Fast Mining of High Dimensional Expressive Contrast Patterns Using Zero-Suppressed Binary Decision Diagrams

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

ABSTRACT

Patterns of contrast are a very important way of comparing multi-dimensional datasets. Such patterns are able to capture regions of high difference between two classes of data, and are useful for human experts and the construction of classifiers. However, mining such patterns is particularly challenging when the number of dimensions is large. This paper describes a new technique for mining several varieties of contrast pattern, based on the use of Zero-Suppressed Binary Decision Diagrams (ZBDDs), a powerful data structure for manipulating sparse data. We study the mining of both simple contrast patterns, such as emerging patterns, and more novel and complex contrasts, which we call disjunctive emerging patterns. A performance study demonstrates our ZBDD technique is highly scalable, substantially improves on state of the art mining for emerging patterns and can be effective for discovering complex contrasts from datasets with thousands of attributes.

Categories and Subject Descriptors: H.2.8 [Database Management]: Database Applications-Data Mining

General Terms: Algorithms, Design, Performance

Keywords: Contrast Patterns, Disjunctive Emerging Patterns, Zero-Suppressed Binary Decision Diagrams

1. INTRODUCTION

The discovery of distinguishing characteristics and contrasts between classes of data is an important objective in data mining. Such patterns are very useful for human experts and can also be used to build powerful classifiers [18, 20]. In this paper, we propose a new technique for mining contrast patterns in high dimensional space. It is able to mine both simple contrasts, such as emerging patterns [8] and also more complex types of contrasts, whose descriptions allow disjunction, as well as conjunction.

A novel feature of our contrast mining technique is that it is based on the use of Zero-Suppressed Binary Decision Diagrams (ZBDDs) [23] as the core data structure. Binary decision diagrams [6] are a graph based data structure which allow efficient representation and manipulation of boolean formulae, and they have proved extremely effective in diverse fields of computer science, such as SAT solvers [7, 2], VLSI and reliability [29]. ZBDDs are an important variation of binary decision diagrams and are particularly appropriate for compactly representing sparse data.

Challenges: A key focus of our study is the mining of contrasts for high dimensional data, such as gene expression datasets, where the number of dimensions can be in the thousands and the search space is huge. Previous techniques for mining contrasts in such datasets, e.g. [12, 9, 18], have been unable to handle more than about 60 dimensions. Another challenge arises when contrast patterns are allowed to be expressed using disjunction, as well as conjunction. This means the pattern search space is considerably larger and therefore, mining becomes even more challenging.

Contributions: We make several important contributions in the paper.

- We show ZBDDs can be employed as a feasible tool for mining contrast patterns, by supplementing them with bit-vectors for support checking and by pushing the support constraints inside the ZBDD manipulation routines. This provides an interesting alternative to popular structures such as the frequent pattern tree [14], whose variants have previously been proposed as an effective contrast mining method [3, 12]. Furthermore, our approach is quite general, in the sense that it is adaptable to a range of other mining objectives.
- We present an algorithm that uses ZBDDs to mine a well-known, simple type of contrast pattern, known as the emerging pattern [8]. Experimental evaluation shows this technique achieves very large speedups over a state of the art emerging pattern miner, based on pattern trees [12].
- We investigate more complex contrast patterns which generalise emerging patterns, by allowing disjunction as well as conjunction. We call these patterns *disjunctive emerging patterns*. We establish the formal characteristics of such patterns, show that our ZBDD mining technique can be adapted to this more complex scenario, and provide experimental evidence that it can be practically feasible for mining very high dimensional datasets. We are not aware of any other work which is suitable for mining this kind of contrast pattern.

KDD'06, August 20–23, 2006, Philadelphia, Pennsylvania, USA.

Table 1: Example Dataset

Positive Class			Negative Class		
A_1 $\{a,b,c\}$	A_2 $\{d,e,f\}$	A_3 $\{g,h,i\}$	A_1 $\{a,b,c\}$	A_2 $\{d,e,f\}$	A_3 $\{g,h,i\}$
a	e	g	a	f	g
a	d	i	b	d	h
b	f	h	b	f	h
c	e	h	c	e	g

Organisation: An outline of the remainder of this paper is as follows. Some basic definitions and terminology are given in Section 2. The ZBDD method for mining emerging patterns is described in Section 3. We show in Section 4 how the mining technique generalises to more complex types of emerging patterns, which we call disjunctive emerging patterns. This is followed by a performance analysis in Section 5, a discussion in Section 6 and a description of related work in Section 7.

2. PRELIMINARIES

Assume we have a dataset D defined upon a set of k attributes (also referred as dimensions) $\{A_1, A_2, \ldots, A_k\}$. Assume a partition of D into two sets, D_p (the positive class) and D_n (the negative class). These are the classes that will be contrasted. For every attribute A_i, the domain of its values (or items) is denoted by $dom(A_i)$. We require domains to be discrete, but they may or may not be ordered. Let $|A_i|$ denote the number of elements in $dom(A_i)$. Let I be the aggregate of the domains across all the attributes, i.e. $I = \bigcup_{i=1}^{k} dom(A_i)$. An *itemset* is a subset of I. Let p and q be two itemsets. We say p *contains* q if q is a subset of p, i.e. $q \subseteq p$. The complement of an itemset p, $\overline{p}$, is the itemset $(I - p)$. A *dataset* is a collection of transactions, where each transaction T is an itemset and we require T to contain exactly one value from the domain of each attribute. The support of an itemset p in dataset D, $support(p, D)$, is the fraction of the transactions in D which contain p ($0 \leq support(p, D) \leq 1$). We next recall the definition of emerging patterns [8] [1], a special type of contrast patterns.

Definition 1. Given a positive dataset D_p, a negative dataset D_n, and support thresholds α and β. An **Emerging Pattern (EP)** is an itemset p satisfying two support constraints, i) $support(p, D_n) \leq \beta$ and ii) $support(p, D_p) \geq \alpha$. Furthermore, p is a **minimal EP** if p does not contain any other itemset that satisfies constraints i-ii.

e.g. Consider Table 1 and suppose $\alpha = 0.25$ and $\beta = 0$. The minimal emerging patterns include $\{a, e\}, \{i\}, \{c, h\}$.

Emerging patterns have proved to be very useful for building accurate classifiers, as well as providing intuitive descriptions of sharp differences between classes of data [10]. They have also been used for bioinformatics applications, such as understanding leukaemia [18]. Indeed, their popularity is evidenced by the fact that over 50 papers have so far been published in the area. Techniques for finding emerging patterns can be found in [9, 12, 4, 3].

3. MINING EMERGING PATTERNS USING ZERO-SUPPRESSED BDDS

In this section, we will describe our approach for mining emerging patterns using Zero-suppressed Binary Decision Diagrams (ZBDDs). Firstly, we need to present some background material.

[1] In [8], emerging patterns were defined using an α threshold and a minimum growth rate ρ. We use α and β thresholds instead, believing it to be more intuitive.

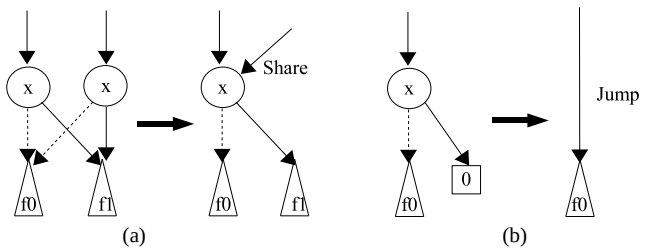


Figure 1: (a)Merging rule; (b)Zero-suppression rule

3.1 Binary Decision Diagrams

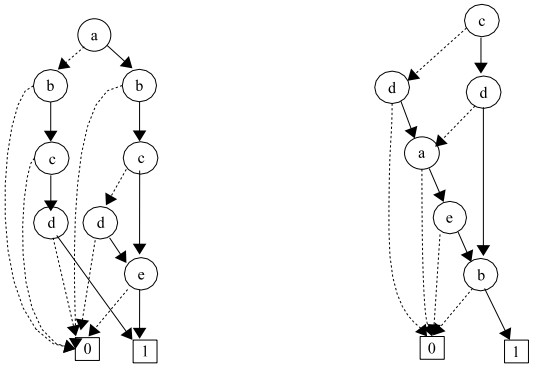
Binary Decision Diagrams (BDDs) are canonical directed acyclic graphs which are efficient representations of boolean formulae, and they allow logical operations (AND, OR, XOR, etc.) to be performed in polynomial time with respect to the number of nodes. A Zero-suppressed BDD (ZBDD) is a special type of BDD, introduced by Minato in [23] for set-manipulation in combinatorial problems. In particular, this structure has been shown to be very efficient for manipulating sets of sparse combinations. ZBDDs are popular in boolean satisfiability solvers [7, 2] and in the field of reliability engineering for fault-tree analysis [29]. However, they have received very little attention in data mining (to be discussed more in Section 7). A survey on ZBDD applications can be found in [24].

