N-gram Analysis Based on Zero-suppressed BDDs

Ryutaro Kurai, Shin-ichi Minato, and Thomas Zeugmann

Division of Computer Science
Hokkaido University, N-14, W-9, Sapporo 060-0814, Japan
{ryu,minato,thomas}@mx-alg.ist.hokudai.ac.jp

Abstract. In the present paper, we propose a new method of n-gram analysis using ZBDDs (Zero-suppressed BDDs). ZBDDs are known as a compact representation of combinatorial item sets. Here, we newly apply the ZBDD-based techniques for efficiently handling sets of sequences. Using the algebraic operations defined over ZBDDs, such as union, intersection, difference, etc., we can execute various processings and/or analyses for large-scale sequence data. We conducted experiments for generating n-gram statistical data for given real document files. The obtained results show the potentiality of the ZBDD-based method for the sequence database analysis.

1 Introduction

One of the important data model that has been used for text analysis are n-grams (cf., e.g., [2, 7, 8, 9]). Recently, n-grams have been used not only for text analysis but also for text indexing in some search engines [1]. If we can compactly represent n-gram data and efficiently manipulate them, it will greatly facilitate text database analysis and machine learning applications.

In the present paper, we propose a method of n-gram computation with a new sequence data structure based on Zero-suppressed BDDs. BDDs (Binary Decision Diagrams) are graph-based representations of Boolean functions, now widely used in system design and verification. Zero-suppressed BDDs (ZBDDs) are a special type of BDDs that 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 n-gram computations, we need to manipulate sets of "sequences," which is a more complicated data model than sets of "combinations." We present a method of manipulating sets of sequences using ZBDDs and generate efficiently n-gram data for given sequence databases. We have implemented a prototype system and show experimental results to evaluate our new n-gram analysis method.

2 ZBDDs and Item Set Manipulation

Within this section, we briefly describe the basic techniques of ZBDDs for representing sets of combinations efficiently.

2.1 Sets of Combinations and their Representation

A *set of combinations* consists of the elements each of which is a combination of a number of items. There are 2^n combinations that can be chosen from n items, so we have 2^{2^n} variations of sets of combinations. For example, for a domain of five items a, b, c, d, e, we can express examples of sets of combinations as: $\{ab, e\}$, $\{abc, cde, bd, acde, e\}$, $\{1, cd\}$, $\emptyset$. Here "1" denotes a combination of null items, and "$\emptyset$" means the empty set. Sets of combinations are one of the basic data structures for handling combinatorial problems. They often appear in real-life problems, such as combinations of switching devices, sets of faults, paths in the networks, etc., and of course, they can be used for representing frequent item set data.

a	b	c	F	
0	0	0	0	
0	0	1	1	$\rightarrow c$
0	1	0	0	
0	1	1	1	$\rightarrow bc$
1	0	0	0	
1	0	1	0	
1	1	0	1	$\rightarrow ab$
1	1	1	1	$\rightarrow abc$

As a Boolean function:
$F = ab + \overline{a}c$

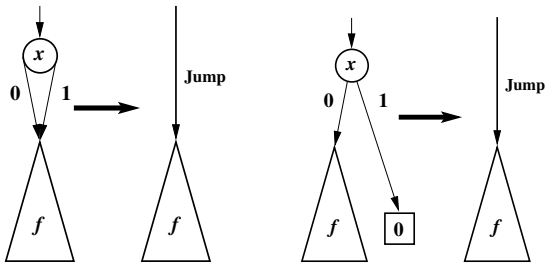
As a set of combinations:
$F = \{ab,\ abc,\ bc,\ c\}$

Fig. 1. Correspondence of Boolean functions and sets of combinations.

