The Network-Complexity of Equivalence and Other Applications of the Network Complexity

(Extended Abstract)

C. P. Schnorr
Fachbereich Mathematik
Universität Frankfurt

1. Introduction

Let $B = \{0,1\}$ be the set of Boolean values, let $V = \{x_i | i \in N\}$ be a countable set of Boolean variables and let Ω be the set of Boolean polynomials with variables in V.

We consider Boolean computations (i.e. logical networks) that are based on the set of all 16 binary Boolean operations $\sigma : B^2 \to B$.

A <u>logical network</u> ß is a finite, directed, acyclic graph with labelled nodes such that

(1) every node ν has either 0 (i.e. ν is an entry) or 2 (i.e. ν is a non-entry) entering edges,

(2) every entry ν is labelled either with a variable or with a constant Boolean function,

(3) every non-entry ν is labelled with some binary Boolean operation $op(\nu)$ such that the entries of ν correspond to the arguments of $op(\nu)$.

In a natural way we associate with every node ν of β an output function $res_\beta^\nu \in \Omega$. We say β computes res_β^ν for all $\nu \in \beta$. Let cost(β) be the number of non-entries of β. We define the Network-Complexity of a set $F \subset \Omega$ of Boolean functions as

$$L(F) = \min \{\text{cost}(\beta) \mid \beta \text{ computes } F\}.$$

We believe that the network complexity is a natural measure for the complexity of Boolean functions. Observe that the asymptotical behaviour of the network complexity does not depend on the choice of the finite base of Boolean operations provided that this base is complete in the sense that every Boolean function can be computed from this base. The choice of the complete finite base only influences the network complexity up to a constant factor.

2. Network Complexity and Turing Machine Complexity

We compare the network complexity and the Turing Machine complexity of finite functions. In the following we consider programs on multitape Turing machines with binary input-output alphabet. In an efficient program p for a function f at least some of the following complexity measures should be rather small with respect to all other programs for f:

1) the time bound T_p of the program
2) the storage requirement S_p of the program, i.e. the total number of all tape squares that are handled by the heads during the computation on inputs of f.
3) the size $\|p\|$, i.e. the number of instructions of the program p.

Experience indicates that we cannot always minimize each of these measures by a unique program. So we expect some trade-off's between these measures. Much attention has been paid to the asymptotical behaviour of S_p and T_p for large inputs. However, the size of the program might also be considerably interesting for the computation of finite functions. It is usually rather hard to write and to check large programs. Therefore, a "table look up"-program for a finite function might be fast and might use little space but it might nevertheless be inefficient. In particular it might be very difficult to find such a "table look up"-program.

In the following we relate the above complexity measures for programs p to the network complexity of the Boolean functions that are computed by p. M. Fischer [2] proved that the network complexity L(f) of $f \in \Omega$ yields a lower bound on the product $c_p\ T_p \lg T_p$ for every Turing program p for f. However, his proof gives no information on the constant c_p that obviously depends on the program p. We improve this result by involving the size $\|p\|$ and the space requirements S_p of program p.

We shall also generalize previous results in that we consider Turing-Machines with oracles. Our concept of an <u>oracle-Turing-Machine</u> is as follows. There is an additional input tape with some finite or infinite inscription A. A is called the oracle. Relative to a fixed oracle A a Turing program acts like a standard Turing program. There are no special oracle-instructions. Let A be an oracle and let p be a Turing-program then we use the following notations:

$res_p^A : B^* \to B$ is the partial 0-1 valued function which is computed by program p with oracle A. Let $res_p^A(n)$ be the restriction $res_p^A|_{B^n}$.

$T_p^A(n)$ is the minimal running time of program p with oracle A on inputs $x \in B^n$

$S_p^A(n)$ is the total number of tape squares that are used in the computation of p on inputs $x \in B^n$

$\|p\|$ is the number of instructions of program p.

Theorem 1 [7]

$\exists c \in N: \forall \text{programs } p: \forall \text{oracles } A: \forall n:$

$L(res_p^A(n)) \leq c \cdot \|p\| \cdot T_p^A(n) \cdot \lg (S_p^A(n))$

Hereby c depends on the number of tapes and the size of the alphabet. c also depends on how the set of possible Turing-instructions is defined.

There is a converse to Theorem 1:

Theorem 2 [7]

$\forall f \in \Omega$ (depending on n variables): $\exists$ program p with oracle A for f:

$\|p\| \cdot T_p^A(n) \cdot \lg S_p^A(n) \leq O(L(f) \lg L(f))^2$

The complete proofs of Theorems 1,2 can be found in [7].

