J. Cryptology (1994) 7: 153–170

Journal of
CRYPTOLOGY

Functional Inversion and Communication Complexity*

Shang-Hua Teng
Xerox Corporation, Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304, U.S.A.

Communicated by Joan Feigenbaum

Received 19 December 1991 and revised 15 September 1993

Abstract. We study the relationship between the multiparty communication complexity of functions over certain communication topologies and the complexity of inverting those functions. We show that if a function of n variables has a *ring-protocol* or a *tree-protocol* of communication complexity bounded by φ, then there is a circuit of size $2^{O(\varphi)}n$ that computes an inverse of the function. Consequently, we prove that although inverting NC^0 Boolean circuits is NP-hard, planar NC^1 Boolean circuits can be inverted in NC, and hence in polynomial time. From the ring-protocol theorem, we derive an $\Omega(n \log n)$ lower bound on the VLSI area required to lay out any one-way function. Our results on inverting boolean circuits can be extended to algebraic circuits over finite rings. We prove that on certain topologies no one-way function can be computed with low communication complexity.

Key words. Communication complexity, Cryptanalysis, Functional inversions, Multiparty problems, One-way functions.

1. Introduction

One of the most fundamental questions in cryptanalysis is to characterize the class of permutations (or functions) whose inverse can be computed in polynomial time or by a polynomial-size circuit [1], [16], [18], [25]. Much research in theoretical cryptography has centered around finding the weakest possible cryptographic assumptions that enable the implementation of major cryptographic primitives [8], [7], [17]; however, little progress has been made on the characterization of permutations with small inversion circuits [18].

In this paper we study the relationship between the multiparty communication complexity of functions over certain communication topologies and the complexity

* The preliminary version of this paper appeared in *CRYPTO 91*. This work was supported in part by National Science Foundation Grant DCR-8713489. Part of this work was done while the author was at the School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, U.S.A. Current address: Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139, U.S.A.

of inverting those functions. We illustrate some nontrivial classes of functions that can be inverted efficiently.

Multiparty communication complexity is a generalization of two-party communication complexity [24]. Suppose f is a boolean function with input variables $X = \{x_0, \ldots, x_{n-1}\}$ and output variables $Y = \{y_0, \ldots, y_{m-1}\}$. Let $p_0, \ldots, p_{N-1}$ be N processors connected by a given topology G. Each processor p_i receives a subset X_i of X such that $(X_0, \ldots, X_{N-1})$ is a partition of X. Multiparty communication complexity measures the maximum amount of information that one processor will have exchange in order for the processors to derive "collectively" the value of output variables for an assignment of the input variables. By "collectively deriving the value of output variables" we mean that each processor p_i need only work out a share, Y_i, of the output variables—there is no need for p_i to send the values of Y_i to any other processor, provided that the value of each output variable is known to some processor. If the communication pattern (topology) is "regular" enough and the communication complexity is "low" enough, then, intuitively, Y_i depends weakly on the set of the input variables of other processors; thus the values of X_i can "almost surely" be inferred from the values of Y_i. This intuition leads to a divide-and-conquer strategy for functional inversion: Each processor p_i independently infers X_i from Y_i and meanwhile checks its communication neighbors for consistency. We show that the consistency-checking step above depends not only on the communication complexity but also (crucially) on the underlying communication topology.

We first address the problem in a ring-shaped topology (see Fig. 1(a)) where each processor can communicate only with its two neighbors. We show that if a function can be computed on a ring with communication complexity bounded by φ, then there is a circuit of size $2^{O(\varphi)}n$ that computes an inverse of the function.

We then show that if a function is computable by a polynomial-size planar circuit (see Section 5 for the definition) of depth d, then it can be computed on a right topology with communication complexity at most $2d$. Therefore, every planar NC^1 Boolean circuit has an inversion circuit of polynomial size and $O(\log n)$ depth. Our proof is constructive: Such an NC^1 inversion circuit of an NC^1 planar circuit can be found in NC^1, and hence in polynomial time. Therefore, there is no one-way fnction computable by an NC^1 planar circit. This contrasts interestingly with the facts that:

(1) If NP-functions (e.g., SATISFIABILITY) that have no polynomial-size circuit (i.e., assume $NP \not\subset P/poly$) exist, then one-way functions computable by NC^0 circuits exist (see [1] and Section 7).

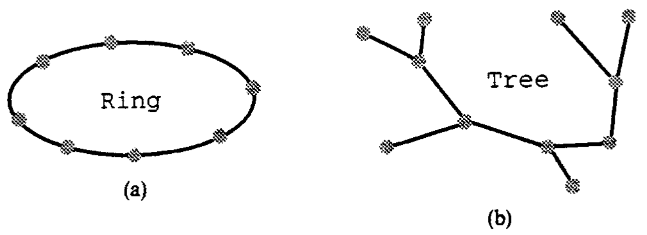

Fig. 1. (a) Ring shaped and (b) tree shaped topologies.

(2) If $P \neq$ uniform NC, then NC^0-permutations that cannot be inverted in NC exist [5].

We also derive an $\Omega(n \log n)$ lower bound on the VLSI area required to lay out any one-way function by using the fact that if a function has a circuit with VLSI layout area A in the Thompson model [10], [21], [22] where all inputs and outputs are on the boundary of the Thompson grid, then it can be computed by a ring protocol with communication complexity $O(A/n)$.

The result on the ring can be extended to one-dimensional meshes and trees with bounded degree (see Fig. 1(b)): if a function has a *tree-protocol* of communication complexity bounded by φ, then it has an inversion circuit of size $2^{O(\varphi)}n$. Our results on inverting boolean circuits can be extended to algebraic circuits over finite rings.

In contrast, we show that if NP is not contained in $P/poly$, then functions of constant communication complexity exist on a fully connected topology that have no polynomial-size inversion circuit.

2. Definitions and Notations

Let $\mathscr{B} = \{0, 1\}$, the field $GF(2)$. Let f be a function from $\mathscr{B}^n$ to $\mathscr{B}^m$. If $m = n$ and f is a bijection, then f is called a *permutation*. A function g is an *inverse* of f if, for all $y \in \mathscr{B}^m$, $g(y)$ is defined and $f(g(y)) = y$ whenever an x exists such that $f(x) = y$. In this paper let $\mathscr{B}_{n,m}$ denote the set of all functions from $\mathscr{B}^n$ to $\mathscr{B}^m$.

