Mining Frequent Boolean Expressions: Application to Gene Expression and Regulatory Modeling

Mohammed J. Zaki[1], Naren Ramakrishnan[2], Lizhuang Zhao[3*]
[1] CS Dept., Rensselaer Polytechnic Institute, Troy, NY 12180
[2] CS Dept., Virginia Tech., Blacksburg, VA 24061
[3] Microsoft Corp., Redmond, WA 98052
{zaki, zhaol2}@cs.rpi.edu, naren@cs.vt.edu

Abstract

Regulatory network analysis and other bioinformatics tasks require the ability to induce and represent arbitrary boolean expressions from data sources. We introduce a novel framework, called BLOSOM, for mining (frequent) boolean expressions over binary-valued datasets. Boolean expressions can be grouped into four categories: pure conjunctions, pure disjunctions, conjunction of disjunctions, and disjunction of conjunctions. Our main focus on mining the simplest expressions (the *minimal generators*), but we also propose *closure* operators that yield *closed* (or unique maximal) boolean expressions. BLOSOM efficiently mines frequent boolean expressions by utilizing a number of methodical pruning techniques. Experiments showcase the behavior of BLOSOM for different input settings and parameter thresholds. Application studies on gene expression and gene regulation patterns showcase the effectiveness of our approach.

Keywords: Minimal Generators, Closed Patterns, Boolean Expressions, Itemset Mining

1 Introduction

A key class of problems in biological knowledge discovery pertain to modeling complex relationships between entities, such as GO category descriptions, genetic regulatory connections, and metabolic pathway networks. To effectively address these tasks, we require data mining techniques that possess the expressiveness to model *arbitrary* boolean connections. However, many state-of-the-art techniques are rather restrictive in the class of expressions they can mine.

For example, itemset mining (Agrawal et al., 1996) takes as input a binary-valued dataset and discovers patterns that are *pure conjunctions* of items. Admittedly, such techniques are useful in bioinformatics, but have limited scope. For example, given the expression levels (high or low) of genes for various cancers, we may find that cancerous tissues have high levels of genes (g_1 AND g_5 AND g_{13}) OR (g_{10} AND g_{15}). This might indicate that these groups of genes are co-regulated and somehow linked to cancer. These techniques can be generalized to include negations of descriptions and used to mine redescriptions (Ramakrishnan et al., 2004), that is, finding alternative ways of describing a group of genes. For instance, from yeast gene expression data, we may find that the gene highly expressed at time point 15 mins ($d183$), but not involved in mannose transportation

[*]This work was performed while the author was at RPI

($d388$) or fructose metabolism ($d515$) or not part of external protective structure ($d460$), are the exact same genes also highly expressed at time point 20 mins ($d184$), provided they do not have unknown molecular function ($d309$). This redescription can be written as an equivalence of boolean expressions: $d183$ AND (NOT $d388$) AND (NOT $d460$) AND (NOT $d515$) $\iff$ $d184$ AND (NOT $d309$).

We seek to generalize these approaches to mine arbitrary boolean expressions. Boolean expression mining can provide tremendous value, but there are two main challenges to contend with. The first deals with the problem of high computational complexity. With n items (or variables), there are 2^{2^n} possible distinct-valued boolean expressions[1], far too many to enumerate. To make the search practical we focus on only the frequent boolean expressions. Also instead of mining all frequent boolean expressions, we focus on mining a lossless subset that retains complete frequency information, namely *closed* boolean expression. The second challenge relates to comprehension of the patterns, i.e., they may be complex and difficult to understand. Here we focus on mining the simplest or *minimal* expressions (which are in fact the *minimal generators* of the closed expressions) that still from a lossless representation of all possible boolean expressions.

In this paper, we present a novel framework, called BLOSOM (an anagram of the bold letters in **BOOL**ean expression **M**ining over attribute **S**ets), the first such approach to simultaneously mine closed boolean expressions over attribute sets and their minimal generators.[2] Our main contributions are as follows: We organize boolean expressions into four categories: (i) pure conjunctions (AND-clauses), (ii) pure disjunctions (OR-clauses), (iii) conjunctive normal form (CNF; conjunction of disjunctions), and (iv) disjunctive normal form (DNF; disjunction of conjunctions). For both CNF and DNF expressions we propose a *closure* operator, and we give a characterization of the *minimal generators*. BLOSOM employs a number of effective pruning techniques for searching over the space of boolean expressions, yielding orders of magnitude in speedup. We conducted several experiments on synthetic datasets to study the behavior of BLOSOM with respect to (w.r.t.) different input settings and parameter thresholds. We also highlight some of the patterns found using BLOSOM on real datasets from bioinformatics applications such as analysis of gene expression patterns, and boolean gene regulatory network discovery.

2 Preliminary Concepts

Lattice Theory: Let us first review a few facts from lattice theory (Davey & Priestley, 1990), which will be useful in our discussion. Let $(P, \subseteq)$ be a partially ordered set (also called a *poset*). Let $X, Y \in P$ and let $f : P \to P$ be a function on P. f is called *monotone* if $X \subseteq Y \Rightarrow f(X) \subseteq f(Y)$. We say that f is *idempotent* if $f(X) = f(f(X))$. f is called *extensive* (or expansive) if $X \subseteq f(X)$. Finally, f is called *intensive* (or contractive) if $f(X) \subseteq X$. A *closure operator* on P is a function $\mathbb{C} : P \to P$ such that $\mathbb{C}$ is monotone, idempotent, and extensive. X is called *closed* if $\mathbb{C}(X) = X$. On the other hand, a *kernel operator* on P is a function $\mathbb{K} : P \to P$, which is monotone, idempotent, and intensive. X is called *open* if $\mathbb{K}(X) = X$. The set of all closed and open members of P form the fixed-point of the closure ($\mathbb{C}$) and kernel ($\mathbb{K}$) operators, respectively. Given two posets $(P, \subseteq)$ and $(Q, \leq)$, a *monotone Galois connection* between them consists of two *order-preserving* functions, $\phi : P \to Q$ and $\psi : Q \to P$, such that for all $X \in P$ and $Y \in Q$, we have: $X \subseteq \psi(Y) \iff \phi(X) \leq Y$. The

[1]With n items, there are 2^n distinct ways of assigning true/false values to the n terms, and for any such assignment the truth value of the expression can be set to true or false, giving the 2^{2^n} value.

[2]Please note that a preliminary version of this paper appeared as a *short paper* in the ACM SIGKDD Conference (Zhao et al., 2006).

composite function $\psi \circ \phi : P \to P$ is a closure operator, whereas the function $\phi \circ \psi : Q \to Q$ is a kernel operator, on P and Q, respectively (Davey & Priestley, 1990). Given posets $(P, \subseteq)$ and $(Q, \leq)$, a *anti-monotone Galois connection* (Davey & Priestley, 1990) between them consists of two *order-reversing* functions, $\phi : P \to Q$ and $\psi : Q \to P$, such that for all $X \in P$ and $Y \in Q$, we have: $X \subseteq \psi(Y) \iff Y \leq \phi(X)$. The composite functions $\psi \circ \phi : P \to P$ and $\phi \circ \psi : Q \to Q$ are both closure operators on P and Q, respectively (Davey & Priestley, 1990).

Boolean Expressions: Let $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ be a set of binary-valued attributes or *items*. Let AND, OR, and NOT denote the usual logical operators. We denote a negated item (NOT i) as $\overline{i}$. We use the symbol $|$ to denote OR, and we simply omit the AND operator whenever there is no ambiguity. For example, (i_3 AND i_4) OR (i_1 AND (NOT i_7)) is rewritten as $(i_3 i_4)|(i_1 \overline{i_7})$. A *literal* is either an item i or its negation $\overline{i}$. A *clause* is either the logical AND or logical OR of one or more literals. An AND-clause is a clause that has only the AND operator over all its literals, and an OR-clause is one that has only the ORoperator over all its literals. We assume without loss of generality that a clause has all distinct literals (since a clause is either an AND- or an OR-clause, repeated literals are logically redundant). A *boolean expression* is the logical AND or OR of one or more clauses.

A boolean expression is said to be in *negation normal form* (NNF) if all NOT operators directly precede literals (any expression can be converted to NNF by pushing all negations into the clauses). An NNF boolean expression is said to be in *conjunctive normal form* (CNF) if it is an AND of OR-clauses. An NNF expression is said to be in *disjunctive normal form* (DNF) if it is an OR of AND-clauses. For example, $(i_3 i_4)|(i_1 i_5 i_7)$ is in DNF, whereas $(i_2|i_3)(i_0|i_1|\overline{i_3})$ is in CNF. Note that by definition, single OR-clauses and single AND-clauses, are in both CNF and DNF. Furthermore, when considering negated literals, we disallow a tautology like the OR-clause containing $(i|\overline{i})$ which is always true. Similarly, we disallow a contradiction like the AND-clause containing $(i\ \overline{i})$ since this is always false. Note that a CNF expression is a tautology if and only if (iff) each one of its clauses contains both an item and its negation. Likewise, a DNF expression is a contradiction iff each one of its clauses contains both a variable and its negation. Thus by disallowing the tautologies in individual clauses, we disallow tautologies in CNF expressions. Likewise, by eliminating contradictions in clauses, we eliminate contradictions in DNF expressions. We also disallow any clause that is a superset of another clause.

$\mathcal{D}$

tid	set of items
1	ACD
2	BC
3	ABCD
4	ADE
5	E

$\mathcal{D}^T$

item	tidset
A	134
B	23
C	123
D	134
E	45

$\overline{\mathcal{D}}$

tid	set of items
1	$\overline{B}\,\overline{E}$
2	$\overline{A}\,\overline{D}\,\overline{E}$
3	$\overline{E}$
4	$\overline{B}\,\overline{C}$
5	$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$

Figure 1: Dataset $\mathcal{D}$ and its transpose $\mathcal{D}^T$ and complement $\overline{\mathcal{D}}$

Dataset: Let $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ be a set of items, let $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ be a set of transaction identifiers (tids). A dataset $\mathcal{D}$ is then a subset of $\mathcal{T} \times \mathcal{I}$; in other words, the dataset $\mathcal{D}$ is a set of tuples of the form $(t, t.X)$, where $t \in \mathcal{T}$ is the tid of the transaction containing the set of items

$t.X \subseteq \mathcal{I}$. Note that any categorical dataset can be transformed into this transactional form, by assigning a unique item for each attribute-value pair.

Given dataset $\mathcal{D}$, we denote by $\mathcal{D}^T$ the transposed dataset that consists of a set of tuples of the form $(i, i.Y)$, where $i \in \mathcal{I}$ and $i.Y \subseteq \mathcal{T}$ is the set of tids of transactions containing i. Fig. 1 shows a dataset and its transpose, which we will use as a running example in this paper. It has five items $\mathcal{I} = \{A, B, C, D, E\}$ and five tids $\mathcal{T} = \{1, 2, 3, 4, 5\}$. Note that in $\mathcal{D}$, transaction t_1 contains the set of items $\{A, C, D\}$ (for convenience, we write it as ACD), and in $\mathcal{D}^T$, the set of tids of transactions that contain item A is $\{1, 3, 4\}$ (again, for convenience we write it as 134).

Let $\aleph(t.X) = \{\bar{i} \mid i \in \mathcal{I} \setminus t.X\}$, denote the negated items that do not appear in a transaction $t.X$. For example, for $(1, ACD)$, we have $\aleph(ACD) = \overline{B}\ \overline{E}$. Define $\overline{\mathcal{D}}$ as the *complement* of database $\mathcal{D}$, given as $\overline{\mathcal{D}} = \{(t, \aleph(t.X)) \mid t.X \in \mathcal{D}\ \}$. Fig. 1 also shows the complemented dataset $\overline{\mathcal{D}}$ for our running example.

Tidset and Support: Given a transaction $(t, t.X) \in \mathcal{D}$, with $t \in \mathcal{T}$ and $t.X \subseteq \mathcal{I}$, we say that tid t *satisfies* an item/literal $i \in \mathcal{I}$ if $i \in t.X$, and t satisfies the literal $\bar{i}$ if $i \notin t.X$. For a literal l, the *truth value* of l in transaction t, denoted $V_t(l)$ is given as follows:

$$V_t(l) = \begin{cases} 1 & \text{if } t \text{ satisfies } l \\ 0 & \text{if } t \text{ does not satisfy } l \end{cases}$$

We say that a transaction $t \in \mathcal{T}$ *satisfies* a boolean expression E if the truth-value of E, denoted $V_t(E)$, evaluates to true when we replace every literal l in E with $V_t(l)$. For any boolean expression E, $\mathbf{t}(E) = \{t \in \mathcal{T} : V_t(E) = 1\}$ denotes the set of tids (also called a *tidset*), that satisfy E.

The *support* of a boolean expression E in dataset $\mathcal{D}$ is the number of transactions which satisfy E, i.e., $|\mathbf{t}(E)|$. An expression is *frequent* if its support is more than or equal to a user-specified *minimum support* (*min_sup*) value, i.e., if $|\mathbf{t}(E)| \geq \mathit{min\_sup}$. For disjunctive expressions, we also impose a *maximum support* threshold (*max_sup*) to disallow any expression with too high a support. Setting $\mathit{min\_sup} = 1$ and $\mathit{max\_sup} = \infty$ allows mining all possible expressions.

