BLOSOM: A Framework for Mining Arbitrary Boolean Expressions over Attribute Sets *

Lizhuang Zhao and Mohammed J. Zaki
Department of Computer Science
Rensselaer Polytechnic Institute, Troy, NY 12180
{zhaol2, zaki}@cs.rpi.edu

Naren Ramakrishnan
Department of Computer Science
Virginia Tech., Blacksburg, VA 24061
naren@cs.vt.edu

February 25, 2006

Abstract

We introduce a novel framework (BLOSOM) for mining (frequent) boolean expressions over binary-valued datasets. We organize the space of boolean expressions into four categories: pure conjunctions, pure disjunctions, conjunction of disjunctions, and disjunction of conjunctions. For each category, we propose a *closure* operator that naturally leads to the concept of a *closed* boolean expression. The closed expressions and their *minimal generators* give the most specific and most general boolean expressions that are satisfied by their corresponding object set. Further, the closed/minimal generator expressions form a lossless representation of all possible boolean expressions. BLOSOM efficiently mines several forms of frequent boolean expressions by utilizing a number of methodical pruning techniques. Experiments showcase the behavior of BLOSOM with respect to different input settings and parameter thresholds. An application study on real datasets is also given.

*This work was supported in part by NSF CAREER Award IIS-0092978, DOE Career Award DE-FG02-02ER25538, NSF grants EIA-0103708, EMT-0432098, EIA-0103660, IBN-0219332, and NIH/NIAID grant N01-AI-40035.

1 Introduction

One of the basic goals of data mining is to discover novel patterns that are potentially useful. Within the database community the frequent pattern mining problem, especially itemset mining, has received a lot of attention, ever since the problem was first introduced in [1].

Itemset mining takes as input a binary-valued dataset and discovers patterns that are pure conjunctions of items. Itemset mining hence covers a very restricted subset of the wider class of arbitrary boolean patterns, which may consist of conjunctions, disjunctions, and negations of items. Mining such boolean patterns can lead to significant nuggets of knowledge, with many potential applications in market basket analysis, web usage mining, social network analysis, and bioinformatics. For example, in market-basket analysis, given customer point-of-sales data, a grocery store may find that many customers buy (*pizza* OR *cookies*) AND (*coke* OR *milk*). In social network analysis, given the voting records for senators in the US senate, we may discover a group of senators who support resolutions on *increased educational spending* OR *welfare spending*, but NOT *voluntary school prayer*. A secondary analysis of the group, based on the liberal or conservative leanings of the senators may reveal additional insight. For example, one would expect the above group to contain mainly liberals, but if we do find some conservatives, that might be potentially interesting. Consider also an application in bioinformatics; 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. Boolean expression mining is also useful in the problem of finding redescriptions [17], that is to find alternative ways of describing a group of objects. For instance, we may find that political candidates who support the *death penalty* but NOT *campaign finance reform* are the same as candidates who oppose *amnesty for illegal immigrants* AND *environment and labor standards* in trade considerations. This is a redescription relating a difference of two properties to an intersection of two other properties.

It is apparent that 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 boolean expressions, far too many to enumerate. To render the search tractable 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 that still from a lossless representation of all possible boolean expressions. These minimal expressions are in fact the *minimal generators* of the closed 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), to simultaneously mine closed boolean expressions over attribute sets and their minimal generators. 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 (conjunction

of disjunctions), and (iv) disjunctive normal form (disjunction of conjunctions). The theory and algorithms developed for (i) and (ii) are then used as building blocks to develop solutions for (iii) and (iv). For each case, we propose a *closure* operator, which yields closed boolean expressions. The closed expressions and their minimal generators give the most specific and most general boolean expressions that are satisfied by their corresponding object set. Further, the closed/minimal generator expressions form a lossless representation of all possible boolean expressions in the respective category. The BLOSOM framework is can efficiently determine minimal generators for each of the four classes of boolean expressions: AND-clauses (BLOSOM-MA), OR-clauses (BLOSOM-MO), and CNF (BLOSOM-MC) and DNF (BLOSOM-MD) expressions. In addition, BLOSOM mines closed expressions for AND-clauses (BLOSOM-CA) and OR-clauses (BLOSOM-CO), and can also mine closed DNF (BLOSOM-CD) and CNF (BLOSOM-CC) expressions. 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 relationship pruning, sibling merging, threshold pruning, and fast subsumption checking. BLOSOM utilizes a novel *extraset* data structure for fast frequency computations, and to identify the corresponding transaction set (or more generally, object set) for a given arbitrary boolean expression. 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.

2 Preliminary Concepts

Boolean Expressions Let $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ be a set of binary-valued attributes or *items* (also called variables). Let $\wedge$, $\vee$ denote the binary operators standing for the logical AND and OR, respv., and let $\neg$ denote the unary operator for logical negation. A *literal* is either an item i or its negation $\neg i$. 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 $\wedge$ operator over all its literals, and an OR-clause is one that has only the $\vee$ operator over all its literals, e.g., $i_2 \wedge i_4$ is an AND-clause, while $i_3 \vee i_5$ is an OR-clause. 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. Parentheses are used to demarcate clauses, and operator precedence, when necessary. The length of an expression, E, written as $|E|$, is the number of literals it has.

A boolean expression is said to be in *negated normal form* (NNF) if all $\neg$ 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, $(i_3 \wedge i_4) \vee (i_1 \wedge i_5 \wedge i_7)$ is in DNF, whereas $(i_2 \vee i_3) \wedge (i_0 \vee i_1 \vee (\neg i_3))$ is in CNF. Note that by definition a single OR-clause is in CNF, since it is an AND of a single OR-clause; likewise, a single AND-clause is in DNF. Furthermore, when considering negated literals, we disallow tautologies like the OR-clause $i|\bar{i}$ which is always

true. Similarly, we disallow contradictions like the AND-clause containing $i \wedge \bar{i}$ 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/contradictions in individual clauses, we disallow tautologies/contradictions in any expression.

For compactness, henceforth, we denote a negated item $\neg i$ as $\bar{i}$, we use the symbol | instead of $\vee$ to denote OR, and simply omit the operator $\wedge$ whenever there is no ambiguity; thus two adjacent clauses without any operator will always denote an AND. For example, $(i_2 \vee i_3) \wedge (i_0 \vee i_1 \vee (\neg i_3))$ is rewritten as $(i_2|i_3)(i_0|i_1|\bar{i_3})$, and $(i_3 \wedge i_4) \vee (i_1 \wedge i_5 \wedge i_7)$ is rewritten as $(i_3 i_4)|(i_1 i_5 i_7)$.

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 $\mathcal{T} \times 2^{\mathcal{I}}$ (note that $2^{\mathcal{I}}$ denotes the power-set of $\mathcal{I}$, i.e., the set of all subsets of $\mathcal{I}$); 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. Figure 1 shows a dataset and its transpose, which we will use as a running example in this paper. It has five items $I = \{A, B, C, D, E\}$ 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).

$\mathcal{D}$	
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

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

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 $|\mathbf{t}(E)| \geq min\_sup$. For disjunctive expressions, we also impose a *maximum support* threshold (max_sup) to disallow any expression with too high a support. Setting $min\_sup = 1$ and $max\_sup = \infty$ allows mining all possible expressions.

Boolean Expression Mining Tasks Given a dataset $\mathcal{D}$ and support thresholds min_sup and/or max_sup, we are interested in mining various types of frequent boolean expressions over the item space, such as AND-clauses (i.e., pure conjunctions), OR-clauses (i.e., pure disjunctions), CNF and DNF. Moreover, instead of mining all such frequent boolean expressions, we focus our attention on mining only (frequent) closed boolean expressions and their minimal generators.

Before presenting the BLOSOM framework, we 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 restrict our examples to only positive literals, but our approach is applicable to negated literals as well.