A *Boolean circuit* [23] is a directed acyclic graph whose nodes have indegree either 0 or 2. A node of indegree 0 is labeled either by a variable, called an *input variable*, or by a boolean constant. A node of indegree 2 is labeled by a binary boolean operator. A node of outdegree 0 has an extra label. It can be either a boolean variable, called an *output variable*, or a special label $\perp$ (called "don't care" nodes). We introduce $\perp$ to simplify the projection of a circuit to a subset of its output variables. A node labeled with an input variable is called an *input node* and a node labeled with an output variable is called an *output node*. We assume that each boolean circuit is *reduced* in the sense that no two input nodes share the same label, and from each input node there is a directed path to an output node. Each circuit C with n input nodes and m output nodes defines a function, denoted by f_C, from $\mathscr{B}^n$ to $\mathscr{B}^m$.

A circuit C *computes* a function f if $f_C = f$. A circuit C' is an *inversion circuit* for f if C' computes an inverse of f.

The nonuniform and uniform versions of the *inversion problem* are defined as follows.

Definition 2.1 (Inversion Problem).

- (*Nonuniform*) Does a function f have an inversion circuit of polynomial size?
- (*Uniform*) Is there a polynomial-time Turing machine that accepts a circuit as input and outputs a circuit that computes an inverse of the input circuit?

In this paper we use the notion of *one-way* functions as defined by Boppana and Lagarias [1]:

Definition 2.2 (One-Way Function) [1]. A Boolean function $f \in \mathcal{B}_{n,m}$ is *one-way* if it is computable by a polynomial-size circuit, but it does not have any inverse which is computable by a polynomial circuit.

Definition 2.3 (NC^k-Functions). A function $f \in \mathcal{B}^{n,m}$ is an NC^k-function if f is computable by an $O((\log n)^k)$-depth circuit of polynomial size.

Throughout this paper, for each function $f \in \mathcal{B}_{n,m}$, we write $f\colon 2^X \rightarrow 2^Y$, where $X = \{x_0, x_1, \ldots, x_{n-1}\}$ denotes the set of n input variables and $Y = \{y_0, y_1, \ldots, y_{m-1}\}$ denotes the set of m output variables.

3. Communication Complexity

3.1. *Two-Party Communication Complexity*

Suppose there are only two processors p_1 and p_2. For each partition (X_1, X_2) of X, p_1 receives the values of variables in X_1 and p_2 receives the values of variables in X_2. The *two-party communication complexity* of f with respect to the partition (X_1, X_2), denoted by $\mathcal{C}_f(X_1, X_2)$, is the number of bits the processors, using an optimal protocol, have to exchange, in the worst case, in order to jointly compute the values of all output variables of f [24], [19]. An optimal protocol for f with respect to the partition (X_1, X_2) also induces a natural partition of Y into (Y_1, Y_2) such that, in the protocol, p_i is responsible only for the values of variables in Y_i ($i \in \{1, 2\}$).

3.2. *Multiparty Communication Complexity*

A *communication topology* is an undirected graph $G = (V, E)$ where $V = \{p_0, \ldots, p_{N-1}\}$ represents the set of processors and each edge $(p_i, p_j) \in E$ represents a communication link between p_i and p_j.

The computation of a function f on a communication topology G is guided by a *distributed protocol* which is a set of rules specifying the order and the content of messages to be sent from one processor to another. Such a protocol specifies a partition $(X_0, \ldots, X_{N-1})$ of X and a partition $(Y_0, \ldots, Y_{N-1})$ of Y such that p_i receives the values of X_i, and is responsible for computing Y_i. We focus on a special class of distributed protocols that are called *oblivious protocols*. In an oblivious protocol $\mathcal{P}$, there are two sets of Boolean variables, called *communication variables*, $K_{i,j} = \{c_0^{i,j}, \ldots, c_{\varphi_{i,j}-1}^{i,j}\}$ and $K_{j,i} = \{c_0^{j,i}, \ldots, c_{\varphi_{j,i}-1}^{j,i}\}$ for each edge $(i, j) \in E$. The variable $c_k^{i,j}$ ($0 \leq k < \varphi(i, j)$) contains the kth bit from p_i to p_j.

Let $S_i^{\mathcal{P}} = \bigcup_{j:(i,j)\in E} K_{i,j}$ and let $R_i^{\mathcal{P}} = \bigcup_{j:(i,j)\in E} K_{j,i}$ be the set of all communication variables whose value p_i sends to and receives from, respectively, its neighbors in protocol $\mathcal{P}$. For simplicity, we omit $\mathcal{P}$ from upper index and use the simplified notation throughout the paper. When projected to a processor p_i, the protocol specifies a boolean function f_i from $X_i \cup R_i$ to $Y_i \cup S_i$. Throughout the paper we assume that X_i, Y_i, f_i, R_i, S_i are associated with the most recently discussed distributed protocol $\mathcal{P}$. The protocol is required to be *fluent* in the sense that there is no cyclic dependency between any pair of communication variables. The protocol ends when no message is due to be sent. An important problem in the field of distributed computing is to design distributed protocols that minimize the maximum number of bits a processor has to receive, to send, or both. In this paper all distributed protocols are assumed to be fluent.

A distributed protocol $\mathscr{P}$ *correctly computes* a function f if, for every assignment x to X, p_i computes an assignment y_i for Y_i such that $y = (y_0, \ldots, y_{N-1}) = f(x)$.

For each function, there is a trivial protocol with null communication cost, i.e., the one which assigns all inputs and all outputs to a single processor. In order to avoid this triviality, we only concern ourselves with the set of *balanced protocols*. We assume $N < n$ in the following definition.

Definition 3.1 (Balanced-Protocols). A partition $(X_0, \ldots, X_{N-1})$ of X is *H-balanced* if $|X_i| \leq H$ for all $0 \leq i < N$. A protocol $\mathscr{P}$ for a function f is *H-balanced* if its input-partition is H-balanced.

Let $\mathscr{P}_{G,H}(f)$ denote the set of all H-balanced oblivious protocols that compute f.

Definition 3.2 (Balanced Communication Complexity). Let H be an integer. For each function f, define

$$\Phi_{G,H}(f) = \min_{\mathscr{P} \in \mathscr{P}_{G,H}(f)} \left[\max_i \left(|R_i| + |S_i|\right) \right].$$

In the definition above, $\Phi_{G,H}(f)$ is the maximum number of bits a processor has to exchange (i.e., send and receive) in an optimal protocol. We first observe that if the balanced communication complexity of a function f is small, then f can be computed by a circuit of a small size. We use the following classical result from Boolean complexity theory [23].

