アルゴリズム　32－4
（１９９3.　3.　１８）

論理関数による組合せアルゴリズムの解法　Ｉ

仙　波　一　郎
茨城大学　教養部

矢　島　脩　三
京都大学　工学部　情報工学科

この論文では、ｎ個の要素からｒ個の要素を取り出す組合せのすべてを表現する論理関数とｎ対のバランスのとれたかっこ列のすべてを表現する論理関数を漸化式を用いて導く。つぎに、これらの論理関数を二分決定グラフ（A Binary Decision Diagram, BDD）で記述し、二分決定グラフを効率的に処理するＢＤＤパッケージを使って、少ないメモリで大きい組合せの集合を高速に生成できることを示す。

Combinatorial Algorithms by Boolean Processing I

Ichiro Semba
College of General Education
Ibaraki University

Shuzo Yajima
Department of Information Science
Faculty of Engineering
Kyoto University

In this paper, we consider two combinatorial problems. One is the problem of generating all the r-combinations of the set {1,2,...,n}. The other is the problem of generating all the balanced parentheses sequences of length 2n. The Boolean functions representing them are derived by using the recurrence relation. The Boolean functions are described by a Binary Decision Diagram (BDD). We have shown that a large number of r-combinations of the set {1,2,...,n} (balanced parentheses sequences of length 2n) are expressed with smaller memory and computed at a high speed by the BDD manipulator.

1. Introduction

It is an important problem to manipulate Boolean functions efficiently in such applications as formal design verification, test generation and logic synthesis and so on. Since the efficient manipulation and the representation of Boolean functions are closely related, various representations of Boolean functions have been proposed.

A Binary Decision Diagram(BDD) is a graph representation of Boolean functions[1][2]. A Shared Binary Decision Diagram(SBDD) is an improvement of BDD[3]. They have excellent properties to realize efficient Boolean functions manipulation. Now, BDD manipulators are implemented on workstations[3][4][5] and widely used.

We have considered two problems of generating fundamental combinatorial objects. One is an r-combination of the set $\{1,2,\ldots,n\}$ (r-combination of n for short). The other is a balanced parenteses sequence of length 2n. These generating algorithms are based on the recurrence relation.

An r-combination of n is defined as an unordered selection of r of the set $\{1,2,\ldots,n\}$. The number of all the r-combinations of n is well known as the binomial coefficient, ${}_nC_r=n!/(r!(n-r)!)$. For example, all the 3-combinations of 5 are 123, 124, 125, 134, 135, 145, 234, 235, 245, 345.

A balanced parentheses sequence of length 2n, $p_1p_2\ldots p_{2n}$, is defined as a sequence of n right parentheses and n left parentheses such that the number of left parentheses of $p_1p_2\ldots p_k$ is greater or equal to the number of right parentheses of $p_1p_2\ldots p_k$ for any k $(1\leqq k\leqq 2n)$. The number of all the balanced parentheses sequences of length 2n is shown to be ${}_{2n}C_n/(n+1)$, the Catalan number[6]. For example, all the balanced parentheses sequences of length 6 are ((())), (()()),(())(),()(()),()()().

In this paper, we propose the algoriothms generating these combinatorial objects by using Boolean functions. We have implemented the programs. They are written in C language and are executed efficiently by BDD manipulator on a Sun SPARC station 2 workstation(64MByte).

This paper is organized as follows: Section 2 describes BDD and SBDD. Section 3 describes the algorithm generating all the r-combinations of n. Section 4 describes the algorithm generating all the balanced parentheses sequences of length 2n. Section 5 concludes the paper.

2. A Binary Decision Diagram (BDD) and A Shared Binary Decision Diagram (SBDD)

We assume that a Boolean function with n variables $x_1,x_2,\ldots,x_n$ is denoted by $f(x_1,x_2,\ldots,x_n)$. When some variables x_i of function f with n variables $x_1,x_2,\ldots,x_n$ is replaced by 0(1), the function f is called a restriction of f. Using the Shannon expansion[7], a function f around variable x_i is given by

$$f(x_1,,\ldots,x_i,\ldots,x_n) = x_i f(x_1,\ldots,x_{i-1},1,x_{i+1},\ldots,x_n) + \bar{x}_i f(x_1,\ldots,x_{i-1},0,x_{i+1},\ldots,x_n)$$

