Are Zero-suppressed Binary Decision Diagrams Good for Mining Frequent Patterns in High Dimensional Datasets?

Elsa Loekito and James Bailey

NICTA Victoria Laboratory
Department of Computer Science and Software Engineering
University of Melbourne, Australia
Email: {eloekito, jbailey}@csse.unimelb.edu.au

Abstract

Mining frequent patterns such as frequent itemsets is a core operation in many important data mining tasks, such as in association rule mining. Mining frequent itemsets in high-dimensional datasets is challenging, since the search space is exponential in the number of dimensions and the volume of patterns can be huge. Many of the state-of-the-art techniques rely upon the use of prefix trees (e.g. FP-trees) which allow nodes to be shared among common prefix paths. However, the scalability of such techniques may be limited when handling high dimensional datasets. The purpose of this paper is to analyse the behaviour of mining frequent itemsets when instead of a tree data structure, a canonical directed acyclic graph namely Zero Suppressed Binary Decision Diagram (ZBDD) is used. Due to its compactness and ability to promote node reuse, ZBDD has proven very effective in other areas of computer science, such as boolean SAT solvers. In this paper, we show how ZBDDs can be used to mine frequent itemsets (and their common varieties). We also introduce a weighted variant of ZBDD which allows a more efficient mining algorithm to be developed. We provide an experimental study concentrating on high dimensional biological datasets, and identify indicative situations where a ZBDD technology can be superior over the prefix tree based technique.

Keywords: data mining, association rule mining, frequent patterns, frequent itemset mining, Binary Decision Diagrams (BDDs), Zero-suppressed Binary Decision Diagrams (ZBDDs), high dimensional datasets.

1 Introduction

Mining frequent patterns such as frequent itemsets is a fundamental and well studied problem in data mining. It has a number of useful applications such as association rule mining and market basket data analysis. Frequent itemsets correspond to combinations of items (or attribute values) which occur frequently in the dataset. Thus, mining them in a high dimensional dataset can be challenging, since the search space is exponential in the number of dimensions.

State-of-the-art frequent itemset mining techniques such as those found in FIMI Repository (Goethals 2004) have made attempts to address this issue by making use of prefix tree data structures, or combinations of prefix trees with other data structures, to compress the data representation. Prefix trees, however, limit node sharing to common prefixes which may limit the scalability of a frequent itemset mining algorithm when handling high dimensional datasets. As we will analyse in this paper, sharing of common suffixes, too, can be useful for mining frequent itemsets, which is made possible using a graph data structure called the Zero-suppressed Binary Decision Diagrams (ZBDDs) (Minato 1993).

In particular, microarray datasets are one of the most challenging kinds of high dimensional datasets for pattern mining. They typically consist of only a few number of samples (i.e. rows), but they have very high dimensionality (i.e. thousands of attributes). The number of patterns in a microarray dataset can be enormous (Creighton & Hanash 2003), and hence, mining them requires considerable time as well as space. Other works such as (Rioult et al. 2003, Li et al. 2003) have also studied other itemset mining problems in microarray datasets.

Binary Decision Diagrams (BDDs) (Bryant 1986) are a compact canonical graph representation of boolean formulae. They are a directed acyclic graph (DAG) data representation, similar to binary decision trees, except identical sub-trees are merged. There exist efficient BDD library routines which promote their canonicity and allow intermediate computation results to be reused. Furthermore, Zero-suppressed Binary Decision Diagrams (ZBDDs) are a special type of BDDs which were introduced for efficient manipulation of sparse item combinations (Minato 2001, Minato & Arimura 2005). There exist efficient library routines for manipulating ZBDDs in (Mishchenko 2001), and they have been shown effective for solving problems in other computer science areas such as boolean SAT solvers (Aloul et al. 2002) and solving graph optimization problems (Coudert 1997). There are only a few works that use ZBDDs in a data mining context, such as (Loekito & Bailey 2006, Minato 2005, Minato & Ito 2007, Minato & Arimura 2006).

A vast number of techniques for mining frequent itemsets and their common varieties have been proposed. A survey can be found in (Zaki & Goethals 2003, Zaki et al. 2004). FP-growth* (Grahne & Zhu 2003) is one of the strongest frequent itemset mining algorithms, which is based on the use of prefix trees such as FP (frequent pattern)-trees. FP-growth* follows a pattern growth framework (Han et al. 2004) which recursively creates database projections, and it uses FP-trees to represent the intermediate databases. Other implementations of pattern growth such as AFOPT (Liu et al. 2003) and LCMv3 (Uno et al. 2005) are also based on the use of modified FP-trees, or prefix trees combined with arrays and bitmap data structures.

The purpose of this paper is to analyse the behaviour of frequent itemset mining when canonical DAGs such as ZBDDs, instead of trees, are used as a

primary data structure. Prefix trees have been used as a means for achieving data compression through allowing node sharing of common prefixes. ZBDDs are different from prefix trees in the sense that node fan in as well as fan out is allowed, and multiple identical sub-trees do not exist. More specifically, by sharing the identical sub-trees in a ZBDD, higher data compression can potentially be achieved and efficient ZBDD library can be employed in the mining procedure. Therefore, ZBDDs are seemingly attractive for mining frequent itemsets in sparse, and challenging, high dimensional datasets.

In this paper, we show how ZBDDs can be used to mine frequent itemsets and their maximal and closed variants, concentrating on high dimensional biological datasets. We consider mining in a column-wise pattern growth framework, and a row-wise framework (Rioult et al. 2003, Pan et al. 2004) (introduced for mining closed frequent itemsets). Our objective is to identify and explain situations where ZBDDs are advantageous compared to FP-trees. In particular, we aim to address questions such as:

1. Does the canonical property of ZBDDs allow a scalable and efficient algorithm for frequent itemset mining to be developed ?
2. How much data compression can a ZBDD achieve compared to an FP-tree ?
3. Does the use of a more compact data structure always mean that mining is more efficient?

Our main contributions in this paper are threefold:

- We present an algorithm that can mine (maximal/closed) frequent itemsets, based on the use of a ZBDD as the primary data structure and a supplementary bitmap (similar to (Burdick et al. 2001)) for support checking. A particularly attractive feature of our technique is the use of multiple shared-ZBDDs to represent the input database, the intermediate databases, as well as the final output, allowing them to share common sub-trees. This is something which is not possible in prefix-tree-based techniques like (Grahne & Zhu 2003, Liu et al. 2003, Pietracaprina & Zandolin 2003). We also show how the ZBDD mining framework can be adapted to the row-wise mining approach (Pan et al. 2003, 2004).
- We introduce an edge-weighted variant of ZBDDs, whose structure is similar to Edge-Valued Binary Decision Diagrams (Vrudhula et al. 1996) which were proposed for efficient representation of discrete functions. Our weighted ZBDDs allow itemsets and their corresponding frequencies to be compactly represented. Hence, support counting can be performed more efficiently compared to when the bitmap is used. Moreover, their canonical property allows a more efficient mining technique to be developed, which is achieved through re-using intermediate results. It is advantageous especially for mining large and dense datasets, in which bitmap manipulations may be costly, and a significant portion of the intermediate computations may share results.
- We experimentally investigate the behaviour of our techniques, according to various characteristics of high dimensional biological datasets. Our techniques are compared against the state-of-the-art FP-tree-based technique FP-growth* (Grahne & Zhu 2003). Our results show a number of situations where the use of a ZBDD (either weighted or not) is able to give improvement over FP-growth*.

2 Preliminaries

In this section we provide background knowledge of the pattern growth framework, which is employed by the existing prefix-tree based mining algorithms, and an overview of Zero-suppressed Binary Decision Diagrams. Firstly though, we define some terminologies which will be used in the remainder of this paper.

Assume we have a dataset D defined upon a set of k attributes. For every attribute A_i, $i \in \{1, 2..k\}$, the domain of its values (or items) is denoted by $dom(A_i)$. Let $\mathcal{I}$ be the aggregate of the domains items across all the attributes, i.e. $\mathcal{I} = \bigcup_{i=1}^{k} dom(A_i)$. An itemset is a subset of $\mathcal{I}$. Let p and q be two itemsets. We say p contains q if q is a subset of p, i.e. $q \subseteq p$. A dataset is a collection of transactions, where each transaction is an itemset. The support of an itemset p in dataset $\mathcal{D}$, i.e. $support(p)$, is the fraction of the transactions in $\mathcal{D}$ which contain p ($0 \leq support(p) \leq 1$). Given a dataset $\mathcal{D}$, and a support threshold α, an itemset p is **frequent** if it satisfies the constraint: $support(p) \geq \alpha$. Furthermore, p is a **maximal frequent itemset** if p is not contained in any other frequent itemset. p is a **closed frequent itemset** if p is not contained in any other frequent itemset which has the same support.