3 Mining Simple Boolean Expressions: Clauses

In this section we give a structural characterization of pure OR-clauses and pure AND-clauses.

3.1 Mining OR-clauses

We first consider OR-clauses or pure disjunctive boolean expressions. Given dataset $\mathcal{D}$, and thresholds min_sup and max_sup, the goal is to mine OR-clauses that occur in at least min_sup and in at most max_sup transactions. To the best of our knowledge, mining closed and minimal OR-clauses have not been studied previously.

Let's review a few facts from lattice theory [7]. Let $(P, \subseteq)$ be a partially ordered set (also called a *poset*). Let $X, Y \in P$ and let $f : P \rightarrow P$ 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 $\mathbb{C} : P \rightarrow P$ 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 $\mathbb{K} : P \rightarrow P$, which is monotone, idempotent, and intensive. X is called *open* if $\mathbb{K}(X) = X$. Thus, 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, $\phi : P \rightarrow Q$ and $\psi : Q \rightarrow P$, such that for all $X \in P$ and

$Y \in Q$, we have: $X \subseteq \psi(Y) \iff \phi(X) \leq Y$. Finally it is known that the 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 [7].

Now, 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.

Definition: 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 the relation $\subseteq$ induces a partial order over $\mathcal{E}^{\vee}$. For example, $A|C \subseteq A|B|C|D$, since $\mathcal{L}(A|C) = AC \subseteq ABCD = \mathcal{L}(A|B|C|D)$.

Lemma 1 *Let $X \in \mathcal{E}^{\vee}$ be an OR-clause, and let $l \in X$ be some literal in X. Then* $\mathbf{t}(X) = \bigcup_{l \in X} \mathbf{t}(l)$. ∎

Let's consider our example dataset from Figure 1. Starting from $\mathcal{D}^T$, which has the tidsets for individual items, we can obtain all OR-clauses by simply taking the union of the tidsets as we enumerate the possible candidate clauses. Figure 2 shows the partial order $(\mathcal{E}^{\wedge}, \subseteq)$ obtained from $\mathcal{D}$ using $min\_sup = 1$ and $max\_sup = |\mathcal{D}| = 5$. Note that in general max_sup should be less than $|\mathcal{D}|$, since an OR-clause that is always true in $\mathcal{D}$ is not likely to be interesting. Figure 2 also shows the tidset for each clause. For example, $\mathbf{t}(A|B|C|D) = 1234$.

Closed OR-Clauses: 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. Let's define a Galois connection between these two partial orders. Below for convenience we also write an OR-clause $l_1|l_2|\cdots|l_k$ as $\bigvee\{l_1 l_2 \cdots l_k\}$.

Theorem 2 *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 $2^{\mathcal{T}}$:*

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

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

Since $(\mathbf{t}, \mathbf{i}^{\vee})$ form a monotone Galois connection, it follows immediately that $\mathbb{C}^{\vee} = \mathbf{i}^{\vee} \circ \mathbf{t} : \mathcal{E}^{\vee} \to \mathcal{E}^{\vee}$ forms a closure operator and $\mathbb{K}^{\vee} = \mathbf{t} \circ \mathbf{i}^{\vee} : 2^{\mathcal{T}} \to 2^{\mathcal{T}}$ forms a kernel operator for OR-clauses. For example, in our example dataset from Figure 1, $\mathbb{C}^{\vee}(A|C) = \mathbf{i}^{\vee}(\mathbf{t}(A|C)) = \mathbf{i}^{\vee}(1234) = A|B|C|D$. Thus $A|B|C|D$ is a closed OR-clause. On the other hand $\mathbb{K}^{\vee}(234) = \mathbf{t}(\mathbf{i}^{\vee}(234)) = \mathbf{t}(B) = 23$. Thus 23 is an open tidset. It is also easy to see that the corresponding tidset for a closed OR-clause is always open. Figure 2 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 $\mathbb{C}^{\vee}(\mathcal{E}^{\vee})$ to denote the set of all closed OR-clauses.

Minimal OR-Clauses: Given any set $\mathbf{X}$ of subsets over a universe U, we denote by $\min_{\subseteq}(\mathbf{X})$ the set of all minimal members in $\mathbf{X}$, w.r.t. the subset operator $\subseteq$. Let $X \in \mathcal{E}^{\vee}$ be a closed OR-clause. We say that $Y \subseteq X, Y \neq \emptyset$ is a *generator* of X if $\mathbb{C}^{\vee}(Y) = X$. Y is called a *proper* generator if $Y \neq X$. By definition, a proper generator cannot be closed. Let $\mathcal{G}(X)$ be the

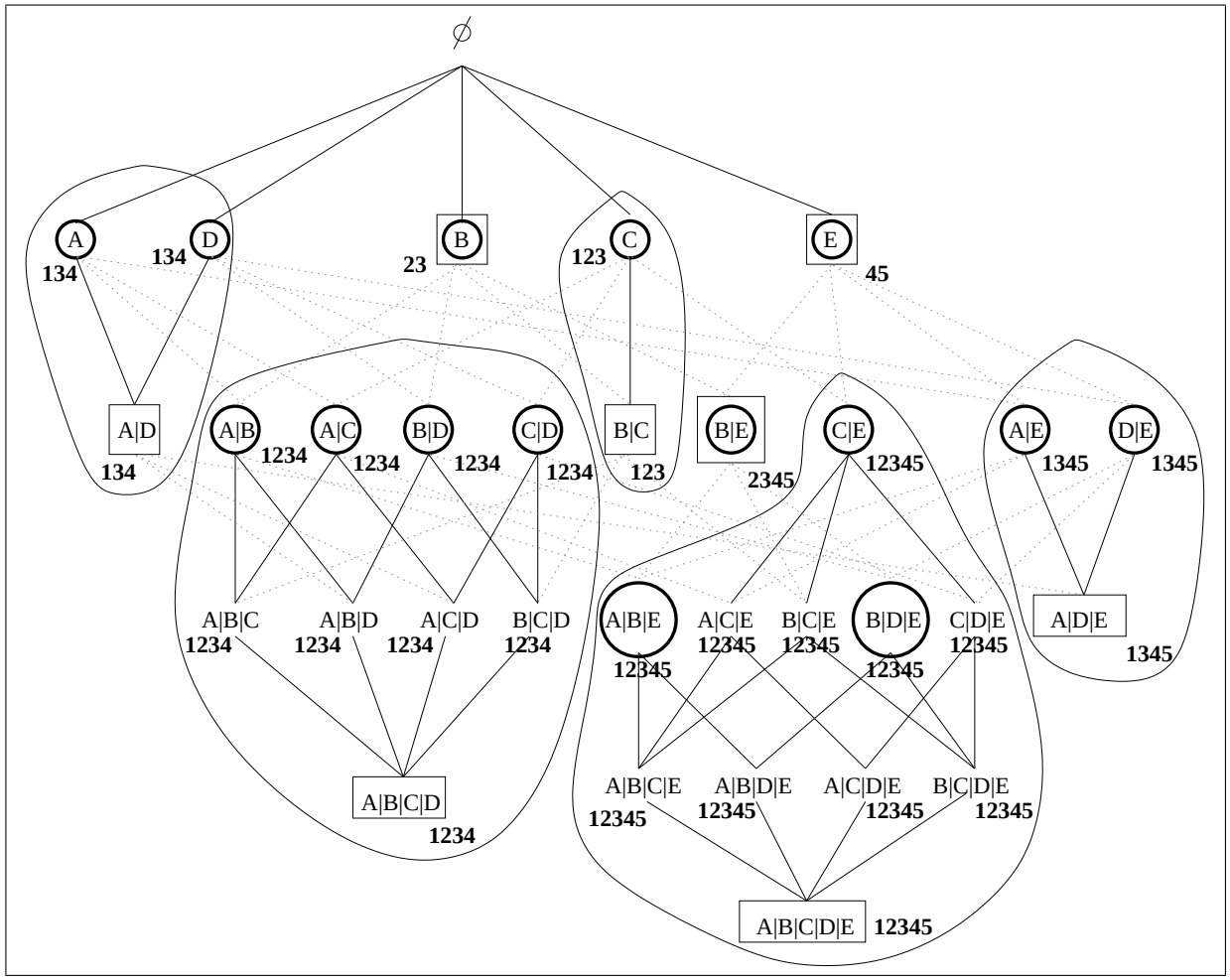

Figure 2: OR-Clauses: Closed (rectangles) and Minimal (circles). Groupings denote generators

set of all generators of X, including X. Then $\mathcal{M}(X) = \min_{\subseteq}\{Y \in \mathcal{G}(X)\}$, the set of all the minimal elements of $\mathcal{G}(X)$ are called the *minimal generators* of X. The unique maximal element of $\mathcal{G}(X)$ is X.