More formally, a BDD is a canonical directed acyclic graph consisting of one source node, multiple internal nodes, and two sink nodes *sink*-0 and *sink*-1. Nodes in a BDD are labelled, and they are ordered. An internal node N with a label x, denoted $N = node(x, N_1, N_0)$, encodes the boolean formula $N = (x \wedge N_1) \vee (\overline{x} \wedge N_0)$. N_1 (resp. N_0) is called the 1-child (resp. 0-child) of N. The edge connecting a node to its 1-child (resp. 0-child) is called the *true-edge* (*false-edge*). In the illustrations shown shortly, the solid lines correspond to true-edges and dotted lines correspond to false-edges. Each path from the root node to sink-1 (resp. sink-0) gives a true (resp. false) assignment for the boolean formula.

Two important properties of a BDD which account for the efficiency of its operations include: 1. identical subtrees are shared, 2. intermediate results from past computations are stored and can be recalled as needed. Moreover, most BDD operations have a polynomial worst-case complexity with respect to the number of nodes.

A ZBDD is a special type of BDD for set combinatorial problems which employs two reduction rules (see Fig.1): 1. **Merging rule**: equivalent subtrees are shared (to obtain canonicity); 2. **Zero-suppression rule**: nodes whose true-edge points to These rules allow a high compression of boolean formulae, i.e. for an n-variable formula, the space of possible truth values is 2^n, the corresponding (Z)BDD can have exponentially fewer nodes.

We follow the ZBDD encodings for representing a collection of itemsets using a strategy similar to work in [23]. An itemset p can be represented by a n-bit binary vector $\mathcal{X} = (x_1, x_2, \ldots, x_n)$, 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 $p \in S$ and 0 otherwise. In ZBDD semantics, a node $N = (x, N_1, N_0)$ represents a set S of itemsets such that $S = S_0 \cup (S_1 \times \{x\})$, where S_1 and S_0 are the sets of itemsets encoded by N_1 and N_0, respectively. 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 ZBDD consisting of only the sink-0 node encodes the empty set ($\emptyset$), and a ZBDD consisting of only the sink-1 node encodes



(a) Variable ordering: $a < b < c < d < e$

(b) Variable ordering: $c < d < a < e < b$

Figure 2: ZBDD representations of a set of itemsets $\{\{a, b, c, e\}, \{a, b, d, e\}, \{b, c, d\}\}$

Table 2: Primitive operations on ZBDDs P and Q

0	The empty set, $\emptyset$
1	The set of an empty itemset, $\{\emptyset\}$
getNode(x, N_1, N_0)	Creates $node(x, N_1, N_0)$ and applies the ZBDD reduction rules
change(P, x)	Invert all occurrences of item x in P
$P \bigcap_Z Q$	Set-intersection of P and Q
$P \bigcup_Z Q$	Set-union of P and Q
$P \bigcup_{Z_{min}} Q$	Minimal (w.r.t inclusion) itemsets of $P \bigcup_Z Q$
$P \bigcup_{Z_{max}} Q$	Maximal (w.r.t inclusion) itemsets of $P \bigcup_Z Q$
$P \setminus Q$	Subtraction of any itemset in Q from P
NotSupSet(P, Q)	Subtraction from P of any itemset which is a superset of an itemset in Q
CrossProd(P, Q)	Pair-wise intersection of the itemsets in P and Q
DotProd(P, Q)	Pair-wise union of the itemsets in P and Q

Examples:
1. $\{\{a, b\}, \{b\}\} \bigcap_Z \{\{a, b\}, \{b, d\}\}$ = $\{\{a, b\}\}$
2. $\{\{a, b\}, \{b\}\} \bigcup_Z \{\{b, d\}\}$ = $\{\{a, b\}, \{b\}, \{b, d\}\}$
3. $\{\{a, b\}, \{b\}\} \bigcup_{Z_{min}} \{\{b, d\}\}$ = $\{\{b\}\}$
4. $\{\{a, b\}, \{b\}\} \bigcup_{Z_{max}} \{\{b, d\}\}$ = $\{\{a, b\}, \{b, d\}\}$
5. CrossProd$(\{\{a, b\}, \{a, d\}\}, \{\{b, d\}\})$ = $\{\{b\}, \{d\}\}$
6. DotProd$(\{\{a, b\}, \{a, c\}\}, \{\{b, d\}\})$ = $\{\{a, b, d\}, \{a, b, c, d\}\}$

the set of empty itemsets ($\{\emptyset\}$). Basic set operations for ZBDDs which will be used in our algorithm include *set-union ($A \bigcup_Z B$), set-difference ($A \setminus B$), and set-intersection ($A \bigcap_Z B$)*. They have been defined in [23, 26] and are polynomial in the number of nodes in the ZBDD. They are listed in Table 2.

Example 1. The possible ZBDD encodings for set $\{\{a, b, c, e\}, \{a, b, d, e\}, \{b, c, d\}\}$ are shown in Fig.2. Fig.2(a) follows a lexicographic ordering, whilst Fig.2(b): $c < d < a < e < b$. In Fig.2(a), itemsets $\{a, b, d, e\}, \{a, b, c, e\}$ share a common prefix $\{a, b\}$ and a common suffix $\{e\}$. In Fig.2(b), itemsets $\{d, a, e, b\}, \{c, a, e, b\}$ share a common suffix $\{a, e, b\}$, and the suffix $\{b\}$ is shared among all the itemsets. This set can also be expressed as a DNF formula: $(a \wedge b \wedge c \wedge e) \vee (a \wedge b \wedge d \wedge e) \vee (b \wedge c \wedge d)$

Variable Ordering: Depending on the function being represented, the number of nodes in a ZBDD may be highly sensitive to its variable ordering. Figure 2 illustrates the different compression that can be achieved by using different variable orderings. The ZBDD in Figure 2(b) contains only 6 non-sink nodes as opposed to the ZBDD using lexicographic ordering which contains 8 nodes. Work in [24] shows that a good variable ordering for compact BDDs (and ZBDDs) has two properties: i. groups of inputs that are closely related should be kept near to each other; ii. inputs that greatly affect the function should be located at higher positions in the structure.

A number of works have investigated various variable orderings. One approach is based on heuristics and find the appropriate ordering before the BDD is constructed [13, 1, 32]. Another approach decides an ordering initially, and allows the variables to be permuted during the construction of the BDD [31]. The latter approach is usually more effective than the former but it may be longer to compute. In this paper, we employ heuristics which are based on the frequency of the variables in the input dataset.

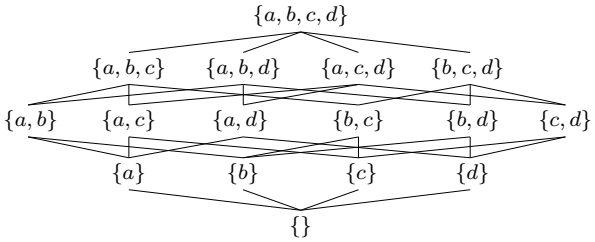


Figure 3: Example of pattern lattice for $I = \{a, b, c, d\}$
A bottom-up enumeration begins with the empty set $\{\}$ and generates the longer itemsets $\{a\}, \{a, b\}$, etc. as subsequent candidates. A top-down enumeration begins with the complete set $\{a, b, c, d\}$ and generates the shorter itemsets $\{a, b, c\}, \{a, b\}$, etc. as subsequent candidates.

3.2 ZBDD Mining Algorithm

This section describes our algorithm for mining EPs. ZBDDs allow compression of sparse itemsets and they also allow efficient set operations. Here we use ZBDDs for generating pattern candidates, and also for storing the output patterns. This is similar to existing methods which use structures such as FP-trees[14] and Pattern trees[12].

