Applications of a Subset-Generating Algorithm to Base Enumeration, Knapsack and Minimal Covering Problems

I. STOJMENOVIĆ* AND M. MIYAKAWA†

* *Institute of Mathematics, University of Novi Sad, dr Ilije Djuricica 4, 21000 Novi Sad, Yugoslavia*
† *Electrotechnical Laboratory, 1-1-4 Umezono, Sakura-mura, Niihari-gun, Ibaraki 305, Japan*

On the basis of a backtrack procedure for lexicographic enumeration of all subsets of a set of n elements, we give an algorithm both for determining all bases consisting of functions from a given complete set in a considered subset of the set of k-valued logical functions, and for enumeration of all classes of bases in the subset. We use the lexicographic algorithm also for solving knapsack and minimal covering problems. A cut technique is described which is used in these algorithms to reduce the number of examined subsets of {1, ..., n}.

Received May 1986

1. GENERATING ALL SUBSETS OF {1, ..., n} IN LEXICOGRAPHIC ORDER

In this section we consider the problem of generating all r-subsets (subsets containing r elements) of the set {1, 2, ..., n} for $1 \leqslant r \leqslant n$ and for $1 \leqslant r \leqslant m \leqslant n$. We assume that each subset will be represented as a sequence $a_1a_2\ldots a_r$, where $1 \leqslant a_1 < \ldots < a_r \leqslant n$.

Recall definition of lexicographic order of subsets. For two subsets $a = (a_1, \ldots, a_p)$ and $b = (b_1, \ldots, b_q)$, $a < b$ is satisfied if and only if there exists i $(1 \leqslant i \leqslant q)$ such that $a_j = b_j$ for $1 \leqslant j < i$ and either $a_i < b_i$ or $p = i-1$. This order has an important property that enables simple calculation with r-subsets.

Ehrlich[2] described a loopless procedure for generating subsets of a set of n elements. A procedure based on Gray code for the same problem is given in Ref. 13, where an algorithm for generating all r-subsets $(1 \leqslant r \leqslant m \leqslant n)$ in lexicographic order is also proposed. Semba[18] improved the efficiency of the algorithm; we shall modify his algorithm by presenting it in Pascal-like notation without goto statements. Application of the algorithm for minimal covering problems results in another modification of the algorithm in the case $1 \leqslant r \leqslant m \leqslant n$.

The lexicographic enumeration of r-subsets goes in the following manner (for example, let $n = 5$):

```
1,  12,  123,  1234,  12345,
               1235,
         124,  1245,
         125,
    13,  134,  1345,
         135,
    14,  145,
    15,
2,  23,  234,  2345,
         235,
    24,  245,
    25,
3,  34,  345,
    35,
4,  45,
5.
```

The algorithm is in *extend* phase when it goes from left to right staying in a row. If the last element of a subset is n, the algorithm shifts to the next row. We call this the *reduce* phase.

Every subset of {1, ..., n} is represented in the algorithm below by a sequence $j_1, \ldots, j_r$, $1 \leqslant r \leqslant n$, $1 \leqslant j_1 < \ldots < j_r \leqslant n$.

First we give an algorithm for generating all r-subsets for $1 \leqslant r \leqslant n$. This algorithm will be used in base enumerations.

```
begin
  read (n);  r: = 0;  j_r: = 0;
  repeat
    if j_r < n then extend else reduce;
    print out j_1,...,j_r
  until j_1 = n
end;
  extend ≡ begin j_{r+1}: = j_r + 1; r: = r + 1 end
  reduce ≡ begin r: = r − 1; j_r: = j_r + 1 end.
```

Note that between any two printed subsets exactly two conditions are checked $j_r < n$ and $j_1 = n$.

The algorithm for generating all r-subsets for $1 \leqslant r \leqslant m \leqslant n$ we modify with respect to its use in minimal covering problems.

```
begin
  read (n); r: = 0; j_r: = 0;
    repeat
      if j_r < n and r < m then extend else cut;
      print out j_1,...,j_r
    until j_1 = n
end;
  extend ≡ begin j_{r+1}: = j_r + 1; r: = r + 1 end
  reduce ≡ begin r: = r − 1; j_r: = j_r + 1 end
  cut ≡ if j_r < n then j_r: = j_r + 1 else reduce.
```

Besides *extend* and *reduce* phases we use in the algorithm a new phase called *cut*. The phase will be used when the algorithm goes from one subset to a subset in a lower row (not necessarily in the subsequent row), skipping several subsets (when the number r of elements in these subsets is greater than m).