Lemma 3 *Let $X = \mathbb{C}^{\vee}(X)$. If Y is a generator of X then $\mathbf{t}(Y) = \mathbf{t}(X)$.* ∎

By Lemma 3, $\mathbf{t}(Y) = \mathbf{t}(X) = T$ for all generators $Y \in \mathcal{G}(X)$. We conclude that X is the *unique* maximal OR-clause that describes the set of objects T. On the other hand, a minimal generator of X is the minimal or simplest OR-clause that still describes the same object set T. In other words, the closed clause X is the most specific expression that describes the tidset T, whereas the minimal generators are the most general expressions that describe T. The set of all minimal generators of closed OR-clauses in $\mathcal{E}^{\vee}$ is given as $\mathcal{M}_{\mathbb{C}^{\vee}}(\mathcal{E}^{\vee}) = \bigcup_{X \in \mathbb{C}^{\vee}(\mathcal{E}^{\vee})} \mathcal{M}(X)$.

Figure 2 shows the groups of all generators of closed OR-clauses in our example dataset (from Figure 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). It is easy to see that the tidsets are the same for each member of a group, but different across groups. The information about tidsets, closed OR-clauses and their minimal generators is summarized in Table 1.

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 1: Closed (CO) and Minimal (MO) OR-Clauses

It should be apparent at this point that both closed OR-clauses (CO) and minimal OR-clauses (MO), 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, 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* 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 $X_i \in \mathbf{X}^\ell$.

Theorem 4 *Let X be a closed* OR*-clause, and let $\mathbf{X}^\delta$ be the differential lower shadow of X. Then $\mathcal{M}(X) = \min_\subseteq\{Z \in \mathcal{E}^\vee \mid Z \in \mathcal{H}(\mathbf{X}^\delta)\}$.* ∎

The theorem states 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 $X = A|B|C|D|E$ in Figure 2. We have 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 1, 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. 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 $\mathcal{R}(T) = \{t.X \subseteq \mathcal{I} | t \in T, (t, t.X) \in \mathcal{D}\}$, i.e., the set transactions in $\mathcal{D}$ with tids $t \in T$.

Theorem 5 *Let $T \in \mathcal{U}$ be a union tidset. Then the set $\min_\subseteq\{Z \in \mathcal{E}^\vee \mid Z \in \mathcal{H}(\mathcal{R}(T))$ and $\mathbf{t}(Z) = T\}$ is identical to the minimal generators of the closed* OR*-clause X which has the corresponding open tidset $T = \mathbf{t}(X)$.* ∎

The above theorem states that every distinct tidset T obtained as a union of other tidsets produces minimal generators for the closed OR-clause associated with the tidset T. For example, let $T = 1345 = \mathbf{t}(A) \cup \mathbf{t}(E)$ in our example database in Figure 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 1 that these minimal hitting sets form the minimal generators of the closed OR-clause $A|D|E$ with tidset $T = 1345$.

3.2 Mining AND-clauses

Closed AND-clauses have been well studied in data mining as closed itemsets [15], as well as in the Formal Concept Analysis as concepts [9]. The notion of minimal generators for AND-clauses has also been previously proposed in [3]. We thus focus on our novel structural insights for AND-clauses.

Given posets $(P, \subseteq)$ and $(Q, \leq)$, a *anti-monotone Galois connection* [7] 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: $X \subseteq \psi(Y) \iff Y \leq \phi(X)$. Also, 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 [7].

Let $\mathcal{E}^\wedge$ be the set of all AND-clauses over the set of items $\mathcal{I}$, let $(\mathcal{E}^\wedge, \subseteq)$ be the partial order over $\mathcal{E}^\wedge$, and let $(2^\mathcal{T}, \subseteq)$ be the partial order over $2^\mathcal{T}$. Let $X \in \mathcal{E}^\wedge$ and $Y \in 2^\mathcal{T}$, then the following two mappings form an anti-monotone Galois connection [9]:

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

Note that for OR-clauses $\mathbf{i}^\vee$ and $\mathbf{t}$ are both order-preserving, and form a monotone Galois connection, whereas for AND-clauses $\mathbf{i}^\wedge$ and $\mathbf{t}$ are both order-reversing and form an anti-monotone Galois connection. Note also that $\mathbf{i}^\wedge(Y)$ is the set of items i with $Y \subseteq \mathbf{t}(i)$, whereas $\mathbf{i}^\vee(Y)$ is the set of items i with $\mathbf{t}(i) \subseteq Y$. This formulation of $\mathbf{i}^\wedge$ is novel, but equivalent to that in [9].

Since $(\mathbf{i}^\wedge, \mathbf{t})$ forms a anti-monotone Galois connection, $\mathbb{C}^\wedge = \mathbf{i}^\wedge \circ \mathbf{t} : \mathcal{E}^\wedge \to \mathcal{E}^\wedge$ forms a closure operator for AND-clauses [9]. Due to the anti-monotonicity, the mapping $\mathbf{t} \circ \mathbf{i}^\wedge$:

$2^{\mathcal{T}} \rightarrow 2^{\mathcal{T}}$ also forms a closure operator (expansive) on tidsets, whereas for OR-clauses the mapping $\mathbf{t} \circ \mathbf{i}^{\vee}$ was a kernel operator (contractive). We use the notation $\mathbb{C}^{\wedge}(\mathcal{E}^{\wedge})$ to denote the set of all closed AND-clauses. Note that as before, Y is a minimal generator for a closed AND-clause X iff Y is a minimal subset of X such that $\mathbf{t}(Y) = \mathbf{t}(X)$. Let $\mathcal{M}(X)$ denote the set of minimal generators of X, and let $\mathcal{M}_{\mathbb{C}^{\wedge}}(\mathcal{E}^{\wedge}) = \bigcup_{X \in \mathbb{C}^{\wedge}(\mathcal{E}^{\wedge})} \mathcal{M}(X)$ be the set of all minimal generators of AND-clauses.

