The size of arrays for a prime implicant generating algorithm

Y. Igarashi*

Computer Science Division, Department of Mathematics, The City University, London EC1V 4PB

A number of interesting combinatorial problems have arisen by investigating a computer implementation of a prime implicant generator called the star-algorithm. These problems have been motivated by the necessity of estimating suitable sizes of arrays to store intermediate results when the algorithm is written in a computer language such as ALGOL 60. $B_n(r)$ denotes the set of r-cubes of the n-variable Boolean algebra and $\circledast$ denotes the star-product. $P(B_n(r)) = \#(\{b \mid b \in B_n(r) \text{ and } a \circledast b \neq \&\})$, where $\#(S)$ is the number of elements of set S, a is an arbitrary element of $B_n(r)$, and $a \circledast b \neq \&$ means that the star-product of a and b is a cube. The main combinatorial result in this paper is that for all $0 < \epsilon$, $k_1(n\,3^{3/4})^n \leqslant \max\{P(B_n(r)) \mid 0 \leqslant r \leqslant n\} \leqslant k_2(3^{3/4} + \epsilon)^n$. where k_1 and k_2 are constants independent of n.

(Received June 1978)

1. Preliminary

In the main we employ definitions and notations used in standard texts of switching theory (Harrison, 1965; Miller, 1965). If d is a real number, $\lfloor d \rfloor$ denotes the largest integer k such that $k \leqslant d$. The i-th Boolean variable is normally denoted by x_i or x_i^1. The complement of x_i is denoted by $\bar{x}_i$ or x_i^0, and is occasionally called the negation of x_i. A literal means a variable or negated variable. The universal upper bound and universal lower bound of the Boolean algebra are denoted by I and $\emptyset$ respectively. For convenience we shall often identify a Boolean formula with the Boolean function expressed as the formula. If S is a set, $\#(S)$ denotes the number of elements of S.

Let e_{i_j} $(1 \leqslant j \leqslant n - r)$ be 0 or 1. Then $x_{i_1}^{e_{i_1}} \ldots x_{i_{n-r}}^{e_{i_{n-r}}}$ is called an r-cube (or r-implicant) over n variables, where

$$x_{i_j}^{e_{i_j}} = \begin{cases} \bar{x}_{i_j} \text{ if } e_{i_j} = 0 \\ x_{i_j} \text{ if } e_{i_j} = 1 \end{cases} \quad \text{for } j = 1, \ldots, n - r,$$

$1 \leqslant i_j \leqslant n$, and $i_s \neq i_t$ if $s \neq t$. We shall occasionally use the following notation to express a cube: A cube of the n-variable Boolean algebra is expressed as an element of $S_n = \{z_1 \ldots z_n \mid$ for each $1 \leqslant i \leqslant n$ $z_i \in \{0, 1, 2\}\} \cup \{\emptyset\}$. The interpretation of 0, 1 or 2 at the i-th position of $z = z_1 \ldots z_n$ is $\bar{x}_i$, x_i or $\bar{x}_i \vee x_i$, respectively. For convenience we shall often identify a cube with its corresponding expression in S_n.

Definition 1

Relation $\leqslant$ and $<$ on n-variable Boolean formulae are defined as follows: $\emptyset < I$. $a \leqslant b$ means $a < b$ or $a = b$, where a and b are in $\{\emptyset, I\}$. For a pair of n-variable Boolean formulae f and g, $f \leqslant g$ if and only if for all $(a_1, \ldots, a_n) \in \{\emptyset, I\}^n$ $f(a_1, \ldots, a_n) \leqslant g(a_1, \ldots, a_n)$. $f < g$ means that $f \leqslant g$ and for at least one $(a_1, \ldots, a_n) \in \{\emptyset, I\}^n$ $f(a_1, \ldots, a_n) < g(a_1, \ldots, a_n)$, where $q(a_1, \ldots, a_n)$ is the evaluation of Boolean formula q when x_i is set to be a_i $(1 \leqslant i \leqslant n)$.

Definition 2

A cube p is called a prime implicant of an n-variable Boolean formula E if and only if $p \leqslant E$ and there does not exist a cube w such that $p < w \leqslant E$.

Definition 3

Let P be a partially ordered set. A subset R of P is called an antichain of P if and only if for any pair of a and b in $R(a \neq b)$ there is no relation between a and b. An antichain Q of P is called a maximum sized antichain if and only if for any antichain B of P $\#(B) \leqslant \#(Q)$.

