International Workshop on Challenges in Web Information Retrieval and Integration

April 8-9, 2005
Tokyo, Japan

IEEE COMPUTER SOCIETY

Sponsored by
The IEEE Computer Society
The Database Society of Japan (DBSJ)
Information Processing Society of Japan (IPSJ)
The Institute of Electronics, Information and Communication Engineering (IEICE)

Supported by
Grant-in-aid for Scientific Research on Priority Area,
Informatics Studies for the Foundation of IT Evolution

Proceedings

International Workshop on Challenges in Web Information Retrieval and Integration

April 8 - 9, 2005
Tokyo, Japan

Co-sponsored by
The IEEE Computer Society
The Database Society of Japan (DBSJ)
Information Processing Society of Japan (IPSJ)
The Institute of Electronics, Information and Communication Engineering (IEICE)

IEEE COMPUTER SOCIETY
http://computer.org
Los Alamitos, California
Washington • Brussels • Tokyo

IEEE Computer Society Order Number P2414
ISBN 0-7695-2414-1
Library of Congress Number 2005928891

Additional copies may be ordered from:

IEEE Computer Society
Customer Service Center
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1314
Tel: + 1 800 272 6657
Fax: + 1 714 821 4641
http://computer.org/cspress
csbooks@computer.org

IEEE Service Center
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
Tel: + 1 732 981 0060
Fax: + 1 732 981 9667
http://shop.ieee.org/store/
customer-service@ieee.org

IEEE Computer Society
Asia/Pacific Office
Watanabe Bldg., 1-4-2
Minami-Aoyama
Minato-ku, Tokyo 107-0062
JAPAN
Tel: + 81 3 3408 3118
Fax: + 81 3 3408 3553
tokyo.ofc@computer.org

Individual paper REPRINTS may be ordered at: reprints@computer.org

Editorial production by Bob Werner

Cover art production by Joe Daigle/Studio Productions

Printed in the United States of America by The Printing House

IEEE Computer Society
Conference Publishing Services
http://www.computer.org/proceedings/

Preface

This volume includes selected papers from the International Workshop on Challenges in Web Information Retrieval and Integration (WIRI 2005), which was held in conjunction with the 21st International Conference on Data Engineering (ICDE 2005), in Tokyo, Japan, April 8-9, 2005.

Web information utilization has become a critical emerging research area due to the exponential increase in the information circulation and dissemination over the Web. Many challenging problems must be solved in order to utilize and exploit web information that is both highly heterogeneous and dynamic in nature. This workshop focuses on the technology for analyzing, integrating and retrieving information on the Web.

In this context, WIRI Workshop has attracted very interesting work which covers crucial and emerging research topics such as querying on the Web, data mining techniques, Web and XML data management, as well as Web applications. There were 47 submissions from 18 countries, and three reviewers were assigned to each paper. The program committee has finally selected 11 regular papers and 20 short papers for presentation at the workshop. The innovative research presented at WIRI 2005 was very interesting and exciting and the Workshop involved lively discussions and fruitful comments.

Moreover, WIRI included a very interesting invited talk by Dr. Daniel Gruhl, who presented "Precognition: Thinking about the the Query before it Happens", a topic which is now emerging and of wide interest. The WIRI chairs thank Dr. Gruhl for his impressive invited talk and his support for WIRI activities.

WIRI chairs thank all the program committee members for their dedicated effort to review papers in their area of expertise and in a timely manner. Their effort was valuable to accommodate high quality papers in the WIRI program. Special thanks to Professor Xiaofeng Meng from the Renmin University of China for his support to the WIRI organization and to Professors Atsuhiro Takasu and Kenro Aihara from the National Institute of Informatics, Japan, for their hard working attitude in organizing the Workshop and in technically supporting its activities. Lastly, we express our sincere gratitude for the support received from Professor Masaru Kitsuregawa of University of Tokyo (ICDE 2005 General Chair) and Professor Masatoshi Yoshikawa of Nagoya University (ICDE 2005 Workshops Co-Chair.)

WIRI Co-Chairs

Jun Adachi (NII)
Wang Shan (Renmin University)
Athena Vakali (Aristotle University)

Table of Contents

International Workshop on Challenges in Web Information Retrieval and Integration — WIRI 2005

Session 4: XML

Session 5: Data and Web Mining (II)

Session 6: Web Information Retrieval (1)

Session 7: Information Integration

Session 8: Information Extraction and Meta Data

Session 9: Web Information Retrieval (II)

Session 10: Web Services

Invited Talk

Precognition: Thinking about the query before it happens

Daniel Gruhl
IBM Almaden Research Center
650 Harry Rd., San Jose, CA, 95120
dgruhl@us.ibm.com

Web Information Retrieval and Integration is one of the most challenging of emerging information retrieval domains. Traditional approaches of returning all of exactly what the user asks for are not feasible; what does a user do with a billion pages in rank order? Identifying relevant documents requires substantially more "thought" go into selecting each result; but the data scale is such that doing the "thinking" at query time as a post-processing of results is not really a viable option.

