Yale University
Department of Computer Science

Lectures on Network Complexity

presented by Prof. Michael J. Fischer
at the University of Frankfurt

YALEU/DCS/TR-1104
June 1974
Revised April 1977, April 1996

This research was supported in part by the National Science Foundation under research grant GJ-43634x to M.I.T.

Lectures on Network Complexity

presented by Prof. Michael J. Fischer
at the University of Frankfurt

Preface

These notes, often referred to as the "Frankfurt Lecture Notes", are perhaps my most widely circulated unpublished work. Resulting from a series of lectures I gave at the University of Frankfurt in June of 1974, they summarize some early work on what is now known as circuit complexity. They circulated originally in the form of xeroxes of my hand-written notes. Later, in April of 1977, I revised the notes and had them typed up; copies of the typewritten version have also circulated widely. Now, with the availability of the world-wide web, I have decided to reissue them once again, this time in electronic form.

In going over the notes again, I have tried to preserve the original style and to resist the temptation to make "improvements" to either the content or its presentation. Nevertheless, I have fixed a few technical errors and have added a few missing assumptions here and there. I have also added a bibliography that was not present in the original. While I make no claims to its completeness, I have tried to add citations for the results referenced in the original notes, as well as giving references to a few related subsequent works.

Even today, more that 20 years after the original lectures, readers may still find some material here of interest:

- Section 1 presents counting arguments that establish upper and lower bounds on the maximum circuit complexity of any n-argument Boolean function over the full basis of 2-input gates. These and closely related results appear in [4, 12, 23, 25]. The particularly slick proof of Theorem 1.1 is due to Schnorr [20].
- Section 2 uses Turing time complexity $T(n)$ to bound circuit complexity for families of Boolean functions. Savage [18] showed that the circuit complexity is at most $O(T(n)^2)$. Here I present a result with Pippenger that reduces this bound to $O(T(n))$ for oblivious Turing machines and to $O(T(n) \log T(n))$ for unrestricted Turing

This research was supported in part by the National Science Foundation under research grant GJ-43634x to M.I.T.

machines. These results subsequently appeared in [17] and were extended in [21].

- Section 3 illustrates the use of Turing machines to construct a small circuit for computing the transitive closure of a symmetric Boolean matrix. The Turing machine construction relies on the mail carrier algorithm that I developed with Paterson. These same ideas underlie the "Fishspear" priority queue algorithm [7] but have never been published in their simple form.
- Section 4 examines *inversion complexity*, the number of *not*-gates needed to compute a set F of Boolean functions. Markov [13] characterizes the exact minimal number of *not*-gates needed for arbitrary F in terms of the number of times F violates monotonicity along any monotone sequence of input vectors. Section 4 begins with a complete proof of his claims. I knew, as a curiosity, of a circuit that negates its three inputs using only two *not*-gates. Surprisingly, a circuit exists of size $O(n^2 \log^2 n)$ that negates its n inputs using only $\lceil \log(n) + 1 \rceil$ *not*-gates. This result can be used to convert an arbitrary circuit of size C into an equivalent one of size $2C+O(n^2 \log^2 n)$ that uses only $\lceil \log(n)+1 \rceil$ *not*-gates. These results appear in [5] and have been recently improved in [3, 26]. The question of how much the size increases as the number of *not*-gates is reduced further to the exact inversion complexity of F remains open.

Related Work Gilbert [9] considered inversion complexity as early as 1954. He was motivated by relay contact networks in which devices for performing negation are more expensive or less reliable than those for performing the other logical operations. Early papers to synthesize circuits with minimal numbers of negations include [5, 10, 15]. General references with a wealth of information relating combinational complexity to other notions of finite function complexity include Savage [19] and more recently Wegener [27]. Both contain extensive bibliographies. Paterson [16] presents a nice collection of recent research papers on Boolean function complexity.

Michael J. Fischer
New Haven, Connecticut
April 1996

1 Boolean Networks

Let $K = \{0, 1\}$, $f: K^n \to K$. We consider computations of f by networks over a basis Ω of Boolean functions. A network η is a directed acyclic graph. Each node v is labeled with an operation $\mathrm{op}(v)$ in $\Omega \cup \{x_1, \ldots, x_n\}$. Let $\rho(f)$ be the rank (number of arguments) of f. Then the indegree of any node v must equal $\rho(\mathrm{op}(v))$. With each node is associated a function $K^n \to K$ in the obvious way. Certain nodes are designated as output nodes, so the network computes a set of functions $F \subseteq K^{K^n}$. Denote K^{K^n} by A_n.

Let $L(\eta) = \sum_{v \in \eta} \mathrm{cost}(\mathrm{op}(v))$. One may use various cost functions. For definiteness, let $\mathrm{cost}(a) = 0$ if $\rho(a) \leq 1$ and $\mathrm{cost}(a) = 1$ otherwise. Extend L to functions and sets of functions:

$$L(F) = \min\{L(\eta) \mid \eta \text{ computes } F\},\ F \subseteq A_n,$$

and $L(f) = L(\{f\})$ for a single function f. Note that L depends on both the basis Ω and the cost function "cost". We sometimes write L^Ω or $L^{\Omega,\mathrm{cost}}$ to emphasize this dependence.

Theorem 1.1 *For the basis* $\Omega = \bigcup_{i=0}^{2} A_i$ *and any* $\varepsilon > 0$*,*

$$\frac{|\{a \in A_n \mid L^\Omega(a) \leq (1-\varepsilon)\frac{2^n}{n}\}|}{|A_n|} \to 0 \text{ as } n \to \infty.$$

Proof: Let $N_q = |\{a \in A_n \mid L^\Omega(a) \leq q\}|$. We want an upper bound on N_q.

Let β be a network over Ω which computes $a \in A_n$. Then there exists an equivalent network β' (i.e., β' also computes a) over A_2 such that $L(\beta') \leq L(\beta)$, assuming only that $L(\beta) \geq 1$. Thus it suffices to consider only networks over A_2.

The maps

$$\sigma: \{1, \ldots, q\} \to (\{1, \ldots, q\} \cup \{x_1, \ldots, x_n\})^2 \times A_2$$

describe precisely the networks β_σ over A_2 of cost $L(\beta_\sigma) = q$. There are $[(q+n)^2 \cdot 16]^q$ such maps, and each network defines q functions in A_n by varying the choice of output wire. However, the same function is described many times.

Let $\mathrm{Res}(\beta, v) =$ function computed by node v of network β.

Claim 1 *If* $L(f) \leq q$*, then* $\exists\sigma$ *such that* β_σ *computes* f*,* $L(\beta_\sigma) \leq q$*, and* $\forall v, v' \in \beta_\sigma\ [v \neq v' \Rightarrow \mathrm{Res}(\beta_\sigma, v) \neq \mathrm{Res}(\beta_\sigma, v')]$. *Call such a network* reduced.

Let π be a permutation of $\{1, \ldots, q\}$. π can be extended to a permutation on maps σ in a natural way. First, define $\pi x_i = x_i$, $1 \le i \le n$. Now, let $(\pi\sigma)$ be the map such that $(\pi\sigma)(\pi v) = (\pi w, \pi w', a)$, where $(w, w', a) = \sigma(v)$.

Claim 2 $\mathrm{Res}(\beta_{\pi\sigma}, \pi v) = \mathrm{Res}(\beta_\sigma, v)$.