The search space of EPs is dictated by the contents of the negative dataset and patterns are grown bottom-up in a depth-first fashion. Figure 3 shows an example of pattern lattice for a given set of items $I = \{a, b, c, d\}$. A bottom-up depth-first enumeration begins with the empty set $\{\}$, and the subsequent candidates are the longer itemsets, e.g. $\{a\}$,$\{a, b\}$, etc. We will refer to the partially grown patterns as *prefixes*. The output ZBDD stores the minimal EPs and it is constructed incrementally. To further optimise the algorithm, a number of pruning strategies are employed.

Early pruning of invalid candidates: In principle, our algorithm could examine a search space covering all possible item combinations. However, this is unnecessary and instead we traverse a search space which avoids generating candidate patterns which could never satisfy the β constraint. For any given prefix p (candidate), we can partition D_n into the set of transactions not containing p (labelled by $D_n^{\overline{p}}$) and transactions which contain p (labelled by D_n^p). If p needs growing, then it only needs to be extended by an item which is not from at least one of the transactions in D_n^p, i.e. from the complement of one of the transactions in D_n^p (otherwise a non minimal pattern will result). It is therefore profitable for the input ZBDD to consist of the complements of the transactions in D_n (i.e. $\overline{D_n}$). Traversing $\overline{D_n}$ ensures that the candidate generation space is much smaller, which is particularly effective if $|D_n|$ is relatively small, as is often the case for biological data.

Algorithm 1 mineEP(P, $prefix$, D_p, D_n, α, β)

Call mineEP($\overline{D_n}$, $\{\}$, D_p, D_n, α, β) to begin mining initially.

Input: P : a ZBDD of the search space which is a projection of $\overline{D_n}$
$prefix$: prefi x of the patterns
D_p : bitmaps of the positive dataset
D_n : bitmaps of the negative dataset
α : a min support (wrt. D_p) threshold
β : a max support (wrt. D_n) threshold

Output: $zOut$: a ZBDD representing the set of minimal itemsets p satisfying $support(p, D_p) \geq \alpha$ and $support(p, D_n) \leq \beta$.

```
1: if P is a sink node, then
2:    // The end of the search space for growing prefix is reached;
3:    // return prefix as a minimal EP if it passes β constraint
4:    if support(prefix, D_n) ≤ β then
5:       return 1
6:    else
7:       return 0 // Remove prefix from the output ZBDD
8:    end if
9: else
10:   // Let P = node(x, P_1, P_0)
11:   // Grow prefix with the next item in the search space
12:   prefix_new = prefix ∪ {x}
13:   if support(prefix_new, D_p) < α, then
14:      // α-constraint pruning: prune prefix_new
15:      zOut_x = 0
16:   else if support(prefix_new, D_n) ≤ β then
17:      // β-constraint pruning: stop growing prefix_new
18:      zOut_x = 1
19:   else
20:      // Explore supersets of prefix_new from instances which do not
21:      // contain x
22:      zOut_x = mineEP(P_0, prefix_new, D_p, D_n, α, β)
23:   end if
24:
25:   // Mine patterns not containing x from the remaining search space
26:   zOut_x̄ = mineEP(P_0 ⋃_Z P_1, prefix, D_p, D_n, α, β)
27:
28:   // Non-minimal pattern elimination
29:   zOut_x = NotSupSet(zOut_x, zOut_x̄)
30:   zOut = getNode(x, zOut_x, zOut_x̄)
31: end if
```

α **constraint pruning**: This strategy is based on the well-known anti-monotonicity, or a-priori principle. Any prefix which doesn't satisfy the α constraint should have its supersets pruned. Also, as a pre-processing step, any item whose $support(D_p) < \alpha$ can be deleted from D_p and D_n.

β **constraint pruning:** This strategy is based on the monotonicity of the β constraint. If a prefix satisfies the β constraint, it is not extended any further, since a non minimal pattern would result.

Non minimal pattern pruning: Due to the recursive decomposition aspect of the algorithm, the generated patterns are locally minimal for each recursion, but they may be non-minimal globally. Hence, it is profitable to immediately prune any non-minimal patterns after the completion of each decomposition.

Our algorithm for finding minimal EPs, namely *mineEP*, is shown in Algorithm 1, which we will explain line by line. The first input parameter, P, is a ZBDD which dictates the remaining candidates. $prefix$ is a partially grown pattern, which satisfies the α constraint but fails the β constraint. D_p and D_n correspond to the bitmaps from the respective datasets, and are used for computing $support$. Note: the bitmap of itemset q in dataset D is denoted $bitmap(q, D)$; $support(q, D) = \frac{\text{num. of ones in } bitmap(q,D)}{|D|}$.

Mining is invoked by calling $mineEP(\overline{D_n}, \{\}, D_p, D_n, \alpha, \beta)$, and then called upon recursive projections of $\overline{D_n}$. Lines 1-8 state the terminal condition of the recursion. When it reaches a sink node, it has reached the end of the search space for growing the given $prefix$. If $prefix$ passes the β constraint, it is a satisfying minimal EP and the ZBDD sink-1 node is returned. Otherwise, $prefix$ cannot be part of the output ZBDD, so the sink-0 node is returned. The core routines in the algorithm are: 1) compute $zOut_x$, which grows $prefix$ with the next item x found in the candidates; 2) compute $zOut_{\overline{x}}$, which contains the patterns not containing x. They will be the two subtrees of the ZBDD output (line 30).

Before attempting to grow $prefix$ with the next item, x, the algorithm first tests whether the α and β prunings can be performed. Line 15 prunes $prefix_{new}$ (= $prefix \cup \{x\}$) and its supersets by the α-constraint pruning. The support of $prefix_{new}$ is calculated incrementally using $bitmap(prefix)$ which has been computed in the previous recursion, i.e. $bitmap(prefix \cup \{x\}) = bitmap(prefix) \cap bitmap(\{x\})$. Line 18 uses β-constraint pruning to stop $prefix$ from being grown. Finally, if none of these two cases is applicable, x is appended to the $prefix$ and instances of P which do not contain x are explored, storing the output in $zOut_x$.

Line 26 computes $zOut_{\overline{x}}$ from a projection of the database by excluding x. Some itemsets in $zOut_{\overline{x}}$ may be contained by some itemsets in $zOut_x$. The non-minimal patterns are pruned using a primitive ZBDD operation $notSupSet$ (line 29).

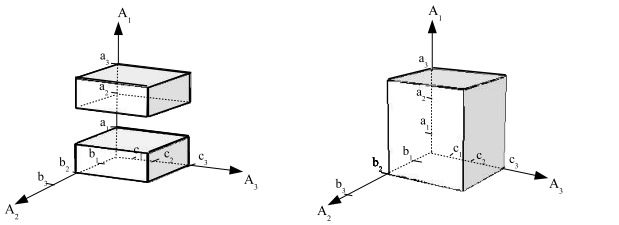
Optimisations: For the special case where $\beta = 0$, which corresponds to the jumping emerging patterns, the EPs must have at least one item in common with each instance in $\overline{D_n}$. Thus, $\overline{D_n}$ and its projections can sufficiently be represented using their minimal itemsets. This allows the computation of $zOut_x$ (line 26) to be optimised by processing ($P_0 \bigcup_{Z_{min}} P_1$) instead. As a result, $zOut_x$ only contains patterns which may be non-minimal by the item x. Non-minimal pattern elimination (line 29) can thus be computed using ($zOut_x \setminus zOut_{\overline{x}}$) which is a simpler, thus faster, operation.

Optimal variable ordering: We investigated a number of heuristics for finding the optimal variable ordering for efficient computation of *mineEP*, based on the item frequencies in D_p and D_n. Three alternative strategies were worthy of consideration.

The first heuristic places the least frequent item in D_p at the top of the ZBDD, with subsequent items being ordered by increasing support in D_p. This aims to achieve early α-constraint pruning which reduces the depth of the recursions, and in turn reduces the number of database projections that are constructed.