The solution that WebFountain[5] has pursued is that of examining the pages and really "thinking" about them before the query occurs. There are many kinds of analysis that are easier to do page by page than over a whole corpus. For example, trying to find all of the pages that contain a mention a drugs which contain aspirin is difficult (there are thousands of drugs and brand names that do). A query with this kind of term fan-out is untenable in a high performance system. Instead, consider a program that scanned a document and was able to add a tag `Drug: Aspirin` whenever it found one of the variants. While doing so it could also add `Drug: Cox I inhibitor` and `Drug-type: Analgesic`. This allows high level queries such as `Drug: Cox II inhibitor NEAR Condition: Cardiac Disease` to be processed as a simple 2 term boolean query.

But there is more to finding information than simple boolean queries. In many cases it is not actually the pages that are of interest, but aggregate information on them. With the proper indexes and complex joining functionality it is possible to explore what trends and relationships occur between entities mentioned on webpages. For example, what I say in my blog about a popular music artist is not a good predictor of their popularity, but the trend of the number of all blog mentions is an excellent predictor of sales two weeks later. We have found the same true for certain classes of book sales as well. Understanding the strong relationships between people, places, universities, companies, products, etc. is another example where the preponderance of existence in a particular set of websites is sufficient evidence to propose a linkage.

These higher level annotations and more complex querying capabilities enable not only better point querying, but also open the door for more interesting higher level applications. Examples include exploring trends in discussions and information diffusion[6], identifying templates[4], detecting collusion[3], finding connections[1], finding the connections between the real world and the web[7], and discovering aliases[8], just to name a few. In short, the creation of this semi-structured metadata from unstructured source data allows the system to begin to perform "business intelligence" type queries over the unstructured corpus.

Traditionally, the thought of generating and storing all this metadata has been prohibitive. However, the price on low end storage has been dropping, recently falling below $.31 a gig. This opens the door for this kind of research even in small scale systems.

References

[1] C. Faloutsos, K. S. McCurley, and A. Tomkins. Connection subgraphs in social networks. In *Workshop on Link Analysis, Counterterrorism, and Privacy. SIAM International Conference on Data Mining*, 2004.

[2] D. Gibson. Surfing the web by site. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 496–497, New York, NY, USA, 2004. ACM Press.

[3] D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. 2005.

[4] D. Gibson, K. Punera, and A. Tomkins. The volume and evolution of web page templates. In *Proceedings of the Fourteenth International World Wide Web Conference*, 2005.

[5] D. Gruhl, L. Chavet, D. Gibson, J. Meyer, P. Pattanayak, A. Tomkins, and J. Zien. How to build a webfountain: An architecture for very large-scale text analytics. *IBM Systems Journal*, 43(1):64–77, 2004.

[6] D. Gruhl, R. Guha, D. Liben-Nowell, and A. Tomkins. Information diffusion through blogspace. In *Proceedings of the Thirteenth International World Wide Web Conference*, 2004.

[7] R. Kumar, J. Novak, P. Raghavan, and A. Tomkins. On the bursty evolution of blogspace. *World Wide Web: Internet and Web Information Systems*.

[8] J. Novak, P. Raghavan, and A. Tomkins. Anti-aliasing on the web. In *Proceedings of the Thirteenth International World Wide Web Conference*, New York, New York, 2004.

0-7695-2414-1/05 $20.00 © 2005 IEEE

Session 1

Data and Web Mining

Efficient Method of Combinatorial Item Set Analysis Based on Zero-Suppressed BDDs

Shin-ichi Minato and Hiroki Arimura
Graduate School of Information Science and Technology, Hokkaido University

Abstract

Manipulation of large-scale combinatorial data is one of the important fundamental technique for web information retrieval, integration, and mining. In this paper, we propose a new approach based on BDDs (Binary Decision Diagrams) for database analysis problems. BDDs are graph-based representation of Boolean functions, now widely used in system design and verification area. Here we focus on Zero-suppressed BDDs (ZBDDs), a special type of BDDs, which are suitable for handling large-scale sets of combinations. Using ZBDDs, we can implicitly enumerate combinatorial item set data and efficiently compute set operations over the ZBDDs. We present some encouraging experimental results of frequent item set mining problem for practical benchmark examples, some of which have never been generated by previous method.