2. FUNCTIONAL COMPLETENESS AND ENUMERATION OF BASES

In this section we describe an application of our lexicographic algorithm to base enumeration for a subset of the set of k-valued logical functions.

Let $E_k = \{0, \ldots, k-1\}$. The set of k-valued logical functions, i.e. the union of all the functions $\{\mid E_k^n \rightarrow E_k$ for $n = 0, 1, 2, \ldots\}$ is denoted by P_k is said to be **closed**

3 CPJ 31

if it contains all superpositions of its members (cf. Refs. 4, 5 and 16). For closed sets F and H such that $F \subset H$ (proper inclusion), F is **H-maximal** set if there is no closed set G such that $F \subset G \subset H$. A subset X of H is **complete in *H*** if H is the least-closed set containing X. If the number d of H-maximal sets is finite a subset of functions in H is complete in H if and only if it is not contained in any one H-maximal set (completeness condition) (cf. Ref. 5). Investigations of completeness and related topics, which are usually called functional completeness problems, are directly related to logical circuit design, and they have a wide area of applications in addition to their mathematical importance.

A complete set X in H is called a **base of *H*** if no proper subset of X is complete in H. A set of functions $F = \{f_1, \ldots, f_s\}$ is called **non-redundant in *H***, if for each i, $1 \leqslant i \leqslant s$, there exists an H-maximal set H_j, $1 \leqslant j \leqslant d$ which does not contain f_i, while all the other functions f_l ($l = 1, \ldots, s, l \neq i$) are elements of H_j (non-redundancy condition). From these definitions it follows that a complete non-redundant set is a base. We call non-redundant incomplete sets simply **addable**. The **rank** of a base (addable set) is the number of its elements.

We classify the set H of functions into non-empty equivalence classes by using all its maximal sets as indicated below. Then we can discuss the completeness properties in H in terms of these classes instead of individual functions; if a set is complete (non-redundant), replacing a function in the set by any function in the corresponding equivalence class yields another complete (non-redundant) set.

The **characteristic vector** of $f \in H$ is $c_1 \ldots c_d$, where $c_i = 0$ if $f \in H_i$ and $c_i = 1$ otherwise ($1 \leqslant i \leqslant d$). Whenever it is possible to avoid confusion we call characteristic vectors simply vectors. All functions $f \in H$ with the same (characteristic) vector form a **class of functions**. For a base its **class of bases** is the set of classes of functions for functions belonging to the base.

The conditions of completeness and non-redundancy of a set of (classes of) functions F can be conveniently expressed by using characteristic vectors of (classes of) functions belonging to F. We can say that a base corresponds to a minimal cover of $1\ldots1$ (unit vector), and a non-redundant set corresponds to a minimal cover of some non-unit vector (in which some 0s may occur; we except null vector).

We define bitwise **or** operation $\vee$ for characteristic vectors in the following way:

$$(a'_1, \ldots, a'_d) \vee (a''_1, \ldots, a''_d) = (a'_1 \vee a''_1, \ldots, a'_d \vee a''_d).$$

Criteria for the completeness and non-redundancy of a set $a_1, \ldots, a_r$ of characteristic vectors are respectively the following:

(1) $a_1 \vee \ldots \vee a_r = 1 \ldots 1$ (completeness) (2.1)

(2) $a_1 \vee \ldots \vee a_{j-1} \vee a_{j+1} \vee \ldots \vee a_r \neq a_1 \vee \ldots \vee a_r$ for each $j = 1, \ldots, r$ (non-redundancy). (2.2)

Thus any set containing null class (whose vector is $0\ldots0$) is redundant. Addable sets are non-redundant, but not conversely.

If we have a complete list of characteristic vectors for non-empty classes of functions of a set, we can enumerate all its classes of bases.

As an example, assume a set M contains 4 maximal sets M_1, M_2, M_3, M_4 and 6 classes of functions: (1) 0011; (2) 0100; (3) 1000; (4) 0010; (5) 0001; (6) 0000.

