*Thirteenth International Workshop on Algebraic and Combinatorial Coding Theory*
*June 15-21, 2012, Pomorie, Bulgaria* *pp. 47–52*

# One more way for counting monotone Boolean functions[1]

Valentin Bakoev v.bakoev@uni-vt.bg
University of Veliko Turnovo, Bulgaria

**Abstract.** Here we represent the outline of a new algorithm for counting the monotone Boolean functions of $n$ variables. It is a continuation of our previous investigation and results, related to this problem.

## 1 Introduction

In 1897 Dedekind sets the 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. Since then the scientists investigate this problem in two main directions. The first one is to compute this number for a given $n$ – by deriving appropriate formulas for it, or by algorithms for counting, etc. The second (when the first one is not successful enough) is to estimate this number – many formulas for evaluating $\psi(n)$ are obtained in [5–8]. In spite of their efforts, the values of $\psi(n)$ are known only for $n \leq 8$ [4, 11, 12]:

| $n$ | $\psi(n)$ | 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 |

Table 1. $\psi(n)$, for $0 \leq n \leq 8$, and the history of their computing.

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 almost a century to compute the last 4 values of $\psi(n)$.

The algorithms for computing $\psi(n)$ that we know, are not too numerous and various. Most of them follow the principle "generating and counting" [3, 4, 9, 10]. The algorithms in [11] use propositional calculus and #SAT-algorithms in computing these numbers. The most powerful algorithms (represented in [4,12]) compute $\psi(8)$ by appropriate decomposition of functions and/or sets.

[1]This work is partially supported by V. Turnovo University Science Fund under Contract RD-642-01/26.07.2010.

This work continues our previous investigations of the Dedekind's problem. In [1, 2] we proposed a new algorithm for generating (and counting) all MBFs up to 6 variables, based on the properties of a certain matrix structure. In spite of its numerous improvements (the running-time for computing $\psi(6)$ was reduced to 0,5 sec.), it is not powerful enough for computing the next values. Here we represent some new ideas about applying the dynamic-programming strategy in solving the Dedekind's problem.

## 2 Basic notions and preliminary results

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 $\alpha$ is the integer $\#(\alpha) = a_1.2^{n-1} + a_2.2^{n-2} + \cdots + a_n.2^0$. Vector $\alpha$ *precedes lexicographically* vector $\beta$, 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 (as we consider further) iff their ordinal numbers form the sequence $0, 1, \ldots, 2^n - 1$.

The relation "$\preceq$" is defined over $\{0,1\}^n \times \{0,1\}^n$ as follows: $\alpha \preceq \beta$ if $a_i \leq b_i$, for $i = 1, 2, \ldots, n$. This relation is reflexive, antisymmetric and transitive and so $\{0,1\}^n$ is a partially ordered set (POSet) with respect to it. When $\alpha \preceq \beta$ or $\beta \preceq \alpha$ we call $\alpha$ and $\beta$ *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 $f(\alpha) \leq f(\beta)$. It is well known that if $f$ is a monotone function, it has an unique *minimal disjunctive normal form* (MDNF), consisting of all prime implicants of $f$, where all literals are uncomplemented.

We define a *matrix of precedences* of the vectors in $\{0,1\}^n$ as follows: $M_n = ||m_{i,j}||$ has dimension $2^n \times 2^n$, and for each pair of vectors $\alpha, \beta \in \{0,1\}^n$, 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, or by Kronecker product:*

$$M_1 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad M_n = \begin{pmatrix} M_{n-1} & M_{n-1} \\ O_{n-1} & M_{n-1} \end{pmatrix}, \quad or \quad M_n = M_1 \otimes M_{n-1} = \bigotimes_{i=1}^{n} M_1,$$

*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.*

**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 $\alpha$ 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. $\alpha$ 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 $\tilde{1}$.*

| $\alpha = (x_1, x_2, x_3)$ | $i = \#(\alpha)$ | $M_3$ | $c_i$ |
|---|---|---|---|
| (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 | $x_1x_2x_3$ |

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

So the vector of any monotone function $f$ can be expressed in terms of a linear combination $f(x_1, x_2, \ldots, x_n) = a_0r_0 \vee a_1r_1 \vee \cdots \vee a_{2^n-1}r_{2^n-1}$, where $r_i$ is the $i$-th row of the matrix $M_n$ and its coefficient $a_i \in \{0,1\}$, 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 $f(x_1, x_2, \ldots, x_n) = r_{i_1} \vee r_{i_2} \vee \cdots \vee r_{i_k}$ corresponds to a MDNF of $f$, then any two rows $r_{i_j}$ and $r_{i_l}$ (corresponding to the prime implicants $c_{i_j}$ and $c_{i_l}$), $1 \le j < l \le k$, are pairwise incomparable.

Our algorithm, called *GEN*, generates all MBFs of $n$ variables, $1 \le n \le 7$, as vectors in lexicographic order. It is based on the properties of the matrix $M_n$ (more details are given in [1]).

**Algorithm:** GEN.
**Input:** the number of the variables $n$.
**Output:** the vectors of all MBFs of $n$ variables in lexicographic order.
**Procedure:**

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. whether 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.