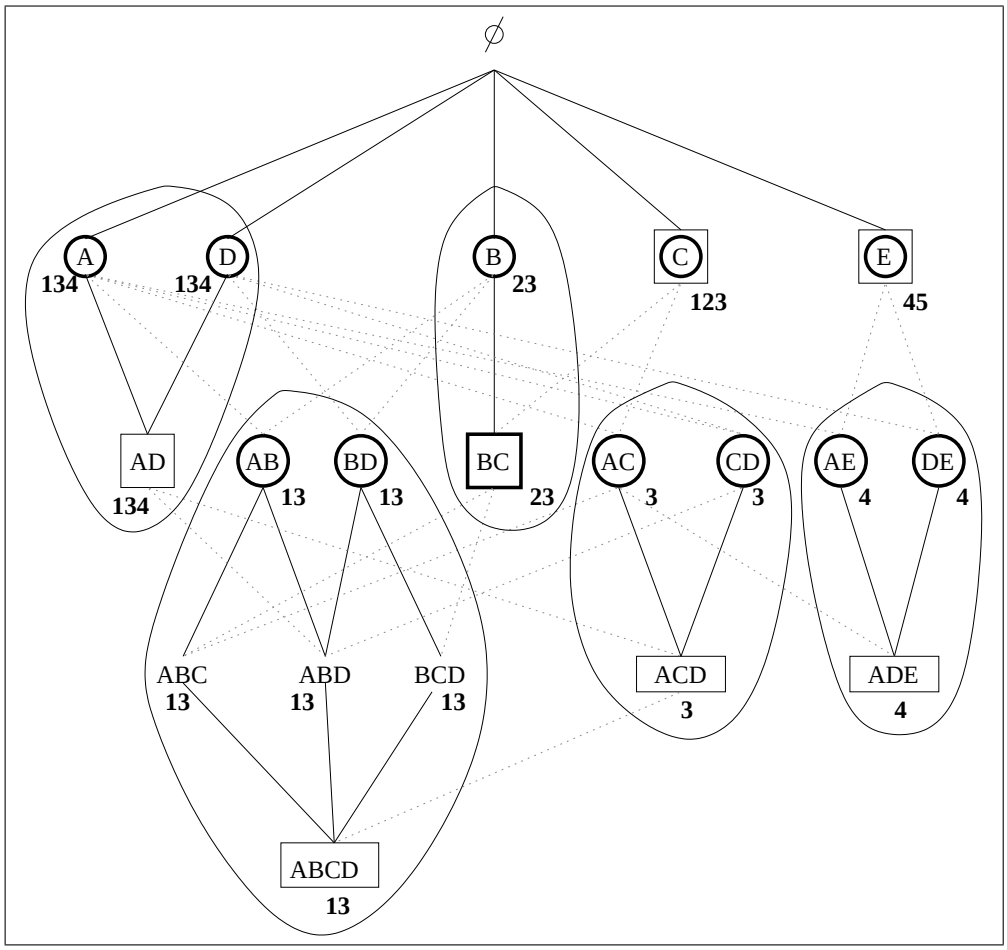

Figure 3: Closed and Minimal AND-Clauses

Consider our example dataset from Figure 1. Figure 3 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 AND-clause. The 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, and they both form a lossless representation of all AND-clauses. Table 2 lists the set of all closed (CA) AND-clauses, as well as their minimal generators (MA).

In terms of the strucure of minimal generators for AND-clauses, an analog of Theorem 4, describing the minimal generators of a closed AND-clause $X = \mathcal{E}^{\wedge}(X)$ as the minimal hitting

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

sets of its differential lower shadow is already known [16]. We focus instead on a novel characterization of the 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, $\mathbf{t}(i)|i \in \mathcal{I}$. Let $\mathcal{M}$ be the set of all distinct intersection tidsets. As before, for an intersection tidset $T \in \mathcal{M}$, we define the *transaction set* of T, denoted $\mathcal{R}(T)$, as the set of transactions with tids $t \in T$. For example, $T = 13$ is an intersection tidset since it can be obtained as the intersection of $\mathbf{t}(A)$ and $\mathbf{t}(C)$. The transaction set of T is the set of transactions with tids 1, and 3, given as $\mathcal{R}(T) = \{ACD, ABCD\}$.

Theorem 6 *Let $T \in \mathcal{M}$ be an intersection tidset. Then the set $\min_{\subseteq}\{Z \in \mathcal{E}^{\wedge} \mid Z \in \mathcal{H}(\mathcal{R}(T))$ and $\mathbf{t}(Z) = T\}$ is identical to the minimal generators of the closed AND-clause X which has the corresponding closed tidset $T = \mathbf{t}(X)$.* ∎

This theorem gives a novel structural description of the minimal AND-clauses. It states that every distinct tidset T obtained as a finite intersection of other tidsets produces minimal generators for some AND-clause. For example, let $T = 13 = \mathbf{t}(A) \cap \mathbf{t}(C)$ in our example database in Figure 1. Then $\mathcal{R}(T) = \{ACD, ABCD\}$. The hitting sets Z of $\mathcal{R}(T)$ with $\mathbf{t}(Z) = T$ and that are minimal are given as follows $\{AC, CD\}$. Note that A is a minimal hitting set, but $\mathbf{t}(A) = 134$, thus we reject it. Likewise we reject hitting sets C, D, AB, BD. We can see from Table 2 that the minimal hitting sets are identical to the minimal generators of the closed AND-clause ACD with tidset $T = 13$.

4 Mining Complex Boolean Expressions: Normal Forms

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

4.1 Mining DNF Expressions

Let $\mathcal{E}^{\text{dnf}}$ denote the set of all boolean expression 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 ($\mathbb{C}^{\wedge}(\mathcal{E}^{\wedge})$) and minimal ($\mathcal{M}_{\mathbb{C}^{\wedge}}(\mathcal{E}^{\wedge})$) AND-clauses, and their corresponding tidsets.

Note that any DNF-expression $X = \bigvee X_i$ is equivalent to the DNF expression $X' = \bigvee \mathbb{C}^{\wedge}(X_i)$, since any tidset that satisfies X_i must satisfy $\mathbb{C}^{\wedge}(X_i)$ as well. Similarly X is also equivalent to the DNF expression $X'' = \bigvee \mathcal{M}(X_i)$, since any $\mathbf{t}(X_i) = \mathbf{t}(\mathcal{M}(X_i))$.

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 $X \sqsubseteq Y$, if there exists a 1-1 mapping f that maps each $X_i \in X$ to $f(X_i) = Y_j \in Y$, such that $X_i \subseteq Y_j$.

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 $X_i \in \mathbb{C}^{\wedge}(\mathcal{E}^{\wedge})$ 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 $X_i \in \mathbb{C}^{\wedge}(\mathcal{E}^{\wedge})$, and let $Y \in 2^{\mathcal{T}}$. Define $\mathbf{t}(X) = \{t \in \mathcal{T} \mid t \text{ satisfies } X\}$, and $\mathbf{i}^d(Y) = \bigvee\{X_i \mid X_i \in \mathbb{C}^{\wedge}(\mathcal{E}^{\wedge}) \wedge \mathbf{t}(X) \subseteq Y\}$. This implies that $\mathbb{C}^d = \mathbf{i}^d \circ \mathbf{t}$ is a closure operator. For example, consider each closed AND-clause in Table 2 as an "item". Consider $X = ACD|E$. $\mathbb{C}^d(X) = \mathbf{i}^d(\mathbf{t}(ACD|E)) = \mathbf{i}^d(1345) = ABCD|ADE|ACD|E|AD$, which is a closed DNF expression. However it is logically redundant. What we want is the maximal min-DNF expression equivalent to $\mathbb{C}^d(X)$, which is $ACD|BC|E$.

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 $\mathcal{R}(T)$, as the set of transactions in $\mathcal{D}$ with tid $t \in T$. Analogous to Theorem 5 for OR-clauses, we can characterize the minimal DNF expressions as the set $\min_{\subseteq}\{Z \in \mathcal{E}^{\text{dnf}} \mid Z \in \mathcal{H}(\mathcal{R}(T)) \text{ and } \mathbf{t}(Z) = T\}$, 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 exactly tids 3 and 4 are $AB|AE, AB|DE, BD|AE, BD|DE$. In fact the minimal DNF hitting sets can be obtained directly from minimal AND-clauses.

Theorem 7 *Let T be a union tidset, and let X be the closed DNF-expression with $\mathbf{t}(X) = T$. Then $\mathcal{M}(X) = \min_{\subseteq}\{Z = \bigvee Z_i | Z_i \in \mathcal{M}_{\mathbb{C}^{\wedge}}(\mathcal{E}^{\wedge})$ and $\mathbf{t}(Z) = T\}$.* ∎

For example, for $T = 34$, we see in Table 2 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 DNF expressions and their minimal generators, in addition to those shown in Tables 1 and 2. Some entries are repeated since, the closed expressions in DNF have changed. Also the new union tidsets are marked in bold.