3. The Network Complexity of Equivalence

Consider the functions $and(n) = \bigwedge_{i=1}^{n} x_i$, $nor(n) = \bigwedge_{i=1}^{n} \neg x_i$,

$Eq(n) = and(n) \vee nor(n)$

Theorem 3

$L(Eq(n)) = 2n-3$

$L\{and(n), nor(n))\} = 2n-2 = L(and(n)) + L(nor(n))$

(i.e. and(n), nor(n) are independent).

One interesting feature of this theorem is that there exist many structurally different optimal computations for Eq(n) as for instance

$$\bigwedge_{i=1}^{n-1} [x_i = x_{i+1}] \quad , \quad \bigwedge_{i=1}^{n-1} [x_i = x_n]$$

We believe that in such a case there are particular difficulties to evaluate the exact value of the network complexity.

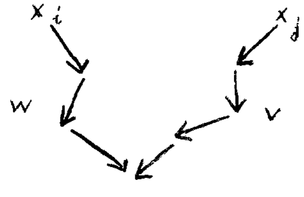
Theorem 3 also implies that the operations $\ominus$ and $\oplus$ do not help in the computation of {and(n), nor(n)} since $\bigwedge_{i=1}^{n} x_i$, $\bigwedge_{i=1}^{n} \neg x_i$ is an optimal computation.

The proof of Theorem 3 uses an inductive argument. The different cases of the induction step are covered by 3 Lemmata. The complete proof of these Lemmata will appear in [5].

<u>Lemma 1</u>

Let ß be an optimal computation for Eq(n). Suppose that there is a variable x_i in ß which is input to exactly one gate ν and this gate is either a $\ominus$ -gate or a $\oplus$ -gate. Then we can compute Eq(n-1) by fixing $res_{ß}^{\nu}$ either to 0 or to 1 and by eliminating at least 2 nodes in ß.

An (x_i, x_j)-<u>path</u> in a logical network is a pair of edge-disjoint paths (w,v) such that w starts at an x_i-variable and v starts at an x_j-variable and w,v have the same head:



The length of an (x_i, x_j)-path is the total number of edges. Let $\Gamma_\beta(x_i, x_j)$ be the minimal length of an (x_i, x_j)-path in ß.

Lemma 2

Suppose ß satisfies (1) - (3): (1) ß computes Eq(n), (2) there is a unique (x_i, x_j)-path in ß, (3) x_i, x_j are not entry of any $\ominus$ -gate and of any $\oplus$ -gate. Then there is a computation ß for Eq(n) which satisfies (1) - (3) such that $\Gamma_{\bar\beta}(x_i, x_j) < \Gamma_\beta(x_i, x_j)$ and $\text{cost}(\bar\beta) = \text{cost}(\beta)$.

Observe that the reduction

$$\beta \mapsto \bar\beta \mapsto \bar{\bar\beta} \mapsto \bar{\bar{\bar\beta}} \ldots\ldots$$

according to Lemma 2 can only be applied finitely often since each step reduces $\Gamma_\beta(x_i, x_j)$.

Lemma 3

Let ß be a Boolean computation depending on the variables in V(ß). Suppose that for all $x_i, x_j \in V(\beta)$ there exist at least 2 different (x_i, x_j)-paths. This implies $\text{cost}(\beta) \geq 2 \cdot \|V(\beta)\| - 2$.

It can easily be seen that Lemmata 1-3 cover all cases of an inductive proof for L(Eq(n)) = 2n-3. The same kind of arguments can be used to prove L{and(n),nor(n)} = 2n-2.

The Boolean functions and(n),nor(n) are <u>independent</u> in the sense that

$$L\{\text{and}(n), \text{nor}(n)\} = L(\text{and}(n)) + L(\text{nor}(n))$$

and we conjecture that this independence holds for any choice of a complete base of Boolean operations. Independence seems to be a basic notion of complexity theory. It should be observed that many fast algorithms which improve standard algorithms are based on hidden dependencies of certain functions. For example, Strassen's fast matrix multiplication yields a particularly interesting example of such a hidden dependence:

Theorem 4

There exist sets $F,G \subset \Omega$ of Boolean functions such that (1) F,G depend on disjoint sets of variables and (2) $L(F \cup G) < L(F) + L(G)$.