2.1 Pattern growth framework for mining frequent itemsets

The pattern growth framework for mining frequent patterns grows prefixes by recursively projecting intermediate databases. For each item x, an *x-conditional DB* is induced. It contains itemsets in the input database which contain x. Then, frequent itemsets which contain prefix $\{x\}$ are grown by recursively creating further projections from the x-conditional DB. FP-growth is one of the strongest techniques that follows this pattern growth approach. In particular, FP-growth (Han et al. 2004) uses frequent pattern (FP) trees for storing the input, the output patterns, and the conditional DBs. An FP-tree is created afresh for each of those databases.

FP-growth uses a dynamic ordering of the items, such that items in an FP-tree are ordered by decreasing frequency, the most frequent item at the top. The FP-trees which are created throughout mining may use different item orderings. One of the most efficient implementations of FP-growth grows prefixes by traversing the database in a bottom-up manner (There exist its variations which use an inverse item ordering and perform a top-down traversal, such as that in (Liu et al. 2003)).

An FP-tree example is shown in Figure 1(a). Each node in an FP-tree contains an item and a value which represents the frequency of that node's prefix path. As a secondary data structure, a header table is used for storing the total frequency of each item. In this example, the first prefix is grown using item d (the least frequent item). The FP-tree representation for d-conditional DB is shown in Figure 1(b).

FP-growth* (Grahne & Zhu 2003) is an optimised implementation of FP-growth which reduces the number of database projections based on a technique called *bi-level projection*, and minimises memory usage of the algorithm by using a *pseudo-projection* instead of physically creating the conditional databases. For further implementation details, readers are referred to that paper.

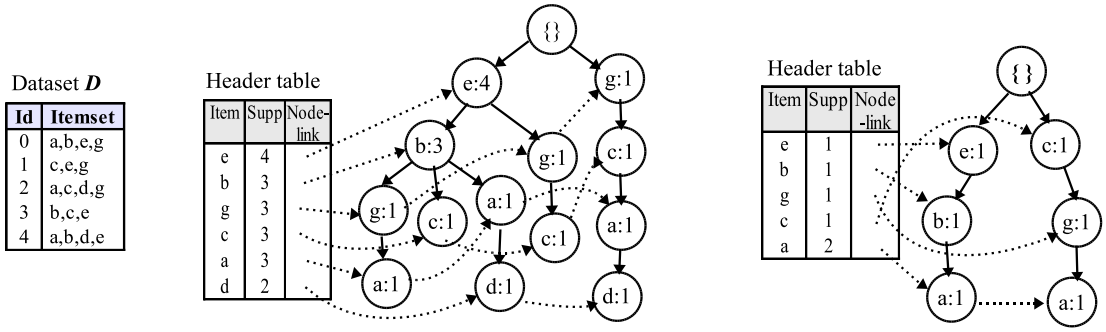


Figure 1: FP-tree representations for sample dataset D

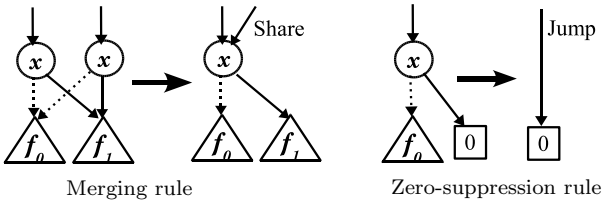


Figure 2: ZBDD Reduction Rules

2.2 Zero-suppressed Binary Decision Diagram(ZBDD):

Binary Decision Diagrams (BDDs) (Bryant 1986) are canonical directed acyclic graphs (DAGs). Their canonical property allows boolean formulae to be compactly represented and their logical operations (AND, OR, XOR, etc.) to be performed in polynomial time with respect to the number of nodes. A Zero-suppressed BDD (ZBDD) (Minato 1993) is a special type of BDD which was introduced for efficient manipulation of sparse combinations. However, it has received little attention in data mining except a few works in (Loekito & Bailey 2006, Minato 2005, Minato & Ito 2007, Minato & Arimura 2006). A survey on other, non data-mining, ZBDD applications can be found in (Minato 2001).

More formally, a BDD is a canonical DAG of labeled nodes. It consists of one source node, multiple internal nodes, and two *sink* nodes which are labeled as 0 and 1 respectively. Each internal node may have multiple parent nodes, but it can only have two child nodes, which we call 0-*child* and 1-*child* nodes. By canonical, it means that multiple identical nodes are not allowed. Two nodes are identical if they have the same label, and their respective child nodes are also identical. The nodes are ordered so that the label of any internal node must be of higher index (i.e. appear earlier in the variable ordering) than the label of its children.

An internal node N with a label x, denoted $N = node(x, N_1, N_0)$, encodes the boolean formula $N = (x \wedge N_1) \vee (\overline{x} \wedge N_0)$. We call N_1 (resp. N_0) the 1-child (resp. 0-child) of N. The edge connecting a node to its 1-child (resp. 0-child) is also called the *true*-edge (resp. *false*-edge). In the illustrations shown shortly, solid lines correspond to true-edges whilst dotted lines correspond to false-edges. Each path from the root to sink-1 (resp. sink-0) gives a true (resp. false) assignment for the represented function. For some node N, with label x, the outgoing true-edge of N represents a true assignment for variable x, whereas the outgoing false-edge represents a false assignment for variable x.

BDDs have two important properties (Bryant 1986):

1. Equivalent subtrees are shared (*canonical*);
2. Computation results are stored for future reuse (referred as BDD's *caching principle*).

These properties make the worst-case complexity of most BDD operations polynomial with respect to the number of nodes. The caching principle allows the result from intermediate computations to be reused if the same computation is re-visited in the future. This utility is particularly effective if many subtrees are being shared within the input BDD, as the same subtree may be encountered multiple times (under the same computation) through out its manipulation.

A Zero-suppressed BDD (ZBDD) is a special kind of BDD which was introduced for efficient combinatorial itemset analysis (Minato 1993, Minato & Arimura 2005). More specifically, a ZBDD is a BDD with two reduction rules (Their illustrations are shown in Figure 2):

1. **Merging rule**: merge identical subtrees (to obtain canonicity);
2. **Zero-suppression rule**: delete nodes whose 1-child is the sink-0, and replace them with their 0-child.

By utilising these rules, a sparse collection of item combinations, which can be seen as a boolean formula, can be represented with high compression, i.e. for an n variable formula, the possible space of truth values is 2^n, however the corresponding BDD/ZBDD can have exponentially fewer nodes.

We follow the ZBDD encodings for representing a collection of itemsets as in (Minato & Arimura 2005). An itemset p can be represented by a n-bit binary vector $(x_1, x_2, \ldots, x_n)$, where $x_i = 1$ if item i is contained in p. A set $\mathbf{S}$ of itemsets can be represented by a characteristic function $\mathcal{X}_S : \{0,1\}^n \rightarrow \{0,1\}$ where $\mathcal{X}_S(p) = 1$ if $p \in \mathbf{S}$ and 0 otherwise. In ZBDD semantics, set $\mathbf{S}$ such that $\mathbf{S} = \mathbf{S_0} \cup (\mathbf{S_1} \times \{x\})$ can be represented by a ZBDD node $N = (x, N_1, N_0)$ where $\mathbf{S_1}$ (resp. $\mathbf{S_0}$) is the set of itemsets encoded by N_1 (resp. N_0). An itemset p in S is interpreted as a conjunction of the items contained in p and yields a true assignment for the boolean formula encoded by N. A sink-0 node encodes an empty set ($\emptyset$), and sink-1 node encodes a set of an empty itemset ($\{\emptyset\}$). For clarity, sink-0 nodes may be omitted from the illustrations shown in this paper. Table 1 lists some pre-defined ZBDD library operations (Minato & Arimura 2005, Mishchenko 2001) which we use in our algorithms.