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

4.2 Mining 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 $\mathcal{E}^{\vee} \subseteq \mathcal{E}^{\text{cnf}}$. Also $\mathcal{E}^{\wedge} \subseteq \mathcal{E}^{\text{cnf}}$, since an AND-clause is a CNF-expression over (OR) clauses consisting of a single literal. 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 $X_i \subseteq X_j$. Note that any CNF expression can easily be made a min-CNF-expression by simply deleting the offending clauses, e.g., for the CNF expression $(B|C)(A|B|C|D)$, we have $B|C \subseteq A|B|C|D$; thus the expression is logically equivalent to its min-CNF form $(B|C)$. For any 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 a 1-1 mapping 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 ($\mathbb{C}^{\vee}(\mathcal{E}^{\vee})$) and minimal ($\mathcal{M}_{\mathbb{C}^{\vee}}(\mathcal{E}^{\vee})$) OR-clauses, and their corresponding tidsets.

Let's treat each $X_i \in \mathbb{C}^{\vee}(\mathcal{E}^{\vee})$ as a *composite item*, we can 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 $X_i \in \mathbb{C}^{\vee}(\mathcal{E}^{\vee})$, and let $Y \in 2^{\mathcal{T}}$. Define $\mathbf{t}(X) = \{t \in \mathcal{T} \mid t \text{ satisfies } X\}$,

and $\mathbf{i}^d(Y) = \bigwedge\{X_i \mid X_i \in \mathbb{C}^\vee(\mathcal{E}^\vee) \wedge Y \subseteq \mathbf{t}(X)\}$. This implies that $\mathbb{C}^c = \mathbf{i}^c \circ \mathbf{t}$ is a closure operator. For example, consider each closed OR-clause in Table 1 as an "item". Consider $X = (A|D)(B|C)$. $\mathbb{C}^c(X) = \mathbf{i}^c(\mathbf{t}((A|D)(B|C))) = \mathbf{i}^d(13) = (B|C)(A|D)(A|B|C|D)(A|D|E)$ $(A|B|C|D|E)$, which is closed. However it is logically redundant. What we want is the maximal min-CNF expression equivalent to $\mathbb{C}^c(X)$, which is $(A|D|E)(B|C)$.

Theorem 8 *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 $\mathcal{M}(X) = \min_{\subseteq}\{Z = \bigwedge Z_i | Z_i \in \mathcal{M}_{\mathbb{C}^\vee}(\mathcal{E}^\vee)$ and $\mathbf{t}(Z) = T\}$.* ■

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

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 1 and 2, or those that have changed.

5 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.

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. Below, we give algorithmic details for mining the simple (OR- and AND-clauses) and complex expressions (DNF and CNF).

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

5.1 BLOSOM-MO: Minimal OR-Clauses

BLOSOM-MO mines all the minimal OR-generators (its pseudo-code is given in Figure 4). It 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 $(I \times T)$, where I is an item set denoting an OR-clause and T is a tidset. (as shown in Figure 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.

5.1.1 Pruning and Optimization Techniques

Threshold Pruning: BLOSOM uses the three thresholds to prune the trivial 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 siblings merging. All sibling nodes of the same parent share a common prefix, containing the item sets on the path from the

Input : min_sup, max_sup, max_item and dataset $\mathcal{D}$ in vertical format
Output : hash table $\mathcal{M}$ containing all MO of $\mathcal{D}$
Initialization: $NL = \emptyset$; for each item i of $\mathcal{D}$, add set pair $(\mathbf{t}(i) \times \{i\})$ to NL; call BLOSOM-MO($\emptyset,\ NL,\ 0,\ 0$)

```
1  BLOSOM-MO( P_0, NL_0, sup_0, sum_0 )
2  quicksort NL_0 = {n_i} in decreasing order of |n_i.T|
3  while ∃ n_1 and n_2 ∈ NL_0 such that n_1.T = n_2.T do
4      n_1.I ← n_1.I ∪ n_2.I /* sibling merging */
5      NL_0 ← NL_0 − n_2
6  foreach n_1 ∈ NL_0 do
7      P_1 ← P_0 + n_1.I
8      sup_1 = sup_0 + |n_1.T|
9      sum_1 = sum_0 + summation(n_1.T) /* to reduce conflicts mapped to the same
       entry further */
10     if sup_1 > max_sup then
11         continue /* goto line 6 */
12     if sup_1 ≥ min_sup then
13         foreach combination I_1 of P_1 do
14             /* I_1 is generated by picking one and only one item from each P ∈ P_1.
               */
15             delete all supersets of I_1 in M[sup_1×sum_1]
16             M[sup_1×sum_1] ← M[sup_1×sum_1] + I_1
17     if |P_1| ≥ max_item then
18         continue /* goto line 6 */
19     NL_1 ← ∅
20     foreach n_2 ∈ NL_0 ranking behind n_1 do
21         n_3.T ← n_2.T − n_1.T /* get extraset */
22         if |n_3.T| ≠ ∅ then
23             n_3.I ← n_2.I
24             NL_1 ← NL_1 + n_3
25     if NL_1 ≠ ∅ then
26         BLOSOM-MO( P_1, NL_1, sup_1, sum_1 )
```

Figure 4: BLOSOM-MO Algorithm

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 for tidsets. The DFS expansion involves a lot of tidset unions, and each node has to maintain such an intermediate set, which is a superset of the parent node's tidset. So for each node, we simply retain the extra-part of its parent's tidset in a data structure we call *extraset*, which loses no information, and yet saves a lot of memory while storing the intermediate tidsets. 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.

5.1.2 Algorithmic Description

We now describe the BLOSOM-MO algorithm in detail, based on the pseudo-code in Figure 4. It is a recursive algorithm that, at each call accepts a current prefix queue $\mathcal{P}_0$ containing the I sets on the path from the root to the current node, a node list NL_0 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, which proves very effective. The initial call is made with an empty $\mathcal{P}_0$, the NL containing all the single items, and the support and summation are set to 0. Line 2 sorts the current sibling nodes in NL_0 in decreasing order of support, which speeds up the convergence of the algorithm (*dynamic sibling reordering*). Lines 3-5 merge the sibling nodes with the same tidset T by unioning their item sets together (*sibling merging*). Lines 6-26 form the main loop to process each of the sibling nodes one by one. Line 7 updates the current prefix queue for further recursive calls. Line 8-9 generates the current transaction set support sup_1 and summation sum_1 by adding tids from the *extraset* $n_1.T$. Lines 10-11 check the *max_sup* threshold. Lines 12-16 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 10) and *min_sup* (line 12), then the algorithm will try to add every possible queue $\mathcal{P}_1$ item combination to the current MO hash table (lines 13-16). Each combination is generated by picking one and only one item from each $P \in \mathcal{P}_1$. At the same time, we also need to do

subsumption checking and to delete any supersets of the new MO I_1 to be added (line 15). The new MO I_1 must be minimal, i.e., it should not be subsumed by any current MO in the hash table (*fast subsumption checking*). Lines 17-18 check the *max_item* threshold. Lines 19-24 produce node n_1's valid children nodes NL_1 generated with other sibling nodes ranked after n_1 (in sorted order). BLOSOM-MO tries every node pair (line 20) and generates children nodes (line 21). It only saves the *extraset* ($n_1.T \cup n_2.T - n_1.T = n_2.T - n_1.T$) 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 22), then it is added to the valid children node list (lines 23-24) (note: line 22 constitutes the *parent-child relationship pruning*). Lines 25-26 make a recursive call with node n_1's valid children.

After the steps showed in Figure 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 for most datasets, the separating stage is not too costly.

