One more way for counting monotone Boolean functions

Valentin Bakoev

”St. Cyril and St. Methodius” University,

Veliko Turnovo, Bulgaria

1. Introduction

The Dedekind's problem (1897) – *a problem of counting* the elements of free distributive lattices of n generators, or equivalently, *the number* $\psi(n)$ *of monotone Boolean functions (MBFs) of* n *variables*.

The investigations of this problem are focused in two main directions:

- to compute this number for a given n (by deriving appropriate formulas for it, or by algorithms for counting, etc.);
- to estimate this number (many formulas for evaluating $\psi(n)$ are obtained by Kleithman, Korshunov, Kisielewicz, Shmulevich etc.).

1. Introduction

Till now, the values of $\psi(n)$ are known only for $n \leq 8$:

n	$\psi(n)$	Computed by
0	2	R. Dedekind, 1897
1	3	R. Dedekind, 1897
2	6	R. Dedekind, 1897
3	20	R. Dedekind, 1897
4	168	R. Dedekind, 1897
5	7 581	R. Church, 1940
6	7 828 354	M. Ward, 1946
7	2 414 682 040 998	R. Church, 1965
8	56 130 437 228 687 557 907 788	D. Wiedemann, 1991

1. Introduction

To feel the complexity of the problem we note that:

- in 1991 Wiedemann used a Cray-2 processor for about 200 hours to compute $\psi(8)$;
- it took more than a century to compute the last 4 values ot $\psi(n)$.

The algorithms for computing $\psi(n)$ are not too numerous and various. Most of them follow the principle "generating and counting". Other algorithms use propositional calculus and #SAT-algorithms. The most powerful algorithms compute $\psi(8)$ by appropriate decomposition of functions and/or sets.

1. Introduction

This work continues our previous investigations of the Dedekind's problem. They are based on the properties of a matrix structure, defined by us. We developed a new algorithm for generating (and counting) all MBFs up to 6 variables. In spite of its numerous improvements, it is not powerful enough for computing the next values in acceptable running-time.

Here we represent some new ideas about applying the dynamic-programing strategy in solving the Dedekind's problem.

2. Basic notions

Let $\{0,1\}^n$ be the $n-$*dimensional Boolean cube* and $\alpha = (a_1, \ldots, a_n)$, $\beta = (b_1, \ldots, b_n)$ be binary vectors in it.

- *Ordinal number* of α is the integer $\#(\alpha) = a_1.2^{n-1} + a_2.2^{n-2} + \ldots + a_n.2^0$;
- Vector α *precedes lexicographically* vector β, if $\exists$ an integer k, $1 \leq k \leq n$, such that $a_i = b_i$, for $i = 1, 2, \ldots, k-1$, and $a_k < b_k$, or if $\alpha = \beta$. The vectors of $\{0,1\}^n$ are in lexicographic order iff their ordinal numbers form the sequence $0, 1, \ldots, 2^n - 1$.

2. Basic notions

- The relation "$\preceq$" *(precedes)* is defined over $\{0,1\}^n \times \{0,1\}^n$ as follows: $\alpha \preceq \beta$ if $a_i \le b_i$, for $i = 1, 2, \ldots, n$;
- When $\alpha \preceq \beta$ or $\beta \preceq \alpha$ we call α and β *comparable*, otherwise they are *incomparable*;

A *Boolean function* f of n variables is a mapping $f : \{0,1\}^n \to \{0,1\}$. The function f is called *monotone* if for any $\alpha, \beta \in \{0,1\}^n$, $\alpha \preceq \beta$ implies $f(\alpha) \le f(\beta)$.

If f is a MBF, it has an unique *minimal disjunctive normal form* (MDNF), where all literals in the prime implicants of f are uncomplemented.

3. Preliminary results

We define a *matrix of precedences* of the vectors in $\{0,1\}^n$: $M_n = ||m_{i,j}||$ has dimension $2^n \times 2^n$, and for each $\alpha, \beta \in \{0,1\}^n$, such that $\#(\alpha) = i$ and $\#(\beta) = j$, we set $m_{i,j} = 1$ if $\alpha \preceq \beta$, or $m_{i,j} = 0$ otherwise.

Theorem 1 *The matrix M_n is a block matrix, defined recursively:*

$$M_1 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad M_n = \begin{pmatrix} M_{n-1} & M_{n-1} \\ O_{n-1} & M_{n-1} \end{pmatrix},$$

where M_{n-1} denotes the same matrix of dimension $2^{n-1} \times 2^{n-1}$, and O_{n-1} is the $2^{n-1} \times 2^{n-1}$ zero matrix.