Variable Ordering: The number of nodes in a ZBDD, and thus, the efficiency of its manipulations may be highly sensitive to its variable ordering. Work in (Minato 2001) shows that a good variable ordering for compact BDDs should have two properties:

Table 1: Primitive ZBDD operations

sink-0	The empty set, $\emptyset$
sink-1	The set of an empty itemset, $\{\emptyset\}$
getNode(x, N_1, N_0)	Create $node(x, N_1, N_0)$ and apply ZBDD reduction rules
$P \bigcup_{max} Q$	Maximal set-union between itemsets in P and Q
$P \setminus Q$	Itemsets of P which do not exist in Q
CrossProd(P, Q)	Pair-wise intersection between itemsets in P and itemsets in Q

E.g.: 1. $P = \{\{a, b, d\}, \{b, c\}\}$, $Q = \{\{b, c, d\}, \{a, c, d\}\}$, $P \bigcup_{max} Q = \{\{a, b, d\}, \{b, c, d\}, \{a, c, d\}\}$
2. $P = \{\{b, c, d\}, \{a, b, d\}\}$, $Q = \{\{a, b, d\}, \{a, c\}\}$, $P \setminus Q = \{\{b, c, d\}\}$
3. $P = \{\{a, d\}, \{b, c\}\}$, $Q = \{\{b, d\}, \{a, b\}\}$, CrossProd$(P, Q) = \{\{a, b, d\}, \{b, c, d\}, \{a, b, c, d\}\}$

1) groups of closely related variables should be kept near to each other; 2) variables that greatly affect the function should be located at higher positions. For our purpose, we use some heuristics described shortly, based on the frequency of the variables in the input dataset, that aims to maximise sharing of sub-structures across the auxiliary ZBDDs and allow efficient mining.

3 Frequent Itemset Mining Based on the Use of ZBDDs

In this section we present our ZBDD-based mining techniques for mining frequent itemsets, and mining their maximal and closed variants.

As a general overview, our techniques adopt the pattern growth framework, but instead of using FP-trees, we use ZBDDs as a primary data structure, and ZBDD library routines are used to compute the conditional DBs. More specifically, we use multiple-shared ZBDDs, which means canonicity is maintained (i.e. node sharing is allowed) across the multiple databases. Unlike in FP-trees, support information is not stored inside the nodes, instead, we use a secondary data structure *bitmap* for support counting.

The core operations in our framework, such as creating database projections and maintaining the output patterns, employ efficient ZBDD library functions which cache computation results. This means, intermediate redundant computations can be avoided if the same database (or substructure of the database) is re-visited through out mining. Due to the recursive nature of the pattern growth algorithm, multiple conditional databases are likely to contain many similar sub-structures. In particular, for a given prefix, its conditional DB contains subsets of itemsets from the conditional DB projected by some subset of that prefix.

Variable Ordering: Items in the ZBDDs in our mining technique are ordered by increasing frequency. This ordering has the following three objectives: 1) to obtain smaller conditional DBs in the first recursions, 2) to allow early pruning of infrequent prefixes, 3) to increase node-sharing across the databases. This ordering allows a very high data compression to be achieved, and increases the effectiveness of ZBDD's caching principle during database projections. Figure 3(a) shows the ZBDD which represents the input dataset $\mathcal{D}$ given in Figure 1. In this example, the ZBDD contains only 10 nodes (excluding sink nodes), whereas the FP-tree contains 14 nodes.

3.1 Frequent Itemset (FI) Miner

Our algorithm for mining frequent itemsets is labeled as **FIMiner**. The algorithm is shown in Algorithm 1. It is invoked by calling FIMiner(Z_D, α, $prefix$) where Z_D contains the input itemsets and $prefix$ is initially empty (i.e. $prefix = \{\}$). Frequent itemsets are grown from the given prefix, using the input ZBDD which is traversed in a top-down fashion. Let x be the top-node's label. For a given input ZBDD, item x is firstly used to grow the current prefix since no computation is needed to create the conditional DB for this item. More specifically, an x-conditional DB can be found in the top 1-child node, which contains all itemsets containing x in Z_D. This routine finds the patterns which contain x. Patterns which do not contain x are grown later, after removing x from the input Z_D and applying the same mining procedure. The output ZBDD is incrementally built from each recursion step, using the same variable ordering as the input ZBDD which allows both ZBDDs to share nodes.

In addition to the standard ZBDD primitive operations, we push the anti-monotonic support constraint deep inside the routine using an *infrequent prefix pruning* strategy (line 4-5) which is based on the anti-monotonic property of *support*. Here, the *bitmap* data representation of the input database is needed to calculate the support of each prefix. For a given prefix P, bitmap(P) refers to the bit-vector of the occurrence of P in the initial input dataset. We compute support(P) by counting the number of 1's in bitmap(P), denoted as |bitmap(P)|. Hence, the support of $prefix \cup \{x\}$ in FIMiner (line 4) can be computed incrementally by re-using bitmap($prefix$) which has been computed in previous mining iteration, and taking its bit-wise intersection with bitmap($\{x\}$).

When a new prefix which is grown using item x is infrequent, it is deleted from the output by returning the sink-0 node and employing ZBDD's zero-suppression rule (line 5). This automatically deletes node x from the output and replaces it by other patterns which do not contain item x, which will be computed later in line 9. Otherwise, the new prefix can be grown further using x-conditional DB (line 7). To remove x from the remaining routines and grow prefixes using the remaining items, the two child-nodes of Z_D are merged using a set-union operation, which is a ZBDD primitive operation (line 9). For mining efficiency and data compression purposes, the non-maximal itemsets are removed from the merged database, which can be obtained by using ZBDD's $\bigcup_{max}$ operation. We refer to this operation as *DB-merging*. Since primitive ZBDD operations store computation results in a cache, DB-merging can be computed efficiently if many of the conditional DBs share common subtrees. Finally, the recursion terminates when the induced database is empty (line 1-2).

Once mining in both the x-conditional database and the merged database have been completed, the output node is created by having item x as its label and the patterns found from the two sub-tasks are assigned to its 1-child and 0-child respectively. Since, this procedure is performed at each recursion level, it incrementally builds the output ZBDD in a bottom-up fashion. More specifically, the frequent patterns which are found from the x-conditional database be-

Algorithm 1 FIMiner(Z_D,α, $prefix$)

Input: Z_D: the database induced by $prefix$
α: minimum support threshold ,
$prefix$: prefix itemset
Output: Z_{FI}: the frequent itemsets in Z_D
Procedure:
1: **if** (Z_D is a sink node) **then**
2: *Terminal case:*
return Z_D
3: **end if**
Let $Z_D = node(x, Z_{D_x}, Z_{D_{\overline{x}}})$,
$prefix_x = prefix \cup \{x\}$
4: **if** (support($prefix_x$) $< \alpha$) **then**
5: *Infrequent prefix pruning*:
$Z_{FI_x} = 0$
6: **else**
7: *Grow new prefix* $prefix_x$ *and mine FIs*:
Z_{FI_x} = FIMiner(Z_{D_x}, α, $prefix_x$)
8: **end if**
9: *DB-merging and grow prefix without* x:
$Z_{FI_{\overline{x}}}$ = FIMiner($Z_{D_{\overline{x}}} \bigcup_{max} Z_{D_x}$, α, $prefix$)
10: **return** Z_{FI} = getNode($x, Z_{FI_x}, Z_{FI_{\overline{x}}}$)

Note:
support($prefix_x$) = |bitmap($prefix$) $\cap$ bitmap($\{x\}$)|.

come the 1-child node, which are seen as patterns which contain item x, and the frequent patterns which are found in the merged database become the 0-child. When creating such a node, ZBDD's library checks whether the node has been existed. If it has, then the existing node is shared.

Let us consider again the sample dataset in Figure 1. Figure 3(a) shows the conditional DB for the first item, i.e. *d*-conditional DB which is given by the 1-child of the top-node. Figure 3(b) illustrates the *DB-merging* operation. The merged database contains the set-union between itemsets in the two child-nodes of Z_D. In order to achieve higher data compression and mining efficiency, the non-maximal itemsets are simultaneously removed while computing the set-union. Identical nodes in the merged ZBDD are shared with the input database, as well as the other databases (nodes which are newly created for the merged ZBDD are drawn with a bold outline).

3.2 Maximal Frequent Itemset (MFI) Miner

We now describe some optimisations that can be applied to our FI mining technique for mining maximal frequent itemsets (MFIs). We call the algorithm **MFI-Miner**. Using the same core operations as FI-Miner, MFI-Miner has an additional procedure for removing the non-maximal patterns. This is performed using a progressive focusing technique (Burdick et al. 2001), which removes the locally non-maximal patterns from each conditional DB. It can be computed using a ZBDD primitive set-subtraction routine i.e. $Z_{FI_{\overline{x}}} \setminus Z_{FI_x}$, where Z_{FI_x} and $Z_{FI_{\overline{x}}}$ are computed as in Algorithm 1. This subtraction operation removes the frequent extensions of $prefix$ which also occur as frequent extensions of $prefix_x$ since they are non-maximal local to the current database.