The second heuristic places the least frequent item in D_n (i.e. most frequent in $\overline{D_n}$) at the top, with other items being ordered by increasing frequency in D_n. This can be justified on two levels. Firstly, consider line 22 in the algorithm. Having a smaller P_0 is likely to be advantageous, particularly when the ZBDD at that point is large. Using the most frequent item in $\overline{D_n}$ at the top level means that P_0 is likely to be small for the early recursive calls. Secondly, this heuristic gives higher preference to the β constraint, in a similar manner to that for the α constraint in the first heuristic, the aim being to achieve early β-constraint pruning.

The third heuristic clusters items from the same attribute domain because any emerging pattern contains at most one item from any one attribute, allowing early pruning. Moreover, this ordering can be combined with the other heuristics by ordering the items within each attribute by increasing support in D_p (based on the first heuristic), or by increasing support in in D_n (based on the second heuristic). The attributes are then ordered by increasing minimum support of its items.



$s_1 = \{a_1, a_3, b_1, b_2\}$ (a)

$s_2 = \{a_1, a_2, a_3, b_1, b_2, c_1, c_2, c_3\}$ (b)

Figure 4: Geometric Representation of Disjunctive EPs

4. DISJUNCTIVE EMERGING PATTERNS

We now investigate a more general type of contrast patterns, which we will hereafter refer to as a *disjunctive emerging patterns*.

Recall that emerging patterns correspond to conjunctions of items that have high support in D_p and low support in D_n., e.g. $a \wedge e$ was an EP for Table 1, given $\alpha = 0.25$ and $\beta = 0$. Disjunctive emerging patterns (disjunctive EPs) generalise EPs by allowing disjunctions as well as conjunctions for pattern descriptions. They essentially correspond to a restricted class of CNF formulae, which use items as variables and are a conjunction of disjunctions, where each disjunction contains only items coming from the same attribute domain. No negation is allowed and there must exist at least one item from each attribute domain in the formula.

e.g. Given a dataset having three attributes A_1, A_2, A_3, with domains $\{a_1, a_2, a_3\}$, $\{b_1, b_2, b_3\}$, $\{c_1, c_2, c_3\}$. A valid disjunctive EP may be represented by a formula f, where $f = (a_1 \vee a_3) \wedge (b_1 \vee b_2) \wedge (c_1 \vee c_2 \vee c_3)$. Without any ambiguity, we can alternately represent f as an itemset $\{a_1, a_3, b_1, b_2, c_1, c_2, c_3\}$, where it is implicitly understood that conjunctions exist across attributes and disjunctions exist within attributes. Henceforth, we will blur the distinction between disjunctive formulae and their itemset representations.

Given a formula describing a disjunctive emerging pattern, we need to be able to calculate its support.

Definition 2. Let s be a disjunctive emerging pattern. The support of s in a dataset D, $support(s, D)$, is the number of instances from D which are contained in (the itemset representation of) s.

Using this revised definition of $support$, we can define appropriate α and β support thresholds for disjunctive EPs.

Definition 3. Given D_p, D_n and support thresholds α and β. A **disjunctive emerging pattern** is an itemset d such that i) d contains at least one item from the domain of every attribute, ii) $support(d, D_p) \geq \alpha$, and iii) $support(d, D_n) \leq \beta$. d is said to be **maximal** if there does not exist another disjunctive emerging pattern d' such that $d \subset d'$.

Observe that a disjunctive EP corresponds to a region of high contrast, i.e. a subspace which contains at least α instances from D_p and at most β instances from D_n. (see Figure 4 for illustration of the geometric representation of itemsets given three attribute domains $\{a_1, a_2, a_3\}$, $\{b_1, b_2, b_3\}$, $\{c_1, c_2, c_3\}$). Also, consider again Table 1 and suppose $\alpha = 0.5$ and $\beta = 0$. The maximal disjunctive EPs include $\{a, c, d, e, f, h, i\}$ and $\{a, b, d, e, g, i\}$. From a classification perspective, an unknown data instance seems more likely to be from the positive class if is contained in one of these itemsets.

It is possible to define variants of disjunctive EPs. One important case arises for datasets with ordered domains. and effectively corresponds to disjunctive EPs having contiguous ranges on each attribute. Suppose an attribute A_i has an ordered domain of items. We define a *contiguous* subset of $dom(A_i)$ as a collection of items which appear consecutively in the order of $dom(A_i)$. An itemset is contiguous if it does not contain any non-contiguous subsets from the domain of each attribute. Consider again Figure 4, s_1 is a not contiguous, whilst s_2 is contiguous.

Definition 4. Given datasets D_p and D_n, an itemset S is a **maximal contiguous disjunctive emerging pattern** if i) S is contiguous, ii) $support(S, D_p) \geq \alpha$, iii) $support(S, D_n) \leq \beta$, and iv) There is no proper superset of S satisfying conditions i-iii.

Compared to disjunctive EPs, contiguous disjunctive EPs might be considered more meaningful to humans, since their corresponding regions are connected, i.e. do not contain any gaps or holes.

4.1 Relationships Between Emerging Patterns and Disjunctive Emerging Patterns

We now examine the relationship between disjunctive EPs and EPs in more detail. Broadly speaking, disjunctive EPs can be viewed as generalisations of EPs, allowing more expressive contrasts.

THEOREM 1. *Let p be an emerging pattern. Then p is contained in some disjunctive emerging pattern using the same α and β support thresholds.*

Observe that the converse of this theorem does not hold. It is often true that a disjunctive EP does not contain any EP. e.g. There is no EP in Table 1 if $\alpha = 0.5$ and $\beta = 0$, yet there exist several disjunctive EPs satisfying these constraints.

Also observe that multiple EPs of lower support can be merged together to form a disjunctive emerging pattern. e.g. Again looking at Table 1, both $\{a, d\}$ and $\{a, e\}$ are EPs when $\alpha = 0.25$ and $\beta = 0$. They correspond to the boolean formulae $a \wedge d$ and $a \wedge e$, each having $support(D_p) = 0.25$. These two EPs can be "unioned" to yield $a \wedge (d \vee e)$, which is equivalent to the disjunctive EP $a \wedge (d \vee e) \wedge (g \vee h \vee i)$ (since $g \vee h \vee i$ is trivially true for any transaction), having $support(D_p) = 0.5$ and $support(D_n) = 0$.

An interesting special case exists when the cardinality of the domain for every attribute is exactly two. In this circumstance, the two types of EPs coincide.

To summarise, the key differences between emerging patterns and disjunctive emerging patterns are:

- Disjunctive emerging patterns are more expressive. They can capture contrast regions of greater complexity. This makes them more suitable for ordered data, where it is frequently desirable for the contrasts to include disjunctions of items within specific dimensions
- For given thresholds α and β, it is often the case that a dataset may contain many disjunctive emerging patterns but no emerging pattern

Being more expressive, disjunctive EPs are more complex to compute. However, it turns out we can still accomplish this efficiently using a technique similar to the algorithm in Section 3.2.

4.2 Mining Disjunctive EPs

We now describe how our *mineEP* algorithm can be adapted for mining maximal disjunctive EPs. The algorithm is called *mineDEP* (shown in Algorithm 2). Being similar to *mineEP*, we will only point out their main differences.

Because of the generality of disjunctive EPs, they are likely to contain many items. Our approach for mining disjunctive EPs explores the pattern lattice in a depth-first top-down manner, rather than the bottom-up manner that was used for mining EPs. A top-down enumeration of the patterns begins with the most general itemset (i.e. containing all the items) and at each step, generate shorter itemsets as candidates (refer to Figure 3 for illustration). For efficiency purposes, it is better to work with pattern complements, which are likely to contain fewer items, rather than the patterns themselves. So, candidates are generated by growing prefixes in this complemented pattern space. The initial input ZBDD is built from D_n. Again, this aims to eliminate the generation of invalid candidates, but D_n is used here instead of $\overline{D_n}$ which was used in *mineEP* since the enumeration of the maximal disjunctive EPs is proceeding top-down, rather than bottom-up.

Pruning based on the α and β constraints is similar to that used in *mineEP*. Support checking, however, must be done using pattern complements and so intersection tests, rather than containment tests are performed on the bitmaps. e.g. If a disjunctive EP p is required to have $support(p, D_n) \leq \beta$, then its complement $\overline{p}$ must satisfy $cover$[2] $(\overline{p}, D_n) \geq (1-\beta)$. Similarly, the α constraint can be translated to $cover(\overline{p}, D_p) \leq (1-\alpha)$.