5.1.3 An Example

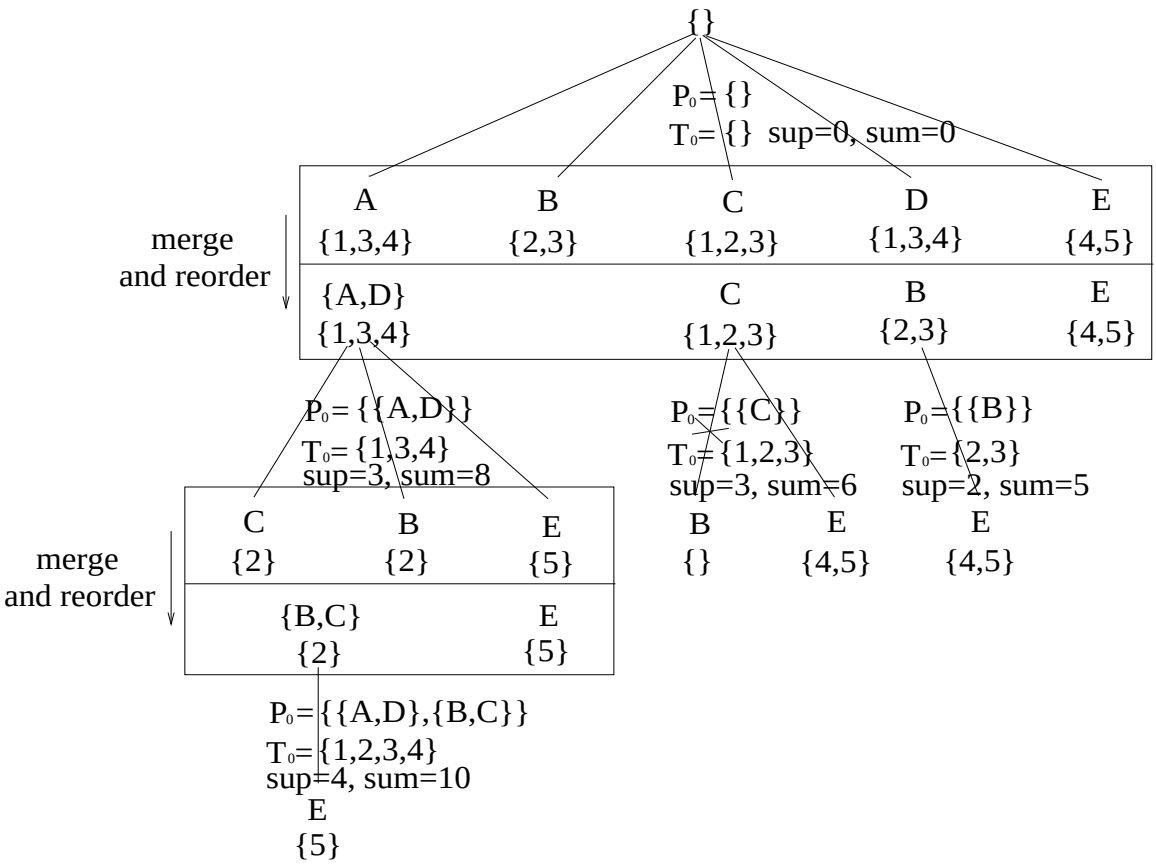

Figure 5: DFS Search Tree

Here we show an example of how BLOSOM-MO works on our example dataset $\mathcal{D}^T$ from Figure 1. Figure 5 shows the DFS search tree. Initially the prefix queue $\mathcal{P}_0$ is empty, and node list NL contains five single items. After sibling merging and reordering, they become $AD{\times}134$, $C{\times}123$, $B{\times}23$ and $E{\times}45$ as shown in Figure 5. From $\mathcal{P}_1{=}\{A, D\}$, we get two item combinations: A and D. We thus add A and D to the MOs hash table $\mathcal{M}$ with the entry $T_1 = \emptyset \cup 134 = 134$, $sup = 3$, 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 expansion results are $123{-}134{=}2$; $23{-}134{=}2$; and $45{-}134{=}5$, respectively. After reordering and merging, $NL_1 = \{\{B, C\}{\times}2,\ E{\times}5\}$. Along with $\mathcal{P}_0{=}\{\{A, D\}\}$ and $T_1{=}134$, BLOSOM is called recursively. Next step from $\mathcal{P}_1{=}\{\{A, D\}, \{B, C\}\}$, we get four combinations: $\{A, B\}$, $\{A, C\}$, $\{B, D\}$ and $\{C, D\}$, and we add them to the hash table with entry: $T_1{=}134 \cup 2{=}1234$, $sup = 4$, $tidsum = 10$. Readers can continue this process until all the MOs are generated as shown in Table 6 in subscript order. Please note in the path $Root{\rightarrow}C{\rightarrow}B$, node $B{\times}\emptyset$ 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 of the entry will be deleted (as marked by underlines) for 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 then have to verify the generators in each entry of $(sup{\times}sum) = (4{\times}10)$, $(5{\times}15)$ and $(4{\times}13)$ to assure they belong to the same entry. For this example, they do. Otherwise, we need to separate them into different entries.

T	sup	$tidsum$	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: Addition of Minimal OR Generators (MOs) to Hash Table $\mathcal{M}$

5.2 BLOSOM-CO: Closed OR-Clauses

Whereas BLOSOM-MO is designed for mining minimal generators, BLOSOM-CO is specific to mining closed expressions. The overall approach is similar to BLOSOM-MO, i.e., BLOSOM-CO performs a DFS search to find all all the close OR-clauses. The main differences are that instead of finding the *minimal* elements, we have to find the *maximal* elements corresponding to the given tidsets. This 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:

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.

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 $T \times \mathcal{M}(T)$. This pruning is required because after sibling containment pruning, there is no one direction property for CO mining when doing subsumption check as in MO mining.

Since most of the algorithmic details of BLOSOM-CO are similar to BLOSOM-MO we do not give a more detailed description.

5.3 BLOSOM-MA/CA: 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 Figure 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 take the OR of $\overline{A}$ and $\overline{B}$ and complement the final results. 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.

5.4 BLOSOM-MD/MC/CD/CC: Minimal and Closed DNF/CNF Expressions

Following the structural characterization of minimal DNF expressions in Section 4.1, BLOSOM-MD follows a two-phase approach. It first extracts all the minimal AND-clauses and then find the minimal DNF expressions using those.

Figure 6 shows the pseudo-code of BLOSOM-MD. First we use BLOSOM-MA to get all MA generators ($\mathcal{M}_{MA}$) on the original dataset (line 2). Second, we generate tidsets for each entry in $\mathcal{M}_{MA}$ (line 3) and assign each MA class a new item label (line 4). These then

Input : dataset $\mathcal{D}$ in vertical format
Output : hash table $\mathcal{M}_{MD}$ containing all MD of $\mathcal{D}$

1 BLOSOM-MD()
2 **call** BLOSOM-MA() on $\mathcal{D}$ and produce $\mathcal{M}_{MA}$
3 **generate** transaction sets for each entry in $\mathcal{M}_{MA}$
4 **assign** each entry in $\mathcal{M}_{MA}$ an item number
5 **form** a new dataset $\mathcal{D}_{new}$ from step 3 and 4
6 **call** BLOSOM-MO() on $\mathcal{D}_{new}$ and produce $\mathcal{M}_{MD}$
7 **replace** each item number in $\mathcal{M}_{MD}$ with the MA generators it represents
8 **delete** the subsumed generators in $\mathcal{M}_{MD}$

Figure 6: BLOSOM-MD Algorithm

form a new dataset (line 5). Third, we call BLOSOM-MO to get all MO generators on the new dataset (line 6). Next, we combine the results from BLOSOM-MA (line 2) and BLOSOM-MO (line 6) to form MD candidates by replacing each item label in $\mathcal{M}_{MD}$ with the MA generators it represents. Finally, we delete the subsumed generators in $\mathcal{M}_{MD}$ to produce min-DNF forms.

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. For closed CNF expressions, reverse the roles of BLOSOM-CO and BLOSOM-CA, and finally output the maximal min-CNF expressions.