Theorem 3.3 [23]. *Every boolean function of n input variables and m output variables can be computed by a circuit of size $O(m2^n/n)$.*

Lemma 3.4. *Let G be a graph and let H be a positive integer. Every function $f \in \mathscr{B}_{n,m}$ can be computed by a circuit of size $O((m/h + n)2^h)$, where $h = H + \Phi_{G,H}(f)$.*

Proof. For each fluent oblivious distributed protocol $\mathscr{P}$ that computes f, let C_i be a boolean circuit that computes f_i: $(X_i \cup R_i) \rightarrow (Y_i \cup S_i)$. Property connecting C_i's, we obtain a boolean circuit C that computes f. The size of C is equal to the sum of the sizes of C_i's. By Lemma 3.3, the size of $|C_i|$ is bounded by $O(|Y_i \cup S_i| 2^h/h)$. Notice that $\sum_{i=0}^{N-1} |Y_i| = m$ and $\sum_{i=0}^{N-1} |S_i| \leq hN = O(hn)$. Thus, the size of C is bounded by $O((m/h + n)2^h)$. □

4. Communication Topologies and Functional Inversion

The topology of a communication network plays an important role in designing communication-efficient protocols. The set of communication topologies studied in this paper includes cliques, meshes, planar graphs, rings, and bounded degree trees. The corresponding protocols are respectively called *ideal protocols*, *mesh protocols*, *planar protocols*, *ring protocols*, and *tree protocols*.

In this section we examine the communication power of rings and trees, and show that no one-way function can be computed on a ring or a tree with bounded information exchange between neighbors.

4.1. *Rings*

Theorem 4.1. *If $f \in \mathscr{B}_{n,m}$ has an H-balanced ring-protocol with communication complexity $\Phi_H(f)$, then f has an inversion circuit of size $(m+n)2^{O(h)}$, where $h = H + \Phi_H(f)$.*

The remainder of this section gives a proof to Theorem 4.1.

Suppose $\mathscr{P}$ is an H-balanced ring-protocol that computes a function f whose communication complexity is $\Phi_H(f)$. Let $(X_0, \ldots, X_{N-1})$ be the H-balanced partition induced by $\mathscr{P}$ on X and let $(Y_0, \ldots, Y_{n-1})$ be the partition induced on Y. Let $U_i = (u_{i,1}, \ldots, u_{i,l_i})$ and $V_i = (v_{i,1}, \ldots, v_{i,r_i})$ be the set of communication variables whose values p_i sends to p_{i-1} and p_{i+1}, respectively, in $\mathscr{P}$ (see Fig. 2), where l_i and r_i are the number of bits that p_i sends to p_{i-1} and p_{i+1}. (For simplicity, we perform all index arithmetic modulo N.) Let $h = H + \Phi_H(f)$. By definition, we have $l_{i+1} + r_{i-1} \leq h$.

Let f_i be the projected boolean function associated with p_i, from $(X_i \cup V_{i-1} \cup U_{i+1})$ to $(Y_i \cup U_i \cup V_i)$. Because f_i has only h input bits, f_i is computable by a circuit of size $O((|Y_i| + h)2^h/h)$ (see Theorem 3.3). Let C_i be such a circuit that computes f_i.

For each $y \in \mathscr{B}^m$, we now define a digraph G_y which has the property that G_y has a cycle of length N iff $x \in \mathscr{B}^n$ exists such that $f(x) = y$. By our construction, G_y will have no cycle of length less than N.

For each output $y \in \mathscr{B}^m$, let T_i be a $2^{l_i} \times 2^{r_{i-1}} \times 2^{l_{i+1}} \times 2^{r_i}$ table whose entries are defined as follows.

For each $i: 0 \leq i < N$, and for each assignment u_i, v_{i-1}, u_{i+1}, v_i to U_i, V_{i-1}, U_{i+1}, V_i.

- **if** there is an assignment x_i to X_i, such that $f_i(x_i, v_{i-1}, u_{i+1}) = (y_i, u_i, v_i)$,
  then $T_i[u_i, v_{i-1}, u_{i+1}, v_i] = x_i$. If there is more than one such satisfying assignment to X_i, then choose such an x_i arbitrarily.
- **else** $T_i[u_i, v_{i-1}, u_{i+1}, v_i] = \varnothing$.

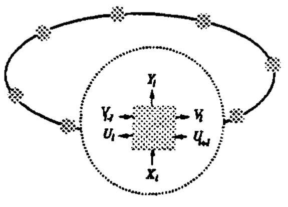

Fig. 2. The communication variables between neighbors.

Notice that $U_i \cup V_{i-1}$ is the set of all communication variables between p_{i-1} and p_i and $U_{i+1} \cup V_i$ is the set of all communication variables between p_i and p_{i+1}. For each assignment $u_i, v_{i-1}, u_{i+1}, v_i$ to $U_i, V_{i-1}, U_{i+1}, V_i$, if $T_i[u_i, v_{i-1}, u_{i+1}, v_i] = x_i$, then we say (u_i, v_{i-1}) and (u_{i+1}, v_i) are *locally consistent* with y_i. We now show that if assignments (u_0, v_{N-1}), (u_1, v_0), $\ldots$, (u_{N-1}, v_{N-2}) to (U_0, V_{N-1}), (U_1, V_0), $\ldots$, (U_{N-1}, V_{N-2}), respectively, exist such that, for all i: $0 \le i < N$, (u_i, v_{i-1}) and (u_{i+1}, v_i) are locally consistent with y_i, and $T_i[u_i, v_{i-1}, u_{i+1}, v_i] = x_i$, then $f(x) = y$, where $y = (y_0, \ldots, y_{N-1})$ and $x = (x_0, \ldots, x_{N-1})$.