Kleitman, Edelberg and Lubell (1971) have shown that for any partially ordered set there exists a maximum sized antichain which is invariant under any automorphism on the partially ordered set. From this result Chandra and Markowsky (1976) have derived the next lemma.

Lemma 1

The set of $\lfloor (n + 1)/3 \rfloor$-cubes is a maximum sized antichain of S_n.

As a corollary of the above lemma, an upper bound $\binom{n}{\lfloor (2n+1)/3 \rfloor} 2^{\lfloor (2n+1)/3 \rfloor}$ for the maximum number of prime implicants of n-variable Boolean formulae is derived (Chandra and Markowsky, 1976). This is the best upper bound known for it. The best lower bound known for the maximum number of prime implicants of n-variable Boolean formulae is $n!/(\lfloor n/3 \rfloor! \lfloor (n + 1)/3 \rfloor! \lfloor (n + 2)/3 \rfloor!) + g(n, \lfloor (n + 2)/3 \rfloor - 2) + g(n, \lfloor (n + 1)/3 \rfloor - 2)$, where $g(n, r)$ is evaluated by the following recursive procedure: $g(n, r) = 0$ for $r < 0$, $g(n, 0) = 1$ and $g(n, r) = n!/(\lfloor r/2 \rfloor! \lfloor (r + 1)/2 \rfloor! (n - r)!) + g(n, \lfloor (r + 1)/2 \rfloor - 2)$ for $1 \leqslant r$ (Igarashi, 1977). This is a marginal improvement of Dunham and Fridshal's lower bound (Dunham and Fridshal, 1959), and is conjectured to be optimal (Igarashi, 1977).

2. The star-algorithm

There are various algorithms for generating all the prime implicants of a given Boolean formula (for example Reuter, 1976). Our interests in this paper are not the algorithms themselves. We are interested in combinatorial problems for deciding the size of arrays to store intermediate data. The idea of the star-algorithm is primarily based on the work of Roth (1958), and its full description is given in Miller (1965). If a Boolean function is given as a disjunction of minterms, the process of the star-algorithm is essentially the same as the Quine-McCluskey algorithm. However, in many cases, Boolean functions are expressed as disjunctions of cubes which are not necessarily minterms. In these cases the star-algorithm will be more desirable than the Quine-McCluskey algorithm.

The following definition of the star-algorithm is from Miller (1965).

Definition 4

Let $C_r = a_1 \ldots a_n$ and $C_s = b_1 \ldots b_n$ be a pair of cubes over n variables. The star-product (denoted by $\circledast$) of C_r and C_s is defined as follows:

1. For a pair of co-ordinate values, the star-product is defined

*Now at Department of Computer Science, University of Gunma, Kiryu-city, Gunma-prefecture, Japan.

in the next table, where y is a new symbol not in $\{0, 1, 2\}$.

$\circledast$	0	1	2
0	0	y	0
1	y	1	1
2	0	1	2

2. If for more than one i, $a_i \circledast b_i = y$, then $C_r \circledast C_s = \&$. If at most one y appears in the products of co-ordinate values, then

$$C_r \circledast C_s = m(a_1 \circledast b_1) \ldots m(a_n \circledast b_n),$$

where m is a mapping from $\{0, 1, 2, y\}$ to $\{0, 1, 2\}$ defined as $m(0) = 0$, $m(1) = 1$ and $m(2) = m(y) = 2$.

Given an n-variable Boolean function f as a disjunction of cubes, the star-algorithm will produce the set of prime implicants of f. Suppose that S_1 is the given set of cubes for f. Then the star-algorithm is described as follows:

1. i is set to be 1.
2. S_i is set to be $S_i \circledast S_i$, where for sets A and B of cubes $A \circledast B = \{c \circledast d \mid c \in A, d \in B \text{ and } c \circledast d \neq \&\}$.
3. S_i is set to be $\{C \mid C \in S_i$ and there does not exist an element D in S_i such that $C \lessdot D\}$.
4. i is set to be $i + 1$.
5. If i is not greater than n, then go to (2).
6. Stop. (S_n is the set of prime implicants of the given Boolean function.)

3. A computer program for the star-algorithm

In this section we give a complete program in ALGOL 60 for the star-algorithm to show how combinatorial problems arise. For the purpose of this paper the program is a straightforward translation from the star-algorithm rather than a sophisticated program for it. A cube is expressed in the program as an integer vector over $\{0, 1, 2\}$, i.e. each component of a cube is stored in an entry of an integer array. Since only a 2-bit memory is sufficient to store 0, 1 or 2, this information representation is obviously inefficient from a memory space viewpoint. However, our purpose here is to describe clearly a straightforward program for the star-algorithm. We therefore do not use the well known packing technique of small integers.