For instance, class 1 is the set $M_1 \cap M_2 \cap \overline{M}_3 \cap \overline{M}_4$, where $\overline{X} = M \backslash X$ (complement set).

M has exactly two classes of bases: {1, 2, 3} and {2, 3, 4, 5}. We consider the class {1, 2, 3}. Bitwise **or** for the set results 1111 (completeness). Bitwise **or** for the set {1, 2} results 0111, for the set {1, 3} results 1011 and for the set {2, 3} results 1100 (non-redundancy).

The set {1, 3, 4} is redundant, because bitwise or for the sets {1, 3, 4} and {1, 3} are equal (to 1011).

3. THE LEXICOGRAPHIC ENUMERATION OF BASES AND CLASSES OF BASES

Let d and n denote the numbers of maximal sets and functions or classes of functions respectively. Then we are given n vectors with length d, indexed by $1, \ldots, n$.

To perform an exhaustive enumeration of classes of bases we should enumerate every r-tuple of vectors $a_1, \ldots, a_r$ for each $r = 2, \ldots, d$ (for $r = 1$ it is trivial) and check the completeness (2.1) and redundancy (2.2) conditions for them (rank r base criteria). However, this direct method does not work because of too many r-tuples to be generated. Suppose we are enumerating r vectors $a_1, \ldots, a_r$ for checking the base criteria. Instead of enumerating whole r vectors and checking criteria for them, we will inspect i-tuple of vectors $a_1, \ldots, a_i$ incrementary for $i = 1, \ldots, r$, and at each ith stage we will certify (by examining simple conditions) that this i-tuple can or cannot be included in a rank r base (addable set). This idea of incremental check can be conveniently implemented in the lexicographic enumeration of subsets.

The lexicographic algorithm enumerates classes of bases and addable sets for every rank at the same time. Moreover, the maximal ranks of bases and addable sets are automatically given as a result.

Suppose we are enumerating taken r elements out of n objects stored in an array consecutively, i.e. $a(1), \ldots, a(n)$. The selected indexes are to be stored in an array j as $j_1, \ldots, j_r$, $1 \leqslant j_i \leqslant n$ for each i, $1 \leqslant i \leqslant r$.

Suppose we are examining taken r-subset $a(j_1), \ldots, a(j_r)$, where selected indexes are stored in an array j as $j_1, \ldots, j_r$, $1 \leqslant j_1 < \ldots < j_r \leqslant n$ and $a(i)$ denotes a_i. There are three possible cases after the examination: redundant, base and addable set (i.e. non-base and non-redundant). The enumeration of subsets in lexicographic order can be controlled in the following manner.

If a r-tuple is either redundant or base it is unnecessary to *extend* it to $r+1$-tuple, since adding a new vector to them will result in redundancy; in the former case the r-tuple is already redundant and in the latter it is already complete. Hence in these cases we can bypass the lexicographic enumeration of subsets to an appropriate point. The next subset is $j_1, j_2, \ldots, j_{r-1}, j_r+1$ if $j_r \neq n$; otherwise it is the next subset in lexicographic order and the bypass effects nothing. Thus only the remaining addable case can be extended.

As an example we consider the same set M as before. The class 6 (null class) is omitted. In this case $n = 5$ and $d = 4$. The notions *extend*, *reduce*, *cut*, 'redundant', 'base' and 'addable' we denote simply by e, r, c, n, b, a respectively.

{1}−a, e; {1, 2}−a, e; {1, 2, 3}−b, c;
{1, 2, 4}−n, c;
{1, 2, 5}−n, c, r;
{1, 3}−a, e; {1, 3, 4}−n, c;
{1, 3, 5}−n, c, r;
{1, 4}−n, c;
{1, 5}−n, c, r;
{2}−a, e; {2, 3}−a, e; {2, 3, 4}−a, e; {2, 3, 4, 5}−b, c, r;
{2, 3, 5}−a, r;
{2, 4}−a, e; {2, 4, 5}−a, r;
{2, 5}−a, r;
{3}−a, e; {3, 4}−a, e; {3, 4, 5}−a, r;
{3, 5}−a, r;
{4}−a, e; {4, 5}−a, r;
{5}−a.

