i.e. this (sub)set is non-closed). If subset's frequency is higher (than its superset's) then it covers more objects and is not redundant. Subset's frequency cannot be lesser.

Finding a superset of already found set with the same frequency is impossible, because at any level MONSA finds all co-existing (i.e. contained in the same objects) elements with equal frequencies as one intersection (i.e. maximal EC).

[2] Theorems 5.3 and 5.4 accordingly

Next we give an example of redundant subsets under consideration. For that we have used MONSA without "uniqueness check" (of a new intersection).

Having the initial data set of nine objects described by three attributes (see Table 23) MONSA (without uniqueness check) finds fifteen intersections (with minimal frequenct allowed =2). Both representation forms – trees and intersections – are given in Fig. 2.

Table 23. Example X(9,3)

Object \ Attribute	*A1*	*A2*	*A3*
O1	4	0	8
O2	5	3	4
O3	5	4	3
O4	5	0	7
O5	3	1	8
O6	3	1	8
O7	4	1	7
O8	4	0	8
O9	5	4	3

(a) Result as a set of trees:

```
(4) 0.500(2)
A1.5=>A2.4&A3.3
    0.250(1)
    =>A2.0&A3.7
    0.250(1)
    =>A2.3&A3.4

(4) 0.500(2)
A3.8=>A1.3&A2.1
    0.500(2)
    =>A1.4&A2.0

 (3) 0.667(2)
A1.4=>A2.0
    0.333(1)
    =>A2.1&A3.7

 (3) 0.333(1)
A2.0=>A3.7

 (3) 0.333(1)
A2.1=>A3.7

 (2)
A3.7
```

(b) Result as a set of intersections:

```
I1)  A1.5=4
I2)  A1.5&A2.4&A3.3=2
I3)  A1.5&A2.0&A3.7=1
I4)  A1.5&A2.3&A3.4=1

I5)  A3.8=4
I6)  A3.8&A1.3&A2.1=2
I7)  A3.8&A1.4&A2.0=2

I8)  A1.4=3
I9)  A1.4&A2.0=2
I10) A1.4&A2.1&A3.7=1

I11) A2.0=3
I12) A2.0&A3.7=1

I13) A2.1=3
I14) A2.1&A3.7=1

I15) A3.7=2
```

Fig. 2. Result found by MONSA (without uniqueness check)

Here we have six trees and six roots accordingly. Intersections I1, I5, I8, I11, I13 and I15 correspond to the roots.

Intersection I11 (A2.0=3) is not redundant although A2.0 is contained in the intersections I3, I7

and I9, because these three intersections have lesser frequencies and none of them cover all objects covered by I11.

Intersections I9, I12 and I14 are redundant:

- I9 (A1.4&A2.0) is a subset of I7 (A3.8&A1.4&A2.0) with the same frequency =2 (both cover objects O1 and O8);
- I12 (A2.0&A3.7) is a subset of I3 (A1.5&A2.0&A3.7), both frequencies are 1 (they cover object O4) and
- I14 (A2.1&A3.7) is a part of I10 (A1.4&A2.1&A3.7), frequencies equal 1 (they cover O7).

There have to be 12 intersections instead of 15.

Starting from the root of a tree the frequencies of intersections (nodes) always (strictly) decrease along any branch of the tree (due to finding maximal EC at every node). As no branch has two intersections with the same frequency, the redundant subsets do not occur in the same branch, they can appear in different branches of a tree or in different trees.

Among siblings (i.e. direct descendants of a common node) any element can appear in only one intersection. Consequently, redundant subsets (under consideration) do not occur among siblings. This is also true for the root-level (which formally consists of descendants of the initial empty set), therefore these redundant subsets never occur at root-level.

Intersection (closed set) and its redundant sub-intersection (with the same frequency) do not have to appear at the same level of a tree (as in all three cases of our example).

Element(s) that appear in the root node are eliminated from further analysis by zerofilling the corresponding cell(s) in the frequency table. Elements that are fully analyzed (exhausted) at deeper levels are prohibited by "backward comparison". All these zeroes are "brought down" to all succeeding levels and therefore prohibited elements never occur in redundant intersections. So (due to the elimination techniques used) no permutation of whole (already found) intersection (*closed set*) is not found. Elements that are partially analyzed (at the non-root levels) are not eliminated. All this is true for any subtree also.