Claim 3 *If β_σ is reduced, then $\pi\sigma = \sigma \implies \pi v = v$.*

Pf. If $\pi\sigma = \sigma$, then by Claim 2, $\mathrm{Res}(\beta_\sigma, \pi v) = \mathrm{Res}(\beta_\sigma, v)$. Since β_σ is reduced, $\pi v = v$. □

For each reduced β_σ, the $q!$ maps $\pi_1\sigma, \pi_2\sigma, \ldots, \pi_{q!}\sigma$, are all distinct, where the π_i's are all the permutations of $\{1, \ldots, q\}$. Thus,

$$\begin{aligned} N_q &\le \frac{q[(q+n)^2 \cdot 16]^q}{q!} < \frac{c'^q \cdot (q+n)^{2q}}{q^q} && \text{by Claim 1 and Stirling's formula} \\ &< (cq)^q \end{aligned}$$

for c a constant, assuming that $q \ge n$.

Now set $q = (1-\varepsilon)\frac{2^n}{n}$. Then

$$N_q \le \left[c(1-\varepsilon)\frac{2^n}{n}\right]^{(1-\varepsilon)2^n/n} \le 2^{(1-\varepsilon)2^n}$$

for $n \ge c(1-\varepsilon)$. Thus,

$$\frac{N_q}{|A_n|} \le 2^{(1-\varepsilon)2^n - 2^n} = 2^{-\varepsilon 2^n} \to 0 \text{ as } n \to \infty.$$

■

Theorem 1.2 $\mathrm{Max}_{a \in A_n} L(a) \le \frac{2^n}{n} + o\left(\frac{2^n}{n}\right)$, *even taking $\Omega = \{0, 1, \oplus, \wedge\}$. ($x \oplus y$ means $x \not\equiv y$.)*

Proof: Let

$$L(m, t) = \max\{L(F) \mid F \subseteq A_m \ \& \ |F| = t\}.$$

We first prove two lemmas.

Lemma 1.3 *For any integer $p \ge 1$,*

$$L(m, t) \le 2^m + \left\lceil \frac{2^m}{p} \right\rceil 2^p + \frac{t 2^m}{p}.$$

Pf.

1. Compute all products $x_1^{\delta_1} \cdots x_m^{\delta_m}$, $0 \le \delta_1, \ldots, \delta_m \le 1$. There are 2^m such products, and each can be computed using at most one $\wedge$-gate from previously computed products. (Do all of length ℓ before any of length $\ell + 1$.)

2. Partition the products of step 1 into $q = \left\lceil \frac{2^m}{p} \right\rceil$ classes $C_1, \ldots, C_q$ each with at most p elements. For each i, $1 \le i \le q$, compute all the linear combinations (sums) of terms in C_i. Call the result B_i. $|B_i| \le 2^p$, and each element in B_i may be computed using at most one $\oplus$-operation from shorter ones. Thus, this step costs at most $q2^p = \left\lceil \frac{2^m}{p} \right\rceil 2^p$.

3. Each of the desired t functions can be expressed as a sum $\bigoplus_{i=1}^q \alpha_i g_i$, where $\alpha_i \in \{0,1\}$ and $g_i \in B_i$. Hence it can be computed given the sets B_i using at most $q-1$ $\oplus$-operations, so the total cost for this step is $\le t(q-1) \le \frac{t2^m}{p}$.

$\Box$

Lemma 1.4 *For any integers* $p \ge 1$, $0 \le m \le n$,

$$L(n,1) \le 3 \cdot 2^{n-m} + 2^m + \left\lceil \frac{2^m}{p} \right\rceil 2^p + \frac{2^n}{p}.$$

Pf.

1. Compute all products $x_{m+1}^{\delta_{m+1}} \cdots x_n^{\delta_n}$, $0 \le \delta_{m+1}, \ldots, \delta_n \le 1$ using at most 2^{n-m} $\wedge$-operations. (Cf. Step 1 of proof of Lemma 1.3.)

2. Any function $f(x_1, \ldots, x_n)$ can be expressed as

$$f(x_1, \ldots, x_n) = \bigoplus_{0 \le \delta_{m+1}, \ldots, \delta_n \le 1} \beta_{\delta_{m+1} \ldots \delta_n}(x_1, \ldots, x_m) \cdot x_{m+1}^{\delta_{m+1}} \cdots x_n^{\delta_n}.$$

   Compute the $t = 2^{n-m}$ β's using Lemma 1.3 for a cost of $2^m + \left\lceil \frac{2^m}{p} \right\rceil 2^p + \frac{2^n}{p}$.

3. Compute f from the results of steps 1 and 2 using an additional 2^{n-m} $\wedge$-gates and $2^{n-m} - 1$ $\oplus$-gates.

$\Box$

To prove the theorem, it remains to choose p and m. Let $m = \lceil \sqrt{n}\, \rceil$, $p = n - m - \lceil \log n \rceil$. Assuming $n \ge 7$, we have $p \ge 1$ and $0 \le m \le n$. Apply Lemma 1.4 and examine the terms in turn:

$$3 \cdot 2^{n-m} \le 3 \cdot 2^{n-\sqrt{n}} = o\left(\frac{2^n}{n}\right).$$

$$2^m \leq 2^{\sqrt{n}+1} = o\left(\frac{2^n}{n}\right).$$

$$\begin{aligned}\left\lceil \frac{2^m}{p} \right\rceil \cdot 2^p &= \left\lceil \frac{2^m}{n-m-\lceil \log n \rceil} \right\rceil \cdot 2^{n-m-\lceil \log n \rceil} \\ &\leq \frac{2^{n-\log n}}{n - \lceil \sqrt{n}\, \rceil - \lceil \log n \rceil} + 2^{n-\lceil \sqrt{n}\, \rceil - \lceil \log n \rceil} = o\left(\frac{2^n}{n}\right).\end{aligned}$$

$$\frac{2^n}{p} = \frac{2^n}{n}\left(1 + \frac{n-p}{p}\right) = \frac{2^n}{n}\left(1 + \frac{m + \lceil \log n \rceil}{n-m-\lceil \log n \rceil}\right) = \frac{2^n}{n} + o\left(\frac{2^n}{n}\right).$$

Summing, we get $\max_{a \in A_n} L(a) = L(n,1) \leq \frac{2^n}{n} + o\left(\frac{2^n}{n}\right)$. ■

Note added in revision: According to Savage [19, p. 123], Shannon [23] first proved Theorem 1.1 as well as a weaker version of Theorem 1.2. Savage credits the stronger version presented here to Lupanov [12]. Our proof of Theorem 1.1 is due to Schnorr [20].

2 Combinational Complexity

We extend the ideas of the previous section to give a complexity measure on certain functions on binary sequences. The results in this section appear in [17].

A function $f\colon K^* \to K^*$ is *length respecting* if $|x| = |y| \implies |f(x)| = |f(y)|$. Let $\lambda_f(n) = |f(0^n)|$. For a length-respecting function f, define the *combinational complexity*

$$C_f(n) = L(F_n),$$

where

$$F_n = \{g \in A_n \mid \exists i, 1 \leq i \leq \lambda_f(n), \forall \vec{x} \in K^n \; g(\vec{x}) = i^{\text{th}} \text{ symbol of } f(\vec{x})\}.$$

Corollary 2.1 $C_f(n) \leq \frac{2^n}{n} + o\left(\frac{2^n}{n}\right)$ *for any function f (even non-recursive) $: K^* \to K$.*

Thus, there are arbitrarily large differences between Turing machine time complexity and combinational complexity. However, we shall see that a small Turing machine complexity implies a small combinational complexity.

By "Turing machine" we mean a multitape Turing machine with separate input and output tapes, as illustrated in Figure 1. The input head is read-only and moves only to the right, and the output tape is write-only and likewise moves only to the right.

