Available online at www.sciencedirect.com
ScienceDirect
Discrete Applied Mathematics 156 (2008) 1633–1636
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

An improvement on the complexity of factoring read-once Boolean functions

Martin Charles Golumbic[a,1], Aviad Mintz[b], Udi Rotics[c]

[a] *Department of Computer Science, Caesarea Rothschild Institute, University of Haifa, Mt. Carmel, Haifa 31905, Israel*
[b] *Caesarea Rothschild Institute, University of Haifa, Mt. Carmel, Haifa 31905, Israel*
[c] *School of Computer Science and Mathematics, Netanya Academic College, P.O. Box 120, 42100 Netanya, Israel*

Received 7 April 2006; received in revised form 15 February 2008; accepted 20 February 2008

Abstract

Read-once functions have gained recent, renewed interest in the fields of theory and algorithms of Boolean functions, computational learning theory and logic design and verification. In an earlier paper [M.C. Golumbic, A. Mintz, U. Rotics, Factoring and recognition of read-once functions using cographs and normality, and the readability of functions associated with partial k-trees, Discrete Appl. Math. 154 (2006) 1465–1677], we presented the first polynomial-time algorithm for recognizing and factoring read-once functions, based on a classical characterization theorem of Gurvich which states that a positive Boolean function is read-once if and only if it is normal and its co-occurrence graph is P_4-free.

In this note, we improve the complexity bound by showing that the method can be modified slightly, with two crucial observations, to obtain an $O(n|f|)$ implementation, where $|f|$ denotes the length of the DNF expression of a positive Boolean function f, and n is the number of variables in f. The previously stated bound was $O(n^2k)$, where k is the number of prime implicants of the function. In both cases, f is assumed to be given as a DNF formula consisting entirely of the prime implicants of the function.

Keywords: Read-once functions; Logic; Boolean functions; Cographs

1. Introduction

A function f is called *read-once* if it can be represented as a Boolean expression using the operations conjunction, disjunction and negation, in which every variable appears exactly once. Such an expression is called a *read-once expression* for f. For example, the function

$$f_0 = ay \vee cxy \vee bw \vee bz$$

is a read-once function since it can be factored into the expression

$$f_0 = y(a \vee cx) \vee b(w \vee z),$$

[1] On sabbatical (October 2005–March 2006) at the Laboratory for Artificial Intelligence, Ecole Polytechnique Federale de Lausanne (EPFL), Lausanne, Switzerland.

E-mail addresses: golumbic@cs.haifa.ac.il (M.C. Golumbic), amintz@vlsi.co.il (A. Mintz), rotics@mars.netanya.ac.il (U. Rotics).

0166-218X/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2008.02.011

which is a read-once expression. Neither of the functions $f_1 = ab \vee bc \vee cd$ nor $f_2 = ab \vee bc \vee ac$ is a read-once function, and they illustrate the two types of forbidden configurations which characterize read-once functions in Theorem 1.

Read-once functions have interesting special properties and, according to experts, account for a large percentage of functions which arise in real circuit applications [16]; they correspond to functions which have tree networks, and can simplify part of the verification process. Read-once functions have also gained interest in the field of computational learning theory [1], where they constitute a very natural class of functions that can be learned exactly with only a polynomial number of queries. In addition, read-once functions appear at the lowest level of recursion in the heuristic algorithm described in [7,15] for factoring general Boolean functions into shorter, more compact logically equivalent expressions (an NP-hard basic operation in the early stages in designing logic circuits).

Since every variable appears once, either in its positive or negative form, in a read-once expression, it may be assumed that every variable of a read-once function will be positive, simply by renaming any negative variable $\overline{x}_i$ as a new positive variable x'_i.

A classical theorem of Gurvich [10,11] characterizes read-once functions.

Theorem 1. *A positive Boolean function f is read-once if and only if its co-occurrence graph Γ_f contains no induced chordless path P_4 and f is normal.*

The *co-occurrence graph* of *f*, denoted by $\Gamma_f = (V, E)$, has vertex set $V = \{x_1, x_2, \ldots, x_n\}$ (the same as the variables), and there is an edge (x_i, x_j) in E if x_i and x_j occur together (at least once) in some prime implicant of *f*. Graphs which are P_4-free are also known as *cographs* (complement reducible graphs) and have a unique canonical representation as a rooted decomposition tree called the *cotree* [3,5]. A Boolean function *f* is called *normal* if every clique in its co-occurrence graph is contained in a prime implicant of *f*. A new proof of Theorem 1, together with other characterizations of read-once Boolean functions, can be found in [6].

Theorem 1 provides the justification for the first polynomial-time recognition algorithm of read-once functions [8,9], where *f* is given as a DNF formula consisting entirely of the prime implicants of the function. The complexity of that algorithm is $O(n^2k)$, where k is the number of prime implicants of the function.

In this paper, we improve the complexity bound by showing that the method can be modified slightly, with two crucial observations in Step 3, in order to obtain an $O(n|f|)$ implementation, where $|f|$ denotes the "length" of the DNF expression of the function *f*, namely, the number of occurrences of variables and operations in the DNF expression. So, the modified algorithm will do significantly better than the original algorithm when a significant fraction of the prime implicants are short, that is, of size less than $\Theta(n)$.

Algorithm (*Golumbic, Mintz and Rotics, GMR [8,9]*). *Read-once Recognition*

Step 1: Build the co-occurrence graph Γ_f.

Step 2: Test whether Γ_f is P_4-free, and construct the cotree T for Γ_f. Otherwise, exit with "failure".

Step 3: Test whether *f* is a normal function, and if so, output T as the read-once expression. Otherwise, exit with "failure".

Let us examine the computational complexity of each step. We assume that *f* is a positive Boolean function, and let $\mathcal{P} = \{P_i\}$ be the list of its prime implicants, that is, $f = P_1 \vee P_2 \vee \cdots \vee P_k$ is its DNF expression (or sum of products SOP). We follow the notation and constructions of Section 4 of [9].

Step 1: The first step of the GMR algorithm is building the graph Γ_f. By traversing the DNF expression once, the edge set of Γ_f can be found in $O(\Sigma|P_i|^2)$ time, summing over all prime implicants $P_i \in \mathcal{P}$. It is easy to see that this is at most $O(n|f|)$, where $|f|$ denotes the length of the DNF expression.

Step 2: The complexity of testing whether the graph Γ_f is P_4-free, and providing a read-once expression (its cotree T), is $O(n + e)$ as first shown in [4], where e is the number of edges in the graph. Other known linear time algorithms such as [2,13] can also be used. This is at worst $O(n^2)$ and is bounded by $O(n|f|)$.

Step 3: The CheckNormality algorithm in [9] assumes that Γ_f has successfully been tested to be P_4-free, and that its cotree T has been constructed (in Step 2). For a node a of T, we denote by T_a the subtree of T rooted at a, and note that T_a is also the cotree representing the subgraph of Γ_f induced by the set of labels of the leaves of T_a. The algorithm constructs the set $C(T)$ of maximal cliques of Γ_f, recursively, by traversing the cotree T from bottom to top. More precisely, it constructs the set of maximal subcliques $C(T_a)$ for each internal node a, combining them as it moves up the tree, using two operations, *set union* $\cup$ and *set join* $\otimes$, and the following lemma in [9].