3. Preliminary results

Theorem 2 *Let* $\alpha = (a_1, a_2, \ldots, a_n) \in \{0, 1\}^n$, $\#(\alpha) = i$, $1 \leq i \leq 2^n - 1$, *and* α *has ones in positions* $(i_1, i_2, \ldots, i_r)$, $1 \leq r \leq n$. *Then the* i*-th row* r_i *of the matrix* M_n *is the vector of functional values of the prime implicant* $c_i = x_{i_1} x_{i_2} \ldots x_{i_r}$, *i.e.* α *is a characteristic vector of the literals in* c_i, *which is a monotone function. When* $\#(\alpha) = 0$, *the zero row of* M_n *corresponds to the constant* $\tilde{1}$.

3. Preliminary results

Illustration of the assertion of Theorem 2, for $n = 3$.

$\alpha = (x_1, x_2, x_3)$	$i = \#(\alpha)$	M_3	c_i
(0 0 0)	0	1 1 1 1 1 1 1 1	$\tilde{1}$
(0 0 1)	1	0 1 0 1 0 1 0 1	x_3
(0 1 0)	2	0 0 1 1 0 0 1 1	x_2
(0 1 1)	3	0 0 0 1 0 0 0 1	x_2x_3
(1 0 0)	4	0 0 0 0 1 1 1 1	x_1
(1 0 1)	5	0 0 0 0 0 1 0 1	x_1x_3
(1 1 0)	6	0 0 0 0 0 0 1 1	x_1x_2
(1 1 1)	7	0 0 0 0 0 0 0 1	$x_1x_2x_3$

3. Preliminary results

So the vector of any monotone function f is *a linear combination*
$f(x_1, x_2, \ldots, x_n) = a_0 r_0 \vee a_1 r_1 \vee \ldots \vee a_{2^n-1} r_{2^n-1}$,
where r_i is the i-th row of the matrix M_n, and
$a_i \in \{0, 1\}$, for $i = 0, 1, \ldots, 2^n - 1$.

In other words, M_n plays the role of a generator matrix for the set of all MBFs of n variables.

When $f(x_1, x_2, \ldots, x_n) = r_{i_1} \vee r_{i_2} \vee \ldots \vee r_{i_k}$ corresponds to a MDNF of f, then any two rows r_{i_j} and r_{i_l}, $1 \leq j < l \leq k$, are pairwise incomparable.

3. Preliminary results

Our algorithm, called *GEN*, *generates all MBFs of n variables* (input) as *vectors in lexicographic order* (output).

Algorithm GEN.

1) Generate the matrix M_n.

2) Set $f = (0, 0, \ldots, 0)$ – the zero constant. Output f.

3) For each row r_i of M_n, $i = 2^n - 1, \ldots, 0$, set $f = r_i$ and:

   a) output f;

   b) for each position j, $j = 2^n - 2, 2^n - 3, \ldots, i + 1$, check whether $f[j] = 0$ (i.e. the i-th and the j-th rows are incomparable). If "Yes" then set (recursively) $f = f \vee r_j$ and go to step 3.a.

4) End.

3. Preliminary results

The essential part of the code of GEN (steps 3.a and 3.b) written in C is:

```
void Gen_I ( bool G[], int i ) {
   bool H [Max_Dim];
   for ( int k=i; k<N; k++ )     // N= 2^n
      H[k]= G[k] || M[i][k];     // M is M_n
   Print ( H );
   for ( int j= N-1; j>i; j-- )     // step 3.b
      if ( !H[j] ) Gen_I ( H, j );
}
```

4. Outline of the new algorithm

Trying to improve and speed-up the algorithm GEN, we observe that:

- the same subfunctions are generated many times;
- their number grows extremely fast when n grows.

So we shall concentrate on counting that avoids generating.

We set the problem *"Let the value of the cell $m_{i,j}$ in matrix M_n be 0, for a given n. How many MBFs can be obtained by disjunction of row r_i and all possible rows (one or more than one), having indices $\geq j$?"*.

4. Outline of the new algorithm

So we modify the algorithm GEN (its new version we call GEN_Cell):

- we add to the function Gen_I a parameter for the depth of the recursion;
- we add a counter for the generated functions;
- we store the integers, computed by this counter, in a $2^n \times 2^n$ matrix Res_n;

So we have to fill only these cells of Res_n, which correspond (i.e. have the same indices) to zero elements above the main diagonal in M_n.

Example. The results for $n = 4$ are:

4. Outline of the new algorithm

M_4	row	Res_4	s_i
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	1	0 0 5 0 3 0 5 0 1 0 2 0 1 0 1 0	19
0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1	2	0 0 0 0 3 5 0 0 1 2 0 0 1 1 0 0	14
0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1	3	0 0 0 0 5 20 11 0 1 5 3 0 1 2 1 0	50
0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1	4	0 0 0 0 0 0 0 0 1 2 1 1 0 0 0 0	6
0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1	5	0 0 0 0 0 0 11 0 1 5 2 3 1 0 1 0	25
0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1	6	0 0 0 0 0 0 0 0 1 4 3 3 1 1 0 0	14
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1	7	0 0 0 0 0 0 0 0 1 5 3 5 1 2 1 0	19
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1	8	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1
0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1	9	0 0 0 0 0 0 0 0 0 0 2 0 1 0 1 0	5
0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1	10	0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0	3
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1	11	0 0 0 0 0 0 0 0 0 0 0 0 1 2 1 0	5
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1	12	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1	13	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0	2
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1	14	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	15	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1
			Total: S=167

4. Outline of the new algorithm

Important observation: the same submatrices in M_4 (more precisely, certain shapes of zeros in them), correspond to the same shapes of non-zero values in the matrix Res_4.