We can write our algorithm as follows. Let b_r be the number of (classes of) bases of rank r.

begin
  read n, d, a(i), i: $= 1, n$; $r := 1$; $j_1 := 1$;
  repeat
    if $a(j_1), \ldots, a(j_r)$ *is addable*
      then if $j_r < n$
        then *extend*
        else *reduce*
      else begin
        if $a(j_1), \ldots, a(j_r)$ *is a base* ***then*** $b_r := b_r + 1$; *cut*;
      end
  until $j_1 = n$;
  print out b_i, $1 \leqslant i \leqslant d$
end.

In the algorithm *extend*, *reduce* and *cut* are defined as before.
Note that the last set $\{n\}$ is not checked in the algorithm. It can easily be done before printing results.

4. REDUNDANCY CHECKS

We describe a technique (called bitwise pivotality checks) to reduce the computation in redundancy checks.

Suppose we are checking redundancy of $a_1, \ldots, a_r$ (for simplicity we write a_i for $a(j_i)$). For every redundancy check we know that $a_1, \ldots, a_{r-1}$ are included in the tuple which we examined just before (only a_r is a newly added vector). Thus we can assume that we already have $R_k = a_1 \vee \ldots \vee a_k$ for $1 \leqslant k \leqslant r-1$ in an array R (for a convenience we add R_0 and assume $R_0 = 0$).

The redundancy condition for the r-tuple can be formulated in the following way (we use a variable B to reduce the number of bitwise **or** operations).

For $r \geqslant 2$.

$$R_r = R_{r-1} \vee a_r \text{ and } R_{r-1} \neq R_r, \quad (4.1)$$
$$B = B \vee a_{k+1} \text{ (initial } B = 0\text{) and}$$
$$R_{k-1} \vee B \neq R_r \text{ for } k = r-1, \ldots, 1 \quad (4.2)$$

For $r = 1$.

a_1 is addable if it is neither null vector nor unit vector (if a_1 is a unit vector then it is a base)

The program checks (4.1) and (4.2) for $k = r, \ldots, 1$; $k \geqslant 2$ in this order, and whenever a condition is not satisfied the check ends immediately with redundancy result.

For a rank r redundancy check we need at most r comparisons and at most $2r-1$ bitwise **or** operations.

If the number of components d in vectors a_i is less than the number of bits (usually 16 or 32) of given computer it is possible to represent a vector a_i by an integer number $c_1 + 2 \cdot c_2 + \ldots + 2^{d-1} \cdot c_d$, where $c_1 c_2 \ldots c_d$ are the components of the vector a_i. In the redundancy check we can treat these vectors as integer numbers because **or** operation between integer numbers is defined as a machine instruction **or** between corresponding components of their binary notations.

Otherwise bitwise **or** can be realized with (characteristic) vectors as an array of d elements. However, in this case there is another technique called counter-redundancy check which is proved faster as well.

In the check of redundancy we use two auxiliary sequences s_i $(1 \leqslant i \leqslant d)$ and p_i $(1 \leqslant i \leqslant r)$. s_i is the number of units in the ith position in the vectors $p(j_1), \ldots, p(j_{r-1})$. The sequence $p_1, \ldots, p_r$ has the following property: p_ith position of each vector is equal to 1 only for $p(j_i)$ (it is equal to 0 for the vectors $p(j_t)$, $1 \leqslant t \leqslant r$, $t \neq i$).

The lexicographic algorithm presented can also be supplemented with this technique.

Note that the algorithm with bitwise redundancy check using machine command is proved to be about twice as fast (when n is about 500 and d is about 15) than one with counter-redundancy check.

Applying this algorithm, classes of bases for several subsets of P_k are determined (cf. Ref. 12).

P_3 has exactly 18 maximal sets[5] and 406 classes of functions[10, 19]. We present the numbers of classes of bases of P_3 of each rank in Table 1. The lexicographic enumeration algorithm with this bitwise redundancy check requires about 16 minutes computer time (the computer FACOM M380 is used). The total number of examined tuples is N = 194759642 for the classes of functions sorted according first to the number of units in the vector, and then sorted lexicographically within the same group. Bearing in mind the total number of subsets 2^{406}, we can calculate the efficiency of cut technique in this case. The program generates in the average 4.41-tuple and consumes in the average 2.17 bitwise or operations to recognize whether it is a base, addable or redundant (bitwise redundancy check is used). Note that computer time depends on the order of characteristic vectors.