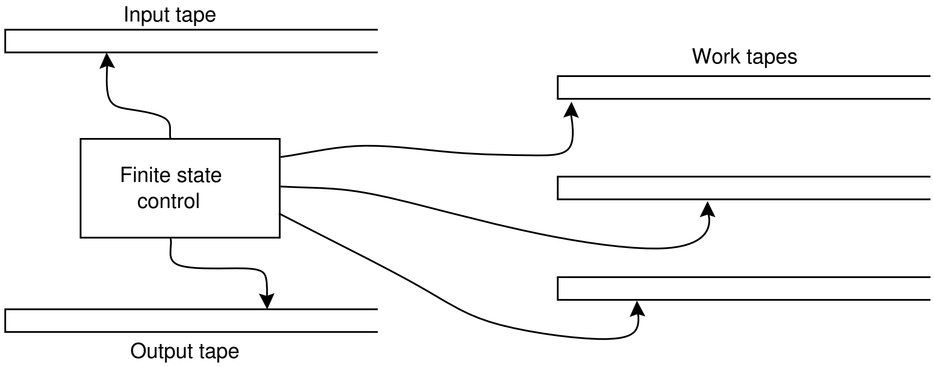


Figure 1: A multitape Turing machine.

Let $\mathcal{M}$ be a Turing machine, $T\colon N \to N$. $\mathcal{M}$ *computes* f *in time* T if for all $x \in K^*$, $\mathcal{M}(x) = f(x)$ and $\mathcal{M}$ on input x uses at most $T(|x|)$ computational steps.

A Turing machine $\mathcal{M}$ is *oblivious* if for fixed n, the positions of all heads at each step s are the same for all $x \in K^n$. (This includes the input and output heads.)

Theorem 2.2 *Let $\mathcal{M}$ be an oblivious Turing machine which computes a length-respecting function f in time T. Then $C_f(n) = O(T(n))$.*

Proof: Encode instantaneous descriptions of the Turing machine by binary sequences of length $O(T(n))$. Let σ_i, τ_i be id's which occur at the i^{th} step of the computations on two length n inputs x, y. Since $\mathcal{M}$ is oblivious, all head positions are the same, so the successor id's differ only at a fixed set of positions. Hence, there exists a network η_i of size independent of n which computes the $(i+1)^{\text{st}}$ id from the i^{th}, as illustrated in Figure 2. A

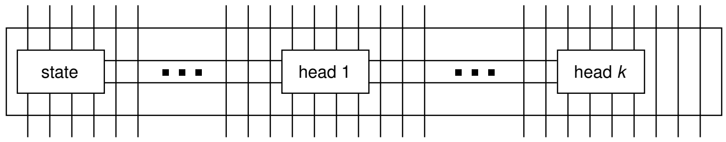
$\eta_i =$ 

Figure 2: Simulating a step of $\mathcal{M}$.

network for f is obtained by composing $\eta_1 \circ \ldots \circ \eta_{T(|x|)}$ and setting the inputs appropriately, as illustrated in Figure 3. ■

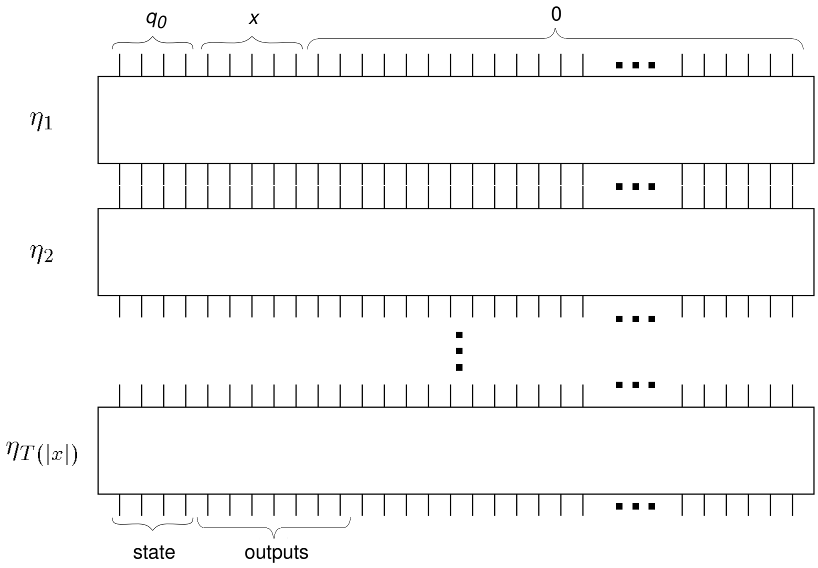


Figure 3: Simulating a full computation of $\mathcal{M}$.

Theorem 2.3 *Let $\mathcal{M}$ be a multitape Turing machine that computes a length-respecting function f in time T. Suppose in addition $\mathcal{M}$ has the property that the steps at which inputs are read or outputs produced depend only on the length n of the input. Then we can find a two tape oblivious Turing machine $\mathcal{M}'$ for f which runs in time $O(T(n)\log T(n))$.*

From Theorems 2.2 and 2.3 we immediately obtain a logical network of size $O(T(n)\log T(n))$ from a Turing machine which satisfies the restrictions of Theorem 2.3. In fact, we can obtain such a network from *any* Turing machine $\mathcal{M}$ which computes a length-respecting function.

Theorem 2.4 *Let $\mathcal{M}$ be a multitape Turing machine that computes a length-respecting function f and runs within time $T(n) > n$ for all inputs of length n. Then $C_f(n) = O(T(n)\log T(n))$.*

Proof: Let y be a string of the form $x01^t$ and $m = |y|$. We construct the program $\mathcal{M}'$ which does the following on input y:

1. It copies y onto work tape PI (pseudo-input), marking the end of x.

2. It behaves like $\mathcal{M}$ for exactly m steps, except that all input comes from the work tape PI, and all output goes on the work tape PO (pseudo-output).

3. It prints out the first m symbols of PO.

$\mathcal{M}'$ computes some function $g: K^* \to K^*$ such that $|x| = |g(x)|$ for all x and hence g is length-respecting.[1] Also, it is clear that $\mathcal{M}'$ satisfies the conditions of Theorem 2.3, and the running time of $\mathcal{M}'$ is $O(m)$. Now, applying Theorems 2.2 and 2.3 to $\mathcal{M}'$ gives a network N_m for $g|K^m$ of size $O(m \log m)$.

We obtain our network for $f|K^N$ as follows. Let $m = T(n)$ and construct the network N_m. By construction, g has the property that $f(x)$ is a prefix of $g(x01^t)$ for all t such that $|x01^t| \geq T(|x|)$. Hence, by setting the last $m - n$ inputs of N_m to the constants 01^{m-n-1} and ignoring all but the first $|f(0^n)|$ outputs, we obtain a new network which computes $f|K^n$. Its size is $O(m \log m) = O(T(n) \log T(n))$. ∎

It remains to sketch the proof of Theorem 2.3.

Proof (sketch): Replace each tape of $\mathcal{M}$ by 2 pushdown stores. A *pushdown store* is a tape with a restricted set of operations:

- PUSH(a) means "shift right and print a".
- POP means "print a blank and shift left".

Thus, the head of a pushdown store always sits on the rightmost non-blank square, which is commonly called the *top* of the store.

The master tape of $\mathcal{M}'$ has one *channel* for each pushdown store of $\mathcal{M}$. Each channel is divided into three *tracks*. Lengthwise, we divide the channels and tracks into *segments*. Segment i consists of squares $2^{i+1} - 1$ through 2^i, $i \geq 0$, where we begin numbering the tape squares from 1. These definitions are illustrated in Figure 4.

