FILOMAT (Niš) 9:2 (1995), 345–355
Filomat '94, Niš, October 22-24, 1994
Geometry. Computer Sciences

THE GENERATION OF PERMUTATIONS THROUGH GDD

Dragan Janković and Milena Stanković

ABSTRACT. *In this paper we consider the generation of permutations, i.e. all ordered n-tuples of different elements from the set* $A_n = \{a_0, a_1, ..., a_{n-1}\}$ *which is a combinatorial problem often occurring in practice. We give a method for the generation of all permutations of n given items through generalized decision diagrams. Each of n! paths in the appropriate decision diagram maps into one of n! permutations. The proposed method is suitable for generating all permutations for direct generation of only one permutation without generating and saving preceding permutations. Our method provides efficient hardware realization.*

1. Introduction

The generation of permutations on a given set $A_n = \{a_0, a_1, ..., a_{n-1}\}$ with n elements is in fact the generation of all ordered n-tuples of elements from A_n. This problem occurs frequently in practice as a part of many complex combinatorial problems. For example, many problems in logic design: minimization, simetry examination, NPN classification or function decomposition are combinatorial problems in solving of which different permutations of variables or function values of examined functions are often required [3,6,7]. Important field for application of permutations are permutation interconnect networks which are consistent parts of many multiprocessor systems for discrete transform calculation (DFT, WHT, ...) [4]. In this paper we use generalized decision diagram to generate the permutations.

The basic idea of the presented method was found in the representation of switching functions by binary decision diagram (BDD) [1,2]. BDD for an n-variable switching function is a binary tree with n-levels and 2^n terminal nodes. The terminal node values are the function values of the represented switching function.

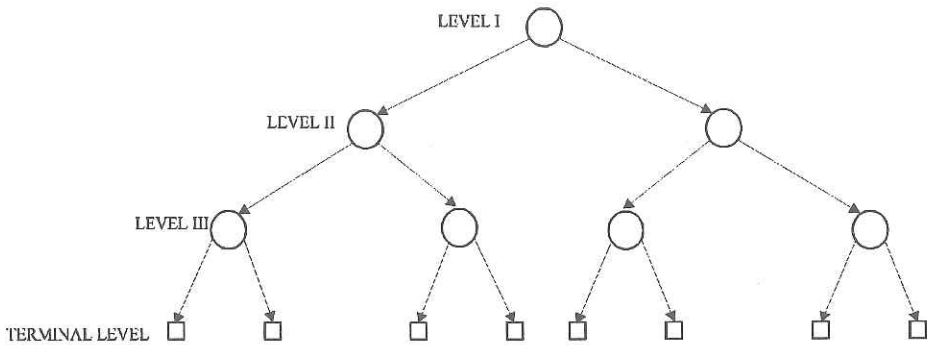

Figure 1. BDD

Example 1. BDD for three variable function is shown in Fig. 1.

If it is allowed that nodes at different levels have different number of edges (assuming that the number of edges of all nodes at one level is equal) we obtain generalized decision diagrams (GDDs) suitable for the representation of the multiple valued functions [5].

Example 2. The typical GDD is shown in Fig 2.

