Algorithms for Formula Manipulation Based upon Hashing Technique and their Applications to Pseudo-Boolean Programming

Masaharu IMAI*, Yuuji YOSHIDA** and Teruo FUKUMURA*

ARSTRACT

We propose algorithms for formula manipulation of Boolean functions and pseudo-Boolean functions. They are based on hashing technique and take $O(m)$ computation time, where m is the number of terms involved in the function being processed. The performance of the algorithms is verified through some experiments. They are found to be successfully applicable to the pseudo-Boolean programming algorithm.

1. INTRODUCTION

We propose algorithms to simplify(pseudo-)Boolean function, which are based on hashing technique[1]. Our algorithms are applicable to (pseudo-)Boolean functions with many terms for the following reasons: (i) Our algorithms can simplify these functions in $O(m)$ computation time, where m denotes the number of terms in the function; (ii) The test of whether a term consists of only one variable or not can be made in constant time.

2. BOOLEAN FUNCTION, PSEUDO-BOOLEAN FUNCTION AND THEIR REPRESENTATION

2.1 BOOLEAN FUNCTION AND PSEUDO-BOOLEAN FUNCTION

First, let $x_j \in \{0, 1\}$ for $j = 1, 2, \ldots, n$, and let X denote the vector $(x_1, x_2, \ldots, x_n)$. In the following discussion, Boolean function and pseudo-Boolean function will have the following form. Let a Boolean function F(X) be written as:

$$F(X) = \cup_{i=1}^{s} F_i(X);\ F_i(X) = \Pi_{j=1}^{n} x_j^{e_{ij}},\ (i = 1, 2, \ldots, s) \qquad (1)$$

where $\cup$ and Π denote logical sum and logical product, respectively, and $x_j^{e_{ij}}$ has the value of $\bar{x}_j$ if $e_{ij} = o$, x_j if $e_{ij} = 1$, 1 if $e_{ij} = t$, or 0 if $e_{ij} = f$.

Let a pseudo-Boolean function y(X) be written as:

$$y(X) = \Sigma_{i=1}^{m} a_i Y_i(X);\ Y_i(X) = \Pi_{j=1}^{n} x_j^{d_{ij}},\ (i= 1, 2, \ldots, m) \qquad (2)$$

where a_i's are integers, and $x_j^{d_{ij}}$ has the value of 1 if $d_{ij} = 0$, or x_j if $d_{ij} = 1$.

The pseudo-Boolean programming algorithm minimizes the pseudo-Boolean function

---

This paper first appeared in Japanese in Joho－Shori (Journal of the Information Processing Society of Japan), Vol. 18, No. 7 (1977), pp. 639～647.

* Faculty of Engineering, Nagoya University
** Computation Center, Nagoya University

given by eq(2) under the Boolean constraint F(X) = 0, where F(X) is given by eq(1).

DEFINITION 1. : A Boolean function F(X) is said to be simplified if all $F_i(X)$'s consist of at least two variables and are distinct to each other.

DEFINITION 2. : A pseudo-Boolean function y(X) is said to be simplified if all $Y_i(X)$'s are distinct to each other.

2.2 INTERNAL REPRESENTATION OF FORMULA

A Boolean function F(X) is effectively represented in computer storage as follows. Every term $F_i(X)$ in F(X) is associated with a pair of integers (σ_i, τ_i) determined by eq(3).

$$\sigma_i = \Sigma_{j\in I_i} 2^{j-1}, \quad \tau_i = \Sigma_{j\in J_i} 2^{j-1} \tag{3}$$

where I_i and J_i are defined using N = {1, 2, ..., n} as follows :

$$I_i = \{j \mid j \in N, (e_{ij} = 1) \cup (e_{ij} = 0)\},$$
$$J_i = \{j \mid j \in N, (e_{ij} = 1) \cup (e_{ij} = f)\} \tag{4}$$

Thus, F(X) is represented as a set of ordered pairs :

$$\{(\sigma_i, \tau_i) \quad 1 \le i \le s\} \tag{5}$$

Pseudo-Boolean function y(X) is represented similarly as above. Every term $Y_i(X)$ in y(X) is associated with a pair of integers (η_i, a_i), where η_i is determined by eq(6).