Obviously, this is due to the recursively defined block structure of the matrix M_n and the nature of the algorithm GEN.

This fact demonstrates the property *overlapping subproblems* – the first key ingredients for applying the dynamic programing strategy.

4. Outline of the new algorithm

The same is valid for the second key property – *optimal substructure*. Indeed, if (for a given n) the subproblems are solved, i.e. the necessary values are computed and stored in the matrix Res_n, we can obtain the solution of the problem (i.e. to find $\psi(n)$) as follows:

(1) sum the numbers in the i-th row of the matrix Res_n and add 1 (because every row of M_n is in itself a monotone function). Denote this sum by s_i, for $i = 0, 1, \ldots, 2^n - 1$;

(2) compute the sum $S = \sum_{i=0}^{2^n-1} s_i$;

(3) set $\psi(n) = S + 1$ (since the constant 0 is yet not counted) and return it.

4. Outline of the new algorithm

Next improvement of algorithm GEN_Cell: after computing the value of $Res_n(i, j)$, *we copy it* in the corresponding cells of the same shapes above – so we prevent from solving the same subproblems more than once.

Even so, executing GEN_Cell for one cell only can cause generating many subfunctions, which have been already generated. Their memoization can take a large amount of memory, and our goal is to restrict the generating as possible.

4. Outline of the new algorithm

The next our idea: let $i < j$, $M_n(i,j) = 0$ and $Res_n(i,j) = 0$. We need to compute the value of $Res_n(i,j)$, i.e. to count all MBFs, which are disjunction of i-th row of M_n with all rows of M_n, having indices $\geq j$.

All cells of the i-th row from the j-th cell to the last one we consider as a vector and denote it by (0α). Analogously for the j-th row, all cells from the j-th to the last cell we consider as a vector and denote it by (1β). For α and β we have 3 cases: (1) $\alpha \preceq \beta$; (2) $\beta \prec \alpha$, and (3) α and β are incomparable. Using the properties of the matrix M_n and the above arguments we can prove:

4. Outline of the new algorithm

Proposition 1 *Case (1): if* $\alpha \preceq \beta$ *then*

$$Res_n(i,j) = 1 + \sum_{k=j+1}^{2^n-1} Res_n(j,k) = s_j + 1.$$

Proposition 2 *Case (2): if* $\beta \prec \alpha$ *then*

$$Res_n(i,j) = 1 + \sum_{k=j+1}^{2^n-1} Res_n(i,k).$$

Suppose we want to compute $Res_n(i,j)$ and we have already computed $Res_n(i,k)$ and $Res_n(j,k)$, for $k = j+1, \ldots, 2^n - 1$.
If $\alpha \preceq \beta$ or $\beta \prec \alpha$, we apply Proposition 1 or 2, respectively.
For the third case we use GEN_Cell, since we have not found a better algorithm (or a formula) till now.

4. Outline of the new algorithm

Proposition 3 *For a given* n*, the matrix* M_n *contains* 4^n *elements and:*
1) 3^n *of them are equal to 1 and they are placed on the main diagonal or above it;*
2) all $(4^n - 2^n)/2$ *elements under the main diagonal are zeros, and also* $(4^n - 2.3^n + 2^n)/2$ *zeros are placed above the main diagonal.*

So our algorithm has to compute and fill in $(4^n - 2.3^n + 2^n)/2$ numbers in the cells of Res_n.
Some of them are obtained in accordance with the considered 3 cases.
The rest of them are simply copies of numbers already computed.

4. Outline of the new algorithm

Experimental results for the number of the cells of Res_n in each case, for $n = 6, 7, 8$:

n	$(4^n - 2.3^n + 2^n)/2$	In case 1	In case 2	In case 3	Copies
6	1351	211	26	544	570
7	6069	665	57	2645	2702
8	26335	2059	120	12018	12138

5. Conclusions

The results in last table seem to be optimistic, especially if we compare them with the values of $\psi(n)$, given in the first table.
The main and still open problem is to develop an efficient way for computing in Case 3.
Some secondary problems also have to be solved: representation and summation of long integers, efficient usage of the memory (especially for M_n and Res_n), etc.
Efficient solutions of these problems will decrease essentially the running-time for computing $\psi(7)$ and $\psi(8)$ and may allow us to compute $\psi(9)$ in a reasonable time.

THANK YOU

FOR YOUR ATTENTION!