We first introduce a digraph $G_y = (V, E)$ where

$$V = \bigcup_{i=0}^{N-1} (\{i\} \times \mathscr{B}^{l_i} \times \mathscr{B}^{r_{i-1}}),$$

$$E = \{((i, u_i, v_{i-1}), (i+1, u_{i+1}, v_i)) \mid T_i[u_i, v_{i-1}, u_{i+1}, v_i] \neq \varnothing\}.$$

Lemma 4.2. *G_y has no cycle of length less than N. Moreover, G_y has a cycle of length N iff $x \in \mathscr{B}^n$ exists such that $f(x) = y$.*

Proof. It follows from our construction that G_y has no cycle of length less than N. Suppose that $x \in \mathscr{B}^n$ exists such that $f(x) = y$. Since $\mathscr{P}$ computes f, for each $0 \le i < N$ there are x_i, u_i, and v_i such that $(y_i, u_i, v_i) = f_i(x_i, v_{i-1}, u_{i+1})$, and hence $((i, u_i, v_{i-1}), (i+1, u_{i+1}, v_i))$ is an edge in G_y. Thus $(0, u_0, v_{N-1})$, $(1, u_1, v_0)$, $\ldots$, $(N-1, u_{N-1}, v_{N-2})$, $(0, u_0, v_{N-1})$ forms a cycle in G_y of length N. On the other hand, suppose there is a cycle of length N in G_y. From our construction, it must contain exactly one node from $\{i\} \times \mathscr{B}^{l_i} \times \mathscr{B}^{r_{i-1}}$, and hence must be of the form

$$(0, u_0, v_{N-1}), (1, u_1, v_0), \ldots, (N-1, u_{N-1}, v_{N-2}), (0, u_0, v_{N-1}).$$

By the definition of G_y, there is an x_i such that $(y_i, u_i, v_i) = f_i(x_i, v_{i-1}, u_{i+1})$, and hence $f(x) = y$. □

The following is a procedure for inverting f given C_i.

```
Algorithm Ring-Inversion
Inputs: $y \in \mathscr{B}^m$.
    1. Compute $T_i$ (for all $0 \le i \le N$) using $C_i$;
    2. Construct the digraph $G_y$, using $T_i$'s;
    3. if $G_y$ has no cycle of length $N$, then output that there is no $x$ such
       that $f(x) = y$;
    4. else compute a cycle $(0, u_0, v_{N-1})$, ..., $(N-1, u_{N-1}, v_{N-2})$,
       $(0, u_0, v_{N-1})$ of length $N$ in $G_y$, and output $x = (x_0, \ldots, x_{N-1})$, where
       $x_i = T_i[u_i, v_{i-1}, u_{i+1}, v_i]$.
```

Step 1 takes at most $(m+n)2^{O(h)}$ time, because, for each processor, there are only $2^{O(h)}$ entries to fill and it takes at most $(|Y_i| + h)2^{O(h)}/h$ time to fill an entry. The graph G_y can be constructed in $(m+n)2^{O(h)}$ time using the information obtained from step 1. Furthermore, using the prefix sum path compression technique [4], we can decide in $(m + n)2^{O(h)}$ time whether G_y has a simple cycle of length N, and in $(m + n)2^{O(h)}$ time we can compute such a cycle if there is one. Therefore, the procedure above takes $(m + n)2^{O(h)}$ time. Let C be a circuit that simulates the procedure above. The size of C is bounded from above by $(m + n)2^{O(h)}$, completing the proof of Theorem 4.1.

A function $f \in \mathscr{B}_{n,m}$ is *h-ring-partitionable* if it can be computed by an H-ring protocol such that $H + \Phi_H(f) \leq h$.

Corollary 4.3. *If f is $O(\log n)$-ring-partitionable, then f has a polynomial-sized inversion circuit. Hence, no one-way function is $O(\log n)$-ring-partitionable.*

It worthwhile to point out that the procedure Ring-Inversion is NC^1 computable. Steps 1 and 2 can be performed in constant time with $(m + n)2^{O(h)}$ processors. Steps 3 and 4 are basically path compression (prefix sum), hence by Ladner and Fischer [9], they are NC^1 computable if $2^{O(h)}$ is a polynomial in $m + n$. Therefore, we have

Corollary 4.4. *If f is $O(\log n)$-ring-partitionable, then there is an NC^1 circuit computing an inverse of f.*

4.2. *Trees*

The above result on rings can readily be extended to one-dimensional meshes (a path of N nodes). A one-dimensional mesh can be viewed as a ring where there is no communication between the first and the last processors. In fact, we can extend the result even further to any bounded degree tree.

Theorem 4.5. *If $f \in \mathscr{B}_{n,m}$ has an H-balanced tree-protocol with communication complexity $\Phi_H(f)$, then f has an inversion circuit of size $(m + n)2^{O(h)}$, where $h = H + \Phi_H(f)$.*

First, notice that the communication complexity $\Phi_H(f)$ in Theorem 4.5 puts an upper bound of $\Phi_H(f)$ on the degree of the underlying tree, because if there is no message between a pair of nodes in the tree, then we can delete the edge that connects them without affecting the protocol. Hence, Theorem 4.5 implicitly assumes that the degree of the tree is bounded by $\Phi_H(f)$.

We prove Theorem 4.5 by reducing the inversion problems on trees to the following *consistency problem* on trees.

A *labeled tree* is a 3-tuple (T, G, Z) where T is a tree with N nodes $\{0, \ldots, N - 1\}$; $Z = \{Z_0, \ldots, Z_{N-1}\}$ is a set of boolean variable sets; and $G = \{g_0, \ldots, g_{i-1}\}$ is a set of boolean functions. Let $NN(i)$ denote the set of neighbors of i in T. Each node i in T is associated with a set Z_i of k_i boolean variables and a boolean function g_i of Δ_i variables from $Z_i \cup (\bigcup_{j \in NN(i)} Z_j)$, where $\Delta_i \leq |Z_i| + \sum_{j \in NN(i)} |Z_j|$. An assignment to variables in Z *satisfies* g_i, if the value of g_i is 1 under this assignment.

Definition 4.6 (Consistency Problem on Trees). Given a labeled tree (T, G, Z), compute an assignment of Z which satisfies all functions g_i, $0 \leq i < N$.

Theorem 4.7 [15]. *The consistency problem on trees can be solved in $2^{O(\max \Delta_i)}n$ time.*

The consistency problem on trees is solved by the dynamic programming approach, where the bottom-up dynamic programming evaluation on an input tree is supported by the RAKE operation [15], [14], [4]. Using the parallel tree contraction technique of Miller and Reif [14], Teng [20] gave an optimal $O(\log n)$ parallel-time algorithm for this problem when $(\max_i \Delta_i) = O(\log n)$.