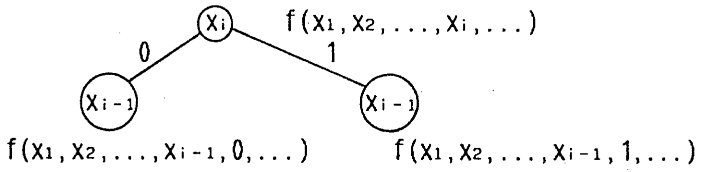
Now, we will consider a graphical representation of a Boolean function $f(x_1,x_2,\ldots,x_n)$.

A Binary Decision Tree (BDT) is a binary tree with two types of nodes, terminal nodes and nonterminal nodes. Terminal nodes are labeled with 0 or 1. Nonterminal nodes are labeled with one of the Boolean variables $x_1,x_2,\ldots,x_n$. Every nonterminal node has exactly two outgoing edges. They are labeled with 0 or 1 and called 0 edge or 1 edge respectively.

By repeating Shannon expansion to a Boolean function $f(x_1,x_2,\ldots,x_n)$ recursively, a BDT corresponding to $f(x_1,x_2,\ldots,x_n)$ is derived. We note that a restriction of f is done according to exactly decreasing order of the index of the variable. That is $x_n,x_{n-1},\ldots,x_2,x_1$.

The process of constructing a BDT is as follows.

We assume variables $x_{i+1},\ldots,x_n$ have already determined 0 or 1. A node x is labeled with x_i and corresponded to a Boolean function $f(x_1,x_2,\ldots,x_i,\ldots)$. A child of the node x is labeled with x_{i-1} and corresponded to a Boolean function $f(x_1,x_2,\ldots,x_{i-1},0,\ldots)$. An edge to the child is 0 edge. Another child of the node x is also labeled with x_{i-1} and corresponded to a Boolean function $f(x_1,x_2,\ldots,x_{i-1},1,\ldots)$. An edge to the child is 1 edge. When the variables $x_1,x_2,\ldots,x_n$ have already been determined and the value of $f(x_1,x_2,\ldots,x_n)$ is 0(1), a child of the node x is a terminal node with labeled 0(1). We note that the root is labeled with x_n and corresponded to a Boolean function $f(x_1,x_2,\ldots,x_n)$.


x_i
$f(x_1,x_2,\ldots,x_i,\ldots)$
0
1
x_{i-1}
x_{i-1}
$f(x_1,x_2,\ldots,x_{i-1},0,\ldots)$
$f(x_1,x_2,\ldots,x_{i-1},1,\ldots)$

By the Shannon expansion, the following equation is obtained.

$$f(x_1,x_2,\ldots,x_i,\ldots) = x_i f(x_1,x_2,\ldots,x_{i-1},1,\ldots) + \bar{x}_i f(x_1,x_2,\ldots,x_{i-1},0,\ldots)$$

A Binary Decision Diagram (BDD) is defined as the directed acyclic graph obtained from the BDT by repeating the following transformations (1),(2),(3) and (4) until they are not applicable.

Transformation (1): When both 0 edge and 1 edge point to the same terminal node, delete the nonterminal node.
Transformation (2): Share isomorphic subgraphs.
Transformation (3): When both 0 edge and 1 edge point to the same nonterminal node, delete the nonterminal node.
Transformation (4): Leave only one terminal node with labeled 0(1).

A Shared Binary Decision Diagram (SBDD) is an improvement of BDD. While a BDD represents a single Boolean function, a SBDD represents multiple Boolean functions by sharing sub-graphs of BDDs representing the same function.

3. The problem of generating all the r-combinations of n

In this section, we consider to generate all the r-combinations of n. An r-combination of n is defined as an unordered selection of r of the set {1,2, ...,n}. For example, for n=5 and r=3:

All the 3-combinations of 5 are 123,124,125,134,135,145,234,235,245,345.

The number of all the r-combinations of n is well known as the binomial coefficient, ${}_nC_r$. It can be shown that ${}_nC_r=n!/(r!(n-r)!)$. We note that all the r-combination of n are often used to generate other combinatorial objects.