5. APPLICATION OF THE BASE ENUMERATION ALGORITHM

Kabulov[6] considered the following problem. Given a complete set F of functions from P_k together with the Boolean matrix displaying the relation $\in$ between the

Table 1.

Rank	1	2	3	4	5	6	Σ
Bases	1	8265	794256	4612601	810474	141124	6239721

3-2

members of F and maximal sets in P_k (i.e. with characteristic vectors of functions in F), determine all bases composed from functions of the set F. He described a method, using Boolean expressions, to solve this problem.

We can apply the same algorithm decribed in section 3 to this problem, because each function is represented by its class of functions. The output in this case is exactly bases instead of classes of bases. Note that in the considered application several functions may have the same characteristic vector. However, they compose different bases.

Our algorithm can be used to calculate the number of (classes of) bases composed from vectors $m+1, \ldots, n$ at the same time (for a given $m \leqslant n$), because in the lexicographic order we examine first all subsets containing vector 1, then all subsets containing vector 2, etc.

In Refs. 9, 14 and 20 procedures for determining the number of bases of P_2 consisting of n-ary functions are described and computational results for $n = 2$ and $n = 3$ are obtained. There exist no formulae for numbers of n-ary functions in some classes of function of P_2, because the number of n-ary monotone functions in P_2 is not known. We present another approach to this problem. It is divided into several subproblems.

(1) determination of classes of functions for considered set (not limited to P_2),
(2) determination of the number of n-ary functions in each class,
(3) determination of all classes of bases,
(4) determination of numbers of bases containing n-ary functions (or functions with at most n variables).

The methods presented in Refs. 9, 14 and 20 use only step (4) for P_2. Our method can be applied for solving (3) assuming that (1) is already solved. Also, our algorithm can be applied for solving (4) assuming that (2) is solved by applying another procedure. Note that (2) can be done without solving (1) because for each function f we can determine a corresponding class of functions. It is sufficient to check inclusion of f in each maximal set of considered closed set; such procedure can be easily written using description of maximal sets[16]. In this manner we can determine classes of functions containing n-ary functions. We can apply our algorithm to count bases. We obtain the number of bases containing n-ary functions in a class of bases by multiplying the numbers of n-ary functions in the classes of functions which compose the base, whenever a class of bases is found. During this procedure we can also enumerate classes of bases consisting of classes of n-ary functions.

Following this description we determined the number of bases of Boolean functions composed from n-ary functions for $n \leqslant 4$. The data obtained are presented in Table 2. For $n = 2$ this result is derived by Wernick,[20] and for $n = 3$ by Kudielka and Oliva[9]. Note that the set P_2 of Boolean functions contains 5 maximal sets[15], 15 classes of functions[4,3 and 8] and 42 classes of bases [3,8].

Table 2.

n	2	3	4
Bases	32	6664	275790502

6. MINIMAL COVERING PROBLEM

Minimal covering problem is a famous combinatorial problem and there exist a list of solutions for it (cf. Refs. 17 and 21). We will give a solution using the lexicographic enumeration of subsets.

The minimal covering problem is the problem of minimising the objective function $x_1 + \ldots + x_n$, subject to constraints

$$(x_1, \ldots, x_n) A \geqslant (1, \ldots, 1) \qquad (6.1)$$

where $A = [a_{ij}]$ is an $n \times d$ coefficient matrix with $a_{ij} = 0$ or 1, and each variable x_j is 0 or 1 for each j.

We will introduce some new notions in order to give a new solution for the problem and to show connection between minimal covering problem and base enumeration.

A vector $(x_1, \ldots, x_n)$ satisfying (6.1) is called **complete** for A. We call a vector $(x_1, \ldots, x_n)$ **non-redundant** in A if

$$(x_1, \ldots, x_n) A > (y_1, \ldots, y_n) A$$

is valid for each vector $(y_1, \ldots, y_n)$ for which $y_i \leqslant x_i$ for each i, $1 \leqslant i \leqslant n$ and $y_1 + \ldots + y_n < x_1 + \ldots + x_n$ is satisfied.

A vector $(x_1, \ldots, x_n)$ is called **base** in A if it is complete and non-redundant in A. Non-redundant non-complete vectors we call simply **addable**. The **rank** of a base (addable set) $(x_1, \ldots, x_n)$ is the sum $x_1 + \ldots + x_n$. Thus minimal covering problem is the problem of finding a base in A with minimal rank.