We now reduce the inversion problem on trees to the consistency problem on trees. Let $\mathscr{P}$ be an H-balanced tree-protocol on T that computes f with communication complexity $\Phi_H(f)$. Let $(X_0, \ldots, X_{N-1})$ be the H-balanced partition induced by $\mathscr{P}$ on X and let $(Y_0, \ldots, Y_{N-1})$ be the partition induced on Y. Recall that S_i and R_i denote the set of all communication variables p_i sends to and receives from, respectively, its neighbors. Let f_i: $X_i \cup R_i \to Y_i \cup S_i$ be the projected function associated with p_i, as defined in Section 3.2. Notice that $|X_i| + |S_i| + |R_i| \leq H + \Phi_H(f) = h$. Because f_i has at most h input bits, f_i is computable by a circuit of size $O([|Y_i| + |S_i|]2^h/h)$ (see Theorem 3.3). Let C_i be such a circuit that computes f_i.

Now, let $Z_i = X_i \cup S_i$ and let g_i be a function from $X_i \cup S_i \cup R_i$ to $\{0, 1\}$ such that, for an assignment x_i to X_i, s_i to S_i, and r_i to R_i, g_i has value 1 if $(y_i \cup s_i) = f_i(x_i \cup r_i)$.

Lemma 4.8. *For each $y \in \mathscr{B}^m$, there are assignments x_i to X_i, s_i to S_i, and r_i to R_i that satisfy all g_i's simultaneously iff there is an assignment x to X such that $f(x) = y$.*

Proof. Suppose there is an x satisfying $f(x) = y$. Then the execution of the tree protocol will produce x_i, s_i, and r_i $(0 \leq i < N)$ such that $(y_i \cup s_i) = f_i(x_i \cup r_i)$. Therefore, all g_i's are satisfied simultaneously. On the other hand, suppose that there is an assignment x_i to X_i, s_i to S_i, and r_i to R_i satisfying all g_i's simultaneously. This implies that $(y_i \cup s_i) = f_i(x_i \cup r_i)$ which in turn implies $f(x) = y$. □

Consequently, for each $y \in \mathscr{B}^m$, in $(m + n)2^{O(h)}$ time, we can, using the algorithm for the consistency problem on trees, compute an $x \in \mathscr{B}^n$ such that $f(x) = y$ (if such an x exists). Let C be a circuit that simulates the above algorithm. C has size bounded from above by $(m + n)2^{O(h)}$, completing the proof of Theorem 4.5.

A function $f \in \mathscr{B}_{n,m}$ is *h-tree-partitionable* if it can be computed by an H-tree-protocol such that $H + \Phi_H(f) \leq h$.

Corollary 4.9. *If f is $O(\log n)$-tree-partitionable, then it has a polynomial-sized inversion circuit. Hence, no one-way function is $O(\log n)$-tree-partitionable.*

5. Inverting Planar Circuits

Definition 5.1 (Planar Circuit) [23]. A boolean circuit C is *planar* if:

(1) The underlying graph of C is planar.
(2) All input nodes are on the same face of the underlying graph, called the *input face*, and all output nodes are on the same face of the underlying graph, called the *output face*.

We now examine the relationship between the depth of a planar boolean circuit and the balanced communication complexity on the ring of the function it computes.

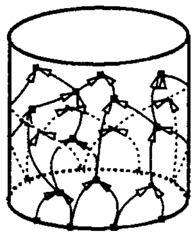

Fig. 3. Embedding planar circuits on a cylinder.

Lemma 5.2. *If a function $f \in \mathscr{B}_{n,m}$ can be computed by a planar circuitof depth d, then f is $O(d)$-ring-partitionable.*

Proof. Without loss of generality, we assume that C is embedded on the surface of a cylinder with the input face at the bottom of the cylinder and the output face at the top of the cylinder [13] (see Fig. 3). Without loss of generality, assume n is a multiple of d, and $x_0, \ldots, x_{n-1}$ appear in the conterclockwise order around the input face. Let P_0 be a path from the input node x_0 to an output node. Use the following process to generate $n/d - 1$ more paths iteratively.

```
• for $i = 1$ to $n/d - 1$
    grow a path from $x_{id}$ upward edge by edge until one of the following
    conditions becomes true (for the first time): (i) we reach an output
    node; (ii) we reach a node of $\bigcup_{j=0}^{i-1} P_i$. Call this path $P_i$.
```

These n/d paths divide the planar circuit into $N = n/d$ regions each of which contains at most d-inputs (see Fig. 4). We call these regions $C_0, \ldots, C_{N-1}$ where C_i has boundaries P_{i-1} and P_i and both P_{i-1} and P_i belong to C_i. Two regions are neighbors if their boundaries have some nodes in common. Notice that some regions, e.g., C_3 and C_4 in Fig. 4, may have more than two neighbors. Fortunately, the planarity of the graph implies that this neighboring relation is *well nested* in the sense that if a region C_i has a neighbor C_{i-k}, then regions $C_{i-k+1}, \ldots, C_{i-1}$ cannot be neighbors to C_{i+1}.

We now assign processor p_i to handle region C_i. In other words, p_i receives the input variables belonging to C_i and is responsible for computing the set of all output

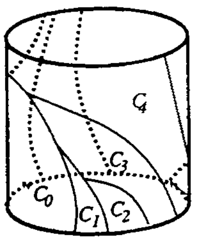

Fig. 4. Partition of a circuit by d-separators.

variables in C_i. To perform the computation, p_i needs to communicate with all its neighbors; however, in a ring protocol p_i is connected only with p_{i-1} and p_{i+1}. To resolve this problem, we use the following simple method. We use examples in Fig. 4 to illustrate the solution. To send a bit from p_3 to p_0, p_3 first sends that bit to p_2, and p_2 passes it to p_1, and p_1 to p_0. Similarly, p_3 can receive a bit from p_0. In general, a bit from p_i to p_j is passed sequentially along the ring. We now need to bound the number of bits sent by each processor. Notice that, for each $0 \leq i < N$, there is a unique path starting from x_{id} to an output node using edges only from $\bigcup_{j=0}^{N-1} P_j$. This path Q_i is called the extension of P_i. Because the regions are well nested, p_i only needs to pass bits which cross Q_{i-1} from right to left and bits which cross Q_i from left to right. Since $|Q_i| \leq d$, the total communication of each processor is bounded by $2d$.