6 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 real datasets are used to highlight the kinds of knowledge mined by our approach, whereas the synthetic datasets are used primarily to study the effect of various parameters and demonstrate scalability.

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 $t(i)$ with equal probability.

6.1 Performance Study

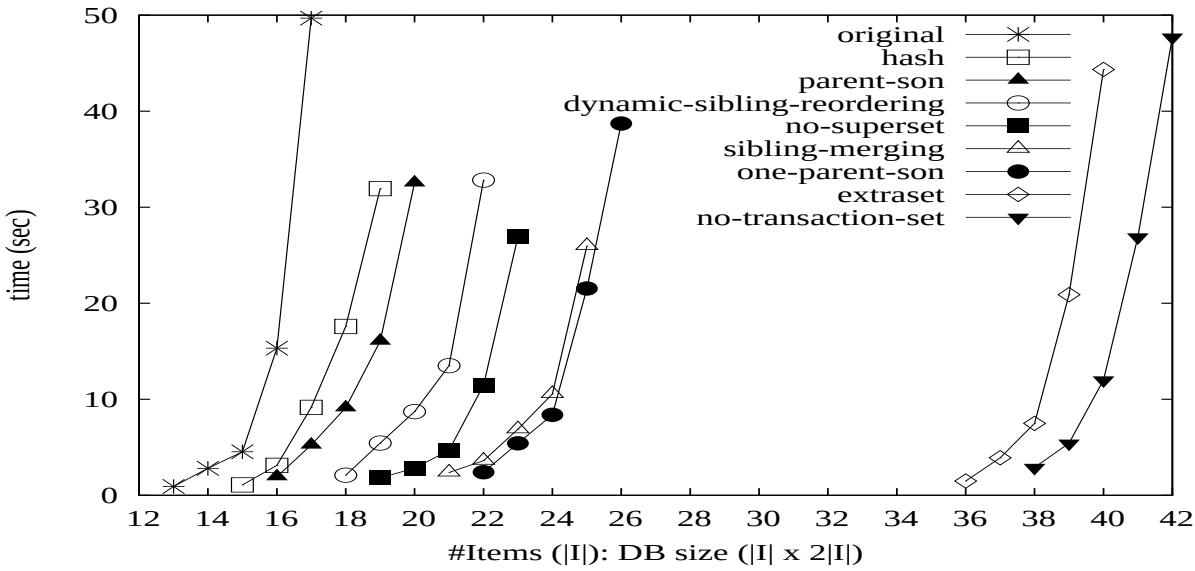

Figure 7: The effects of the speedup optimizations

Effect of Optimizations: We first study the effect of various optimizations proposed in Section 5.1.1 on the performance of BLOSOM-MO, as shown in Figure 7. The x-axis shows the number of items $|\mathcal{I}|$ 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 when making use of the parent-child relationship to do the pruning on the basis of 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 datsset 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.

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. Figure 8 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 column), minimal CNF/CNF expressions (MC/MD – middle column), and closed clauses (CO/CA – right column).

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

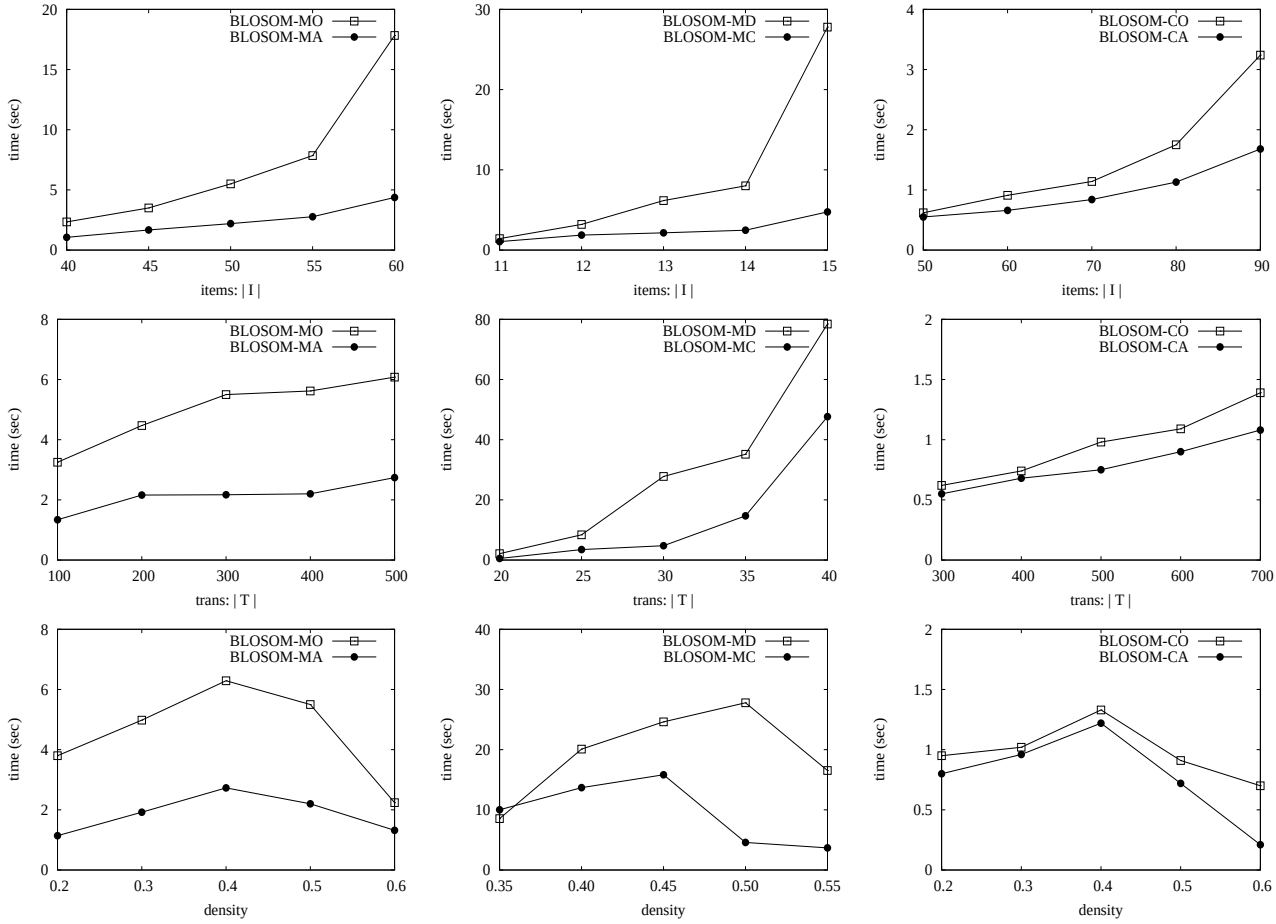

Figure 8: Synthetic Data: Effect of number of items, number of transactions, and density

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.

