Etudes on the methods of designing efficient circuits

Igor S. Sergeev[1]

preliminary version 7e — Febrary 21, 2025

[1]e-mail: isserg@gmail.com

Preface

Fast computations is one of the most significant areas attracting attention of mathematicians. In the modern world, the realm of various electronic forms and artificial intelligence, almost all aspects of life are penetrated with the fruits of this theory.

The goal of the present notes is to provide a systematic view of the most fruitful ideas leading to fast computation methods. The exposition is arranged around the computational model of circuits of functional elements and its particular case, formulae. This is done deliberately: on the one hand, so as not to overload the conceptual apparatus, on the other — so as not to try to embrace the immensity. Though, it slightly limits the choice of applications for illustrating ideas.

The proposed examples cover mainly boolean and arithmetic (algebraic) computations as the most practically demanded and intensively studied by complexity theory. It is worth noting that popular, classical monographs on the complexity of boolean functions [331, 82, 230, 139] were written with an emphasis on lower complexity bounds. Fast computation methods are given a secondary role in them, mainly to demonstrate the accuracy of lower bounds. The present work partly fills this gap.

Algebraic algorithms, which usually have a direct connection to applications, receive more attention. Perhaps, the most thorough exposition of the modern theory of fast algebraic algorithms is provided by [107]. Polynomial and matrix operations are given more attention in [34, 238]. A nice introduction to the theory of numeric algorithms are the books [167, 51]. This work includes only a few fragments of the theory, demonstrating the diversity of computational techniques.

Indeed, the choice of etudes for the book was dictated by the author's tastes. In addition to the widely known classical methods of synthesis, the book provides some results of the last 10–20 years. The author hopes that in several cases he has managed to offer more transparent proofs than in the original works and other sources. The theory of fast computations continues to develop, although each new step is becoming increasingly difficult.

Igor S. Sergeev
Skhodnya/Moscow,
April 2022.

Contents

Basic notions

The basic computational model we consider is *circuits of functional elements* (hereinafter, simply *circuits*). A *circuit over a basis* (set of functions) $\mathcal{B}$ is a directed acyclic graph, in which vertices with no incoming edges are marked as inputs, and some vertices are marked as outputs.

Inputs are associated with variables or constants of the basis $\mathcal{B}$, other vertices (called *functional elements* or *gates*) are associated with functions of the basis $\mathcal{B}$. The functioning of the circuit is defined in a natural way, from inputs to outputs: at each node, the function associated with it is applied, the arguments of which are functions arriving along the edges entering the node.

A circuit *implements* an operator F if all components of the operator are computed at the circuit outputs. Fig. 1a) shows a circuit computing the arithmetic sum of three bits $2y_2 + y_1 = x_1 + x_2 + x_3$ according to the rules[1)] $y_1 = x_1 \oplus x_2 \oplus x_3$, $y_2 = x_1(x_2 \oplus x_3) \oplus x_2x_3$. In the most common situation, a circuit has a single output and implements some function.

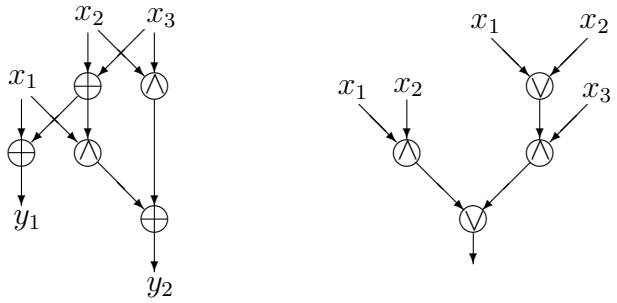


Figure 1: Examples of a circuit (a) and of a formula (b)

The *circuit complexity* is defined as the number of non-input nodes in its graph. The *complexity of an operator* F when implemented by circuits over a basis $\mathcal{B}$ is defined as the minimal complexity of a circuit implementing it and is denoted by $\mathsf{C}_{\mathcal{B}}(F)$.

Circuit depth is the length (measured in edges or functional elements) of the longest directed path connecting an input and an output of the circuit. By analogy, *depth of an operator* F is defined as the minimum depth of a circuit implementing it, and is denoted by $\mathsf{D}_{\mathcal{B}}(F)$. The complexity and the depth of a class (i.e. set) of operators $\mathcal{F}$ are defined as $\mathsf{C}_{\mathcal{B}}(\mathcal{F}) = \max_{F\in\mathcal{F}} \mathsf{C}_{\mathcal{B}}(F)$ and $\mathsf{D}_{\mathcal{B}}(\mathcal{F}) = \max_{F\in\mathcal{F}} \mathsf{D}_{\mathcal{B}}(F)$.

It is known that the complexity of computing an operator in any complete finite boolean basis is the same up to an order of magnitude.

Circuits over the basis of a single semigroup operation $\{+\}$ are called *additive circuits*. In view of the special role of additive circuits in synthesis theory, for the complexity of the implementation of a linear operator L_A with a matrix A by such

[1]Here and below, the conjunction symbol will be omitted, as is usual for multiplicative operations.

circuits, we use a special notation $\mathsf{L}_+(A)$ instead of $\mathsf{C}_{\{+\}}(L_A)$; in this case, it is convenient to say that the circuit computes the matrix A itself. Let $\mathsf{L}(A)$ also denote the complexity of computation by universal additive circuits, i.e., those that correctly compute the matrix A in any commutative semigroup.

Formula is a special case of a circuit in which branching of outputs of elements is prohibited: no more than one edge goes out of any vertex of the graph. By *formula complexity* we usually mean the number of inputs of variables[2)]. Fig. 1b) shows an example of a formula that computes the majority function $\mathrm{maj}_3(x_1, x_2, x_3) = (x_1 \vee x_2)x_3 \vee x_1x_2$.

A formula can also be interpreted as an expression that can be written in one line (hence the name of the term). The following inductive formal definition is more suitable for such an interpretation. *Formula over a basis* $\mathcal{B}$, *formula complexity*, *formula depth*, and the *function computed by the formula* are defined as follows: 0) the basis constants are formulae of complexity and depth 0; 1) the symbols of variables are formulae of complexity 1, depth 0, and implement the corresponding identity functions; 2) the expression $G(F_1, ..., F_k)$, where G is a symbol denoting a non-constant k-input function $g \in \mathcal{B}$, and F_i is a formula of complexity L_i and depth D_i, implementing a function f_i, is a formula of complexity $L_1 + \ldots + L_k$, depth $\max\{D_1, ..., D_k\} + 1$ and implements the function $g(f_1, \ldots, f_k)$. In the binary case ($k = 2$), it is customary to use symbols of binary operations to write formulae: instead of $G(F_1, F_2)$, we write $F_1 \circ F_2$, where $g = x \circ y$.

In the above definition, $F_1, \ldots, F_k$ are called *principal subformulas* of the formula $G(F_1, ..., F_k)$.

Complexity of an operator F when implemented by formulae over the basis $\mathcal{B}$ (independently of the way of definition) will be denoted by $\Phi_{\mathcal{B}}(F)$. By analogy with circuits, the notation $\Phi_{\mathcal{B}}(\mathcal{F})$ is introduced for the formula complexity of a class of operators $\mathcal{F}$. The depth of implementation of any operator by circuits and formulae over the same basis coincides, so we use a single notation $\mathsf{D}_{\mathcal{B}}$. When studying the depth of computations, it is often convenient to consider formulae as a topologically simpler object compared to circuits.

An important subclass of formulae are *read-once formulae*. A formula is read-once if the symbol of each variable occurs no more than once in it. Functions computed by read-once formulae over a basis $\mathcal{B}$ are also called read-once or, more precisely, read-once expressible over the basis $\mathcal{B}$.

The most popular boolean bases are: the standard basis $\mathcal{B}_0 = \{\vee, \wedge, {}^-\}$, Zhegalkin basis $\mathcal{B}_1 = \{\oplus, \wedge, 1\}$, the monotone basis $\mathcal{B}_M = \{\vee, \wedge\}$, basis $\mathcal{B}_2$ of all two-input boolean functions (binary basis), the unate basis $\mathcal{U}_2 = \mathcal{B}_2 \setminus \{\oplus, \sim\}$. Here "$\sim$" means the boolean equivalence operation.

The main arithmetic bases for computations over a semiring R are: the complete basis $\mathcal{A}^R = \{+, -, *\} \cup \{ax | a \in R\}$, the linear basis $\mathcal{A}^R_L = \{+, -\} \cup \{ax | a \in R\}$, the monotone basis $\mathcal{A}^R_+ = \{+, *\}$, the complete basis with division $\mathcal{A}^R_D = \{+, -, *, /\} \cup \{ax | a \in R\}$ (in the case of a basis with division, it is assumed that R is a ring).

[2]But this is not very important, since in a formula over a finite basis the number of inputs and the number of functional elements are of the same order of magnitude.

Considerations

To simplify the notation of expressions, we will omit the basis notation in complexity functionals (say, write $\mathsf{C}(F)$ instead of $\mathsf{C}_{\mathcal{B}}(F)$) in cases where the basis is clear from the context, for example, within the proof of assertions.

Similarly, in the notations of arithmetic bases and operators we will omit references to the semiring R over which calculations are performed (for example, write $\mathcal{A}$ instead of $\mathcal{A}^R$ or M_n instead of M_n^R) when this is clear from the context.

To compare the orders of growth of nonnegative functions, we use standard notations: $f \prec g$ is equivalent to $f = o(g)$; $f \preccurlyeq g$ is equivalent to $f = O(g)$; $f \succ g$ is equivalent to $f = \omega(g)$; $f \succcurlyeq g$ is equivalent to $f = \Omega(g)$; $f \asymp g$ is equivalent to $f = \Theta(g)$; $f \sim g$, $f \lesssim g$, $f \gtrsim g$ denote asymptotic equality and inequalities.

Further, X (analogously, Y, Z) will usually mean a set of variables x_i, possibly organized as a matrix $(x_{i,j})$.

Notation

$\mathbb{B}$ — boolean set $\{0, 1\}$
$\mathbb{N}$ — natural numbers $1, 2, 3, \ldots$
$\mathbb{N}_0$ — nonnegative integer numbers $0, 1, 2, \ldots$
$\mathbb{P}$ — prime numbers
$[\![n]\!]$ — set $\{0, 1, \ldots, n-1\}$
C_n^k — binomial coefficient, often denoted as $\binom{n}{k}$
$\mathcal{P}^n$ — class of boolean functions of n variables
$\mathcal{M}^n$ — class of monotone boolean functions of n variables
$\mathcal{S}^n$ — class of symmetric boolean functions of n variables
$\mathbf{P}(A)$ — probability of an event A
$\mathbf{E}[A]$ — mathematical expectation of an event A
$\|X\|$ — weight of a boolean vector (number of ones)
$|A|$ — weight of a matrix A (number of nonzero entries)
$\det A$ — determinant of a matrix A
$u \odot v$ — componentwise product of vectors u and v
$A \otimes B$ — Kronecker product of matrices A and B 59
$T_1 \otimes T_2$ — tensor product of systems of bilinear forms T_1 and T_2 59
$T_1 \oplus T_2$ — direct sum of systems of bilinear forms T_1 and T_2 117
$\mathsf{C}_{\mathcal{B}}(F)$ — complexity of implementing an operator F by circuits over a basis $\mathcal{B}$ 6
$\Phi_{\mathcal{B}}(F)$ — complexity of implementing an operator F by formulae over a basis $\mathcal{B}$ 7
$\mathsf{D}_{\mathcal{B}}(F)$ — circuit (formula) depth of an operator F over a basis $\mathcal{B}$ 6
$\mathsf{C}(F), \Phi(F)$ — complexity of an operator F when implemented by circuits/formulae over an arbitrary complete boolean basis 8
$\mathsf{L}_+(A)$ — complexity of the implementation of a matrix A by additive circuits over the basis $\{+\}$ 7
$\mathsf{L}(A)$ — minimal complexity of a universal additive circuit for a matrix A 7
$\mathsf{C}_d^{AC}(F)$ — complexity of implementing an operator F by AC-circuits of depth d 141

MM_n^R — operator of multiplication of $n \times n$ matrices over a semiring R 24
$MM_{m,n,p}^R$ — operator of multiplication of $m \times n$ and $n \times p$ matrices over R 59
MOD_n^m — operator of summation of n variables modulo m 43
$\mathrm{MOD}_n^{m,r}$ — indicator function of equality of the sum of n variables to a number r modulo m 43
$\mu_n(X;Y)$ — order n multiplexor function in 2^n data variables y_i: it takes a value y_X 41
$\mathrm{GCD}_n(a,b)$ (or $\mathrm{GCD}(a,b)$) — greatest common divisor of n-bit numbers or of polynomials a and b of degree $< n$ 13 30
$QR_{n,m}, QR_{n,m}^R$ — operators of division with remainder: of an n-bit number by an m-bit number, and of a polynomial of degree $< n$ by a polynomial of degree m over R 30 54
$PERM_n(X)$ — logical permanent of order n 109
Σ_n — operator of addition of two n-bit numbers 12
$\Sigma_{m,n}$ — operator of summation of m n-bit numbers 48
$SH_n(v,x)$ — shift operator of an n-bit number x by v positions to the left 14
$\mathrm{MRG}_{m,n}$ — operator of merging of sorted arrays of lengths m and n 32
SORT_n — operator of sorting of an array of length n 31
$ST_G(X)$ — Kirchhoff polynomial (spanning tree polynomial) of a graph G 19
$T_{n,b}(x)$ — operator for conversion a number $x < 2^n$ from b-ary representation to binary 30
T_n^k — monotone symmetric boolean function of n variables with threshold k, $T_n^k = (x_1 + \ldots + x_n \geqslant k)$ 31

▷ Proof of a lemma or corollary. □

▶ Proof of a theorem. ■

• Additional notes.

Chapter 1

Sequential method $\boxed{S}$

In fact, the method does not have a certain, generally accepted name. But this is the simplest way of computation that immediately comes to mind: to try to reduce a problem of size n to a problem of size $n-1$.

Circuits for linear functions $\boxed{s}$

Linear boolean function of n variables $\Lambda_n(x_1,\ldots,x_n) = x_1 \oplus x_2 \oplus \ldots \oplus x_n$ is easy to compute by the rule

$$\Lambda_n = x_n\overline{\Lambda_{n-1}} \vee \overline{x}_n\Lambda_{n-1}, \quad \text{or} \quad \Lambda_n = x_n\overline{\Lambda_{n-1}} \vee \overline{(x_n \vee \overline{\Lambda_{n-1}})}, \tag{1.1}$$

where Λ_{n-1} is a linear function of variables $x_1,\ldots,x_{n-1}$.

Theorem 1.1. $\mathsf{C}_{\mathcal{U}_2}(\Lambda_n) \leqslant 3n-3, \quad \mathsf{C}_{\mathcal{B}_0}(\Lambda_n) \leqslant 4n-4.$

▶ Formulas (1.1) lead to the relations $\mathsf{C}_{\mathcal{U}_2}(\Lambda_n) \leqslant \mathsf{C}_{\mathcal{U}_2}(\Lambda_{n-1}) + 3$ and $\mathsf{C}_{\mathcal{B}_0}(\Lambda_n) \leqslant \mathsf{C}_{\mathcal{B}_0}(\overline{\Lambda_{n-1}}) + 4$. It remains to note that $\mathsf{C}_{\mathcal{B}_0}(\overline{\Lambda}_n) \leqslant \mathsf{C}_{\mathcal{B}_0}(\Lambda_{n-1}) + 4$ is also true, since formulas (1.1) remain valid under the inversion of the linear functions involved in them. The corresponding circuits are shown in Fig. 1.1[1)]. ■

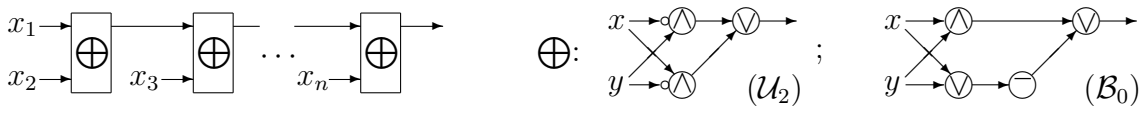


Figure 1.1: Circuits for linear functions

The simple method of computation turns out to be optimal. In this case, the proof of lower bounds is dual to the proof of upper bounds. The corresponding

[1]The circuit over the basis $\mathcal{B}_0$ in Fig. 1.1 implements a linear function or its negation depending on the parity of n. Small circles at inputs of elements are negations.

method of reasoning in the theory of lower complexity bounds is called the method of substitution by constants or the method of gate elimination. Let us provide a simple example as an illustration. The following result is due to C. Schnorr [274].

Theorem 1.2 ([274]). $\mathsf{C}_{\mathcal{U}_2}(\Lambda_n) \geqslant 3n - 3$.

► Assume $n \geqslant 2$. Consider an arbitrary minimal circuit computing a linear function $f = \Lambda_n \oplus \sigma$, where $\sigma \in \mathbb{B}$. There is a gate e_1 in the circuit with two inputs connected to different variables, denote them by x and y. By the property of the functions of the basis $\mathcal{U}_2$, there exist constants $\alpha, \beta \in \mathbb{B}$ such that under each of the substitutions $x = \alpha$ and $y = \beta$ the output of the gate e_1 becomes a constant. Note that at least one of the variables x, y (in fact, both), say x, is connected to some other gate e_2 (otherwise the substitution $x = \alpha$ would eliminate the dependence of function f on y, and vice versa).

Then, under assignment $x = \alpha$, the circuit is simplified: at least the gates e_1, e_2, and some gate e_3 to which the output of the gate e_1 was attached can be removed from it. The new circuit implements a linear function of $n - 1$ variables. We obtain $\mathsf{C}_{\mathcal{U}_2}(f) \geqslant \min\{\mathsf{C}_{\mathcal{U}_2}(\Lambda_{n-1}), \mathsf{C}_{\mathcal{U}_2}(\overline{\Lambda_{n-1}})\}$, from which the required bound immediately follows. ■

• The tightness of the bound of Theorem 1.1 for the basis $\mathcal{B}_0$ was proved by N. P. Red'kin in [257] also by the method of substitution by constants, but the proof requires consideration of several cases. Moreover, Red'kin [260] proved that in any complete boolean basis $\mathcal{B}$, $\mathsf{C}_{\mathcal{B}}(\Lambda_n) \leqslant 7(n-1)$ holds for $n \geqslant 2$, and this bound is tight, for example, over the basis $\mathcal{B} = \{\wedge, \bar{}\}$.

Standard adder circuits $\boxed{s}$

By Σ_n we will denote the boolean $(2n, n+1)$-operator of addition of n-digit numbers: $\Sigma_n(A, B) = A + B$. Let in binary notation

$$A = [a_{n-1}, a_{n-2}, \ldots, a_0], \quad B = [b_{n-1}, b_{n-2}, \ldots, b_0], \quad A + B = [z_n, z_{n-1}, \ldots, z_0].$$

Theorem 1.3. $\mathsf{C}_{\mathcal{B}_2}(\Sigma_n) \leqslant 5n - 3$.

► The result is provided by a circuit implementing a simple school method of addition. Its structure is shown in Fig. 1.2.

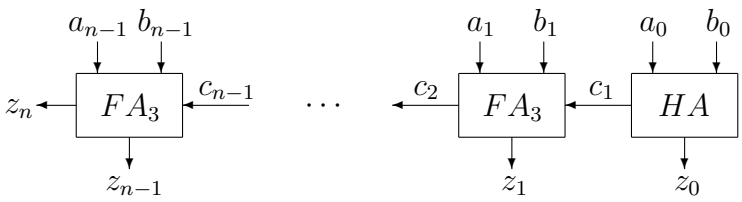


Figure 1.2: Standard adder circuit

Sequentially moving from the least significant digits to the most significant ones, each time the next digit of the sum and the carry to the next position is computed by a three-bit adder circuit, denoted FA_3 (full adder), according to the formulas

$$x_i = a_i \oplus b_i, \quad y_i = a_i b_i, \quad z_i = x_i \oplus c_i, \quad c_{i+1} = y_i \oplus x_i c_i \tag{1.2}$$

with complexity 5, see Fig. 1a). There is no carry in the least significant digit, so the addition is performed by a simpler HA (half adder) circuit: $z_0 = x_0 = a_0 \oplus b_0$, $c_1 = y_0 = a_0 b_0$. ■

• N. P. Red'kin in [259] showed that the described circuit is minimal, i.e., $\mathsf{C}_{\mathcal{B}_2}(\Sigma_n) = 5n - 3$. This proof is much more involved than for the circuit complexity of a linear function.

If the basis does not contain linear functions (case $\mathcal{B}_0$ or $\mathcal{U}_2$), then it is more convenient to compute carries by the formulas[2)]

$$c_{i+1} = y_i \vee x_i c_i, \quad x_i = a_i \vee b_i, \quad y_i = a_i b_i. \tag{1.3}$$

For example, this way we can obtain the bounds $\mathsf{C}_{\mathcal{U}_2}(\Sigma_n) \leqslant 7n - 4$ and $\mathsf{C}_{\mathcal{B}_0}(\Sigma_n) \leqslant 9n - 5$.

The circuits for subtraction are constructed similarly.

Greatest common divisor. Binary algorithm $\boxed{s}$

The greatest common divisor of two n-digit numbers $\mathrm{GCD}(a, b)$, as almost everybody knows, can be computed via the Euclidean algorithm in $O(n)$ iterations of the form[3)] $(a, b) := (b,\ a \bmod b)$.

When working in binary arithmetic, the binary algorithm proposed by J. Stein seems somewhat more natural [310]. It constitutes the cyclic execution of iterations:

1. If $a < b$, then $\mathrm{GCD}(a, b) = \mathrm{GCD}(b, a)$.
2. If $b = 0$, then $\mathrm{GCD}(a, b) = a$.
   Let $a' = a \bmod 2$, $b' = b \bmod 2$.
3. If $a' = b' = 0$, then $\mathrm{GCD}(a, b) = 2 \cdot \mathrm{GCD}(a/2, b/2)$;
   else if $a' = 0, b' = 1$, then $\mathrm{GCD}(a, b) = \mathrm{GCD}(a/2, b)$;
   else if $a' = 1, b' = 0$, then $\mathrm{GCD}(a, b) = \mathrm{GCD}(a, b/2)$;
   else, then $\mathrm{GCD}(a, b) = \mathrm{GCD}((a - b)/2, b)$.

Thus, the method scans the given numbers (in binary notation) from right to left and modifies them along the way. It is quite obvious that at each iteration the total length of the numbers a and b decreases by at least 1, so computing the GCD of n-digit numbers requires no more than $2n - 1$ such iterations.

[2)] The difference between (1.2) and (1.3) comes from two ways of representing the majority function: $\mathrm{maj}_3(a, b, c) = ab \oplus ac \oplus bc = ab \vee ac \vee bc$.

[3)] As soon as $(r, 0)$ is obtained at some step, the conclusion $\mathrm{GCD}(a, b) = r$ is made.

Theorem 1.4. $\mathsf{C}(\mathrm{GCD}_n) \preccurlyeq n^2$.

▶ As usual, representing a GCD algorithm as a circuit requires a little extra work. We connect $2n-1$ blocks in series, each implementing the next iteration of the algorithm. More precisely, the i-th block performs the transformation $(a_i, b_i, e_i, k_i) \to (a_{i+1}, b_{i+1}, e_{i+1}, k_{i+1}, r_i)$, where (a_i, b_i) and (a_{i+1}, b_{i+1}) denote a pair of numbers (a, b) at the input and output assuming $a_0 = a$ and $b_0 = b$. Here $e_{i+1} = e_i \vee (\min\{a_i, b_i\} = 0)$ is an indicator of the fulfillment of the condition of step 2 at some of the first i iterations (we assume $e_0 = 0$), $k_{i+1} = k_i + \overline{a'_i} \cdot \overline{b'_i}$ is the exponent of the power of two accumulated during the execution of the case 1 of step 3 (initially, $k_0 = 0$), $r_i = \min\{a_i, b_i\}$ is the odd component of $\mathrm{GCD}(a, b)$ in the case when the condition of step 2 is met.