$$\eta_i = \Sigma_{j\in K_i} 2^{j-1} \tag{6}$$

where K_i is defined as :

$$K_i = \{j \mid j \in N, d_{ij} = 1\} \tag{7}$$

And then, y(X) is represented as a set of ordered pairs :

$$\{(\eta_i, a_i) \mid 1 \le i \le m\} \tag{8}$$

3. ALGORITHMS

3.1 ALGORITHM FOR PSEUDO-BOOLEAN FUNCTION

The internal representation of y(X) is implemented by using a hash table. η_i of eq(6) is used as the key for hash address. The hash function is defined as :

$$h(\eta_i) = \mathrm{mod}(\eta_i, p) + 1 \tag{9}$$

In what follows, p denotes a prime number. The terms having same hash address are linked in a linear list as shown in Fig. 1. Algorithm H used to simplify pseudo-Boolean function is described below.

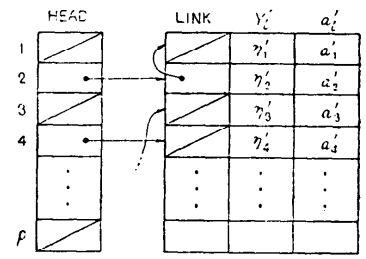

Fig.1 Data structure used in Algorithm H

ALGORITHM H

COMMENT : Two arrays HEAD [1:p]and LINK [1:m] are used in the algorithm.

INPUT : y(X) of eq(2); OUTPUT : $y'(X) = \sum_{i=1}^{r} a_i' Y_i'(X)$.

STEP 1: [Initialize] Set i = 1, r = 0 and HEAD(j) = 0 for j = 1, 2, ..., p.

STEP 2: [Compute hash address] Set $j = h(\eta_i)$ and $k = HEAD(j)$.

STEP 3: [Check if similar term exists] If k = 0, then go to STEP 4.

Else, if $\eta_i = \eta_k'$ then set $a_k' = a_k' + a_i$ and go to STEP 5.

Otherwise, set k = LINK(k) and go to STEP 3.

STEP 4: [Store term] Set r = r + 1, $\eta_r' = \eta_i$, $a_r' = a_i$, LINK(r) = HEAD(j) and HEAD(j) = r.

STEP 5: [Check for termination] Set i = i + 1. If i > m then HALT.

Else, go to STEP 2.

3.2 ALGORITHM FOR BOOLEAN FUNCTION

The simplification of Boolean function is successfully performed by an algorithm similar to Algorithm H above. In this case, σ_i of eq(3) is used as the key for hash address. The following property is important to determine if a term consists of only one variable or not -- because the internal representation σ_i of $F_i(X)$ is equal to 2^{j-1} for some j, where $1 \leq j \leq n$.

PROPERTY 1 : Let p be a prime number having 2 as its primitive root. Then the following relation holds : $2^i \not\equiv 2^j \bmod p$ for $1 \leq i < j \leq p$ (10)

It is obvious from this property that two terms which consist of distinct single variables will have distinct hash addresses.

4. TIME COMPLEXITY OF ALGORITHM H

In this section, we evaluate the computational time complexity of Algorithm H only for pseudo-Boolean functions, because the time complexity for Boolean functions is of the same order. In the discussion below, the following assumption is made.

ASSUMPTION 1 : The expected length of all linked lists are equal at any stage of Algorithm H.

If Assumption 1 holds, Algorithm H would take maximum computation time to simplify y(X) which has already been simplified. The maximum computation time of Algorithm H is given by : $T = a(m-1)(m-2)/(2p) + bm + cp + d$ (11)

Where a, b, c and d are constants independent of m. The partial derivative of T with respect to p is given by the following equation :

$$\partial T / \partial p = -a(m-1)(m-2) / (2p) + c \qquad (12)$$

Let p^* be the value of p which minimizes T. p^* is obtained by solving $\partial T / \partial p = 0$.

$$p^* = \sqrt{a(m-1)(m-2)/(2c)} \qquad (13)$$

Assuming $m >> 1$, and letting $\lambda = \sqrt{a / (2c)}$, we have : $p^* = \lambda m$ (14)

The minimum value T^* of T is obtained by substituting p^* to p in eq(15). That is,

$$T^* = \sqrt{2ac} + bm + d \qquad (15)$$

Then, we have $T^* = O(m)$.