A set of combinations can be mapped into Boolean space of n input variables. For example, Fig. 1 shows the truth table of the Boolean function $(ab+\overline{a}c)$, but it also represents the set of combinations $\{ab, abc, bc, c\}$. Such Boolean functions are called *characteristic functions* for the sets of combinations. Using BDD manipulation for characteristic functions, we can implicitly represent and manipulate large-scale sets of combinations. In addition, we can enjoy more efficient manipulation using "Zero-suppressed BDDs" (ZBDD) (cf. [4]), which are a special type of BDDs optimized for handling sets of combinations.

ZBDDs are based on the reduction rule different from the one used in ordinary BDDs. As illustrated in Fig. 2(a), the ordinary reduction rule deletes the nodes whose two edges point to the same node. However, in ZBDDs, we do not delete such nodes but delete another type of nodes whose 1-edge directly points to the 0-terminal node, as shown in Fig. 2(b).

In ZBDDs, a 0-edge points to the subset (cofactor) of combinations not including the decision variable x, and a 1-edge points to the subset (cofactor) of combinations including x. If the 1-edge directly points to the 0-terminal node, it means that the item x never appears in the set of combinations. The Zero-suppressed reduction rule automatically deletes such a node with respect to the



(a) Ordinary BDD. (b) ZBDD.

Fig. 2. Different reduction rules for BDD and ZBDDs.

irrelevant item x, and thus ZBDDs more compactly represent sets of combinations than ordinary BDDs do.

The detailed techniques of ZBDD manipulation are described in the articles [4, 5]. A typical ZBDD package supports cofactoring operations to traverse 0-edge or 1-edge, and binary operations between two sets of combinations, such as union, intersection, and difference. The computation time for each operation is almost linear to the number of ZBDD nodes related to the operation.

2.2 Item Set Histograms and ZBDD Vectors

An *item set histogram* is the table for counting the number of appearances of each item combination in the given database. An example of an item set histogram is shown in Fig. 3. This is just a compressed table of the database to combine the same tuples appearing more than once into one line with the frequency.

Record ID	Comb.
1	*abc*
2	*ab*
3	*abc*
4	*bc*
5	*ab*
6	*abc*
7	*c*
8	*abc*
9	*abc*
10	*ab*
11	*bc*

Original Database

Comb.	Freq.
abc	5
ab	3
bc	2
c	1

Item set histogram

Fig. 3. Database example and item set histogram.

Comb.	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

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

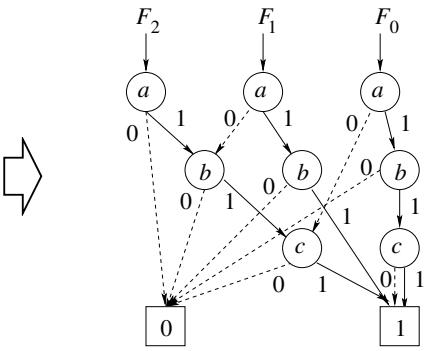


Fig. 4. ZBDD vector for item set histogram.

Our n-gram data structure is based on the item set histogram representation [6] using ZBDDs. Since ZBDDs are representation of sets of combinations, a simple ZBDD distinguishes only the existence of each combination in the database. In order to represent the numbers of combination'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. 4. Namely, we encode the appearance numbers into binary digital code, where F_0 represents a set of tuples appearing odd times (LSB = 1), F_1 represents a set of combinations whose appearance number has a 1 in the second lowest bit, and similarly we define the set of each digit up to F_{m-1}.

In the example of Fig. 4, the item set 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 share their sub-graphs to one another.

When we construct a ZBDD vector of an item set histogram, the number of ZBDD nodes in each digit is bounded by the total appearance of items in all combinations. If there are many partially similar combinations in the database, the sub-graphs of ZBDDs are shared very well, and a 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 vector for the item set histogram, various operations can be executed efficiently. Here are the instances of operations used in our pattern mining algorithm.