There is another definition of minimal covering problem[7]. For a given collection C of subsets of a finite set and positive integer $r \leqslant |C|$ decide whether C contains a cover for S of size r or less, i.e. a subset $C' \subseteq C$ with $|C'| \leqslant r$ such that every element of S belongs to at least one member of C'. This problem is exactly to find a base with rank r or less, if we represent a subset by n bits characteristic vector. Karp[7] proved that this problem is NP-complete.

The notions of addable sets, bases and rank have almost the same meaning in both base enumeration and minimal covering problem. Minimal covering problem corresponds directly to finding a base with minimal rank. This we can modify our algorithm so that once we find a base with rank r no subset of rank $\geqslant r$ will be considered further.

In the presented branch and bound algorithm $a(i)$ denotes the ith row of matrix A $(1 \leqslant i \leqslant n)$, i.e. $a(i) = (a_{i1}, \ldots, a_{in})$. We suppose that minimal rank of bases (solution of our problem) is between 2 and $n-1$ to make our algorithm shorter. It is easy to improve our algorithm to deal with these cases. Also some techniques for eliminating some rows or columns (cf. Ref. 17) can be applied before running the algorithm.

```
begin
  read n, d, a(i), i: = 1, n; minrank: = d; r: = 1; j1: = 1;
  T: = {1};
    repeat
      if a(j1), ..., a(jr) is addable in A
        then if jr < n and r < minrank - 1
          then extend
          else cut
        else begin
          if a(j1), ..., a (jr) is a base in A then
```

```
        begin
          minrank = r;
          t: = {j1, ..., jr};
        end;
      cut
    end
  until j1 = n or minrank = 2;
  print out minrank, T
end.
```

extend and *cut* are defined as before. Note that T corresponds to a solution $(x_1, \ldots, x_n)$ of minimal covering problem so that $x_j = 1$ if and only if $j \in T$.

7. KNAPSACK PROBLEM

An input for the knapsack problem consists of integer numbers $a_1, \ldots, a_n, C$. The problem is to find a subset T of $\{1, \ldots, n\}$ to maximise $\Sigma_{i \in T} a_i$ subject to the requirement that $\Sigma_{i \in T} a_i \leqslant C$. A more general formulation of the knapsack problem has more applications than this. Namely the input consists of C and two sequences $a_1, \ldots, a_n$ and $p_1, \ldots, p_n$. The problem is to maximise $\Sigma_{i \in T} p_i$ subject to the restraint $\Sigma_{i \in T} a_i \leqslant C$ where T, as before, is a subset of the indexes.

We give a solution for a more general knapsack problem based on the lexicographic order of subsets. Elements i that are a_i greater than C should be eliminated. In the presented algorithm $a(j_i)$ denotes a_{j_i}.

```
begin
  read n, d, ai, pi, i = 1, n;
  r: = 1; j1: = 1; maxsum: = p1; T: = {1};
  repeat
    S: = a(j1)+ ... +a(jr);
    if S ≤ C
      then begin
        P: = p(j1)+ ... +p(jr);
        if P > maxsum then begin
                             maxsum: = P;
                             T: = {j1, ..., jr}
                           end;
        if jr < n then extend else reduce
      end
      else cut;
  until j1 = n;
  print out maxsum, T
end.
```

In the algorithm *extend*, *reduce* and *cut* are defined as before. The set $\{n\}$ should be examined before printing.

8. CONCLUDING REMARKS

In this paper we modified backtrack procedures for lexicographic enumeration of subsets and applied the procedure to the base enumeration, knapsack and minimal covering problems. Several variational uses of base enumeration algorithm are presented. The presented *cut* techniques use special properties of bases and addable sets, owing to which, for instance, base enumeration was possible for about $n = 600$ (for the case $n = 605$, $d = 15$ it took about 8 hours using bitwise redundancy check by FACOM 380 computer with 24 mips).

Karp[7] proved that the problem of determining a covering set with rank $\leqslant r$ for given r is NP-complete. Our algorithms are directly related to the problem. Thus any algorithm for solving these problems takes exponential time according to numbers of rows and columns n and d. There exist a number of algorithms for exact and approximate solution of knapsack and minimal covering problems (see. for example, Refs. 1, 17 and 21).