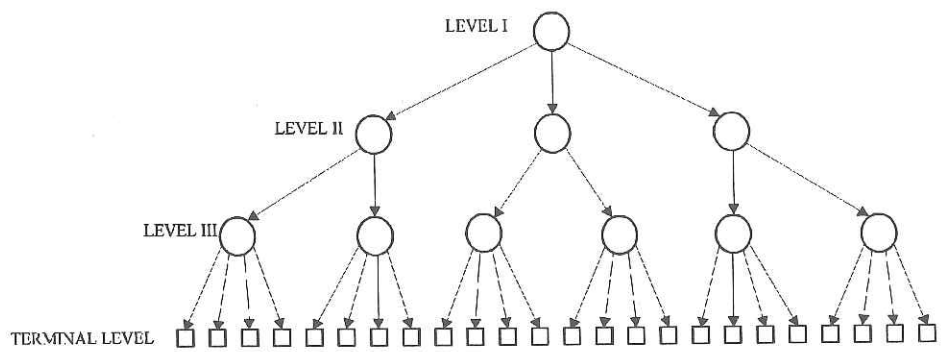

Figure 2. GDD

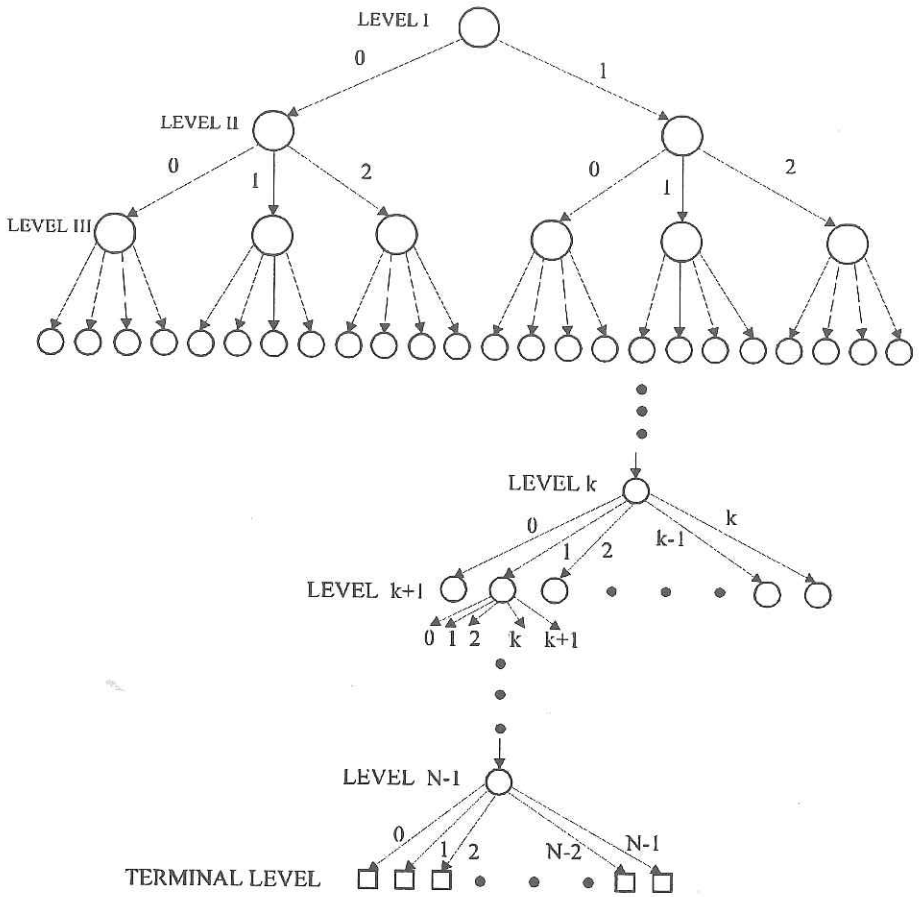

Figure 3. GDD for generation of the permutations

2. The representation of permutations with decision diagram

It is possible to represent all permutations of items from set A_n through particular GDD consisting of $n-1$ levels. The first level consists of one node (root node) with two edges, the second of two nodes with three edges, the third of six nodes with four edges, etc. There are $k!$ nodes with $k+1$ edges at k-th level (Fig. 3).

Thus defined GDD with $n-1$ level has $n!$ terminal nodes. Therefore, we can assign one of $n!$ permutations to each node. The GDD nodes are

denoted as shown in Fig. 4., where we have a node at k-th level denoted by q connected to the root node by the path p. The node q is connected to $k+1$ nodes $(q_0, q_1, \ldots, q_k)$ at (k+1)-st level by the output edges denoted by $0, 1, \ldots, k$, respectively. If the string $x_0 x_1 \ldots x_{k-1}$, $(x_i \in A_n$, for $i = 0, 1, \ldots, k-1)$ is assigned to the node q, to the each node q_i may be assigned the string derived by the following rule:

$$q_i = x_0 x_1 x_2 \ldots x_{i-1} a_k x_i \ldots x_{k-1} \quad for \quad i \neq k$$

$$q_k = x_0 x_1 x_2 \ldots x_{k-1} a_k$$

Each node q_i is connected to the root node by the path $p_i = pi$.
For k=0 (root node) $q = a_0$ and $p = 0$.