- H.factor0(v): Extracts sub-histogram of combinations not including item v.
- H.factor1(v): Extracts sub-histogram of combinations including item v and then delete v from the combinations. (also considered as the quotient of H/v)
- $v \cdot H$: Attaches an item v on each combination in the histogram F.
- $H_1 + H_2$: Generates a new item set histogram with sum of the frequencies of corresponding tuples.
- H.tuplecount: The number of tuples appearing at least once.

These operations can be composed as a sequence of ZBDD operations and the result is also compactly represented by a ZBDD vector. The computation time is roughly linear in the total ZBDD sizes. For a more detailed description of the techniques of ZBDD vector manipulation, we refer the reader to [6].

3 ZBDD-Based Representation for Sets of Sequences

Next we explain, in some more detail, our method of generating n-gram data representations using ZBDDs.

3.1 Sets of Sequences and Sets of Combinations

First, we describe the way to extend the representation of sets of combinations to sets of sequences. In our method, we define an item for each symbol (or character) used in each position of sequences. Namely, the item represents not only a symbol but also its position in the sequence. An example is given below.

Sequences	Combinations
AABC	$a_1a_2b_3c_4$
CABC	$c_1a_2b_3c_4$

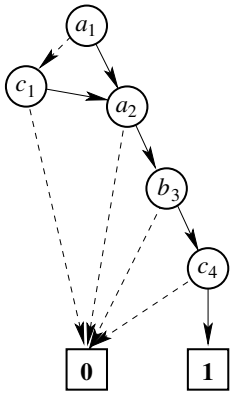


Fig. 5. ZBDDs example which expresses sets of sequences.

In the combination for the sequence "AABC," the item a_1 represents the symbol "A" at the first (leftmost) position. Similarly, c_4 express the symbol "C" at the 4th position. In our method, if the same symbol appears at different positions, we treat them as different items. For example, the sequence "AABC" includes the same symbol "A" at the first and the second position, but we treat

them as a_1 and a_2, i.e., as different items. After such an assignment of items, a set of sequences can be represented by a set of combinations. Then, the set of combinations can be represented by a ZBDD. An example is shown in Fig. 5. Here, the ZBDD represents the set of sequences {AABC, CABC}. In this way, the sets of sequences are decomposed to sets of combinations, and they can be manipulated by ZBDD-techniques.

3.2 *n*-gram Representation

An n-gram data is a histogram of all possible subsequences of length n included in a given sequence. As described in Subsection 2.2, we can compute item set histograms by using ZBDD vectors. So, the ZBDD-based representation for sets of sequences can be extended to sequence histograms by using ZBDD vectors.

3.3 Binary Coding of Symbols

In our method, we use a number of items for the respective positions in the sequences. The number is the product of the number of symbols $|\Sigma|$ and the sequence length m. If we apply this method for a language using not so many symbols such as gene sequence data, the number of items would be feasible. However, for a language that has many symbols such as Japanese Kanji, too many items would be required. To address this problem, we propose a method of using binary coding to save the number of items.

When we assume the three symbols A, B and C for each position, we may use the 2-bit binary coding shown in the following table.

Symbols	Items	Binary code $(x_k^1 x_k^0)$	Encoded combinations
A	a_k	01	x_k^0
B	b_k	10	x_k^1
C	c_k	11	$x_k^1 x_k^0$

Using this encoding, we can express the sequences "AABC" and "CABC" by the following table, and the resulting ZBDD is shown in Fig. 6.

Sequences	Encoded combinations
AABC	$x_1^0 x_2^0 x_3^1 x_4^1 x_4^0$
CABC	$x_1^1 x_1^0 x_2^0 x_3^1 x_4^1 x_4^0$

We may use ASCII code for this purpose. ASCII code expresses conventional western characters with 8bit vectors. We treat one bit as one item. One can distinguish at most 255 symbols with a combination of eight items for each position. Examples are shown below.

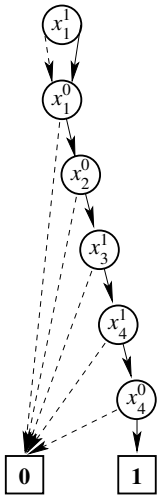


Fig. 6. ZBDDs example which expresses sets of sequences.