The computations are finalized by a subcircuit for choosing the correct value. The iteration number at which the algorithm terminates is determined by the condition $\overline{e_{i-1}} \cdot e_i = 1$. Then $\mathrm{GCD}(a, b) = 2^{k_i} r_i$.

Each of the blocks in the chain includes comparison, subtraction, addition, and selection (multiplexor) subcircuits and therefore has linear complexity. The final circuit includes a selection subcircuit of complexity $O(n^2)$ (the choice can be performed by the formula[4)] $[k,\, r] = \bigvee_i (\overline{e_{i-1}} \cdot e_i)[k_i,\, r_i]$) and a shift subcircuit whose complexity is estimated as $O(n \log n)$ by the following lemma (see also in [205]).

Let $SH_n(v, x)$ denote the operator of shifting an n-digit number x by $v < n$ positions to the left: $(v, x) \to 2^v x$.

Lemma 1.1 ([205])**.** $\mathsf{C}(SH_n) \preccurlyeq n \log n$.

▷ A simple circuit implementing a shift may be built via the sequential method. We use the binary notation of the shift value: $v = [v_k, \ldots, v_0]$. Set $x_0 = x$ and further $x_i = 2^{2^i v_i} x_{i-1}$ for $i = 1, \ldots, k$. Then $x_k = 2^v x$.

The shifting circuit is obtained by connecting $k \sim \log_2 n$ subcircuits of $(2, 1)$-multiplexors in series, each of which, depending on the value of v_i, selects either x_{i-1} or $2^{2^i} x_{i-1}$. □

■

• There are at least a dozen variations of both the Euclidean algorithm and the binary algorithm: in particular, there is a right-handed version of the former and a left-handed version of the latter. Theoretically faster GCD algorithms are discussed in the next chapter, see p. 30.

Formulae for natural numbers $\boxed{s}$

The next problem is a rather entertaining. Write a natural number n as a formula using addition, multiplication, brackets and as few 1s as possible. For example, the shortest formula for the number 11 is $(1+1)(1+1+1+1+1)+1$. In other words, the problem is to determine the value $\Phi_{\mathcal{A}^{\mathbb{N}}_{+}}(n)$, which we will simply denote[5)] $\Phi_{+}(n)$.

[4] Disjunction is performed bitwise.

[5] Usually this value is denoted by $||n||$.

The first mention of this problem is found in the paper of K. Mahler and J. Popken [211] from 1953. Despite the interest in the problem and several particular results, for a long time only trivial general bounds were known: $3\log_3 n \leqslant \Phi_+(n) < 3\log_2 n \approx 4.33\ln n$ (for $n \geqslant 2$). The upper bound here is provided by Horner's scheme. Only in 2009 J. Zelinsky obtained the first nontrivial upper bound and later improved it to $\Phi_+(n) < 3.76\ln n$ [339]. Below, we present a simplified version of his method.

Theorem 1.5 ([339]). *For any $n \geqslant 2$, we have $\Phi_+(n) \leqslant 10\log_{12} n \approx 4.02\ln n$.*

► Let $v_p(n)$ denote the number of digits $p-1$ at the lower end of p-ary notation of n. In other words, this is the largest k for which $n \equiv -1 \bmod p^k$.

Also consider a transformation $\left[\frac{*-a}{b}\right] : n \to \frac{n-a}{b}$. Assuming $b \mid (n-a)$, note that

$$\Phi(n) \leqslant \Phi\left(\left[\frac{*-a}{b}\right] n\right) + \Phi(a) + \Phi(b). \tag{1.4}$$

We will prove by induction on n that $\Phi(n) \leqslant C\ln n$ for a suitable (possibly smaller) constant C — its value will be established in the course of the proof.

0) For $2 \leqslant n \leqslant 6$, the inequality holds with the constant $C = 5/\ln 5 \approx 3.11$. Now, assuming $n \geqslant 7$, we prove the induction step from $n-1$ to n. Further, for brevity, we set $v_p = v_p(n)$.

1) Let $v_2 \leqslant 1$, in other words, $n \not\equiv 3 \bmod 4$. If the least significant binary digits of n are n_1 and n_0, then we pass from n to $n' = \left[\frac{*-n_1}{2}\right]\left[\frac{*-n_0}{2}\right] n > 1$ and apply (1.4). So we obtain $\Phi(n) \leqslant \Phi(n') + 5$. As a consequence, the induction step is proved for any $C \geqslant 5/\ln 4 \approx 3.61$.

2) Let $v_3 = 0$. Then $a = n \bmod 3 \leqslant 1$. We pass from n to $n' = \left[\frac{*-a}{3}\right] n$. According to (1.4), we obtain $\Phi(n) \leqslant \Phi(n') + 4$. In this case, the induction step is proved for any $C \geqslant 4/\ln 3 \approx 3.64$.

3) In the remaining case, $v_2 \geqslant 2$ and $v_3 \geqslant 1$. In particular, $n \geqslant 11$. Note that $n \equiv (2^{v_2}3^{v_3} - 1) \bmod 2^{v_2+1}3^{v_3}$.

3.1) Consider $n' = \left[\frac{*-1}{2}\right]^{v_2} n$. It is easy to check that $2 \mid n'$ and $v_3(n') = v_3(n)$. Therefore, for $n'' = n'/2$ we have $n'' \equiv \frac{3^{v_3}-1}{2} \bmod 3^{v_3}$. Thus, the ternary notation of n'' ends in v_3 ones. Therefore, for $n'' \neq \frac{3^{v_3}-1}{2}$ the transition to $n''' = \left[\frac{*-1}{3}\right]^{v_3} n''$ is possible, otherwise — to $1 = \left[\frac{*-1}{3}\right]^{v_3-1} n''$. In the former (general) case, we obtain

$$\Phi(n) \leqslant \Phi(n''') + 3v_2 + 2 + 4v_3 \leqslant C\ln(n/(2^{v_2+1}3^{v_3})) + 3v_2 + 4v_3 + 2.$$

As a consequence, to justify the induction step, we need

$$3v_2 + 4v_3 + 2 \leqslant C((v_2+1)\ln 2 + v_3\ln 3).$$

For $C \geqslant 2/\ln 2 \approx 2.88$, this inequality follows from the simpler one:

$$3v_2 + 4v_3 \leqslant C(v_2\ln 2 + v_3\ln 3). \tag{1.5}$$

In the particular case $n'' = \frac{3^{v_3}-1}{2}$ we directly obtain a formula for $n = 2^{v_2}3^{v_3} - 1$ of size $3v_2 + 4v_3 - 2$. It remains to notice that condition (1.5) is sufficient to verify the induction step, since $\ln(n+1) \leqslant \ln n + \frac{1}{11}$ for $n \geqslant 11$.

3.2) Let $v_3 < v_2/2$. Set $n' = \left[\frac{*-2}{3}\right]^{v_3} n$. It is easy to check that $a = (n' \bmod 3) \neq 2$ and $v_2(n') = v_2(n)$. Then n' has the form

$$n' = 2^{v_2+1}(3m - \delta) + 2^{v_2} - 1, \qquad \delta = \begin{cases} 0, & a = (v_2 \bmod 2) \\ 1, & a \neq (v_2 \bmod 2) \end{cases}.$$

Consider $n'' = \left[\frac{*-a}{3}\right] n'$. Then

$$(n'' \bmod 2^{v_2+1}) \in \left\{ \frac{2^{v_2}-1}{3}, \ \frac{2^{v_2}-2}{3}, \ \frac{5\cdot 2^{v_2}-2}{3}, \ \frac{5\cdot 2^{v_2}-1}{3} \right\}.$$

In the base-4 representation, these remainders have the form, respectively

$$4^{\frac{v_2}{2}-1} + \ldots + 4 + 1, \qquad 2(4^{\frac{v_2-1}{2}-1} + \ldots + 4 + 1),$$
$$4^{\frac{v_2}{2}} + 2(4^{\frac{v_2}{2}-1} + \ldots + 4 + 1), \qquad 3\cdot 4^{\frac{v_2-1}{2}} + (4^{\frac{v_2-1}{2}-1} + \ldots + 4 + 1).$$

Therefore, for $n'' \neq \frac{2^{v_2}-1}{3}$, a transition to $n''' = \left[\frac{*-d}{4}\right]^{\lfloor v_2/2 \rfloor} n''$ is possible, where $d \in \{1, 2\}$. Representing, as in item 1), the transformation $\left[\frac{*-d}{4}\right]$ as the composition $\left[\frac{*-n_1}{2}\right]\left[\frac{*-n_0}{2}\right]$, we obtain

$$\Phi(n) \leqslant \Phi(n''') + 5v_3 + 4 + 5\lfloor v_2/2 \rfloor \leqslant C \ln(n/(3^{v_3+1}4^{\lfloor v_2/2 \rfloor})) + 5\lfloor v_2/2 \rfloor + 5v_3 + 4.$$

In this case, to justify the induction step, we require the condition

$$5\lfloor v_2/2 \rfloor + 5v_3 + 4 \leqslant C(2\lfloor v_2/2 \rfloor \ln 2 + (v_3+1)\ln 3).$$

For $C \geqslant 4/\ln 3$, this inequality follows from

$$5\lfloor v_2/2 \rfloor + 5v_3 \leqslant C(2\lfloor v_2/2 \rfloor \ln 2 + v_3 \ln 3). \tag{1.6}$$

In the remaining case $n'' = \frac{2^{v_2}-1}{3}$ (with $2 \mid v_2$ and $a = 0$) the transition to $1 = \left[\frac{*-1}{4}\right]^{(v_2/2)-1} n''$ is performed. Thus we obtain a formula for $n = 2^{v_2}3^{v_3} - 1$ of size $5v_3 + 5\lfloor v_2/2 \rfloor - 2$. As in item 3.1) above, condition (1.6) ensures the induction step.

3.3) It remains to determine the minimal constant C with which the proof proceeds. Its value is determined by inequalities (1.5) and (1.6). For any v_2, v_3, we need to satisfy

$$C \geqslant \min\left\{ \frac{3v_2 + 4v_3}{v_2 \ln 2 + v_3 \ln 3}, \ \frac{5\lfloor v_2/2 \rfloor + 5v_3}{2\lfloor v_2/2 \rfloor \ln 2 + v_3 \ln 3} \right\}.$$

For $v_3 \geqslant v_2/2$, we use the first estimate, its maximum is achieved at $v_3 = v_2/2$, and for $v_3 \leqslant \lfloor v_2/2 \rfloor$, we use the second, the maximum of which is achieved at $v_3 = \lfloor v_2/2 \rfloor$. In both cases the maximal values are $10/\ln 12$. ■

• A stronger bound $\Phi_+(n) < 3.76 \ln n$ was obtained by Zelinsky by analyzing expansions over a large number of prime bases. It is assumed that the maximum of the ratio $\Phi_+(n)/\ln n$ is achieved at $n = 1439$ and is equal to $26/\ln 1439 \approx 3.58$. Of course, this does not exclude the possibility of obtaining bounds of the form $(C + o(1)) \ln n$ with much smaller constants C. K. Amano [13], relying on computer calculations, proved that $\Phi_+(n) < 3.24 \ln n$ for almost all n.

Dynamic programming algorithms. Computation of the connectivity function and the Hamiltonian $\boxed{s}$

Let us associate the edges of a complete directed graph K_n on n vertices with boolean variables $X = \{x_e\}$. The values of the variables define an arbitrary graph $G \subset K_n$: $x_e = 1$ means that $e \in G$; $x_e = 0$ — that $e \notin G$. The (s,t)-*connectivity function* of a graph determines the presence of a directed path connecting vertices s and t. It can be defined by the formula

$$CONN_n(X) = \bigvee_{P \text{ — } (s,t)\text{-path in } K_n} \bigwedge_{e \in P} x_e.$$

Richard Ernest Bellman
University of South Carolina, 1965 to 1984

Even though the number of paths connecting two vertices in a graph may be exponentially large, they can all be "searched" by a simple algorithm of polynomial complexity. This algorithm was independently discovered by L. Ford [91], E. Moore [221], and R. Bellman [22] in the 1950s.

Theorem 1.6 ([22, 91, 221]). $\mathsf{C}_{\mathcal{B}_M}(CONN_n) \preccurlyeq n^3$.

▶ Let the graph vertices be numbered with natural numbers from 1 to n. For convenience, we assume $s = 1$ and $t = n$.

Let $y_j^{(l)}$ denote the function indicating the presence of a path of length l from vertex 1 to vertex j. All $y_j^{(l)}$ can be computed by the recurrent formulas

$$y_j^{(1)} = x_{1,j}, \qquad y_j^{(l+1)} = \bigvee_{i=2}^{n-1} y_i^{(l)} \cdot x_{i,j} \tag{1.7}$$

with complexity $\preccurlyeq n^3$. Finally, $CONN_n = y_n^{(1)} \vee y_n^{(2)} \vee \ldots \vee y_n^{(n-1)}$. ■

The principle of progressive approaching towards solving a problem via solving similar subproblems is known as *dynamic programming* (the term was proposed by Bellman). It is the basis for constructing efficient algorithms for many optimization problems (the Steiner tree problem, finding a maximal independent set in a graph, etc.).

• The Bellman—Ford—Moore algorithm works not only in boolean, but also in many other arithmetic semirings. It has a considerable practical significance for tropical semirings $(\mathbb{R}, \min, +)$. The boolean connectivity problem corresponds to the problem of finding the shortest path connecting two vertices in a graph. In the latter problem variables x_e take real values and are interpreted as weights assigned to edges, the computed function has the form $\min_P \sum_{e \in P} x_e$, where P runs over a set of paths. The algorithm relies heavily on the idempotency of the additive operation of a semiring (note that formulas (1.7) involve paths with cycles). The common real polynomial — the analogue of the function $CONN_n$ — already has superpolynomial monotone arithmetic complexity $\succcurlyeq n^2 2^n$, as shown by M. Jerrum and M. Snir [135].

Let $\Pi(j_1, \ldots, j_k)$ denote the set of cyclic permutations of numbers $j_1, \ldots, j_k$. *Hamiltonian* of order n is a polynomial

$$HAM_n(X) = \sum_{(i_1,\ldots,i_{n-1})\in\Pi(1,2,\ldots,n-1)} x_{0,i_1}x_{i_1,i_2}\cdot\ldots\cdot x_{i_{n-2},i_{n-1}}x_{i_{n-1},0},$$

in which each monomial corresponds to a Hamiltonian cycle in the complete directed graph on n vertices $0, 1, \ldots, n-1$. When substituting the variables for indicators of the presence of edges in an arbitrary graph G (as above), the polynomial takes the value of the number of Hamiltonian cycles in G.

The dynamic programming algorithm of Bellman [23], M. Held and R. Karp [126] allows one to construct an optimal monotone arithmetic circuit for the Hamiltonian.

Theorem 1.7 ([23, 126]). $\mathsf{C}_{\mathcal{A}_+}(HAM_n) \preccurlyeq n^2 2^n$.

▶ Denote by $H_{s,J}$, where $J \subset [\![n]\!] \setminus \{0, s\}$ and $|J| = k$, the polynomial

$$H_{s,J} = \sum_{(i_1,\ldots,i_k)\in\Pi(J)} x_{s,i_1}x_{i_1,i_2}\cdot\ldots\cdot x_{i_{k-1},i_k}x_{i_k,0}$$

enumerating Hamiltonian paths from a vertex s to a vertex 0 in the complete graph on the vertex set $J \cup \{0, s\}$.

Let $H_{s,\varnothing} = x_{s,0}$. Generally, we have

$$H_{s,J} = \sum_{j\in J} x_{s,j}\cdot H_{j,J\setminus\{j\}} \qquad \text{and} \qquad HAM_n = \sum_{j=1}^{n-1} x_{0,j}\cdot H_{j,[\![n]\!]\setminus\{0,j\}}. \tag{1.8}$$

If all polynomials $H_{s,J}$ for $1 \leqslant s \leqslant n-1$ and $|J| = k-1$ are computed, then the calculation of polynomials $H_{s,J}$, $|J| = k$, according to formula (1.8) requires no more than $2knC^k_{n-2}$ operations. Therefore,

$$\mathsf{C}(HAM_n) \leqslant 2n + \sum_{k=1}^{n-2} 2knC^k_{n-2} \asymp n^2 2^n.$$

■

• Jerrum and Schnir [135] showed that the bound of Theorem 1.7 is tight in order not only for the real arithmetic ring, but also for the tropical semiring $(\mathbb{R}, \min, +)$, in which the Hamiltonian corresponds to the optimization problem of the travelling salesman — finding the shortest tour visiting all vertices of the graph. At the same time, the multiplicative complexity of the algorithm of Theorem 1.7, which, if calculated carefully, is $(n-1)((n-2)2^{n-3}+1)$, is the minimum possible. The complexity of optimization problems solved by the dynamic programming approach is discussed in detail in [140].

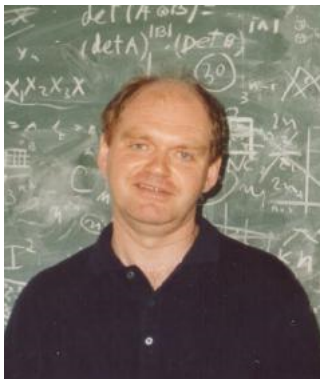
Monotone circuits for the Kirchhoff polynomial $\boxed{s}$

The idea of sequential computation allows us to obtain, among others, quite nontrivial results, as we will see in the example of computing Kirchhoff polynomials.

Consider a connected undirected graph G whose edges are labeled by symbols of real variables x_e (weights). Recall that a *spanning tree* of a graph is a subgraph that is a tree connecting all the vertices of the graph. *Kirchhoff polynomial* (spanning tree polynomial) of graph G is defined as

$$ST_G(X) = \sum_{T \text{ — span. tree in } G} \prod_{e \in T} x_e.$$

The Kirchhoff polynomial of a disconnected graph is conveniently defined as the product of Kirchhoff polynomials of connected components. Multiple edges are allowed.



Dmitry Yur'evich Grigoriev
French National Centre for Scientific Research, Lille, since 1998

The monotone arithmetic complexity of the Kirchhoff polynomial is large: S. Jukna and H. Saiwert [141] proved[6] that $\mathsf{C}_{\mathcal{A}^{\mathbb{R}}_{+}}(ST_{K_n}) = 2^{\Omega(\sqrt{n})}$. The situation may change when the computational basis extends. So, recently S. Fomin, D. Grigoriev and G. Koshevoy [90] proposed simple monotone circuits with division for this problem[7]. The method is based on sequentially adding nodes to a graph.

Theorem 1.8 ([90]). *For any graph G on n vertices, $\mathsf{C}_{\mathcal{A}^{\mathbb{R}}_{D+}}(ST_G) \preccurlyeq n^3$, where $\mathcal{A}_{D+} = \{+, *, /\}$.*

▶ It suffices to construct a circuit for the complete graph K_n. A circuit for an arbitrary graph is obtained by assigning zeros to some variables.

Label the graph vertices by numbers from 1 to n, and the edges — by pairs of numbers.

The circuit is constructed inductively. Let $w_n = x_{1,n} + x_{2,n} + \ldots + x_{n-1,n}$. Remove vertex n from the graph and add new edges $e_{i,j}$ connecting all possible pairs i, j of the remaining vertices[8]. Assign weight $x'_{i,j} = x_{i,n} \cdot x_{j,n}/w_n$ to the edge $e_{i,j}$. Denote the new graph by K'_{n-1}.

Our goal is to prove the equality

$$ST_{K_n}(X) = w_n \cdot ST_{K'_{n-1}}(X, X'). \tag{1.9}$$

If we select edges incident to vertex n in an arbitrary spanning tree of graph K_n, we obtain the following method of enumerating all spanning trees. Let $J = \{J_1, \ldots, J_s\}$ be some partition of vertex set $\{1, \ldots, n-1\}$ into subsets. In each complete graph on vertices J_i we select a spanning tree and connect it with vertex n by an edge, see Fig. 1.3a.

[6] The result is valid not only in the arithmetic but also in the tropical semiring $(\mathbb{R}, \min, +)$.

[7] In a certain sense, circuits with division are not quite monotone, since they allow computing some polynomials with negative coefficients, for example, $x^2 - xy + y^2$ as $(x^3 + y^3)/(x + y)$.

[8] In the course of the proof, we allow multiple edges to connect the same pair of vertices.

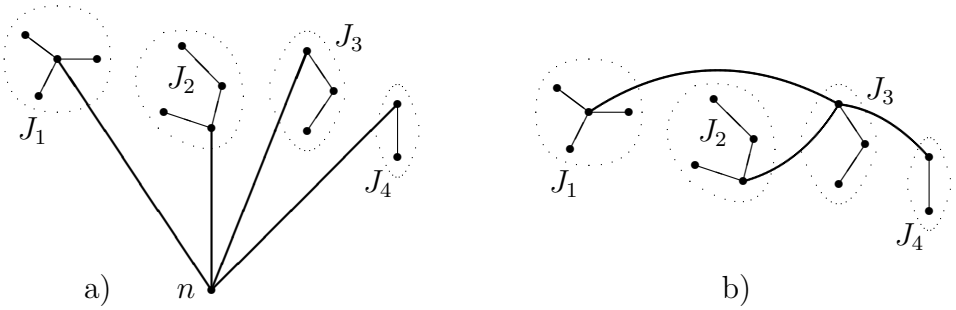


Figure 1.3: Structure of spanning trees of graphs K_n (a) and K'_{n-1} (b)

Let K_A be a complete subgraph of K_n on a vertex set A. Denote $w_A = \sum_{i\in A} x_{i,n}$. Then

$$ST_{K_n} = \sum_J \prod_{A\in J} w_A \cdot ST_{K_A}. \tag{1.10}$$

Similarly, by selecting a set of additional edges $e_{i,j}$ from a spanning tree of K'_{n-1}, we obtain a method for enumerating all spanning trees in K'_{n-1}. Again, let $J = \{J_1, \ldots, J_s\}$ be a partition of the vertex set $\{1, \ldots, n-1\}$. In each graph K_{J_i}, choose a spanning tree. Connect these trees by edges $e_{i,j}$, guided by some (external) spanning tree of the complete graph constructed on the sets J_i as on vertices, see Fig. 1.3b. For any $1 \leqslant k, l \leqslant s$, set

$$X'_{k,l} = \sum_{i\in J_k,\, j\in J_l} x'_{i,j} = w_{J_k} \cdot w_{J_l}/w_n. \tag{1.11}$$

Let K_J denote the complete graph on "vertices" J_i with edge weights $X'_{k,l}$. We obtain

$$ST_{K'_{n-1}} = \sum_J ST_{K_J} \prod_{A\in J} ST_{K_A}. \tag{1.12}$$

Now (1.9) follows from (1.10) and (1.12), if $ST_{K_J} = \left(\prod_{A\in J} w_A\right)/w_n$. Let us check this.

Consider an auxiliary problem. Let the weight of an edge (i, j) in the complete graph K_s on s vertices be $x_i x_j$ (in fact, we now assign weights to vertices). The following lemma is a variant of A. Cayley's theorem [54].

Lemma 1.2 ([54]). $ST_{K_s} = x_1 x_2 \cdot \ldots \cdot x_s (x_1 + x_2 + \ldots + x_s)^{s-2}$.

▷ The proof is based on a well-known method of counting spanning trees.

By definition, the contribution of each spanning tree to the polynomial ST_{K_s} is $\prod_{i=1}^s x_i^{d_i}$, where d_i is the degree of a vertex i in the tree.