Boolean Expression Mining Tasks: Given a dataset $\mathcal{D}$ and support thresholds *min_sup* and *max_sup*, the task is to mine minimal and closed frequent boolean expressions, such as AND-clauses, OR-clauses, CNF and DNF, as specified by the user.

One of the goals of mining for different boolean expressions to find the minimal (and/or closed) boolean expressions that characterize a set of transactions (the tidset), i.e., the tids in the tidset satisfy only the given boolean expressions. In this view, both AND-clauses and OR-clauses, by definition, convey different kinds of information. For example, some tidsets can only be characterized by an AND-clause, and while others may only be characterized by an OR-clause. For some others, the simpler AND- and OR-clauses are not enough, and we have to consider CNF (or DNF) expressions to completely characterize them. As such, CNF and DNF are logically equivalent, in the sense that any CNF expression can be transformed into a DNF expression and vice-versa. However, sometimes such a transformation can lead to an exponential growth in the number of clauses. For example, the DNF expression with n clauses, $(a_1\ b_1)|(a_2\ b_2)| \cdots |(a_n\ b_n)$, yields a CNF expression with 2^n clauses. For instance, $(a_1\ b_1)|(a_2\ b_2) = (a_1|a_2)(a_1|b_2)(b_1|a_2)(b_1|b_2)$. For the sake of simplicity, we may desire to retain either the DNF or CNF expression for the analyst. Furthermore, note that for different *minsup* and *maxsup* settings, due to support based pruning, the two sets of expressions from CNF or DNF mining may be different. For these reasons, we explicitly consider both CNF and DNF expression in this paper.

Before presenting the BLOSOM framework, we will first study the structure and properties of four classes of boolean expressions; we consider each case separately – AND-clauses, OR-clauses, CNF and DNF. For simplicity of exposition, we will restrict our examples to only positive literals, but our approach is applicable to negated literals as well. In essence, we treat each negated literal as a separate item. We eliminate simple tautologies (contradictions) in OR(AND)-clauses by eliminating any clause containing both literals i and $\bar{i}$. However, since our current approach does not check for tautologies/contradictions in more complex expressions, it can find logically redundant or contradictory patterns. Integrating full tautology/contradiction checking is part of future work.

3 Related Work

Mining frequent itemsets (i.e., pure conjunctions) has been extensively studied within the context of itemset mining (Agrawal et al., 1996). The closure operator for itemsets (AND-clauses) was proposed in (Ganter & Wille, 1999), and the notion of minimal generators for itemsets was introduced in (Bastide et al., 2000). Many algorithms for mining closed itemsets (see (Goethals & Zaki, 2003)), and a few to mine minimal generators (Bastide et al., 2000; Zaki & Ramakrishnan, 2005; Dong et al., 2005) have also been proposed in the past. The work in (Dong et al., 2005) focuses on finding the succinct (or essential) minimal generators for itemsets. CHARM-L (Zaki & Ramakrishnan, 2005) finds the minimal generators for itemsets (MA), based on the differential lower shadows of closed itemsets. The task of mining closed and minimal monotone DNF expressions was proposed in (Shima et al., 2004). It gives a direct definition of the closed and minimal DNF expressions (i.e., a closed expression is one that doesn't have a superset with the same support and a minimal expression is one that doesn't have a subset with the same support). The authors further give a level-wise Apriori-style algorithm. In contrast, the novel contribution of our work is the structural characterization of the different classes of boolean expressions via the use of closure operators and minimal generators, as well as the framework for mining arbitrary expressions.

Within the association rule context, there has been previous work on mining negative rules (Savasere et al., 1998; Yuan et al., 2002; Wu et al., 2004; Antonie & Zaiane, 2004), as well as disjunctive rules (Nanavati et al., 2001). Unlike these methods we are interested in characterizing such rules within the general framework of boolean expression mining. However, as pointed out above, our current framework relies on a relatively straightforward approach to handle negated literals. Using more sophisticated methods, as suggested by some of the existing works on negative rules, can speed up the computation time. One can also use approaches that approximate the support of arbitrary boolean expressions (Calders & Goethals, 2005; Jaroszewicz & Simovici, 2002; Mannila & Toivonen, 1996) to deliver further performance gains.

Also related is the mining of optimal rules according to some constraints (Bayardo & Agrawal, 1999), since the boolean expressions can be considered as constraints on the patterns. More general notions of itemsets (including negated items and disjunctions) have been considered in the context of concise representations (Calders & Goethals, 2003; Kryszkiewicz, 2001; Kryszkiewicz, 2005). Another point of comparison is w.r.t. the work in (Gunopulos et al., 2003) where the authors aim to find frequent and (maximally) interesting sentences w.r.t. a variety of criteria. Many data mining tasks, including inferring boolean functions, are instantiations of this problem. Also of relevance is the task of mining redescriptions. The CARTwheels algorithm (Ramakrishnan et al., 2004) mines redescriptions only between length-limited boolean expressions in disjunctive normal form and CHARM-L (Zaki & Ramakrishnan, 2005) is restricted to redescriptions between conjunctions. None of these algorithms can mine redescriptions between *arbitrary* boolean expressions, as done here.

The theoretical machine learning (PAC) community has focused on learning boolean expressions (Bshouty, 1995) in the presence of membership queries and equivalence queries. Mitchell (Mitchell, 1982) proposed the concept of version spaces (which are basically a partial order over expressions) to organize the search for expressions consistent with a given set of data. However, these works conform to the classical supervised learning scenario where both positive and negative examples of the unknown function are supplied. In contrast, our work aims to find boolean expressions without explicit direction about the examples they cover. We note that a preliminary version of this paper appeared in (Zhao et al., 2006).

4 Characterizing AND- and OR-Clauses

As already noted, mining AND-clauses, has beed widely studied ever since frequent itemsets (i.e., pure conjunctions) were introduced (Agrawal et al., 1996). Mining of OR-clauses, on the other hand has not received that much attention. Note that OR and AND clauses are *duals* of each other, in the sense that $Y = (l_1 l_2 \ldots l_k)$ is an AND-clause in $\mathcal{D}$ iff $\overline{Y} = (\overline{l_1}|\overline{l_2}|\cdots|\overline{l_k})$ is an OR-clause in $\overline{\mathcal{D}}$. This duality between OR- and AND-clauses allows one to transform each problem into the other for the purposes of mining. Since the theory of minimal and closed AND-clauses is well developed, we include only a brief overview here, for the sake of completeness.

4.1 Characterizing AND-Clauses

Closed AND-clauses have been well studied in data mining as closed itemsets (Pasquier et al., 1999), as well as in the Formal Concept Analysis as concepts (Ganter & Wille, 1999). The notion of minimal generators for AND-clauses has also been previously proposed in (Bastide et al., 2000). Many algorithms for mining closed AND-clauses (eg. Charm (Zaki & Hsiao, 2005) and others (Goethals & Zaki, 2003)), and a few to mine minimal clauses (Bastide et al., 2000; Zaki & Ramakrishnan, 2005), have been proposed.

Given dataset $\mathcal{D}$, and thresholds *min_sup* and *max_sup*, the goal here is to mine AND-clauses that occur in at least *min_sup* and in at most *max_sup* transactions. Let $\mathcal{E}^\wedge$ be the set of all AND-clauses over the set of items $\mathcal{I}$. Given posets $(\mathcal{E}^\wedge, \subseteq)$ and $(2^\mathcal{T}, \subseteq)$, and $X \in \mathcal{E}^\wedge, Y \in 2^\mathcal{T}$, the following two mappings form an anti-monotone Galois connection (Ganter & Wille, 1999):

$$\phi = \mathbf{t} : \mathcal{E}^\wedge \mapsto 2^\mathcal{T}, \quad \mathbf{t}(X) = \{t \in \mathcal{T} \mid \ t \text{ satisfies } X\}$$
$$\psi = \mathbf{i} : 2^\mathcal{T} \mapsto \mathcal{E}^\wedge, \quad \mathbf{i}(Y) = \{i \in \mathcal{I} \mid \ Y \subseteq \mathbf{t}(i)\}$$

Since $(\mathbf{t}, \mathbf{i})$ forms an anti-monotone Galois connection, it follows that $\mathbb{C} = \mathbf{i} \circ \mathbf{t} : \mathcal{E}^\wedge \to \mathcal{E}^\wedge$, and $\mathbf{t} \circ \mathbf{i} : 2^\mathcal{T} \to 2^\mathcal{T}$ both form a closure operator for AND-clauses (Ganter & Wille, 1999). An AND-clause X is called *closed* if $\mathbb{C}(X) = X$. Y is called a *minimal generator* for a closed AND-clause X iff Y is a minimal subset of X such that $\mathbf{t}(Y) = \mathbf{t}(X)$. We use the notation $\mathbb{C}(\mathcal{E}^\wedge)$ and $\mathcal{M}(\mathcal{E}^\wedge)$ to denote the set of all closed and minimal generators of AND-clauses, respectively. Consider our example dataset (Fig. 1) for which Fig. 2 shows the set of all closed AND-clauses (in rectangles) and their minimal generators (in circles), as well as the groupings of the generators for the AND-clauses. The

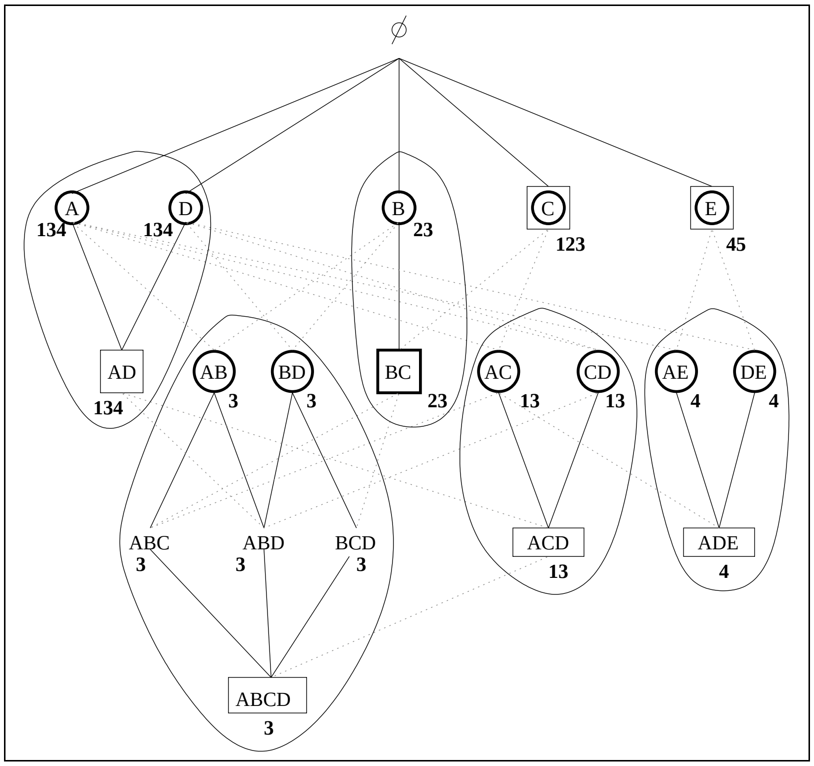


Figure 2: The Lattice of AND-Clauses ($min\_sup = 1$, $max\_sup = 5$)

tidset for each clause is also shown. As in the case of OR-clauses, it is clear that the closed and minimal AND-clauses are the most specific and most general clauses relating to a group of tidsets.

Tidset	Closed	Min Generators
3	ABCD	AB, BD
4	ADE	AE, DE
13	ACD	AC, CD
23	BC	B
45	E	E
123	C	C
134	AD	A, D

Table 1: Closed (CA) and Minimal Generators (MA) for AND-clauses

In terms of the structure of minimal generators for a closed AND-clause X, they can be considered as the minimal hitting sets of its differential lower shadow (Pfaltz & Jamison, 2001). Furthermore, closed AND-clauses have also been characterized as those resulting from finite intersections of transactions (Mielikainen, 2003). We focus instead on a direct characterization of the closed and minimal generators for AND-clauses. We define an *intersection tidset* to be a tidset obtained by finite intersections over the set of tidsets for single items, $\{\mathbf{t}(i)|i \in \mathcal{I}\}$. It is easy to see that every distinct tidset T obtained as a finite intersection of other tidsets is closed, with an associated closed AND-clause, and its minimal generators. Table 1 lists the set of all closed (CA) AND-clauses, the minimal generators (MA), and the corresponding tidsets.