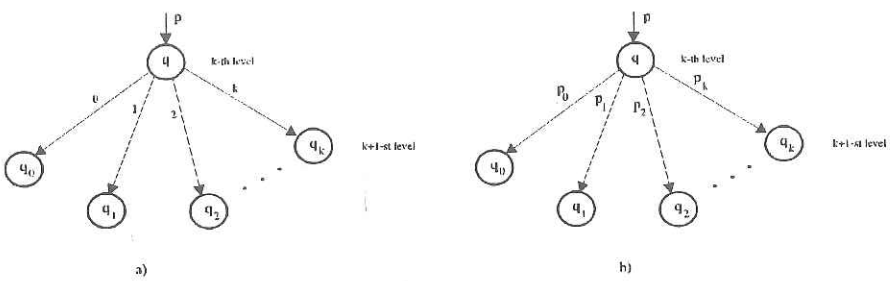

Figure 4. a) The node notation b) The path notation

With the introduced notation, each of the n! terminal nodes corresponds to one permutation, as shown in Fig. 5. for $A_4 = \{0, 1, 2, 3\}$.

3. The procedure for generation of permutations

For the generation of a particular permutation the corresponding path from the root node must be found. Moving from one to another level along this path we generate the required permutation. When we move from the level k to the level $k+1$ trough the edge i we insert the value a_k at i-th position in the generated string. Repeating this procedure for all terminal nodes (moving along all pats in the GDD) we obtain all permutations of n items (Fig. 5).

The decimal index of permutation path, $Dec(p)$, is defined as:

$$Dec = \sum_{i=1}^{N-1} g_i N!/(i+1)!$$

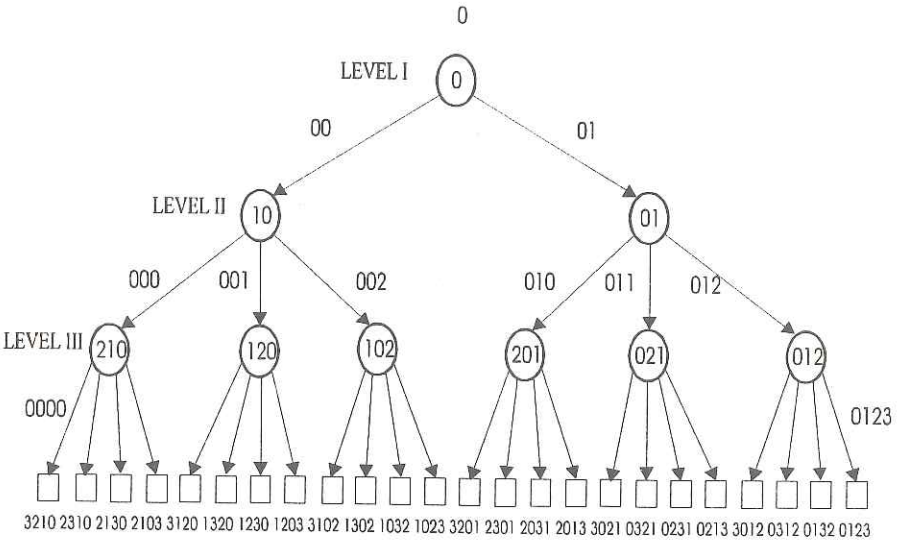

Figure 5. GDD for n=4

where $p[i]$ is i-th element in permutation path p.

All permutations are ordered on the basis of $Dec(p)$. The ordering $\diamond$ can be defined as follows:

Let P and Q be two distinct permutations with paths p and q, respectively.

$$P \diamond Q \quad if \quad Dec(p) < Dec(q).$$

For example, for $n = 4$, the permutations of $A_4 = \{0, 1, 2, 3\}$ ordered according to $\diamond$ are given in Table 1.