Additionally, our framework can adopt some advanced pruning techniques such as those used in (Burdick et al. 2001, Grahne & Zhu 2003, Wang et al. 2003). For this purpose, an itemset *tail* is maintained for each conditional DB. It contains the items that occur in the relevant database, and ZBDD library functions are employed for manipulating each database with its *tail*. For instance, to remove infrequent items from a database D, *crossProd* can be employed upon the ZBDDs of D and $freqTail$, where $freqTail$ is obtained from *tail* by removing the infrequent items. Since $freqTail$ is a single itemset, then the *crossProd* operation returns the intersection between each itemset in D with $freqTail$.

3.3 Closed Frequent Itemset (CFI)-Miner

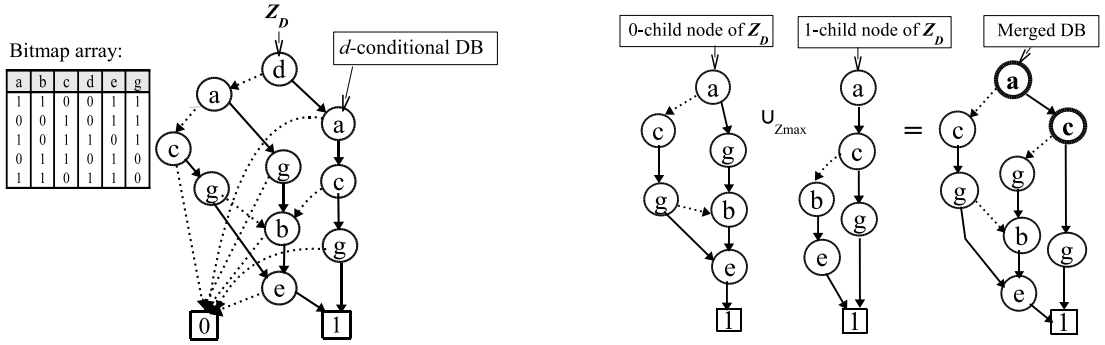
Our algorithm for mining closed frequent itemsets namely **CFI-Miner** has a similar framework as MFI-Miner, which uses a progressive focusing technique for removing the non-closed patterns. However, the closed constraint requires the support of an itemset to be compared against its subset(s). Thus, the support information has to be represented in the output data structure, which was not necessary for FI/MFI-Miner. Note that the support information does not need to be represented in the input ZBDD as it may reduce its compactness. Additionally, CFI-Miner can also adopt the more advanced pruning techniques found in existing algorithms, using a similar mechanism to MFI-Miner which maintains a *tail* itemset.

To represent the patterns' support in the ZBDD output, additional variables are used, which are appended to each pattern. We refer to these extended pattern representations as *item-support-sets*. In order to achieve higher compression, we use the binary representation of the support values. For a database of n transactions, $log_2(n)$ binary variables are reserved. Suppose the database containing 5 transactions, 3 support-encoding variables are reserved. Let r_0 r_1 r_2 be the support-encoding binary variables, such that $r_2 = 0$, $r_1 = 0$, $r_0 = 1$ represents a support of 1, $r_2 = 0$, $r_1 = 1$, $r_0 = 0$ represents 2, etc. For instance, the *item-support-set* representation of itemsets (with their corresponding frequencies) {*be:3, abe:2, ab:2*} is $\{ber_1r_0, aber_1, abr_1\}$. Furthermore, itemsets in the maximal item-support-sets correspond to closed itemsets. Item-support-set abr_1 is not maximal since $aber_1$ exists. However, $aber_1$ is maximal, and *abe* is a closed itemset.

4 Weighted Zero-suppressed Binary Decision Diagrams

Support counting using bitmap in our above-described techniques can be costly, especially in dense high dimensional datasets which contain long patterns. To eliminate this overhead, we introduce a weighted variant of ZBDD, namely **Weighted Zero-suppressed Binary Decision Diagram (WZDD)** which allows the support values to be represented using edge-weights and in turn allows a more efficient frequent itemset mining. There exist other weighted types of Decision Diagrams (Bryant & Chen 1995, Vrudhula et al. 1996, Ossowski & Baier 2006) for manipulating pseudo-boolean functions, but the semantics are different.

In a WZDD, every edge is attributed by a positive integer value. Formally, we define an internal WZDD node as a pair of $\langle \varphi, \vartheta \rangle$, where φ is the total weight of this node's outgoing edges, ϑ is a ZBDD node. The edge-weights are anti-monotonic, with their values descreasing as the nodes are positioned lower in the structure. The weight on its incoming edge corresponds to the total support of itemsets being represented(see Figure 4(a)). WZDDs are also canonical, that is, nodes which contain the same set of itemsets with the same corresponding supports are merged. Consequently, ZBDD's set-union routine needs to be adapted for WZDDs to add the supports of any itemset which occurs in both of its operands. Figure 4(b) shows an example of a WZDD which represents itemsets (with their corresponding support) : {*ace:1, abe:2, ab:2*}. The weight on node *a*'s incoming

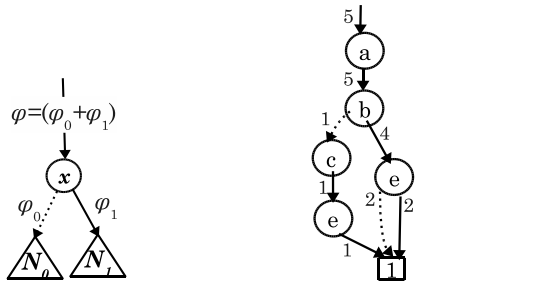


Var. ordering: $d < a < c < g < b < e$

(a) Z_D = ZBDD representation for dataset $\mathcal{D}$, d-conditional DB is Z_D's 1-child node

(b) DB merging (nodes marked with bold lines are the newly created nodes)

Figure 3: ZBDD representations for sample dataset $\mathcal{D}$



WZDD node

(a)

Var. ordering: $a < b < c < d < e$

WZDD representation for $\{abe : 2, ace : 2, ab : 1\}$

(b)

Figure 4: Weighted ZBDDs

link is 5, i.e. the total support of the itemsets. The two node e's are not shared since their edges have different weights, respectively.

Although WZDDs may use more nodes than the ZBDDs, due to their weighted-edges, they allow further use of the caching utility for mining frequent itemsets. More specifically, WZDDs allow more efficient mining algorithm to be developed by caching the computed patterns from each conditional DB. In WZDDs, any two identical nodes contain the same set of itemsets and their corresponding support, thus, the two nodes also contain the same set of frequent patterns. Suppose different prefixes project the same conditional DB, patterns from the earlier computation can be reused and redundant computation can be avoided.

5 Row-wise Mining of Closed Frequent Itemsets using ZBDDs

Let us now describe how our ZBDD-based mining framework can be adopted to the row-wise mining framework for finding CFIs which have been introduced in (Pan et al. 2003, Liu et al. 2006, Pan et al. 2004). In this framework, the patterns are mined by searching for possible row (instead of item) combinations. We can employ the column-wise mining framework described earlier, except now the ZBDD variables correspond to row-IDs (instead of items). Before we describe our technique, we firstly provide some background and terminology.

Let R be the set of row-ids. A *rowset* is a subset of R. For a given rowset X, $row\_support\_set(X)$ is the maximal itemset which occurs in all the rows in X. Moreover, for a given item x, we call the set of row-ids in which x occurs as x's *bit_support*. By definition, every *row_support_set* is a closed itemset. Works in (Pan et al. 2003, Liu et al. 2006) proposed a row-wise mining framework which is efficient for mining of CFIs in microarray datasets. It performs a depth-first search in the lattice of row-id (instead of item) combinations. We will describe a pattern growth, bottom-up algorithm (Pan et al. 2003) based on the use of ZBDD, although adapting it to the top-down algorithm (Liu et al. 2006) is certainly possible.

Our row-wise mining technique adopts the algorithm in (Pan et al. 2004), using a similar approach to FIMiner for creating database projections and performing a top-down traversal of the input ZBDD. Here, the input ZBDD is used to generate possible row combinations. A supplementary bitmap data structure is used accordingly for computing the *row_support_set* for each row combination. Unlike in our column-wise mining algorithms, there is no sharing between the input and the output ZBDDs in a row-wise mining since now they contain different sets of variables. To maximise node sharing within the output ZBDD which stores the closed frequent itemsets, we use the same variable ordering as in our column-wise algorithm i.e. by increasing frequency. The mining procedure is summarised as the following. Due to lack of space, we omit the implementation details of our algorithm.