The conditions for α and β pruning are also different to that of the previous algorithm. These conditions for α and β pruning are inverted from *mineEP*, since maximal, rather than minimal patterns are being computed. More precisely, exploration of the search space stops if one of the following conditions is satisfied:
1) if $support(\overline{prefix_{new}}, D_p) < \alpha$, i.e. $cover(prefix_{new}, D_p) > (1-\alpha)$, then do α constraint pruning (line 15);
2) if $support(\overline{prefix_{new}}, D_n) \leq \beta$, i.e. $cover(prefix_{new}, D_n) \geq (1-\beta)$, then do β constraint pruning (line 18).
Finally, the terminal case tests whether $support(\overline{prefix}) \leq \beta$, i.e. $cover(prefix, D_n) \geq (1-\beta)$ (line 1-8). The algorithm is initialised by passing the negative dataset D_n to its first parameter, i.e. $mineDEP(D_n, \{\}, D_p, D_n, \alpha, \beta)$.

Finally, the ZBDD variable ordering locates the item which most frequently occurs in D_n at the top and items are ordered decreasingly by their frequency in D_n thereafter. This is essentially the inverse of the second ordering heuristic that was used for *mineEP*, again due to the top-down nature of the search strategy.

4.3 Mining g-Contiguous Disjunctive EPs

As we have seen, contiguous disjunctive EPs are subclass of disjunctive EPs. We now define another more general subclass of disjunctive EPs, namely *g-contiguous disjunctive EPs*, and describe a technique for mining them.

Suppose an attribute A_i has an ordered domain of items. We define a *g-contiguous* subset of $dom(A_i)$ as a collection of items which appear in the same order in $dom(A_i)$ and the gap between any two consecutive items is not larger than g. An itemset is g-contiguous if it does not contain any non-g-contiguous subsets from the domain of each attribute. Furthermore, an itemset p is a **maximal g-contiguous disjunctive EP** if: i) p is a disjunctive EP, ii) p is g-contiguous, iii) none of its proper supersets satisfies conditions i-ii. When $g = 0$, p is a contiguous disjunctive EP.

We propose a post-processing operation, *contigSplit*, to derive the maximal g-contiguous disjunctive EPs from the disjunctive EPs found using *mineDEP*. It complements each of the input itemsets and splits it into maximal subsets satisfying the given g and α constraints (it is guaranteed that they satisfy the β constraint). The pseudo code is shown in Algorithm 3. It begins mining by calling

[2] $cover(p, D)$ = the fraction of the transactions in D which contain some item in p; $cover(p, D) = 1 - support(\overline{p}, D)$.

Algorithm 2 mineDEP(P, $prefix$, D_p, D_n, α, β)

Call mineDEP(D_n, {}, D_p, D_n, α, β) to begin mining initially.
Input: P : a ZBDD of the search space which is a projection of D_n
$prefix$: prefi x of the patterns
D_p : Bitmaps of the positive dataset
D_n : Bitmaps of the negative dataset
α : a min support (wrt. D_p) threshold
β : a max support (wrt. D_n) threshold
Output: $zOut$: a ZBDD representing the set of minimal itemsets p satisfying $cover(p, D_p) \leq 1-\alpha$ and $cover(p, D_n) \geq 1-\beta$.

```
 1: if P is a ZBDD sink node, then
 2:   // The end of the search space for growing prefix is reached
 3:   // prefix is a satisfying pattern if it passes β constraint
 4:   if cover(prefix, D_n) ≥ 1 − β then
 5:     return 1
 6:   else
 7:     return 0 // Remove prefix from the output ZBDD
 8:   end if
 9: else
10:   // Let P = node(x, P_1, P_0)
11:   // Grow prefix with the next item in the search space
12:   prefix_new = prefix ∪ {x}
13:   if cover(prefix_new, D_p) > 1 − α, then
14:     // α constraint pruning: prune prefix_new
15:     zOut_x = 0
16:   else if cover(prefix_new, D_n) ≥ 1 − β then
17:     // β constraint pruning: stop growing prefix_new
18:     zOut_x = 1
19:   else
20:     // Explore supersets of prefix_new from instances which do not
21:     // contain x
22:     zOut_x = mineDEP(P_0, prefix_new, D_p, D_n, α, β)
23:   end if
24:
25:   // Explore candidates from the remaining search space
26:   zOut_x̄ = mineDEP(P_0 ⋃_Z P_1, prefix, D_p, D_n, α, β)
27:
28:   // Non-minimal patterns elimination
29:   zOut_x = notSupSet(zOut_x, zOut_x̄)
30:   zOut = getNode(x, zOut_x, zOut_x̄)
31: end if
```

$contigSplit(Z_{dEP}, attrDomains, g)$, where Z_{dEP} is a ZBDD of the complement of maximal disjunctive EPs, $attrDomains$ is a vector of ZBDDs, each of which contains the domain items from each attribute, g is the gap size threshold. Conceptually, every disjunctive EP has a set of maximal g-contiguous subsets induced in each dimension, computed using a $splitComplement$ subroutine which we will explain shortly, and these subsets across dimensions are pair-wise unioned using an efficient ZBDD operation, $DotProd$. The α constraint is pushed inside the routines in a similar manner to that in the *mineDEP* algorithm.

The subroutine $splitComplement$ complements a given itemset Q with respect to a set of domain items D, and simultaneously splits it into maximal subsets satisfying a g constraint. $prefix$ is the output candidate. Two ZBDDs containing Q and D, respectively, are traversed in parallel, and $prefix$ is grown by appending items in D which do not occur in Q (line 13-14). The parameter $gapSize$ indicates the number of items that have been skipped since the last item that was inserted to $prefix$ ($gapSize = 0$ when $prefix = \{\}$). Thus, every sequential item occurring in Q increments $gapSize$ by 1 (line16). If $gapSize$ has reached the threshold, then $prefix$ is a maximal g-contiguous subset, and a new empty prefix is grown using the remaining items (line 18). Finally, if there are no items in Q, D gives the complement of Q and it is a maximal contiguous subset of $\overline{Q}$. Thus, the union of the respective itemsets D and $prefix$ is returned (line 3).

Algorithm 3 contigSplit(P,$attrDomains$, g)

Input: P: a ZBDD of the complement of maximal disjunctive EPs,
$attrDomains = [dom_1, dom_2, ...dom_k]$: dom_i is a ZBDD of $dom(A_i)$, where $i \in [1, 2..k]$,
g: a maximum gap size constraint
Output: $zOut$: maximal g-contiguous disjunctive EPs
1: $zOut$ = {} // initialisation
2: **for all** itemsets p in P **do**
3: // Compute projection of p in the domain of each attribute
4: for all i in 1,..k, $p_i = p \cap dom_i$
5:
6: // Compute split-complement of p in the dimension of each attribute,
7: // and conjugate the g-contig. subsets from across dimensions
8: $prefixes$ = {{}}
9: **for all** i in 1,..k **do**
10: $splits_i$ = splitComplement$(p_i, dom_i, \{\}, 0)$
11: $prefixes$ = DotProd($prefixes, splits_i$)
12: **end for**
13: $zOut = zOut \bigcup_{Zmax} prefixes$
14: **end for**
15: **return** $zOut$

splitComplement(Q**,** D**,** $prefix$**,** $gapSize$**) =**
Input: Q: a ZBDD containing the complement of the itemset to be split,
D: a ZBDD containing the domain items ($Q \subseteq D$),
$prefix$: a ZBDD containing a prefi x itemset,
$gapSize$: gap size in $prefix$,
Output: $zOut$: the set of g-contiguous subsets of $\overline{Q}$, w.r.t. D, which satisfy the g constraint
1: **if** (Q is a ZBDD sink node) **then**
2: // Q contains an empty itemset; $\overline{Q} = D$ and it has no gap
3: $zOut$ = DotProd(D, $prefix$)
4: **else**
5: // Let $Q = node(x, Q_1, Q_0)$, $D = node(y, D_1, D_0)$; $Q_0 = 0$
6: // and $D_0 = 0$ since each of Q and D contains only one itemset
7:
8: // Append y to $prefix$ if it does not occur in Q (i.e. y occurs in $\overline{Q}$);
9: // otherwise, increment $gapSize$, or, if $gapSize = g$,
10: // $prefix$ is fully grown and a new empty prefi x is grown.
11: **if** (x has higher index than y) **then**
12: // y is not in Q, turn on the bit of y in $prefix$
13: $prefix_{new}$ = change$(prefix, y)$
14: $zOut$ = splitComplement$(Q, D_1, prefix_{new}, 0)$
15: **else if** ($gapSize < g$) **then**
16: $zOut$ = splitComplement$(Q_1, D_1, prefix, gapSize$+1)
17: **else if** ($gapSize = g$) **then**
18: $zOut = prefix \bigcup_Z$ splitComplement$(Q_1, D_1, \{\}, 0)$
19: **end if**
20: **end if**
21: **return** $zOut$

5. PERFORMANCE STUDY

In this section we assess the performance of our techniques for mining emerging patterns and disjunctive emerging patterns.