The portion of a track which lies within a given segment is called a *block*. A given block is either completely full or completely empty (denoted by ϕ). The contents of a simulated pushdown store is obtained by concatenating together the full blocks of the corresponding channel in order, beginning with the smallest numbered segments, and within a segment, the blocks are taken in track order. For example, Channel 1 of Figure 4 encodes the pushdown store: "abcdefghi", where "a" is at the top of the store.

From now on, we restrict attention to a single channel, for all channels are handled in parallel and in the same manner. We say that a segment is *clean* (for a given channel) if the number of its filled blocks is 1 or 2;

[1] To make this strictly true, one has to worry about what $\mathcal{M}'$ does on inputs in 1^*. We can assume it does the same thing on 1^t as it does on 01^{t-1}.

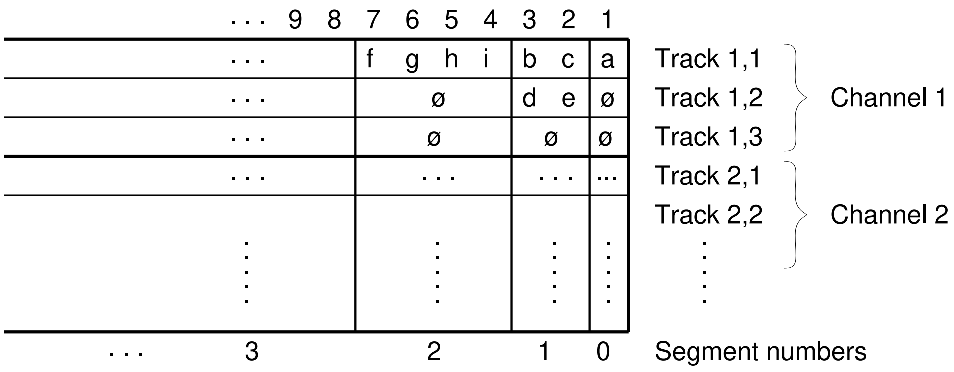


Figure 4: Master tape.

otherwise it is *dirty* and the number of filled blocks is 0 or 3. We assume initially that the first track is entirely filled with blanks and the remaining two tracks are empty; hence every segment is initially clean.

Claim *Assume segments 0 through n are clean. Then* $\mathrm{Sim}(n)$ *will simulated* $\mathcal{M}$ *for 2^n steps, never move its head past the end of segment n, and upon completion, leave segments 0 through $n-1$ clean. The number of filled blocks in segment n is changed by at most ± 1.*

To $\mathrm{Sim}(n)$**:**

1. If $n = 0$, simulate one step of $\mathcal{M}$ by updating the contents of square 1.
2. If $n \geq 1$, do the following steps:
   (a) $\mathrm{Sim}(n-1)$;
   (b) $\mathrm{Clean}(n-1)$;
   (c) $\mathrm{Sim}(n-1)$;
   (d) $\mathrm{Clean}(n-1)$;

$\mathrm{Clean}(n-1)$ is a routine that makes segment $n-1$ clean by moving data to or from segment n as necessary. In particular, if segment $n-1$ is dirty because it has no filled blocks, then a block from segment n is fetched, split into two half-length blocks, and these two blocks are written in segment $n-1$. If segment $n-1$ is dirty because it contains 3 filled blocks, then two of them are combined and copied into a hole in segment n. Since we wish Clean to be oblivious, it must put its heads through the motions for both copy operations even though on any given call, at most one of them will be necessary.

To show that Sim works, it is necessary to check that the cleanliness condition is preserved. A difficulty would arise if steps (2b) and (2d) each, say, attempted to copy a block from segment n into segment $n-1$, for we are only assuming that segment n initially contains at least *one* full block, so step (2d) might then find segment n empty. However, the condition that the number of filled blocks in the last segment changes by at most ± 1 prevents this situation from ever occurring. The details are left for the reader.

The toplevel routine is the following:

1. $k = 0$
2. If $\mathcal{M}'$ has halted, then stop.
3. Sim(k)
4. Clean(k)
5. $k \leftarrow k + 1$
6. Go to step 2.

Toplevel is oblivious, and it runs in time

$$\sum_{k=0}^{\lceil \log T(n) \rceil} (\text{time of } \mathrm{Sim}(k) + O(2^k)) \;=\; O(T(n) \log T(n))$$

since

$$\text{time of } \mathrm{Sim}(k) \leq 2 \cdot (\text{time of } \mathrm{Sim}(k-1)) + O(2^k),$$

and thus

$$\text{time of } \mathrm{Sim}(k) = O(k 2^k).$$

This completes the proof of Theorem 2.3. ■

3 Application to Symmetric Transitive Closure

Let $A = (a_{i,j})$, $B = (b_{i,j})$ be matrices over K of dimension $m \times n$ and $n \times p$ respectively. Define $C = (c_{i,j}) = A \binom{\vee}{\wedge} B$, where

$$c_{i,j} = \bigvee_{k=1}^{n} (a_{i,k} \wedge b_{k,j})$$

and C is of dimension $m \times p$. C is sometimes called the *Boolean product* of A and B.

Let M be an $n \times n$ Boolean matrix. The *transitive closure* M^* of M is $\bigvee_{k \geq 0} M^k$, where

$$M^0 \stackrel{\mathrm{df}}{=} I = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

and $M^{k+1} = M^k \binom{\vee}{\wedge} M$.

If G is a directed graph with n nodes $\{1, \ldots, n\}$, one can describe G by the $n \times n$ Boolean matrix M, where $M_{i,j} = 1$ iff $i \rightarrow_G j$ (i.e., there is an edge from i to j in G). Then M^* describes the transitive closure G^* of G, where $i \rightarrow_{G^*} j$ iff $i = j$ or there is a path from i to j in G.

Fact (Fischer and Meyer [6], Munro [14]). $\binom{\vee}{\wedge}$ matrix product and transitive closure can be done in essentially the same time, that is, a fast algorithm for one implies the existence of a fast algorithm for the other, and both can be done in time $O(n^{\log_2 7} \cdot (\log n)^{1+\varepsilon})$ for any $\varepsilon > 0$ using the fast matrix product method of Strassen [24] and the fast integer arithmetic of Schönhage and Strassen [22].

In case G is undirected, M becomes symmetric, and the above fact, restricted to symmetric matrices, is *not* known to hold. In fact, symmetric transitive closure is the same problem as finding the connected components of a graph and can be done in time $O(n^2)$ by the algorithm below.

Note added in revision: The algorithm below is known in the folklore. The remaining results in this section are due to Fischer, Paterson, and Pippenger [8] and have never been published.

Let

$$\mathrm{next}(i) = \begin{cases} \text{least } j > i \text{ s.t. } M_{i,j} = 1 \\ 0 \text{ if no such } j \end{cases}$$

To compute M^*:

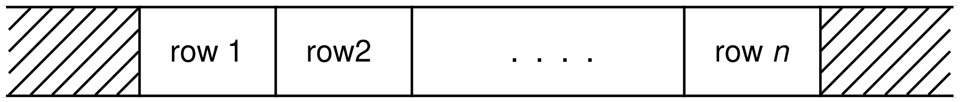
1. for $i \leftarrow 1$ to n
2. $\quad$ [$j \leftarrow \mathrm{next}(i)$;
3. $\quad\quad$ if $j \neq 0$ then Row $j \leftarrow$ Row $j \vee$ Row i

4. else [$M_{i,i} \leftarrow 1$;
5. for $k \leftarrow 1$ to $i-1$
6. if $M_{i,k} = 1$ then Row $k \leftarrow$ Row i]].