Now we will derive the Boolean function representing all the r-combinations of n. First, we define Boolean variables $x_1,x_2,\ldots,x_n$ as follows.

$$x_i = \begin{array}{ll} 1, & \text{if the value } i \text{ is taken out of } n \text{ values.} \\ 0, & \text{if the value } i \text{ is not taken out of } n \text{ values.} \end{array}$$

Since the solution satisfying the equation $x_1x_2x_3\bar{x}_4\bar{x}_5=1$ is $x_1=1,x_2=1,x_3=1,$ $x_4=0,x_5=0$, the equation $x_1x_2x_3\bar{x}_4\bar{x}_5=1$ corresponds to the 3-combination of 5, 123.

Therefore, we can realize that the following equation represents all the 3-combinations of 5.

$$\underset{123}{x_1x_2x_3\bar{x}_4\bar{x}_5} + \underset{124}{x_1x_2\bar{x}_3x_4\bar{x}_5} + \underset{125}{x_1x_2\bar{x}_3\bar{x}_4x_5} + \underset{134}{x_1\bar{x}_2x_3x_4\bar{x}_5} + \underset{135}{x_1\bar{x}_2x_3\bar{x}_4x_5} +$$
$$\underset{145}{x_1\bar{x}_2\bar{x}_3x_4x_5} + \underset{234}{\bar{x}_1x_2x_3x_4\bar{x}_5} + \underset{235}{\bar{x}_1x_2x_3\bar{x}_4x_5} + \underset{245}{\bar{x}_1x_2\bar{x}_3x_4x_5} + \underset{345}{\bar{x}_1\bar{x}_2x_3x_4x_5} = 1$$

We will denote a Boolean function representing all the j-combinations of i+j by $f_{i,j}(x_1,x_2,\ldots,x_{i+j})$. The Boolean function $f_{i,j}(x_1,x_2,\ldots,x_{i+j})$ satisfies the following recurrence relation.

Theorem 1

$$f_{0,j}(x_1,x_2,\ldots,x_j) = x_1x_2\ldots x_j \quad (j\geqq 1) \qquad (1.1)$$

$$f_{i,0}(x_1,x_2,\ldots,x_i) = \bar{x}_1\bar{x}_2\ldots\bar{x}_i \quad (i\geqq 1) \qquad (1.2)$$

$$f_{i,j}(x_1,x_2,\ldots,x_{i+j}) = f_{i,j-1}(x_1,x_2,\ldots,x_{i+j-1})x_{i+j} + f_{i-1,j}(x_1,x_2,\ldots,x_{i+j-1})\bar{x}_{i+j} \quad (i\geqq 1, j\geqq 1) \qquad (1.3)$$

Proof. The equations (1.1) and (1.2) are obvious. We will denote the set including all the j-combinations of i+j, by U. The set U can be divided into two subset V and W such that $V\cap W=\phi, V\cup W=U$. A combination included in a subset V surely contains the value i+j. Thus, the Boolean function representing all the combinations in V is $f_{i,j-1}(x_1,x_2,\ldots,x_{i+j-1})x_{i+j}$. A combination included in a subset W does not contain the value i+j. Thus, the Boolean function representing all the combinations in V is $f_{i-1,j}(x_1,x_2,\ldots,x_{i+j-1})\bar{x}_{i+j}$. By these facts, the recurrence relation (1.3) is derived.

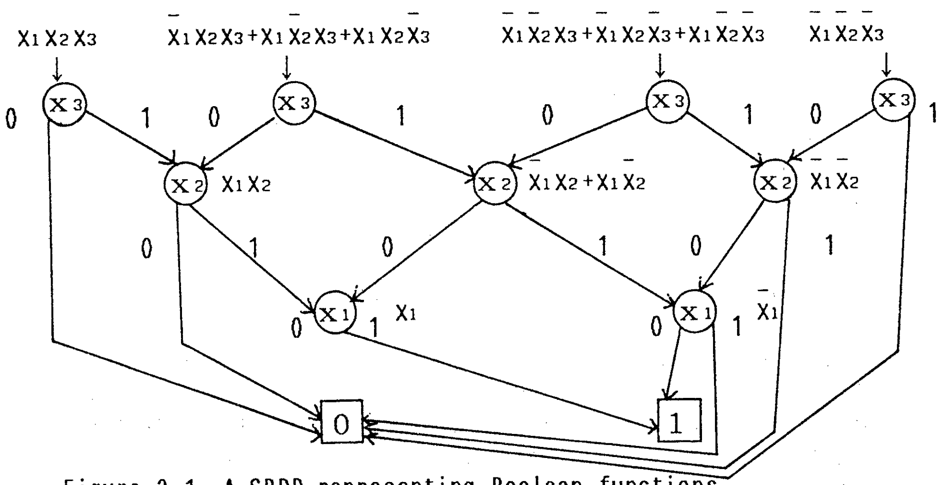
Thus, all the j-combinations of i+j can be obtained by the following algorithm 1.