Therefore, we have a d-balanced ring-protocol with $N = n/d$ processors where p_i evaluates C_i and communicates with p_{i-1} and p_{i+1} to evaluate nodes on the path Q_i. The communication complexity $\Phi_H(f)$ is bounded by $2d$. □

Consequently, by Theorem 4.1, we have

Theorem 5.3. *If f can be computed by a planar circuit of depth d, then it has an inversion circuit of size $(m + n)2^{O(d)}$.*

Corollary 5.4. *No one-way function is computable by a polynomial-sized planar circuit whose depth is $O(\log n)$.*

6. Area Requirement of One-way Functions

In his doctoral dissertation, Thompson [21], [22] introduced a grid model, which we refer to as the Thompson grid, for the study of VLSI layout. A Thompson grid (see Fig. 5) of size $m \times n$ is a regular $m \times n$ grid, where at each grid point we can embed a gate that computes a binary boolean operator.[1] Each grid edge can hold a constant number of wires, where we embed a constant number of edges of a boolean circuit. All inputs and outputs are located at grid points on the boundary

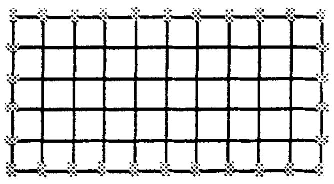

Fig. 5. The Thompson model for VLSI.

[1] In general, we can embed gates that compute constant-sized boolean functions. It is straightforward to extend our result to handle this case.

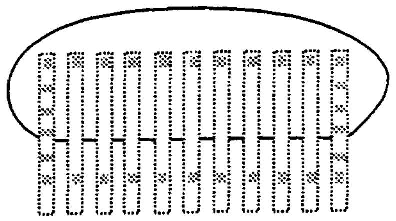

Fig. 6. The ring protocol for the Thompson layout.

of the grid.[2] The area of an $m \times n$ grid is mn. The *VLSI area complexity* of a function f is the area of the smallest Thompson grid on which we can embed a circuit that computes f. In this section we give a lower bound of $\Omega(n \log n)$ on the VLSI area requirement of one-way functions in the Thompson model.

This VLSI area lower bound is derived from the following simple observation.

Lemma 6.1. *If a function f has a circuit with layout area A, then f is $O(A/n)$-ring-partitionable, where n is the total size of the input and the output.*

Proof. Let x be the width and let y be the height of the optimal layout of f. Because all inputs and outputs are on the boundary of the Thompson grid, we have

$$2(x + y) \geq n,$$

$$x \cdot y = A.$$

Assume, without loss of generality, that $y \leq x$. We thus have $x \geq n/4$, and hence $y \leq 4A/n$.

We assign each column of the grid to a processor in a ring protocol with x processors as shown in Fig. 6. Each processor performs the function of its associated column. Because the height of the grid is y, each processor receives at most $O(y)$ bits during the execution of the ring protocol. Furthermore, the first and the last processors have at most y input bits and all others have at most 2 bits. Therefore, f is $O(3y + 2) = O(A/n)$ ring-partitionable. □

Consequently, by Theorem 4.1,

Theorem 6.2. *If a function f has a circuit with layout area A, then f has an inversion circuit of size $2^{O(A/n)}n$.*

Corollary 6.3. *The area required to layout any one-way function is at least $\Omega(n \log n)$.*

[2] The model with inputs and outputs on the boundary of grids is one of the most commonly used models, especially for studying IO-comjplexity in VLSI.

7. Negative Results

We have shown that if the balanced communication complexity of a function on trees or rings is small, then the function has "small" inversion circuit. The following observation shows that it is unlikely that the result can be extended to all topologies with a bounded degree. Our proof makes use of a result due to Boppana and Lagarias [1].

Theorem 7.1 (Boppana and Lagarias). *If SATISFIABILITY does not have a polynomial-size circuit, then a one-way function exists.*

Each boolean function $f \in \mathscr{B}_{n,m}$ induces m functions $g_0, g_1, \ldots, g_{m-1}$, where $g_i \in \mathscr{B}_{n,1}$ is a boolean function which computes the ith output variable of f. In other words, for each assignment x to X, $(g_0(x), g_1(x), \ldots, g_{m-1}(x)) = f(x)$.

In general, g_i depends only on a subset D_i of input variables X. Let k be a positive integer. The function f is a (k, k)-*function* if $|D_i| \leq k$ for all i, $0 \leq i < m$, and each input variable belongs to at most k D_i's. The next lemma follows immediately from Theorem 3.3.

Lemma 7.2. *If $f \in \mathscr{B}_{n,m}$ is a (k, k)-function, then f is computable by a circuit of size $O(m2^k/k)$ and depth k. In particular, if k is a constant, then f is an NC^0-function.*

7.1. (3, 3)-*One-Way Functions*

The following is a useful lemma for establishing the negative results.

Lemma 7.3. *If one-way functions exist, then one-way* (3, 3)-*functions exist.*

Proof. The lemma can be established by an argument similar to that used by Cook [2] and Håstad [6].

Suppose $f \in \mathscr{B}_{n,m}$ is a boolean function computable by a polynomial-size circuit C. Based on C, we construct a (3, 3)-boolean function $F \in \mathscr{B}_{n',m'}$ such that:

(1) n' and m' are polynomial in n and m.
(2) F is computable by a polynomial-size circuit.
(3) If f is a one-way function, then F is also a one-way function.

To construct such an F, we first construct a boolean function F' such that F' satisfies conditions (1), (2), and (3) above. Moreover, each output variable of F' depends on at most three input variables of F'.

For each internal node s of C, we introduce two boolean variables u_s and v_s, where u_s and v_s are called the *output variable* and *input variable* of s, respectively. For convenience, we call the variable of an input node of C *its input variable* and the variable of an output node of C *its output variable*. The set of input variables of F' is the set of all input variables of nodes in C and the set of output variables of F' is the set of all output variables of nodes in C.

For each node s, let $c_1(s)$ and $c_2(s)$ denote its children and let $v_{c_1(s)}$ and $v_{c_2(s)}$ denote their input variables, respectively. In function F' let u_s be an output variable of F'

defined to be the function of v_s, $v_{c_1(s)}$, and $v_{c_2(s)}$ such that $u_s = 1$ if $v_s = v_{c_1(s)} \odot v_{c_2(s)}$, and 0 otherwise, where $\odot$ is the binary boolean operator associated with node s. For each output node t with variable y_i, binary operator $\odot$, and children $c_1(t)$ and $c_2(t)$, we require $y_i = v_{c_1(t)} \odot v_{c_2(t)}$. Clearly, each output variable in F' depends on at most three input variables. Note that we do not need the u-variables for output nodes in the construction of F', but we may introduce them for simplicity.