The input ZBDD initially contains the *bit_support* of each item in the input dataset. Rowset prefixes are grown using the first variable (i.e. row id) in the ZBDD, which has a similar mechanism as growing itemset prefixes in our column-wise framework, while the minimum support threshold has not yet been reached by the rowset prefix or while the *row_support_set* is not empty. More specifically, as each rowset is being grown, its *row_support_set* is incrementally computed from the bitmap data representation. When the length of the prefix reaches the minimum support, its *row_support_set* is a fully-grown CFI, and it is inserted into the output ZBDD using ZBDD's set-union operation.

6 Performance Study

In this section, we analyse the performance of our ZBDD-based techniques for mining maximal frequent itemsets (MFIs) and closed frequent itemsets (CFIs). The algorithms were implemented in C++ using the BDD library, JINC, which was developed by the author of (Ossowski & Baier 2006) and used in their

study of another weighted variant of BDD. All experiments were performed on four 4.0 GHz CPUs, 32 GB RAM, running RedHat Linux 5, with a CPU-timeout limit of 100,000 seconds per mining task. We used two gene-expression datasets: Leukaemia ALL-AML [1], and lung cancer[2] which was previously studied in (Pan et al. 2004). The ALL-AML dataset contains 72 samples (i.e. transactions), each sample is described by 7129 genes (i.e. attributes). The lung cancer dataset contains 32 samples, described by 12533 genes.

Continuous attribute values are discretised using an entropy discretisation method, then, the discretised attributes are ordered according to their entropy values from highest to lowest (i.e. attr.1 has highest entropy reduction value). Entropy-based discretisation is the commonly used method for discretising microarray datasets (Creighton & Hanash 2003). To obtain results for low support threshold values, we performed our experiments using the first 100 attributes from ALL-AML dataset (All methods could not complete mining at low support thresholds when all of the attributes were used due to the CPU time out constraint). For a similar reason, we used the first 750 attributes from the lung cancer dataset. In the following discussions, we refer to the respective datasets as ALLAML-100 and lung-cancer-750.

We performed experiments for mining CFIs and MFIs in both datasets, but we do not include the results from mining MFIs in the ALLAML-100 dataset in this paper due to the similar output characteristics between the MFIs and CFIs. The pattern characteristics from both datasets will be shown shortly. We did some experiments for mining CFIs in the row-wise mining framework, but we do not include the results in this paper. Overall, our ZBDD-based method outperforms the FP-tree based method (Pan et al. 2004) when mining at low support thresholds.

In the item-wise framework, we implemented our techniques, i.e. *ZBDDMiner*, *WZDDMiner*, which are our ZBDD and WZDD based algorithms, using only the basic infrequent prefix pruning, and *ZBDDMiner** and *WZDDMiner** which use the more advanced pruning techniques. Their performance is compared against the state-of-the-art FP-tree based algorithms for mining CFIs and MFIs, i.e. *FP-close** and *FP-max** (Grahne & Zhu 2003)[3], which performed best on dense datasets. From each algorithm, we measure (i) the CPU time which is the total time spent for mining, (ii) the size of the output FP-trees, ZBDDs, or WZDDs which store the mined patterns in terms of the number of nodes, and (iii) the total nodes usage which is the total number of nodes used through out mining, which include the data structures for storing the input database, the intermediate databases, and the final output patterns. In ZBDDs or WZDDs, shared nodes are counted only once. When WZDDs are used for mining MFIs, storing support values in the output is not necessary, thus, they are removed from the output WZDDs

6.1 Patterns Distribution

In the ALLAML-100 dataset, the length distribution of closed frequent itemsets at support threshold 40% is shown in Figure 5(a). It shows that there are millions of relatively long patterns (the longest pattern contains 32 items). In lung cancer dataset, the length distribution of the closed frequent itemsets given support threshold of 40% is shown in Figure 5(b), and the length distribution of the maximal frequent itemsets is shown in Figure 5(c). Both output characteristics show there are only a relatively small number of patterns in this dataset, compared to the patterns contained in ALLAML-100 dataset. Thus, we categorise ALLAML-100 dataset as a dense dataset, and lung cancer-750 as a sparse dataset.

6.2 Mining CFI in a Sparse Dataset

Firstly, let us observe the performance comparison between our ZBDD-based and WZDD-based algorithms for mining CFIs in lung cancer-750 dataset against FP-close*. Our WZDD-based algorithms could not complete mining for support threshold $< 40\%$, whilst the ZBDD-based algorithms could not complete mining for support threshold $< 50\%$, due to the CPU timeout constraint.

Even though this dataset is relatively sparse, in Figure 6(a) it is shown that the ZBDD representations (with support encoding variables) for storing the output patterns are smaller than the FP-trees for support threshold $< 60\%$, with the weighted ZBDDs being the most compact. Figure 6(b) shows that FP-close* has the fastest running times, and the WZDD based algorithms are faster than the ZBDD based algorithms. This is expected from this dataset due to its sparse characteristics, in which there is less likely that many nodes are shared across the conditional databases. Now let us look closer at the performance of our algorithms which use advanced pruning strategies, i.e. ZBDDMiner* and WZDDMiner*. It shows that ZBDDMiner* improves the efficiency of ZBDDMiner, but WZDDMiner* does not improve WZDDMiner except for the low support threshold of 36%. When the support threshold is low, many conditional databases are being induced, and WZDDMiner* benefits from its ability to re-use intermediate pruning computations.

The total nodes usage of all algorithms is shown in Figure 6(c). It shows that FP-close* uses the least number of FP-tree nodes for representing all the databases (including the input, the output patterns, and the intermediate structures). However, the discrepancies with our ZBDD and WZDD based algorithms are decreasing as the support threshold decreases, with the WZDDs having a similar nodes usage to FP-trees at support threshold of 37%.

Furthermore, ZBDDMiner* uses about 5 times fewer nodes than ZBDDMiner, which shows the effectiveness of the advanced pruning strategies for reducing the size of the intermediate structures, and in turn results in a more efficient mining of ZBDDMiner* over ZBDDMiner. On the other hand, WZDDMiner* has a slightly more nodes usage than WZDDMiner. This shows that there are more nodes being shared between the input database and the conditional databases when no advanced pruning is used. Given high support threshold, moreover, there are not many patterns in this sparse dataset, hence, not much nodes are shared across the conditional databases.

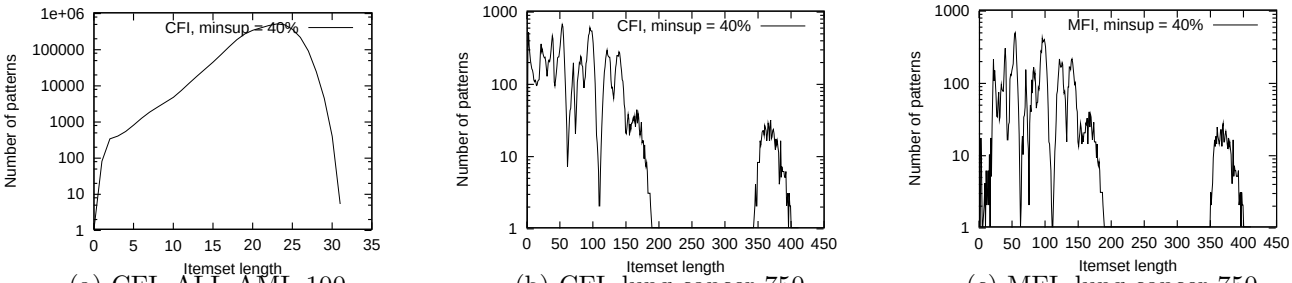
6.3 Mining CFI in a Dense Dataset

When mining CFIs in ALLAML-100 dataset, ZBDDMiner could not complete within the CPU time limit for support threshold $< 50\%$, and FP-close* exceeds the memory limits for support threshold as low as 27%. The WZDDs for storing the output patterns are significantly smaller than FP-trees, as shown in Figure 7(a), which demonstrates the ability of WZDDs to compactly represent huge volume of patterns. The ZBDD (with additional support-encoding variables) representations are about 100 times larger than WZDDs.