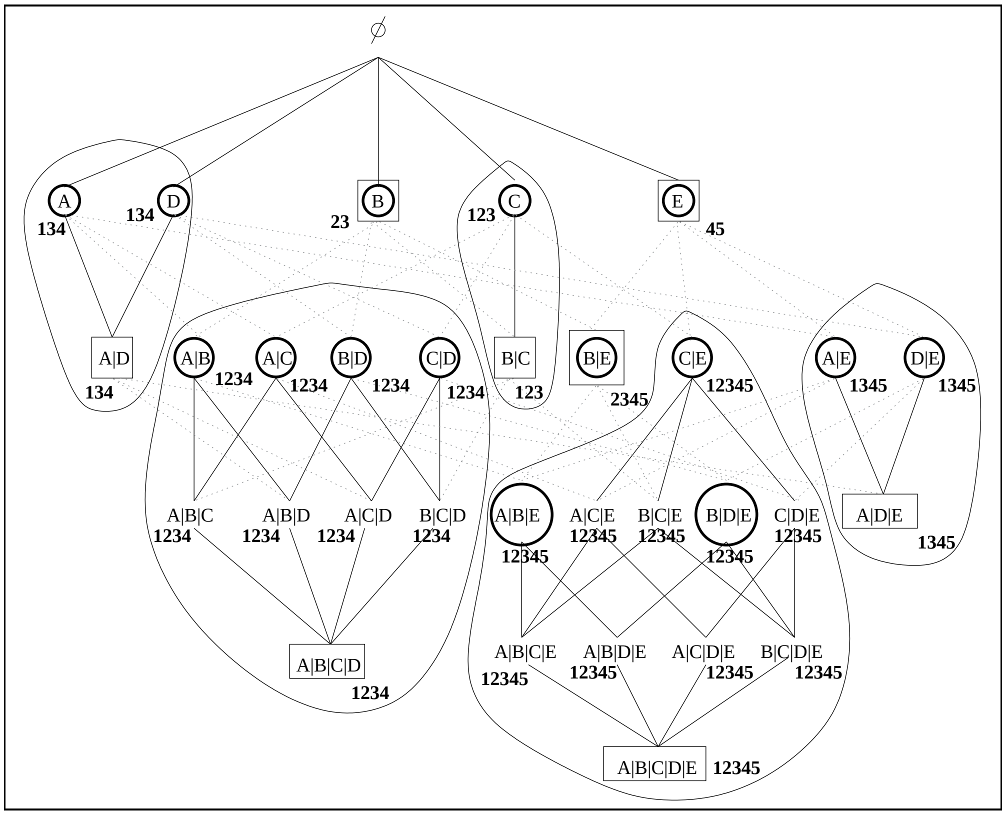


Figure 3: The Lattice of OR-clauses ($min\_sup = 1$, $max\_sup = 5$)

4.2 Characterizing OR-Clauses

Given dataset $\mathcal{D}$, and thresholds min_sup and max_sup, the goal here is to mine OR-clauses that occur in at least min_sup and in at most max_sup transactions. Let $\mathcal{E}^{\vee}$ be the set of all possible OR-clauses over the set of items $\mathcal{I}$; $\mathcal{T}$ is the set of all tids as before. For an OR-clause $X \in \mathcal{E}^{\vee}$, let $\mathcal{L}(X) = \{l | l \text{ is a literal in } X\}$ denote the set of its literals. Given $X, Y \in \mathcal{E}^{\vee}$, we define the relation $\subseteq$ between OR-clauses as follows: $X \subseteq Y$ iff $\mathcal{L}(X) \subseteq \mathcal{L}(Y)$. Then $\subseteq$ induces a partial order over $\mathcal{E}^{\vee}$. For example, $A|C \subseteq A|B|C$, since $\mathcal{L}(A|C) \subseteq \mathcal{L}(A|B|C)$. Let $X \in \mathcal{E}^{\vee}$ be an OR-clause, and let $l \in X$ be some literal in X. Then $\mathbf{t}(X) = \bigcup_{l \in X} \mathbf{t}(l)$. For example, $\mathbf{t}(A|B) = \mathbf{t}(A) \cup \mathbf{t}(B) = 134 \cup 23 = 1234$. For convenience we write the OR-clause $l_1 | l_2 | \cdots | l_k$ as $\bigvee\{l_1 l_2 \cdots l_k\}$. Also for OR-clauses, define $\mathbf{t}(\emptyset) = \emptyset$.

Let $(\mathcal{E}^{\vee}, \subseteq)$ be the partial order over OR-clauses, and let $(2^{\mathcal{T}}, \subseteq)$ be the partial order over the tidsets under the usual subset ($\subseteq$) relationship.

Theorem 1 *Given posets $(\mathcal{E}^{\vee}, \subseteq)$ and $(2^{\mathcal{T}}, \subseteq)$. Let $X \in \mathcal{E}^{\vee}$ and $Y \in 2^{\mathcal{T}}$. Then the following two mappings form a monotone Galois connection over $\mathcal{E}^{\vee}$ and $2^{\mathcal{T}}$:*

$\phi = \mathbf{t} : \mathcal{E}^{\vee} \mapsto 2^{\mathcal{T}}, \quad \mathbf{t}(X) = \{t \in \mathcal{T} \mid \ t \text{ satisfies } X\}$

$\psi = \mathbf{i} : 2^{\mathcal{T}} \mapsto \mathcal{E}^{\vee}, \quad \mathbf{i}(Y) = \bigvee\{i \in \mathcal{I} \mid \ \mathbf{t}(i) \subseteq Y\}$

PROOF: *We have to show $X \subseteq \mathbf{i}(Y) \iff \mathbf{t}(X) \subseteq Y$.*

a) First we prove $X \subseteq \mathbf{i}(Y) \Rightarrow \mathbf{t}(X) \subseteq Y$. Observe that $X \subseteq \mathbf{i}(Y) \Rightarrow \mathbf{t}(X) \subseteq \mathbf{t}(\mathbf{i}(Y))$. However, by definition of $\mathbf{i}$, $\forall i \in \mathbf{i}(Y)$, we have $\mathbf{t}(i) \subseteq Y$. This means that $\mathbf{t}(\mathbf{i}(Y)) = \bigcup_{i \in \mathbf{i}(Y)} \mathbf{t}(i) \subseteq Y$. Thus $\mathbf{t}(X) \subseteq Y$.

b) We now prove $\mathbf{t}(X) \subseteq Y \Rightarrow X \subseteq \mathbf{i}(Y)$. Assume that $X \not\subseteq \mathbf{i}(Y)$. This implies that $\exists i \in X$, such that $i \notin \mathbf{i}(Y)$. By definition of $\mathbf{i}$, this means that $\mathbf{t}(i) \not\subseteq Y$. But, this is a contradiction since $i \in X \Rightarrow \mathbf{t}(i) \subseteq Y$. Thus $X \subseteq \mathbf{i}(Y)$. ∎

Since $(\mathbf{t}, \mathbf{i})$ forms a monotone Galois connection, it follows immediately that $\mathbb{C} = \mathbf{i} \circ \mathbf{t} : \mathcal{E}^\vee \rightarrow \mathcal{E}^\vee$ is a closure operator and $\mathbb{K} = \mathbf{t} \circ \mathbf{i} : 2^{\mathcal{T}} \rightarrow 2^{\mathcal{T}}$ is a kernel operator for OR-clauses (see Sec. 2). For example, in our example dataset from Fig. 1, $\mathbb{C}(A|C) = \mathbf{i}(\mathbf{t}(A|C)) = \mathbf{i}(1234) = A|B|C|D$. Thus $A|B|C|D$ is a closed OR-clause. On the other hand $\mathbb{K}(234) = \mathbf{t}(\mathbf{i}(234)) = \mathbf{t}(B) = 23$. Thus 23 is an open tidset. It is also easy to see that the corresponding tidset for a closed OR-clause is always open. Fig. 3 shows all the closed OR-clauses (enclosed in boxes) and their corresponding open tidsets obtained from our example dataset. For example, the closed OR-clause $A|D$ has the open tidset 134. We use the notation $\mathbb{C}(\mathcal{E}^\vee)$ to denote the set of all closed OR-clauses. The set of all minimal generators of closed OR-clauses in $\mathcal{E}^\vee$ is given as $\mathcal{M}(\mathcal{E}^\vee) = \bigcup_{X \in \mathbb{C}(\mathcal{E}^\vee)} \mathcal{M}(X)$. Recall that a closed OR-clause X is the *unique* maximal OR-clause that describes a given set of objects $T = \mathbf{t}(X)$. On the other hand, a minimal generator of X is the minimal or simplest OR-clause that still describes the same object set T. Fig. 3 shows the groups of all generators of closed OR-clauses in our example dataset (from Fig. 1). Within each group the unique maximal element is the closed clause (enclosed in a rectangle). The minimal elements of each group are the minimal generators (enclosed in dark circles). One can observe that the tidsets are the same for each member of a group, but different across groups. All the minimal generators and closed OR-clauses, and their corresponding tidsets, for our example database (from Fig. 1), are summarized in Table 2.

Tidset	Closed	Min Generators
23	B	B
45	E	E
123	$B\|C$	C
134	$A\|D$	A, D
1234	$A\|B\|C\|D$	$A\|B, A\|C, B\|D, C\|D$
1345	$A\|D\|E$	$A\|E, D\|E$
2345	$B\|E$	$B\|E$
12345	$A\|B\|C\|D\|E$	$A\|B\|E, B\|D\|E, C\|E$

Table 2: Closed (CO) and Minimal (MO) OR-Clauses

Both closed OR-clauses (CO) and minimal OR-clauses (MO), along with their tidsets, individually serve as a lossless representation of the set of all possible (frequent) OR-clauses. We are particularly interested in minimal OR-clauses, since they represent the most general expressions that characterize the corresponding tidsets, and as such may be easier to comprehend. We would like to gain further insight into the structure of these minimal generators.

We give two separate characterizations of the minimal generators, one based on the closed clauses and the other a direct one. They are both based on the notion of hitting sets. Let $\mathbf{X} = \{X_1, X_2, \cdots, X_k\}$ be a set of subsets over some universe U. The set $Z \subseteq U$ is called a *hitting set* (or *transversal*) of $\mathbf{X}$ iff $Z \cap X_i \neq \emptyset$ for all $i \in [1, k]$. Let $\mathcal{H}(\mathbf{X})$ denote the set of all hitting sets of $\mathbf{X}$. Z is called a *minimal hitting set* if there does not exist another hitting set Z', such that $Z' \subset Z$. Let X be a closed OR-clause, we define the *lower-shadow* of X as the set $\mathbf{X}^\ell = \{X_i\}_{i\in[1,k]}$ where for all $i \in [1,k]$, $X_i \subset X$, X_i is closed, and there doesn't exist any other closed OR-clause Y, such that $X_i \subset Y \subset X$. In other words, the lower shadow of X is the set of closed OR-clauses that are immediate subsets of X. We further define the *differential lower-shadow* of X as the set $\mathbf{X}^\delta = \{X - X_i\}_{i\in[1,k]}$, where $X_i \in \mathbf{X}^\ell$.

Using the duality between AND- and OR-clauses, and the characterization of minimal AND-clauses (Pfaltz & Jamison, 2001), it is not hard to show that the minimal generators of a closed OR-clause X are exactly the minimal hitting sets of the differential lower shadow of X. For example, consider the closed clause $X = A|B|C|D|E$. From Fig. 3 we obtain as its lower shadow the set of closed clauses $\mathbf{X}^\ell = \{A|B|C|D,\ A|D|E,\ B|E\}$. Thus the differential lower shadow of X is given as $\mathbf{X}^\delta = \{E, B|C, A|C|D\}$. The minimal hitting sets of $\mathbf{X}^\delta$ are given as $\min_\subseteq\{\mathcal{H}(\mathbf{X}^\delta)\} = \{C|E, A|B|E, B|D|E\}$. We can see from Table 2, that the minimal hitting sets are identical to the minimal generators of $A|B|C|D|E$.

The above characterization of minimal generators relies on knowing the closed sets and their lower shadows. There is in fact a direct structural description of the closed and minimal OR-clauses. We define a *union tidset* to be a tidset obtained by finite unions over the set of tidsets for single items, $\{\mathbf{t}(i)|i \in \mathcal{I}\}$. Let $\mathcal{U}$ be the set of all distinct union tidsets. For a union tidset $T \in \mathcal{U}$, we define the *transaction set* of T, as the set $\mathcal{R}(T) = \{t.X|t \in T, (t, t.X) \in \mathcal{D}\}$, i.e., the set of transactions (i.e., itemsets) in $\mathcal{D}$ with tids $t \in T$. One can show that every distinct tidset T obtained as a union of other tidsets produces minimal generators for the closed OR-clause associated with the open tidset T. For example, let $T = 1345 = \mathbf{t}(A) \cup \mathbf{t}(E)$ in our example database in Fig. 1. Then $\mathcal{R}(T) = \{ACD, ABCD, ADE, E\}$. The hitting sets Z with $\mathbf{t}(Z) = T$ and that are minimal are given as follows $\{A|E, D|E\}$. Note that $C|E$ is a minimal hitting set of $\mathcal{R}(T)$, but $\mathbf{t}(C|E) = 12345 \neq T$, thus we reject it. We can see from Table 2 that these minimal hitting sets form the minimal generators of the closed OR-clause $A|D|E$ with tidset $T = 1345$.

5 Characterizing Normal Forms

Our approach for DNF and CNF mining builds upon the pure OR- and AND-clauses. Here we give structural characterizations for the minimal DNF and CNF expressions.

5.1 Characterizing DNF Expressions

Let $\mathcal{E}^{\text{dnf}}$ denote the set of all boolean expressions in DNF, i.e., each $X \in \mathcal{E}^{\text{dnf}}$ is an OR of AND-clauses. For convenience we denote a DNF-expression X as $X = \bigvee X_i$, where each X_i is an AND-clause. By definition $\mathcal{E}^\wedge \subseteq \mathcal{E}^{\text{dnf}}$. Also $\mathcal{E}^\vee \subseteq \mathcal{E}^{\text{dnf}}$, since an OR-clause is a DNF-expression over single literal (AND) clauses. We assume we have already computed the closed ($\mathbb{C}(\mathcal{E}^\wedge)$) and minimal ($\mathcal{M}(\mathcal{E}^\wedge)$) AND-clauses, and their corresponding tidsets.