Symbols	ASCII (decimal)	Binary code($x^7x^6x^5x^4x^3x^2x^1x^0$)	Encoded combinations
a	97	01100001	$x^6x^5x^0$
b	98	01100010	$x^6x^5x^1$
c	99	01100011	$x^6x^5x^1x^0$
z	122	01111010	$x^6x^5x^4x^3x^1$
A	65	01000001	x^6x^0

If the probability distribution of the symbols is given, we shall be able to design a more efficient binary coding assignment.

4 Experimental Results

4.1 Generating *n*-gram Data Based on ZBDDs

For evaluating the feasibility of our method, we performed experiments for generating ZBDDs of the n-grams for a given text data. We used the English plain text data "Alice in Wonderland." This text consists of over 3,000 lines and 26,000 words, having a total of 138KB of plane text. In our experiments, we used a 3.00GHz Pentium 4 Linux PC with 512MB main memory.

The results of generating n-grams for $n = 1$ through 10 are shown in Table 1. In this table, the column "ZBDD nodes" shows the total number of nodes in the ZBDD vector representing the n-gram. "#Sequences" stands for the number of different n-grams actually found in the text. "Max frequency" shows the number

Table 1. Experimental results

	ZBDD				ruby-Hash
n	ZBDD nodes	#Sequences	Max frequency	Time(sec)	Time(sec)
1	121	26	13,681	3.17	0.28
2	1,428	518	3,813	6.18	0.29
3	9,487	4,750	2,402	9.45	0.41
4	33,199	19,004	500	12.96	1.01
5	73,940	38,913	417	16.73	2.12
6	129,738	57,510	221	22.4	2.83
7	198,311	72,285	211	30.33	3.90
8	283,544	82,891	116	44.02	5.01
9	388,863	90,247	116	63.49	6.05
10	522,175	95,366	58	89.51	7.45

of appearances of the most popular subsequences of length n in the text. Finally, "Time" displays the time needed to generate the data structure ZBDD. The ZBDD structure is generated by VSOP (see [6]), the script language which can manipulate large scale ZBDDs. In this experiment, we used the symbol encoding method with ASCII code, as described in the previous section.

From the result, we can observe that the size of ZBDDs is almost proportional to the product of the length n and the number of sequences. The CPU time is also almost linear in size of the ZBDD.

Then we compare the performance of generating speed between ZBDD and other data structure. As an existing method, we use the Ruby [10] "Hash" class implementation. Ruby is the object oriented script language which is popularly used in web applications. The "Hash" class provides the function of indexing tables, and is included in the ruby standard library. The result of this experiment is the rightmost column "Time" in the Table 1. The result shows that our method is almost ten times slower than the ruby-Hash implementation, so our method is not so effective if we simply generate n-grams. However, ZBDD-based method has an advantage of flexible post processing after generating n-gram data. If we use ruby-Hash data structure we can only operate simple matching of sequences. So we would have to write several ruby codes to calculate complicated operations.

After generating the ZBDD vector of the n-gram, we can easily retrieve the frequent sequences by using algebraic operations over ZBDDs. The additional computation cost is relatively smaller than the cost for generating n-grams. Table 2 shows the top 10 frequent sequences in the 5-gram of "Alice in Wonderland." The most frequent sequence is "alice," and it appears 417 times. The 2nd, 3rd, 4th, and 10th sequences seem a part of same sequence "he/she said the." In this experiment, we removed spaces, special symbols such as question mark, and periods from the text. We only used case-independent alphabets and numbers to create n-grams.

Table 2. Top 10 subsequences in the 5-gram of "Alice in Wonderland."