1. Introduction

Manipulation of large-scale combinatorial data is one of the fundamental technique for web information retrieval integration, and mining[16]. In particular, frequent item set analysis is important in many tasks that try to find interesting patterns from web documents and databases, such as association rules, correlations, sequences, episodes, classifiers, and clusters. Since the introduction by Agrawal et al.[2], the frequent item set and association rule analysis have been received much attentions from many researchers, and a number of papers have been published about the new algorithms or improvements for solving such mining problems[7, 9, 17].

In this paper, we propose a new approach based on BDDs (Binary Decision Diagrams) for database analysis problems. BDDs are graph-based representation of Boolean functions, now widely used in system design and verification area. Here we focus on Zero-suppressed BDDs (ZBDDs), a special type of BDDs, which are suitable for handling large-scale sets of combinations. Using ZBDDs, we can implicitly enumerate combinatorial item set data and efficiently compute set operations over the ZBDDs.

For a related work, *FP-Tree*[9] is recently received a great deal of attention because it supports fast manipulation of large-scale item set data using compact tree structure on the main memory. Our ZBDD-based method is a similar approach to handle sets of combinations on the main memory, but will be more efficient in the following points:

- ZBDDs are a kind of DAGs for representing item sets, while FP-Trees are tree representation for the same objects. In general, DAGs can be more compact than trees.
- ZBDD-based method provides not only compact data structures but also efficient item set operations written in a simple mathematical set algebra.

We present some encouraging experimental results of frequent item set mining problem for practical benchmark examples, some of which have never been generated by previous method.

Recently, the data mining methods are often discussed in the context of Inductive Databases[4, 12], the integrated processes of knowledge discovery. In this paper, we place the ZBDD-based method as a basis of integrated discovery processes to efficiently execute various operations finding interest patterns and analyzing information involved in large-scale combinatorial item set databases.

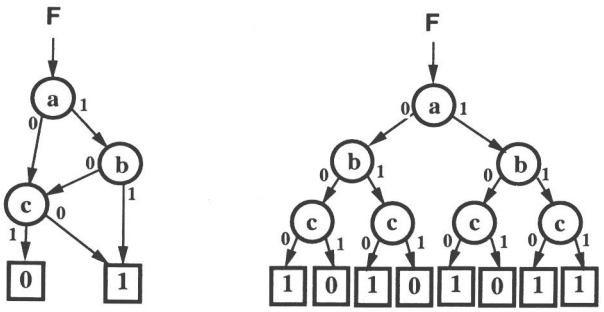
2. BDDs and ZBDDs

2.1. BDDs

BDD is a directed graph representation of the Boolean function, as illustrated in Fig. 1(a). It is derived by reducing a binary tree graph representing recursive *Shannon's expansion*, indicated in Fig. 1(b). The following reduction rules yield a *Reduced Ordered BDD (ROBDD)*, which can efficiently represent the Boolean function. (see [5] for details.)

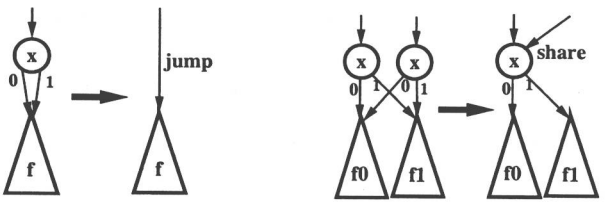
- Delete all redundant nodes whose two edges point to the same node. (Fig. 2(a))

0-7695-2414-1/05 $20.00 © 2005 IEEE



(a) BDD. (b) Binary tree.

Figure 1. BDD and binary tree: $F = (a \wedge b) \vee \overline{c}$ **.**



(a) Node deletion. (b) Node sharing.

Figure 2. Reduction rules of ordinary BDDs

- Share all equivalent sub-graphs. (Fig. 2(b))

ROBDDs provide canonical forms for Boolean functions when the variable order is fixed. Most research on BDDs are based on the above reduction rules. In the following sections, ROBDDs will be referred to as BDDs (or ordinary BDDs) for the sake of simplification.

As shown in Fig. 3, a set of multiple BDDs can be shared each other under the same fixed variable ordering. In this way, we can handle a number of Boolean functions simultaneously in a monolithic memory space.

Using BDDs, we can uniquely and compactly represent many practical Boolean functions including AND, OR, parity, and arithmetic adder functions. Using Bryant's algorithm[5], we can efficiently construct a BDD for the result of a binary logic operation (i.e. AND, OR, XOR), for given a pair of operand BDDs. This algorithm is based on hash table techniques, and the computation time is almost linear to the data size unless the data overflows the main memory. (see [14] for details.)