The program consists of the initial part, three procedures and the main program. The three procedures are named CUBE, REDUCE and STAR respectively. Input data is given in the following form:

$m, n,$

$a_{11}a_{12} \ldots a_{1n}, a_{21}a_{22} \ldots a_{2n}, \ldots, a_{m1}a_{m2} \ldots a_{mn}$

where n is the number of variables, m is the number of cubes of the given Boolean formula, and the disjunction of $a_{11} \ldots a_{1n}$, $\ldots$, and $a_{m1} \ldots a_{mn}$ is the given Boolean formula.

Example 1

Let $f(x_1, x_2, x_3, x_4) = x_1x_2\bar{x}_3 \vee x_1\bar{x}_2x_4 \vee \bar{x}_1x_2x_4 \vee \bar{x}_1x_3x_4 \vee x_2x_3\bar{x}_4 \vee x_1x_3\bar{x}_4$. Then the input for f to the program is

6, 4,

1102, 1021, 0121, 0211, 2110, 1210

```
! ------------------------------------------------------------
  THE FOLLOWING PROGRAM IS FOR THE STAR-
  ALGORITHM TO PRODUCE ALL THE PRIME
  IMPLICANTS OF THE GIVEN BOOLEAN FORMULA.
  ------------------------------------------------------------
  INITIAL PART OF THE PROGRAM: SA, WHICH
  WILL BE INITIALLY READ FROM THE INPUT
  DATA, DETERMINES THE ACTUAL SIZE OF ARRAY
  A. NVAR IS THE NUMBER OF VARIABLES. IN THIS
  PART THE VALUE OF SIZE WILL BE COMPUTED.
  IT WILL BE NVAR!/(⌊2*NVAR + 1)/3⌋!⌊(NVAR +
  1)/3⌋!)*2↑⌊(2*NVAR + 1)/3⌋ WHICH IS AN UPPER
  BOUND OF THE MAXIMUM NUMBER OF PRIME
  IMPLICANTS GIVEN BY CHANDRA AND
  MARKOWSKY. THE SIZE OF ARRAYS A AND B
  WILL BE [1:2*SIZE, 1:NVAR] IN ORDER TO STORE
  INTERMEDIATE RESULTS.
  ------------------------------------------------------------ ;
BEGIN INTEGER I, J, K, SIZE, NVAR, SA, SB, IND;
 READ (SA, NVAR);
 I := K := 1; IND := (NVAR + 1)/3;
 FOR J := 0 UNTIL IND-1 DO
  BEGIN I := I*(IND-J);
        K := K*(NVAR-J)
  END;
 SIZE := K/I*(2↑((2*NVAR + 1)/3));
 BEGIN INTEGER ARRAY A, B [1:2*SIZE, 1:NVAR];
! ------------------------------------------------------------
  PROCEDURE COMB IS TO COMBINE TWO LISTS OF
  CUBES. SA AND SB ARE POINTERS TO INDICATE
  THE NUMBERS OF MEANINGFUL CUBES STORED
  IN ARRAY A AND ARRAY B RESPECTIVELY. IF
  COMB (A, B, SA, SB) IS CALLED, THE LIST IN B WILL
  BE MERGED INTO A. SOME CUBES IN A MAY BE
  DUPLICATED AFTER EXECUTION OF THIS PRO-
  CEDURE. ANY DUPLICATED ELEMENT WILL BE
  ELIMINATED BY CALLING PROCEDURE REDUCE.
  ------------------------------------------------------------ ;
  PROCEDURE COMB (A, B, SA, SB);
  VALUE B;
   BEGIN FOR I := 1 UNTIL SB DO
     FOR J := 1 UNTIL NVAR DO
      A[SA + I, J] := B[I, J];
    SA := SA + SB; SB := 0
   END;
! ------------------------------------------------------------
  PROCEDURE REDUCE IS TO DERIVE AN ANTI-
  CHAIN FROM THE GIVEN LIST SUCH THAT THE
  DISJUNCTION OF CUBES IN THE ANTICHAIN
  EXPRESSES THE SAME BOOLEAN FUNCTION
  EXPRESSED AS THE DISJUNCTION OF CUBES OF
  THE GIVEN LIST. THAT IS, REDUNDANT CUBES
  ARE ELIMINATED BY CALLING REDUCE.
  ------------------------------------------------------------ ;
  PROCEDURE REDUCE (A, SA, NVAR);
   BEGIN INTEGER IND2, IND2; INTEGER ARRAY
    RI[1:SA];
    FOR I := 1 UNTIL SA-1 DO
     FOR J := I + 1 UNTIL SA DO
      BEGIN IND1 := IND2 := 0;
       FOR K := 1 UNTIL NVAR DO
        IF A[I, K] = A[J, K] THEN
         BEGIN IND2 := IND1 + 1;
               IND2 := IND2 + 1
         END
         ELSE IF A[I, K] = 2 THEN IND1 := IND1 + 1
          ELSE IF A[J, K] = 2 THEN IND2 := IND2 + 1;
       IF IND1 = NVAR THEN RI[J] := 1
        ELSE IF IND2 = NVAR THEN RI[I] := 1
      END;
    I := 1;
    FOR K := 1 UNTIL SA DO
     IF RI[K] = 0 THEN BEGIN FOR J := 1 UNTIL
                                      NVAR DO
                                A[I, J]=A[K, J];
                       I := I + 1
```

```
        END;
    SA := I-1
    END;
!   ------------------------------------------------------------
    PROCEDURE STAR (A, B, SA, SB) IS TO PRODUCE
    THE STAR-PRODUCTS OF CUBES IN ARRAY A,
    AND ITS RESULT WILL BE STORED IN ARRAY B.
    WHENEVER SB BECOMES GREATER THAN SIZE,
    PROCEDURE REDUCE IS CALLED TO ELIMINATE
    REDUNDANT ELEMENTS.
    ------------------------------------------------------------;
    PROCEDURE STAR (A, B, SA, SB):
    VALUE A, SA:
    BEGIN INTEGER ARRAY Z[1:NVAR];
      SB := 0;
    FOR I := 1 UNTIL SA-1 DO
     BEGIN
      FOR J := I + 1 UNTIL SA DO
       BEGIN IND := 0;
        FOR K := 1 UNTIL NVAR DO
         IF A[I, K] = A[J, K] OR A [J, K] = 2
          THEN Z[K] := A[I, K]
          ELSE IF A[I, K] = 2 THEN Z[K] := A[J, K]
           ELSE
           BEGIN Z[K] := 2;
                 IND := IND + 1
           END;
        IF IND = 1 THEN
         BEGIN SB := SB + 1;
          FOR K := 1 UNTIL NVAR DO
           B[SB, K] := Z[K]
         END
       END;
      IF SB > SIZE THEN REDUCE (B, SB)
     END
    END;
!   ------------------------------------------------------------
    THE FOLLOWING IS THE MAIN PART OF THE
    PROGRAM. THE GIVEN CUBES ARE READ INTO
    ARRAY A. PROCEDURE STAR IS CALLED NVAR
    TIMES, AND THE LIST OF PRIME IMPLICANTS OF
    THE GIVEN BOOLEAN FORMULA IS PRINTED OUT.
    ------------------------------------------------------------;
    FOR I := 1 UNTIL SA DO
     FOR J := 1 UNTIL NVAR DO
      READ (A[I, J]);
    FOR I := 1 UNTIL NVAR DO
     BEGIN STAR (A, B, SA, SB);
           COMB (A, B, SA, SB);
           REDUCE (A, SA)
     END;
    FOR I := 1 UNTIL SA DO
      BEGIN NEWLINE;
            FOR K := 1 UNTIL NVAR DO
             PRINT (A[I, K])
      END
    END
END
```