This ordering is very useful for generation one permutation or the permutations from interval. We generate the permutation from decimal index. The decimal index to be mapped to the permutation path after which the described procedure is applied. For this method generating and saving all previous permutations are not necesserly. This method is not recursive, which is very important for execution time and permutation length. The permutation length is practically unlimited in this method. No permutation is generated again. Therefore, there is no need to check whether the permutation has been generated earlier.

Table 1: Decimal indices of the permutations

Dec.ind.	path	perm.
0	0000	3210
1	0001	2310
2	0002	2130
3	0003	2103
4	0010	3120
5	0011	1320
6	0012	1230
7	0013	1203
8	0020	3102
9	0021	1302
10	0022	1032
11	0023	1023
12	0100	3201
13	0101	2301
14	0102	2031
15	0103	2013
16	0110	3021
17	0111	0321
18	0112	0231
19	0113	0213
20	0120	3012
21	0121	0312
22	0122	0132
23	0123	0123

4. Implementation

The implementation of the described procedure for the generation of all permutations is given as follows:

1. initialization (the length of the permutations and the beginning path)
2. for i=2,n do
   begin
   2.1 shift all the permutation elements from p-th element for one position to the right (the element p is the weight of the i-th element in the path)
   2.2 set the i-th element at the p-position

end

3. print the generated permutation

4. generate the new path

5. if the generated path is different from the beginning path go to step 2

6. stop

Some advantages of GDD can be used in implementation. There is no need to move along the complete path for each permutation. It may be continued from the position where the new path is different from the old one. In this way the execution time may be decreased considerably with the increase of N, as shown in Table 2 where the execution time is given (in millisecond) using the complete path and a part of the path too.

Table 2: The execution times when the complete path and a part of path are used

len.perm	complete path (ms)	a part of path (ms)
2	0.017	0.023
3	0.05	0.06
4	0.3	0.28
5	1.9	1.4
6	15	9
7	134	70
8	1318	606
9	14240	5830
10	171600	62200
11	2158000	738000

5. The modification of basic procedure

Described procedure can be modified according to some specific requirements of the application of permutations. If the generation of a permutation or permutations from interval are needed, then it is enough to run the corresponding initialization (set the value for array $NIVO(i)$, $i = 0, n-1$). The generation of a permutation from another one is the problem that often appears in practice. In this case, our method is very successful. The move from one permutation to another one is executed by the following procedure:

1. starting from the terminal node corresponding to the beginning permutation and then moving up to the crossing of the beginning and the desired permutations.