Based on these techniques, a number of BDD packages have been developed in 1990's and widely used for large-scale Boolean function manipulation, especially popular in VLSI CAD area.

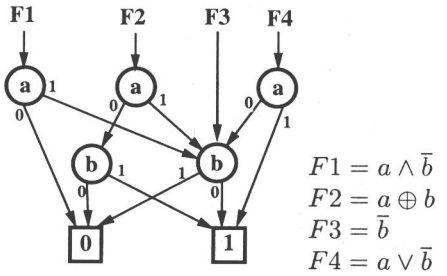


Figure 3. Shared multiple BDDs.

2.2. Sets of Combinations and ZBDDs

BDDs are originally developed for handling Boolean function data, however, they can also be used for implicit representation of sets of combinations. Here we call "sets of combinations" for a set of elements each of which is a combination out of n items. This data model often appears in real-life problems, such as combinations of switching devices(ON/OFF), fault combinations, and sets of paths in the networks.

A combination of n items can be represented by an n-bit binary vector, $(x_1 x_2 \ldots x_n)$, where each bit, $x_k \in \{1, 0\}$, expresses whether or not the item is included in the combination. A set of combinations can be represented by a list of the combination vectors. In other words, a set of combinations is a subset of the power set of n items.

A set of combinations can be mapped into Boolean space by using n-input variables for each bit of the combination vector. If we choose any one combination vector, a Boolean function determines whether the combination is included in the set of combinations. Such Boolean functions are called *characteristic functions*. The set operations such as union, intersection, and difference can be performed by logic operations on characteristic functions.

By using BDDs for characteristic functions, we can manipulate sets of combinations efficiently. They can be generated and manipulated within a time roughly proportional to the BDD size. When we handle many combinations including similar patterns (sub-combinations), BDDs are greatly reduced by node sharing effect, and sometimes an exponential reduction benefit can be obtained.

Zero-suppressed BDD (ZBDD)[13, 15] is a special type of BDDs for efficient manipulation of sets of combinations. ZBDDs are based on the following special reduction rules.

- Delete all nodes whose 1-edge directly points to the 0-terminal node, and jump through to the 0-edge's destination, as shown in Fig. 4.
- Share equivalent nodes as well as ordinary BDDs.

Notice that we do not delete the nodes whose two edges point to the same node, which used to be deleted by the

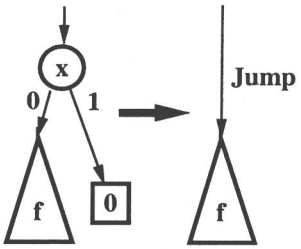


Figure 4. ZBDD reduction rule.

$S(abc)$:

abc	S
000	0
100	1
010	1
110	0
001	0
101	0
011	0
111	0

$S(abcd)$:

abcd	S
0000	0
1000	1
0100	1
1100	0
0010	0
1010	0
0110	0
1110	0
0001	0
1001	0
0101	0
1101	0
0011	0
1011	0
0111	0
1111	0

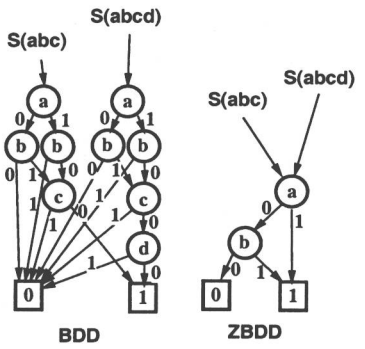


Figure 5. Example of ZBDD effect.

original rule. The zero-suppressed deletion rule is asymmetric for the two edges, as we do not delete the nodes whose 0-edge points to a terminal node. It is proved that ZBDDs are also gives canonical forms as well as ordinary BDDs under a fixed variable ordering.

Here we summarise the features of ZBDDs.

- In ZBDDs, the nodes of irrelevant items (never chosen in any combination) are automatically deleted by ZBDD reduction rule. In ordinary BDDs, irrelevant nodes still remain and they may spoil the reduction benefit of sharing nodes. (An example is shown in Fig. 5.)
- ZBDDs are especially effective for representing sparse combinations. For instance, sets of combinations selecting 10 out of 1000 items can be represented by ZBDDs up to 100 times more compact than ordinary BDDs.
- Each path from the root node to the 1-terminal node corresponds to each combination in the set. Namely, the number of such paths in the ZBDD equals to the number of combinations in the set. In ordinary BDDs, this property does not always hold.

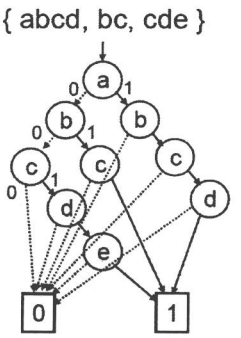


Figure 6. Explicit representation by ZBDD.