Observe that in F' some input variables could influence more than three output variables. Suppose there are k output variables $u_1, u_2, \ldots, u_k$ that depend on an input variable v. We now introduce $2k - 1$ variables $z(v)_1, z(v)_2, \ldots, z(v)_k$ and $w(v)_1, w(v)_2, \ldots, w(v)_{k-1}$. Let $Z(v) = \{z(v)_1, z(v)_2, \ldots, z(v)_k\}$ and $W(v) = \{w(v)_1, w(v)_2, \ldots, w(v)_{k-1}\}$. The set of input variables X' of F is defined as

$$X' = \bigcup_{v \text{ is an input variable of } F'} Z(v).$$

Let $Y(F')$ be the set of output variables of F'. Then the set Y' of output variables of F is defined as

$$Y' = Y(F') \cup \left(\bigcup_{v \text{ is an input variable of } F'} W(v) \right).$$

In F, we require $w(v)_i = z(v)_i \oplus z(v)_{i+1}$ for all $1 \leq i < k$. Also we replace v by $z(v)_i$, symbolically, in set of variables on which u_i depends in F', for all $1 \leq i \leq k$. (2) Clearly, F is a (3, 3)-function.

From our construction, it is easy to check that F satisfies conditions (1) and (2). We now prove that F satisfies condition (3). Suppose f is a one-way function and suppose F is not. Let C' be a polynomial-sized boolean circuit that computes an inverse of F. The set of input variables of C' and Y' and the set of output variables of C' is X'. The following process constructs a polynomial-size circuit C'' from C' such that C'' computes an inverse of f.

1. If an input node of C' has a label from one of the $W(v)$ above, then label it 0.
2. If an input node of C' has a label from $Y(F')$ which is associated with an output variable of an internal node of C, then label it 1.
3. If an input node of C' has a label from $Y(F')$ which is associated with an output variable y_i in C, label it y_i.
4. If an output node of C' has a label from $Z(v)$, such that v is an input variable of an internal node of C, then label it $\perp$.
5. If an output node of C' has a label from $Z(v)$, such that v is an input node of C with variable x_i, then, if the label is $z(x_i)_1$, label it x_i, otherwise label it $\perp$.

From our construction of F', F, and C'', it can easily be shown that C'' computes an inverse of f. A similar construction can be used to show that F' satisfies condition (3). □

As a consequence of Theorems 7.1 and 7.3, we have the following corollary.

Corollary 7.4. *If SATISFIABILITY does not have a polynomial-sized circuit, then NC^0 one-way* (3, 3)*-functions exist*:

7.2. *Cliques*

We say a function $f \in \mathscr{B}_{n,m}$ is *h-partitionable* if there is an H-balanced protocol for f on an N-clique such that $H + \Phi_H(f) \leq h$.

Lemma 7.5. *If SATISFIABILITY does not have a polynomial-size circuit, then there is a* 4*-partitionable function f that has no polynomial-size inversion circuit.*

Proof. By Lemma 7.3 and Theorem 7.1, if SATISFIABILITY does not have a polynomial-size circuit, then a one-way (3, 3)-function exists. For such a one-way function f, we assign each y_i and x_i to a processor. To evaluate y_i, each processor needs to receive at most three bits from other processors. Moreover, each processor needs to send one bit to at most three other processors. Therefore, f is 4-partitionable. □

Consequently,

Corollary 7.6. *If SATISFIABILITY does not have a polynomial-size circuit, then there is a* 4*-partitionable function f which is neither $O(\log n)$-ring partitionable, nor $O(\log n)$-tree partitionable.*

Similarly, we have

Corollary 7.7. *Let k be a positive integer. If SATISFIABILITY does not have a circuit of size $O(2^{\log^k n})$, there is a* 4*-partitionable function f which is neither $(\log^k n)$-ring partitionable nor $(\log^k n)$-tree partitionable.*

8. Final Remarks

8.1. *The Uniform Inversion Problem*

Most results presented in this paper are stated in a nonuniform form. Our constructions from Sections 4.1 and 4.2 yield the following set of uniform results.

In the following statements, "given a protocol $\mathscr{P}$ for f" means that $\mathscr{P}$ is given in a circuit form, i.e., we are given a set of circuits C_i that compute f_i which is defined in Section 3.2 (see also Lemma 3.4).

Theorem 8.1.

1. *Given an oblivious H-balanced ring-protocol or tree-protocol for f, an inversion circuit for f with size $(m + n)2^{O(h)}$ and height $O(\log(m + n) + h)$ can be constructed in $(m + n)2^{O(h)}$ time sequentially, and in $O(\log(m + n) + h)$ parallel time, using $(m + n)2^{O(h)}$ processors, where $h = H + \Phi_H(f)$.*
2. *Given a planar circuit C of depth d, an inversion circuit for C of size $(m + n)2^{O(d)}$ and height $O(\log(m + n) + h)$ can be found in $O(d + \log(m + n))$ parallel time using $(m + n)2^{O(d)}$ processors.*

The uniform assumption makes it much easier to establish negative results. Instead of addressing the question of whether small inversion circuits exist, the uniform problem asks whether small inversion circuits that can be found efficiently exist. For example, it has been shown (see p. 259 in [3]) that the satisfiability problem, in which each clause contains at most three variables or the negation of variables and each variable or its negative is in at most three clauses (referred as the 3-3-*condition*), is NP-complete. Simply from the NP-completeness result above, it can be concluded that if $NP \neq P$ (rather than $NP \not\subset P/poly$) then an NP language exists that is 4-partitionable but its inversion problem is not in P.

In the definition of a planar circuit, it is crucial to impose the restriction that all inputs are one the same face. When this restriction is removed, the class of resulting circuits is called *general planar circuits*. We show that, in the uniform case, the computational power of general planar circuits is greater than that of planar circuits.

Theorem 8.2. *The uniform problem of inverting general planar circuits with constant depth and polynomial size is NP-hard.*

Proof. To prove the theorem, we use the following result due to Lichtenstein [11] (also see p. 259 in [3]).