A spanning tree is uniquely determined by the code $[a_1, a_2, \ldots, a_{s-2}]$, $1 \leqslant a_i \leqslant s$, as follows: a_1 is the number of a vertex with which the minimum number dangling (i.e. degree-1) vertex is connected; we delete this dangling vertex together with the

edge incident to it, then a_2 is defined similarly, and so on until only one vertex remains in the tree[9)].

It remains to note that a tree encoded by $[a_1, a_2, \ldots, a_{s-2}]$ corresponds to the monomial $x_1 x_2 \cdot \ldots \cdot x_s \prod_{i=1}^{s-2} x_{a_i}$ in the polynomial ST_{K_s}. □

According to (1.11), in Lemma 1.2 we should set $x_i = w_{J_i}/\sqrt{w_n}$, so that taking into account $w_{J_1} + \ldots + w_{J_s} = w_n$ we obtain $ST_{K_J} = \left(\prod_{A \in J} w_A\right)/w_n$, which proves (1.9).

Finally, it is easy to see that any set of multiple edges (in the graph K'_{n-1}) can be replaced by one whose weight is the sum of the weights of these edges — the Kirchhoff polynomial will not change. Therefore, formula (1.9) implies the relation

$$\mathsf{C}(ST_{K_n}) \leqslant \mathsf{C}(ST_{K_{n-1}}) + O(n^2),$$

from which the bound of the theorem follows.

In conclusion, we note that when moving from graph K_n to an arbitrary connected graph G, no divisions by zero occur in the circuit, since $w_n \neq 0$ in all calculation formulas. In the case of a disconnected graph, it is sufficient to compute the Kirchhoff polynomials of all connected components. ■

• The boolean version of the Kirchhoff polynomial $\bigvee_T \bigwedge_{e \in T} x_e$ (the sum is taken over spanning trees of $T \subset G$) determines the connectivity of G. It also has monotone boolean complexity $\preccurlyeq n^3$. This can be verified via the dynamic programming algorithm of B. Roy [265], R. Floyd [89], and S. Warshall [329] that determines whether there are paths connecting every pair of vertices in a graph. The tropical version of the problem has superpolynomial complexity $2^{\Omega(\sqrt{n})}$ [141].

[9]This method of enumerating spanning trees was proposed by H. Prüfer [253].

Chapter 2

Bisection method

Dividing a problem into similar parts, two or more (also known as divide-and-conquer approach), is one of the most productive techniques in the theory of fast computing, allowing one to construct efficient recursive algorithms in a variety of situations.

Formula complexity of linear functions

Short formulae for the linear boolean function $\Lambda_n(X)$ over the basis $\mathcal{B}_0$ are constructed by the generalizing (1.1) rule

$$\Lambda_n(X) = \Lambda_{n_1}(X^1) \cdot \overline{\Lambda_{n_2}(X^2)} \vee \overline{\Lambda_{n_1}(X^1)} \cdot \Lambda_{n_2}(X^2), \tag{2.1}$$

where $X = (X^1, X^2)$, $|X^i| = n_i$ and $n = n_1 + n_2$. The following result was obtained by S. V. Yablonskii [334] back in the 1950s.

Theorem 2.1 ([334]). *For any n,*

$$\Phi_{\mathcal{B}_0}(\Lambda_n) \leqslant n^2 + \left(n - 2^{\lfloor \log_2 n \rfloor}\right)\left(2^{\lceil \log_2 n \rceil} - n\right) < (9/8)n^2. \tag{2.2}$$

In particular, for $n = 2^m$, we have $\Phi_{\mathcal{B}_0}(\Lambda_n) \leqslant n^2$.

▶ The required bound is proved by induction using (2.1) via partition of the set of variables into equal parts: $n_1 = \lfloor n/2 \rfloor$ and $n_2 = \lceil n/2 \rceil$. ■

Theorem 2.1 provides the correct order of complexity of linear functions due to the well-known lower bound obtained by V. M. Khrapchenko [155] $\Phi_{\mathcal{B}_0}(\Lambda_n) \geqslant n^2$, and in the case $n = 2^m$ the constructed formulae are simply minimal.

• The question of the accuracy of the bound (2.2) for any n (Yablonskii's problem [334]) remains open. Thanks to the efforts of K. L. Rychkov [267, 268] and D. Yu. Cherukhin [61], the tightness of Theorem 2.1 was established for all $n \leqslant 8$.

Note that the lower bound for the formula complexity may be proved by a method that is, in a certain sense, dual to the proof of the upper bound. Namely, a suitable

functional $\mu(f)$ is introduced on the set of functions, which, when moving from simple functions to complex ones, grows about the same as the complexity, but is easier to calculate for specific functions.

- Such a functional is called a *formal complexity measure*. If it satisfies the conditions

$$\mu(0)=\mu(1)=0,\quad \mu(x),\mu(\overline{x})\leqslant 1,\quad \mu(f\vee g)\leqslant \mu(f)+\mu(g),\quad \mu(f\cdot g)\leqslant \mu(f)+\mu(g),$$

then automatically $\Phi_{\mathcal{B}_0}(f)\geqslant \mu(f)$. The Khrapchenko method [155] corresponds to the measure

$$\mu(f)=\max_{N\subset f^{-1}(0),\ P\subset f^{-1}(1)}\frac{|R(N,P)|^2}{|N|\cdot|P|},$$

where $R(N,P)$ is the set of pairs of adjacent sets of N and P, i.e. differing in one coordinate. For more details, see, for example, [331, 139].

Integer multiplication. Karatsuba's method $\boxed{/_2}$

Anatoly Alekseevich Karatsuba
Moscow University, 1959 to 2008

Let's apply the above technique to the problem of multiplying numbers. We divide $2n$-digit numbers X and Y into blocks of size n: $X=X_1 2^n+X_0$, $Y=Y_1 2^n+Y_0$. Now the multiplication of the original numbers can be performed via four multiplications of "halves" and several additions according to the formula

$$XY=X_1Y_1 2^{2n}+(X_0Y_1+X_1Y_0)2^n+X_0Y_0.$$

This method yields a quadratic complexity bound $\mathsf{C}(M_n)\preccurlyeq n^2$, as does the direct column multiplication method, where M_n denotes the boolean $(2n,2n)$-operator of multiplication of n-digit numbers.

Unexpectedly[1)] in 1960, A. A. Karatsuba [147] discovered a more economical formula including only three half-size multiplications:

$$XY=X_1Y_1 2^{2n}+((X_0+X_1)(Y_0+Y_1)-X_1Y_1-X_0Y_0)2^n+X_0Y_0. \tag{2.3}$$

Theorem 2.2 ([147]). $\mathsf{C}(M_n)\preccurlyeq n^{\log_2 3}\prec n^{1.59}$.

▶ Note that $\mathsf{C}(M_{n+1})\leqslant \mathsf{C}(M_n)+O(n)$. Then formula (2.3) leads to the recurrence relation

$$\mathsf{C}(M_{2n})\leqslant \mathsf{C}(M_{n+1})+2\mathsf{C}(M_n)+O(n)=3\mathsf{C}(M_n)+O(n),$$

which resolves as $\mathsf{C}(M_n)\preccurlyeq n^{\log_2 3}$. ■

Despite its apparent simplicity, Karatsuba's method revolutionized the theory of fast computing in the early 1960s and opened ways to creating fast algorithms for many other problems.

[1] While many experts thought that efforts should be concentrated on proving the bound $\mathsf{C}(M_n)\asymp n^2$.

• Karatsuba's method is widely used in practice. A careful estimate of the complexity of the method for $n = 2^k$ is $\mathsf{C}_{\mathcal{B}_2}(M_n) < 25\frac{83}{405} \cdot 3^k$ [295]. The method is applied even more successfully to the polynomial multiplication. For example, in the most interesting case of binary polynomials, the complexity of multiplication by the optimized Karatsuba method is estimated as $\mathsf{C}_{\mathcal{A}^{\mathbb{F}_2}}(M_n^{\mathbb{F}_2}) < 5\frac{13}{18} \cdot 3^k$ [29].

A. L. Toom [320] generalized Karatsuba's method. Multiplication of long numbers, if divided into k blocks each, reduces to $2k-1$ multiplications of short numbers (their length corresponds to the block size). With an appropriate choice of the parameter k, we can deduce $\mathsf{C}(M_n) \preccurlyeq 2^{O(\sqrt{\log n})} n$.

Matrix multiplication. Strassen's method /2

Let us turn to the problem of implementing the product of square matrices $Z = XY$ over some semiring R by arithmetic circuits. Multiplication of $2n \times 2n$ matrices can be reduced to multiplication of matrices of size $n \times n$:

$$\begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} \cdot \begin{bmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{bmatrix} = \begin{bmatrix} X_{11}Y_{11} + X_{12}Y_{21} & X_{11}Y_{12} + X_{12}Y_{22} \\ X_{21}Y_{11} + X_{22}Y_{21} & X_{21}Y_{12} + X_{22}Y_{22} \end{bmatrix}. \tag{2.4}$$

Direct computation by formulas (2.4) leads to an algorithm of cubic complexity, $\mathsf{C}_{\mathcal{A}^R}(MM_n) \preccurlyeq n^3$, where MM_n^R denotes the operator of multiplication of $n \times n$ matrices over R.

V. Strassen [313] observed that one submatrix multiplication can be saved if the computations are performed by formulas

$$\begin{gathered} Z_{11} = U_1 + U_2 - U_3 + (X_{12} - X_{22})(Y_{21} + Y_{22}), \qquad Z_{12} = U_3 + U_5, \\ Z_{21} = U_2 + U_4, \qquad Z_{22} = U_1 - U_4 + U_5 + (X_{21} - X_{11})(Y_{11} + Y_{12}), \\ U_1 = (X_{11} + X_{22})(Y_{11} + Y_{22}), \qquad U_2 = X_{22}(Y_{21} - Y_{11}), \\ U_3 = (X_{11} + X_{12})Y_{22}, \qquad U_4 = (X_{21} + X_{22})Y_{11}, \qquad U_5 = X_{11}(Y_{12} - Y_{22}). \end{gathered} \tag{2.5}$$

The use of subtractions implies that R is a ring.

Theorem 2.3 ([313]). *If R is a ring, then $\mathsf{C}_{\mathcal{A}^R}(MM_n) \preccurlyeq n^{\log_2 7} \prec n^{2.81}$.*

▶ Formulas (2.5) lead to the relation

$$\mathsf{C}_{\mathcal{A}^R}(MM_{2n}) \leqslant 7\mathsf{C}_{\mathcal{A}^R}(MM_n) + O(n^2),$$

which is resolved as $\mathsf{C}_{\mathcal{A}^R}(MM_n) \preccurlyeq n^{\log_2 7}$. ■

• If computations are performed over a monotone basis[2)], then the cubic bound cannot be improved, $\mathsf{C}_{\mathcal{A}^R_+}(MM_n) \succcurlyeq n^3$ [152]. Already for the boolean semiring $(\mathbb{B}, \vee, \wedge)$ we have $\mathsf{C}_{\mathcal{B}_M}(MM_n) = 2n^3 - n^2$ [240].

Formulas (2.5), in addition to 7 multiplications, include 18 additive operations with submatrices. S. Winograd reduced the number of additions/subtractions to 15 (see, e.g., [167]), and in [332] he showed that the multiplicative complexity of 2×2 matrix multiplication is indeed 7. Later it was proved that if R is a field, then 15 additive operations are required for any 2×2 matrix multiplication algorithm of multiplicative complexity 7 [252, 52]. But this limitation can be circumvented by restructuring matrices, which also changes the definition of the multiplication operation, see further on p. 87.

[2)] This will be the case, for example, in a semiring R with an irreversible addition operation.

Fast Fourier transform $\boxed{/_2}$

Let ζ be a primitive root of order N in a commutative associative ring R with unity. The *discrete Fourier transform (DFT) of order* N is a linear (N,N)-operator over R,

$$\mathrm{DFT}_{N,\zeta}[R](x_0,\dots,x_{N-1}) \to (x_0^*,\dots,x_{N-1}^*), \qquad x_j^* = \sum_{i=0}^{N-1} \zeta^{ij} x_i. \tag{2.6}$$

• *Basic properties of DFT.* The inverse DFT coincides with the forward one up to another primitive root and normalization:

$$\mathrm{DFT}_{N,\zeta}^{-1} = N^{-1} \cdot \mathrm{DFT}_{N,\zeta^{-1}}.$$

In the polynomial notation, DFT computes the values of a polynomial at the points ζ^i, $i = 0,\dots,N-1$. Let $\Gamma(t) = x_0 + x_1 t + \ldots + x_{N-1} t^{N-1}$. Then

$$\mathrm{DFT}_{N,\zeta}(x_0,\dots,x_{N-1}) = \left(\Gamma(\zeta^0),\dots,\Gamma(\zeta^{N-1})\right).$$

The DFT matrix is a Vandermonde matrix composed of the powers of a primitive root, $\zeta^0,\zeta^1,\dots,\zeta^{N-1}$.

At the heart of the theory of fast Fourier transforms lies a technique of decomposing composite-order DFTs from the work of J. Cooley and J. Tukey [70].

Lemma 2.1 ([70])**.** *Let ζ be a primitive root of degree $P \cdot Q$. Then*

$$\mathsf{C}_{\mathcal{A}_L}(\mathrm{DFT}_{PQ,\zeta}) \leqslant P \cdot \mathsf{C}_{\mathcal{A}_L}(\mathrm{DFT}_{Q,\zeta^P}) + Q \cdot \mathsf{C}_{\mathcal{A}_L}(\mathrm{DFT}_{P,\zeta^Q}) + (P-1)(Q-1).$$

▷ For any $p = 0,\dots,P-1$ and $q = 0,\dots,Q-1$, write

$$x_{pQ+q}^* = \sum_{I=0}^{PQ-1} \zeta^{I(pQ+q)} x_I = \sum_{i=0}^{Q-1}\sum_{j=0}^{P-1} \zeta^{(iP+j)(pQ+q)} x_{iP+j} =$$

$$= \sum_{i=0}^{Q-1}\sum_{j=0}^{P-1} \zeta^{iqP+jpQ+jq} x_{iP+j} = \sum_{j=0}^{P-1} (\zeta^Q)^{jp} \cdot \zeta^{jq} \cdot \sum_{i=0}^{Q-1} (\zeta^P)^{iq} x_{iP+j}. \tag{2.7}$$

The inner sums are computed by DFTs of order Q. The results are multiplied by powers of the primitive root ζ (among PQ multiplications, $P+Q-1$ multiplications are performed by unit — those with $j=0$ or $q=0$). Finally, the outer sums are computed via DFTs of order P. □

• If $\mathrm{GCD}(P,Q) = 1$, then a more economical way of computation is possible, which does not require additional multiplications by powers of ζ. It was found by I. Good [111] in the late 1950s.

For $I = 0,\dots,PQ-1$ denote $x_I = x_{i,j}$, where $i = I \bmod Q$ and $j = I \bmod P$. Let a,b be the Bezout coefficients from the equality $aP + bQ = 1$. Note that $I = (iaP + jbQ) \bmod PQ$.

For an arbitrary $K = 0,\dots,PQ-1$ set $s = bK \bmod P$ and $t = aK \bmod Q$. It is easy to see that $K = (sQ + tP) \bmod PQ$.

Finally, observe that if ζ is a primitive root of order PQ, then $\zeta^{bQ^2} = \zeta^{Q(1-aP)} = \zeta^Q$ and similarly $\zeta^{aP^2} = \zeta^P$. Now the following identity is easy to verify:

$$x_K^* = \sum_{I=0}^{PQ-1} \zeta^{IK} x_I = \sum_{j=0}^{P-1} \sum_{i=0}^{Q-1} \zeta^{(iaP+jbQ)(sQ+tP)} x_{i,\,j} =$$

$$= \sum_{j=0}^{P-1} \sum_{i=0}^{Q-1} \zeta^{(iatP^2+jbsQ^2)} x_{i,\,j} = \sum_{j=0}^{P-1} \sum_{i=0}^{Q-1} (\zeta^P)^{it} (\zeta^Q)^{js} x_{i,\,j} = \sum_{j=0}^{P-1} (\zeta^Q)^{js} \sum_{i=0}^{Q-1} (\zeta^P)^{it} x_{i,\,j}.$$

Calculations by these formulas require only Q DFTs of order P and P DFTs of order Q. However, the scope of application of Good's method is narrower than for the Cooley—Tukey method.

By recursively applying Lemma 2.1 and taking into account $\mathsf{C}_{\mathcal{A}_L}(\mathrm{DFT}_2) = 2$, we prove[3)]

Theorem 2.4 ([70]). $\mathsf{C}_{\mathcal{A}_L}(\mathrm{DFT}_{2^k}) \leqslant 3k2^{k-1} - 2^k + 1$.

- The circuit from Theorem 2.4 contains $k2^k$ additions/subtractions and $(k-2)2^{k-1}+1$ scalar multiplications by powers of the root other than ± 1. The first bound has not been improved yet, however, the scalar multiplicative complexity of DFT of order n is actually $O(n)$ [333, 125], but this bound comes at the cost of a significant increase in additive complexity.

Parallel prefix circuits /2

Consider a system of prefix sums

$$x_1 \circ x_2 \circ \ldots \circ x_i, \qquad i = 1, \ldots, n, \tag{2.8}$$

over some semigroup $(G, \circ)$ with an associative, but not necessarily commutative, addition operation. Circuits over the basis of a single operation $\{\circ\}$ that implement this system are usually called *prefix circuits*.

Michael John Fischer
Yale University, since 1981

The complexity of system (2.8) is obviously $n-1$, but the depth of the minimal circuit is also $n-1$. Back in the 1960s, due to the needs of some applications, it became necessary to construct parallel prefix circuits, i.e. circuits of depth $O(\log n)$. The question is: what could be the complexity of such circuits. A more specific question is: is it possible to construct a prefix circuit of the minimal possible depth $\lceil \log_2 n \rceil$ and complexity $O(n)$? An affirmative answer was given by R. Ladner and M. Fischer [182].

In [182] a family of prefix circuits $\Pi_k(n)$ is proposed. The circuit $\Pi_k(n)$ satisfies two conditions:

$(*)$ implements system (2.8) with depth $\lceil \log_2 n \rceil + k$;

$(**)$ implements the maximal prefix sum $x_1 \circ x_2 \circ \ldots \circ x_n$ with depth $\lceil \log_2 n \rceil$.

The Ladner—Fischer method combines two variants of the dividing-by-halves principle: partitioning the set of variables into groups with major and minor indices, and partitioning into groups with even and odd indices.

[3)] Strictly speaking, in [70] a weaker complexity estimate of $2k2^k$ is proven.

The design of the circuit $\Pi_0(n)$ follows the first variant. The circuit is arranged as follows:

a) the sums $\sigma_i = x_1 \circ \ldots \circ x_i$, $1 \leqslant i \leqslant \lceil n/2 \rceil$, are computed by the circuit $\Pi_1(\lceil n/2 \rceil)$;

b) the sums $\tau_i = x_{\lceil n/2 \rceil + 1} \circ \ldots \circ x_i$, $\lceil n/2 \rceil < i \leqslant n$, are computed by the circuit $\Pi_0(\lfloor n/2 \rfloor)$;

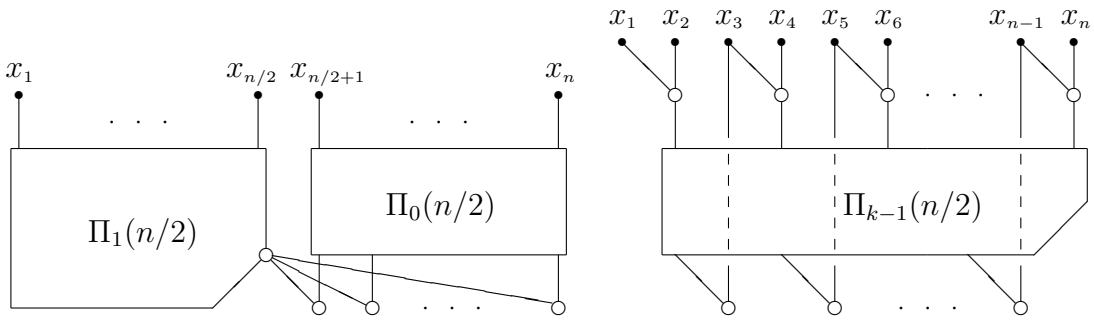
c) the remaining sums $x_1 \circ \ldots \circ x_i$, $\lceil n/2 \rceil < i \leqslant n$, are computed as $\sigma_{\lceil n/2 \rceil} \circ \tau_i$.

For $k \geqslant 1$, the constructions of circuits $\Pi_k(n)$ are obtained from the even-odd partition:

a) first, the sums $z_i = x_{2i-1} \circ x_{2i}$, $1 \leqslant i \leqslant \lfloor n/2 \rfloor$, are computed. For odd n, we additionally set $z_{\lceil n/2 \rceil} = x_n$;

b) the sums $\sigma_i = x_1 \circ \ldots \circ x_{2i}$, $1 \leqslant i \leqslant \lfloor n/2 \rfloor$, and the sum $x_1 \circ \ldots \circ x_n = z_1 \circ \ldots \circ z_j$ are computed by the circuit $\Pi_{k-1}(\lceil n/2 \rceil)$;

c) the remaining sums $x_1 \circ \ldots \circ x_{2i-1}$, $2 \leqslant i \leqslant \lfloor n/2 \rfloor$, are computed as $\sigma_{i-1} \circ x_{2i-1}$.



a) $\Pi_0(n)$ b) $\Pi_k(n)$

Figure 2.1: Ladner—Fischer prefix circuits

The described circuits are shown in Fig. 2.1 (rounding in indices and arguments is omitted). It is easy to verify that the circuits $\Pi_k(n)$ indeed satisfy conditions $(*)$, $(**)$.

Let $\{\Phi_k\}$ denote the sequence of Fibonacci numbers: $\Phi_1 = \Phi_2 = 1$, $\Phi_{k+1} = \Phi_k + \Phi_{k-1}$. Denote by $P_k(n)$ the minimal complexity of a prefix circuit on n inputs satisfying the depth constraint $\lceil \log_2 n \rceil + k$.

Theorem 2.5 ([182]). *For $k \leqslant m$,*

$$\mathsf{C}_{\{\circ\}}(\Pi_k(2^m)) = 2(1 + 2^{-k})2^m - \Phi_{5+m-k} + 1 - k. \tag{2.9}$$

Hence, $P_k(n) \leqslant 2(1 + 2^{-k})n$.

▶ By construction,

$$\mathsf{C}(\Pi_0(n)) = \mathsf{C}(\Pi_0(\lfloor n/2 \rfloor)) + \mathsf{C}(\Pi_1(\lceil n/2 \rceil)) + \lfloor n/2 \rfloor, \tag{2.10}$$

$$\mathsf{C}(\Pi_k(n)) = \mathsf{C}(\Pi_{k-1}(\lceil n/2 \rceil)) + 2\lfloor n/2 \rfloor - 1, \tag{2.11}$$

from where the required relations can be easily deduced by induction. ■

In particular, in the most interesting case $k = 0$ we obtain

Corollary 2.1. $P_0(2^m) \leqslant 4 \cdot 2^m - \Phi_{m+5} + 1 \sim 4 \cdot 2^m$.

- The author in [291, 292] established the tight estimate

$$P_0(2^m) \leqslant 3.5 \cdot 2^m - (8.5 + 3.5(m \bmod 2))2^{\lfloor m/2 \rfloor} + m + 5, \tag{2.12}$$

which is attained in some semigroups $(G, \circ)$. As a consequence (from the construction), $P_0(n) \leqslant 3.5n$ for any n. Also in [291, 292] the minimal possible complexity of universal prefix circuits satisfying conditions $(*)$, $(**)$ in the case $n = 2^m$ was established. It turns out that the way of constructing $\Pi_k(2^m)$ circuits in the Ladner—Fischer method is optimal for all k except $k = 1$. The question of the existence of more economical commutative prefix circuits[4)] remains open. In [291, 292] it is also shown that in the special case of the group $(\mathbb{B}, \oplus)$ the upper bound (2.12) can be improved to $36n/11$.