Our algorithms were implemented in C++ using the ZBDD library functions in the CUDD package [34] and EXTRA library [26]. All experiments were conducted on a IBM eServer pSeries 650 (eight POWER4+ 1.45GHz CPU, 16 GB RAM) running AIX 5L 5.2 with a cpu-time limit 100,000 seconds. The ordering used for our ZBDD algorithms was decreasing frequency in D_n, based on the second heuristic. This section will conclude with a study comparing the performance of mining disjunctive EPs using different variable ordering heuristics.

We carried out experiments on two gene-expression datasets[3], the Leukaemia dataset ALL-AML, previously studied in [18] and lung cancer. Table 3 shows their characteristics. Column 1 (resp. Column 2) shows the class which was chosen as positive (resp. negative) class and its corresponding number of instances. These datasets were chosen due to their challenging characteristics. As is common for biological data, they contain a huge number of dimensions but only have a few instances. Work in [19, 20, 18] have studied mining minimal emerging patterns for these datasets.

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

Table 3: Data Characteristics

Dataset	# trans. in D_p	# trans.in D_n	# attr.
ALL-AML	27 (ALL)	11 (AML)	7129
lung cancer	16 (Mesothelioma)	16 (ADCA)	12535

Both datasets have continuous attribute domains. The values were discretised using an entropy discretisation method, which had the effect of removing some of the attributes. After discretisation, the ALL-AML dataset is reduced to 865 attributes, lung cancer is reduced to 2172 attributes. The discretised attributes are ordered by decreasing entropy value.

5.1 EP Mining Performance

We study the scalability of our ZBDD technique for mining (minimal) emerging patterns and compare it against a state of the art technique based on a variant of frequent pattern trees [12] (hereafter referred to as Pattern-Tree EP-miner). The authors of this paper provided us with an implementation of their algorithm. Other techniques for mining emerging patterns exist (e.g. [4, 3]), but have similar, or inferior running behaviour to that of [12] and so we do not include them in our comparison.

The first scenario uses the ALL-AML data with constraints $\alpha = 90\%$ and $\beta = 0$, and an increasing number of dimensions. Looking at Figure 5a and Figure 5b, the mining time of ZBDD EP-miner is substantially faster than Pattern-tree EP-miner by a factor of approximately 100 times for between 40 and 68 attributes (the ZBDD miner running time is very close to the x-axis in this region). For more than 68 attributes, mining was impossible for the Pattern-Tree EP-miner due to memory limits being exceeded, whereas the ZBDD EP-miner was able to run effectively for up to 800 attributes. This is in line with previously published results from [18, 9], where EPs were only mineable for datasets with no more than around 70 attributes. For the Lung Cancer dataset which appears to be an easier dataset due to the smaller number of patterns, the Pattern-Tree miner is able to mine EPs for a larger number of attributes. The ZBDD EP-miner is substantially superior in running time to the Pattern-Tree miner, giving speedups of over 100 times, and it was able to run effectively for up to 1700 attributes.

5.2 Disjunctive EP Mining Performance

We now study the scalability of our ZBDD algorithm for mining (maximal) disjunctive EP. In particular, we focus on its behaviour as we vary number of attributes and the value of α. No comparison is made against other systems, since we are not aware of any other work that is suitable for mining these patterns.

Varying the number of dimensions. Figure 5c and Figure 5d show the time for mining maximal disjunctive emerging patterns as the number of attributes is varied for both datasets. Support constraints $\alpha = 90\%$ and $\beta = 10\%$ are used. The number of patterns output is shown in Figure 5e and Figure 5f. Not surprisingly, including more dimensions increases the search space, the size of the output patterns, and also the running time, exponentially.

Importantly though, the ZBDD technique is able to mine this complex kind of patterns even when there are a very large number of attributes. The maximal disjunctive EPs for the lung cancer

dataset are mined in around 60000 seconds using all its attributes (2172 attributes). For the ALL-AML dataset, up to 700 attributes can be handled in around 1000 seconds. Mining beyond this attribute limit was proved impossible because of memory limits being exceeded, due to the very large number of output patterns.

Since the output patterns are stored in a ZBDD, it is interesting to reflect on the compression being achieved. Figure 5g shows the number of ZBDD nodes in the output, for the lung cancer data with respect to varying the number of attributes, given $\alpha = 90\%$ and $\beta = 10\%$. When there are 2172 attributes, the ZBDD requires 1236100 nodes, to store the 2080960 maximal disjunctive emerging patterns. Figure 5h gives a more detailed picture, presenting a histogram of the pattern lengths. We can see that most of the patterns are close to the maximum length of 4371 items having an average length of around 4367 items.

Varying the support thresholds. Figure 5i and Figure 5j show the output patterns in ALL-AML dataset (using 1000 items) and in lung cancer dataset (using 3500 items) according to an increasing α constraint (given $\beta = 0$). We can see that for both datasets, the number of patterns output is highly sensitive to α up to a certain limit (around 45%), with its sensitivity thereafter decreasing.

Comparative pattern volumes and mining time: Finally, we compare the volumes of the different kinds of EP in a given dataset. Figure 5k looks at the lung cancer dataset, using 230 items and $\beta = 0$, allowing α to vary. The figure shows the number of i) minimal emerging patterns, ii) maximal disjunctive emerging patterns, and iii) maximal contiguous disjunctive emerging patterns. For this scenario, it is clear that there exist fewer EPs than the disjunctive EPs and their contiguous variants. This is expected, since EPs are more specific versions of the disjunctive patterns. Though it is not shown here, in our experience, it can often be the case that under given support thresholds, a dataset may contain zero EPs, but may contain hundreds of (possibly contiguous) disjunctive EPs.

The corresponding mining times for this dataset is shown in Figure 5l. The mining times for mining EPs and disjunctive EPs lie on the x-axis, and the times for mining contiguous EPs are higher due to the postprocessing splitting operation. It can be seen that the splitting time is constant with respect to a varying number of patterns from varying α. Indeed, all the algorithms have a roughly constant time with respect to α for this scenario.

5.3 Variable Ordering

We also study the effect of using various variable orderings in the ZBDD for mining disjunctive EPs. Figures 5m, 5n, and 5o show a comparison between the different heuristics we considered. The first heuristic is employed by ordering the variables by decreasing frequency in D_p. The second heuristic is employed by ordering the variables by decreasing frequency in D_n. Lastly, the third heuristic is employed by arranging items from the same attribute close to each other and two-level ordering is used, i.e. items within each attribute are ordered by decreasing support in D_n and the attributes are ordered by decreasing maximum support of its items.

Figure 5m shows that employing the second ordering on the ZBDDs achieves the fastest mining time as it reduces the complexity of the decomposed subtasks. Shown in Figure 5n, the corresponding input ZBDDs have similar sizes using either the second or the third ordering, but the mining times for the third ordering grow exponentially as α decreases. The first ordering produces larger input ZBDDs, which explains its mining time being the slowest. Furthermore, Figure 5o shows that the output ZBDDs are the smallest when the second and the third orderings are used.

6. DISCUSSION

The results in the previous section are only a snapshot of the experiments we performed. We also tested our techniques on a number of other biological datasets, with performance being similarly pleasing overall. A general conclusion from our work is that ZBDDs can be used for very effective mining of both emerging patterns and disjunctive emerging patterns. A natural question to ask is, what advantages does a ZBDD technique have over a frequent-pattern tree (fp-tree) technique for mining contrasts? Here we provide some preliminary observations.