5. EXPERIMENTS

5.1 EXPERIMENT I (Computational time complexity)

The computation time needed to simplify y(X) by Algorithm H and by a conventional algorithm (denoted by "Algorithm A") are measured. y(X)'s are generated from a series of uniform random numbers such that y(X) is already simplified.

The average computation time of Algorithm H, for 20 different y(X)'s for each value of m, is shown in Fig. 2. From this figure, we can determine the coefficient in eq(14). A comparison of the computation time of Algorithm H*, which is Algorithm H using p^*, with that of Algorithm A is shown in Fig. 3. From this figure the following observation can be made : the computational time complexity of Algorithm H* is $O(m)$; whereas that of Algorithm A, is $O(m^2)$.

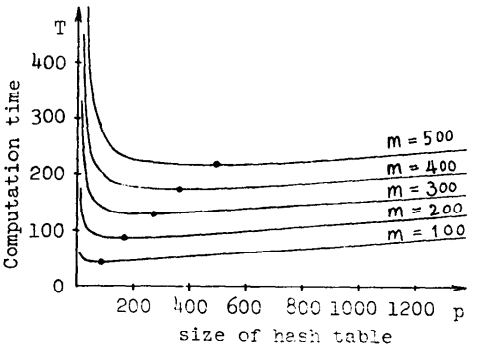

Fig.2 Computation time for y(X) by Alg.H
Symbol .'s denote the minimum of T.

5.2 EXPERIMENT II (Application of Algorithm H*)

We implemented two pseudo-Boolean programming systems, Sys. H and Sys. A, which utilize Algorithm H* and Algorithm A, respectively, to simplify y(X) and F(X). Then, the computation time needed to solve pseudo-Boolean programming problems were measured, where the problems used were generated from uniform random numbers.

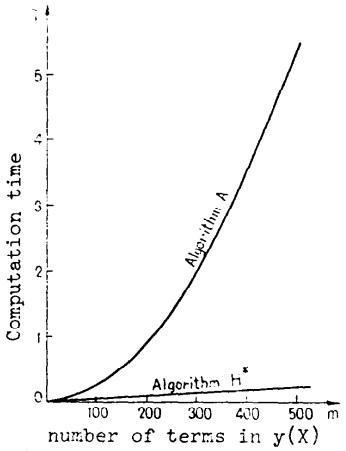

Fig.3 Comparison of computation time for y(X) by Algorithm H* and Algorithm A

We define I, the index of improvement of Sys. H over Sys. A, as follows:

$$I = (T_A - T_H) / T_A \times 100 \ (\%) \qquad (16)$$

where T_A and T_H denote the computation time of Sys. A and Sys. H, respectively. The average improvement index for nine cases of 20 problems each is shown in Fig. 4. It is obvious from this figure that Algorithm H* does work successfully in the pseudo-Boolean programming system. The relatively low improvement index of total system observed in some cases is due to low fraction of formula manipulation time against total computation time.

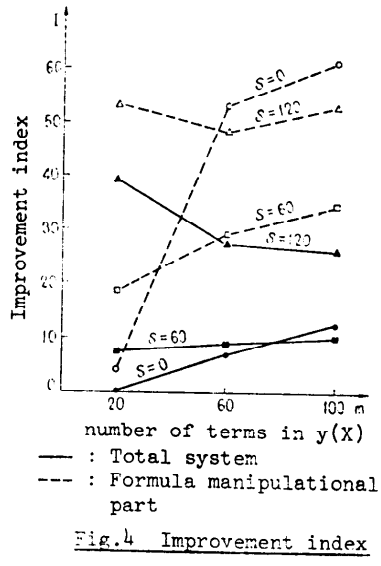

Fig.4 Improvement index

6. REMARKS

The basic idea used in Algorithm H* is also applicable to the simplification of formula in polynomial form consisting of many terms.

COMPUTER USED : FACOM 230-38 FORTRAN IV S

ACKNOWLEDGEMENT : The authors would like to thank Prof. Namio HONDA of Nagoya University for valuable advice, and also the members of our laboratory for providing helpful discussion.

REFERENCES :

(1) D.E.Knuth : The Art of Computer Programming, Vol. 3, Chap. 6, Addison-Wesley (1973)

(2) Y.Yoshida et.al. : Algorithms of Pseudo-Boolean Programming Based on the Branch and Bound Method, Jour. Inst. Electronics Comm. Engrs, of Japan, Vol. 50, No. 10, pp 1995-2002 (1967)