Rank	Sequence	Appearance
1	"alice"	417
2	"saidt"	265
3	"aidth"	223
4	"idthe"	220
5	"thing"	170
6	"andth"	169
7	"dalic"	163
8	"ofthe"	156
8	"ndthe"	156
10	"esaid"	133

4.2 Post Processing for *n*-gram Data

An advantageous feature of our ZBDD-based method is that it allows flexible post processing after the n-gram data have been generated. The post processing is done by using algebraic operations such as intersection, union, difference, and, additionally, some numeric operations.

For example, the operation that calculates the rank of frequency of the sequences is represented by the following short command scripts. These command scripts are written in VSOP. The uppercase letters are register variables which show sequences of n-grams, while the lowercase letters are symbols which represent each character in n-grams.

```
T1 = {a1 a2 b3 + a1 b2 c3 + ...}
M = T1.MaxVal
print M
print T1 / M
```

Using only four commands we can get the ranking and the frequency of appearance. The first command generates the ZBDD of item sets of n-gram. The three symbols "a1 a2 b3" represent the 3-gram "aab". The meaning of symbol "a1" is as follows: "a" means that symbol "a1" expresses the character "a", and "1" specifies that character "a" exists as 1st character in the 3-gram. Similarly, the three symbols "a1 b2 c3" express another 3-gram, i.e., "abc". Therefore, the variable "T1" contains some 3-grams including "aab", "abc" as item sets. `MaxVal` is the calculation to get the number of the largest coefficient. So, the 2nd command returns the frequency of the most popular item sets as M. And the last line returns the most popular item sets.

For having another example, suppose that we want to obtain the subset of the n-gram data such that the last letter is either "a," "e," "i," "o," or "u." In this case, we first generate the five subsets of the n-gram data such that the first subset includes all n-grams having the last letter "a," the second subset includes

all n-grams having the last letter "e," ..., and the fifth subset includes all n-grams having the last letter "u." Then we apply union operations to combine all the subsets.

These instructions can easily be described in a command script of ZBDD operations as shown below.

```
T2 = {a1 a2 b3 + a1 b2 c3 + ...}
T3 = (T2 / a3) * a3
T4 = (T2 / e3) * e3
T5 = (T2 / i3) * i3
T6 = (T2 / o3) * o3
T7 = (T2 / u3) * u3
T8 = T3 + T4 + T5 + T6 + T7
```

In the first line command, we store all n-grams as T2. T3 to T7 are the subsets including the corresponding letter at the last position. Finally, we get the combination of all subsets T3 to T7 as T8.

Next, we show a good example of applying ZBDD operations after generating n-gram data. We prepared a command script to compare two sets of n-grams. That is, we compare the first half of the text in "Alice in Wonderland" to the second half of it. The corresponding VSOP commands to compare these two sets are displayed below.

```
S1 = {a1 b2 c3 + b2 c3 d4 + ...}
S2 = {c1 d2 e3 + d2 e3 f4 + ...}
C = S1 - S2
M = C.MaxVal
print M
print C / M
```

In the first and the second command we store the first and second half of n-grams as S1 and S2, respectively. Then we store the difference of S1 and S2 as C. Finally, `C.MaxVal` and `C / M` return the frequency of the most popular sequence and the sequence itself, respectively.

The obtained results are displayed in Table 3, 4, and 5 below.

Table 3 and 4 do not differ much, since the same sequences appear many times in the first and the second half of the text. Therefore, we also check the histogram of differences of the histogram. As shown in Table 5 below, the sequence "the queen" appears only in this table. This is caused by the fact that "queen" appears in the story only in the second half of the text (frequently).

5 Conclusions

In the present paper, we have proposed a new method to represent sequences in sets of item sets, i.e., by using ZBDDs. In order to demonstrate the efficiency of this method, we have performed a preliminary experiment of generating an

Table 3. Top 5 subsequences in the 5-gram of "The first half of Alice in Wonderland."