Only $O(n)$ row operations are performed since the outer loop is executed only n times, and each row is copied into at most once (in step 6). We leave the (nontrivial) proof of correctness to the reader.

We bound from above the combinational complexity of this problem by constructing a Turing machine and applying Theorem 2.4.

A naive implementation takes $O(n^3)$ steps. That is, if one represents M "by rows" on a tape



a single row operation may take time $O(n^2)$ because of the time to position the head(s).

We consider the mail carrier problem. The houses, numbered $1, \ldots, n$, are visited in turn. At each house, the postman delivers all the letters addressed to that house and then picks up some number of new letters to be sent to houses further on down the route (i.e., with higher numbers). The postman carries a fixed number of mailbags which he must use as pushdown stacks. Thus, to get at a letter at the bottom of a bag, he must remove the other letters one at a time and place them into other bags. We wish to minimize the total number of times a letter is handled.

Theorem 3.1 *The mail carrier problem can be solved so that each letter is handled $O(\log n)$ times, where n is the number of houses.*

Proof: Let L be a set of letters addressed to houses $i, i+1, \ldots, i+n-1$. "Deliver" visits the n houses $i, i+1, \ldots, i+n-1$, delivers the mail in L together with any new mail destined for them, and returns the new mail for houses $\geq i+n$.

To Deliver(L, i, n):

1. If $n = 1$, deliver L to house i and return { new letter }.
2. else[Partition L into

$$L_1 = \{x \in L \mid x < i + \lfloor n/2 \rfloor\}$$
$$L_2 = \{x \in L \mid x \geq i + \lfloor n/2 \rfloor\}$$

3. $M \leftarrow$ Deliver$(L_1,\ i,\ \lfloor n/2 \rfloor)$
4. Partition M into
$M_1 = \{x \in M \mid i + \lfloor n/2 \rfloor \leq x < i + n\}$
$M_2 = \{x \in M \mid x \geq i + n\}$
5. $N \leftarrow$ Deliver$(L_2 \cup M_1,\ i + \lfloor n/2 \rfloor,\ \lceil n/2 \rceil)$
6. Return $N \cup M_2$.]

To count the number of times a letter is handled, consider the tree of recursive calls on deliver shown in Figure 5. Each node is labeled with

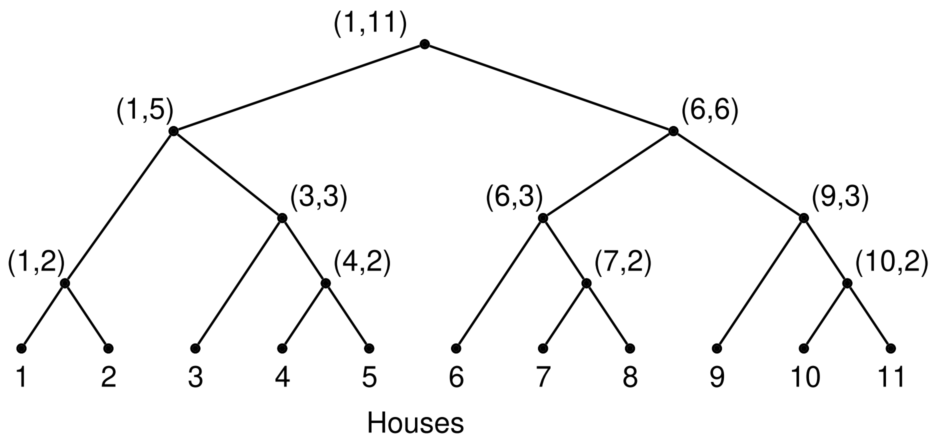


Figure 5: Mail carrier delivery tree.

the pair (i, n) giving the first house and number of houses visited by that recursive call.

A letter from house r to house s is handled once at each call (node) on the unique path through the tree from r to s. Since the depth of the tree is at most $\lceil \log n \rceil$, this path has length $\leq 2\lceil \log n \rceil$. ■

In the application to symmetric transitive closure, each "letter" is a row i of the matrix at stage i to be carried to row j and *or*'ed into it (line 3 of the algorithm). The correctness of the algorithm does not depend on row i being delivered immediately—it is only necessary that all rows sent to row j be delivered before row j itself is processed at stage j. It is also not necessary to perform steps 5 and 6 of the transitive closure algorithm to determine the connected components, for each component is described by row i (after $M_{i,i}$ has been set to 1) when next(i) is 0. After all the components have been determined, then M^* can be constructed by performing steps 5 and 6 on a second pass, using "Deliver" from right to left. This yields:

Theorem 3.2 *The transitive closure of an $n \times n$ symmetric Boolean matrix may be computed on a Turing machine in time $O(n^2 \log n)$.*

Corollary 3.3 *The combinational complexity of symmetric transitive closure is $\leq O(n^2 \log^2 n)$.*

4 Inversion Complexity

Let $\Omega = \{\wedge, \vee, \neg\}$. Let $I(F)$ be the minimum number of negations ($\neg$) in any network over Ω which computes the set of functions F. $I(F)$ is called the *inversion complexity* of F.

Let $|F| = m$ and treat F as a function $K^n \to K^m$. Order Boolean vectors by $(x_1, \ldots, x_r) \leq (y_1, \ldots, y_r)$ iff $x_i \leq y_i$ for all i, $1 \leq i \leq r$.

A sequence of n-vectors $C = (X_1, \ldots, X_k)$ such that $X_i \leq X_{i+1}$, $1 \leq i < k$, is called a *chain* of length k. For such a chain, define

$$\mathrm{alt}_F(C) = \#\{\, i \mid 1 \leq i < k \;\&\; F(X_i) \not\leq F(X_{i+1}) \,\}.$$

Let $A(F) = \max_{\text{chains } C} \mathrm{alt}_F(C)$.

Theorem 4.1 *(Markov [13]) $I(F) = \lceil \log_2(A(F) + 1) \rceil$.*

Proof: We first show $A(F) \leq 2^{I(F)} - 1$ by induction on $I(F)$.
Base: $I(F) = 0$. Then F is monotone, so $A(F) = 0$.
Induction: Suppose $A(F') \leq 2^{I(F')} - 1$ for all $F' \in A_n$ such that $I(F') < I(F)$. In any network for F, there is a $\neg$-gate whose input does not depend on the outputs of any other $\neg$-gates, so F may be decomposed as

$$F(x_1, \ldots, x_n) = G(\neg h(x_1, \ldots, x_n), x_1, \ldots, x_n)$$

where $I(G) = I(F) - 1$ and $I(h) = 0$.

Let $C = (X_1, \ldots, X_k)$ be a chain. By monotonicity of h, there is an r, $0 \leq r \leq k$ such that $h(X_i) = 0$ for $1 \leq i \leq r$ and $h(X_i) = 1$ for $r+1 \leq i \leq k$. Let

$$C_0 = (X_1, \ldots, X_r)$$
$$C_1 = (X_{r+1}, \ldots, X_k).$$

Now let $G_i(x_1,\ldots,x_n) = G(\neg i, x_1,\ldots,x_n)$, $i \in \{0,1\}$. Then $I(G_i) \leq I(G) = I(F) - 1$, so by the induction hypothesis, $\mathrm{alt}_{G_i}(D) \leq 2^{I(G_i)} - 1$ for any chain D. In particular,

$$\mathrm{alt}_F(C_0) = \mathrm{alt}_{G_0}(C_0) \leq 2^{I(G_0)} - 1 \leq 2^{I(F)-1} - 1$$

and

$$\mathrm{alt}_F(C_1) = \mathrm{alt}_{G_1}(C_1) \leq 2^{I(G_1)} - 1 \leq 2^{I(F)-1} - 1.$$

It follows that

$$\mathrm{alt}_F(C) \leq \mathrm{alt}_F(C_0) + \mathrm{alt}_F(C_1) + 1 \leq 2^{I(F)} - 1.$$

Hence, $I(F) \geq \lceil \log(A(F)+1) \rceil$.