Algorithm 1

```
STEP 1: Construct a BDD by using the above recurrence relation.
STEP 2: Traverse all the paths of a BDD from the root corresponding to
        $f_{i,j}(x_1,x_2,\ldots,x_{i+j})$ to terminal node labeled 1.
```

If 1(0) edge is going out of a node labeled x_i in a path, then 1(0) edge corresponds to the Boolean variable $x_i(\bar{x}_i)$. We note that the number of 1 edges is j and the number of 0 edges is i.

As an example. we show a SBDD representing Boolean functions $f_{i,j}(x_1,x_2,\ldots,x_{i+j})$ $(1\leqq i+j\leqq 3,\ 0\leqq i, 0\leqq j)$ in Figure 3.1.



$x_1x_2x_3$
$\bar{x}_1x_2x_3+x_1\bar{x}_2x_3+x_1x_2\bar{x}_3$
$\bar{x}_1\bar{x}_2x_3+\bar{x}_1x_2\bar{x}_3+x_1\bar{x}_2\bar{x}_3$
$\bar{x}_1\bar{x}_2\bar{x}_3$
x_3
x_2
x_1
x_1x_2
$\bar{x}_1x_2+x_1\bar{x}_2$
$\bar{x}_1\bar{x}_2$
x_1
$\bar{x}_1$
0
1

Figure 3.1. A SBDD representing Boolean functions $f_{i,j}(x_1,x_2,\ldots,x_{i+j})$ $(1\leqq i+j\leqq 3,\ 0\leqq i,0\leqq j)$.

Lastly, we will mention two important combinatorial objects. They are often used to generate other combinatorial objects.

One is the ways of selecting at least r from the set $\{1,2,\ldots,n\}$. The Boolean function representing these ways is

$$f_{n-r,r}(x_1,\ldots,x_n) + f_{n-r-1,r+1}(x_1,\ldots,x_n) + \ldots + f_{0,n}(x_1,\ldots,x_n) = 1$$

The other is the ways of selecting at most r from the set $\{1,2,\ldots,n\}$. The Boolean function representing these ways is

$$f_{n,0}(x_1,\ldots,x_n) + f_{n-1,1}(x_1,\ldots,x_n) + \ldots + f_{n-r,r}(x_1,\ldots,x_n) = 1$$

Experimental Results

For some n and r, we have computed ${}_nC_r$ and the number of nodes used to construct the BDD representing all the r-combinations of n. We have also measured the running time required to construct the same BDD, coded in C, on a Sun SPARC station 2 workstation(64MByte). The results are shown in Table 3.1.

n	r	${}_nC_r$	number of nodes	running time
200	100	0.90549×10^{59}	10199	1.86
600	300	0.13511×10^{180}	90599	7.14
1000	500	0.27029×10^{300}	250999	23.31
1400	700	0.58991×10^{420}	491399	45.86

Table 3.1. ${}_nC_r$ and the number of nodes used to construct the BDD representing all the r-combinations of n and the running time required to construct the same BDD(times in seconds).

It is interesting that a large number of combinations can be represented by a small number of nodes contained in the corresponding BDD. It can be shown the number of nodes contained in the BDD representing all the r-combinations of n is less than or equal to $n(r+1)-r^2$.

4. The problem of generating all the balanced parentheses sequences

In this section, we consider the sequence consisted of left parenthesis "(" and right parenthesis ")". The sequence of length n is denoted by $p_1p_2\ldots p_n$. The number of left parentheses in the sequence $p_1p_2\ldots p_n$ is denoted by left($p_1p_2\ldots p_n$) and the number of right parentheses in the sequence $p_1p_2\ldots p_n$ is denoted by right($p_1p_2\ldots p_n$).