Note that any DNF-expression $X = \bigvee X_i$ is equivalent to the DNF expression $X' = \bigvee \mathbb{C}(X_i)$, since any tidset that satisfies X_i must satisfy $\mathbb{C}(X_i)$ as well. Similarly X is also equivalent to the DNF expression $X'' = \bigvee \mathcal{M}(X_i)$, since $\mathbf{t}(X_i) = \mathbf{t}(\mathcal{M}(X_i))$. We say that $X = \bigvee X_i$ is a *min-DNF-expression* if for each AND-clause X_i there does not exist another X_j ($i \neq j$) such that $X_i \subseteq X_j$. Note that any DNF-expression can easily be made a min-DNF-expression by simply deleting the offending clauses. For example in the DNF-expression $(AD)|(ADE)$, we have $AD \subseteq ADE$; thus the expression is logically equivalent to its min-DNF form (AD). For any DNF-expression X, we use the notation $\min^{\text{dnf}}(X)$ to denote its min-DNF form. Given $X, Y \in \mathcal{E}^{\text{dnf}}$, with $X = \bigvee X_i$ and $Y = \bigvee Y_i$, we say that X is more general than Y, denoted $X \dot{\subseteq} Y$, if there exists an injective mapping f that maps each $X_i \in X$ to $f(X_i) = Y_j \in Y$, such that $X_i \subseteq Y_j$.

Tidset	Closed (maximal min-DNF)	Min Generators
34	$(ABCD)\|(ADE)$	$(AB)\|(AE)$, $(AB)\|(DE)$, $(BD)\|(AE)$, $(BD)\|(DE)$
123	$(ACD)\|(BC)$	C
134	$(ACD)\|(ADE)$	A, D
234	$(ADE)\|(BC)$	$B\|(AE)$, $B\|(DE)$
345	$(ABCD)\|E$	$(AB)\|E$, $(BD)\|E$
1234	$(ACD)\|(ADE)\|(BC)$	$A\|B, A\|C, B\|D, C\|D$
1345	$(ACD)\|E$	$A\|E, D\|E$
2345	$(BC)\|E$	$B\|E$
12345	$(ACD)\|(BC)\|E$	$A\|B\|E, B\|D\|E, C\|E$

Table 3: Additional/changed Closed (CD) and Minimal Generators (MD) for DNF

Closed DNF: We now define a closure operator for DNF expressions. First, we consider DNF expressions consisting only of closed AND-clauses. Then if we treat each $X_i \in \mathbb{C}(\mathcal{E}^\wedge)$ as a *composite item*, we can define two monotone mappings that form a monotone Galois connection as follows: Let $X = \bigvee X_i$ be a DNF expression, such that $X_i \in \mathbb{C}(\mathcal{E}^\wedge)$, and let $Y \in 2^{\mathcal{T}}$. Define $\mathbf{t}(X) = \{t \in \mathcal{T} \mid t \text{ satisfies } X\}$, and $\mathbf{i}(Y) = \bigvee\{X_i \mid X_i \in \mathbb{C}(\mathcal{E}^\wedge) \text{ and } \mathbf{t}(X_i) \subseteq Y\}$. This implies that $\mathbb{C} = \mathbf{i} \circ \mathbf{t}$ is a closure operator. For example, consider each closed AND-clause in Table 1 as an "item". Consider $X = AD|E$. $\mathbb{C}(X) = \mathbf{i}(\mathbf{t}(AD|E)) = \mathbf{i}(1345) = ABCD|ADE|ACD|E|AD$, which is a closed DNF expression. However it is logically redundant. What we want is the maximal min-DNF expression equivalent to $\mathbb{C}(X)$, which is $ACD|E$. That is, among all the min-DNF expressions (i.e., with no clause a subset of another) with tidset equal to $\mathbf{t}(X)$, the maximum one is defined as the closure.

Minimal DNF: Let $T \in 2^{\mathcal{T}}$ be a union tidset obtained as the finite union of tidsets of closed AND-clauses. As before the transaction set of T, denoted $\mathcal{R}(T)$, is the set of transactions in $\mathcal{D}$ with tid $t \in T$. We can characterize the minimal DNF expressions as the set $\min_{\subseteq}\{Z \in \mathcal{E}^{\text{dnf}} \mid Z \in \mathcal{H}(\mathcal{R}(T)) \text{ and } \nexists T' \supset T \text{ such that } Z \in \mathcal{H}(\mathcal{R}(T'))\}$, which is the set of all minimal hitting sets of $\mathcal{R}(T)$ having the tidset T. For example, consider the union tidset $T = 34$ (which is the union of $\mathbf{t}(ABCD)$ and $\mathbf{t}(ADE)$. The minimal DNF hitting sets that hit only the tidset 34 are $AB|AE, AB|DE, BD|AE, BD|DE$. In fact minimal hitting sets can be obtained directly from minimal AND-clauses.

Lemma 2 *Let T be a union tidset, and let X be the closed DNF-expression with $\mathbf{t}(X) = T$. Then $\mathcal{M}(X) = \min_{\subseteq}\{Z = \bigvee Z_i | Z_i \in \mathcal{M}(\mathcal{E}^\wedge)$ and $\mathbf{t}(Z) = T\}$.*
PROOF: *Let $G = \bigvee_i G_i \in \mathcal{M}(X)$. Then $\mathbf{t}(G) = T$. Further, each $G_i \in \mathcal{M}(\mathcal{E}^\wedge)$, since otherwise G would not be minimal. Finally minimality of G implies it is a minimal DNF expression satisfying the right hand side. For the opposite direction, let $Z = \bigvee Z_i \in \mathcal{M}(\mathcal{E}^\wedge)$, be a minimal expression with $\mathbf{t}(Z) = T$. But this means Z is a minimal generator of X.* ■

For example, for $T = 34$, we see in Table 1 that $\{AB,BD\}$ are the minimal generators with tidset 3, and $\{AE, DE\}$ have tidset 4. Taking the minimal OR expressions obtained from these two sets, we get all the minimal generators having tidset $T = 34$, namely $AB|AE, AB|DE, BD|AE, BD|DE$. Table 3 shows the closed and minimal DNF expressions, in addition to those shown in Tables 2 and 1. Some entries are repeated since the closed expressions in DNF have changed. Also the new union tidsets are marked in bold. These new tidsets represent expressions that cannot be generated using pure clauses; they require the full power of a DNF expression to characterize them.

5.2 Characterizing CNF Expressions

Let $\mathcal{E}^{\text{cnf}}$ denote the set of all boolean expressions in CNF, i.e., each $X \in \mathcal{E}^{\text{cnf}}$ is an AND of OR-clauses. By definition $\mathcal{E}^{\vee} \subseteq \mathcal{E}^{\text{cnf}}$ and $\mathcal{E}^{\wedge} \subseteq \mathcal{E}^{\text{cnf}}$. For convenience we denote a CNF-expression X as $X = \bigwedge X_i$, where each X_i is an OR-clause. We say that X is a *min-CNF-expression* if for each OR-clause X_i there does not exist another X_j $(i \neq j)$ such that $X_i \subseteq X_j$. For a CNF expression X, we use the notation $\min^{\text{cnf}}(X)$ to denote its min-CNF form. Let $\mathcal{E}^{\text{cnf}}$ denote the set of all min-CNF-expressions. Given $X, Y \in \mathcal{E}^{\text{cnf}}$, with $X = \bigwedge X_i$ and $Y = \bigwedge Y_i$, we say that X is more general than Y, denoted $X \subseteq Y$, if there exists an injective f that maps each $X_i \in X$ to $f(X_i) = Y_j \in Y$, such that $X_i \subseteq Y_j$. Analogously to DNF expressions, we can define the closed and minimal CNF expressions directly from the set of all closed ($\mathbb{C}(\mathcal{E}^{\vee})$) and minimal ($\mathcal{M}(\mathcal{E}^{\vee})$) OR-clauses, and their corresponding tidsets.

Let us treat each $X_i \in \mathbb{C}(\mathcal{E}^{\vee})$ as a *composite item*; we can then define two anti-monotone mappings that form an anti-monotone Galois connection as follows: Let $X = \bigwedge X_i$ be a CNF expression, such that $X_i \in \mathbb{C}(\mathcal{E}^{\vee})$, and let $Y \in 2^{\mathcal{T}}$. Define $\mathbf{t}(X) = \{t \in \mathcal{T} \mid t \text{ satisfies } X\}$, and $\mathbf{i}(Y) = \bigwedge\{X_i \mid X_i \in \mathbb{C}(\mathcal{E}^{\vee}) \text{ and } Y \subseteq \mathbf{t}(X)\}$. This implies that $\mathbb{C} = \mathbf{i} \circ \mathbf{t}$ is a closure operator. For example, consider each closed OR-clause in Table 2 as an "item". Consider $X = (A|D)(B|C)$. $\mathbb{C}(X) = \mathbf{i}(\mathbf{t}((A|D)(B|C))) = \mathbf{i}(13) = (B|C)(A|D)(A|B|C|D)(A|D|E)(A|B|C|D|E)$, which is closed. However it is logically redundant. What we want is the maximal min-CNF expression equivalent to $\mathbb{C}(X)$, which is $(A|D|E)(B|C)$. That is among all the min-CNF expressions with tidset equal to $\mathbf{t}(X)$ the maximum one is defined as the closure.

Lemma 3 *Let $T \in 2^{\mathcal{T}}$ be an intersection tidset obtained as the finite intersection of tidsets of closed* OR*-clauses. Let X be the closed CNF-expression with $\mathbf{t}(X) = T$. Then $\mathcal{M}(X) = \min_{\subseteq}\{Z = \bigwedge Z_i | Z_i \in \mathcal{M}(\mathcal{E}^{\vee})$ and $\mathbf{t}(Z) = T\}$.*

PROOF: *Let $G = \bigwedge_i G_i \in \mathcal{M}(X)$. Then $\mathbf{t}(G) = T$. Further, each $G_i \in \mathcal{M}(\mathcal{E}^{\vee})$, since otherwise G would not be minimal. Finally minimality of G implies it is a minimal CNF expression satisfying the right hand side. In the other direction, let $Z = \bigwedge Z_i \in \mathcal{M}(\mathcal{E}^{\vee})$, be a minimal expression with $\mathbf{t}(Z) = T$. But this means Z is a minimal generator of X.* ■

For example, let $T = 13$. We can obtain 13 as the intersection of several minimal OR-clauses' tidsets, e.g., the minimal OR-clauses C and $\{A|E, D|E\}$. However the minimal among all of these are C and $\{A, D\}$, giving the two minimal CNF expressions: AC and CD. Table 4 shows the closed CNF expressions and their minimal generators in addition to those already shown in Tables 2 and 1, or those that have changed. The bold tidsets represent expressions that cannot be generated using pure clauses; they require the full power of a CNF expression to characterize them.

Tidset	Closed (maximal min-CNF)	Min Generators
3	$B(A\|D)$	AB, BD
13	$(B\|C)(A\|D\|E)$	AC, CD
34	$(A\|D)(B\|E)$	$A(B\|E), D(B\|E)$
234	$(A\|B\|C\|D)(B\|E)$	$(A\|B)(B\|E)$, $(A\|C)(B\|E)$, $(B\|D)(B\|E)$, $(C\|D)(B\|E)$
345	$(A\|D\|E)(B\|E)$	$(A\|E)(B\|E)$, $(D\|E)(B\|E)$

Table 4: Additional/changed Closed (CC) and Minimal Generators (MC) for CNF

6 The BLOSOM Framework

The BLOSOM framework for mining arbitrary boolean expressions supports several different algorithms, as listed in Table 5. Our main focus in on efficiently mining the minimal boolean expressions due to their inherent simplicity. We do propose algorithms for mining closed clauses, which can easily be extended to mine closed normal forms.

Algorithm	Mining Task
BLOSOM-MO	Minimal OR-clauses
BLOSOM-MA	Minimal AND-clauses
BLOSOM-MD	Minimal DNF expressions
BLOSOM-MC	Minimal CNF expressions
BLOSOM-CO	Closed OR-clauses
BLOSOM-CA	Closed AND-clauses
BLOSOM-CD	Closed DNF expressions
BLOSOM-CC	Closed CNF expressions

Table 5: Algorithms in the BLOSOM Framework

BLOSOM assumes that the input dataset is $\mathcal{D}$, and it then transforms it to work with the transposed dataset $\mathcal{D}^T$. Starting with the single items (literals) and their tidsets, BLOSOM performs a depth-first search (DFS) extending an existing expression by one more "item". BLOSOM employs a number of effective pruning techniques for searching over the space of boolean expressions, yielding orders of magnitude in speedup. These include: dynamic sibling reordering, parent-child pruning, sibling merging, threshold pruning, and fast subsumption checking. Further BLOSOM utilizes a novel *extraset* data structure for fast frequency computations, and to identify the corresponding transaction set for a given arbitrary boolean expression.