Parallel prefix circuits have numerous applications (see, e.g., the survey [36]), the most famous of which are prefix adders. This method goes back to the work of Yu. P. Ofman [231]. The advantage of this approach is the possibility of constructing adders with various useful properties depending on the type of the chosen prefix circuit (there are many known variants of such circuits). In particular, it is easy to achieve logarithmic depth while maintaining a linear order of complexity (the depth of the standard adder is linear, see Fig. 1.2).

Recall that the heart of an n-digit integer adder is the computation of a system of functions (carries)

$$c_i = F_i(X, Y) = y_{i-1} + x_{i-1}(y_{i-2} + \ldots + x_2(y_1 + x_1 y_0)\ldots), \quad i = 1, \ldots, n, \tag{2.13}$$

see (1.2), (1.3).

Let us introduce a binary operation $\circ$ on vectors of height 2:

$$\begin{pmatrix} f_1 \\ p_1 \end{pmatrix} \circ \begin{pmatrix} f_2 \\ p_2 \end{pmatrix} = \begin{pmatrix} f_2 + p_2 f_1 \\ p_2 p_1 \end{pmatrix}. \tag{2.14}$$

It is easy to verify that this operation is associative:

$$\begin{pmatrix} f_2 + p_2 f_1 \\ p_2 p_1 \end{pmatrix} \circ \begin{pmatrix} f_3 \\ p_3 \end{pmatrix} = \begin{pmatrix} f_3 + p_3 f_2 + p_3 p_2 f_1 \\ p_3 p_2 p_1 \end{pmatrix} = \begin{pmatrix} f_1 \\ p_1 \end{pmatrix} \circ \begin{pmatrix} f_3 + p_3 f_2 \\ p_3 p_2 \end{pmatrix}.$$

Now formula (2.13) for the carry function can be written as

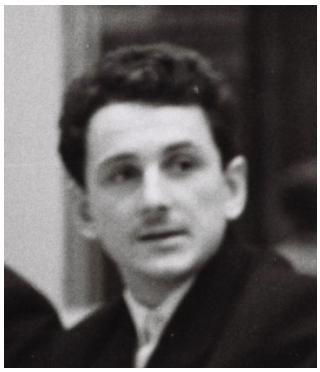
$$\begin{pmatrix} F_i \\ x_{i-1} \cdot \ldots \cdot x_0 \end{pmatrix} = \begin{pmatrix} y_0 \\ 1 \end{pmatrix} \circ \begin{pmatrix} y_1 \\ x_1 \end{pmatrix} \circ \ldots \circ \begin{pmatrix} y_{i-1} \\ x_{i-1} \end{pmatrix}. \tag{2.15}$$

Thus, the problem of computing all carries is reduced to the implementation of a system of prefix sums of the form (2.8).

A single operation $\circ$ can be implemented by a circuit over $\mathcal{B}_2$ of complexity 3 and depth 2. Therefore, based on a circuit that computes n prefix sums with complexity C and depth D, one can construct an adder circuit of complexity $3(C+n)-1$ and depth $2D + 2$.

[4)] That is, over groups $(G, \circ)$ with a commutative operation $\circ$.

Parallel adders. Khrapchenko's method /2



Valery Mikhailovich Khrapchenko
Institute of Applied Math., USSR Acad. Sci./RAS, Moscow, 1966 to 2019

The depth of a prefix n-digit adder cannot be less than $2\log_2 n$. A more careful study of expressions (2.13) led V. M. Khrapchenko [154] to construct an adder of asymptotically optimal depth $\sim \log_2 n$. For generality, we will assume that the calculations are performed over the monotone arithmetic basis $\mathcal{A}_+$.

Theorem 2.6 ([154]). *For the function F_n from (2.13), $\mathsf{D}_{\mathcal{A}_+}(F_n) \leqslant \log_2 n + \sqrt{2\log_2 n} + O(1)$.*

▶ Consider the following functions:

$$F_{r,m}(X,Y) = x_{m+r-1}\cdot\ldots\cdot x_{m+1}\cdot x_m \cdot F_m(X,Y). \quad (2.16)$$

In particular, $F_{0,m} = F_m$. From (2.13) follows a decomposition of the form

$$F_{r,m+n} = F_{r,m} + F_{r+m,n} \quad (2.17)$$

(we omit the arguments of the functions here for brevity). Denote $d(s,k) = \mathsf{D}(F_{s\cdot 2^k,\,2^k})$. Obviously, $d(s_1,k) \leqslant d(s_2,k)$ when $s_1 \leqslant s_2$. Therefore, as a consequence of (2.16) and (2.17),

$$d(2^l,k) \leqslant \max\{k+l,\, d(0,k)\} + 1, \quad (2.18)$$
$$d(s,k) \leqslant d(2s+1,k-1) + 1. \quad (2.19)$$

Lemma 2.2. *For $C_l^2 < m \leqslant C_{l+1}^2$, we have $d(0,m) \leqslant m + l + 1$.*

▷ The inequality is proved by induction on m with base $m = 1$. To prove the induction step from $m-1$ to $m = C_l^2 + r$, by applying inequality (2.19) r times, and then (2.18), we obtain

$$d(0,m) \leqslant d(2^r - 1, C_l^2) + r \leqslant C_{l+1}^2 + r + 1 = m + l + 1.$$

□

To finish the proof of the theorem, it remains to substitute the value $m = \lceil\log_2 n\rceil$ into Lemma 2.2 and observe that the condition $C_l^2 < m \leqslant C_{l+1}^2$ implies $l = \lceil(\sqrt{1+8m} - 1)/2\rceil$. ■

When implementing carries according to rules (1.3) we establish

Corollary 2.2. $\mathsf{D}_{\mathcal{B}_0}(\Sigma_n) \leqslant \log_2 n + \sqrt{2\log_2 n} + O(1)$.

• In [176] S. R. Kosaraju proved the lower bound $\mathsf{D}_{\mathcal{A}_+}(F_n) \geqslant \log_2 n + \sqrt{(2-o(1))\log_2 n}$. Consequently, the result of Theorem 2.6 cannot be significantly improved without additional assumptions about the properties of a semiring in which the calculations are performed. M. I. Grinchuk showed that in the boolean semiring $\{\mathbb{B}, \vee, \wedge\}$ the bound can be improved [114], see below on p. 96. The question of the possibility of refining the bound in the field $\mathbb{F}_2$ remains open.

Direct application of the formulas of Theorem 2.6 leads to an adder of nonlinear complexity. Having slightly modified the computation method, Khrapchenko [154] constructed an n-bit adder of linear complexity $(12 + o(1))n$ over the basis $\mathcal{B}_0$ at the cost of increasing the depth within $\log_2 n + O(\sqrt{\log n})$. In [103] it is shown that an adder of such complexity can be implemented with the depth $\log_2 n + \sqrt{(2 + o(1)) \log_2 n}$. Over the basis $\mathcal{B}_2$ the complexity of adders decreases to $(8 + o(1))n$.

Other applications

The scope of the method in the theory of fast computing is so vast that any detailed description of all possible applications would take up a whole book, or even more than one. Guided by the principle of respect for information[5)], we will only briefly list a few more important problems where the method works efficiently.

Transition between number systems. A simple and asymptotically fast algorithm for conversion a number between number systems with different bases was proposed by A. Schönhage[6)].

To convert a number A from a b-ary system to binary, split it in half, $A = A_1 b^k + A_0$, then convert the "halves" A_0 and A_1 to the binary representation by a recursive call of the algorithm. For number b^k the binary representation can be considered precomputed; it remains to perform the multiplication and addition of binary numbers. In the opposite direction, an n-digit binary number A is divided with remainder by the power $b^k \asymp 2^{n/2}$, so we obtain $A = A_1 b^k + A_0$. It remains to convert the halves A_0 and A_1 to the b-ary representation.

Thus, for the complexity of the operator $T_{n,b}$ of converting a number of size $< 2^n$ from a b-ary representation to binary, and for the complexity of the inverse operator, the following recurrence relations hold:

$$\mathsf{C}(T_{n,b}) \leqslant 2\mathsf{C}(T_{n/2,b}) + \mathsf{C}(M_{n/2}) + O(n), \qquad \mathsf{C}(T^{-1}_{n,b}) \leqslant 2\mathsf{C}(T^{-1}_{n/2,b}) + \mathsf{C}(QR_{n,n/2}),$$

where $QR_{n,m}$ is the operator of division with remainder of an n-digit number by an m-digit number.

In terms of the smoothed multiplication complexity functional[7)] $\mathsf{M}(n)$ the above relations resolve as $\mathsf{C}(T_{n,b}), \mathsf{C}(T^{-1}_{n,b}) \preccurlyeq \mathsf{M}(n) \log n$, since $\mathsf{C}(QR_{2n,n}) \preccurlyeq \mathsf{M}(n)$ (proved by S. Cook [69]; see further on p. 53). By a result of D. Harvey and J. van der Hoeven [119], $\mathsf{M}(n) \preccurlyeq n \log n$.

Fast GCD computation. The dividing-by-halves approach allowed to radically reduce the complexity of classical Euclidean algorithms for computing GCD. Having improved the original method of D. Knuth [166], A. Schönhage in [276] obtained the upper bound[8)] $\mathsf{C}(\mathrm{GCD}_n) \preccurlyeq \mathsf{M}(n) \log n$.

The idea of the method is as follows. An operator $\mathrm{HGCD}_n(A, B) = (a, b, M)$ is introduced, which, given two n-digit numbers A, B, produces a pair of numbers a, b of length $\approx n/2$ while preserving GCD, as well as a transition matrix M, whose elements also have size $\approx n/2$. For example, one can require the fulfillment of the conditions

$$\mathrm{GCD}(a, b) = \mathrm{GCD}(A, B), \qquad a \geqslant 2^{n/2} > b, \qquad \begin{pmatrix} a \\ b \end{pmatrix} = M \begin{pmatrix} A \\ B \end{pmatrix}.$$

Thus, the computation of $\mathrm{GCD}(A, B)$ is reduced to the operation $(a, b, M) = \mathrm{HGCD}_n(A, B)$ and the computation of $\mathrm{GCD}(b, a \bmod b)$ with half-size numbers. Therefore,

$$\mathsf{C}(\mathrm{GCD}_n) \leqslant \mathsf{C}(\mathrm{GCD}_{n/2}) + \mathsf{C}(\mathrm{HGCD}_n) + \mathsf{C}(QR_{2n,n}) + O(n \log n), \tag{2.20}$$

[5] Reduce the volume of presentation where there is a lot of material, and reproduce the material in full where there is little (formulated by A. T. Fomenko in relation to the analysis of historical chronicles).

[6] Apparently, the method was never published by the author. Cited from [167].

[7] Satisfying the conditions $\mathsf{M}(n) \geqslant \mathsf{C}(M_n)$ and $\mathsf{M}(x + y) \geqslant \mathsf{M}(x) + \mathsf{M}(y)$ for all x, y.

[8] Strictly speaking, GCD algorithms are usually stated for the model of programs allowing branching. The adaptation to a circuit implementation is carried out similarly to how it was done for the binary GCD algorithm above, and does not affect the order of complexity.

where the last term takes into account the specifics of the implementation of the algorithm by circuits (as in the proof of Theorem 1.4).

The central point of the method is the implementation of HGCD. It is based on the observation due to D. Lehmer [185]: the first steps of the Euclidean algorithm are determined by the most significant digits of numbers A and B. In general terms, the sequence of steps for computing $\mathrm{HGCD}_n(A,B)$ is as follows. Let $A = A_1 2^{n/2} + A_0$ and $B = B_1 2^{n/2} + B_0$.

1. First, compute $(a', b', M_1) = \mathrm{HGCD}_{n/2}(A_1, B_1)$. With the use of the resulting transition matrix, we can obtain a new pair of numbers A', B' that satisfy the condition $\mathrm{GCD}(A', B') = \mathrm{GCD}(A, B)$ and have a length within $3n/4$ digits. In the first approximation, $\begin{pmatrix} A' \\ B' \end{pmatrix} = M_1 \begin{pmatrix} A \\ B \end{pmatrix}$, but a correction of several division-with-remainder steps forward or even backward may be required.

2. Writing $A' = A_3 2^{n/4} + A_2$ and $B' = B_3 2^{n/4} + B_2$, we perform another recursive call to the HGCD procedure producing $(a'', b'', M_2) = \mathrm{HGCD}_{n/2}(A_3, B_3)$. Applying matrix M_2 of transition from A', B', we obtain a desired pair a, b of length $\approx n/2$ bits, as well as the transition matrix M. Up to the required correction, $\begin{pmatrix} a \\ b \end{pmatrix} = M_2 \begin{pmatrix} A' \\ B' \end{pmatrix}$ and $M = M_2 M_1$.

The need of correcting results at each step and additional checks significantly complicate a careful presentation of the method. Principally, it is necessary to ensure that the intermediate steps of the algorithm do not generate too small numbers. This would lead to the impossibility of applying the Lehmer rule: the result would depend on those lower digits that were not taken into account in the calculations.

As a result, we obtain a recurrence relation

$$\mathsf{C}(\mathrm{HGCD}_n) \leqslant 2\mathsf{C}(\mathrm{HGCD}_{n/2}) + O(\mathsf{C}(M_n) + \mathsf{C}(QR_{2n,n}) + n\log n),$$

which leads to $\mathsf{C}(\mathrm{HGCD}_n) \preccurlyeq \mathsf{M}(n)\log n$ and, together with (2.20), to $\mathsf{C}(\mathrm{GCD}_n) \preccurlyeq \mathsf{M}(n)\log n$.

By now the method has received a number of modifications. Schönhage himself in [279] proposed to use division iterations with the choice of the largest remainder for calculating the GCD, which made it possible to simplify the control of convergence rate and the verification of correctness of the method. This algorithm, together with a more efficient HGCD procedure proposed by N. Möller, is described in [219]. D. Stehlé and P. Zimmermann [309] constructed a fast GCD algorithm based on the right-hand division with remainder (in this case, the control of the method parameters is also simpler). Finally, the method of D. Bernstein and B.-Y. Yang [30] is essentially based directly on the iterations of the binary GCD algorithm (see p. 13). The method [30] has advantage comparing to the above-mentioned ones due to a small number of branches and extremely simple control of convergence and correctness.

An arrangement of the fast GCD algorithm for a polynomial ring was made by R. Moenck in [218], a more modern version of the method is described in [30], and a substantially optimized version — in [130]. The analysis of the polynomial algorithm is somewhat simpler due to the absence of carries inside additions (so the correction of intermediate results is simpler). Circuits for the GCD of polynomials over general rings must inevitably include not only arithmetic operations, but also, for example, comparison operations.

Sorting and monotone circuits for threshold functions. Monotone threshold boolean functions of n variables together form a *sorting* operator (of a boolean set) $\mathrm{SORT}_n = (T_n^1, T_n^2, \ldots, T_n^n)$, where $T_n^k = (x_1 + \ldots + x_n \geqslant k)$ is the monotone symmetric function of n variables with threshold k. Therefore, the problems of computing threshold functions and sorting (or selection, if we are speaking about a single component of the operator) are closely related.

A popular model for implementing sorting is *comparator circuits*. A comparator is a subcircuit that orders two input numbers; in the boolean case, it performs the mapping $(x, y) \to (x \wedge y, x \vee y)$. Comparator circuits do not allow branching of comparator outputs[9], see also below on p. 60.

[9] A comparator circuit can be represented as a sequence of ordered n-element sets that satisfies the following conditions: 1) it starts with the input set; 2) each subsequent set is obtained from the previous one by ordering one pair of elements; 3) it ends with a linearly ordered set (or, more generally, contains the required partial order).

In a complete basis, $\mathsf{C}(\mathrm{SORT}_n) \asymp n$ trivially holds. Bounds on the complexity of monotone circuits are obtained by transferring known results on the complexity of comparator circuits. The naive method (compare all elements pairwise) leads to $\mathsf{C}_{\mathcal{B}_M}(\mathrm{SORT}_n) \preccurlyeq n^2$. A significantly more efficient sorting method with an elegant application of the bisection principle was proposed by K. Batcher [16]. The method is as follows. The set to be sorted is divided in half, sorting is performed in each of the halves, then the sorted subsets are merged. Thus,

$$\mathsf{C}_{\mathcal{B}_M}(\mathrm{SORT}_{2n}) \leqslant 2\mathsf{C}_{\mathcal{B}_M}(\mathrm{SORT}_{2n}) + \mathsf{C}_{\mathcal{B}_M}(\mathrm{MRG}_{n,n}),$$

where $\mathrm{MRG}_{m,n}$ denotes the operator[10)] of merging of sorted sets of size m and n.

Merging two sets is also done recursively by the bisection method. Separately, merging all even elements of both sets and all odd elements is done. To sort the two lists together, only $n-1$ comparisons are sufficient. Consequently,

$$\mathsf{C}_{\mathcal{B}_M}(\mathrm{MRG}_{n,n}) \leqslant 2\mathsf{C}_{\mathcal{B}_M}(\mathrm{MRG}_{n/2,n/2}) + 2(n-1),$$

whence $\mathsf{C}_{\mathcal{B}_M}(\mathrm{MRG}_{n,n}) \preccurlyeq n\log n$, therefore, $\mathsf{C}_{\mathcal{B}_M}(\mathrm{SORT}_n) \preccurlyeq n\log^2 n$. For a detailed exposition of the method, see [168, 57].

Later, M. Ajtai, J. Komlós, and E. Szemerédi [4, 5] proved that in fact $\mathsf{C}_{\mathcal{B}_M}(\mathrm{SORT}_n) \preccurlyeq n\log n$ (also proven in Theorem 5.6 below), but for reasonable values of n Batcher's method takes advantage.

When computing individual functions T_n^k with small thresholds k, the method of A. Yao [337] which is asymptotically optimal in the comparator circuit model works well. In it, input elements are divided into pairs, which are ordered. Note that among the younger elements of the pairs there are at most $k/2$ of the k largest elements of the input set. Therefore, we choose k largest elements among the greater elements of the pairs and $\lfloor k/2\rfloor$ largest elements among the younger elements of the pairs, and then determine k largest elements among them. So we obtain the recurrence relation

$$\mathsf{C}_{\mathcal{B}_M}\left(T_n^1,\ldots,T_n^k\right) \leqslant n + \mathsf{C}_{\mathcal{B}_M}\left(T_{n/2}^1,\ldots,T_{n/2}^k\right) + \mathsf{C}_{\mathcal{B}_M}\left(T_{n/2}^1,\ldots,T_{n/2}^{k/2}\right) + \mathsf{C}_{\mathcal{B}_M}(\mathrm{MRG}_{k,k/2}).$$

A careful transition of Yao's method to the model of monotone circuits, carried out by the author [301][11)], leads to a bound valid for constant or slowly growing k:

$$\mathsf{C}_{\mathcal{B}_M}(T_n^k) \lesssim (\lfloor\log_2 k\rfloor + \lfloor\log_2(4k/3)\rfloor)n.$$

Combining the results of V. E. Alekseev [7] or Yao [337] (designs of comparator circuits for threshold functions) with [4, 5] provides a general upper bound $\mathsf{C}_{\mathcal{B}_M}(T_n^k) \preccurlyeq n\log k$. The best known universal bound, valid for any threshold value, $\mathsf{C}_{\mathcal{B}_M}(T_n^k) \lesssim 6n\log_3 n$ was obtained by S. Jimbo and A. Maruoka in [136].

Multiplicative complexity of polynomials. Consider the problem of computing an arbitrary polynomial $f(x) = a_nx^n + \ldots + a_1x + a_0$ by circuits over the complete arithmetic basis $\mathcal{A}^{\mathbb{R}}$ or $\mathcal{A}^{\mathbb{C}}$. The simplest way to compute a polynomial is Horner's scheme:

$$f(x) = (\ldots((a_nx + a_{n-1})x + \ldots)x + a_0.$$

It involves n additions and n multiplications. The first estimate (for the number of additive operations) is generally best possible, as shown by E. G. Belaga [21] and V. Ya. Pan [233, 234]. They also proved (for the fields $\mathbb{C}$ and $\mathbb{R}$, respectively) that $n/2 + O(1)$ multiplications are always sufficient. Moreover, both estimates can be achieved with an accuracy of $O(1)$ in a single circuit.

The methods of Pan and Belaga are quite nontrivial and require rather complicated preliminary processing of the coefficients[12)]. An elegant way of computation, leading to only slightly worse estimate of $n/2 + O(\log n)$ multiplications, was proposed by M. Rabin and S. Winograd [254], see

[10] This is a partially defined boolean operator.

[11] A special method of circuit synthesis for the function T_n^2 is additionally used, see Theorem 11.1.

[12] If the coefficients of a polynomial are also treated as the inputs of the circuit, in other words, preliminary processing is impossible, then Horner's circuit is optimal, as Pan proved in [235].

also [244]. The method is based on the observation that a monic (i.e., with the leading coefficient 1) polynomial $f(x)$ of degree $2^{k+1}-1$ can be represented as

$$f(x) = (x^{2^k} + a)f_1(x) + f_0(x), \tag{2.21}$$

where f_0, f_1 are monic polynomials of degree $2^k - 1$. Therefore, after all powers $x^2, x^{2^2}, \ldots, x^{2^k}$ are prepared, any intermediate polynomial of degree $2^m - 1$ can be computed by formulas (2.21) via $2^{m-1} - 1$ multiplications.

The number of nonscalar multiplications required to compute a polynomial of degree n is generally bounded from below by $\sqrt{n}$ [244]. The best upper bound, due to M. Paterson and L. Stockmeyer [244], is $\sqrt{2n} + O(\log n)$. The method is conceptually close to the method [254] and also exploits the idea of division in half. A monic polynomial $f(x)$ of degree $p(2^{k+1} - 1)$ can be represented as

$$f(x) = (x^{p2^k} + c(x))f_1(x) + f_0(x), \tag{2.22}$$

where f_0, f_1 are monic polynomials of degree $p(2^k - 1)$ and $\deg c < p$. After all powers $x^2, x^3, \ldots, x^p$ and $x^{2p}, x^{2^2 p}, \ldots, x^{2^k p}$ are computed, each intermediate polynomial of degree $p(2^m - 1)$ may be computed by formulas (2.22) via $2^{m-1} - 1$ multiplications. The required estimate follows when choosing $p \approx \sqrt{n/2}$.

Chapter 3

Method of common parts $\boxed{C}$

The essence of the method is to select fragments that can be used multiple times. Surprisingly, this idea alone leads to a number of nontrivial asymptotically tight results.

Addition chains. Brauer's method $\boxed{c}$

Addition chains are (universal) additive one-input one-output circuits. They implement transforms $x \to nx$, but traditionally the input of the chain is supposed to be the constant 1, then the output is a natural number n.

Starting from the binary representation of a number, an addition chain for $n = [n_k, n_{k-1}, \dots, n_0]_2$ can be constructed by Horner's formula

$$n = (\dots(2n_k + n_{k-1})2 + \dots + n_1)2 + n_0. \tag{3.1}$$

This is known as the binary method — it yields the estimate $\mathsf{L}(n) \leqslant \log_2 n + \nu(n) - 1 < 2\log_2 n$, where $\nu(n)$ is the weight of n (the number of ones in binary notation). An obvious lower bound is $\mathsf{L}(n) \geqslant \log_2 n$ (at each step, the size of the computed number increases by a maximum of two times). A. Brauer [46] observed that in fact the lower bound is asymptotically tight.

Alfred Theodor Brauer
University of North Carolina, 1942 to 1964

Theorem 3.1 ([46]). $\mathsf{L}(n) \leqslant \log_2 n + (1 + o(1))\frac{\log_2 n}{\log_2 \log n}$.