Our PROCEDURE STAR does not produce the star-product of cubes a and b such that $a \circledast b \leqslant a$ or $a \circledast b \leqslant b$. This is the reason why COMB is called after STAR in the main program. This situation is formally described as follows:

$$A \circledast A = A \cup AS,$$

where $AS = \{a \circledast b \mid a \in A, b \in A, a \prec a \circledast b \text{ and } b \prec a \circledast b\}$. When STAR($A$, B, SA, SB) is executed, A does not change and AS is stored in B. Then A and AS are combined by calling COMB(A, B, SA, SB).

4. Consideration of SIZE

In INITIAL PART of the program, SIZE is set to be $\binom{n}{\lfloor (2n+1)/3 \rfloor} 2^{\lfloor (2n+1)/3 \rfloor}$. As described in Section 1, this value is the maximum size of antichains of S_n. Even if we frequently call REDUCE, SIZE positions are required to store intermediate results for some input data. For example, if all the n-variable minterms are supplied as input data, then all the $\lfloor (n+1)/3 \rfloor$-cubes are produced at the $\lfloor (n+1)/3 \rfloor$ th loop in MAIN PART of the program. Since the number of $\lfloor (n+1)/3 \rfloor$-cubes over n-variables is $\binom{n}{\lfloor (2n+1)/3 \rfloor} 2^{\lfloor (2n+1)/3 \rfloor}$, SIZE is the necessary array size to store intermediate results for the star-algorithm.