Acknowledgements

This work was done during the first author's stay at the Electrotechnical Laboratory, Ibaraki, between 17 November 1985 and 14 February 1986. The authors thank Drs Nobuyuki Otsu, Shinji Umeyama, Takio Kurita and Hideki Asoh for their kind help. They also express their gratitude to Drs Akio Tojo and Kiochiro Tamura for their encouragement to the project. Special thanks go to RIPS (Research Information Processing System) for offering computational facilities. Financial support by the Science and Technology Agency, the Government of Japan, is acknowledged.

REFERENCES

1. S. Baase, *Computer Algorithms: Introduction to Design and Analysis.* Addison-Wesley, Reading, Mass. (1978).
2. G. Ehrlich, Loopless algorithms for generating permutations, combinations and other combinatorial configurations. *J. ACM* **20** (3) (1973), 500–513.
3. K. Ibuki, K. Naemura and A. Nozaki, General theory of complete sets of logical functions. *IECE of Japan* **46**, 7 (1963), 934–940.
4. S. V. Jablonskij, On superpositions of the functions of algebra of logic (in Russian). *Matematicheskii Sbornik* **30** (72) 2 (1952), 329–348.
5. S. V. Jablonskij, Functional constructions in a k-valued logic (in Russian). *Trudi Matematicheskogo instituta imeni V. A. Steklova* **51** (1958), 5–142.
6. A. V. Kabulov, Synthesis of bases of complete systems of logical functions (in Russian). *Doklady Akademii Nauk UzSSR* (1982), no. 4, 3–5.
7. R. M. Karp, Reducibility among combinatorial problems. In *Complexity of Computer Computations*, edited R. E. Miller and J. W. Thatcher, pp. 85–103, Plenum Press, New York (1972).
8. L. Krnic, Types of bases in the algebra of logic (in Russian). *Glasnik Matematičko-fizički i astronomski* **20** (1–2) 23–32 (1965).
9. V. Kudielka and P. Oliva, Complete sets of functions of two and three binary variables. *IEEE Transactions on Electronic Computers* **EC-15** (1966), 930–931.
10. M. Miyakawa, Functional completeness and structure of three-valued logics. I-Classification of P_3 *Researches, Electrotechnical Laboratory* **717** (1971), 1–85.
11. M. Miyakawa, Enumerations of bases of three-valued logical functions. In *Coll. Math. Soc. J. Bolyai* **28** 469–487, Szeged, 1979, North-Holland, 1981.
12. M. Miyakawa and I. Stojmenović, Classifications and base enumerations of the maximal sets of three-value logical functions. Comptes Rendus Mathematiques de l'Academie des Sciences Canada **9** (2) (1987), to be published.
13. A. Nijenhuis and H. S. Wilf, *Combinatorial Algorithms*, 2nd ed. Academic Press, New York (1978).

14. S. R. Petrick and G. C. Sethares, On the determination of complete sets of logical functions, *IEEE Transactions on Computers* **C-17**, 3 (1968), 273.
15. E. L. Post, The two-valued iterative systems of mathematical logic, *Annals of Mathematics Studies* **5**, Princeton University Press (1941).
16. I. G. Rosenberg, Completeness properties of multiple-valued logic algebra. In *Computer Science and Multiple-valued logic: Theory and Applications* edited D. C. Rine, 144–186. North-Holland Amsterdam (1977).
17. R. Roth, Computer solutions to minimum-cover problems, *Operational Research* **17** (1969), 455–473.
18. I. Semba, An efficient algorithm for generating all k-subsets ($1 \leqslant k \leqslant m \leqslant n$) of the set $\{1, 2, \ldots, n\}$ in lexicographic order, *Journal of Algorithms* **5** (1984), 281–283.
19. I. Stojmenović, Classification of P_3 and the enumeration of bases of P_3, *Rev. of Res., Fac of Sci., math. ser., Novi Sad,* **14**, 1 (1984), 73–80.
20. W. Wernick, Complete sets of logical functions, *Transactions of the American Mathematical Society* **51** (1942), 117–132.
21. M. H. Young and S. Muroga, Symmetric minimal covering problem and minimal PLA's with symmetric variables, *IEEE Transactions on Computers* **C-34**, 6 (1985), 523–541.