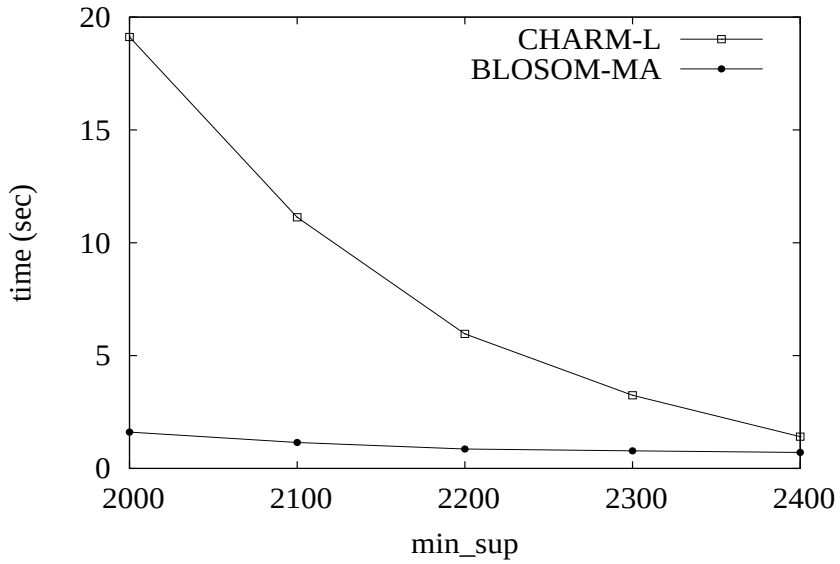

Figure 9: BLOSOM-MA vs. CHARM-L

Comparison: BLOSOM-MA **vs. CHARM-L:** We also compared BLOSOM-MA with CHARM-L [21], which can also mine the minimal generators for AND-clauses (i.e., itemsets). We used the `chess` dataset, from the UCI machine learning repository[1] for the evaluation. From Figure 9 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. It is worth noting that we know of no other algorithms to mine minimal OR-clauses, and closed/minimal CNF and DNF expressions.

6.2 Application Study

We applied our BLOSOM framework to mine several real datasets. We wanted to discover whether a given object set affords alternative boolean descriptions, which are non-trivial. We report some of the interesting patterns found.

House Votes: We analyzed the `voting-records` dataset from the UCI machine learning repository, which has the US congressional voting records from 1984, for 435 congressmen

[1] www.ics.uci.edu/∼mlearn

(267 democrats, 168 republican). Each record measures the Yes or No votes of the congressmen on 16 issues. Taking the Yes or No values as separate attributes for each issue yields 32 attributes for 435 records. One of the patterns we found was that the four representatives who voted in favor of bills on *el-salvador-ad, duty-free-exports, export-administration-act-south-africa*, but against bills on *water-project-cost-sharing*, *budget-resolution* were those exact representatives who voted in favor of bills on *anti-satellite-test-ban* but against the bills on *aid-to-nicaraguan-contras*, *mx-missile*, *synfuels-corporation-cutback*, *superfund-right-to-sue*. All four were republicans.

Senate Voting: We looked at another voting dataset, this time from the US Senate (102th Congress - 1st Session), for 1991. The dataset was obtained from the US Library of Congress THOMAS database[2]. It has the voting record for 101 senators on 280 rolls. Counting both Yes and No votes as separate items, we obtain a dataset with 101 transactions and 560 items. We found that out that a group of 97 senators voting for *H.R.4, Amdt.No.36*, were the exact same senators who agreed to *S.Con.Res.5*, but rejected *Amdt.No.35*. Note that all theses rolls related to the 1991 Iraq War. *H.R.4* allowed some IRS extensions for people involved in operation DesertShield, *Amdt.No.36* disallowed money for rebuilding of Iraq as long as Saddam Hussein remained in power, *S.Con.Res.5* demanded that Iraq abide by the Geneva Convention regarding prisoners of war, and *Amdt.No.36* disallowed the use of US tax-payer dollars for rebuilding Iraq. It is interesting that all 97 senators rejected *Amdt.No.36*.

Gene Expression: We applied BLOSOM to mine frequent boolean expressions as well as redescriptions between descriptors on a gene expression dataset from [17]. This dataset involves 74 genes participating in 824 descriptors, derived from Gene Ontology (GO) categories[3], gene expression bucketing, and k-means clustering. We specifically focus on finding minimal generators to help redescribe the descriptors corresponding to k-means clusters. One potential application is to obtain more expressive functional enrichments of these clusters in terms of GO categories. For instance, a set of six genes participating in a k-means cluster, represented by descriptor $d512$ was approximately redescribed in DNF form as: $d512 \Leftrightarrow d507 \vee d685 \vee d700$, 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.

7 Related Work

Mining frequent itemsets (i.e., pure conjunctions) has been extensively studied within the context of itemset mining [1]. The closure operator for itemsets (AND-clauses) was proposed

[2] www.senate.gov

[3] www.geneontology.org

in [9], and the notion of minimal generators for itemsets was introduced in [3]. Many algorithms for mining closed itemsets (see [10]), and a few to mine minimal generators [3, 21, 8] have also been proposed in the past. The work in [8] focuses on finding the succinct (or essential) minimal generators for itemsets. CHARM-L [21] finds the minimal generators for itemsets (MA). It first finds the closed itemsets and then uses the notion of differential lower shadows to mined the minimal generators. In contrast, BLOSOM-MA directly mines the MAs and runs much faster than CHARM-L. The task of mining closed and minimal monotone DNF expressions was proposed in [19]. 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 structural characterization of the different classes of boolean expressions via the use of closure operators and minimal generators, as well as the efficient, extensible BLOSOM framework for mining arbitrary expressions, are the novel contributions of our work.

Within the association rule context, there has been previous work on mining negative rules [18, 20, 2], as well as disjunctive rules [14]. Unlike these methods we are interested in characterizing such rules within the general framework of boolean expression mining. Also related is the mining of optimal rules according to some constraints [4], 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 [6, 12]. Another point of comparison is w.r.t. the work in [11] 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 [17] mines redescriptions only between length-limited boolean expressions in disjunctive normal form and CHARM-L [21] 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 [5] in the presence of membership queries and equivalence queries. Mitchell [13] 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.

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, we are led to the general challenge of, given a general propositional reasoning framework, mining only the simplest boolean expressions necessary for inference in that framework.

References

[1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In U. Fayyad and et al, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI Press, Menlo Park, CA, 1996.

[2] M.-L. Antonie and O. Zaiane. Mining positive and negative association rules: An approach for confined rules. In *European PKDD Conf*, 2004.

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

[4] R. J. Bayardo and R. Agrawal. Mining the most interesting rules. In *ACM SIGKDD Conf*, 1999.

[5] N. Bshouty. Exact learning boolean functions via the monotone theory. *Information and Computation*, 123(1):146–153, 1995.

[6] T. Calders and B. Goethals. Minimal k-free representations of frequent sets. In *European PKDD Conf.*, 2003.

[7] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

[8] G. Dong, C. Jiang, J. Pei, J. Li, and L. Wong. Mining succinct systems of minimal generators of formal concepts. In *Int'l Conf. Database Systems for Advanced Applications*, 2005.

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

[10] B. Goethals and M. Zaki. Advances in frequent itemset mining implementations: report on FIMI'03. *SIGKDD Explorations*, 6(1):109–117, June 2003.

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

[12] M. Kryszkiewicz. Concise representation of frequent patterns based on disjunction-free generators. In *Int'l Conf. on Data Mining*, 2001.

[13] T. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.

[14] A. Nanavati, K. Chitrapura, S. Joshi, and R. Krishnapuram. Association rule mining: Mining generalised disjunctive association rules. In *ACM CIKM Conf.*, 2001.

[15] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *7th Intl. Conf. on Database Theory*, Jan. 1999.

[16] J. Pfaltz and R. Jamison. Closure systems and their structure. *Information Sciences*, 139:275–286, 2001.

[17] N. Ramakrishnan, D. Kumar, B. Mishra, M. Potts, and R. Helm. Turning cartwheels: An alternating algorithm for mining redescriptions. In *ACM SIGKDD Conf.*, 2004.

[18] A. Savasere, E. Omiecinski, and S. Navathe. Mining for strong negative associations in a large database of customer transactions. In *ICDE Conf*, 1998.

[19] Y. Shima, S. Mitsuishi, K. Hirata, and M. Harao. Extracting minimal and closed monotone dnf formulas. In *Int'l Conf. on Discovery Science*, 2004.

[20] X. Wu, C. Zhang, and S. Zhang. Efficient mining of both positive and negative association rules. *ACM Trans. on Information Systems*, 22(3):381–405, 2004.

[21] M. Zaki and N. Ramakrishnan. Reasoning about sets using redescription mining. In *ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, 2005.