The size of array A and B in our program is designed to be 2*SIZE. Therefore at least a half of the array space is prepared for temporary storage of redundant cubes which will be eliminated later by calling REDUCE. In PROCEDURE STAR of the program, $AS = \{a \circledast b \mid a \in A, b \in B, a \prec a \circledast b \text{ and } b \prec a \circledast b\}$ is temporarily stored in B. Whenever the actual size of B (i.e. the value of SB) exceeds SIZE, REDUCE is called to eliminate redundant cubes in B. Obviously 2*SIZE is sufficiently large as the size of B for our program. It is interesting to ask whether it is possible to reduce the size of B without changing the remaining program. With this motivation some combinatorial problems will be formulated, and these problems will be discussed in this section.

Definition 5

If $C \subseteq S_n$ and $a \in C$, then $P(C, a) = \#\{b \mid b \in C \text{ and } a \circledast b \neq \&\}$ and $Q(C) = \#\{\{i, j\} \mid i \in C, j \in C \text{ and } i \circledast j \neq \&\}$. PMX $= \max\{P(C, a) \mid C \subseteq S_n, C \text{ is an antichain with relation } \leqslant, \text{ and } a \in C\}$. $QMX(n) = \max\{Q(C) \mid C \subseteq S_n \text{ and } C \text{ is an antichain with relation } \leqslant\}$.

$B_n(r)$ denotes the set of r-cubes over n variables. For any pair of a and b in $B_n(r)$ $P(B_n(r), a) = P(B_n(r), b)$. $P(B_n(r))$ denotes $P(B_n(r), a)$, where a is an arbitrary element of $B_n(r)$. From Definition 5 a necessary size of array B for our program is at most SIZE + $PMX(n)$. Therefore if $PMX(n)$ is considerably smaller than SIZE, and if its value can be easily evaluated, then SIZE + $PMX(n)$ will be a better size for array B. This modification of the size of B does not need any change of the remaining part of the program, and does not increase the computing time of it except additional computing time for evaluating $PMX(n)$. The size of array A can also be reduced with a minor change of the program, but the modified program will call REDUCE more frequently and will require more computing time. In general $QMX(n)$ is much larger than 2*SIZE. If core memory resource restriction is not a serious problem compared with computing time, the size of array B may be changed to $QMX(n)$ to reduce computing time. That is, if the size of array B is $QMX(n)$, then the statement IF SB>SIZE THEN REDUCE(B, SB) in PROCEDURE STAR can be removed and then REDUCE(B, SB) is added at the end of the scope of STAR. For this modified STAR, REDUCE is called only once during executing PROCEDURE STAR. However, changing the size of B to $QMX(n)$ is not practical unless n is small.

Lemma 2

$$P(B_n(r)) = \sum_{i=0}^{r} \left(\binom{r}{i} \binom{n-i}{r} \right) + (n-r) \sum_{i=0}^{r} \left(\binom{r}{i} \binom{n-i-1}{r} \right).$$

Proof

Without loss of generality we may choose $a = 1^{n-r}\, 2^r$ to

evaluate $P(B_n(r)) = P(B_n(r), a)$, where w^k means the concatenation of k w's. Then for $b \in B_n(r)$ $a \circledast b \neq \&$ if and only if b contains at most one 0 in the first $n - r$ positions. The number of elements b in $B_n(r)$ such that b does not contain any 0 in the first $n - r$ positions is $\sum_{i=0}^{r} \left(\binom{r}{i} \binom{n-i}{r} \right)$. The number of elements b in $B_n(r)$ such that b contains exactly one 0 in the first $n - r$ positions is $(n - r) \left(\sum_{i=0}^{r} \left(\binom{r}{i} \binom{n-i-1}{r} \right) \right)$.

Hence the lemma holds. □

Corollary 1

$P(B_n(r))$ can also be expressed as $\sum_{i=0}^{r} \left(\binom{n-r}{i} \binom{r}{i} 2^i \right) + (n - r) \left(\sum_{i=0}^{r} \left(\binom{n-r-1}{i} \binom{r}{i} 2^i \right) \right)$.