We now show $I(F) \leq \mathrm{LA}(F)$ by induction on $\mathrm{LA}(F) = \lceil \log_2(A(F)+1) \rceil$.

Base: $\mathrm{LA}(F) = 0$. Then $A(F) = 0$, so F is monotone and $I(F) = 0$.

Induction: Suppose $I(F') \leq LA(F')$ for all F' such that $\mathrm{LA}(F') < LA(F)$.

For a chain $C = (X_1,\ldots,X_k)$, define $\mathrm{first}(C) = X_1$ and $\mathrm{last}(C) = X_k$. If $\mathrm{last}(C) = \mathrm{first}(C')$, where $C' = (X'_1,\ldots,X'_\ell)$, define $C \circ C' = (X_1,\ldots,X_k,X'_2,\ldots,X'_\ell)$. Clearly $\mathrm{alt}_F(C \circ C') = \mathrm{alt}_F(C) + \mathrm{alt}_F(C')$.

Let

$$S = \{x \in K^n \mid \text{ for all chains } C \text{ with } \mathrm{first}(C) = x, \mathrm{alt}_F(C) < 2^{\mathrm{LA}(F)-1}\}.$$

Let c_S be the characteristic function of S, i.e., $c_S(x) = 1$ if $x \in S$ and $c_S(x) = 0$ if $x \notin S$. c_S is monotone since if $x \leq y$, then every chain C' with $\mathrm{first}(C') = y$ is a suffix of some chain C with $\mathrm{first}(C) = x$, and hence $x \in S \Rightarrow y \in S$.

Claim *Let $y \in K^n - S$. Then for all chains C with* $\mathrm{last}(C) = y$

$$\mathrm{alt}_F(C) < 2^{\mathrm{LA}(F)-1}.$$

Pf. Suppose there is a chain C with $\mathrm{last}(C) = y$ and $\mathrm{alt}_F(C) \geq 2^{\mathrm{LA}(F)-1}$. Since $y \notin S$, there is a chain C' with $\mathrm{first}(C') = y$ and $\mathrm{alt}_F(C') \geq 2^{\mathrm{LA}(F)-1}$. Then $\mathrm{alt}_F(C \circ C') \geq 2^{\mathrm{LA}(F)} \geq 2^{\log_2(A(F)+1)} > A(F)$, contradicting the definition of $A(F)$. $\square$

Now let

$$F_0 = \{f \vee c_S \mid f \in F\},$$

$$F_1 = \{f \wedge c_S \mid f \in F\}.$$

Regarded as functions $K^n \to K^m$, $F_0(x) = F(x)$ when $x \in K^n - S$ and $F_0(x) = (1, \ldots, 1)$ when $x \in S$. Similarly, $F_1(x) = (0, \ldots, 0)$ when $x \in K^n - S$ and $F_1(x) = F(x)$ when $x \in S$. Hence it follows that $A(F_i) \leq 2^{\mathrm{LA}(F)-1} - 1$, $i \in \{0, 1\}$, so

$$\begin{aligned} \mathrm{LA}(F_i) &= \lceil \log_2(A(F_i) + 1) \rceil \\ &\leq \mathrm{LA}(F) - 1 \\ &< \mathrm{LA}(F). \end{aligned}$$

By the induction hypothesis, $I(F_i) \leq \mathrm{LA}(F_i) \leq \mathrm{LA}(F) - 1$.

Let $G\colon K^{n+1} \to K^m$ be defined by

$$G(y, x_1, \ldots, x_n) = F_y(x_1, \ldots, x_n).$$

Then

$$F(x_1, \ldots, x_n) = G(c_S(x_1, \ldots, x_n), x_1, \ldots, x_n).$$

Our proof is completed by showing $I(G) \leq \max(I(F_0), I(F_1)) + 1$.

We show by induction on q that, for all n, m, if F_0 and F_1 are arbitrary functions in $K^n \to K^m$ with $I(F_i) \leq q$, $i \in \{0, 1\}$, then there exists a function M in $K^{n+2} \to K^m$ such that

$$M(y, \neg y, x_1, \ldots, x_n) = F_y(x_1, \ldots, x_n)$$

and $I(M) \leq q$.

Base: $q = 0$ Then F_i are monotone, so a network for M is given in Figure 6.

Induction: As before, we may decompose F_i as

$$F_i(x_1, \ldots, x_n) = F_i'(\neg h_i(x_1, \ldots, x_n), x_1, \ldots, x_n),$$

where h_i is monotone and $I(F_i') = I(F_i) - 1 \leq q - 1$. By induction, there is an M' such that

$$M'(y, \neg y, z, x_1, \ldots, x_n) = F_y'(z, x_1, \ldots, x_n)$$

and $I(M') \leq q - 1$. A network for M is given in Figure 7. Then $I(M) \leq 1 + I(M') \leq q$.

Finally we note that

$$G(y, x_1, \ldots, x_n) = M(y, \neg y, x_1, \ldots, x_n).$$

Then $I(F) \leq I(G) = 1 + I(M) \leq \max(I(F_0), I(F_1)) + 1 \leq \mathrm{LA}(F)$. ■

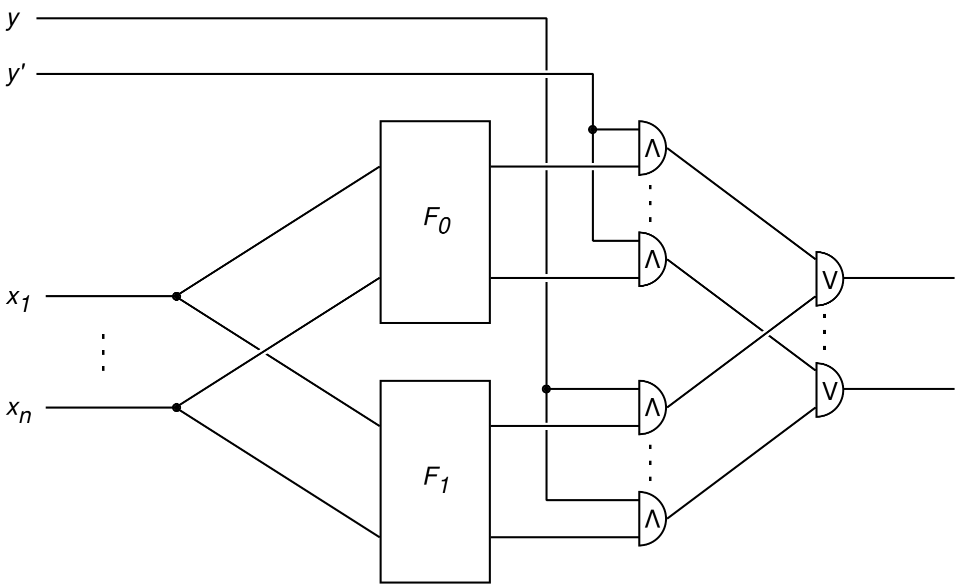


Figure 6: Network for M in base case.

Corollary 4.2 *(Markov [13])*

1. $\max_{F \subseteq A_n} I(F) = \lceil \log_2(n+1) \rceil$
2. $\max_{a \in A_n} I(a) = \lfloor \log_2(n+1) \rfloor$.