Define a *planar* 3-*formula* to be a set of clauses C over a variable set U such that $|c| \leq 3$ for all $c \in C$ and the bipartite graph $G = (V, E)$, where $V = U \cup C$ and E contains exactly those pairs (u, c) that either u or its negation belongs to the clause c, is planar. Lichtenstein [11] showed that the stability problem over planar 3-formulae is NP-complete.

Observe that each planar 3-formula defines a boolean function f whose input variables are U and whose output variables are C. Clearly, f can be computed by a general planar circuit of depth 3. If the uniform problem of inverting general planar circuits with constant depth can be solved in polynomial time, then we can use it to find the preimage of the all 1 assignment to solve the satisfiability problem of planar 3-formulae in polynomial time. $\square$

Theorem 8.2 implies that for a polynomial-time Turing machine T, a general planar circuit of constant depth exists such that either it does not have a polynomial-size inversion circuit or T cannot find such an inversion circuit in polynomial time, provided that $NP \neq P$.

It remains open whether the problem of inverting general planar circuits with constant depth and constant fan-out is in polynomial time.

8.2. *Other Communication Topologies*

We have observed that our result can be obtained for $O(\log n)$ by n meshes.

Theorem 8.3. *If $f \in \mathscr{B}_{n,m}$ has an H-balanced mesh-protocol on the $O(\log n) \times n$ mesh with communication complexity $\Phi_H(f)$, then f has an inversion circuit of size $2^{O(h \log n)}n$, where $h = H + \Phi_H(f)$.*

It is remains open whether a similar result can be extended to n by n meshes.

8.3. *Two-Party Communication Complexity*

Finally, is there a nice way to relate the two-party communication complexity to the complexity of inverting permutations? We make the following conjecture: Let S_n be the set of all permutations from $\{1, \dots, n\}$ to $\{1, \dots, n\}$. For each $\pi \in S_n$, let

$$\mathcal{P}_\pi(X) = \{(\{x_{\pi(1)}, \dots, x_{\pi(k)}\}, \{x_{\pi(k+1)}, \dots, x_{\pi(n)}\}): 1 \leq k \leq n\}.$$

The *permutational communication complexity* of f denoted by $\mathcal{PC}(f)$, is defined to be

$$\mathcal{PC}(f) = \min_{\pi \in S_n} \left[\max_{(X_1, X_2) \in \mathcal{P}_\pi(X)} \mathcal{C}_f(X_1, X_2) \right].$$

Conjecture 8.4. *If the permutation communication complexity of a function $f \in \mathcal{B}_{n,m}$ is φ, then f has a ring protocol of communication complexity $O(\varphi)$.*

Acknowledgments

I would like to thank Alan Frieze, Merrick Furst, Hillel Gazit, Manpreet Khaira, Gary Miller, and Zhi-Li Zhang for helpful discussions. I would like to express my special thanks to all my referees for pointing out errors in the earlier version of the paper, and for valuable suggestions that greatly improved the paper. I thank Dan Spielman for his careful proof-reading and helpful discussions.

References

[1] R. Boppana and J. Lagarias. One-way functions and circuit complexity. In *Proc. Structure in Complexity Theory*. Lecture Notes in Computer Science, Vol. 223. Springer-Verlag, Berlin, pages 51–65, 1986.

[2] S. A. Cook. The complexity of theorem-proving procedures. *Proceedings of the 3nd Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[3] M. R. Garey and D. S. Johnson. *Computer and Intractability: a Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

[4] H. Gazit, G. L. Miller, and S.-H. Teng. Optimal tree contraction in the EREW model. In *Current Computations* (S. K. Tewsburg, B. W. Dickinson, and S. C. Schwartz, eds.), Plenum, New York, pages 139–156, 1988.

[5] J. Håstad. *Computational Limitations for Small Depth Circuits*. MIT Press, Cambridge, MA, 1986.

[6] J. Håstad. One-way permutations in NC^0. *Inform. Process. Lett.*, 26:153–156, 1987.

[7] M.-D. A. Huang and S.-H. Teng. Security, verifiability, and universality in distributed computing. *J. Algorithms*, 11:492–521, 1990.

[8] R. Impagliazzo and S. Rudich. Limits on the provable consequence of one-way permutations. *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 44–61, 1989.

[9] R. E. Ladner and M. J. Fischer. Parallel prefix computation. *J. Assoc. Comput. Mach.*, 27(4):831–838, 1980.

[10] F. T. Leighton. *Complexity Issues in VLSI*. Foundations of Computing. MIT Press, Cambridge, MA, 1983.

[11] D. Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.* 11(2):329–343, 1982.

[12] R. J. Lipton and R. E. Tarjan. Applications of planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1981.

[13] G. L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*, pages 376–382, 1984.

[14] G. L. Miller and J. H. Reif. Parallel tree contraction and its applications. *Proceedings of the 26th Symposium on Foundations of Computer Science*, pages 478–489, 1985.

[15] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Reading, MA, 1984.

[16] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digitial signatures and public-key cryptosystems. *Comm. ACM*, 21(2): 120–126, 1978.

[17] J. Rompel. One-way functions are necessary and sufficient for secure signatures. *Proceedings of the 22th Annual ACM Symposium on Theory of Computing*, pages 387–394, 1990.

[18] C. Sturtivant and Z.-L. Zhang. Efficiently inverting bijections given by straight line programs. *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, pages 327–334, 1990.

[19] M. Szegedy. Functions with bounded symmetric communication complexity and circuit with mod *m* gates. *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 278–286, 1990.

[20] S. H. Teng. Fast parallel algorithms for tree-based constraint satisfaction problems. Manuscript, Carnegie Mellon University, 1990.

[21] C. D. Thompson. A Complexity Theory for VLSI. Ph.D. thesis, Department of Computer Science, Carnegie Mellon University, 1980.

[22] C. D. Thompson. The VLSI complexity of sorting. *IEEE Trans. Comput.*, 32(12): 1171–1184, 1983.

[23] I. Wegener. *The Complexity of Boolean Functions*. Wiley–Teubner Series in Computer Science. Wiley, New York, Teubner, Stuttgart, 1987.

[24] A. C.-C. Yao. Some complexity questions related to distributive computing. *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*, pages 209–213, 1979.

[25] A. C.-C. Yao. Theory and application of trapdoor functions. *Proceedings of the 23th Anual Symposium on Foundations of Computer Science*, pages 80–91, 1982.