Proof

The number of elements b in $B_n(r)$ such that b does not contain any 0 in the first $n - r$ positions can be expressed as $\sum_{i=0}^{r} \left(\binom{n-r}{i} \binom{r}{i} 2^i \right)$. The number of elements b in $B_n(r)$ such that b contains exactly one 0 in the first $n - r$ positions is $(n - r) \left(\sum_{i=0}^{r} \binom{n-r-1}{i} \binom{r}{i} 2^i \right)$. Hence the corollary holds. □

Lemma 3

For any $0 \leqslant r \leqslant n$ $Q(B_n(r)) = (P(B_n(r)) + 1) \binom{n}{r} 2^{n-r-1}$.

Proof

Since $\#(B_n(r)) = \binom{n}{r} 2^{n-r}$ and $P(B_n(r)) = P(B_n(r), a)$ for any a in $B_n(r)$, $Q(B_n(r)) = (P(B_n(r))\#(B_n(r)) - \#(B_n(r)))/2 + \#(B_n(r))$.

$= (P(B_n(r)) + 1) \binom{n}{r} 2^{n-r-1}$. □

Hall's Theorem can be stated as a property on bipartite graphs as follows (for example, see Bondy and Murty, 1976). We assume that a graph G is bipartite. That is, G can be partitioned into two sets, X_1 and X_2, such that no two vertices in the same set are adjacent. Then there exists a matching of size $\#(X_1)$ if and only if for all subsets $N \subseteq X_1$ the number of vertices in X_2 jointed directly to the vertices of N is at least $\#(N)$.

The next theorem can be derived as a corollary of Hall's Theorem.

Theorem 1

Let P be a partially ordered set, and let R be a maximum sized antichain of P. Then for any antichain Q of P there exists a mapping f from Q to R satisfying the following two conditions:

1. For any element a in Q $f(a) \leqslant a$ or $a \leqslant f(a)$, and
2. for any pair of elements a and b in Q $(a \neq b)$ $f(a) \neq f(b)$, where $\leqslant$ is the relation of P.

Proof

We draw a bipartite graph G associated with R and Q as follows: Let the two antichains R and Q be combined. If an element v is in both R and Q, then two vertices named v are in the join of R and Q (one is in group R and the other is in group Q). Vertices a and b are jointed if and only if $a \in R$ (or $a \in Q$), $b \in Q$ (or $b \in R$) and there is a relation between a and b (i.e. $a \leqslant b$ or $b \leqslant a$). Let G be a graph constructed in this way. Then G is bipartite. That is, G can be partitioned into two groups R and Q, and no two vertices in the same group are adjacent.

Suppose that there does not exist a mapping satisfying the conditions in the theorem (i.e. we suppose that there is no matching of size $\#(Q)$ on G). Then from Hall's Theorem, for some $N \subseteq Q$ $\#(M) < \#(N)$, where M is the set of elements b in R such that b is jointed to an element of N. $(R - M) \cup N$ is an antichain and $\#(R) < \#((R - M) \cup N)$. Therefore R cannot be a maximum sized antichain. □

Corollary 2

Let A be an antichain of cubes with relation $\leqslant$ over n variables. If for any element a in A there exists an $\lfloor (n + 1)/3 \rfloor$-cube b such that $a \leqslant b$, then $P(A, c) \leqslant P(B_n(\lfloor (n + 1)/3 \rfloor))$, where c is an arbitrary element of A.

Proof

From Lemma 1 $B_n(\lfloor (n + 1)/3 \rfloor)$ is a maximum sized antichain of S_n. Therefore there is a mapping f from A to $B_n(\lfloor (n + 1)/3 \rfloor)$ satisfying the conditions described in Theorem 1. For any pair of a and b $(a \neq b)$ in A $f(a) \neq f(b)$, $a \leqslant f(a)$ and $b \leqslant f(b)$. Therefore, if $a \circledast b \neq \&$, then $f(a) \circledast f(b) \neq \&$. Hence for an arbitrary c in A $P(A, c) \leqslant P(B_n(\lfloor (n + 1)/3 \rfloor))$. □

The next theorem is straightforward from the definitions of $PMX(n)$ and Corollary 2.

Theorem 2

$\max\{P_n(r) \mid 0 \leqslant r \leqslant n\} = \max\{P_n(r) \mid \lfloor (n + 1)/3 \rfloor \leqslant r \leqslant n\} \leqslant PMX(n)$.