[1] http://research.i2r.a-star.edu.sg

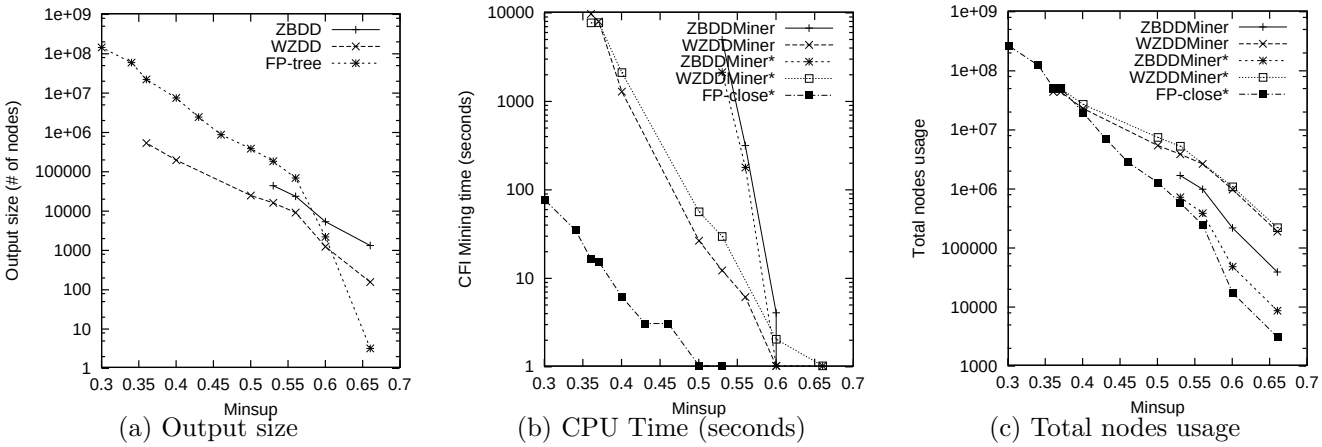
[2] http://www-genome.wi.mit.edu/cgi-bin/cancer

[3] FP-close* is a variant of FP-growth* for mining CFIs, FP-max* is a variant of FP-growth* for mining MFIs. Their implementation can be found in (Goethals 2004)



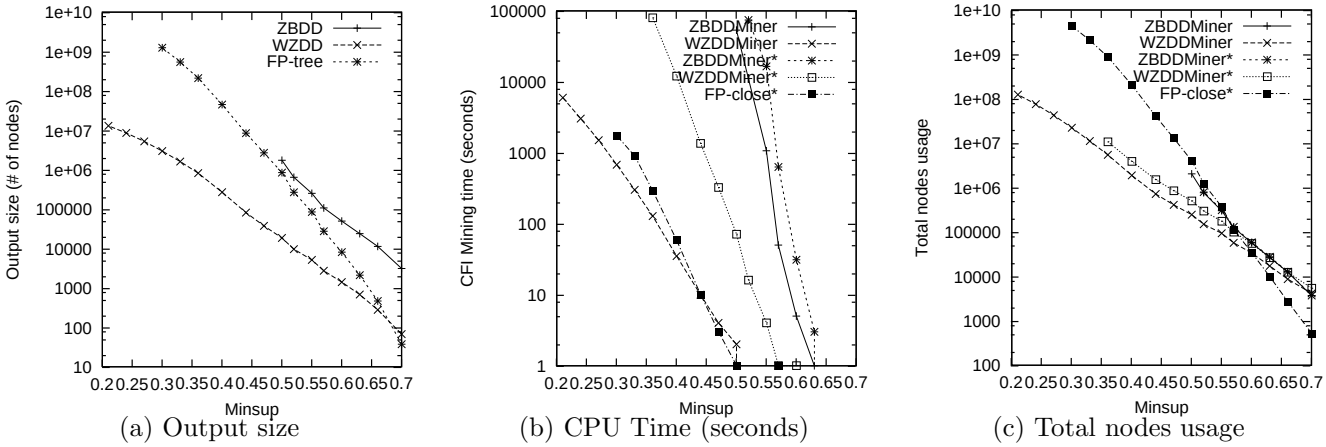
(a) CFI, ALL-AML-100 (b) CFI, lung-cancer-750 (c) MFI, lung-cancer-750

Figure 5: Patterns Length Distribution



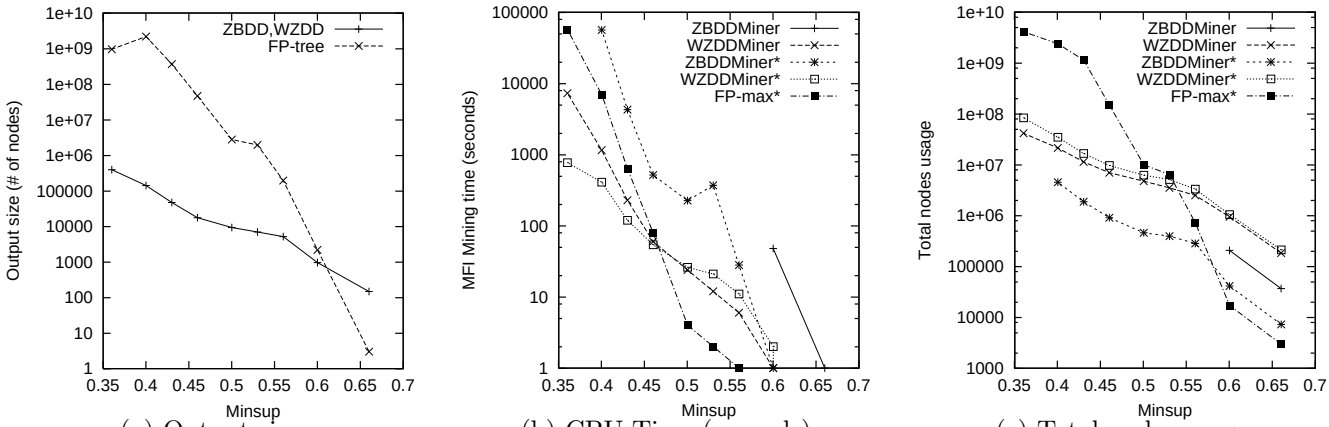
(a) Output size (b) CPU Time (seconds) (c) Total nodes usage

Figure 6: Results from mining CFIs in lung-cancer-750 dataset



(a) Output size (b) CPU Time (seconds) (c) Total nodes usage

Figure 7: Results from mining CFIs in ALL-AML-100 dataset



(a) Output size (b) CPU Time (seconds) (c) Total nodes usage

Figure 8: Results from mining MFIs in lung-cancer-750 dataset

Figure 7(b) shows the runtime comparison between the algorithms. WZDDMiner has the fastest runtime for low support thresholds , i.e. $< 40\%$. When advanced pruning strategies are used, WZDDMiner* is up to 500 times slower than WZDDMiner, but on the other hand, ZBDDMiner* can achieve up to 100 times speedup factor over ZBDDMiner.

Furthermore, Figure 7(c) shows WZDDMiner has the least total nodes usage compared to all the other algorithms except for support threshold $\geq 60\%$ for which the FP-trees have the least total nodes usage.

6.4 Mining MFI in a Sparse Dataset

When mining MFIs in this sparse dataset, Figure 8(a) shows that the ZBDD representations for storing the output patterns are smaller than the FP-trees for support threshold $< 60\%$, being up to 1000 times smaller at support threshold of 40%. This shows the significant data compression being achieved by ZBDDs over FP-trees.

Figure 8(b) shows the runtime comparison between our algorithms against FP-max*. It shows that for high support threshold, i.e. $\geq 50\%$, FP-max* is the fastest, but WZDDMiner* is the fastest for lower support threshold. Unlike in mining CFIs, WZDDMiner* does improve the mining efficiency of WZDDMiner for low support threshold, which indicates that in this dataset, re-using the pruning computations from multiple conditional databases in WZDDMiner* is beneficial.

Although ZBDDMiner* is slower, but it uses the least number of nodes through out mining for support threshold $< 60\%$, which is shown in Figure 8(c). More specifically, ZBDDMiner* could achieve about 50 times data compression over both WZDDMiner and WZDDMiner*, FP-max* uses up to 100-1000 times more nodes than both WZDDMiners and ZBDDMiners when support threshold is low, i.e. $< 50\%$.

7 ZBDD VS FP-tree

In this section we discuss some advantages and disadvantages of using ZBDDs (and their weighted variants) in an FI mining framework, based on our experimental results, with respect to a number of dimensions: the compactness of ZBDD data structure, the effectiveness of pruning in a ZBDD-based technique, and the effectiveness of ZBDD's caching utility.