When a sequence $p_1p_2\ldots p_{i+j}$ is consisted of i right parentheses and j left parentheses and left($p_1p_2\ldots p_k$) is greater than or equal to right($p_1p_2\ldots p_k$) for any $k(1\leqq k\leqq i+j)$, a sequence $p_1p_2\ldots p_{i+j}$ is called Left Parentheses Preceding Sequence with i right parentheses and j left parentheses and denoted by (i,j)-LPPS. When i=j, (i,i)-LPPS $p_1p_2\ldots p_{2i}$ is called a Balanced Parentheses Sequence with i right parentheses and i left parentheses and denoted by (i,i)-BPS. The number of (n,n)-BPS is shown to be ${}_{2n}C_n/(n+1)$. This number is often found in combinatorial problems and well known as the Catalan Number.

Now, we will derive the Boolean function representing all (i,j)-LPPS. First, we define Boolean variables $x_1,x_2,\ldots,x_n$ as follows.

$$x_i = \begin{cases} 1, & \text{if the ith character } p_i \text{ is a left parenthesis.} \\ 0, & \text{if the ith character } p_i \text{ is a right parenthesis.} \end{cases}$$

We will denote a Boolean funcion representing (i,j)-LPPS by $g_{i,j}(x_1,x_2,\ldots,x_{i+j})$. The Boolean function $g_{i,j}(x_1,x_2,\ldots,x_{i+j})$ satisfies the following recurrence relation.

Theorem 2

$$g_{0,j}(x_1,x_2,\dots,x_j) = x_1x_2\dots x_j \quad (j\geqq 1) \qquad (2.1)$$

$$g_{i,i-1}(x_1,x_2,\dots,x_{2i-1}) = 0 \quad (i\geqq 1) \qquad (2.2)$$

$$g_{i,j}(x_1,x_2,\dots,x_{i+j}) = g_{i,j-1}(x_1,x_2,\dots,x_{i+j-1})x_{i+j} + g_{i-1,j}(x_1,x_2,\dots,x_{i+j-1})\bar{x}_{i+j} \quad (i\leqq j) \qquad (2.3)$$

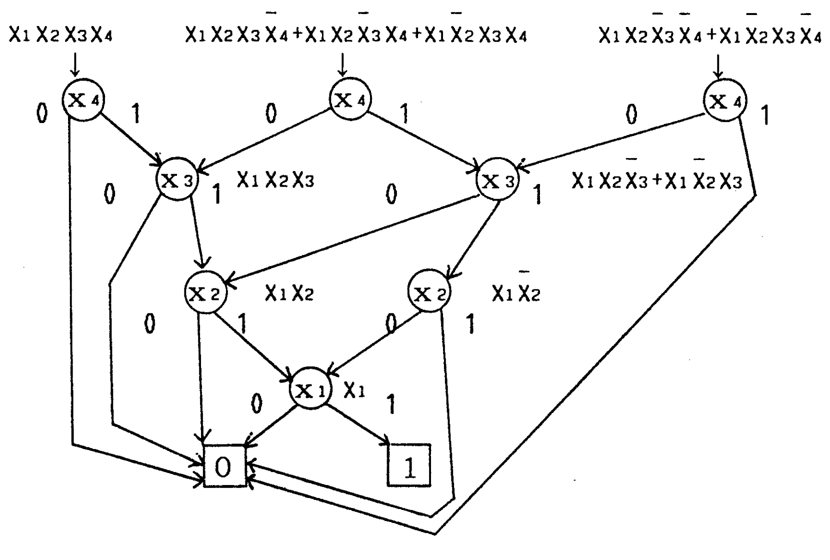
Proof. The equations (2.1) and (2.2) are obvious. Let (i,j)-LPPS be $p_1p_2 \dots p_{i+j}$. If p_{i+j}="(", then $p_1p_2\dots p_{i+j-1}$ is (i,j-1)-LPPS. If p_{i+j}=")", then $p_1p_2 \dots p_{i+j-1}$ is (i-1,j)-LPPS. Since all the (i,j)-LPPSs consist of all the (i,j-1)-LPPSs and all the (i-1,j)-LPPSs, the equation (2.3) can be obtained.