▶ Let $n < 2^{tk}$. By organizing the digits of n into a matrix A of size $t \times k$, we can write

$$n = \left(1,\, 2^k,\, 2^{2k}, \dots,\, 2^{(t-1)k}\right) \cdot \begin{pmatrix} n_0 & n_1 & \cdots & n_{k-1} \\ n_k & n_{k+1} & \cdots & n_{2k-1} \\ \vdots & \vdots & \ddots & \vdots \\ n_{(t-1)k} & n_{(t-1)k+1} & \cdots & n_{tk-1} \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \\ \vdots \\ 2^{k-1} \end{pmatrix}. \tag{3.2}$$

The central point of the method is the choice of parameters satisfying $t > 2^k$. Thus, the matrix A ends up having many repeating rows.

Lemma 3.1. *For an arbitrary boolean $t \times k$ matrix A, $\mathsf{L}(A) \leqslant 2^k - k - 1$ holds.*

▷ There exist only $2^k - k - 1$ different rows of length k and weight $\geqslant 2$. It is easy to compute all such rows using one operation for each. □

Carrying out calculations according to formula (3.2) from right to left, by Lemma 3.1 we obtain

$$\mathsf{L}(n) \leqslant (k-1) + (2^k - k - 1) + (k+1)(t-1),$$

having in mind that the 2^k-ary version of formula (3.1) is used for the last multiplication. It remains to choose $k \approx \log_2 \log_2 n - 2 \log_2 \log_2 \log_2 n$. ■

- P. Erdős [84] showed that for almost all n we have

$$\mathsf{L}(n) \geqslant \log_2 n + (1 - o(1)) \frac{\log_2 n}{\log_2 \log n}.$$

Moreover, as V. V. and D. V. Kochergin [170] established, both in the upper bound of Theorem 3.1 and in the lower bound of Erdős we can substitute $(2 + o(1)) \frac{\log_2 \log \log n}{\log_2 \log n}$ for $o(1)$. The best absolute lower bound up to date is $\mathsf{L}(n) \geqslant \log_2 n + \log_2 \nu(n) - 2.13$ proven by A. Schönhage [277] (the accuracy of this result is emphasized by an easily verifiable fact that the bound $\lfloor \log_2 n \rfloor + \lceil \log_2 \nu(n) \rceil$ is achieved for any value of $\nu(n)$).

Asymptotically minimal circuits for boolean functions [c]

A boolean function of n variables can be specified by a size-2^n table of its values, and in general there is no shorter code[1)]. A well-known result of C. Shannon [307] shows that $O(2^n/n)$ elements are sufficient for encoding functions by (switching) circuits — so it occurs that the information about a function is contained rather in the topology of circuits than in elements. D. Muller [224] adapted Shannon's method to boolean circuits, and soon O. B. Lupanov developed an asymptotically optimal synthesis method [201]. Lupanov's method caused a sensation: it was hard to expect that a seemingly crude cardinality lower bound $\gtrsim 2^n/n$ would be so precise and constructively achievable in the asymptotic sense. The method is based on a simple result of Lupanov on the additive complexity of boolean matrices [200].

Lemma 3.2 ([200]). *Assume $q \succcurlyeq \log p$. Then for any boolean $p \times q$ matrix A,*

$$\mathsf{L}(A) \leqslant \left(1 + O\left(\frac{\log \log p}{\log p}\right)\right) \frac{pq}{\log_2 p}.$$

[1] Since there are 2^{2^n} different functions of n variables.

▷ Divide the matrix into vertical strips of width k: $A = (A_1|A_2|\ldots|A_s)$, where $s = \lceil q/k \rceil$. By Lemma 3.1, we obtain

$$\mathsf{L}(A) \leqslant \mathsf{L}(A_1) + \ldots + \mathsf{L}(A_s) + (s-1)p \leqslant s2^k + pq/k.$$

It remains to choose $k \approx \log_2 p - \log_2 \log_2 p$. □

Note that the lemma uses the same technique as Theorem 3.1 — reduction to the computation of supernarrow matrices. Thus, using repeated fragments of sums in different rows allows one to improve a trivial bound $\mathsf{L}(A) \leqslant p(q-1)$.

• In [200] the complexity bound is proved for rectifier circuits (of depth 2), but the proof for additive circuits is the same. It is easy to check that the bound of Lemma 3.2 is asymptotically tight for $\log q \prec \log p$ [200] (i.e., when the matrix A is narrow enough).

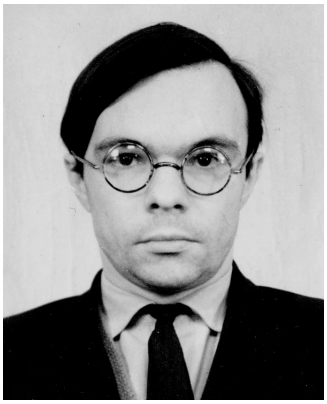
Theorem 3.2 ([201]). *For any boolean function f of n variables,*

$$\mathsf{C}_{\mathcal{B}_2}(f) \leqslant \left(1 + O\left(\frac{\log n}{n}\right)\right)\frac{2^n}{n}.$$

▶ Divide the set of n variables into two groups X, Y, where $|X| = k$ and $|Y| = n-k$. Let $X_J = \prod_{j\in J} x_j$, $J \subset \{1,\ldots,k\}$, denote products (monomials) of the variables of the first group, and let $Y_I = \prod_{i\in I} y_i$, $I \subset \{1,\ldots,n-k\}$ denote monomials of the variables of the second group. Any boolean function $f(X,Y)$ can be represented as a Zhegalkin polynomial

$$f(X,Y) = \bigoplus_I Y_I \bigoplus_J f_{I,J} X_J, \qquad (3.3)$$

where $F = (f_{I,J})$ is a boolean matrix of size $2^{n-k} \times 2^k$ (note that in fact formulas (3.2) and (3.3) are similar).



Oleg Borisovich Lupanov
Moscow University, 1959 to 2006

The computation of all X_J is performed by a linear operator of size $2^k \times k$ over $(\mathbb{B}, \wedge)$. According to Lemma 3.1 it can be implemented by a circuit of complexity 2^k. Similarly, the complexity of computing all Y_I does not exceed 2^{n-k}. The complexity of the linear transform with the matrix F (over $(\mathbb{B}, \oplus)$) is estimated by Lemma 3.2. Another $2\cdot 2^{n-k}$ operations are required for multiplying intermediate sums by monomials Y_I and for computing the outer sum in (3.3). Thus,

$$\mathsf{C}_{\mathcal{B}_2}(f) \leqslant 3\cdot 2^{n-k} + 2^k + \left(1 + O\left(\frac{\log(n-k)}{n-k}\right)\right)\frac{2^n}{n-k}.$$

By choosing $k \approx 2\log_2 n$ we obtain the statement of the theorem. ■

• In the basis $\mathcal{B}_0$, the same bound can be obtained if instead of (3.3) we apply the formula $f = \bigvee_I Y_I \bigvee_J f_{I,J} X_J$ derived from a DNF (disjunctive normal form), where X_J and Y_I are elementary conjunctions (products of variables and their negations). All elementary conjunctions of k variables may be computed with complexity $\sim 2^k$ (see Lemma 3.5 below).

Actually, in [201] Lupanov obtained a more general result $\mathsf{C}_{\mathcal{B}}(f) \sim 2^n/((s-1)n)$ for almost all functions of n variables, where s is the maximum number of essential variables for functions of a complete finite basis $\mathcal{B}$. Asymptotic high-precision complexity estimates were obtained by S. A. Lozhkin [193, 196]: for example, over the basis $\mathcal{B} \in \{\mathcal{B}_0, \mathcal{B}_2\}$, they take the form

$$\left(1 + \frac{\log_2 n - O(1)}{n}\right) \frac{2^n}{n} \lesssim \mathsf{C}_{\mathcal{B}}(\mathcal{P}_n) \lesssim \left(1 + \frac{\log_2 n + \log_2 \log n + O(1)}{n}\right) \frac{2^n}{n}.$$

Lemma 3.2, establishing the complexity of the class of boolean $p \times q$ matrices, is of independent value. For example, it allows to generalize Theorem 3.1 to the case of computing several numbers. Let $n = \max_i n_i$ and $s \log n \to \infty$. Then

$$\mathsf{L}(n_1, \ldots, n_s) \leqslant \log_2 n + (1 + o(1)) \frac{s \log_2 n}{\log_2 (s \log n)} + O(s). \tag{3.4}$$

▷ Let $n < 2^{tk}$. By transposing (3.2), we can write

$$n_i = bA_iC^T, \qquad b = (1,\, 2,\, 2^2, \ldots,\, 2^{k-1}), \qquad C = \left(1,\, 2^k,\, 2^{2k}, \ldots,\, 2^{(t-1)k}\right),$$

where A_i is a boolean $k \times t$ matrix composed of the digits of n_i. Then

$$\begin{pmatrix} n_1 \\ n_2 \\ \vdots \\ n_s \end{pmatrix} = BAC^T, \qquad B = \begin{pmatrix} b & 0\ldots0 & \cdots & 0\ldots0 \\ 0\ldots0 & b & \cdots & 0\ldots0 \\ \vdots & \vdots & \ddots & \vdots \\ 0\ldots0 & 0\ldots0 & \cdots & b \end{pmatrix}, \qquad A = \begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_s \end{pmatrix}.$$

As a consequence,

$$\mathsf{L}(n_1, \ldots, n_s) \leqslant \mathsf{L}(B) + \mathsf{L}(A) + \mathsf{L}(C^T) \leqslant 2s(k-1) + \mathsf{L}(A) + (t-1)k.$$

If $\log s \preccurlyeq \log\log n$, we set $t \approx \log_2^2 \log_2 n$. Otherwise (i.e., for large s), we choose $k = 1$. Further, the complexity of the matrix A (of size $sk \times t$) is estimated following Lemma 3.2. □

Bound (3.4) is a special case of a more general result of N. Pippenger [249] on the complexity of integer matrices. It is easy to verify by the cardinality method that in the class of all sets of s numbers of size $\leqslant n$ the above bound is asymptotically tight [249]. In the case of individually specified constraints $n_i \leqslant r_i$ S. B. Gashkov and V. V. Kochergin [104] refined bound (3.4) to

$$\mathsf{L}(n_1, \ldots, n_s) \leqslant \log_2 n + (1 + o(1)) \frac{R}{\log_2 R} + O(s), \qquad R = \log_2(r_1 \cdot \ldots \cdot r_s). \tag{3.5}$$

It is also tight in its class. Later Kochergin refined secondary terms in both estimates (3.4) and (3.5), see [169].

Circuits for linear boolean operators [c]

Lemma 3.2 establishes an asymptotic complexity for a class of sufficiently narrow boolean matrices. In the most interesting case of square matrices, an asymptotically tight result may be obtained by a more subtle application of the common part method proposed by E. I. Nechiporuk [226] (see also [228]).

Theorem 3.3 ([226]). *For an arbitrary boolean $n \times n$ matrix A,*

$$\mathsf{L}(A) \lesssim \frac{n^2}{2 \log_2 n}.$$

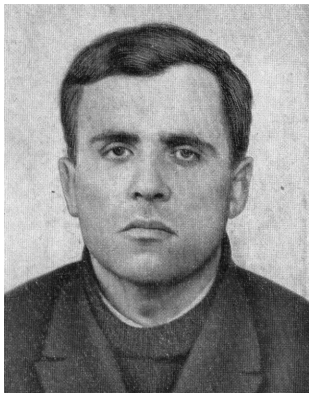
► First, we prove an auxiliary estimate.

Lemma 3.3. *For any boolean $p \times q$ matrix A,*

$$\mathsf{L}(A) \leqslant |A|/2 + q + p^2.$$

▷ We divide the ones in each column of the matrix into pairs. Thus, an arbitrary row σ_i is represented as the sum of substrings $\sigma_{\{i,j\}}$ common with other rows σ_j, $j \neq i$, and an unpaired substring σ'_i. We denote by A_1 the total weight of substrings $\sigma_{\{i,j\}}$, $i \neq j$, and by A_2 the total weight of substrings σ'_i. Then $2A_1 + A_2 = |A|$ and $A_2 \leqslant q$.

We compute all substrings $\sigma_{\{i,j\}}$ and σ'_i, and then the sums in each row. This requires at most $A_1 + A_2 + p^2 \leqslant |A|/2 + q + p^2$ operations. □



Eduard Ivanovich Nechiporuk
Leningrad University, 1960s to 1970

• A slightly simpler proof of the lemma may be obtained by applying the transposition principle (Lemma 8.1).

The proof of the theorem begins in the same way as in Lemma 3.2. We divide the matrix into vertical strips of width k: $A = (A_1|A_2|\dots|A_s)$, where $s = n/k$ (for simplicity, assume $k|n$). Calculating the sums in the strips corresponds to representing the matrices A_i as products B_iU_k of matrices B_i of size $n \times 2^k$, having a single one in each row, and $2^k \times k$ matrices U_k composed of all possible rows of length k:

$$A = (B_1|B_2|\dots|B_s) \cdot \begin{pmatrix} U_k & 0 & \cdots & 0 \\ 0 & U_k & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & U_k \end{pmatrix}.$$

We split the matrix $B = (B_1|B_2|\dots|B_s)$ into horizontal strips of height p and compute the sums in each strip (which is a matrix of size $p \times s2^k$ and weight ps) independently by the method of Lemma 3.3. As a result, we obtain

$$\mathsf{L}(A) \leqslant s\mathsf{L}(U_k) + |B|/2 + \lceil n/p \rceil (s2^k + p^2) = ns/2 + O(ns2^k/p + np).$$

The required upper bound follows when choosing $k \approx \log_2 n - 3\log_2\log_2 n$ and $p \asymp n/\log^2 n$. ■

• In fact, Nechiporuk [226] obtained a more general result, showing that the asymptotically tight bound

$$\mathsf{L}(A) \leqslant (1+\varepsilon)\frac{mn}{\log_2(mn)}, \qquad \varepsilon = o(1), \tag{3.6}$$

for the complexity of boolean $m \times n$ matrices holds for $\log_2 n \sim \alpha \log_2 m$, where $\alpha = r - 1 + r/s$ and $r, s \in \mathbb{N}$. The matching lower bound $\mathsf{L}(A) \geqslant (1-\delta)mn/\log_2(mn)$, $\delta \asymp \frac{\log\log n}{\log n}$, for almost all $m \times n$ matrices A can be established by the standard cardinality argument.

N. Pippenger [248] removed the restrictions on m, n, proving the validity of (3.6) for any m, n, where $\log m \asymp \log n$, with $\varepsilon \asymp \sqrt{\frac{\log\log n}{\log n}}$. The author in [298] showed that for $m \in n^\mu \log^{\pm O(1)} n$, $\mu \in \mathbb{Q}$, we can assume $\varepsilon \asymp \frac{\log\log n}{\log n}$, as in the lower bound.

The result for boolean matrices implies the most general form of the complexity bound for integer matrices [249] (slightly refined in [305]). Let A be an $m \times n$ matrix with elements $\leqslant q$, and $mn \log q \to \infty$. Then

$$\mathsf{L}(A) \leqslant \min\{m, n\} \log_2 q + (1 + o(1)) \frac{mn \log_2 q}{\log_2 (mn \log q)} + n. \tag{3.7}$$

It can be proved similarly to bound (3.4) on the complexity of vectors, see [305].

Complexity of monotone circuits. Principle of local coding $\boxed{c}\boxed{e}$

The problem of asymptotic complexity of the class $\mathcal{M}^n$ of monotone functions of n variables is related to their efficient enumeration (encoding). A suitable procedure was first proposed by D. Kleitman in 1969 [163], who established that $|\mathcal{M}^n| = 2^{(1+o(1))C_n^{n/2}}$. After this, it became possible to develop optimal methods for implementing monotone functions. For boolean circuits over complete bases, the corresponding result was obtained by A. B. Ugol'nikov [321]: in particular,

$$\mathsf{C}_{\mathcal{B}_0}(\mathcal{M}^n) \sim \frac{C_n^{n/2}}{n} \sim \frac{2^n}{n^{3/2}\sqrt{\pi/2}}.$$

The proof of the same asymptotic bound in the model of circuits over the monotone basis $\mathcal{B}_M$ turned out to be much more complicated. This problem was finally solved by A. E. Andreev [14].

• The asymptotics of the number of monotone functions was found by A. D. Korshunov [174] in a rather difficult way. Another proof was proposed by A. A. Sapozhenko in [271]. For more details on the history of the question, see survey [175].

We present a simpler, although nontrivial, result on the order of complexity of monotone circuits, which was established by N. P. Red'kin [258]. A simplified proof was proposed by A. V. Chashkin in [58].

The proof method illustrates a general approach to constructing optimal circuits proposed by O. B. Lupanov [205] and called by him the *principle of local coding*. The idea is to represent a function by such a binary code that the value of the function on each specific input is determined only by a local fragment of the code. Computing the function thus consists of (1) determining the required fragment and (2) decoding this fragment. Both stages must allow for simple implementation.

To encode monotone functions, we use partitions of a boolean cube into monotone chains and the fact that a function can only change value once on each chain.

Theorem 3.4 ([258]). $\mathsf{C}_{\mathcal{B}_M}(\mathcal{M}^n) \preccurlyeq C_n^{n/2}/n$.

▶ Let $n = 2k + m$. Recall that the boolean cube $\mathbb{B}^k$ can be partitioned into $C_k^{k/2}$ monotone chains (a consequence of Dilworth's theorem, see, for example, [138]). We write

$$\mathbb{B}^{2k} = \mathbb{B}^k \times \mathbb{B}^k = \Big(\bigcup \alpha\Big) \times \Big(\bigcup \beta\Big) = \bigcup_{\alpha \subset \mathbb{B}^k} \bigcup_{\beta \subset \mathbb{B}^k} (\alpha \times \beta),$$

where α, β are monotone chains from the (chosen) optimal partition of $\mathbb{B}^k$.

Lemma 3.4 ([258]). *The number of different monotone boolean functions defined on the Cartesian product of monotone chains $\alpha \times \beta$ is $C^{|\alpha|}_{|\alpha|+|\beta|}$.*

▷ Let $\alpha = (\alpha_1 \prec \ldots \prec \alpha_p)$ and $\beta = (\beta_1 \prec \ldots \prec \beta_q)$. Any monotone function $\varphi : \alpha \times \beta \to \mathbb{B}$ uniquely corresponds to a sequence of numbers $0 \leqslant k_1 \leqslant \ldots \leqslant k_p \leqslant q$, where k_i is the number of elements $x \in \beta$ for which $\varphi(\alpha_i, x) = 1$. The number of such sequences is C^p_{p+q}. □

Fix a partition $\mathbb{B}^{2k} = \bigcup I_j$, where any set I_j (except, perhaps, one) is the union of products $\alpha \times \beta$ such that $k+2 \leqslant \sum(|\alpha| + |\beta|) \leqslant 2k+2$. It follows from Lemma 3.4 that the number of monotone functions defined on I_j does not exceed 2^{2k+2}. Since $\sum_{\alpha,\beta \subset \mathbb{B}^k}(|\alpha| + |\beta|) = 2^{k+1}C^{k/2}_k$, the total number t of sets I_j in the partition is at most $2^{k+1}C^{k/2}_k/(k+2)$.

Let $|X| = 2k$, $|Y| = m$. Starting from the monotone DNF, we represent an arbitrary function $f \in \mathcal{M}_n$ as

$$f(X,Y) = \bigvee_{\sigma \in \mathbb{B}^m} f_\sigma(X) \cdot Y^\sigma = \bigvee_{\sigma \in \mathbb{B}^m} Y^\sigma \cdot \bigvee_{j=1}^{t} f_{\sigma,j}(X), \qquad Y^\sigma = \prod_{\sigma_i = 1} y_i,$$

where a monotone function $f_{\sigma,j}$ coincides with f_σ on the set I_j and is 0 wherever it is does not contradict monotonicity. Therefore,

$$f(X,Y) = \bigvee_{j=1}^{t} \bigvee_{s} h_{j,s}(X) \cdot \bigvee_{\sigma : f_{\sigma,j} = h_{j,s}} Y^\sigma, \tag{3.8}$$

where $h_{j,s}$ runs over a set of distinct functions $f_{\sigma,j}$ (recall that there are no more than 2^{2k+2} of them for any j).

Any function $h_{j,s}$ can be computed via a monotone DNF as a disjunction of at most $k+1$ monomials of variables X. Finally, the circuit constructed by formula (3.8) contains $2^{2k} + 2^m$ conjunctors for computing monomials of variables X and Y (Lemma 3.1), at most $k2^{2k+2}t < 2^{4k+3}$ disjunctors for computing functions $h_{j,s}$, at most $2^m t$ elements for computing the internal disjunctions in (3.8), and another $2^{2k+2}t$ internal conjunctors and external disjunctors to complete the computations. Choosing $k = n/4 - \log_2 n$, we obtain

$$C_{\mathcal{B}_M}(\mathcal{M}^n) \leqslant 2^m t + (k+2)2^{2k+2}t + 2^{2k} + 2^m \lesssim \frac{2^{m+k+1}C^{k/2}_k}{k} \sim 16 \cdot \frac{C^{n/2}_n}{n}. \tag{3.9}$$

■

In this case, a monotone function is defined independently on each of the sets $I_j \times \mathbb{B}^m$, the main complexity resides in the decoding procedure. Theorem 3.2 can also serve as an illustration of the local coding principle, but coding of a function by the table of values used in it is trivial.

• The complexity bound (3.9) is quite rough. For example, since almost all chains α in the partition of the cube $\mathbb{B}^k$ have length $O(\sqrt{k})$, for almost all sets I_j one can ensure that the sum of lengths of the chains satisfies $\sum(|\alpha| + |\beta|) = 2k - O(\sqrt{k})$. Then the bound on t will be reduced by

about half, to $t \lesssim 2^k C_k^{k/2}/k$. One can reduce the factor in bound (3.9) a little further by applying Lemma 3.3 to compute the internal disjunctions in (3.8). However, to obtain the correct complexity asymptotics [14] one needs a much more sophisticated technique.

Various applications of the local coding principle were provided by O. B. Lupanov in [205].

Circuit complexity of the multiplexor function $\boxed{c}\boxed{/_2}$

The idea of extracting common parts is useful not only in general synthesis methods, but also in computing specific functions. A simple illustration is provided by optimal circuits for the multiplexor function. The *multiplexor function* of order n is defined as $\mu_n(X;Y) = y_X$, where X is a boolean vector of n address variables, also interpreted as a number $X \in [\![2^n]\!]$, and Y is a vector of 2^n data variables y_i.

The simplest way to construct a multiplexor circuit is provided by the cascade method: decomposing in the first variable, we obtain

$$\mu_n(X;Y) = \overline{x_1} \cdot \mu_{n-1}(X';Y_0) \vee x_1 \cdot \mu_{n-1}(X';Y_1),$$

where $X = (x_1, X'), Y = (Y_0, Y_1)$. Continuing recursively, we arrive at the estimate[2)] $\mathsf{C}_{\mathcal{B}_0}(\mu_n) \leqslant 3 \cdot (2^n - 1)$. This estimate turns out to be non-optimal. The reason is that the factors — elementary conjunctions of variables $x_1, x_2, \ldots$ — independently computed on different "branches" of the cascade circuit intersect, and the computations are partially duplicated. In an optimal synthesis method proposed by P. Klein and M. Paterson [162], this drawback is eliminated.

Theorem 3.5 ([162])**.** $\mathsf{C}_{\mathcal{B}_0}(\mu_n) \lesssim 2^{n+1}$.

▶ We will need a simple lemma about the complexity of the system $\{X^\sigma \mid \sigma \in \mathbb{B}^n\}$ of all elementary conjunctions of n variables, $X^\sigma = \prod x_i^{\sigma_i}$, where x^τ is the boolean power $x^\tau = x \sim \tau = (x = \tau)$.

Lemma 3.5. $\mathsf{C}_{\mathcal{B}_0}(\{X^\sigma \mid \sigma \in \mathbb{B}^n\}) \sim 2^n$.