A function $g(n)$ is said to be $O(f(n))$ if and only if there exists a constant k such that $g(n) \leqslant kf(n)$ for all n but some finite set of non-negative integers. $g(n) \sim f(n)$ if and only if there exist constants k_1 and k_2 such that $g(n) \leqslant k_1 f(n)$ and $f(n) \leqslant k_2 g(n)$ for all n but some finite set of non-negative integers. For a fixed n, $P(B_n(r)) = \sum_{i=0}^{r} \left(\binom{r}{i} \binom{n-i}{r} \right) + (n - r) \sum_{i=0}^{r} \left(\binom{r}{i} \binom{n-i-1}{r} \right)$ takes its maximum value when $r = \lfloor (n - 1)/2 \rfloor$. The proof of this fact is rather complicated. It is straightforward from a simple combinatorial argument that $\sum_{i=0}^{r} \left(\binom{r}{i} \binom{n-i}{r} \right)$ takes its maximum value when $r = \lfloor n/2 \rfloor$, and that $(n - r) \sum_{i=0}^{r} \left(\binom{r}{i} \binom{n-i-1}{r} \right)$ takes its maximum value when $r = \lfloor (n - 1)/2 \rfloor$. Hence, $P(B_n(n))$ takes its maximum value when $r = \lfloor n/2 \rfloor$ or $r = \lfloor (n - 1)/2 \rfloor$. This weaker fact is enough to prove the next theorem.

Theorem 3

For all $0 < \varepsilon$, $k_1(n3^{3/4})^n \leqslant \max\{P(B_n(r)) \mid 0 \leqslant r \leqslant n\} \leqslant k_2(n^2 3^{3n/4}) \leqslant k_3(3^{3/4} + \varepsilon)^n$, where k_1, k_2 and k_3 are constants independent of n.

Proof

Since $O(P(B_n(\lfloor n/2 \rfloor)) \sim O(P(B_n(\lfloor n - 1)/2 \rfloor))$, from the argument described above this theorem

$$\max\{P(B_n(r)) \mid 0 \leqslant r \leqslant n\} \sim P(B_n(\lfloor n/2 \rfloor))$$

$$n \sum_{i=0}^{\lfloor n/4 \rfloor} ((n-i)!/(i!(\lfloor n/2 \rfloor - i)!(\lfloor n/2 \rfloor - i)!)) \quad (1)$$

$$= n[\sum_{i=0}^{\lfloor n/4 \rfloor} ((n-i)!/(i!(\lfloor n/2 \rfloor - i)!(\lfloor n/2 \rfloor - i)!)) + \sum_{i=\lfloor n/4 \rfloor + 1}^{\lfloor n/2 \rfloor} ((n-i)!/(i!(\lfloor n/2 \rfloor - i)!(\lfloor n/2 \rfloor - i)!))] \quad (2)$$

Since for $0 \leqslant i \leqslant \lfloor n/4 \rfloor$, $(\lfloor n/2 \rfloor - i - \lfloor (n-4i)/2 \rfloor) = i$ and $(n-i) - (n-4i) = 3i$,

$$\sum_{i=0}^{\lfloor n/4 \rfloor} ((n-i)!/(i!(\lfloor n/2 \rfloor - i)!\lfloor (n/2 \rfloor - i)!)) \sim \sum_{i=0}^{\lfloor n/4 \rfloor} ((\prod_{k=0}^{n-4i} ((n-i-k)/(\lfloor n/2 \rfloor - i - \lfloor k/2 \rfloor)))3^{3i}) \quad (3)$$

Fig. 1(*a*) may be helpful for the reader to understand relation (3) above. Since for $\lfloor n/4 \rfloor + 1 \leqslant i \leqslant \lfloor n/2 \rfloor$, $i - (2i - \lfloor n/2 \rfloor) = \lfloor n/2 \rfloor - i$ and $(n-i) - (2i - \lfloor n/2 \rfloor) = n + \lfloor n/2 \rfloor - 3i$,

$$\sum_{i=\lfloor n/4 \rfloor + 1}^{\lfloor n/2 \rfloor} ((n-i)!/(i!(\lfloor n/2 \rfloor - i)!(\lfloor n/2 \rfloor - i)!)) \sum_{i=\lfloor n/4 \rfloor + 1}^{\lfloor n/2 \rfloor} ((\prod_{k=0}^{2i - \lfloor n/2 \rfloor} ((n-i-k)/(i-k)))3^{\lfloor n/2 \rfloor - i}) \quad (4)$$

Fig. 1(*b*) may be helpful for the reader to understand relation (4) above. From (1)-(4) above, $n\,3^{3n/4}$ is $O(\max\{P(B_n(r)) \mid 0 \leqslant r \leqslant n\})$ and $\max\{P(B_n(r)) \mid 0 \leqslant r \leqslant n\})$ is $O(n^2\,3^{3n/4})$. Hence, the theorem holds. □