Thus, all the (i,j)-LPPSs can be obtained by the following Algorithm 2.

Algorithm 2

STEP 1: Construct a BDD by using the above recurrence relation.
STEP 2: Traverse all the paths of a BDD from the root corresponding to $g_{i,j}(x_1,x_2,\dots,x_{i+j})$ to terminal node labeled 1.

As an example. we show the SBDD representing Boolean functions $g_{i,j}(x_1,x_2,\dots,x_{i+j})$ $(1\leqq i+j\leqq 4,\ 0\leqq i\leqq j)$ in Figure 4.1.



$x_1x_2x_3x_4$
$x_1x_2x_3\bar{x}_4+x_1x_2\bar{x}_3x_4+x_1\bar{x}_2x_3x_4$
$x_1x_2\bar{x}_3\bar{x}_4+x_1\bar{x}_2x_3\bar{x}_4$
x_4
x_3
$x_1x_2x_3$
$x_1x_2\bar{x}_3+x_1\bar{x}_2x_3$
x_2
x_1x_2
$x_1\bar{x}_2$
x_1
0
1

Figure 4.1. A SBDD representing a Boolean functions $g_{i,j}(x_1,x_2,\dots,x_{i+j})$ $(1\leqq i+j\leqq 4,\ 0\leqq i\leqq j)$

Experimental Results

For some 2n, we have computed ${}_{2n}C_n/(n+1)$ and the number of nodes used to represent the BDD representing all the (n,n)-BPSs and measured the running time required to construct the same BDD. The results are shown in Table 4.1.

2n	${}_{2n}C_n/(n+1)$	number of nodes	running time
400	0.51220×10^{117}	20299	2.18
800	0.46893×10^{237}	80599	5.11
1200	0.65975×10^{357}	180900	10.05
1600	0.11071×10^{478}	321200	17.24
2000	0.20461×10^{598}	501500	32.21

Table 4.1. ${}_{2n}C_n/(n+1)$ and the number of nodes used to construct the BDD representing all the (n,n)-BPSs and the running time required to construct the same BDD (times in seconds)

We also note a large number of balanced parentheses sequences of length 2n can be represented by a small number of nodes contained in the correspoding BDD. It can be shown the number of nodes contained in the BDD is less than or equal to n(n+3)/2.

5. Conclusion

We have considered the problem of generating all the r-combinations of n and the problem of generating all the balanced parentheses sequences of length 2n. These generating algorithms are constructed by using Boolean functions and the recurrence relations. The Boolean functions representing them have been implemented compactly by a BDD and executed at a high speed.

Acknowledgement

The authors would like to thank Mr.Shin-ichi Minato and Mr.Hiroyuki Ochi who offered them the Boolean function manipulator. They would also like to express their sincere appreciation to Professor Naofumi Takagi, Mr.Kiyoharu Hamaguti and Mr.Yasuhiko Takenaga for valuable discussions and comments.

References

[1] S.B.Akers: "Binary Decision Diagrams", IEEE Trans. Comput., vol.c-27, no.6, pp.509-516, (June 1978).
[2] R.E.Bryant: "Graph-Based Algorithms for Boolean Function Manipulation", IEEE Trans. Comput., vol.c-35, no.8, pp.667-691, (Aug. 1985)
[3] S.Minato, N.Ishiura and S.Yajima: "Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation", Proc. 27th ACM/IEEE DAC, pp.52-57, (June 1990).
[4] K.S.Brace, R.L.Rudell and R.E.Bryant: "Efficient Implementation of a BDD Package", Proc. 27th ACM/IEEE DAC, pp.40-45, (June 1990).
[5] H.Ochi, K.Yasuoka and S.Yajima: "A Breadth-First Algorithm for Efficient Manipulation of Shared Binary Decision Diagrams in the Secondary Memory", The 45th General Convention of IPSJ, 6-137, (Oct. 1992).
[6] N.J.A.Sloane: "A Handbook of Integer Sequences", 1973.
[7] C.E.Shannon: "A symbolic analysis of relay and switching circuits", Trans. AIEE, vol.57, pp.713-723, (1938).