6.1 BLOSOM-MO: Minimal OR-Clauses

BLOSOM-MO mines all the minimal OR-generators, and is broadly based on CHARM (Zaki & Hsiao, 2005). However, CHARM mines only the closed AND-clauses, whereas BLOSOM-MO mines the minimal OR-clauses. BLOSOM-MO takes as input the set of parameter values *min_sup*, *max_sup*, *max_item* and a dataset $\mathcal{D}$ (we implicitly convert it to $\mathcal{D}^T$). The *max_item* constraint is used to limit the maximum size of any boolean expression, if desired. BLOSOM-MO conceptually utilizes a DFS tree to search over the OR-clauses. Each OR-clause is stored as a set of items (the OR is implicitly assumed). Thus each node of the search tree is a pair of $(T \times I)$, where I is an item set denoting an OR-clause and T is a tidset (as shown in Fig. 5). In the description below, we use MO to denote a minimal generator of OR-clauses. Before describing the pseudo-code we briefly describe each of the optimizations used in the BLOSOM-MO.

6.1.1 Pruning and Optimization Techniques

Threshold Pruning: BLOSOM uses the three thresholds to prune the generators, based on min_sup, max_sup and max_item. Furthermore, if the item set I of the current node equals the set of all items $\mathcal{I}$, then we stop its expansion immediately, since any of its descendants will be pruned.
Dynamic Sibling Reordering: Before expanding a group of sibling nodes in the search tree, BLOSOM-MO dynamically reorders them by their tidset size dynamically. Since smaller tidsets are more likely to be contained in longer previous tidsets, this can prune out many branches.
Relationship Pruning: BLOSOM-MO makes use of two kinds of relationship pruning: parent-child and sibling based. During the DFS expansion, if the tidset of a node is the same as any of its parents', the node and all of its descendants are pruned (based on the definition of minimal generators). If some sibling nodes of the same parent node have the same tidset, they are merged together by unioning their itemsets into a "composite node". A prefix item set data structure is utilized to deal with sibling merging. All sibling nodes of the same parent share a common prefix, containing the item sets on the path from the root to the current node. The prefix also saves memory, since all the siblings do not need to keep their separate prefix copies.
Fast Subsumption Checking: BLOSOM-MO maintains a hash table for storing the current MO; the hash key of the MO is the *tidsum* (summation of tids in T) of its transaction set T. An element of the hash table is a pair $(T \times MS)$, where T is a tidset and MS is the set of T's MOs. Subsumption checking on MOs guarantees that the current generators are minimal, i.e. for some transaction set T, any two of its generators do not contain each other. Before adding a new generator G we remove any of its supersets in MS. Due to the nature of the enumeration process, it is not necessary to check if G is minimal, i.e., G cannot be a superset of any MO of entry T, since otherwise G would have been pruned previously by one of its ancestor nodes using the parent-child pruning. So we only need to do the subsumption checking in one direction.
Extraset Technique: BLOSOM-MO utilizes a novel *extraset* technique to save set operation time and memory usage for tidsets. The DFS expansion involves a lot of tidset unions, and each node has to keep an intermediate set, which is a superset of the parent node's tidset. So for each node, inspired by the diffset (Zaki & Hsiao, 2005) technique, we simply retain the extra-part of its parent's tidset in a data structure we call *extraset*, which can save a lot of memory while storing the intermediate tidsets. For example, in Fig. 1, we have $\mathbf{t}(A) = 134$, and $\mathbf{t}(A|B) = 1234$. The extraset of $A|B$ is given as $\mathbf{t}(A|B) - \mathbf{t}(A) = 2$. In effect an *extraset* keeps track of the additional items added while performing unions of tidsets; as such it is the opposite of a diffset (Zaki & Hsiao, 2005), which keeps tracks of the items that drop out when performing intersections of tidsets, while mining AND-clauses. In addition, the set union operations on the original tidset are changed to the operations on the extraset, which also saves a lot of time, due to the small size of extrasets.
No-Transaction-Set Optimization: Sometimes one needs only the frequency of a MO, instead of the entire tidset. If tidsets are not required, we can avoid keeping large tidsets for each generator and the corresponding costly comparisons. The additional overhead is to separate those generators that share the same hash entry (if they share the same tidsum, but no the same tidset); however, since most entries have one or only a few generators for most datasets, this overhead is minimal.

```
**Input**          : $min\_sup$, $max\_sup$, $max\_item$ and $\mathcal{D}^T$
**Output**         : hash table $\mathcal{M}$ containing all MO of $\mathcal{D}^T$
**Initialization**: $NL = \{\mathbf{t}(i) \times \{i\} | i \in \mathcal{I}\}$, call BLOSOM-MO($\emptyset$, $NL$, 0, 0)

BLOSOM-MO($\mathcal{P}$, $NL$, $sup$, $sum$):
1    sort $NL = \{n_i.T \times n_i\}$ in decreasing order of $|n_i.T|$
2    while $\exists\ n_i$ and $n_j \in NL$ such that $n_i.T = n_j.T$ do
3        $n_i.I \leftarrow n_i.I \cup n_j.I$ /* sibling merging */
4        $NL \leftarrow NL - n_j$
5    foreach $n_i \in NL$ do
6        $\mathcal{P}' \leftarrow \mathcal{P} + n_i.I$
7        $sup' = sup + |n_i.T|$
8        $sum' = sum + summation(n_i.T)$
9        If $sup' > max\_sup$ then skip to next $n_i$ /* goto line 6 */
10       if $sup' \geq min\_sup$ then
11           foreach combination $I'$ of $\mathcal{P}'$ do
12               delete all supersets of $I'$ in $\mathcal{M}[sup' \times sum']$
13               $\mathcal{M}[sup' \times sum'] \leftarrow \mathcal{M}[sup' \times sum'] + I'$
14       If $|\mathcal{P}'| \geq max\_item$ then skip to next $n_i$ /* goto line 6 */
15       $NL' \leftarrow \emptyset$
16       foreach $n_j \in NL$ ranking after $n_i$ do
17           $n'.T \leftarrow n_j.T - n_i.T$ /* get extraset */
18           if $|n'.T| \neq \emptyset$ then
19               $n'.I \leftarrow n_j.I$
20               $NL' \leftarrow NL' + n'$
21       if $NL' \neq \emptyset$ then
22           BLOSOM-MO( $\mathcal{P}'$, $NL'$, $sup'$, $sum'$ )
```

Figure 4: BLOSOM-MO Algorithm

6.1.2 Algorithmic Description

We now describe the BLOSOM-MO algorithm in detail, based on the pseudo-code in Fig. 4. It is a recursive algorithm that, at each call accepts a current prefix queue $\mathcal{P}$ containing the I sets on the path from the root to the current node, a node list NL containing a group of sibling nodes that share the same parent, and two parameters of the current tidset T: support ($|T|$) and summation ($\sum_{t \in T} t$). We use these instead of the original tidset T to save memory and corresponding copying cost. The initial call is made with an empty $\mathcal{P}$, the NL containing all the single items, and the support and summation are set to 0. Line 1 sorts the current sibling nodes in NL in decreasing order of support, which speeds up the convergence of the algorithm (*dynamic sibling reordering*). Lines 2-4 merge the sibling nodes with the same tidset T by unioning their item sets together (*sibling merging*). Lines 7-20 form the main loop to process each of the sibling nodes one by one. Line 6 updates the current prefix queue for further recursive calls. Lines 7-8 generate the transaction set support sup' and summation sum' by adding tids from the *extraset* $n_i.T$. Line 9 checks the *max_sup* threshold. Lines 10-13 try to add the current sibling node satisfying *min_sup* and *max_sup* to the hash table $\mathcal{M}$. If the node satisfies *max_sup* (line 9) and *min_sup* (line 10), then the algorithm will try to add every possible queue $\mathcal{P}'$ item combination to the current MO hash table (lines 11-13). Each combination is generated by picking one and only one item from each $P \in \mathcal{P}'$. At the same time, we also need to do subsumption checking and to delete any supersets of the new MO I' to be added (line 12). The new MO I' must be minimal, i.e., it should not be subsumed by any current MO in the hash table (*fast subsumption checking*). Line 14 checks the *max_item* threshold. Lines 15-22 produce node n_i's valid children nodes NL' generated with other sibling nodes ranked after n_i (in sorted order). BLOSOM-MO tries every node pair (line 16) and generates children nodes (line 17). It only saves the *extraset* ($n_i.T \cup n_j.T - n_i.T = n_j.T - n_i.T$) to save memory and set operation time, which proves very efficient for dense datasets. If a child node is not subsumed by its parents (line 18), then it is added to the valid children node list in lines 19-20 (*parent-child relationship pruning*). Lines 21-22 make a recursive call with node n_i's valid children. After the steps showed in Fig. 4, BLOSOM-MO needs to do some further work to separate those generators that have the same support and tidsum but actually have different tidsets. Since most entries of hash table $\mathcal{M}$ have only one or a few generators, the separating stage is not too costly.

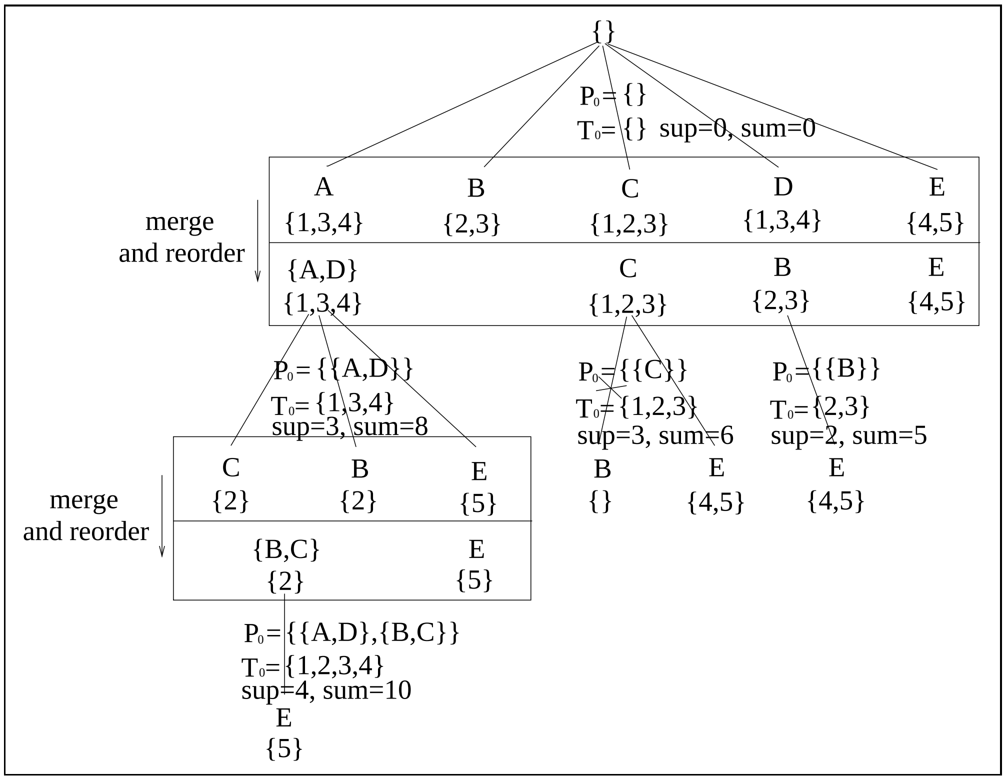


Figure 5: OR-clause Mining Example: DFS Search Tree

An Example: Consider how BLOSOM-MO works on our example dataset $\mathcal{D}^T$ (from Fig. 1). Fig. 5 shows the DFS search tree. Initially the prefix queue $\mathcal{P}$ is empty, and node list NL contains five single items. After sibling merging and reordering, they become $AD\times134$, $C\times123$, $B\times23$ and

$E{\times}45$ as shown in Fig. 5. From $\mathcal{P}' = \{A, D\}$, we get two item combinations: A and D. We thus add A and D to the MOs hash table $\mathcal{M}$ with the entry $T' = \emptyset \cup 134 = 134$, $sup = 3$, and $tidsum = 1 + 3 + 4 = 8$, as shown in Table 6. In the table the subscript numbers represent the order in which MOs are added to the table. Then we expand node $\{A, D\}{\times}134$ by combining with node $C{\times}123$, $B{\times}23$ and $E{\times}45$, and save the *extraset* of the tidset union for each combination pair as follows. Since $X{\cup}Y{-}X{=}Y{-}X$, the extrasets for the three new candidates (children of the node $\{A, D\} \times 134$) are 123−134=2, 23−134=2, and 45−134=5, respectively. After reordering and merging, $\mathcal{NL}' = \{\{B, C\}{\times}2,\ E{\times}5\}$. Along with $\mathcal{P}{=}\{\{A, D\}\}$ and $T'{=}134$, BLOSOM is called recursively. Next step from $\mathcal{P}'{=}\{\{A, D\}, \{B, C\}\}$, we get four combinations: $\{A, B\}$, $\{A, C\}$, $\{B, D\}$ and $\{C, D\}$, and we add them to the hash table with entry: $T'{=}134 \cup 2{=}1234$, $sup = 4$, $tidsum = 10$. Readers can continue this process until all the MOs are generated as shown in Table 6 in subscript order. Note that node $B{\times}\emptyset$ under the path $Root{\to}C{\to}B$ is pruned using parent-child relationship because of its empty *extraset* (i.e. generator BC is not minimal and generates the same tidset as one of its parents). In addition, when generator CE is added to entry $T = 12345$ of $\mathcal{M}$, previously added MOs ACE and CDE will be deleted (as marked by underlines) since they are supersets of generator CE. Note that in Table 6, the tidsets T are actually *not kept* during the mining process, if only support is desired. In this case we have to verify that the generators in each cell ($sup{\times}sum$) of the hash table do belong to the same entry. In Table 6, this check must be done for cells (4×10), (5×15) and (4×13) to assure they belong to the same entry. For our example, all entries are correct. Otherwise, we would need to separate them into different entries.