"$\emptyset$"	Returns empty set. (0-termial node)
"**1**"	Returns the set of only null-combination. (1-terminal node)
P.top	Returns the item-ID at the root node of P.
P.offset(v)	Selects the subset of combinations each of which does not include item v.
P.onset(v)	Selects the subset of combinations including item v, and then delete v from each combination.
P.change(v)	Inverts existence of v (add / delete) on each combination.
$P \cup Q$	Returns union set.
$P \cap Q$	Returns intersection set.
$P - Q$	Returns difference set. (in P but not in Q.)
P.count	Counts number of combinations.

Table 1. Primitive ZBDD operations

- When no equivalent nodes exist in a ZBDD, that is the worst case, the ZBDD structure explicitly stores all items in all combinations, as well as using an explicit linear linked list data structure. An example is shown in Fig. 6. Namely, (the order of) ZBDD size never exceeds the explicit representation. If more nodes are shared, the ZBDD is more compact than linear list. Ordinary BDDs have larger overhead to represent sparser combinations while ZBDDs have no such overhead.

Table 1 shows the most of primitive operations of ZBDDs. In these operations, $\emptyset$, **1**, $P.top$ are executed in a constant time, and the others are almost linear to the size of graph. We can describe various processing on sets of combinations by composing of these primitive operations.

Data name	$\#I$	$\#T$	$avg\|T\|$	$avg\|T\|/\#I$
T40I10D100K	942	100,000	39	4.14%
mushroom	119	8,124	23	19.32%
BMS-WebView-1	497	59,602	2	0.40%
basket	13,103	41,373	9	0.06%

Table 2. Statistics of typical benchmark data.

3. ZBDD-based Database Analysis

In this section, we discuss the method of manipulating large-scale item set databases using ZBDDs. Here we consider binary item set databases, each record of which holds a combination of items chosen from a given item list. Such a combination is called a *tuple* (or a *transaction*).

For analyzing those large-scale tuple databases efficiently, basic problems of data mining, such as *frequent item set mining*[3] and *maximum frequent item set mining*[6], are very important and they have been discussed actively in last decade. Recently, graph-based methods, such as FP-Tree[9], are received a great deal of attention, since they can quickly manipulate large-scale tuple data by constructing compact graph structure on the main memory. ZBDD technique is a similar approach to handle sets of combinations on the main memory, so we hope to apply ZBDD-based method effectively in this area.

3.1. Property of Practical Databases

Table 2 shows the basic statistics of typical benchmark data[7] often used for data mining/analysis problems. $\#I$ shows the number of items used in the data, $\#T$ is the number of tuples included in the data, $avg|T|$ is the average number of items per tuple, and $avg|T|/\#I$ is the average appearance ratio of each item. From this table, we can observe that the item's appearance ratio is very small in many cases. This is reasonable as considering real-life problems, for example, the number of items in a basket purchased by one customer is usually much less than all the items displayed in a shop. For another example, the number of links from one web page is much less than all the web pages in the network. This observation means that we often handle very sparse combinations in many practical data mining/analysis problems, and in such cases, the ZBDD reduction rule is extremely effective. If the average appearance ratio of each item is 1%, ZBDDs may be more compact than ordinary BDDs up to 100 times. In the literature, there is a first report by Jiang et al.[10] applying BDDs to data mining problems, but the result seems not excellent due to the overhead of ordinary BDDs. We must use ZBDDs in stead of ordinary BDDs for success in many practical data mining/analysis problems.

tuple	frequency	F_2	F_1	F_0
abc	5 (101)	1	0	1
ab	3 (011)	0	1	1
bc	2 (010)	0	1	0
c	1 (001)	0	0	1

F_0 = {abc, ab, c}
F_1 = {ab, bc}, F_2 = {abc}

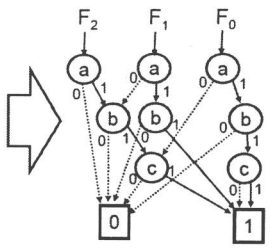


Figure 7. ZBDD vector for tuple-histogram.

3.2. Tuple-Histograms based on ZBDDs

A *Tuple-histogram* is the table for counting the number of appearance of each tuple in the given database. In practical databases, the same tuple often appears many times. For example, "BMS-WebView-1" in Table 2 includes 59,602 records, and the most frequent tuple appears 1,533 times in the records. The top 10 frequent tuples appears 8,404 times in total. (Shares 14% in the records.)