▷ The upper bound is obtained trivially by dividing the set of variables in half. □

Let $X = (X_0, X_1)$, $|X_0| = q$, $|X_1| = n - q$. Decompose the function μ_n in variables X_0:

$$\mu_n(X;Y) = \bigvee_{\tau \in \mathbb{B}^q} X_0^\tau \cdot \mu_{n-q}(X_1;Y_\tau), \tag{3.10}$$

where Y_τ are independent groups of 2^{n-q} variables Y. Each of the functions μ_{n-q} can be computed as

$$\mu_{n-q}(X_1;Y_\tau) = \bigvee_{\sigma \in \mathbb{B}^{n-q}} y_{\tau,\sigma} \cdot X_1^\sigma, \tag{3.11}$$

where we introduce a two-index numbering on the set of variables y_i.

All elementary conjunctions of groups of variables X_0 and X_1 are computed with complexity of order $2^q + 2^{n-q}$ according to Lemma 3.5. Another $2^{n+1} + 2^q$ operations

[2)] Or, in general, $\mathsf{C}_{\mathcal{B}}(\mu_n) \leqslant \mathsf{C}_{\mathcal{B}}(\mu_1) \cdot (2^n - 1)$ taking into account $\mu_n = \mu_1(x_1; \mu_{n-1}, \mu_{n-1})$.

are sufficient to complete the computations by formulas (3.10), (3.11). It remains to choose $q \approx n/2$. ■

In fact, the problem of constructing multiplexors is close to the problem of computing arbitrary functions, since the multiplexor function contains as subfunctions all possible boolean functions of n variables (and even $n+1$ variables, if, in addition to substituting constants, it is allowed to identify variables with other variables or their negations).

• As we see, the method of Theorem 3.5 yields the bound $\mathsf{C}_{\mathcal{B}_0}(\mu_n) \leqslant 2^{n+1}+O(2^{n/2})$. P. V. Rumyantsev [266] announced the tightness of this estimate, which means $\mathsf{C}_{\mathcal{B}_0}(\mu_n) = 2^{n+1} + \Theta(2^{n/2})$. In a wider basis, $\mathsf{C}_{\mathcal{B}_2}(\mu_n) \sim 2^{n+1}$ follows from the lower bound of W. Paul [247]. A similar bound also holds for the formula complexity of the function. The only difference is in the remainder term. In fact, the result of Theorem 3.5 follows directly from O. B. Lupanov's method of synthesis of switching circuits [202] in the form $\Phi_{\mathcal{B}_0}(\mu_n) \leqslant 2^{n+1} + O(2^n/n)$. S. A. Lozhkin and N. V. Vlasov [197] proved $\Phi_{\mathcal{B}_0}(\mu_n) = 2^{n+1}+(1\pm o(1))2^n/n$. For narrower bases, $\mathcal{B} = \{\vee, {}^-\}$ or $\mathcal{B} = \{\wedge, {}^-\}$, as V. V. Korovin [173] showed, the method of Theorem 3.5 also yields an optimal bound, $\mathsf{C}_{\mathcal{B}}(\mu_n) \sim 3 \cdot 2^n$ (you just need to express the missing conjunction or disjunction operation through basis functions).

Concerning the complexity of the system of all elementary conjunctions (Lemma 3.5), it is easy to show that $\mathsf{C}_{\mathcal{B}}(\{X^\sigma \mid \sigma \in \mathbb{B}^n\}) = 2^n + \Theta(2^{n/2})$ for $\mathcal{B} \in \{\mathcal{B}_0, \mathcal{B}_2\}$.

Chapter 4

Potential method $\boxed{U}$

The potential method is not so much an independent method of synthesis as a method of control over the key parameters of a circuit, allowing one to select the best one from a certain family of circuits. The idea is to assign to the expressions that arise in the process of calculations a simply determined numerical characteristic (potential), which turns out to be appropriately related to complexity.

Depth of circuits for summation modulo 3 $\boxed{U}\boxed{/_2}$

Consider the problem of computing the sum of n boolean variables modulo 3 over the basis $\mathcal{B}_0$. The boolean function that checks whether the sum of n boolean variables is equal to a number r modulo m will be denoted by

$$\mathrm{MOD}_n^{m,r}(X) = (x_1 + \ldots + x_n \equiv r \bmod m).$$

The operator of summation of n variables modulo m is defined as $\mathrm{MOD}_n^m = (\mathrm{MOD}_n^{m,0}, \ldots, \mathrm{MOD}_n^{m,m-1})$.

Let $X = (X^1, X^2)$, $|X| = n$, $|X^i| = n_i$. Recursive application of simple formulas [206]

$$\mathrm{MOD}_{n_1+n_2}^{m,r}(X) = \bigvee_{k=0}^{m-1} \mathrm{MOD}_{n_1}^{m,k}(X^1) \cdot \mathrm{MOD}_{n_2}^{m,r-k}(X^2) \tag{4.1}$$

by the division-in-half method leads to the bound $\mathsf{D}_{\mathcal{B}_0}\mathrm{MOD}_n^m) \leqslant (\lceil \log_2 m \rceil + 1) \log_2 n$. In particular, for $m = 3$ we obtain $\mathsf{D}_{\mathcal{B}_0}(\mathrm{MOD}_n^3) \leqslant 3 \log_2 n$. However, A. Chin [62] discovered a more economical way of computing via a partition of the set of variables into unequal parts (a simpler proof is proposed by the author in [297]).

Theorem 4.1 ([62]). $\mathsf{D}_{\mathcal{B}_0}(\mathrm{MOD}_n^3) \lesssim 2.89 \log_2 n$.

▶ Recall that a function f and its negation $\overline{f}$ have the same depth over the basis $\mathcal{B}_0$. Now note that formula (4.1) has a dual alternative:

$$\mathrm{MOD}_{n_1+n_2}^{m,r}(X) = \bigwedge_{k=0}^{m-1} \left(\mathrm{MOD}_{n_1}^{m,k}(X^1) \vee \overline{\mathrm{MOD}_{n_2}^{m,r-k}(X^2)} \right). \tag{4.2}$$

Let N_k denote the maximal n such that the functions $\mathrm{MOD}_n^{3,r}$ are representable as both conjunctions and disjunctions of formulae of depth k and $k-1$. Then from (4.1), (4.2) it follows that $N_{k+2} \geqslant N_k + N_{k-2}$, which immediately implies

$$\mathsf{D}_{\mathcal{B}_0}(\mathrm{MOD}_n^3) \leqslant 2\log_\varphi n + O(1) < 2.89\log_2 n + O(1),$$

where $\varphi = \frac{1+\sqrt{5}}{2}$. ■

The proof, although not entirely explicit, exploits the potential function $d \to \varphi^{d/2}$, which indicates the approximate number of terms in the sum that can be computed with depth d.

• In [62], Theorem 4.1 was proved in a more complicated way in terms of communication complexity. For the depth of the summation operator modulo 5, the method yields the bound $\mathsf{D}_{\mathcal{B}_0}(\mathrm{MOD}_n^5) \lesssim 3.48\log_2 n$ [62]. Applying special formulas with partitioning the set of variables into three groups, the author [297] improved the result of Theorem 4.1 to $\mathsf{D}_{\mathcal{B}_0}(\mathrm{MOD}_n^3) \lesssim 2.8\log_2 n$. In [297], a general method is also proposed that allows to decrease the depth of summation modulo small primes, see below on p. 89.

Formula complexity of linear functions in a ternary basis $\boxed{U}$

Consider the problem of minimizing the size of formulae for the linear function $\Lambda_n = x_1 \oplus x_2 \oplus \ldots \oplus x_n$ over the basis $\mathcal{B}_3 = \{\mathrm{maj}_3(x,y,z),\ \overline{x},\ 1\}$[1)].

Since any function from $\mathcal{U}_2$ is read-once expressible in $\mathcal{B}_3$, $\Phi_{\mathcal{B}_3}(\Lambda_n) \leqslant \Phi_{\mathcal{U}_2}(\Lambda_n) \preccurlyeq n^2$ trivially holds. An illustration of the more powerful expressive capabilities of the basis $\mathcal{B}_3$ is the formula [65]

$$x \oplus y \oplus z = \mathrm{maj}_3(x,\ \mathrm{maj}_3(\overline{x},y,z),\ \mathrm{maj}_3(\overline{x},\overline{y},\overline{z})). \qquad (4.3)$$

Instead of variables, you can substitute some linear functions into it, which, for $n = p+q+r$, will lead to the relation

$$\Phi_{\mathcal{B}_3}(\Lambda_n) \leqslant 3\Phi_{\mathcal{B}_3}(\Lambda_p) + 2\Phi_{\mathcal{B}_3}(\Lambda_q) + 2\Phi_{\mathcal{B}_3}(\Lambda_r). \qquad (4.4)$$

Uri Zwick
Tel Aviv University, since 1991

When choosing $p = q = r$ we obtain $\Phi_{\mathcal{B}_3}(\Lambda_{3n}) \leqslant 7\Phi_{\mathcal{B}_3}(\Lambda_n)$, hence, $\Phi_{\mathcal{B}_3}(\Lambda_n) \preccurlyeq n^{\log_3 7} < n^{1.78}$. But we can do even better by taking advantage of the non-uniform dependence of formula (4.3) on the variables.

Theorem 4.2 ([65]). $\Phi_{\mathcal{B}_3}(\Lambda_n) \prec n^{1.74}$.

[1] In general, we consider the basis $\mathcal{U}_k$, which serves as a generalization of the basis $\mathcal{U}_2$ to the set of k-ary boolean functions. The basis $\mathcal{U}_k$ includes functions that are monotonically non-increasing or monotonically non-decreasing in each variable, i.e. exactly those functions from which it is impossible to obtain a linear function of two variables by substituting constants, inversions, and identifications of variables; for more details, see [60, 65]. It is easy to check that the basis $\mathcal{B}_3$ is equivalent to the basis $\mathcal{U}_3$, which means $\Phi_{\mathcal{B}_3}(f) = \Phi_{\mathcal{U}_3}(f)$ for any boolean function f.

▶ We will look for a complexity estimate in the form

$$\Phi_{\mathcal{B}_3}(\Lambda_n) \leqslant c\, n^{\mu}, \tag{4.5}$$

applying (4.4) recursively with the choice $q = r = \gamma n$ and $p = (1 - 2\gamma)n$ (here γ, μ are unknown parameters). Let us relax an assumption that p, q, r should be integers for a while. To derive (4.5), we need an induction base (small values of n), which is provided by the choice of the constant c, and the induction step, which is provided by substituting estimates (4.5) into (4.4) if only the inequality

$$3(1 - 2\gamma)^{\mu} + 4\gamma^{\mu} \leqslant 1 \tag{4.6}$$

holds. In order to minimize the exponent μ, we choose $\gamma \approx 0.4$ and finally obtain $\Phi_{\mathcal{B}_3}(\Lambda_n) \prec n^{1.74}$. ■

• Taking into account the condition $p, q, r \in \mathbb{Z}$ does not change anything in essence, it only leads to a more cumbersome calculation. For example, you can look for an estimate in the form

$$\Phi_{\mathcal{B}_3}(\Lambda_n) \leqslant c_1 n^{\mu} - c_2 n \tag{4.7}$$

assuming $n \geqslant n_0$.

▷ Let $q = r = \lfloor \gamma n \rfloor$. Then $|q - \gamma n| \leqslant 1/2$ and $|p - (1 - 2\gamma)n| \leqslant 1$. Note that for any $a > 0$, $\mu \geqslant 1$ and $x \in [0,\, 1]$ the inequality $(a + x)^{\mu} \leqslant a^{\mu} + \mu(a + 1)x$ holds. Let $x = 1/n$. Now the induction step is ensured by the inequality

$$\begin{aligned} n^{-\mu}(3\Phi_{\mathcal{B}_3}(\Lambda_p) + 4\Phi_{\mathcal{B}_3}(\Lambda_q)) \leqslant \\ 3c_1(1 - 2\gamma + x)^{\mu} + 4c_1(\gamma + x/2)^{\mu} - c_2(3p + 4q)n^{-\mu} \leqslant \\ c_1(3(1 - 2\gamma)^{\mu} + 4\gamma^{\mu} + 3(2 - 2\gamma)\mu x + 2(1 + \gamma)\mu x) - c_2((3 - 2\gamma)n^{1-\mu} - n^{-\mu}) \leqslant \\ c_1 - c_2(3 - 2\gamma - x)n^{1-\mu} + c_1(8 - 4\gamma)\mu x \leqslant c_1 - c_2 n^{1-\mu}, \end{aligned}$$

if we choose $c_2 = \beta c_1$, where $\beta = (8 - 4\gamma)\mu/(1 - 2\gamma)$. The threshold n_0 is chosen such that $n_0^{\mu-1} \geqslant \beta + 1$ (then the right-hand side of (4.7) is not less than $c_1 n$). Finally, the choice of the parameter c_1 secures the base of induction, namely, the fulfillment of (4.7) for all $n \in [n_0,\, n_1]$, where $n_1 = (n_0 + 1)/(1 - 2\gamma)$ (for example, $c_1 = 2n_1$ will do, since obviously $\Phi_{\mathcal{B}_3}(\Lambda_n) \leqslant \Phi_{\mathcal{U}_2}(\Lambda_n) < 2n^2$). □

The above bound has not been improved yet. In [243] it is shown that a better bound cannot be obtained using only (4.3). This will be discussed next (see p. 49). In [302] the author proved a lower bound $\Phi_{\mathcal{B}_3}(\Lambda_n) \succ n^{1.53}$.

Depth of circuits for multiple addition $\boxed{U}$

In the school method, multiplication of n-digit numbers is reduced to addition of n numbers (generally, $2n$-digit numbers). The natural idea to perform addition by a tree of usual adders, even parallel ones, leads to a circuit of only $\Omega(\log^2 n)$ depth. Designs of logarithmic depth circuits are based on the idea of compressors, proposed at the turn of the 1950s and 60s independently by many researchers. The essence of the idea is to calculate intermediate sums in a certain sense "not completely".

Let us consider the problem in its general formulation as computing the sum of n elements in some commutative group $(G, +, 0)$. (k, l)-*compressor* is a circuit that performs the transform $(x_1, \ldots, x_k) \to (y_1, \ldots, y_l)$ with the condition of preserving the sum: $\sum_{i=1}^{k} x_i = \sum_{j=1}^{l} y_j$, where $k > l$.

A circuit composed of (k, l)-compressors[2] allows us to reduce the summation of n elements to the summation of l elements (we assume that the group unit $0 \in G$ is available as an additional input of the circuit). An arbitrary (k, l)-compressor A can be characterized by the mapping $\mathbf{a} \to \mathbf{b}$ of the depth vector[3] of inputs $\mathbf{a} = (a_1, \ldots, a_k)$ to the depth vector of outputs $\mathbf{b} = (b_1, \ldots, b_l)$. Relying on the physical meaning of the problem, we assume $\max_i a_i < \max_j b_j$ for any $\mathbf{a}$ (i.e., none of the compressor inputs is also an output).

Michael Stewart Paterson
University of Warwick, since 1971

For instance, a family of parallel copies of the adder FA_3 constitutes an integer $(3, 2)$-compressor[4]. If the depths of the compressor inputs are 0, 0, 1, then the outputs are computed at depths 2 and 3, see Fig. 1a). An example of a circuit built from such compressors is shown in Fig. 4.1 (compressors are depicted as polygonal blocks with three inputs and two outputs; the intermediate sums are arranged according to the depth scale).

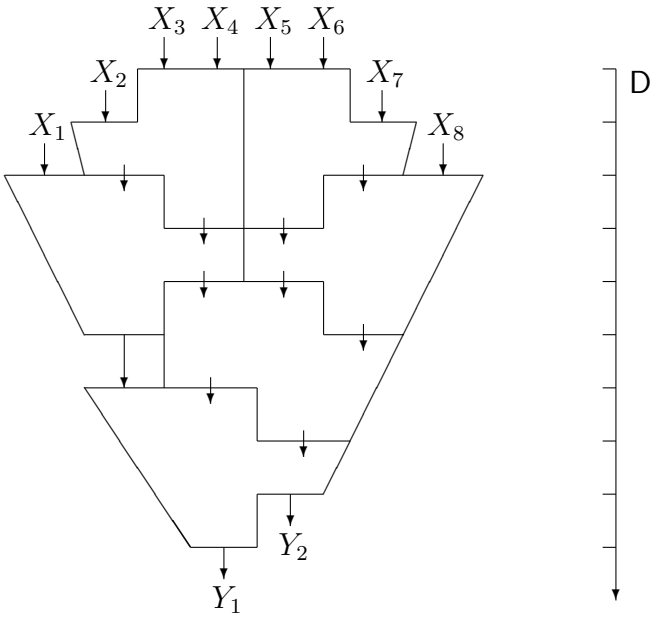


Figure 4.1: Multiple addition circuit of $(3, 2)$-compressors [243]

The question of how to arrange compressors in a multiple addition circuit to minimize its depth appeared to be nontrivial. It was not until the early 1990s that M. Paterson, N. Pippenger, and U. Zwick [243] proposed a general approach to solving this problem.

[2]The circuit is a tree in the (non-strict) sense that the outputs of intermediate elements do not branch.

[3]The term "depth" is rather conventional here. We assume that the numbers a_i are integers only for convenience of reasoning.

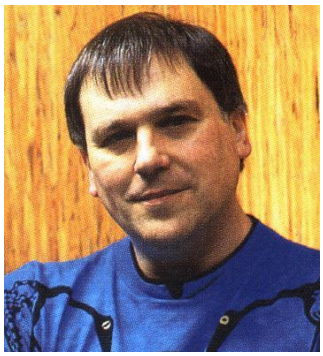
[4]Indeed, $X+Y+Z = U+V$ holds if the digits of numbers satisfy relations $x_i+y_i+z_i = 2u_{i+1}+v_i$ for all i, and we assume $u_0 = 0$.

Let us denote the minimum possible depth of an (n,l)-compressor composed of (k,l)-compressors A by $D_A(n)$. The potential method allows to bound this depth from below. Let $g_A(\mathbf{a};x)$ denote the characteristic polynomial of a compressor A:

$$g_A(\mathbf{a};x) = \sum_{j=1}^{l} x^{b_j} - \sum_{i=1}^{k} x^{a_i}. \tag{4.8}$$

Lemma 4.1 ([243]). *Let $\lambda(\mathbf{a})$ be the maximal positive root of a polynomial $g_A(\mathbf{a};x)$. Denote $\lambda = \sup_{\mathbf{a}\in\mathbb{Z}^k} \lambda(\mathbf{a})$. Then*

$$D_A(n) \geqslant \log_\lambda(n/l).$$



Nicholas John Pippenger
University of British Columbia, 1988 to 2003

▷ First of all, we note that the polynomial $g_A(\mathbf{a};x)$ has the maximal root satisfying $\lambda(\mathbf{a}) > 1$, since $g_A(\mathbf{a};1) = l-k < 0$, and $g_A(\mathbf{a};x) \to +\infty$ as $x \to +\infty$. Then the supremum $\lambda > 1$ also exists due to $\lambda(\mathbf{a}) \leqslant k$, since as $x \geqslant k$,

$$g_A(\mathbf{a};x) \geqslant x^{\max_j b_j} - k \cdot x^{\max_i a_i} \geqslant (x-k) \cdot x^{\max_i a_i} \geqslant 0.$$

Consider a step-by-step procedure for constructing a circuit from compressors, from inputs to outputs, and follow the change in the set of depths of the terms of intermediate sums. Initially, we have n inputs of depth 0. When adding a next compressor, k numbers from the list (the depths of the compressor inputs) are replaced by l other numbers (the depths of the compressor outputs).

We assign a numerical value (potential) λ^d to an intermediate term located at depth d. By the definition of λ, the sum of the potentials of the terms does not decrease when adding compressors. Therefore, $n \leqslant l \cdot \lambda^{D_A(n)}$. □

If a compressor is not just an abstract device but a boolean circuit with depth defined in the usual way, then the upper bound λ in the condition of Lemma 4.1 is achieved for some $\mathbf{a}$, see [243].

The proof of the lower bound serves as a guide to obtaining the upper bound.

Theorem 4.3 ([243]). *Let λ be the maximal positive root of a polynomial $g_A(\mathbf{a};x)$. Then*

$$D_A(n) \leqslant \log_\lambda n + O(1).$$

► Without loss of generality, we can assume $\min_i a_i = a_1 = 0$. Let $b = \max_j b_j$. Under the level of a compressor (in a circuit composed of compressors) we understand the depth of its first input.

Let us construct a circuit of compressors A such that for any d it contains

$$C_d = \begin{cases} \left\lceil n\lambda^{-d} + \frac{l}{k-l} \right\rceil, & -b < d \leqslant \log_\lambda n \\ 0, & \text{otherwise} \end{cases}$$

compressors of level d. Compressor inputs can be inputs of the circuit and outputs of other compressors, computed at appropriate depths. Compressor inputs that do not come from the outputs of other compressors are considered to be circuit inputs. Compressor outputs that are not connected to the inputs of other compressors are considered to be circuit outputs.

A compressor of level d receives inputs at depths $a_i + d$ and produces outputs at depths $b_j + d$. By construction, the total number of inputs of depth d required by compressors of the circuit is $\sum_{i=1}^{k} C_{d-a_i}$, and the number of outputs produced by compressors at the same depth is $\sum_{i=1}^{l} C_{d-b_j}$.

Let us establish several properties of the circuit.

a) At depth d, $0 \leqslant d \leqslant \log_\lambda n$, the circuit has no outputs. Indeed, the difference between the number of inputs and the number of outputs of the compressors at depth d is positive:

$$\sum_{i=1}^{k} C_{d-a_i} - \sum_{j=1}^{l} C_{d-b_j} > \sum_{i=1}^{k} \left(n\lambda^{a_i-d} + \frac{l}{k-l} \right) - \sum_{j=1}^{l} \left(n\lambda^{b_j-d} + \frac{l}{k-l} + 1 \right) = -n\lambda^{-d} g_A(\mathbf{a}; \lambda) = 0. \quad (4.9)$$

b) The circuit has more than n inputs at depth 0. Indeed, for $d = 0$, due to $C_{-b} = 0$, in expression (4.9) the difference between the left (input) and right (ouput) parts is at least $n\lambda^b > n$.

c) The total number of outputs of the circuit at depths greater than $\log_\lambda n$ is $O(1)$ (the circuit may also have outputs at negative depths). Indeed, this number does not exceed the total number of outputs of compressors of levels $d > \log_\lambda n - b$, i.e.,

$$l \sum_{d > \log_\lambda n - b} C_d < lb \left(n\lambda^{b - \log_\lambda n} + \frac{l}{k-l} + 1 \right) = lb \left(\lambda^b + k \right).$$

Thus, if n variables x_i are connected to the inputs of the circuit at depth 0, and the constant $0 \in G$ is supplied to the remaining inputs, then the constructed circuit will be an (n, m)-compressor, $m = O(1)$ (outputs at negative depth do not count — they implement $0 \in G$). By adding several more compressors A to the circuit, the number of outputs will be reduced to l. Finally, the total depth of the resulting circuit is $\log_\lambda n + O(1)$. ■

The characteristic polynomial of the mentioned above integer $(3, 2)$-compressor FA_3 is

$$g_{FA_3}(0, 0, 1; x) = x^3 + x^2 - x - 2. \quad (4.10)$$

Its only positive root is $\lambda \approx 1.2056$. Therefore, for the depth of the operator $\Sigma_{m,n}$ of addition of n m-digit numbers, with the use of Corollary 2.2 we obtain

Corollary 4.1. $\mathsf{D}_{\mathcal{B}_2}(\Sigma_{m,n}) \leqslant 3.71 \log_2 n + (1 + o(1)) \log_2(m + \log_2 n)$.

Corollary 4.2. $\mathsf{D}_{\mathcal{B}_2}(M_n) \lesssim 4.71 \log_2 n$.