Proof: Consider the negations of the inputs $x_1, \ldots, x_n$:

$$F_n = \{\neg x_i \mid 1 \leq i \leq n\}.$$

For any maximal chain C, $\mathrm{alt}_F(C) = n$, so by Theorem 4.1, $I(F_n) = \lceil \log_2(n+1) \rceil$. Clearly $A(F) \leq n$ for all $F \subseteq A_n$, so equality 1 follows.

To prove 2, let

$$s(x_1, \ldots, x_n) \stackrel{\mathrm{df}}{=} \neg \bigoplus_{i=1}^{n} x_i = \begin{cases} 1 & \text{if } \sum_i^n x_i \equiv 0 \pmod 2; \\ 0 & \text{otherwise.} \end{cases}$$

Every maximal chain C is of length $n+1$, and

$$\mathrm{alt}_s(C) = \left\lfloor \frac{n+1}{2} \right\rfloor.$$

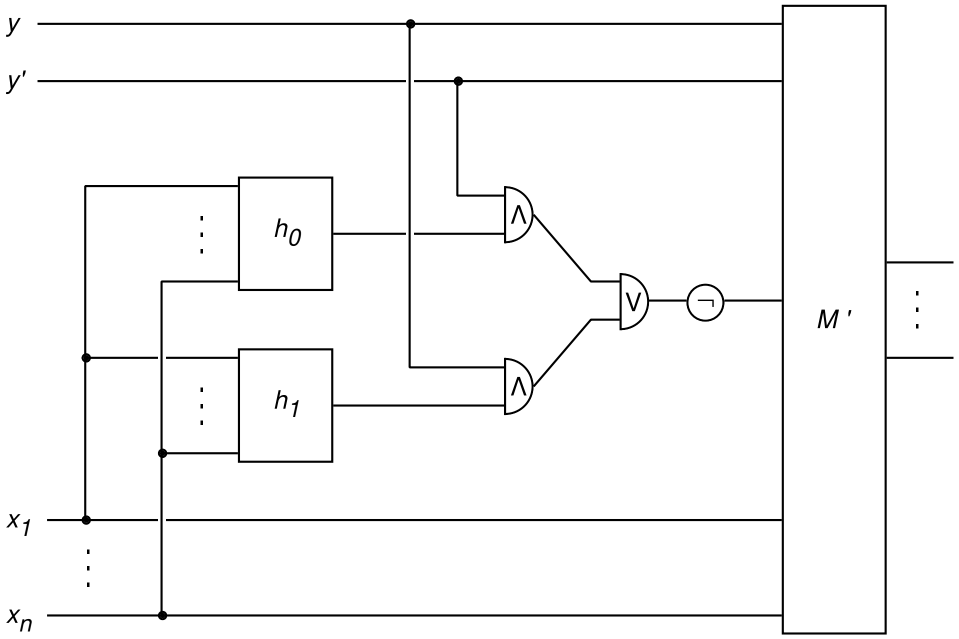


Figure 7: Network for M in recursive construction.

By Theorem 4.1,

$$I(s) = \left\lceil \log_2 \left(\left\lfloor \frac{n+1}{2} \right\rfloor + 1 \right) \right\rceil = \lfloor \log_2(n+1) \rfloor.$$

Clearly $A(a) \le \left\lfloor \frac{n+1}{2} \right\rfloor$ for all $a \in A_n$, so equality 2 follows. ■

Theorem 4.1 says nothing about the size network that might be needed to achieve the minimal numbers of negations. We now show that the number of inverters may always be reduced to $\lceil \log_2(n+1) \rceil$ with only a small increase in the size of the network. In what follows, we assume $\wedge$ and $\vee$ have equal cost c.

Theorem 4.3 *For all $F \subseteq A_n$, $I(F) \le \lceil \log_2(n+1) \rceil$. Moreover, there is a single network β for F such that $I(\beta) \le \lceil \log_2(n+1) \rceil$ and $L(\beta) \le 2L(F) + O(n^2 \log^2 n)$.*

Note added in revision: Beals, Nishino and Tanaka [3, 26] note in passing that this bound can be improved to $2L(F) + O(n^2 \log n)$ by using the sorting network of Ajtai, Komlós, and Szemerédi [1] instead of the Batcher network [2] in the proof of Theorem 4.5. The bound of that theorem then becomes $O(n^2 \log n)$. Still better bounds appear in [3, 26].

To prove Theorem 4.3 we need some additional concepts.

Definition: Let $F\colon K^n \to K^m$, $G\colon K^{2n} \to K^m$. G is a *monotone cover* of F if G is monotone and for all $x_1, \ldots, x_n \in K$,

$$F(x_1, \ldots, x_n) = G(x_1, \neg x_1, x_2, \neg x_2, \ldots, x_n, \neg x_n).$$

■

Theorem 4.4 *Let $F\colon K^n \to K^m$. Then there is a monotone cover G of F with $L(G) \leq 2L(F)$.*

Proof: By induction on the number of gates in a least cost network β for F.

Base: If β has no gates, then F is monotone, so G is trivially constructed.

Induction: Assume β has s gates. By choosing an initial gate of β, F may be decomposed in one of three ways:

1. $F(x_1, \ldots, x_n) = F'(x_i \vee x_j, x_1, \ldots, x_n)$,
2. $F(x_1, \ldots, x_n) = F'(x_i \wedge x_j, x_1, \ldots, x_n)$,
3. $F(x_1, \ldots, x_n) = F'(\neg x_i, x_1, \ldots, x_n)$

for some function F' of $n+1$ variables. A least cost network for F' exists with $s-1$ gates, and $L(F') = L(F) - c$ in cases 1 and 2, and $L(F') = L(F) - \text{cost}(\neg)$ in case 3.

By the induction hypothesis, there is a monotone cover

$$G'(y, y', x_1, x_1', \ldots, x_n, x_n')$$

for F' s.t. $L(G') \leq 2L(F')$. Define G according to the case:

1. $G(x_1, x_1', \ldots, x_n, x_n') = G'(x_i \vee x_j, x_i' \wedge x_j', x_1, x_1', \ldots, x_n, x_n')$,
2. $G(x_1, x_1', \ldots, x_n, x_n') = G'(x_i \wedge x_j, x_i' \vee x_j', x_1, x_1', \ldots, x_n, x_n')$,
3. $G(x_1, x_1', \ldots, x_n, x_n') = G'(x_i', x_i, x_1, x_1', \ldots, x_n, x_n')$.

It follows easily using DeMorgan's law that G is a monotone cover for F. Also, in cases 1 and 2,

$$L(G) \leq 2c + L(G') \leq 2c + 2L(F') = 2L(F).$$

and in case 3,

$$L(G) = L(G') \leq 2L(F') \leq 2L(F).$$

■

Theorem 4.5 *Let $F_n = \{\neg x_i \mid 1 \leq i \leq n\}$. Then $I(F_n) \leq \lceil \log_2(n+1) \rceil$. Moreover, there is a network β_n for F_n achieving this bound with $L(\beta_n) = O(n^2 \log^2 n)$.*

Proof: Let

$$\tau_i^k(x_1, \ldots, x_n) = \begin{cases} 1 \text{ if } \left(\sum_{\substack{j=1 \\ j \neq i}}^{n} x_j \right) \geq k; \\ 0 \text{ otherwise} \end{cases}$$

We use the fact that

$$\begin{aligned} \neg x_i = 1 \quad &\iff \quad x_i = 0 \\ &\iff \quad \forall k [\tau_0^k(\vec{x}) \rightarrow \tau_i^k(\vec{x})] \\ &\iff \quad \forall k [\neg \tau_0^k(\vec{x}) \vee \tau_i^k(\vec{x})]. \end{aligned}$$

Fact (Batcher [2]): For each i, $\{\tau_i^k \mid 1 \leq k \leq n\}$ can be be computed by a monotone network of size $O(n \log^2 n)$. (Ref: [11, pp. 220–235].)