The essential part of its code (steps 3.a and 3.b), written in C, looks like this:

```
void Gen_I ( bool G[], int i ) {
   bool H [Max_Dim];
   for ( int k=i; k<N; k++ )         // N= 2^n is a global variable
      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 );
}
```

## 3 Outline of an algorithm for counting MBFs

While trying to improve and speed-up the algorithm GEN, we observe that the same subfunctions are generated many times in the different iterations of the main cycle in GEN. Their number grows extremely fast when the number of variables 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 $\geq j$?"*. To solve this problem (for $n \leq 7$) we modify the algorithm GEN and its new version we call GEN_Cell. So we add to the function Gen_I a parameter for the depth of the recursion, and also a counter for the generated functions (when the depth is 0 we return the value of this counter for storing and then set it to 0 for the next recursive call). The integers computed in this way we store in a $2^n \times 2^n$ matrix denoted by $Res_n$, which elements are set to 0 initially. 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 $M_n$. For example, the results for $n = 4$ are given in Table 3.

| $M_4$ | 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: $S = 167$ |

Table 1: $M_4$, $Res_4$ and $s_i$ – the number of MBFs on rows.

We can see that the same submatrices in $M_4$ or, 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 $M_n$, the nature of generating and hence the nature of the algorithm GEN. This fact can be proved rigorously by induction on $n$ and it demonstrates the property *overlapping subproblems* – one of the key ingredients that enables us to apply the dynamic programming strategy. The same is valid for the other 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 $s_i, i = 0, 1, \dots, 2^n - 1$;

(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.

For example, the last column of Table 3 contains the sums $s_i$ of the elements of each row mentioned in (1). If we take the sum $s_{14} + s_{15} + 1$ (since $M_1$ is placed in the lower right corner of $M_4$), we obtain $3 = \psi(1)$. If we do the same for the last 4 rows: $s_{12} + \dots + s_{15} + 1$ (since $M_2$ is also placed in the lower right corner of $M_4$), we obtain $6 = \psi(2)$. For the the last 8 rows, we obtain $s_8 + \dots + s_{15} + 1 = 20 = \psi(3)$, and finally, for all rows, we obtain $s_0 + \dots + s_{15} + 1 = S + 1 = 168 = \psi(4)$.

The next improvement of algorithm GEN_Cell seems obvious: 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 memorization can take a large amount of memory, and our goal is to restrict the generating as possible. We continue with our next idea. Assume that $i < j$, $M_n(i,j) = 0$ 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 $(0\alpha)$. 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 $(1\beta)$. For $\alpha$ and $\beta$ we have 3 cases: (1) $\alpha \preceq \beta$; (2) $\beta \prec \alpha$, and (3) $\alpha$ and $\beta$ are incomparable. Using the properties of the matrix $M_n$ and the above arguments we can prove:

**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 $k = j+1, \dots, 2^n - 1$. 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.

Next proposition follows directly from the definition of matrix $M_n$.

**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 these numbers are obtained in accordance with the considered 3 cases. The rest of them are simply copies of numbers already computed. Experimental results for the number of the cells in each case, for $n = 6, 7, 8$, are given in Table 4.

# 4 Conclusions

The results in Table 4 seem to be optimistic, especially if we compare them with the values of $\psi(n)$, given in Table 1. The main and still open problem is to develop an

| $n$ | $(4^n - 2.3^n + 2^n)/2$ | 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 |

Table 2: Number of cells from each case in the matrix $Res_n$.

efficient way for computing in Case 3. Some secondary problems also have to be solved – for example, representation and summation of long integers, efficient usage of the memory, especially for the matrices $M_n$ and $Res_n$, etc. Their efficient solutions 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.

# References


[1] V. Bakoev, Generating and identification of monotone Boolean functions, Mathematics and education in mathematics, Sofia (2003), pp. 226–232.

[2] V. Bakoev, Some properties of one matrix structure at monotone Boolean functions, Proc. EWM Intern. Workshop Groups and Graphs, Varna, Bulgaria (2002), pp. 5–8.

[3] J. Dezert, F. Smarandache, On the generation of hyper-powersets for the DSmT, Proc. of the 6th Int. Conf. of Information Fusion (2003), pp. 1118–1125.

[4] R. Fidytek, A. Mostowski, R. Somla, A. Szepietowski, Algorithms counting monotone Boolean functions, Inform. Proc. Letters 79 (2001), pp. 203-209.

[5] A. Kisielewicz, A solution of Dedekinds problem on the number of isotone Boolean functions, J. reine angew. math., Vol. 386 (1988) pp. 139–144.

[6] D. Kleitman, On Dedekinds problem: the number of monotone Boolean functions, Proc. of AMS, 21(3) (1969), pp. 677-682.

[7] A. Korshunov, On the number of monotone Boolean functions, Problemy Kibernetiki, Vol. 38, Moscow, Nauka (1981), pp. 5-108 (in Russian).

[8] A. Korshunov, I. Shmulevich, On the distribution of the number of monotone Boolean functions relative to the number of lower units, Discrete Mathematics, 257 (2002), pp. 463-479.

[9] http://mathpages.com/home/kmath094.htm, Generating the M. B. Functions

[10] http://angelfire.com/blog/ronz/, Ron Zeno's site

[11] M. Tombak, A. Isotamm, T. Tamme, On logical method for counting Dedekind numbers, Lect. Notes Comp. Sci., 2138, Springer-Verlag (2001), pp. 424–427.

[12] D. Wiedemann, A computation of the eighth Dedekind number, Order, no. 8 (1991), pp. 5–6.