Both fp-trees and ZBDDs are tree-like structures for storing transactions using a variable ordering. A structural difference between the two is that ZBDDs allow sharing of transactions via fan-in, whereas fp-trees do not allow fan in. The use of fan-in allows not only prefix sharing but also suffix sharing, resulting a high compression of both input and output. Furthermore, ZBDDs also allow sharing between all structures throughout mining since it uses a global variable ordering. In particular, it is possible for the input ZBDD (the transactions), the input projection in the intermediate mining steps, and the output ZBDD, to share subtrees even though each of them represents different kinds of data! This provision of increased opportunities for sharing and compression is particularly important for high dimensional datasets, where the number of candidates can be very large and mining practicality may be dependent on memory consumption considerations.

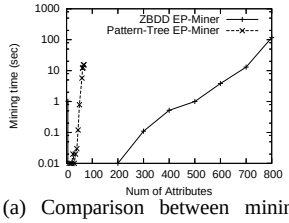
Another significance of our technique is in the recursive decompositions. When a ZBDD is recursively decomposed, its decompositions are able to share substructures with one another. A shared structure is constructed only once and results from past manipulations are re-used as needed. This is different from fp-trees, whose recursive decompositions (conditional fp-trees) are created afresh and do not share with one another.

A final important aspect of ZBDD is the existence of polynomial time operations, such as set (minimal/maximal) union, set difference, etc, especially when there is a large amount of sharing within the structure. One implication of this is an efficient removal of non-maximal patterns. Considerable work has gone into developing efficient library implementations of these and our algorithms make considerable use of them. It is an open question as to whether provably efficient counterpart operations exist for fp-trees.

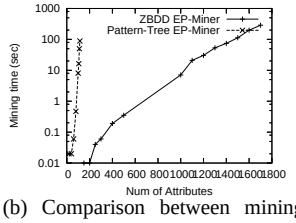
7. RELATED WORK

We have already referred to the general work in the area of ZBDD in Section 3.1. However, we are only aware of one paper [25] where ZBDD is used for pattern mining. They propose a method for finding frequent patterns. Their approach is different from ours in the sense that they explicitly store the support information by constructing multiple shared-ZBDDs which groups itemsets based on their (binary-encoded) supports. It enumerates every pattern occurring at least once in the dataset, regardless of the input threshold value α supplied by the user, making it inefficient for high values of α or for mining in high dimensional data since millions of patterns may exist. On the other hand, our proposal stores the input transactions in a single ZBDD, reducing its overall memory consumption, and pushes constraints deep inside the ZBDD operations. Additionally, we use a secondary data structure such as bitmaps for counting support, instead of storing support information inside the ZBDD.

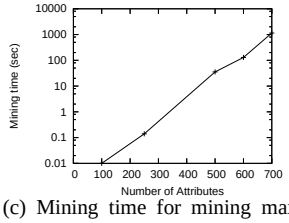
Emerging patterns were introduced in [8], and have been successfully used for constructing highly accurate classifiers [17]. In particular, work in [10] have proposed a strong EP-based classifier using an α support constraint and a minimum growth rate constraint. Moreover, emerging patterns have also been used for pre-



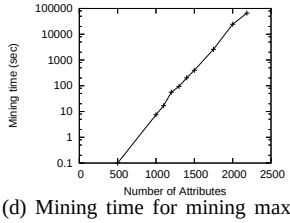
(a) Comparison between mining time for finding minimal EPs using Pattern-Tree and ZBDD w.r.t num. of attr., ALL-AML dataset (α = 90%, β = 0)



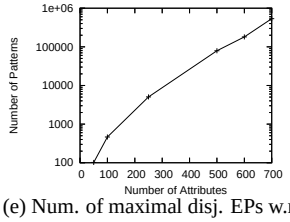
(b) Comparison between mining time for finding minimal EPs using Pattern-Tree and ZBDD w.r.t num. of attr., lung cancer dataset (α = 90%, β = 0)



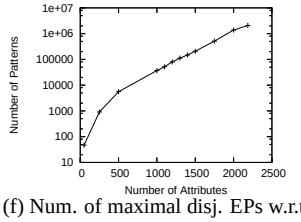
(c) Mining time for mining maximal disj. EPs (sec.) w.r.t num. of attr., ALL-AML dataset ($\alpha = 90\%, \beta = 10\%$)



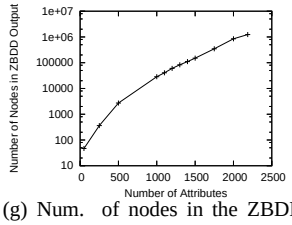
(d) Mining time for mining maximal disj. EPs (sec.) w.r.t Num. of attr., lung cancer dataset ($\alpha = 90\%, \beta = 10\%$)



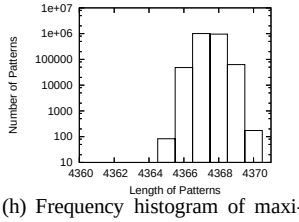
(e) Num. of maximal disj. EPs w.r.t num. of attr., ALL-AML dataset ($\alpha = 90\%, \beta = 10\%$)



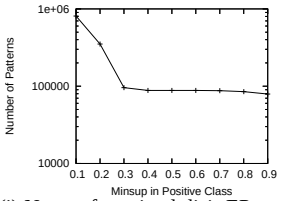
(f) Num. of maximal disj. EPs w.r.t num. of attr., lung cancer dataset ($\alpha = 90\%, \beta = 10\%$)



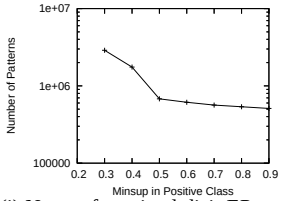
(g) Num. of nodes in the ZBDD output of maximal disj. EPs w.r.t. num. of attr., lung cancer dataset (α = 90%, β = 10%)



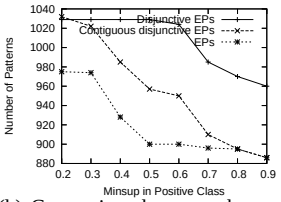
(h) Frequency histogram of maximal disj. EPs, lung cancer dataset (4371 items; α = 90%, β = 10%)



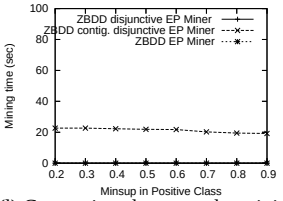
(i) Num. of maximal disj. EPs w.r.t minimum support in D_p, ALL-AML dataset (1000 items; max support in D_n = 10%)



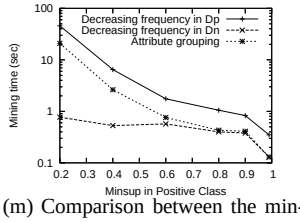
(j) Num. of maximal disj. EPs w.r.t minimum support in D_p, lung cancer dataset (3500 items; max supp in D_n = 10%)



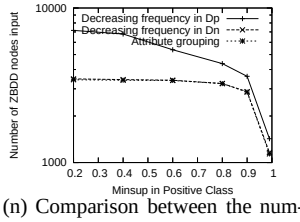
(k) Comparison between the number of minimal EPs, maximal disj. EPs, and maximal contig. disj. EPs w.r.t minimum support in D_p, lung cancer dataset (230 items; β = 0)



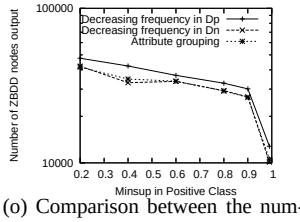
(l) Comparison between the mining time for mining minimal EPs, maximal disj. EPs, and maximal contig. disj. EPs using ZBDD w.r.t minimum support in D_p, lung cancer dataset (230 items; β = 0)



(m) Comparison between the mining time for mining maximal disj. EPs using different variable orderings lung cancer dataset (1000 items; β = 0%)



(n) Comparison between the number of nodes in the ZBDD input for mining disj. EPs using different variable orderings; lung cancer dataset (1000 items; β = 0%)



(o) Comparison between the number of nodes in the ZBDD output of maximal disj. EPs using different variable orderings; lung cancer dataset (1000 items; β = 0%)

Figure 5: Performance Results

dicting the likelihood of diseases such as leukaemia [18] using gene expression data [19]. A recent method for mining emerging patterns with zero support in the negative dataset appears in [12], based on modifications to fp-tree [14]. Fp-trees have also been used as the basis for mining contrasts given other types of constraints, such as risk and odds ratio [16]. Connections between the computation of certain kinds of emerging patterns and hypergraph transversals are identified in [4].