Proof Let A_n be an (n,n)-Boolean matrix and let x be a vector of n Boolean variables. We associate with A_n and x the Boolean function $\tilde{A}_n : B^n \to B$ as follows: $\tilde{A}_n : x \mapsto A_n \cdot x$ where $A_n \cdot x$ is the Boolean matrix product with respect to addition mod(2) and multiplication mod(2). There exist 2^{n^2} different Boolean (n,n)-matrices. Therefore, a standard counting argument implies that for all n there exists A_n such that $L(A_n) \geq c\, n^2/\lg n$ where $c > 0$ is some fixed real number. Let $x^1, x^2, \ldots, x^n$ be a set of n-vectors that consist of disjoint sets of Boolean variables. Let $\tilde{A}_n^i$ be the function $\tilde{A}_n^i : x^i \mapsto A_n \cdot x^i$. If Theorem 4 does not hold then

$$L(A_n^1, A_n^2, \ldots, A_n^n) \geq c\, n^3 \lg n$$

However, $A_n x^1, A_n x^2, \ldots, A_n x^n$ is the matrix product of A_n and the matrix with column vectors $x^1, \ldots, x^n$. Therefore, Strassen's fast algorithm for matrix multiplication yields

$$L(A_n^1, \ldots, A_n^n) \leq O(n^{\lg 7})$$

This proves Theorem 4 by contradiction.

4. Satisfiability is quasi-linear complete in NQL

A fundamental problem of computer science is the power of non-deterministic machines. Cook raised the question whether the classes P (NP,resp.) of all decision problems that can be solved within polynomially bounded time on deterministic (non-deterministic, resp.) Turing-machines coincide. Cook proved that Satisfiability (i.e. the problem to decide whether a given conjunctive Boolean normal form is satisfiable) is polynomial complete in NP. This means that Satisfiability is in NP and for every problem A in NP there is a polynomially time bounded reduction of A to Satisfiability. Cook, Karp and others established a long list of polynomial complete problems in NP.

These results are improved by the following Theorem 5 which shows that Satisfiability is one of the most hardest polynomial complete problems in NP. In order to formulate this result we restrict the class of polynomially time bounded reductions.

A time bound T(n) is called quasi-linear in n if $T(n) = O(n(\lg n)^k)$ for some fixed k. Consider the following classes of functions and problems.

Let QL^f be the class of functions that are computable on Turing machines in quasi-linear time. Let QL be the class of decision problems that can be solved on Turing-machines in deterministid quasi-linear time.

Let NQL be the class of decision problems that can be solved on Turing machines in non-deterministic quasi-linear time.

Then we can prove that Satisfiybility is quasi-linear complete in NQL:

Theorem 5 [8]

(1) Satisfiability is in NQL

(2) $\forall A \in NQL : \exists \psi \in QL^f :$

$[\forall x : x \in A : \Longleftrightarrow \psi(x)$ is satisfiable$]$

(i.e. ψ is a quasi-linear reduction of A to Satisfiability)

An immediate consequence of Theorem 5 is the following

Corollary QL = NQL $\Longleftrightarrow$ Satisfiability $\in$ QL

The main step in the proof of part (2) in Theorem 5 is an application of the simulation of Turing-machines by logical networks according to Theorem 1.

The question whether QL = NQL? is very much like the famous P = NP?-problem. However, a proof for QL $\neq$ NQL seems to be not as hard as for P $\neq$ NP. Using Theorem 1 it would satisfy to prove a slightly higher than quasi-linear lower bound on the network complexity of Satisfiability. However, a proof for P $\neq$ NP by this way requires a superpolynomial lower bound on the network complexity of Satisfiability.

References

1. Cook, S.A.: The Complexity of Theorem-Proving Procedures. Symposium on Theory of Computing 1971. 151-158

2. Fischer, M.J.: Lectures on Network Complexity. Preprint Universität Frankfurt, 1974

3. Karp, R.M.: Reducibility among Combinatorial Problems. in: Complexity of Computer Computations. R.E. Miller and J.W. Thatcher, Eds., Plenum Press, New York (1972) 85-1o4

4. Paul, W.J.: 2.25N - Lower Bound on the Combinational Complexity of Boolean Functions. Symposium on Theory of Computing, 1975

5. Schnorr, C.P.: The Combinational Complexity of Equivalence. Preprint Universität Frankfurt 1975, to appear in Theoretical Computer Science

6. Schnorr, C.P.: Zwei lineare untere Schranken für die Komplexität Boolescher Funktionen. Computing 13, (1974)

7. Schnorr, C.P.: The Network Complexity and the Turing Machine Complexity of Finite Functions. Preprint Universität Frankfurt, 1975

8. Schnorr, C.P.: Satisfiability is Quasi-Linear Complete in NQL. Preprint Universität Frankfurt, 1975.