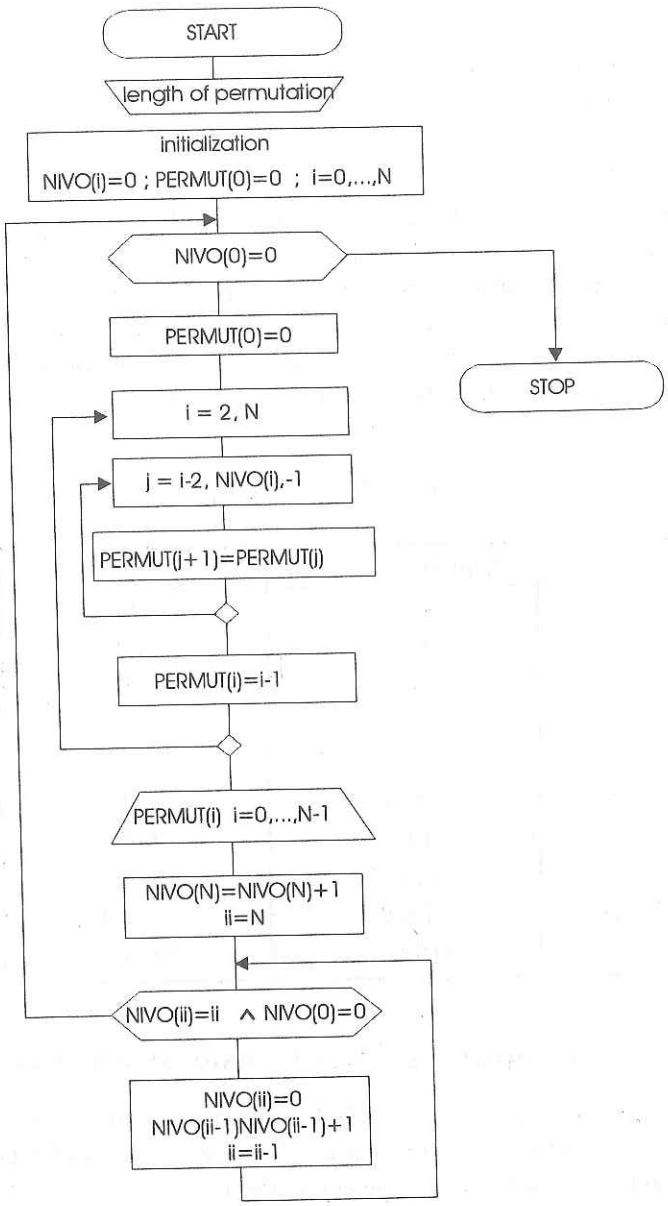

Figure 6. Algorithm for the generation of all permutations with length n

2. moving down to the terminal node corresponding to the desired permutation. Moving up, the elements are ejected from the sequence (i.e. permutation), and moving down, the elements are inserted into the sequence.

The example for the generation of permutation 1032 from 2310 permutation is shown in figure 7. The moving through graph is depicted by a dotted line.

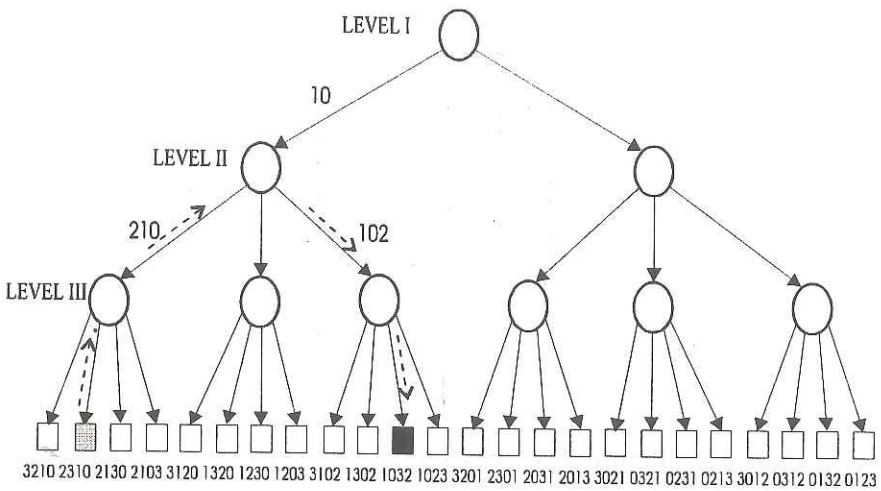

Figure 7. The generation 1032 permutation from 2310 permutation

6. The hardware implementation

Our method provides efficient hardware realization. The types of hardware realization depend on the actual application. As an example, the descriptions of pipeline realization, shown in figure 8, follow. The generation of permutation of n items requires n processing elements (PEs). Every PE has two inputs and one output. PE passes one of the two inputs depending on the state of the counter which runs as the adder modulo k if PE is at k level (i.e. k-th in pipe). If the immediate state of the counter is p, the PE passes p inputs X, (and) afterwards input Y and finally $n-p-1$ inputs X. The counter of PE at level k changes its state when the state of counter of PE on level $k+1$ becomes $k+1$. In other words, every PE activates the counter of the previous PE (Figure 9).