Here we present a method of representing tuple-histograms by using ZBDDs. Since ZBDDs are representation of sets of combinations, a simple ZBDD distinguishes only existence of each tuple in the databases. In order to represent the numbers of tuple's appearances, we decompose the number into m-digits of ZBDD vector $\{F_0, F_1, \ldots, F_{m-1}\}$ to represent integers up to $(2^m - 1)$, as shown in Fig. 7. Namely, we encode the appearance numbers into binary digital code, as F_0 represents a set of tuples appearing odd times (LSB = 1), F_1 represents a set of tuples whose appearance number's second lowest bit is 1, and similar way we define the set of each digit up to F_{m-1}.

In the example of Fig. 7, The tuple frequencies are decomposed as: $F_0 = \{abc, ab, c\}$, $F_1 = \{ab, bc\}$, $F_2 = \{abc\}$, and then each digit can be represented by a simple ZBDD. The three ZBDDs are shared their sub-graphs each other.

Now we explain the procedure for constructing a ZBDD-based tuple-histgram from given tuple database. We read a tuple data one by one from the database, and accumulate the single tuple data to the histogram. More concretely, we generate a ZBDD of T for a single tuple picked up from the database, and accumulate it to the ZBDD vector. The ZBDD of T can be obtained by starting from "**1**" (a null-combination), and applying "Change" operations several times to join the items in the tuple. Next, we compare T and F_0, and if they have no common parts, we just add T to F_0. If F_0 already contains T, we eliminate T from F_0 and carry up T to F_1. This ripple carry procedure continues until T and F_k have no common part. After finishing accu-

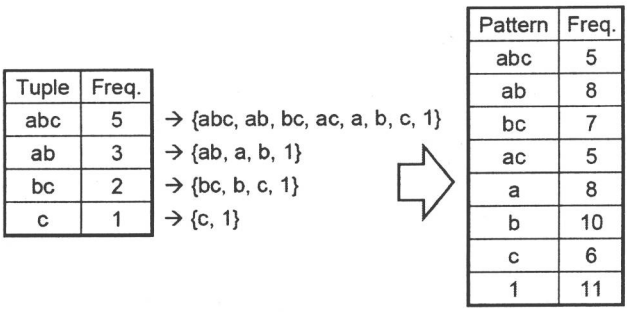


Figure 8. Tuple- and pattern-histogram.

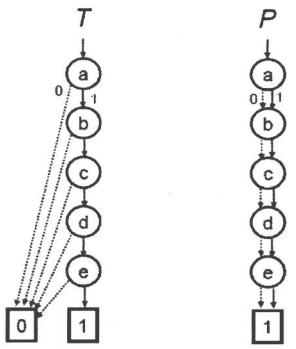


Figure 9. ZBDDs for a tuple and all sub-patterns

mulations for all data records, the tuple-histogram is completed.

Using the notation $F.\mathrm{add}(T)$ for addition of a tuple T to the ZBDD vector F, we describe the procedure of generating tuple-histogram F_T for given database D.

$F_T = \mathbf{0}$
forall $T \in D$ **do**
  $F_T = F_T.\mathrm{add}(T)$
return F_T

When we construct a ZBDD vector of tuple-histogram, the number of ZBDD nodes in each digit is bounded by total appearance of items in all tuples. If there are many partially similar tuples in the database, the sub-graphs of ZBDDs are shared very well, and compact representation is obtained. The bit-width of ZBDD vector is bounded by $\log S_{max}$, where S_{max} is the appearance of most frequent items.

Once we have generated a ZBDD-based tuple-histogram, it is easy to extract the set of frequent tuples which appears more than α times. By encoding the given threshold α into binary code, we can compose an algorithm of bit-wise arithmetic comparison between α and F_T based on ZBDD operations. After execution of those ZBDD operations, the result of frequent tuples can be obtained as a ZBDD. The computation time is almost linear to total ZBDD size.

3.3. Pattern-Histograms based on ZBDDs

In this paper, a *pattern* means a subset of items included in a tuple. A *pattern-histogram* is the table for counting the number of appearance of each patterns in any tuple in the given database. An example is shown in Fig. 8.

In general, a tuple of k items includes 2^k patterns, so computing a pattern-histogram is much harder than computing a tuple-histogram. In many cases, it is difficult to generate a complete pattern-histogram for a practical size of tuple database. Therefore, conventional methods extract only frequent patterns which appears more than α-times, for a given thresholds α, within a feasible computation time and space[7].