Although prohibited elements are eliminated from the frequency tables they still appear in the subsets of objects extracted by non-prohibited elements. Remind that some set (of elements) and its subset with the same frequency define the same set of objects. If some prohibited (and eliminated) element appears in all objects (of subset) it means that this subset has been analyzed already (this element can not be contained in the value combination (potential intersection) on which basis that subset of objects was extracted). All intersections containing a prohibited element are already found and such subsets need no more analysis. This situation indicates a repetitive extraction of certain subset of objects.

To exclude redundant subsets (sub-intersections) we have to detect a situation when some prohibited element occurs in all objects and stop analyzing such branch.

In our example (see Table 23 and Fig. 2) intersection I7 (A3.8&A1.4&A2.0) has one more element than redundant set I9 (A1.4&A2.0), namely A3.8. The set of objects extracted by I9 is the same as by I7. Consequently, actual frequency of A3.8 is as much as number of objects in that set (=2). As A3.8 was prohibited after exhaustive analyze, the frequency table contains zero in the corresponding cell. Detecting such situation we can say that A1.4&A2.0 (I7) is a redundant set.

Such "uniqueness check" is used by MONSA. It is not necessary to look through the already found intersections to ensure the new one is non-redundant.

Correctness of 'uniqueness check' explained here is proved in [4].

It is interesting that those redundant subsets seem to be the same ones for what ChARM [1] [2] needs "subsumption checking". In order to ensure that a candidate set is really closed ChARM looks through the (certain) already found closed sets, only those that have 1) the same "tidsum" and 2) the same support (frequency) as the candidate set. (Tidsums of different closed sets with equal frequency tend to be different). For the complete description see [1] [2].

As shown already we perform the uniqueness check of a new intersection (potential closed set) otherwise, without looking through the already found (closed) sets.

4 Conclusion

In this paper we have described algorithm MONSA for finding closed sets. The first version of MONSA was created in 1993 and we have developed several versions during last years.

From the theoretical viewpoint the basic concepts used in MONSA are compared to the ones used by Zaki and Hsiao and it is shown that substantially they are the same.

In MONSA we use some new effective pruning techniques: zeros down, backward comparison, uniqueness check. These are simple original techniques to prevent repetitious finding of already found closed sets whitout using additional data

structures. Uniqueness (non-redundancy) check of a new potential closed set is made without looking through the already found closed sets.

During the process of finding all closed sets MONSA can find also rules between them. The algorithm works not only on the binary scale.

MONSA is a basic algorithm we have used not only in data mining [5] [6] [7], but also in solving of graph theoretical problems as extracting of all maximal cliques [4] and decision trees constructing [8]. We have some approaches how to use the algorithm presented here in machine learning for the future works.

Acknowledgement

This work was supported in part by the Estonian Information Technology Foundation under Grant 07-03-00-22.

References:

[1] M. J. Zaki and C.-J. Hsiao, CHARM: An efficient algorithm for closed association rule mining, *Technical Report 99-10*, Department of Computer Science, Rensselaer Polytechnic Institute, October 1999.

[2] M. J. Zaki and C.-J. Hsiao, CHARM: An efficient algorithms for closed itemset mining, *Proceedings of the Second SIAM International Conference on Data Mining*, 2002.

[3] R. Kuusik, The Super-Fast Algorithm of Hierarchical Clustering and the Theory of Monotone Systems, *Transactions of Tallinn Technical University*, No 734, 1993, pp. 37-62.

[4] R. Kuusik, Extracting of all maximal cliques: monotone system approach, *Proceedings of the Estonian Academy of Sciences. Engineering*, No 1, 1995, pp. 113-138.

[5] R. Kuusik, G. Lind, L. Võhandu, Data mining: pattern mining as a clique extracting task, *Proceedings of the Sixth International Conference on Enterprise Information Systems*, Vol. 2, Porto, Portugal, 2004, pp. 519-522.

[6] R. Kuusik, G. Lind, New frequency pattern algorithm for data mining, *Proceedings of the 13th Turkish Symposium on Artificial Intelligence and Neural Networks*, Foça, Izmir, Turkey, 2004, pp. 47-54.

[7] G. Lind, Method for Data Mining – Generator of Hypotheses, *Databases and Information Systems. Proceedings of the 4th International Baltic Workshop*, Vol. 2, Vilnius, 2000, pp. 304-305.

[8] A. Torim, R. Kuusik, Problem and algorithms for finding the best decision, *WSEAS Transactions on Information Science and Applications*, 9 (2), 2005, pp. 1462-1469.