T	sup	sum	Minimal OR Generators
134	3	8	$\{A\}_1, \{D\}_1$
1234	4	10	$\{AB\}_2, \{AC\}_2, \{BD\}_2, \{CD\}_2$
12345	5	15	$\{ABE\}_3, \underline{\{ACE\}_3}, \{BDE\}_3, \underline{\{CDE\}_3}, \{CE\}_6$
1345	4	13	$\{AE\}_4, \{DE\}_4$
123	3	6	$\{C\}_5$
23	2	5	$\{B\}_7$
2345	4	14	$\{BE\}_8$
45	2	9	$\{E\}_9$

Table 6: Hash Table $\mathcal{M}$ for MOs

6.2 BLOSOM-CO: Closed OR-Clauses

Whereas BLOSOM-MO is designed for mining minimal generators, BLOSOM-CO is specific to mining closed expressions. The main difference is that instead of finding the *minimal* elements, we have to find the *maximal* elements corresponding to the given tidsets. Thus the logic of subsumption checking, as well as relationship pruning (both parent-child and sibling), has to be reversed. There are two additional pruning optimizations utilized in BLOSOM-CO, which are: a) *Sibling Containment Pruning:* After sibling merging, another relationship among siblings can be utilized for CO generation, i.e., containment relationship. When expanding a node N, any sibling node ranking after N whose tidset is a subset of that of N, can be pruned in the next level of N's expansion tree, and its item set is merged with the item set of the current node. b) *Hash Pruning:* Assume $\mathcal{M}$ is the hash table of COs. If we find the item set of the current tree node ($T \times I$) is subsumed by the entry of $\mathcal{M}$, i.e. $I \subset \mathcal{M}(T)$, then all the descendant nodes of the current node will be pruned, since their item sets will be subsumed by that of the descendant nodes of $T \times \mathcal{M}(T)$.

6.3 Mining AND-Clauses

To mine the minimal and closed AND-clauses, we build upon BLOSOM-MO and BLOSOM-CO, respectively. Note that by DeMorgan's law, to mine the minimal AND-clauses, we can mine the minimal OR-clauses over the complemented tidsets. For example in our example dataset in Fig. 1, with $\mathcal{T} = 12345$, we have $\mathbf{t}(AB) = \mathbf{t}(A) \cap \mathbf{t}(B) = 134 \cap 23 = 3$. We can obtain the same results if we mine for $\overline{A}|\overline{B}$ in the complemented database. For example $\overline{\mathbf{t}(\overline{A})|\mathbf{t}(\overline{B})} = \overline{\mathbf{t}(\overline{A}) \cup \mathbf{t}(\overline{B})} = \overline{25 \cup 145} = \overline{1245} = 3$. Thus to mine MA, we mine MO over complemented tidsets. Likewise, to mine CA, we mine CO over the complemented tidsets. Note that if $\mathcal{D}$ is sparse and large it is better to mine it directly instead of the complement database $\overline{\mathcal{D}}$, which will be large and dense. In such cases it is easy to modify BLOSOM-MO to perform intersections of tidsets instead of unions.

6.4 Minimal DNF and CNF Expressions

```
   Input     : $min\_sup \geq min\_sup'$, $max\_sup$, $max\_item$ and dataset $\mathcal{D}^T$
   Output    : hash table $\mathcal{M}_{MD}$ containing all MD of $\mathcal{D}^T$

   BLOSOM-MD($min\_sup$, $minsup'$, $max\_sup$, $max\_item$, $\mathcal{D}^T$):
1    $\mathcal{M}_{MA} \leftarrow$ BLOSOM-MA($min\_sup'$, $max\_sup$, $max\_item$, $\mathcal{D}^T$)
2    foreach $G \in \mathcal{M}_{MA}$ do
3        re-generate $\mathbf{t}(G)$
4    $\mathcal{D}^T_{new} \leftarrow$ CreateNewDB($\{G \times \mathbf{t}(G) | G \in \mathcal{M}_{MA}\}$)
5    $\mathcal{M}'_{MD} \leftarrow$ BLOSOM-MO($min\_sup$, $max\_sup$, $max\_item$, $\mathcal{D}^T_{new}$)
6    foreach $G' \in \mathcal{M}'_{MD}$ do
7        $G \leftarrow$ min-DNF($G'$)
8        $\mathcal{M}_{MD} \leftarrow \mathcal{M}_{MD} + G$
```

Figure 6: The BLOSOM-MD Algorithm

Following the structural characterization of minimal DNF expressions outlined in Sec. 5.1, BLOSOM-MD follows a two-phase approach. It first extracts all the minimal AND-clauses and then finds the minimal DNF expressions using those. Fig. 6 shows the pseudo-code of BLOSOM-MD. We use BLOSOM-MA to get all minimal AND-clause generators ($\mathcal{M}_{MA}$) using the original dataset (line 1). Then, we regenerate the tidsets for each entry in $\mathcal{M}_{MA}$ (lines 2-3). Using the minimal AND-clause generators along with their tidsets, we create a new dataset $\mathcal{D}^T_{new}$ (line 4). The CreateNewDB method assigns a new label to each generator $G \in \mathcal{M}_{MA}$ and uses the tidsets $\mathbf{t}(G)$ as the new dataset. Note that, as an optimization, only one label is used per distinct tidset, even though there may be multiple corresponding minimal generators. Next, we call BLOSOM-MO to get all OR-clause generators over the new labels using the new dataset $\mathcal{D}^T_{new}$ (line 5), which in fact represent the initial set of minimal DNF generators. Note that in this step we implicitly replace each label by the original set of minimal generators. Finally, in lines 6-8, we delete produce the final set of min-DNF generators $\mathcal{M}_{MD}$.

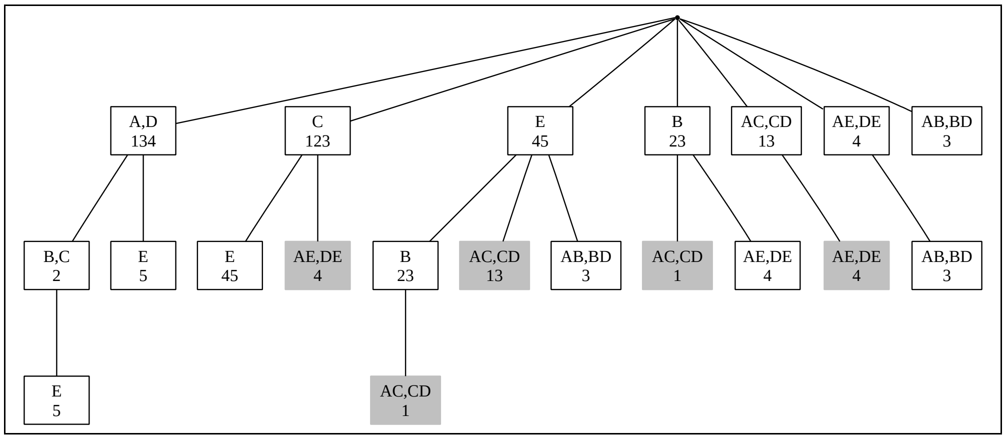


Figure 7: DNF Mining Example: DNF expressions are added to $\mathcal{M}'_{MD}$ in DFS manner. Shaded boxes represent expressions that are not minimal, and are thus pruned. The extrasets are shown under each expression.

Another point that needs further elaboration is the effect *min_sup* when mining DNF expressions. If we use the *min_sup* constraint while mining AND-clauses, then it is possible that we will miss some frequent DNF expressions, which have support more than *min_sup*, after considering the tidset unions. For example, with $min\_sup = 3$ in our running example, only the minimal AND-generators $C \times 123$ and $\{A, D\} \times 134$ remain frequent from Table 1. It is clear that in this case, we will fail to generate minimal DNF generators like $(AB|E, BD|E) \times 345$, $(B|AE, B|DE) \times 234$, and so on. To guarantee completeness, one has to set $min\_sup = 1$ to mine all possible DNF expressions. Nevertheless, from a practical view-point this may generate to many AND-clauses to mine the DNF expressions. One may adopt a compromise solution, whereby we impose a new lower minimum support constraint $min\_sup' < min\_sup$ for the AND-clauses, and use the *min_sup* constraint for the DNF expressions. This is not unreasonable, since it imposes the constraint that any individual clause in the DNF expression is above min_sup'. Note that *max_sup* constraint does not impact AND-clause mining (in line 1), and really applies only for the DNF expression mining (in line 5).

An Example: Fig. 7 shows the steps involved in mining DNF expressions, using the example dataset in Fig. 1. The figure shows as starting point the dataset $\mathcal{D}^T_{new}$ obtained after applying BLOSOM-MA on $\mathcal{D}$ and regenerating the tidsets (i.e., after steps 1-4 in Fig. 6). Mining the AND-clauses yields the seven unique tidsets, and the corresponding minimal generators shown in Table 1. After sorting the tidsets in decreasing order of cardinality, these AND-clauses form the initial level in Fig. 7 (note that for simplicity, original minimal generator groups are shown, as opposed to using new labels to represent them). Applying BLOSOM-MO on the new dataset, yields the full DFS tree, as shown in Figure 7. As an example, consider the extensions under node A, D. New candidates are created by performing tidset unions will all other siblings. However, if for a node X, its tidset is a subset of $\mathbf{t}(A, D)$, then X is discarded, since it will yield a non-minimal generator. Thus the only valid children under A, D are C (with extraset $123 - 134 = 2$), E (with extraset $45 - 134 = 4$), and B (with extraset $23 - 134 = 2$). Due to the sibling merging technique (see Sec. 6.1.1), C and B are collapsed into a single node B, C (with extraset 2). Note that the complete DNF expression at each node can be obtained by adding the OR operator to the sets on the path from the root to that node. Likewise the complete tidset can be obtained by taking the union of the extrasets on the path from the root to that node. For example, the child B, C under A, D, represents the composite DNF expression $\{A, D\}|\{B, C\}$, which when expanded, yields the four minimal DNF generators $A|B$, $A|C$, $B|D$, $C|D$ characterizing the tidset 1234. Note that as each new composite expression is produced it is added the hash-table $\mathcal{M}'_{MD}$, and fast-subsumption checking is used to prune non-minimal branches. For example, consider the child node AE, DE under C, representing the generators $C|\{AE, DE\}$ with tidset 1234. When we try to add this to the hash-table, we find that there is already an entry in the cell with tidset 1234, inserted earlier (in DFS order) for the generators $\{A, D\}|\{B, C\}$. Since $C|\{A, D\} \subseteq C|\{AE, DE\}$, the new node does not represent a minimal generator and is pruned. Note that even after such pruning we need to finally generate min-DNF expressions, once we expand the composite generators. For example, for the tidset 12345, we have the following entries in the hash table: $\{A, D\}|\{B, C\}|E$, and $C|E$. When we expand the first composite entry, we obtain the generators $A|B|E$, $A|C|E$, $D|B|E$, $D|C|E$. However, $A|C|E$ and $D|C|E$ are not minimal, due to $C|E$. Thus, the final min-DNF generators for 12345 are $A|B|E$, and $B|D|E$, and $C|E$.

6.5 Mining Closed DNF, and CNF Expressions

For mining the minimal CNF expressions, the roles of BLOSOM-MA and BLOSOM-MO are reversed. BLOSOM-MC starts by mining the minimal OR-clauses and then computes the minimal AND-clauses by treating each of them as a new item. Finally any subsumed generators are purged to obtain the min-CNF forms.

Finally, consider the approach for mining the closed DNF and CNF expressions. For mining closed DNF expressions, we follow the same pseudo-code as for BLOSOM-MD, replacing BLOSOM-MA and BLOSOM-MO with BLOSOM-CA and BLOSOM-CO, respectively. That is, we assume we know the closed AND-clauses, and then treating each of these as a new items, we mine the closed OR-clauses. Finally, convert each such expression in the maximal min-DNF form. Note that the roles of *min_sup* and *max_sup* are also reversed in CNF mining. That is while mining OR-clauses, we use $max\_sup' \geq max\_sup$ to extract the base OR-generators, and use the original *max_sup* to mine the CNF generators. For closed CNF expressions, reverse the roles of BLOSOM-CO and BLOSOM-CA, and output the maximal min-CNF expressions.

7 Experiments

All experiments were done on a Ubuntu virtual machine (over WindowsXP & VMware) with 448MB memory, and a 1.4GHz Pentium-M processor. We used both synthetic and real datasets to evaluate BLOSOM. The synthetic datasets are generated with three parameters: the number of items $|\mathcal{I}|$, the number of transactions $|\mathcal{T}|$ and dataset density, δ. The size of the dataset is $|\mathcal{I}| \times |\mathcal{T}|$. For each item i, the average size of its transaction set $\mathbf{t}(i)$, is given as $\delta \times |\mathcal{T}|$. The average size is distributed uniformly in the interval $[0, |\mathcal{T}|]$, and the tids are distributed in $\mathbf{t}(i)$ with equal probability.