• Exploiting a slightly more complex $(6,3)$-compressor, Paterson and Zwick [245] obtained the bound $\mathsf{D}_{\mathcal{B}_2}(\Sigma_{1,n}) \lesssim 3.57\log_2 n$, and E. Grove [117], applying a special $(7,3)$-compressor, established the bound $\mathsf{D}_{\mathcal{B}_0}(\Sigma_{1,n}) \lesssim 4.94\log_2 n$.

The authors of [243] also extended the method to the complexity of multiple addition formulae. Theorem 4.2 serves as a simple example of the application of this method (formula (4.3) describes a $(3,1)$-compressor in $(\mathbb{B},\oplus)$). In general, a compressor A used for constructing formulae is characterized by the mapping $\mathbf{a}\to\mathbf{b}$ of the vector of input sizes $\mathbf{a}=(a_1,\ldots,a_k)$ to the vector of output sizes $\mathbf{b}=(b_1,\ldots,b_l)$. The characteristic function of the compressor is defined as

$$g_A(\mathbf{a};x)=\sum_{j=1}^{l} b_j^x-\sum_{i=1}^{k} a_i^x.$$

The maximal positive root of the function is denoted by $\mu(\mathbf{a})$. Let $\mu=\sup_{\mathbf{a}\in\mathbb{Z}^k}\mu(\mathbf{a})$. Then the minimal complexity of an (n,l)-compressor composed of compressors A is $(n/l)^{1/\mu+o(1)}$. The synthesis method is similar to the method of Theorem 4.3, but for classification of summands the rounded logarithm of the formula size $\lceil\log_2 s\rceil$ is used instead of the depth (and rounding issues lead to slightly more cumbersome calculations).

In the paper [245] the authors proposed special compressor constructions to obtain the bounds $\Phi_{\mathcal{B}_2}(\Sigma_{1,n})\prec n^{3.13}$ and $\Phi_{\mathcal{B}_0}(\Sigma_{1,n})\prec n^{4.57}$ by this method. Stronger bounds are obtained with the use of a special composite encoding of symmetric functions, see below on p. 92.

Parallel restructuring of arithmetic formulae $\boxed{U}$

Richard Peirce Brent
Australian National University, Canberra, since 1972

In the problem of parallel evaluation of an arithmetic expression, which is of practical interest, it is required, given a formula of length n, to construct an equivalent[5] formula of the smallest possible depth (in a binary basis, it is desirable to be as close as possible to $\log_2 n$). It is known that in any complete boolean basis and in general arithmetic bases there always exists an equivalent formula of depth $O(\log n)$. This follows from the results of V. M. Khrapchenko (announced in [336]) and R. Brent [47], respectively. A more precise form of the relationship between depth and complexity depends on the basis. So, the concept of the uniformity constant[6] $c_{\mathcal{B}}$ of a basis $\mathcal{B}$ is introduced. It is defined as the supremum of numbers c for which $\Phi_{\mathcal{B}}(f)\leqslant(c+o(1))\log_2\mathsf{D}_{\mathcal{B}}(f)$ holds, where $\Phi_{\mathcal{B}}(f)\to\infty$. In the study of uniformity constants of arithmetic bases, we assume that a semiring over which the computations are performed is commutative.

For simplicity, we restrict ourselves to considering formulae over the monotone arithmetic basis $\mathcal{A}_+$. The uniformity of this basis (i.e., the existence of the constant $c_{\mathcal{A}_+}$) was proved by R. Brent, D. Kuck, and K. Maruyama in [49].

Lemma 4.2 ([49]). *Let a read-once formula of complexity n over $\mathcal{A}_+$ compute a function $f(X,y)$. Then $f(X,y)=f_1(X)\cdot y+f_2(X)$, where $\Phi_{\mathcal{A}_+}(f_1),\Phi_{\mathcal{A}_+}(f_2)\leqslant n$.*

[5] That is, computing the same function.
[6] Proposed by V. M. Khrapchenko [158].

▷ The proof is trivial by induction on the complexity of a formula. □

Theorem 4.4 ([49]). *For any monotone polynomial f,*

$$\mathsf{D}_{\mathcal{A}_+}(f) < 2.47 \log_2 \Phi_{\mathcal{A}_+}(f) + O(1).$$

▶ Consider a recurrent number sequence:

$$N_0 = 1, \quad N_1 = 2, \quad N_2 = 3, \qquad N_k = N_{k-2} + N_{k-3}, \quad k \geqslant 3.$$

We will prove by induction that a formula of length $\leqslant N_k$ can be reconstructed into a formula of depth $\leqslant k$. For $k \leqslant 2$ the statement is obvious. Let us prove the induction step. We will need a simple

Lemma 4.3. *For any $m \leqslant n$, in an arbitrary binary formula of complexity n, there is a subformula of complexity $\geqslant m$, the principal subformulas[7] of which have complexity $< m$.*

▷ It is enough to search the formula tree in the direction from the root to leaves, moving along subformulas of complexity $\geqslant m$ as long as possible. The last subformula in the chain will be the desired one. □

Without loss of generality, we can assume that the formula is read-once (the case of repeated inputs is reducible to the one under consideration). Consider a formula F of complexity N_k and, relying on Lemma 4.3, select a subformula G of complexity $\geqslant N_{k-3}+1$, which has the form $G_1 \circ G_2$, where $\Phi(G_1), \Phi(G_2) \leqslant N_{k-3}$ and $\circ \in \{+, \cdot\}$, see Fig. 4.2.

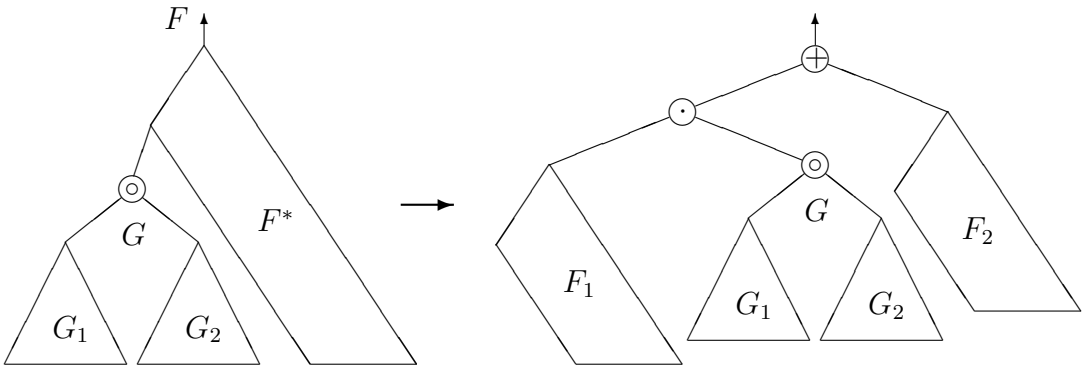


Figure 4.2: Restructuring of an arithmetic formula

Let the formulae F, G_1, G_2 implement functions (polynomials) f, g_1, g_2, respectively. Let $f^*(X, y)$ denote the function into which $f(X)$ turns when substituting a new variable y into the formula F instead of the subformula G. By construction,

[7] Recall that the principal subformulas of a formula $F_1 \circ F_2$ are simply the formulae F_1, F_2.

$\Phi(f^*) \leqslant \Phi(F) - \Phi(G) + 1 \leqslant N_{k-2}$. According to Lemma 4.2, $f^*(X, y) = f_1 \cdot y + f_2$, where $\Phi(f_1), \Phi(f_2) \leqslant N_{k-2}$. Finally, the function f can be computed as

$$f = f_1 \cdot (g_1 \circ g_2) + f_2$$

with depth $\leqslant k$, since by the induction hypothesis, $\mathsf{D}(g_i) \leqslant k-3$ and $\mathsf{D}(f_i) \leqslant k-2$. It remains to note that $N_k \asymp a^k$, where $a \approx 1.325$ is the root of the equation $x^3 = x+1$. ■

• Neither adding subtraction to the basis nor expanding it to a full basis changes the conclusion of Theorem 4.4 (for example, because subtractions and multiplications by constants can always be moved to the level of inputs). Thus, $c_{\mathcal{A}} \leqslant c_{\mathcal{A}_+} < 2.47$. A more sophisticated method of decomposing formulae with the analysis of several cases allowed S. R. Kosaraju [176] to improve the estimate to $c_{\mathcal{A}_+} \leqslant 2$. At the same time, according to the result of D. Coppersmith and B. Schieber [73], $c_{\mathcal{A}_+} \geqslant 1.5$.

David Eugene Muller
University of Illinois,
1953 to 1992

The method of restructuring arithmetic expressions also applies to boolean monotone formulae. In fact, $c_{\mathcal{B}_0} \leqslant c_{\mathcal{B}_M} \leqslant c_{\mathcal{A}_+}$ (the basis $\mathcal{B}_M$ is a special case of an arithmetic basis; the inequality for the basis $\mathcal{B}_0$ holds due to De Morgan's laws, which allow negations to be lowered to the input level). However, the mutual distributivity of the boolean basis operations significantly extends the possibilities of transforming boolean formulae, as we will see in the result due to F. Preparata and D. Muller [251].

Theorem 4.5 ([251]). *For any boolean function f,*

$$\mathsf{D}_{\mathcal{B}_0}(f) < 1.82 \log_2 \Phi_{\mathcal{B}_0}(f) + O(1). \qquad (4.11)$$

▶ In view of the possibility of lowering negations in formulae to the input level (De Morgan's laws), it suffices to prove the relation (4.11) for the monotone basis $\mathcal{B}_M$. The key point is the following observation, which strengthens Lemma 4.2 for the boolean case.

Lemma 4.4 ([251]). *Let a read-once formula G of complexity n over $\mathcal{B}_m$ compute a function $f(X, y)$. Then the function f can be represented either as $f(X, y) = f_1(X) \cdot y \vee f_2(X)$, or as $f(X, y) = (f_1(X) \vee y) \cdot f_2(X)$, where $\Phi_{\mathcal{B}_M}(f_1) < n/2$ and $\Phi_{\mathcal{B}_M}(f_2) < n$.*

▷ In the tree representing a formula G, select the path from the variable y to the root, see Fig. 4.3. We can write

$$G(X, y) = (\ldots((y \circ_1 G_1) \circ_2 G_2)\ldots) \circ_k G_k,$$

where G_i are formulae attached along the path.

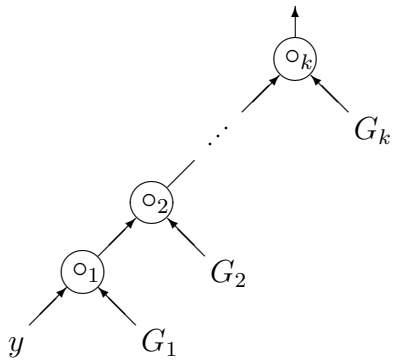


Figure 4.3: Decomposition of a boolean formula

Then the function f is implemented by any of the two formulas

$$G^{\wedge} \cdot y \vee G(X,0), \qquad (G^{\vee} \vee y) \cdot G(X,1), \qquad \text{where} \quad G^{\wedge} = \bigwedge_{\circ_i = \wedge} G_i, \quad G^{\vee} = \bigvee_{\circ_i = \vee} G_i.$$

Since $\Phi(G^{\wedge}) + \Phi(G^{\vee}) = n - 1$, one of the formulas satisfies the conditions of the lemma. □

Next we follow the proof of Theorem 4.4. Now a number sequence is defined as

$$N_0 = 1, \quad N_1 = 2, \quad N_2 = 3, \qquad N_k = N_{k-1} + N_{k-3}, \quad k \geqslant 3.$$

We will prove that a formula of size $\leqslant N_k$ can be reconstructed into a formula of depth $\leqslant k$ (it remains to prove the induction step).

Let a read-once formula F of complexity N_k compute a function $f(X)$. According to Lemma 4.3, we can select a subformula G of complexity $\geqslant N_{k-3} + 1$, which has the form $G_1 \circ G_2$, where $\Phi(G_1), \Phi(G_2) \leqslant N_{k-3}$ and $\circ \in \{\vee, \wedge\}$. Let $f^*(X, y)$ denote the function into which $f(X)$ turns when a new variable y is substituted into the formula F instead of the subformula G. By construction, $\Phi(f^*) \leqslant \Phi(F) - \Phi(G) + 1 \leqslant N_{k-1}$. Representing f^* by Lemma 4.4, we finally obtain

$$f = (f_1 \circ_1 (g_1 \circ g_2)) \circ_2 f_2,$$

where $\circ, \circ_1, \circ_2 \in \{\vee, \wedge\}$, $\Phi(f_2) < N_{k-1}$ and $\Phi(f_1) < N_{k-1}/2 \leqslant N_{k-2}$. Thus, by the induction hypothesis, $\mathsf{D}(g_i) \leqslant k - 3$, $\mathsf{D}(f_1) \leqslant k - 2$ and $\mathsf{D}(f_2) \leqslant k - 1$. Therefore, $\mathsf{D}(f) \leqslant k$. It remains to note that $N_k \asymp a^k$, where $a \approx 1.465$ is the root of the equation $x^3 = x^2 + 1$. ■

• The bound $c_{\mathcal{B}_M} < 1.82$ of Theorem 4.5 was improved by V. M. Khrapchenko to $c_{\mathcal{B}_M} < 1.73$ in [157]. The lower bound $c_{\mathcal{B}_M} > 1.06$ was obtained by the author in [300]. For complete bases of arithmetic type, for example, $\mathcal{A}$, $\mathcal{B}_0$, nontrivial lower bounds on the uniformity constants are unknown. For non-arithmetic bases, such bounds exist: in particular, for the complete boolean basis $\mathcal{B} = \{\backslash\}$ containing the single "Sheffer stroke" operation, the bound $c_{\mathcal{B}} \geqslant 2$ was proved by Khrapchenko in [154].

Chapter 5

Method of approximations $\boxed{\mathcal{E}}$

> The approximation method is based on the following observation: it is often advantageous to solve the subproblems into which the original problem is divided not completely but approximately, with controlled accuracy. An exact solution to the original problem is constructed from approximate solutions to the subproblems.

Fast integer division $\boxed{\varepsilon}\boxed{/_2}$

In computing practice, equations $f(x) = 0$ are often solved by the Newton—Raphson method (the tangent method) by iterating $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$. If the initial value x_0 is chosen well, and the function $f(x)$ satisfies certain simple constraints, then $x_i \to x^*$, where $f(x^*) = 0$, and the rate of convergence of the method is generally quadratic. Around 1966, S. Cook [69] noted that this method can be adapted for fast division of numbers.

Stephen Arthur Cook
University of Toronto, since 1970

Since division reduces to inversion and multiplication, and methods for fast multiplication are well known, it is sufficient to discuss the inversion operation. We define the $(n, n+1)$-operator of (approximate) inversion I_n on the set of n-digit numbers from the interval $[1/2, 1]$ by the condition $|I_n(a) - 1/a| \leqslant 2^{-n}$ (the definition is ambiguous and in fact defines a whole family of operators). By restricting ourselves to the interval $[1/2, 1]$, we mean that the general case when $a \notin [1/2, 1]$ reduces to the considered one by replacing a with $2^k a$ and then multiplying the result by 2^k (these operations are implemented by shifting the position of the binary point, which is done for free in the circuit model). Recall that $\mathsf{M}(n)$ denotes the smoothed complexity function of multiplication.

Theorem 5.1 ([69]). $\mathsf{C}(I_n) \preccurlyeq \mathsf{M}(n)$.

▶ The proof follows closely the version of J. Håstad [123]. Let $a_{..k}$ denote the number a with truncated digits below the k-th after the binary point — this is an approximation to a with accuracy 2^{-k}. We define the sequence r_i as follows[1)]:

$$r_0 = 1, \qquad \tilde{r}_{i+1} = 2r_i - a_{..4+2^i} \cdot r_i^2, \qquad r_{i+1} = (\tilde{r}_{i+1})_{..4+2^{i+1}}. \tag{5.1}$$

Lemma 5.1. $|1 - r_i a_{..4+2^i}| < 2^{-2^{i-1}-1/2}$.

▷ Obviously, the inequality is valid for $i = 0$. Let us prove the induction step from i to $i + 1$.

By the induction hypothesis,

$$0 \leqslant 1 - \tilde{r}_{i+1} a_{..4+2^i} = (1 - r_i a_{..4+2^i})^2 \leqslant 2^{-2^i-1}.$$

Hence, $0 < \tilde{r}_{i+1} \leqslant r_{i+1} \leqslant a^{-1}_{..4+2^i} \leqslant 2$. Now the inequality of the lemma follows from the calculation:

$$\begin{aligned}
&|1 - r_{i+1} a_{..4+2^{i+1}}| \leqslant \\
&\quad |1 - \tilde{r}_{i+1} a_{..4+2^i}| + |r_{i+1} a_{..4+2^i} - \tilde{r}_{i+1} a_{..4+2^i}| + |r_{i+1} a_{..4+2^{i+1}} - r_{i+1} a_{..4+2^i}| \leqslant \\
&\qquad 2^{-2^i-1} + a_{..4+2^i} \, |r_{i+1} - \tilde{r}_{i+1}| + r_{i+1} \, |a_{..4+2^{i+1}} - a_{..4+2^i}| \leqslant \\
&\qquad\qquad 2^{-2^i-1} + 1 \cdot 2^{-2^{i+1}-4} + 2 \cdot 2^{-2^i-4} < \frac{11}{16} \cdot 2^{-2^i} < 2^{-2^i-1/2}.
\end{aligned}$$

□

As a consequence of the lemma, we obtain

$$|1 - r_i a| \leqslant |1 - r_i a_{..4+2^i}| + r_i \, |a - a_{..4+2^i}| < 2^{-2^{i-1}-1/2} + 2^{-2^i-4} < 2^{-2^{i-1}}.$$

To compute a^{-1} with an accuracy of 2^{-n}, it is sufficient to determine r_i up to $i = \lceil \log_2 n \rceil + 1$. Calculation by formulas (5.1) leads to the estimate

$$\mathsf{C}(I_n) \leqslant \sum_{i=0}^{\lceil \log_2 n \rceil + 1} (2\mathsf{C}(M_{4+2^i}) + O(2^i)) \preccurlyeq \mathsf{M}(n).$$

■

• In a similar way, circuits are constructed for some other elementary numerical functions, for example, for the square root, see, e.g., [48].

Fast division with remainder of complex polynomials $\boxed{\varepsilon}\boxed{s}$

The method of successive approximations for polynomials works even better than for numbers, allowing similar problems to be solved more efficiently.

An elegant method of reducing the operation of division of polynomials with remainder to simpler operations — division modulo x^n and multiplication — was proposed by V. Strassen in [314]. Let $QR^R_{2n,n}$ and D^R_n denote operations with polynomials in $R[x]$: respectively, the operator computing the quotient and remainder of the division of a polynomial of degree $\leqslant 2n - 1$ by a polynomial of degree n and the operator of division of polynomials modulo x^n.

[1)] Guided by the tangent method for finding a zero of the function $f(x) = a - 1/x$.

Lemma 5.2 ([314]). $\mathsf{C}_{\mathcal{A}}(QR^R_{2n,n}) \leqslant \mathsf{C}_{\mathcal{A}}(D^R_n) + \mathsf{C}_{\mathcal{A}}(M^R_n) + O(n)$.

▷ Let $q(x)$ and $r(x)$ be the quotient and remainder of $a(x)$ divided by $b(x)$, where $\deg a \leqslant 2n-1$, $\deg b = n$, and $\deg q, r < n$. Then

$$a(x) = q(x)b(x) + r(x). \tag{5.2}$$

By $\widetilde{a}(x)$, $\widetilde{q}(x)$, $\widetilde{b}(x)$, $\widetilde{r}(x)$ we denote, respectively, the polynomials $x^{2n-1}a(1/x)$, $x^{n-1}q(1/x)$, $x^n b(1/x)$ and $x^{n-1}r(1/x)$. Substituting $1/x$ in place of x in (5.2) and multiplying by x^{2n-1}, we obtain

$$\widetilde{a}(x) = \widetilde{q}(x)\widetilde{b}(x) + x^n\widetilde{r}(x). \tag{5.3}$$

Then

$$\widetilde{q} = \widetilde{a}/\widetilde{b} \bmod x^n, \qquad \widetilde{r} = (\widetilde{a} - \widetilde{q}\,\widetilde{b})/x^n. \tag{5.4}$$

Calculations by formulas (5.4) are reduced to division modulo x^n, multiplication and subtraction of polynomials. □

Joris van der Hoeven
École Polytechnique, Paris, since 2009

Division can be done by inversion modulo x^n and multiplication. Inversion is performed by a polynomial analog of the numerical method described in the previous section. In fact, it is more advantageous in practice and theory to apply the approximation method directly to the division operation. In the most interesting case of field $\mathbb{C}$ (and $\mathbb{R}$ as well), given that the multiplication of complex polynomials is implemented via the DFT, the complexity of the method can be conveniently expressed in terms of the complexity of the DFT. Let us describe the fastest known method, due to J. van der Hoeven [129].

Theorem 5.2 ([129]). *Let the constant c_F satisfy the condition $\mathsf{C}_{\mathcal{A}}(\mathrm{DFT}_N) \leqslant c_F N \log_2 N$ for any $N = 2^k$. Then*

$$\mathsf{C}_{\mathcal{A}}(QR_{2n,n}) \lesssim 12c_F \cdot n \log_2 n.$$

Note that the complexity of multiplication is trivially estimated as $\mathsf{C}_{\mathcal{A}}(M_n) \lesssim 6c_F \cdot n \log_2 n$ (we can ignore the fact that n may be not close to a power of two)[2]. Thus, the complexity of division with remainder is approximately equal to the complexity of two multiplications. According to Theorem 2.4, for the basis $\mathcal{A}^{\mathbb{C}}$ we can set $c_F = 1.5$.

▶ We keep the notation of Lemma 5.2. Let $(k+1)m > n \geqslant km$. Divide the polynomials under consideration into blocks of length m:

$$\widetilde{a} = a_0 + a_1x^m + a_2x^{2m} + \ldots, \quad \widetilde{b} = b_0 + b_1x^m + b_2x^{2m} + \ldots, \quad \widetilde{q} = q_0 + q_1x^m + q_2x^{2m} + \ldots$$

[2] One can always choose an approximation to n of the form $b2^k$, where b is small enough. Then Lemma 2.1 applies.

Next, let $(a)^*$ denote the vector[3] $\mathrm{DFT}_{2m}(a)$.

Let $b_0^{-1} = 1/b_0 \bmod x^m$. From the condition $\widetilde{b}\,\widetilde{q} = \widetilde{a} \bmod x^n$ we obtain a recurrence formula for expressing the blocks of the quotient $\widetilde{q}$. For $0 \leqslant j < k$, we have

$$a_j = \left(q_j b_0 + \sum_{i=0}^{j-1} (q_i b_{j-i} + \lfloor q_i b_{j-i-1}/x^m \rfloor) \right) \bmod x^m. \tag{5.5}$$

Assuming $b_{-1} = 0$ and introducing the notation[4] $\beta_i = b_{i-1} + b_i x^m$ for $0 \leqslant i \leqslant k+1$, we derive

$$q_j = b_0^{-1} \left(a_j - \sum_{i=0}^{j-1} \lfloor q_i \beta_{j-i}/x^m \rfloor \right) \bmod x^m, \qquad 0 \leqslant j < k. \tag{5.6}$$

For $j = k$, formulas (5.5), (5.6) are valid modulo x^{n-km}.

Consider the following algorithm:

I. Find q_0. To do this, compute b_0^{-1}, then $(b_0^{-1})^*$ and $(a_0)^*$, then $(a_0 b_0^{-1})^*$, and finally (via the inverse DFT) restore $q_0 = a_0 b_0^{-1} \bmod x^m$.