Using the ZBDD-based data structure, we may have more compact representation than previous methods, as a number of similar patterns can be shared in ZBDDs, and in some cases, this makes possible to generate complete pattern-histograms which have never succeeded in previous methods. Figure 9 shows a ZBDD for a tuple $T = abcde$ with five items and a ZBDD representing a set of all 32 patterns $P = \{1, a, b, c, d, e, ab, ac, bc, cd, abc, \ldots, abcde\}$ included in T. Clearly we can see that 2^k patterns in a k-item tuple can be represented by only k nodes of ZBDDs. As well as generating tuple-histograms, we can generate pattern-histograms by accumulating such a single ZBDD P for a set of patterns one by one, Here we summarise the procedure for computing a pattern-histogram F_P from a given database D as follows.

$F_P = \mathbf{0}$
forall $T \in D$ **do**
  $P = T$
  forall $v \in T$ **do**
    $P = P \cup P.\mathrm{onset}(v)$
  $F_P = F_P.\mathrm{add}(P)$
return F_P

Unfortunately, ZBDDs grows larger as repeating accumulations, and eventually may overflow the memory for some large examples. While tuple-histgrams are bounded by the total items in the tuples, pattern-histograms are not bounded and so many patterns will be generated.

However, if we have succeeded in generating a ZBDD-based pattern-histogram for a given instance, we can enjoy

very powerful data processing by using efficient ZBDD operations. It is interesting and important how large-scale instances we can generate complete pattern-histograms. The experimental results will be shown in later section.

In addition, we present an alternative procedure for generating ZBDD-based pattern-histograms. We can generate a pattern-histgram F_P from a complete tuple-histogram F_T.

$F_P = F_T$
forall $v \in F_T$ **do:**
  $F_P = F_P.\mathrm{add}(F_P.\mathrm{onset}(v))$
return F_P

We have not determined which algorithm is faster in practical environments. Anyway, the final form of ZBDD vectors must be the same if the two algorithms are computing for the same instance.

3.4. Utilities of Tuple/Pattern-Histograms

Once we generate tuple-/pattern-histograms using ZBDDs, various operations can be executed efficiently. We show several examples in this section. Suppose that we have obtained $F : \{F_0, F_1, \ldots, F_{m-1}\}$, the ZBDD vector representing a tuple- or pattern-histogram.

- We can efficiently extract a subset of tuples/ patterns including a given item or sub-pattern P.

$S = \bigcup F_k$
forall $v \in P$ **do:**
  $S = S.\mathrm{onset}(v).\mathrm{change}(v)$
return S

  Inversely, we can extract a subset of tuples/ patterns not satisfying the given conditions. It is easily done by computing $\bigcup F_k - S$. After extracting a subset, we can quickly count a number of tuples/patterns by using a primitive ZBDD operation S.count. The computation time is linearly bounded by ZBDD size, not depending on the amount of tuple/pattern counts.

- For given α, we can extract all frequent tuples/patterns appearing more than α times. Computation time is almost linear to the ZBDD size. Repeating this procedure with different α's, we can determine the threshold α_m to pick up the top m frequent tuples/patterns. After generating ZBDD-based histograms, it is quite easy to extract frequent sets with different α's, while previous methods need almost recomputing again for each α.
- From ZBDD-based histograms, we can efficiently calculate indexes, such as *Support* and *Confidence*, which are often used in probabilistic/ statistic analysis and machine learning area.

Data name	$\#T$	$total\|T\|$	|ZBDD|	Time(s)
T10I4D100K	100,000	1,010,228	552,429	43.2
T40I10D100K	100,000	3,960,507	3,396,395	895.0
chess	3,196	118,252	40,028	1.4
connect	67,557	2,904,951	309,075	58.1
mushroom	8,124	186,852	8,006	1.5
pumsb	49,046	3,629,404	1,750,883	188.5
pumsb_star	49,046	2,475,947	1,324,502	123.6
BMS-POS	515,597	3,367,020	1,350,970	895.0
BMS-WebView-1	59,602	149,639	46,148	18.3
BMS-WebView-2	77,512	358,278	198,471	138.0
accidents	340,183	11,500,870	3,877,333	107.0

Table 3. Generation of tuple-histograms.

ZBDD nodes	Time(s)	#Pattern
513,762	214.0	(> 2G)

Table 4. Pattern-histogram for "mushroom".

A feature of ZBDD-based method is to construct powerful data structure on the main memory, and we can interactively execute various queries to the database. Moreover, it is very interesting that the queries can be specified by mathematical set operations.

4. Experimental Results

4.1. Generation of Tuple-Histograms

For evaluation of our method, we conducted experiments to construct ZBDD-based tuple- and pattern-histograms for typical benchmark examples[8] used in data mining/analysis problems.