7.1 ZBDD's canonicity

The canonical structure of a ZBDD is a a powerful feature which we have shown useful for not only compressing high dimensional output patterns, but also for compressing the intermediate structures used for mining them, by allowing the various databases to share nodes. This compression has been proven in our experimental results when mining MFIs and CFIs at relatively low support threshold, for which the FP-trees had billions of overall nodes usage. In particular, in the dense dataset, WZDDs are able to achieve up to 500 times overall data compression over FP-trees, as shown in the total nodes usage comparison. In such a circumstance, many long patterns exist and many sub-trees may be shared across multiple databases (including the conditional DBs) which is not possible using FP-trees. On the other hand, in the sparse dataset in which not too many conditional DBs are being projected, ZBDDs allow a higher overall data compression than WZDDs, being able to share more nodes when representing the intermediate databases but they require the use of *bitmap* for support counting as a tradeoff.

Furthermore, when ZBDDs are used for computing CFIs and the output data structure contains additional support-encoding variables, we found that although the size of the ZBDDs is increased, they can still contain fewer nodes than FP-trees when the support threshold is relatively low.

7.2 Pruning effectiveness

Our techniques use a basic infrequent prefix pruning as well as some advanced pruning strategies. We will discuss the effectiveness of each type of pruning shortly. Firstly, let us consider the infrequent prefix pruning. FP-trees may allow earlier pruning of infrequent prefixes by allowing different item orderings to be used for different database projections. On the other hand, ZBDDs and WZDDs use static item ordering as a tradeoff for achieving data compression. This explains a number of situations in our experimental results when ZBDDs and WZDDs are less efficient than FP-trees, when mining CFIs in the sparse dataset or mining MFIs/CFIs in the other dataset given a high support threshold, in which a large search space can be pruned earlier by the FP-trees.

Secondly, the effectiveness of advanced pruning strategies rely upon the amount of search space reduction over the overhead of performing the pruning routines. As part of the advanced pruning routines, ZBDDMiner* uses a bit-wise-and operation for support counting, whereas WZDDMiner* has to traverse the database which is an expensive computation to perform in high-dimensional datasets. Based on our experiments, ZBDDMiner* does improve mining efficiency of ZBDDMiner when the total size of the conditional DBs is significantly reduced, such as when mining MFIs/CFIs in the sparse dataset.

WZDDMiner*, on the other hand, although it has a higher computation overhead for performing the advanced pruning, it is able to use ZBDD's caching utility for remembering the set of frequent items in each database. It is therefore beneficial when there is a large number of nodes being shared across the various conditional databases, as shown in our experiments when mining MFIs/CFIs at low support threshold in the sparse dataset. In other circumstances, WZDDMiner* is less efficient than WZDDMiner. For instance when the support threshold is relatively high, only a few database projections are required (and the databases are less likely to share many nodes), for which ZBDD's ability to re-use the intermediate pruning computations is not beneficial. Also, in the dense dataset, since a large volume of patterns exist, the effects of pruning do not greatly reduce the size of the databases, thus, the overhead of performing advanced pruning in WZDDMiner* is not compensated.

7.3 Cached computation results

Another powerful feature of ZBDDs is their caching principle. The ZBDD routines used in our framework allow the result from intermediate computations, to be cached. More particularly in the WZDD-based framework, the output patterns from every conditional DB are cached and it has been proven useful to allow more efficient mining. In dense datasets, or in sparse datasets with low support threshold, millions of long patterns exist and different prefix itemsets may project similar conditional DBs (and thus, similar patterns). This has proven a significant time improvement by WZDDs over FP-trees, despite the use of static item ordering which may hurt its efficiency as we have discussed earlier. In particular, our experimental results show the WZDD-based technique (without advanced pruning) outperforms the FP-tree based technique (with advanced pruning)

when mining huge volume of MFIs (indicated by the number of nodes in the output data structures) at low support threshold, and it has similar time performance as the FP-tree based technique for mining CFIs in the relatively denser dataset.

Summary of Performance: We now return to the questions which were posed at the beginning of the paper:

1. *Does the canonical property of ZBDDs allow an efficient and scalable algorithm for frequent itemset mining to be developed ?*

   As we have seen in our experimental results, the WZDD based algorithm is superior over both ZBDDs and FP-trees for mining MFIs at low support threshold, or mining CFIs in a dense dataset. In such a circumstance, millions of auxiliary conditional DBs are induced, as indicated by the volume of patterns. Their canonical WZDD representations (which are substantially smaller than FP-trees) allow the computed patterns and other intermediate results to be reused, which in turn allow mining to be performed efficiently.

2. *How much data compression can a ZBDD achieve compared to an FP-tree ?*

   (a) When mining CFIs, the total size of non-weighted ZBDDs used throughout mining may be larger than the FP-trees, this being a result of having the extra support encoding variables on the output. They, however, could achieve slightly higher overall data compression by a factor of 2 compared to FP-trees, as shown in our results when mining CFIs at low support in dense datasets.

   More specifically, for storing the output CFIs, ZBDDs with support-encoding variables increase the size of the traditional ZBDDs (without support-encoding variables) by 100 times, yet, they may contain fewer nodes than FP-trees when mining CFIs at low support threshold.

   (b) When mining MFIs, non-weighted ZBDDs are able to achieve further overall data compression up to 100 times more compact than WZDDs, as found in our experiments. This shows WZDDs achieve lower data compression than ZBDDs due to their weighted edges. However, WZDDs are still much more compact than FP-trees for representing all of the databases created throughout mining, more particularly if the support threshold is not too high, as shown in our results that they used up to 1000 times fewer nodes than the FP-trees

3. *Does the use of a more compact data structure really mean that mining is more efficient?*

   Not always. When compared to FP-trees, there are situations where the intermediate ZBDDs or WZDDs are highly compressed and the total nodes usage is much less, for which intermediate results can be reused effectively and mining speedups are increased over the FP-tree based technique. More particularly, such situations were found in our experimental results when mining huge volume of MFIs/CFIs. On the other hand, if the ZBDD/WZDD data representations were of similar size than FP-trees, the FP-tree based techniques are more efficient as they use a more flexible variable ordering which allows earlier search space pruning.

   Moreover, if we compare ZBDDs against the weighted ZBDDs, although the ZBDD representations in the intermediate computations are often smaller than the WZDDs, based on our findings, it does not always mean a more efficient mining was obtained. This can be explained by the extra cost of using a secondary data bitmap with (non-weighted) ZBDDs, which may require expensive computation in high dimensional datasets.

8 Related Work

We are aware of several other works which use ZBDDs for itemset mining (Minato & Arimura 2006, Minato & Ito 2007). Work in (Minato & Ito 2007) demonstrated that ZBDDs are useful for mining patterns in high dimensional amino acid datasets.

Work in (Minato & Arimura 2006) proposed a ZBDD-based pattern growth mining of frequent itemsets, which uses a different data representation schema to that used in our proposed technique. Its optimised variant for mining closed frequent itemsets is proposed in (Minato & Arimura 2007). Their ZBDD representation of the databases encode the support values by storing the itemsets in multiple ZBDD functions based on their binary support values, whose representations are referred as *tuple histograms*. Their experiments show their technique outperforms FP-growth (Han et al. 2004) for mining huge volume of patterns in the traditional, low dimensional, type of datasets. Such a representation is less compact and requires more complex routines to construct, instead of the simpler, and faster, basic ZBDD routines used in our framework. Therefore, their technique does not seem scalable for mining the more challenging microarray datasets which have exponentially large search space. Furthermore, we have shown our technique can be adapted to the row-wise mining framework.

A more recent work in (Iwasaki et al. 2007) proposed a method for choosing a good variable ordering for ZBDDs in data mining applications. The method computes the variable ordering after the ZBDD has been built, and re-arranges the nodes. This method is different to ours which decides the variable ordering prior to constructing the ZBDDs. Our variable ordering heuristics aim to achieve an efficient mining of frequent itemsets, as well as to achieve an overall compact data representation across multiple intermediate data structures used throughout mining, whereas their method finds a variable ordering which is optimised for a particular ZBDD.

Other tree data structures have been proposed in other frequent itemset mining techniques (Zaki et al. 2004, Liu et al. 2003, Pietracaprina & Zandolin 2003). None of them, however, allows node sharing across the auxiliary databases, which is a key feature of our technique. AFOPT (Liu et al. 2003) is a prefix-tree structure which is designed to work for a top-down traversal in projecting the conditional DBs similar to our traversal strategy. However, it cannot achieve much data compression. Patricia trie (Pietracaprina & Zandolin 2003) combines prefix trees with array-list to achieve a more compressed structure but it does not allow sharing among multiple databases, and it does not yet have optimisations for mining maximal/closed frequent itemsets.