Rank	Sequence	Appearance
1	"alice"	201
2	"thing"	93
2	"saidt"	93
4	"littl"	85
4	"ittle"	85

Table 4. Top 5 subsequences in the 5-gram of "The Second half of Alice in Wonderland."

Rank	Sequence	Appearance
1	"alice"	216
2	"saidt"	172
3	"idthe"	145
4	"aidth"	143
5	"andth"	103

n-gram set for a given text file. The experimental result shows that we can construct the ZBDD-based n-gram data structure in a feasible computation time and memory space. Regrettably, our data structure does not perform faster than previous data structure implemented by Ruby. So we improve our method to utilize ZBDD's effect that share the ZBDD nodes and compress size of data.

Though we restricted ourselves within this paper to generate n-gram statistical data for a given real document file, i.e., "Alice in Wonderland," ZBDDs can also be used to perform more advanced calculations. For example, we may also apply our method to represent a text dictionary and to perform quickly dictionary search operations.

Table 5. Top 5 subsequences in the 5-gram of "Difference between the First half and the Second half of Alice in Wonderland."

Rank	Sequence	Appearance
1	"saidt"	79
2	"idthe"	70
3	"queen"	68
4	"thequ"	65
5	"heque"	65
5	"equee"	65
5	"urtle"	65

It should be noted that our new method can also be applied to analyze and to manipulate DNA data or some semi-structured data such as linguistic data annotated by NLP tools (e.g., [3]). This will be done in the future.

References

[1] B. Hoffmeister and T. Zeugmann: Text Mining Using Markov Chains of Variable Length, In (K.P. Jantke, A. Lunzer, N. Spyratos, Y. Tanaka, eds), *Federation over the Web, International Workshop, Dagstuhl Castle, Germany, May 2005, Revised Selected Papers*, Lecture Notes in Artificial Intelligence Vol. 3847, pp. 1–24, Springer, Feb. 2006.

[2] P. Jokinen, E. Ukkonen: Two algorithms for approximate string matching in static texts, In *Mathematical Foundations of Computer Science 1991, 16th International Symposium, MFCS'91, Kazimierz Dolny, Poland, September 9-13, 1991, Proceedings*, Lecture Notes in Computer Science Vol. 520, 240-248, Springer-Verlag 1991.

[3] T. Kudo, K. Yamamoto, Y. Tsuboi, Y. Matsumoto: Text mining using linguistic information, IPSJ SIG-NLP NL-148, (in Japanese), pp. 65–72, 2002.

[4] S. Minato: Zero-suppressed BDDs for set manipulation in combinatorial problems, In Proc. 30th Design Automation Conference (DAC-93), pp. 272–277, ACM Press, June 1993.

[5] S. Minato: Zero-suppressed BDDs and their applications, International Journal on Software Tools for Technology Transfer (STTT), 3(2):156–170, Springer, May 2001.

[6] S. Minato: VSOP (Valued-Sum-of-Products) Calculator for Knowledge Processing Based on Zero-Suppressed BDDs, In (K.P. Jantke, A. Lunzer, N. Spyratos, Y. Tanaka, eds), *Federation over the Web, International Workshop, Dagstuhl Castle, Germany, May 2005, Revised Selected Papers*, Lecture Notes in Artificial Intelligence Vol. 3847, pp. 40-58, Springer, Feb. 2006.

[7] M. Nagano, and S. Mori: A new method of N-gram statistics for large number of n and automatic extraction of words and phrases from large text data of Japanese, In *Proc. 15th Conference on Computational Linguistics*, Vol. 1, pp. 611–615, Association for Computational Linguistics, Morristown, NJ, USA, 1994.

[8] Y. Tsuboi: Mining frequent substrings, Technical Report of IEICE, NLC2003-47, 2003 (in Japanese).

[9] E. Ukkonen: Approximate string-matching with q-grams and maximal matches, *Theoretical Computer Science*, 92(1):191–211, 1992.

[10] Ruby Home Page http://www.ruby-lang.org/en/