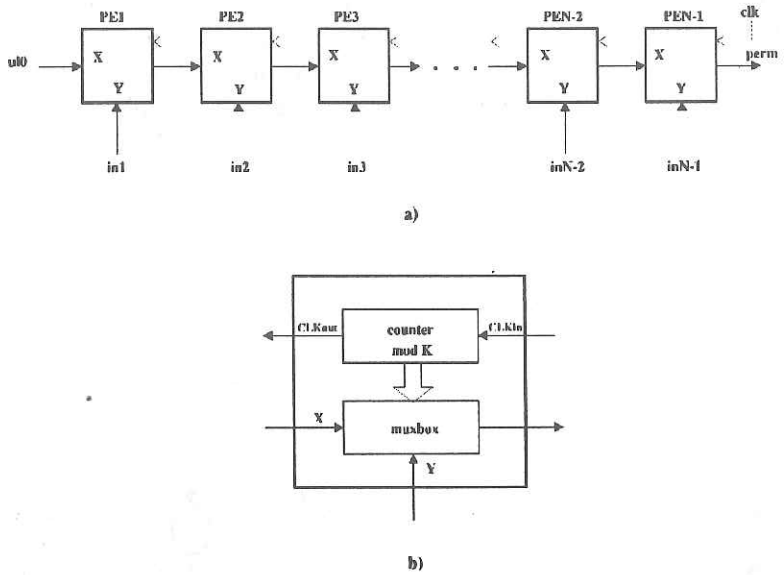

Figure 8. a) Pipeline system b) PE

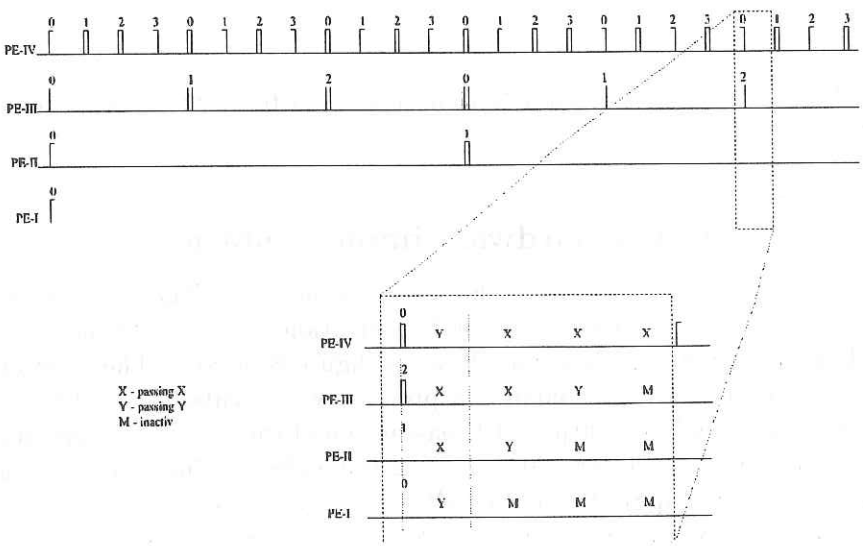

Figure 9. The state diagram of the counter of PEs for n=4

7. Conclusions

In this paper, we propose the method for the generation of the permutations with unlimited length, through GDD. We define GDD appropriated for the generation of all the $n!$ permutations of n given items. Based on GDD, the efficient procedure for mapping the paths in GDD into permutations is also presented. The proposed method is suitable for both software and different hardware realizations. As an example, pipeline realization is described.

References

[1] S. B. Akers, *Binary decision diagram*, IEEE Transaction on Computers **C-27, No. 6** (June 1978), 509–516.

[2] R. Bryant, *Graph-based algorithms for Boolean function*, IEEE Transaction on Computers **C-35, No. 8** (August 1986), 677–691.

[3] D. Cvetković, *Diskretne matematičke strukture*, Naučna Knjiga, Beograd, 1987.

[4] P. Fragopoulou, S. G. Akl, *A parallel algorithm for computing Fourier transforms on the Star graph*, IEEE Transaction on Parallel and Distributed Systems **5, No. 5** (May 1993), 525–531.

[5] D. Janković, R. Stankovi c, M. Nikić, *Calculation of the Fourier transform on finite Abelian groups through GDD* (1994), Proc. Yugoslav Conference for ETRAN, Niš, Yugoslavia.

[6] C. J. Lin, *Parallel generation of permutations on systolic arrays*, Parallel Computing **15** (1990), North-Holland, 267–276.

[7] R. Sedwick, *Permutation generation method*, Computing Surveys, **9, No. 2** (1977), 137–164.

Faculty of Electronic Engineering Computer Science Department, Beogradska 14, 18000 Niš