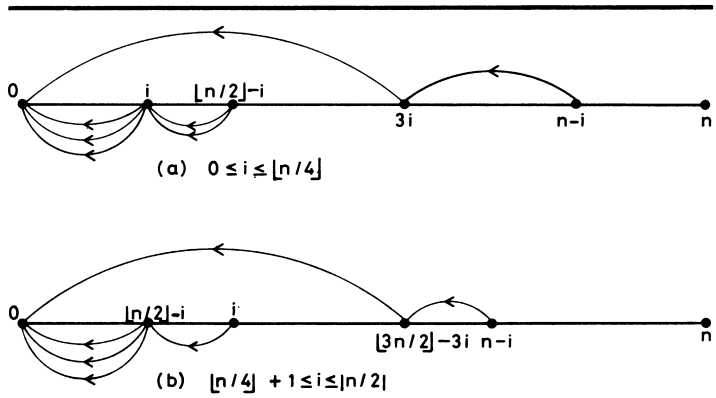

Fig. 1 Illustration of relations (3) and (4)

5. Conclusions and open problems

The estimation of the value of $PMX(n)$ seems to be difficult. A trivial upper bound of $PMX(n)$ is SIZE which is $O(3^n/\sqrt{n})$ (Chandra and Markowsky, 1976). For almost all n $P(B_n(\lfloor (n-1)/2 \rfloor))$ is much smaller than SIZE. For example, $P(B_{15}(7)) = 502{,}379$ whereas SIZE for 15 variables is 3,075,075. In general, $\max\{P(B_n(r)) \mid 0 \leqslant r \leqslant n\}$ is not equal to $PMX(n)$. For example, $P(B_6(2)) = 141$ whereas $P(B_6(3) \cup \{002222\} - \{000222, 001222\}, 002222) = 150$. However, for most cases SIZE + $P(B_n(\lfloor (n-1)/2 \rfloor))$ is large enough for the size of array B in our program. It is a preferable modification from the memory space viewpoint to change the size of array B to be SIZE + $P(B_n(\lfloor (n-1)/2 \rfloor))$, and to insert an additional REDUCE in order to call REDUCE whenever the actual size of array B (i.e. SB) exceeds this modified size. This modification does not cause a big increase in computing time since the added REDUCE is seldom called.

In general, flexible arrays or extensible arrays are less efficient from the computing time viewpoint, and for many combinatorial problems both computing time and memory space are serious factors restricting input size. Therefore, even if we write a program in a more powerful language such as ALGOL 68, it may be preferable to use fixed sized arrays whenever the array size can be properly estimated. There are, of course, a number of ways to reduce computing time for our program. For example, if we employ suitable data structure of intermediate results stored in arrays, it is possible to eliminate a considerable amount of unnecessary computation in PROCEDURE REDUCE and PROCEDURE STAR. Since these techniques are well known, we do not discuss them in this paper. We invite the reader to consider the following open problems:

1. Find a better lower bound or upper bound of $PMX(n)$.
2. Find a nontrivial lower bound or upper bound of $QMX(n)$.

Acknowledgements

The author would like to thank Leslie G. Valiant for suggesting the proof of Theorem 1 (Valiant, 1978), William F. McColl for his comments, and Jean V. Scott for pointing out errors in an earlier draft of this paper. The program in Section 3 was tested on the DEC System 10 at Leeds University.

References

BONDY, J. A. and MURTY, U. S. R. (1976). *Graph Theory with Applications*, Macmillan Press.

CHANDRA, A. K. and MARKOWSKY, G. (1976). *On the number of prime implicants*, IBM Research Report, RC 6108.

DUNHAM, B. and FRIDSHAL, R. (1959). The problem of simplifying logical expressions, *J. Symbolic Logic*, Vol. 24, p. 17.

HARRISON, M. A. (1965). *Introduction to Switching and Automata Theory*, McGraw-Hill.

IGARASHI, Y. (1977). Analysis of Dunham and Fridshal's formulas consisting of large numbers of prime implicants, Centre for Computer Studies, University of Leeds, Technical Report No. 98.

KLEITMAN, D. J., EDERBERG, M. and LUBELL, D. (1971). Maximal sized antichains in partial orders, *Discrete Mathematics*, Vol. 1, p. 47.

MILLER, R. E. (1965). *Switching Theory*, Vol. 1, John Wiley.

REUTER, B. (1976). On the generation of prime implicants, Computer Science Dept., Cornell University, Technical Report TR 76-266.

ROTH, J. P. (1958). Algebraic topological methods for synthesis of switching systems, I, *Transactions of AMS*, Vol. 88, p. 301.

VALIANT, L. G. (1978). Personal communication.