Emerging patterns are closely related to association rules with large confidence [37] and also to work on detecting group differences [5]. Quantitative association rules [36] aim to find contiguous regions containing a minimum number of points. Moreover, contiguous disjunctive emerging patterns are similar to quantitative association rules having high confidence and a single item consequent. Another related notion is version spaces [27, 15], which correspond to emerging patterns with constraints $\alpha = 1$ and $\beta = 0$. A disjunctive version space [33] is a disjunction of version spaces, as opposed to the disjunctive emerging patterns presented here, which are a conjunction of disjunctions on attribute values. Several papers have examined the computation of empty regions or 'holes' in datasets [11, 21]. A contiguous disjunctive emerging pattern with $\beta = 0$ corresponds to a hole in D_n.

Recent work have examined mining of closed patterns from high dimensional datasets using row, instead of column (item), enumeration [28, 30, 22]. The emphasis on closed patterns, as opposed to minimal patterns means this is not directly applicable for finding minimal contrasts. However, alternative variants of emerging patterns based on closure properties can certainly be defined, e.g. see [35]. In contrast to the row enumeration work, our paper seeks to investigate the limits of column-wise mining and indeed our results showed that column-wise mining of contrasts in high dimensional datasets is feasible using ZBDDs.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we have developed efficient algorithms for mining contrast patterns in high dimensional data. We presented an algorithm based on the use of Zero Suppressed BDD as a data structure and demonstrated how mining constraints could be integrated with the standard ZBDD library routines. Our experimental results showed the technique scales well for a number of high dimensional biological datasets and allows the computation of both simple contrasts such as emerging patterns, and also more complex type of contrasts which use both disjunction and conjunction. We showed our method substantially improves on a state of the art EP-mining technique [12]. We are not aware of other work suitable for computing the complex contrasts considered. As future work, we intend to explore the use of ZBDDs for mining other types of patterns, and also their use in row enumeration mining approaches.

Acknowledgements: We would like to thank Rao Kotagiri for his comments, and Hongjian Fan for making the executable of Pattern-Tree EP-miner available for us. 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.

9. REFERENCES

[1] F. A. Aloul, I. L. Markov, and K. A. Sakallah. MINCE: A static global variable ordering for SAT and BDD. In *Int'l Workshop on Logic Synthesis*, 2001.

[2] F. A. Aloul, M. N. Mneimneh, and K. Sakallah. ZBDD-based backtrack search SAT solver. In *Int'l Workshop on Logic Synthesis*, 2002.

[3] J. Bailey, T. Manoukian, and K. Ramamohanarao. Fast algorithms for mining emerging patterns. In *Proc. of PKDD 2002*, pages 39–50.

[4] J. Bailey, T. Manoukian, and K. Ramamohanarao. A fast algorithm for computing hypergraph transversals and its application in mining emerging patterns. In *Proc. of ICDM*, pages 485–488, 2003.

[5] S. D. Bay and M. J. Pazzani. Detecting group differences: Mining contrast sets. *Data Mining and Knowledge Discovery.*, 5(3):213–246, 2001.

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

[7] P. Chatalic and L. Simon. Multi-resolution on compressed sets of clauses. In *Proc. of ICTAI*, pages 2–10, 2000.

[8] G. Dong and J. Li. Effi cient mining of emerging patterns: Discovering trends and differences. In *Proc. of ACM KDD*, pages 43–52, 1999.

[9] G. Dong and J. Li. Mining border descriptions of emerging patterns from dataset pairs. *Knowledge and Information Systems*, 8(2):178–202, 2005.

[10] G. Dong and X. Zhang and L. Wong and J. Li. CAEP: Classifi cation by Aggregating Emerging Patterns. In *Proc. of the 2nd Int'l Conf. on Discovery Science*, pages 30–42, 1999.

[11] J. Edmonds, J. Gryz, D. Liang, and R. J. Miller. Mining for empty spaces in large data sets. *Theor. Comput. Sci.*, 296(3):435–452, 2003.

[12] H. Fan and K. Ramamohanarao. Fast discovery and the generalization of strong jumping emerging patterns for buildihng compact and accurate classifi ers. *IEEE Transactions on Data Engineering*, To appear.

[13] H. Fujii, G. Ootomo, and C. Hori. Interleaving based variable ordering methods for ordered binary decision diagrams. In *Proc. of IEEE/ACM ICCAD '93*, pages 38–41, 1993.

[14] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. of the Int'l Conf. on Management of Data*, pages 1–12, 2000.

[15] H. Hirsh. Generalizing version spaces. *Machine Learning*, 17(1):5–45, 1994.

[16] H. Li, J. Li, L. Wong, M. Feng, and Y. P. Tan. Relative risk and odds ratio: A data mining perspective. In *PODS*, 2005.

[17] J. Li, G. Dong, and K. Ramamohanarao. Making use of the most expressive jumping emerging patterns for classifi cation. In *Proc. of PAKDD 2000*, pages 220–232.

[18] J. Li, H. Liu, J. R. Downing, A. Yeoh, and L. Wong. Simple rules underlying gene expression profi les of more than six subtypes of Acute Lymphoblastic Leukaemia (ALL) patients. *Bioinformatics*, 19:71–78, 2003.

[19] J. Li and L. Wong. Emerging patterns and gene expression data. In *Proc. of the 12th Workshop on Genome Informatics*, pages 3–13, 2001.

[20] J. Li and L. Wong. Identifying good diagnostic gene groups from gene expression profi les using the concept of emerging patterns. *Bioinformatics*, 18(10):1406–1407, 2002.

[21] B. Liu, L. P. Ku, , and W. Hsu. Discovering interesting holes in data. In *Proc. of IJCAI*, pages 930–935, 1997.

[22] H. Liu, J. Han, D. Xin, and Z. Shao. Top-down mining of interesting patterns from very high dimensional data. In *To appear in Proc. of ICDE'06*.

[23] S. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proc. of the 30th Int'l Conf. on Design Automation*, pages 272–277, 1993.

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

[25] S. Minato and H. Arimura. Combinatorial itemset analysis based on Zero-suppressed BDDs. In *IEEE/IEICE/IPSJ Int'l Workshop on Challenges in Web Information Retrieval and Integration (WIRI)*, pages 3–10, 2005.

[26] A. Mishchenko. An introduction to Zero-suppressed Binary Decision Diagrams.

[27] T. M. Mitchell. Generalization as Search. *AI*, 18(2):203–226, 1982.

[28] F. Pan, G. Cong, A. K. H. Tung, J. Yang, and M. Zaki. Carpenter: Finding closed patterns in long biological datasets. In *Proc. of KDD'03*, 2003.

[29] A. Rauzy. Mathematical foundations of minimal cutsets. *IEEE Transactions on Reliability*, 50(4), 2001.

[30] F. Rioult, J. Boulicaut, D. Crémilleux, and J. Besson. Using transposition for pattern discovery from microarray data. In *DMKD*, pages 73–79, 2003.

[31] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proc. of the Int'l Conf. on CAD*, pages 42–47, 1993.

[32] C. Scholl, B. Becker, and A. Brogle. The multiple variable order problem for binary decision diagrams: theory and practical application. In *Proc. of the 2001 Conf. on Asia South Pacifi c Design Automation*, pages 85–90, 2001.

[33] M. Sebag. Delaying the choice of bias: A disjunctive version space approach. In *Proc. of ICML 1996*, pages 444–452.

[34] F. Somenzi. CUDD: CU decision diagram package, 1997. Public software, Colorado University, Boulder.

[35] A. Soulet, B. Crmilleux, and F. Rioult. Condensed representation of emerging patterns. In *Proc. of PAKDD 04*, pages 127–132, 2004.

[36] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *SIGMOD96*, pages 1–12.

[37] G. I. Webb, S. Butler, and D. Newlands. On detecting differences between groups. In *Proc. of KDD03*, pages 256–265, 2003.