7.1 Performance Study

Effect of Optimizations: We first study the effect of various optimizations proposed in Sec. 6.1.1 on the performance of BLOSOM-MO, as shown in Fig. 8. The x-axis shows the number of items

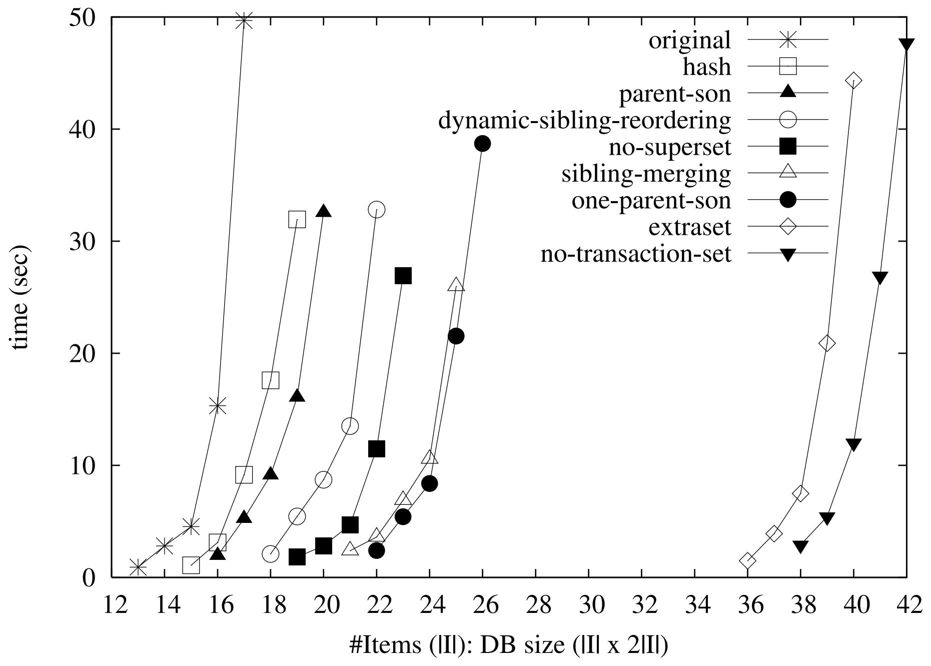


Figure 8: The effects of the speedup optimizations

$|\mathcal{I}|$ in the synthetic datasets. The number of transactions were generated as $|\mathcal{T}| = 2|\mathcal{I}|$. Each curve in the figure shows the running time after applying the optimizations specified in succession. Thus the final curve for `no-transaction-set` includes all previous optimizations. In the legends, `original` stands for the unoptimized version, `hash` means using a hash table for subsumption, `parent-son` means doing parent-child pruning, `dynamic-reordering` is the reordering of nodes, `no-superset` stands for one directional subsumption checking, `sibling-merging` is self explanatory, `one-parent-son` means we check only the left parent in parent-child relationship pruning due to the application of dynamic-sorting, `extrasets` is obvious, and finally `no-transaction-set` means we avoid storing the entire tidset. We can see that the cumulative effect of the optimizations is substantial; BLOSOM-MO can process a dataset around 10 times $((38 \times 76)/(12 \times 24) = 10)$ larger than the base algorithm can in the same running time. Thus all the optimizations together deliver a speedup of over an order of magnitude compared to the base version.

Parameters	MO/MA	MC/MD
$\|\mathcal{I}\|$	50	15
$\|\mathcal{T}\|$	300	30
δ	0.5	0.5
min_sup	1	1
max_sup	$\|\mathcal{T}\|$	$\|\mathcal{T}\|$
max_item	4	3

Table 7: Common experimental parameters

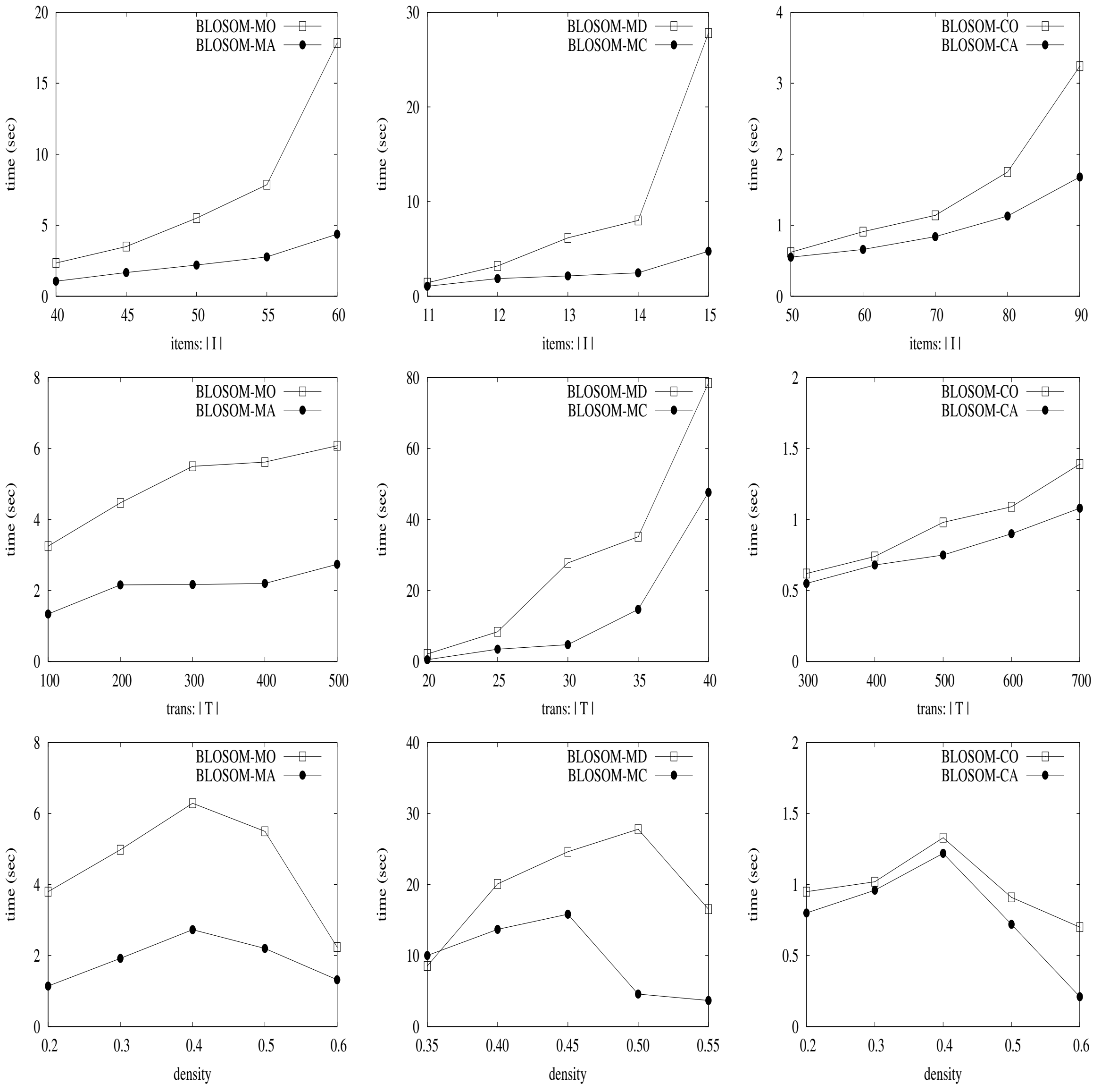


Figure 9: Synthetic Data: Effect of number of items, number of transactions, and density

Effect of Parameters: The base-line parameters for the synthetic datasets are shown in Table 7. Note that we set $min\_sup = 1$ and $max\_sup = |\mathcal{T}|$, which means the entire set of all possible expressions will be mined. In the experiments we vary one parameter at a time, and study the effect. Fig. 9 shows how the the mining time varies with $|\mathcal{I}|$ (top row), with $|\mathcal{T}|$ (middle row), and with density δ (bottom row). The figure shows the effect for minimal clauses (MO/MA; left col.), minimal CNF/CNF expressions (MC/MD; middle col.), and closed clauses (CO/CA; right col.). From these graphs we observe several trends. Notice that as we increase number of items/transactions and density, the disjunction times (for CO, MO, MD) tend to be higher than the conjunction times (CA, MA, MC). This is mainly because "unions" of tidsets tend to produce more distinct tidsets than "intersections", resulting in a larger search space. The trend for increasing $|\mathcal{I}|$ and $|\mathcal{T}|$ are as expected; the more the number of items or transactions, the longer the running time. However, with increasing density, the running time reaches the peak and then comes down. This is mainly because when density is closer to 50% we tend to mine patterns in the middle of the lattice, resulting in larger search spaces.

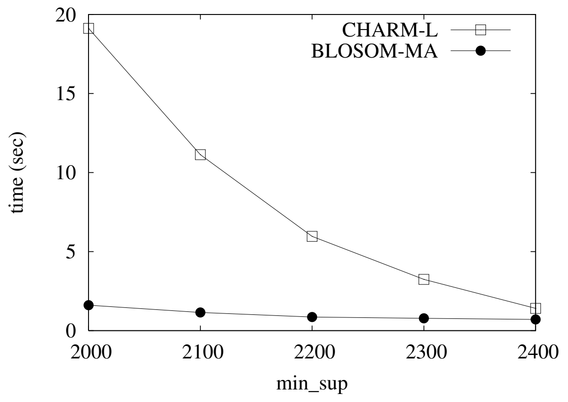
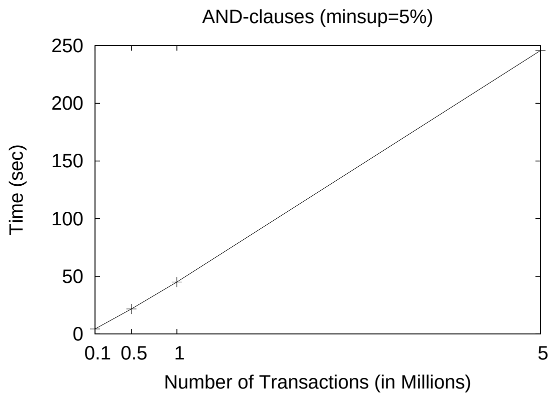


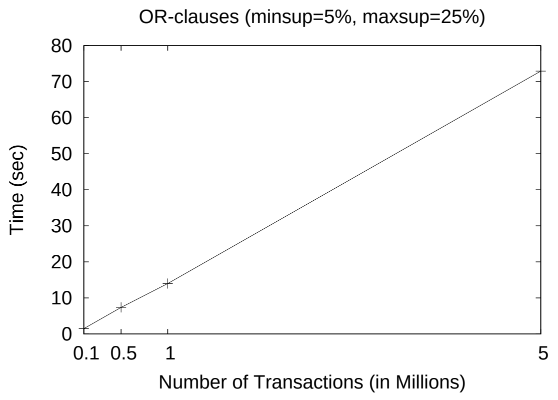
Figure 10: BLOSOM-MA vs. CHARM-L

Comparison with CHARM-L: We compared BLOSOM-MA with CHARM-L (Zaki & Ramakrishnan, 2005), which can also mine the minimal generators for AND-clauses (i.e., itemsets). We used the `chess` dataset, from the UCI machine learning repository, which has 3196 rows and 75 items. From Fig. 10 we can see that BLOSOM-MA can be about ten times faster than CHARM-L, and the gap is increasing with decreasing support. This is mainly because CHARM-L first finds all closed expressions and then uses their differential lower shadows to compute the minimal AND-clauses. In contrast, BLOSOM-MA directly mines the minimal generators, and uses effective optimizations to speed up the search. We do not compare with the approach in (Bastide et al., 2000), since it is Apriori-based and not likely to be as effective as BLOSOM-MA. We know of no other algorithms to mine minimal OR-clauses, and closed/minimal CNF and DNF expressions. However, note that is should be possible to modify several of the newly proposed closed itemset mining algorithms in the FIMI (Goethals & Zaki, 2003) repository to give effective algorithms for minimal clauses. Our work is the first step in this direction.

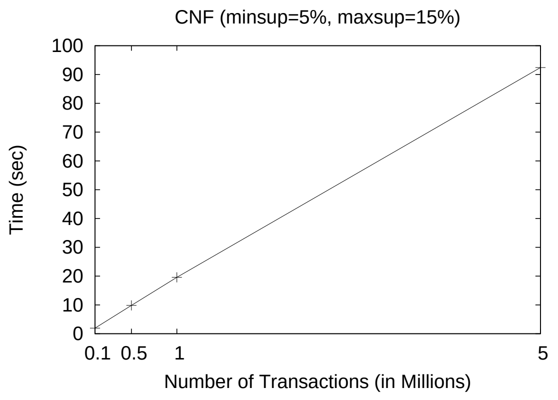
Scalability: Finally, to test the scalability of our framework with respect to the number of transactions, we show used the IBM synthetic dataset generator (Agrawal et al., 1996), to generate



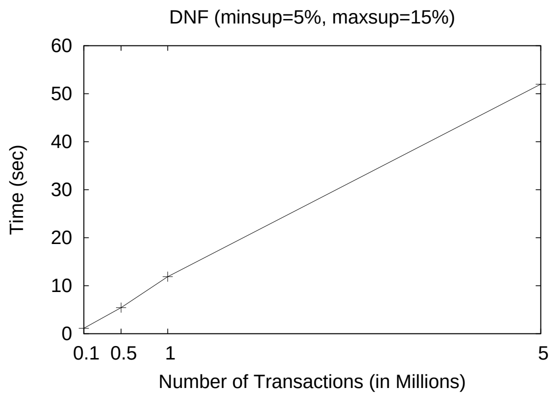
(a) AND-clauses



(b) OR-clauses



(c) CNF



(d) DNF

Figure 11: Scalability of BLOSOM.

datasets with up to 5 million transactions, and with 50 base items, and all other default parameters. Fig. 11 shows the scalability of the different instances of the BLOSOM framework. In essence, the methods all scale linearly in the number of transactions.