Work in (Loekito & Bailey 2006) demonstrated that ZBDDs can be used to efficiently mine contrast patterns, which include two support constraints on two respective datasets, one being anti-monotonic and the other monotonic. Their technique also uses a supplementary bitmap data representation for sup-

port counting in a similar mechanism to the ZBDD-based techniques proposed in this paper.

9 Conclusions and Future Work

In this paper, we have examined the use of advanced data structures, ZBDDs, for mining (maximal/closed) frequent itemsets, and identified situations where they are superior over state of the art FP-tree-based technique. Overall, we found that ZBDD allows much higher data compression for storing huge volume of long patterns as well as the intermediate structures used in mining them. We also introduced a weighted type of ZBDD, which is able to improve mining efficiency than the classic ZBDD. Although our ZBDD-based framework is not uniformly superior than FP-trees, it can sometimes allow more efficient mining in relatively dense high dimensional datasets at low support thresholds. We believe this result suggests that ZBDDs can be a very valuable tool in data mining. We would like to further investigate their use in other scenarios, considering more complex constraints and other types of patterns.

Acknowledgement

We would like to thank Jian Pei, one of the authors of the FP-growth technique, for his useful comments on the compactness of a ZBDD compared to an FP-tree, and the efficiency of the mining algorithm between those two database representations.

We also would like to thank Joern Ossowski for providing us the JINC package and for his helpful comments on the implementation of our WZDD data structure which relies on JINC's core library.

This work is partially supported by National ICT Australia. National ICT Australia is funded by the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

References

Aloul, F. A., Mneimneh, M. N. & Sakallah, K. (2002), ZBDD-based backtrack search SAT solver, *in* 'International Workshop on Logic Synthesis', University of Michigan.

Bryant, R. E. (1986), 'Graph-based algorithms for boolean function manipulation', *IEEE Transactions on Computers* **35**(8), 677–691.

Bryant, R. E. & Chen, Y.-A. (1995), Verification of arithmetic circuits with Binary Moment Diagrams, *in* 'DAC'95: Proceedings of the 32nd ACM/IEEE Conference on Design Automation', pp. 535–541.

Burdick, D., Calimlim, M. & Gehrke, J. (2001), MAFIA: A maximal frequent itemset algorithm for transactional databases, *in* 'International Conference on Data Engineering (ICDE'01)', pp. 443–452.

Coudert, O. (1997), 'Solving graph optimization problems with ZBDDs', *In Design, Automation and Test in Europe* pp. 224–228.

Creighton, C. & Hanash, S. (2003), 'Mining gene expression databases for association rules', *Bioinformatics* **19**(1), 79–86.

Goethals, B. (2004), 'Frequent itemset mining implementations (FIMI) repository'.
URL: *http://fimi.cs.helsinki.fi/*

Grahne, G. & Zhu, J. (2003), Efficiently using prefix-trees in mining frequent itemsets, *in* 'Proceedings of the 1st IEEE ICDM FIMI'03 Workshop on Frequent Itemset Mining Implementations'.

Han, J., Pei, J., Yin, Y. & Mao, R. (2004), 'Mining frequent patterns without candidate generation: An FP-Tree approach', *Data Mining and Knowledge Discovery* **8**(1), 53–87.

Iwasaki, H., Minato, S. & Zeugmann, T. (2007), A method of variable ordering for Zero-suppressed Binary Decision Diagrams in data mining applications, *in* 'Proceedings of the 3rd IEEE International Workshop on Databases for Next-Generation Researchers, SWOD 2007', pp. 85–90.

Li, J., Liu, H., Downing, J. R., Yeoh, A. & Wong, L. (2003), 'Simple rules underlying gene expression profiles of more than six subtypes of Acute Lymphoblastic Leukemia (ALL) patients', *Bioinformatics* **19**, 71–78.

Liu, G., Lu, H., Yu, J. X., Wei, W. & Xiao, X. (2003), AFOPT: An efficient implementation of pattern growth approach, *in* 'Proceedings of the 1st IEEE ICDM FIMI'03 Workshop on Frequent Itemset Mining Implementations'.

Liu, H., Han, J., Xin, D. & Shao, Z. (2006), Top-down mining of interesting patterns from very high dimensional data, *in* 'Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)', p. 114.

Loekito, E. & Bailey, J. (2006), Fast mining of high dimensional expressive contrast patterns using ZBDDs, *in* 'Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'06)', pp. 307–316.

Minato, S. (1993), Zero-suppressed BDDs for set manipulation in combinatorial problems, *in* 'Proceedings of the 30th International Conference on Design Automation', pp. 272–277.

Minato, S. (2001), 'Zero-suppressed BDDs and their applications', *International Journal on Software Tools for Technology Transfer (STTT)* **3**(2), 156–170.

Minato, S. (2005), Finding simple disjoint decompositions in frequent itemset data using Zero-suppressed BDD, *in* 'Proceedings of IEEE ICDM'05 Workshop on Computational Intelligence in Data Mining', pp. 3–11.

Minato, S. & Arimura, H. (2005), Combinatorial item set analysis based on Zero-suppressed BDDs, *in* 'IEEE Workshop on WIRI 2005', pp. 3–10.

Minato, S. & Arimura, H. (2006), Frequent pattern mining and knowledge indexing based on Zero-suppressed BDDs, *in* 'The 5th International Workshop on Knowledge Discovery in Inductive Databases (KDID'06)', pp. 83–94.

Minato, S. & Arimura, H. (2007), 'Frequent closed item set mining based on Zero-suppressed BDDs', *Transaction of the Japanese Society of Artificial Intelligence* **22**(2), 165–172.

Minato, S. & Ito, K. (2007), 'Symmetric item set mining method using Zero-suppressed BDDs and application to biological data', *Transaction of the Japanese Society of Artificial Intelligence* **22**(2), 156–164.

Mishchenko, A. (2001), 'An introduction to Zero-suppressed Binary Decision Diagrams'.
URL: *http://www.ee.pdx.edu/ãlanmi/research.htm*

Ossowski, J. & Baier, C. (2006), Symbolic reasoning with weighted and normalized decision diagrams, *in* 'Proceedings of the 12th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning', pp. 35–96.

Pan, F., Cong, G., Tung, A. K. H., Yang, J. & Zaki, M. (2003), CARPENTER: Finding closed patterns in long biological datasets, *in* 'Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)', pp. 637–642.

Pan, F., Cong, G. & Tung, A. K. H.and Tan, K. (2004), Mining frequent closed patterns in microarray data, *in* 'Proceedings of the 2nd IEEE ICDM FIMI'04 Workshop on Frequent Itemset Mining Implementations', pp. 363–366.

Pietracaprina, A. & Zandolin, D. (2003), Mining frequent itemsets using patricia tries, *in* 'Proceedings of the 1st IEEE ICDM FIMI'03 Workshop on Frequent Itemset Mining Implementations'.

Rioult, F., Boulicaut, J., Crémilleux, D. & Besson, J. (2003), Using transposition for pattern discovery from microarray data., *in* 'Proceedings of 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'03)', pp. 73–79.

Uno, T., Kiyomi, M. & Arimura, H. (2005), LCM ver. 3: Collaboration of array, bitmap and prefix tree for frequent itemset mining, *in* 'Proceedings of ACM SIGKDD Open Source Data Mining Workshop on Frequent Pattern Mining Implementations (OSDM'05)', pp. 77–85.

Vrudhula, S., Pedram, M. & Lai, Y. (1996), 'Edge-valued binary decision diagram', *Representation of Discrete Functions* pp. 109–132.

Wang, J., Han, J. & Pei, J. (2003), CLOSET+: Searching for the best strategies for mining frequent closed itemsets, *in* 'Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)', pp. 236–245.

Zaki, M., Goethals, B. & Bayardo, R., eds (2004), *Proceedings of the 2nd IEEE ICDM FIMI'04 Workshop on Frequent Itemset Mining Implementations*, Vol. 126 of *CEUR Workshop Proceedings*.

Zaki, M. & Goethals, B., eds (2003), *Proceedings of the 1st IEEE ICDM FIMI'03 Workshop on Frequent Itemset Mining Implementations*, Vol. 80 of *CEUR Workshop Proceedings*.