We used a Pentium-4 PC, 800MHz, 512MB of main memory, with SuSE Linux 9. We can deal with up to 10,000,000 nodes of ZBDDs in this machine. Table 3 shows the results of generating tuple-histograms. In this table, $\#T$ shows the number of tuples, $total|T|$ is the total of tuple sizes (total appearances of items), and |ZBDD| is the number of ZBDD nodes for the tuple-histograms. We can see that tuple-histograms can be constructed for all instances in a feasible time and space. The ZBDD sizes are almost same or less than $total|T|$.

4.2. Generation of Pattern-Histograms

Next we tried generating pattern-histograms for the same benchmark set. Within an available memory space (up to 10,000,000 ZBDD nodes), we succeeded in constructing a complete pattern-histgram only for "mushroom", which is a

Threshold α	Time(s)	#Pattern
81	22.60	91,273,269
40	67.96	295,117,613
16	244.06	1,176,182,553
8	494.05	1,983,493,667
4	891.31	(> 2G)
1	1,322.48	(> 2G)

Table 5. FP-Tree-based method for "mushroom".

relatively small instance (Table 4). In this case, the pattern-histogram requires about 65 times more ZBDD nodes than the tuple-histogram for the same data.

The "mushroom" pattern-histogram is implicitly representing at least 2,000,000,000 patterns. (The counter overflows the range of 32-bit integers.) The number of ZBDD nodes are 4,000 times smaller than number of patterns. This shows a great benefit of compression ratio obtained by ZBDDs.

To compare with a previous method, we applied a frequent pattern mining program based on FP-Tree[9], to extract the set of frequent patterns appearing more than α times, for the same example "mushroom". The results are shown in Table 5. For smaller α's, more patterns are extracted, and the computation time increases up to hundreds or thousands of seconds. Notice that the FP-Tree-based method only extracts a set of frequent patterns for a given α, but does not generates a complete histogram for all possible patterns. Our ZBDD-based method generates a complete histogram for all patterns in 214 seconds, and it corresponds to computing frequent pattern sets for all α's at once. Based on ZBDD data structure, we need only 48 second of additional time to extract frequent pattern sets for all different α's from 1 to 100, namely, only 0.48 second needed for each extraction in average. The result shows that ZBDD-based implicit method is especially effective for handling a huge number of patterns.

4.3. Utility of ZBDD-Based Histograms

We conducted another experiment of the set operations on ZBDD data structure. First we generate a ZBDD for S, a set of all the patterns contained in "mushroom", and then compute S.onset(v) for an item v. We tested this onset operation for all different 119 items, and only 0.10 second needed for each operation in average. The other primitive operations, such as offset, change, union, intersection, etc., are in the almost same range of computation time. Consequently, ZBDD-besed implicit method is very powerful for representing huge number of patterns and efficiently applying various set operations for database analysis.

Data name	$\#T_{all}$	$\#T_{done}$	ratio%	|ZBDD|	Time(s)
T10I4D100K	100,000	43,395	43.40	9,968,062	1,711
T40I10D100K	100,000	740	0.74	9,863,736	123
chess	3,196	703	22.00	8,242,212	919
connect	67,557	2,095	3.10	9,146,429	2,502
mushroom	8,124	8,124	100.00	513,762	214
pumsb	49,046	61	0.12	7,018,198	89
pumsb_star	49,046	288	0.59	8,343,418	375
BMS-POS	515,597	9,837	1.90	9,679,161	266
BMS-WebView-1	59,602	17,757	29.79	9,372,436	134
BMS-WebView-2	77,512	39,045	50.37	9,515,386	1,113
accidents	340,183	133	0.04	8,394,811	223

Table 6. Pattern-histograms for sampling data.

Table 6 shows the results for generating pattern-histograms for other larger instances. Here we cannot generate complete pattern-histograms for all tuples, but we can construct them for a part of tuples in the database. In this experiment, we started from first line of the data file, and stopped at the line just before memory overflow. The table shows the number of tuples we completed. The partial results are still meaningful as sampling computation. For smaller or medium size of instances, we can generate pattern-histograms for more than 20% of tuples in the database. Here the tuples are selected in a deterministic manner, but we may shuffle the order of tuples in real applications.

5. Conclusion

In this paper, we presented a new method of using ZBDDs for database analysis problems. Our work is just starting now, and we have many future works to be considered, such as ZBDD variable ordering problem for reducing graph size, and more efficient implementation of ZBDD set operations.

We expect that it would be too memory-consuming to construct ZBDDs of the complete pattern-histograms for the large-scale benchmarks, besides "mushroom". However, hopefully we will be able to handle those practical size of databases by using well-known improvement techniques, such as preprocessing of pruning not frequent items and patterns, or handling only maximum item set data[6], etc. ZBDD-based method will be useful as a fundamental techniques for various processing of database analysis, and will be utilized for web information retrieval and integration.