Hence, $\{\tau_i^k \mid 1 \leq k \leq n, 0 \leq i \leq n\}$ can be computed by a network of size $O(n^2 \log^2 n)$. It remains to compute $T = \{\neg \tau_0^k \mid 1 \leq k \leq n\}$.

It suffices to find a network γ_n for the functions $\nu_k \colon K^n \rightarrow K$ such that

$$\nu_k(\tau_0^1(\vec{x}), \ldots, \tau_0^n(\vec{x})) = \neg \tau_0^k(\vec{x}),$$

that is, when the inputs are sorted, the network computes their complements.

Let $n = 2^r - 1$. We define γ_n inductively on r.

$r = 1$: γ_n is the network: $x_1 \longrightarrow \neg \longrightarrow \nu_1$

$r > 1$: γ_n is given in Figure 8. To see that it works we consider separately the two cases $x_{2^{r-1}} = 0$ and $x_{2^{r-1}} = 1$ and note that by our assumption that the inputs are sorted, $x_{2^{r-1}} = 0$ implies that $x_k = 0$ for all $k > 2^{r-1}$, and $x_{2^{r-1}} = 1$ implies $x_k = 1$ for all $k < 2^{r-1}$. The details are left to the reader. ■

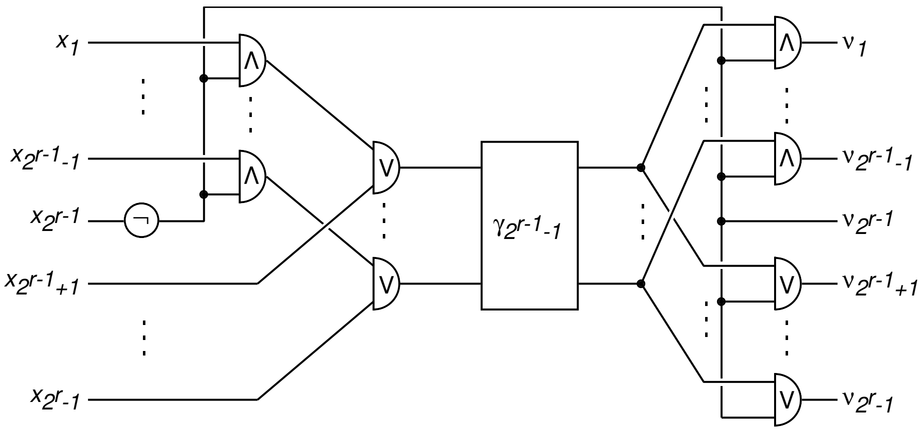


Figure 8: Recursive construction of γ_{2^r-1}.

Theorem 4.3 now follows by a direct application of Theorems 4.4 and 4.5.

Open problem: For F with $I(F) < \lceil \log_2(n+1) \rceil$, how much must the size of the network increase in order to achieve the minimal number of negations?

Acknowledgement

I would like to thank Albert Meyer, Mike Paterson, and Nick Pippenger for their many contributions to the work reported here. I am grateful to M.I.T., the University of Toronto, and the University of Frankfurt for providing me with a supportive environment in which to carry out the research that led to these notes, and to the University of Washington and Yale University for support in revising them and making them available to the community. I am especially grateful to Claus Schnorr for his encouragement and enthusiasm, and for making my visit to the University of Frankfurt in 1974 possible.

References

[1] M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 1–9, Boston, Massachusetts, April 1983.

[2] Kenneth E. Batcher. Sorting networks and their applications. In *Proc. AFIPS Spring Joint Computer Conf.*, volume 32, pages 307–314, Montvale, N. J., 1968. AFIPS Press.

[3] Robert Beals, Tetsuro Nishino, and Keisuke Tanaka. More on the complexity of negation-limited circuits. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, pages 585–595, Las Vegas, Nevada, May-June 1995.

[4] Andrzej Ehrenfeucht. Practical decidability. *Journal of Computer and System Sciences*, 11(3):392–396, December 1975.

[5] Michael J. Fischer. The complexity of negation-limited networks—a brief survey. In H. Brakhage, editor, *Automata Theory and Formal Languages, 2nd GI Conference*, volume 33 of *Lecture Notes in Computer Science*, pages 71–82. Springer-Verlag, 1975.

[6] Michael J. Fischer and Albert R. Meyer. Boolean matrix multiplication and transitive closure. In *Proc. 12th IEEE Symposium on Switching and Automata Theory*, pages 129–131, October 1971.

[7] Michael J. Fischer and Michael S. Paterson. Fishspear: A priority queue algorithm. *Journal of the ACM*, 41(1):3–30, January 1994.

[8] Michael J. Fischer, Michael S. Paterson, and Nicholas Pippenger. An efficient message-forwarding algorithm using sequential storage. Unpublished, August 1982.

[9] E. N. Gilbert. Lattice theoretic properties of frontal switching functions. *J. Mathematics and Physics*, 33:57–67, 1954.

[10] I. Ibaraki and S. Muroga. Synthesis of networks with a minimum number of negative gates. *IEEE Transactions on Computers*, C-20:49–58, January 1971.

[11] Donald E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, Mass., 1973.

[12] O. B. Lupanov. A method of circuit synthesis. *Izvestia v.u.z. Radiofizike*, 1:120–140, 1958.

[13] A. A. Markov. On the inversion complexity of a system of functions. *Journal of the ACM*, 5(4):331–334, October 1958. Translated by Morris D. Friedman.

[14] Ian Munro. Efficient determination of the transitive closure of a directed graph. *Information Processing Letters*, 1:56–58, 1971.

[15] K. Nakamura, N. Tokura, and T. Kasami. Minimal negative gate networks. *IEEE Transactions on Computers*, C-21(1):5–11, January 1972.

[16] Michael S. Paterson, editor. *Boolean Function Complexity*, volume 169 of *Lecture Note Series*, Cambridge, 1992. London Mathematical Society, Cambridge University Press.

[17] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *Journal of the ACM*, 26(2):361–381, April 1979.

[18] John E. Savage. Computational work and time on finite machines. *Journal of the ACM*, 19(4):660–674, October 1972.

[19] John E. Savage. *The Complexity of Computing.* John Wiley & Sons, New York, 1976.

[20] Claus P. Schnorr. Personal communication, 1974.

[21] Claus P. Schnorr. The network complexity and the Turing machine complexity of finite functions. *Acta Informatica*, 7(1):95–107, 1976.

[22] Arnold Schönhage and Volker Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7:281–292, 1971.

[23] Claude E. Shannon. The synthesis of two-terminal switching circuits. *Bell System Technical Journal*, 28:59–98, 1949.

[24] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

[25] Volker Strassen. Berechungen in partiellen Algebren endlichen Typs. *Computing*, 11:181–196, 1973.

[26] Keisuke Tanaka and Tetsuro Nishino. On the complexity of negation-limited Boolean networks (preliminary version). In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, pages 38–47, Montréal, Québec, Canada, May 1994.

[27] Ingo Wegener. *The Complexity of Boolean Functions.* Wiley-Teubner Series in Computer Science. B. G. Teubner and John Wiley & Sons, Stuttgart; Chichester; New York, 1987.