7.2 Bioinformatics Applications

We applied our BLOSOM framework on the problem of gene exression patterns and regulatory network discovery.

Gene Expression: We applied BLOSOM to mine frequent boolean expressions as well as redescriptions between descriptors on a gene expression dataset from (Ramakrishnan et al., 2004). This dataset involves 74 genes participating in 824 descriptors, derived from Gene Ontology (GO; `www.geneontology.org`) categories, gene expression bucketing, and k-means clustering. We specifically focus on finding minimal generators to help redescribe the descriptors corresponding to k-means clusters. There are 70 clusters, and we are interested if any of these can be redescribed by the other terms, at a particular Jaccard level (i.e., the Jaccard coefficient of the corresponding tidsets should be above a certain level). One potential application is to obtain more expressive functional

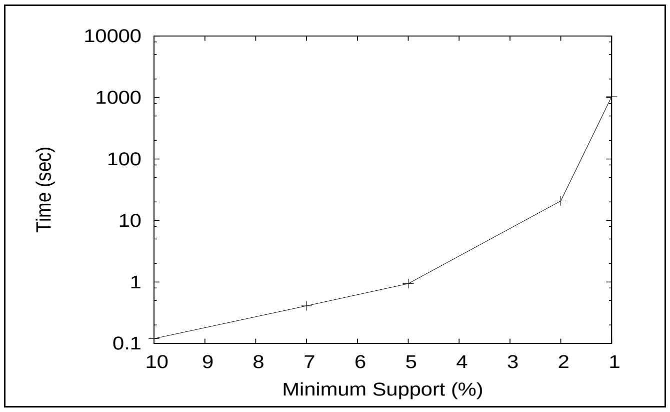


Figure 12: Gene Expression: Running Time

enrichments of these clusters in terms of GO categories. Fig. 12 shows the running time with respect to *min_sup*. With the minimum support set to 1, we were able to extract redescriptions for 9 of the 70 clusters, involving 221 minimal OR-generators at Jaccard level 0.75. For instance, a set of six genes participating in a k-means cluster, represented by descriptor $d512$ was approximately redescribed in DNF form as: $d512 \Leftrightarrow d507$ OR $d685$ OR $d700$, with Jaccard's coefficient 0.83 (i.e., $\frac{\mathbf{t}(d512) \cap \mathbf{t}(d507|d685|d700)}{\mathbf{t}(d512) \cup \mathbf{t}(d507|d685|d700)} = \frac{5}{6} = 0.83$), where $d507$ denotes genes in the GO biological process category 'response to heat,' $d685$ corresponds to genes in GO cellular location category 'extracellular,' and $d700$ corresponds to genes in GO molecular function category 'exopeptidase.' This redescription shows that the concerted activity of a set of genes in a heat shock experiment derives from their role as either heat shock factors, extracellular signaling, or the (downstream) catalytic removal of an amino acid from a polypeptide chain. This showcases the power of BLOSOM to uncover meaningful biological descriptions of gene clusters.

Gene Regulatory Networks: Another application of BLOSOM is in finding complex gene regulatory networks, which can be represented in a simplified form, as boolean networks (Akutsu et al., 1998). Consider the network involving 16 genes, taken from (Akutsu et al., 1998), shown in Fig. 13.

Here $\oplus$ and $\ominus$ denote gene *activation* and *deactivation*, respv. For example, genes B, E, H, J, and M are expressed if their parents are not expressed. On the other hand G, L, and D express if all of their parents express. For example, D depends on C, F, $X1$ and $X2$. Note that F expresses if A does, but not L. Finally A, C, I, K, N, $X1$ and $X2$ do not depend on anyone, and can thus be considered as *input* variables for the boolean network. We generated the truth table corresponding to the 7 input genes but BLOSOM was provided the values for all genes, without explicit instruction about which are inputs and which are outputs. This yields a dataset with 128 rows and 16 items (genes). We then ran BLOSOM to discover the boolean expression corresponding to this gene network. We mined using $min\_sup = 1$ and extracted only those patterns with support 100%, since we want to find expressions that are true for the entire set of assignments. BLOSOM output 65 expressions in 0.36s, which hold true for the entire dataset. After simplification these can be reduced to the equivalent expression, as shown in Fig. 14. We verified that indeed this expression is true for all the rows in the dataset! It also allows us to reconstruct the boolean gene network shown in Fig. 13. For example, the first component of the

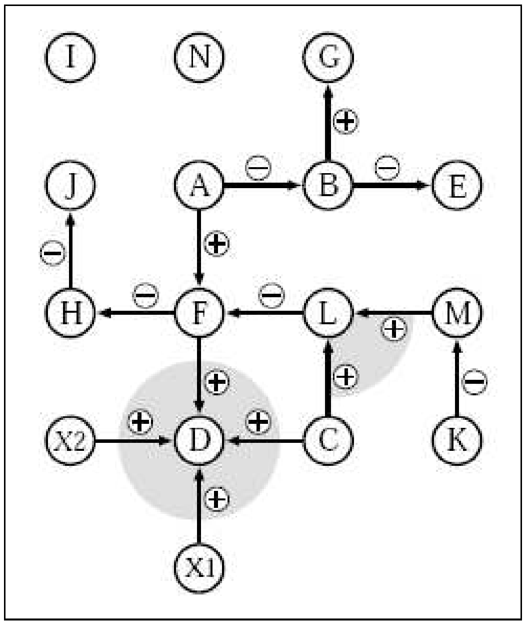


Figure 13: Gene Network

expression in the first row $\overline{D} \mid (A\overline{B}CEF\overline{G}\overline{H}JK\overline{L}\overline{M}X1X2)$ can be converted into the implication $D \Rightarrow (A\overline{B}CEF\overline{G}\overline{H}JK\overline{L}\overline{M}X1X2)$, which means that D depends on the variables on the right hand side (RHS). If, at this point, we supply any partial knowledge about the input variables or of the maximum fan-out of the network, we could project the RHS only on those variables to obtain $(ACKX1X2)$, which happens to be precisely the relationship given in Fig. 13. The second row tells us that L depends on the activation of C and inactivation of K, i.e., $\overline{K}$, if we restrict ourselves to the input variables. Also C and $\overline{K}$ give the values for the remaining variables in the second row. Note that other dependencies in the boolean network are also included in the mined expression. For example, we find that B and A always have opposite values, and so do B and E, and K and M. G and B always have the same values, and so on. Thus this example shows the power of BLOSOM in mining gene regulatory networks.

$$\begin{array}{l}
\big(\overline{D} \mid (A\,\overline{B}\,C\,E\,F\,\overline{G}\,\overline{H}\,J\,K\,\overline{L}\,\overline{M}\,X1\,X2)\big) \text{ AND} \\
\big(\overline{L} \mid (C\,\overline{F}\,H\,\overline{J}\,\overline{K}\,M)\big) \text{ AND} \\
\big((\overline{A}\,B\,\overline{E}\,G) \mid \overline{C} \mid D \mid L \mid \overline{X1} \mid \overline{X2}\big) \text{ AND} \\
\big((\overline{A}\,B\,\overline{E}\,G) \mid (C\,L) \mid (F\,\overline{H}\,J)\big) \text{ AND} \\
\big((\overline{F}\,H\,\overline{J}) \mid (A\,\overline{B}\,\overline{C}\,E\,\overline{G}) \mid (A\,\overline{B}\,E\,\overline{G}\,K\,\overline{M})\big)
\end{array}$$

Figure 14: Boolean Network Expression

8 Conclusions

In this paper we present the first algorithm, BLOSOM, to simultaneously mine closed boolean expressions over attribute sets and their minimal generators. Our four-category division of the space of boolean expressions yields a compositional approach to the mining of arbitrary expressions, along with their minimal generators. The pruning operators employed here have resulted in orders of magnitude speedup, producing highly efficient implementations.

There are still many interesting issues to consider. The first one involves the effective handling of negative literals without being overwhelmed by dataset density. The second issue is to push tautological considerations deeper into the mining algorithm by designing new pruning operators. Finally, given a general propositional reasoning framework, we are interested in mining the simplest boolean expressions necessary for inference in that framework.

Acknowledgments

This work was supported in part by NSF grants EMT-0829835 and EIA-0103708, and NIH grant 1R01EB0080161-01A1.

References

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., & Verkamo, A. I. (1996). Fast discovery of association rules. *Advances in Knowledge Discovery and Data Mining* (pp. 307–328). AAAI Press, Menlo Park, CA.

Akutsu, T., Kuhara, S., Maruyama, O., & Miyano, S. (1998). Identification of gene regulatory networks by strategic gene disruptions and gene overexpressions. *ACM-SIAM Symposium on Discrete Algorithms*.

Antonie, M.-L., & Zaiane, O. (2004). Mining positive and negative association rules: An approach for confined rules. *European PKDD Conf.*

Bastide, Y., Taouil, R., Pasquier, N., Stumme, G., & Lakhal, L. (2000). Mining frequent patterns with counting inference. *SIGKDD Explorations*, *2*.

Bayardo, R. J., & Agrawal, R. (1999). Mining the most interesting rules. *ACM SIGKDD Conf.*

Bshouty, N. (1995). Exact learning boolean functions via the monotone theory. *Information and Computation*, *123*, 146–153.

Calders, T., & Goethals, B. (2003). Minimal k-free representations of frequent sets. *European PKDD Conf.*.

Calders, T., & Goethals, B. (2005). Quick inclusion-exclusion. *ECML-PKDD Workshop on Knowledge Discovery in Inductive Databases*.

Davey, B. A., & Priestley, H. A. (1990). *Introduction to lattices and order*. Cambridge University Press.

Dong, G., Jiang, C., Pei, J., Li, J., & Wong, L. (2005). Mining succinct systems of minimal generators of formal concepts. *Int'l Conf. Database Systems for Advanced Applications*.

Ganter, B., & Wille, R. (1999). *Formal concept analysis: Mathematical foundations*. Springer-Verlag.

Goethals, B., & Zaki, M. (2003). Advances in frequent itemset mining implementations: report on FIMI'03. *SIGKDD Explorations*, *6*, 109–117.

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

Jaroszewicz, S., & Simovici, D. A. (2002). Support approximations using bonferroni-type inequalities. *6th European Conference on Principles of Data Mining and Knowledge Discovery.*

Kryszkiewicz, M. (2001). Concise representation of frequent patterns based on disjunction-free generators. *Int'l Conf. on Data Mining.*

Kryszkiewicz, M. (2005). Generalized disjunction-free representation of frequent patterns with negation. *Journal of Experimental & Theoretical Artificial Intelligence*, *17*, 63–82.

Mannila, H., & Toivonen, H. (1996). Multiple uses of frequent sets and condensed representations. *International Conference on Knowledge Discovery and Data Mining.*

Mielikainen, T. (2003). Intersecting data to closed sets with constraints. *Workshop on Frequent Itemset Mining Implementations.*

Mitchell, T. (1982). Generalization as search. *Artificial Intelligence*, *18*, 203–226.

Nanavati, A., Chitrapura, K., Joshi, S., & Krishnapuram, R. (2001). Association rule mining: Mining generalised disjunctive association rules. *ACM CIKM Conf.*.

Pasquier, N., Bastide, Y., Taouil, R., & Lakhal, L. (1999). Discovering frequent closed itemsets for association rules. *7th Intl. Conf. on Database Theory.*

Pfaltz, J., & Jamison, R. (2001). Closure systems and their structure. *Information Sciences*, *139*, 275–286.

Ramakrishnan, N., Kumar, D., Mishra, B., Potts, M., & Helm, R. (2004). Turning cartwheels: An alternating algorithm for mining redescriptions. *ACM SIGKDD Conf.*.

Savasere, A., Omiecinski, E., & Navathe, S. (1998). Mining for strong negative associations in a large database of customer transactions. *ICDE Conf.*

Shima, Y., Mitsuishi, S., Hirata, K., & Harao, M. (2004). Extracting minimal and closed monotone dnf formulas. *Int'l Conf. on Discovery Science.*

Wu, X., Zhang, C., & Zhang, S. (2004). Efficient mining of both positive and negative association rules. *ACM Trans. on Information Systems*, *22*, 381–405.

Yuan, X., Buckles, B. P., Yuan, Z., & Zhang, J. (2002). Mining negative association rules. *7th International Symposium on Computers and Communications.*

Zaki, M., & Ramakrishnan, N. (2005). Reasoning about sets using redescription mining. *ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining.*

Zaki, M. J., & Hsiao, C.-J. (2005). Efficient algorithms for mining closed itemsets and their lattice structure. *IEEE Transactions on Knowledge and Data Engineering*, *17*, 462–478.

Zhao, L., Zaki, M., & Ramakrishnan, N. (2006). Blosom: A framework for mining arbitrary boolean expressions. *ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining.*