II. Then, by formula (5.6), we sequentially determine blocks q_j. For each $j = 1, \ldots, k$:

1) Compute $(\beta_j)^*$ and $(q_{j-1})^*$. Let

$$\alpha = q_0 \beta_j + q_1 \beta_{j-1} + \ldots + q_{j-1} \beta_1. \tag{5.7}$$

2) Compute the vector $(\alpha)^*$ from the known $(\beta_i)^*$ and $(q_i)^*$, following (5.7).

3) Via the inverse DFT, compute the polynomial $\alpha \bmod (x^{2m} - 1)$. Note that its leading m coefficients coincide with the middle m coefficients of α. Denote

$$\gamma = a_j - \lfloor \alpha/x^m \rfloor \bmod x^m.$$

4) Successively computing γ, $(\gamma)^*$ and $(b_0^{-1}\gamma)^*$, we finally find $q_j = b_0^{-1}\gamma \bmod x^m$ for $j < k$ and $q_k = b_0^{-1}\gamma \bmod x^{n-km}$.

III. Thus, the quotient $\widetilde{q} = \sum_{i=0}^{k} q_i x^{im} \bmod x^n$ has been found; it remains to find the remainder $\widetilde{r}$. According to (5.4), for this it is enough to compute the missing part of the product $\widetilde{q}\,\widetilde{b}$. Write

$$\widetilde{q}\,\widetilde{b} = c_0 + c_1 x^m + c_2 x^{2m} + \ldots + c_{2k+1} x^{(2k+1)m},$$

where $\deg c_i < m$. By construction, $c_j = a_j$ for $0 \leqslant j < k$ and $c_k \equiv a_k \bmod x^{n-km}$. The remaining blocks c_j may be computed as

$$c_j = \left\lfloor \sum_i q_i \beta_{j-i}/x^m \right\rfloor \bmod x^m. \tag{5.8}$$

[3] This refers to the polynomial interpretation of the DFT, when the transform is applied to the vector of coefficients of a polynomial.

[4] The polynomials β_i are introduced only for notational convenience.

For the boundary block c_k, formula (5.8) provides the missing $(k+1)m-n$ leading coefficients.

For calculations by formulas (5.8) it is required to compute $(\beta_{k+1})^*$, $(q_k)^*$, for all $j=k,\dots,2k+1$ find $(\sum_i q_i\beta_{j-i})^*$, and restore the sums $\sum_i q_i\beta_{j-i} \bmod (x^{2m}-1)$, from which the blocks c_j are extracted. Finally, we obtain $\widetilde{r}=(\widetilde{a}-\widetilde{q}\,\widetilde{b})/x^n$.

Let us estimate the complexity of the algorithm. It starts with inversion modulo x^m. Then, at all stages, $3k+4$ DFTs and $3k+3$ inverse DFTs of order $2m$ are performed. The complexity of the remaining operations, among which the computations of the Fourier transforms of sums (5.7) and (5.8) dominate, is estimated as $O(mk^2)$.

It is advisable to choose the parameters $m=2^t$ and $k \asymp \sqrt{\log n}$. The initial inversion can be performed by any algorithm of complexity $O(m\log m)$, for example, a polynomial analogue of Cook's method from the previous section. As a result, we obtain

$$\mathsf{C}_{\mathcal{A}}(QR_{2n,n}) \leqslant (6k+7)c_F\cdot 2m\log_2(2m)+O(m(k^2+\log m)) = 12c_F\cdot n\log_2 n+O(n\sqrt{\log n}).$$

■

Note that, unlike the method of Theorem 5.1, where the accuracy was doubled at each step, the method of Theorem 5.2 is based on iterations (5.6) with a fixed accuracy increment in order to reduce the specific weight of inversions.

• Separately for the inversion operation modulo x^n, the best known upper complexity bound $(7.5+o(1))c_F\cdot n\log_2 n$ was established by the author in [290].

Matrix multiplication. Border rank $\boxed{\varepsilon}$

Dario Andrea Bini
University of Pisa, since 1990

Theorem 2.3 is based on a special method of multiplying 2×2 matrices and can be easily generalized. The *rank* $\mathrm{rk}^R T$ of a system of bilinear forms $T(X,Y)$ over a semiring R is the minimum number r for which the following representation is possible:

$$T=\sum_{l=1}^{r} C_l X_l Y_l, \tag{5.9}$$

where C_l is a vector with components in R, and X_l, Y_l are linear combinations over R of the variables X and Y, respectively. For example, $\mathrm{rk}^F MM_2=7$ in any field F [332]. Now Theorem 2.3 can be extended as

Theorem 5.3. *Let $r=\mathrm{rk}\,MM_m$ in a ring R. Then*

$$\mathsf{C}_{\mathcal{A}^R}(MM_n) \leqslant r\left(1+\frac{6rm^2}{r-m^2}\right)n^{\log_m r}.$$

▶ Consider some representation of the form (5.9) for the operator MM_m involving r nonscalar multiplications and s linear operations. It is certainly true that $s \leqslant 6rm^2$, since any sum X_l or Y_l requires at most $2m^2$ linear operations, and any element of the matrix product is a linear combination of r products X_lY_l, hence, it is computed in $2r$ linear operations.

It can be easily verified by induction that

$$\mathsf{C}(MM_{m^h}) \leqslant \left(1 + \frac{s}{r - m^2}\right) r^h - \frac{s}{r - m^2} m^{2h} \tag{5.10}$$

(split an $m^h \times m^h$ matrix into submatrices of size $m^{h-1} \times m^{h-1}$). Finally, for $m^{h-1} < n \leqslant m^h$ we obtain

$$\mathsf{C}(MM_n) \leqslant \mathsf{C}(MM_{m^h}) \leqslant \left(1 + \frac{s}{r - m^2}\right) r^h \leqslant r \left(1 + \frac{s}{r - m^2}\right) n^{\log_m r}.$$

■

The *border rank* $\underline{\mathrm{rk}}\, T$ is the minimum number r for which the following representation is possible:

$$u^d T = \sum_{l=1}^{r} C_l(u) X_l(u) Y_l(u) \bmod u^{d+1}, \tag{5.11}$$

where $d \in \mathbb{N}_0$, $C_l(u) \in R[u]^{\dim T}$, and X_l, Y_l are linear combinations over $R[u]$ of the variables X and Y, respectively, $\dim T$ is the number of forms in the system T. If we imagine that $u \to 0$, then formulas (5.11) when dividing by u^d define approximate matrix multiplication. In fact, approximate computation was the goal of the work of Italian mathematicians [33], in which the concept of border rank actually appears. Then D. Bini [32] observed that from formulas (5.11) one can effectively pass to exact matrix multiplication.

Further, formulas of type (5.11) will be called (d, r)-*representations*.

Theorem 5.4 ([32])**.** *Let R be a ring and $m = const$. Then*

$$\mathsf{C}_{\mathcal{A}^R}(MM_n) \preccurlyeq n^{\log_m(\underline{\mathrm{rk}}\, MM_m) + o(1)}.$$

▶ The key point of the proof is Bini's lemma (we prove it in a weakened form).

Lemma 5.3. *If a system T has a (d, r)-representation, then* $\mathrm{rk}\, T \leqslant C_{d+2}^2 \cdot r$.

▷ Let T be represented by formulas (5.11). Write

$$C_l(u) = \sum_{i=0}^{d} C_{l,i} u^d, \qquad X_l(u) = \sum_{i=0}^{d} X_{l,i} u^d, \qquad Y_l(u) = \sum_{i=0}^{d} Y_{l,i} u^d,$$

where $C_{l,i}$ are vectors over R, and $X_{l,i}$, $Y_{l,i}$ are linear combinations of variables over R. Then from (5.11) we obtain

$$T = \sum_{l=1}^{r} \sum_{a+b+c=d} C_{l,a} X_{l,b} Y_{l,c}.$$

It remains to note that the inner sum contains C^2_{d+2} terms. □

We will also need a simple lemma on the composition of (d,r)-representations. The *tensor product* of bilinear systems T_1 and T_2 is defined as follows. If $T_1 = \sum C_{i,j} x_i y_j$, where $C_{i,j} \in R^{\dim T_1}$, then

$$T_1 \otimes T_2 = \sum C_{i,j} \otimes T_2(X^i, Y^j), \tag{5.12}$$

where X^i, Y^j are independent groups of variables[5]. In particular,

$$MM_{m,p,q} \otimes MM_{m',p',q'} = MM_{mm',pp',qq'}, \tag{5.13}$$

where $MM_{m,p,q}$ is the operator of multiplication of $m \times p$ and $p \times q$ matrices (over R).

Lemma 5.4. *If a system T_1 has a (d_1, r_1)-representation and a system T_2 has a (d_2, r_2)-representation, then $T_1 \otimes T_2$ admits a $(d_1 + d_2, r_1 r_2)$-representation.*

▷ With the use of the given representation for T_1, write

$$u^{d_1}(T_1 \otimes T_2) = \sum_{l=1}^{r_1} C_l(u) \otimes T_2(X_l(u), Y_l(u)) \bmod u^{d_1+1}, \tag{5.14}$$

where $C_l(u) \in R[u]^{\dim T_1}$, and $X_l(u)$, $Y_l(u)$ are linear combinations[6] of vectors of variables X^i and Y^j over $R[u]$. Next, substituting into (5.14) the representations for T_2 and multiplying by u^{d_2}, we obtain

$$u^{d_1+d_2}(T_1 \otimes T_2) = \sum_{l=1}^{r_1} C_l(u) \otimes \left(\sum_{s=1}^{r_2} C'_s(u) X_{l,s}(u) Y_{l,s}(u) \right) \bmod u^{d_1+d_2+1} =$$
$$= \sum_{l=1}^{r_1} \sum_{s=1}^{r_2} (C_l(u) \otimes C'_s(u)) X_{l,s}(u) Y_{l,s}(u) \bmod u^{d_1+d_2+1},$$

where $C'_s(u) \in R[u]^{\dim T_2}$, and $X_{l,s}(u)$, $Y_{l,s}(u)$ are linear combinations of components of vectors $X_l(u)$ and $Y_l(u)$, i.e. ultimately just linear combinations of variables. □

Now we complete the proof of the theorem. Let $r = \underline{\mathrm{rk}}\, MM_m$. Applying Lemma 5.4 h times to the (d,r)-representation for MM_m, we derive an (hd, r^h)-representation for MM_{m^h}. Then from Lemma 5.3 we deduce $\mathrm{rk}\, MM_{m^h} \leqslant (hd+2)^2 r^h$. By Theorem 5.3 we finally obtain

$$\mathsf{C}(MM_n) \leqslant 7(hd+2)^4 r^{2h} m^{2h} n^{\log_m r + \frac{2}{h} \cdot \log_m (hd+2)}.$$

A suitable choice of the parameter is $h \asymp \sqrt{\log n \log\log n}$. ■

A simple illustration of the advantage that the consideration of the border rank provides is the following example by A. Schönhage [278].

[5] The symbol $\otimes$ on the right-hand side of (5.12) denotes the Kronecker product of matrices (in this case, vectors). The *Kronecker product* of $m \times n$ matrix $A = (a_{i,j})$ and $p \times q$ matrix B is defined as the $mp \times nq$ matrix obtained by substituting $a_{i,j}B$ for $a_{i,j}$ into A.

[6] Recall that due to bilinearity $T_2(aX^1 + bX^2, Y) = aT_2(X^1, Y) + bT_2(X^2, Y)$ and $T_2(X, aY^1 + bY^2) = aT_2(X, Y^1) + bT_2(X, Y^2)$.

Theorem 5.5 ([278]). *If R is a ring, then* $\mathsf{C}_{\mathcal{A}^R}(MM_n) \preccurlyeq n^{\log_3 21+o(1)} \prec n^{2.772}$.

▶ Let us show that $\underline{\mathrm{rk}}\, MM_3 \leqslant 21$. The entries z_{ij} of the product $Z = XY$ of 3×3 matrices can be determined by the formulas

$$\begin{aligned} u^2 z_{jj} &= (x_{j1} + u^2 x_{j2})(y_{2j} + u^2 y_{1j}) + v_{jj} - w_j \mod u^3, \\ u^2 z_{ij} &= (x_{j1} + u^2 x_{i2})(y_{2j} - u y_{1i}) + v_{ij} - w_j + u(v_{ji} - v_{ii}) \mod u^3, \quad i \neq j, \\ v_{jj} &= (x_{j1} + u^2 x_{j3}) y_{3j}, \qquad v_{ij} = (x_{j1} + u^2 x_{i3})(y_{3j} + u y_{1i}), \qquad w_j = x_{j1}(y_{2j} + y_{3j}), \end{aligned}$$

involving 21 nonscalar multiplications. It remains to apply Theorem 5.4. ■

• Considering 2×2 matrices does not improve the estimates of Theorem 2.3 due to $\underline{\mathrm{rk}}\, MM_2 = \mathrm{rk}\, MM_2 = 7$ [184]. On the other hand, for the rank of 3×3 matrix multiplication, only the bound $\mathrm{rk}\, MM_3 \leqslant 23$ is known so far [181]. A. V. Smirnov [308] showed that $\underline{\mathrm{rk}}\, MM_3 \leqslant 20$ (but the corresponding representation is more difficult to describe) — when substituting this estimate into Theorem 5.5, we establish $\mathsf{C}_{\mathcal{A}^R}(MM_n) \prec n^{2.727}$.

The bound of Lemma 5.3 can be refined to $\mathrm{rk}\, T \leqslant (2d+1)r$ for a field F of cardinality at least $2d+2$ [32].

Monotone sorting circuits $\boxed{\varepsilon}\boxed{P}\boxed{\because}$

Soon after K. Batcher [16] proposed an elegant way to construct monotone circuits for sorting n inputs with complexity $O(n \log^2 n)$, E. Lamagna and J. Savage [183] proved the lower bound $\mathsf{C}_{\mathcal{B}_M}(\mathrm{SORT}_n) \succcurlyeq n \log n$. Finally, another 10 years later, M. Ajtai, J. Komlós, and E. Szemerédi [4, 5] obtained the tight bounds $\mathsf{C}_{\mathcal{B}_M}(\mathrm{SORT}_n) \asymp n \log n$ and $\mathsf{D}_{\mathcal{B}_M}(\mathrm{SORT}_n) \asymp \log n$ inventing a construction now known as AKS-circuits. The method exploits several ideas at once, the central one being the idea of approximate computations. It turned out to be advantageous to construct sorting circuits from subcircuits that perform sorting approximately, with a controlled error probability.

The original method [4, 5], like most of its modifications, is difficult to describe and analyze. We will present a relatively simple version composed by J. Seiferas [282] in development of M. Paterson's approach [241].

Endre Szemerédi
Institute of Mathematics, Hungarian Acad. Sci., Budapest, since 1965

The circuits considered below belong to a particular model of *comparator circuits* (in the terminology of [168], sorting networks). The comparator circuit receives n elements of a linearly ordered set as inputs and, by pairwise comparison operations $x, y \to \min(x, y), \max(x, y)$ (in the boolean case, $x, y \to xy, x \vee y$), produces at the outputs a permutation of the inputs in accordance with some partial order. The outputs of internal comparator subcircuits do not branch. In a sense, a comparator circuit is almost a formula. The circuit naturally decomposes into layers of parallel comparators. For more details, see [168].

Let us introduce the concept of approximate sorting. Assume $0 < \varepsilon \leqslant 1$ and $0 < \lambda \leqslant 1/2$. A comparator circuit on n inputs is called a (λ, ε)-*separator* if for any

$m \leqslant \lambda n$ the circuit places at least $(1-\varepsilon)m$ of the m largest elements among the λn right outputs and at least $(1-\varepsilon)m$ of the m smallest elements — among the λn left outputs. The following two lemmas establish the existence of separators of constant depth (and hence of linear complexity).

Lemma 5.5 ([5]). *For any $\varepsilon > 0$ and any n, there exists a $(1/2, \varepsilon)$-separator*[7] *on $2n$ inputs of depth $O(1/\varepsilon^3)$ as $\varepsilon \to 0$.*

▷ If n is small, say, $n < 64/\varepsilon^3$, then apply any sorting circuit. Therefore, we further assume that $n \geqslant 64/\varepsilon^3$.

We will show that the required circuit can be composed of several layers of comparators, where at each layer the elements from the junior (left) and senior (right) halves are compared according to a randomly selected matching. An example of a circuit is shown in Fig. 5.1a.

Let us associate such a circuit with a bipartite (n, n)-graph: the vertices of one part correspond to the positions of elements from the junior half, and the other – from the senior half. Two vertices of the graph are connected by an edge if a comparison of the corresponding elements was performed at some layer of the circuit, see Fig. 5.1b. In other words, the graph is a composition of matchings that define the layers of the circuit.

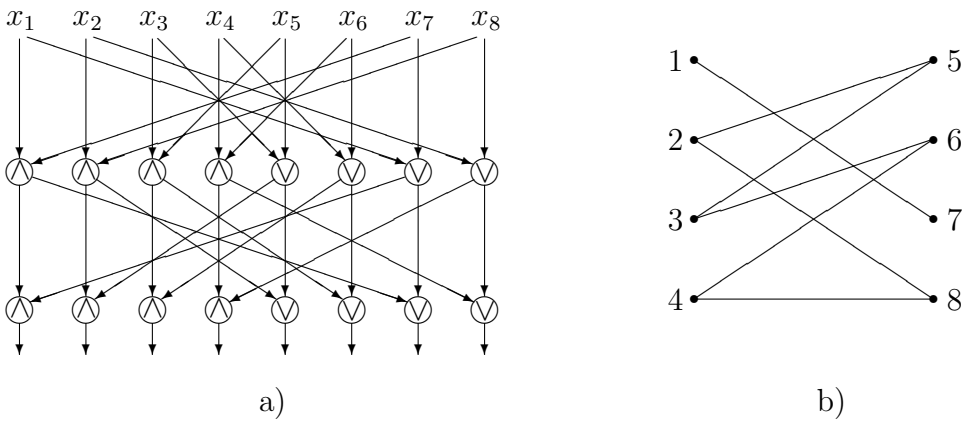


Figure 5.1: Comparator circuit (a) and its graph (b)

I. First, we prove that a circuit is a $(1/2, \varepsilon)$-separator if its graph is an (ε, α)-*expander*, $\alpha = 1/\varepsilon - 1/2$, which means: for every $m \leqslant \varepsilon n$, any set of m vertices in one part is connected by edges to at least αm vertices in the other part.

Note that if two nodes are connected by an edge in the graph, then the functions computed at the corresponding outputs of the circuit satisfy the relation $\leqslant$ (one part collects only the minima of ordered pairs, and the other only the maxima).

Now suppose that the circuit is not a separator, i.e., for some input set, say, among $k \leqslant n$ largest elements $p > \varepsilon k$ are placed into the junior half. Consider the set of nodes in the graph corresponding to the latter elements. In this set $m = \min\{p, \lfloor \varepsilon n \rfloor\}$

[7] $(1/2, \varepsilon)$-separators are also called ε-halvers.

nodes are connected by edges with at least αm nodes in the other part. This means that the minimum of p elements is inferior to at least $p-1+\alpha m$ other elements. But for $m = p$ we have

$$p - 1 + \alpha m = p - 1 + m/\varepsilon - m/2 > p/\varepsilon - 1 > k - 1,$$

and for $m = \lfloor \varepsilon n \rfloor < p$,

$$p-1+m/\varepsilon-m/2 > (p-1)/2+m/\varepsilon > (\varepsilon n-1)(1/\varepsilon+1/2) > (1+\varepsilon/2)n-1/\varepsilon-1 \geqslant n-1.$$

This contradicts the fact that the chosen element is among the k largest.

II. It remains to prove that a bipartite graph composed of a suitable number $r = r(\varepsilon)$ of random matchings[8] is an (ε, α)-expander with a positive probability.

Note that a graph is an (ε, α)-expander if it does not contain empty (i.e. edgeless) $(k, n - \alpha k)$-subgraphs for any $k \leqslant \varepsilon n$. The probability that a random matching does not intersect (by edges) a given $(k, n - \alpha k)$-subgraph is $C^k_{\alpha k}/C^k_n$. Then the probability P_k that r random matchings do not intersect at least one of the $(k, n-\alpha k)$-subgraphs is estimated with the help of simple relations $\frac{1}{4\sqrt{k}}(\frac{en}{k})^k \leqslant C^k_n \leqslant (\frac{en}{k})^k$ as

$$P_k \leqslant 2C^k_n C^{\alpha k}_n \left(\frac{C^k_{\alpha k}}{C^k_n}\right)^r \leqslant 2(4\sqrt{k})^r \left(\frac{e^{1+\alpha} n^{1+\alpha} (\alpha k)^r}{k^{1+\alpha} \alpha^\alpha n^r}\right)^k =$$

$$2(4\sqrt{k})^r \left(\alpha e^{1+\alpha} \left(\frac{\alpha k}{n}\right)^{r-\alpha-1}\right)^k \leqslant \left(c_2 \left(\frac{c_1 \alpha k}{n}\right)^{r-\alpha-1}\right)^k,$$

where $c_1 = (4\sqrt{k})^{1/k} \leqslant 4$ and $c_2 = 2^{1/k}\alpha(c_1 e)^{1+\alpha} \leqslant (8e)^{1+\alpha}$.

For $k \leqslant \varepsilon n/4$, we have $c_1 \alpha k \leqslant (1 - \varepsilon/2)n$. Otherwise $k \geqslant \varepsilon n/4 \geqslant 16/\varepsilon^2$, so $c_1 = e^{\ln(16k)/2k} < 1+\ln(16k)/k \leqslant 1+\frac{\varepsilon^2}{8} \ln \frac{16}{\varepsilon} \leqslant 1+\varepsilon/2$. Therefore, $c_1 \alpha k < (1-\varepsilon^2/4)n$. In either case, if r is chosen somewhat larger than $\alpha+4\ln(2c_2)/\varepsilon^2$, we obtain $P_k < 2^{-k}$. Then the probability that the graph under consideration is not an (ε, α)-expander does not exceed $\sum_k P_k < 1$. □

• The circuit whose existence is established by the lemma has depth $r \asymp 1/\varepsilon^3$ as $\varepsilon \to 0$. A more careful argument allows us to refine the estimate to $r \preccurlyeq \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}$, see, e.g., [241].

It is easy to check that any r-regular (all vertices have degree r) bipartite graph is a union of r matchings, so a circuit can be constructed from a graph. Many explicit constructions of regular expander graphs are also known, for example, [212, 95].

Lemma 5.6 ([241]). *For any constant $\varepsilon > 0$, any $\lambda < 1/2$ and n, there exists a (λ, ε)-separator on $2n$ inputs of depth $O(\log^4(1/\lambda))$ as $\lambda \to 0$.*

[8] The distribution is uniform: all matchings are equally probable.

Miklós Ajtai
IBM Research center,
San Jose, 1983 to 2015

▷ Let $s = \lfloor 2\lambda n \rfloor$ and $k = \lceil \log_2(1/\lambda) \rceil$. The circuit is composed of $k+1$ layers of $(1/2, \varepsilon_0)$-separators (the parameter ε_0 will be chosen later): on the first layer — a separator on $2n$ inputs, on the next two layers — two separators on the left and right for $2^{k-1}s$ marginal elements[9], on the next layer — separators for $2^{k-2}s$ elements from each end, and so on up to the last layer of two separators on $2s$ marginal elements, see Fig. 5.2.

Of the $m \leqslant s$ largest (smallest) elements, the separator of the first layer allocates in the "wrong" half no more than $\varepsilon_0 m$ elements. In the next pair of separators, regardless of their order: the separator of the right (left) $2^{k-1}s$ elements leaves a maximum of $\varepsilon_0 m$ elements beyond the outermost interval, and the separator on the other side definitely does not worsen the characteristics of cutting off the largest (smallest) elements[10]. In each of the subsequent layers, the separator on the corresponding side erroneously throws out no more than $\varepsilon_0 m$ elements from the outermost interval.

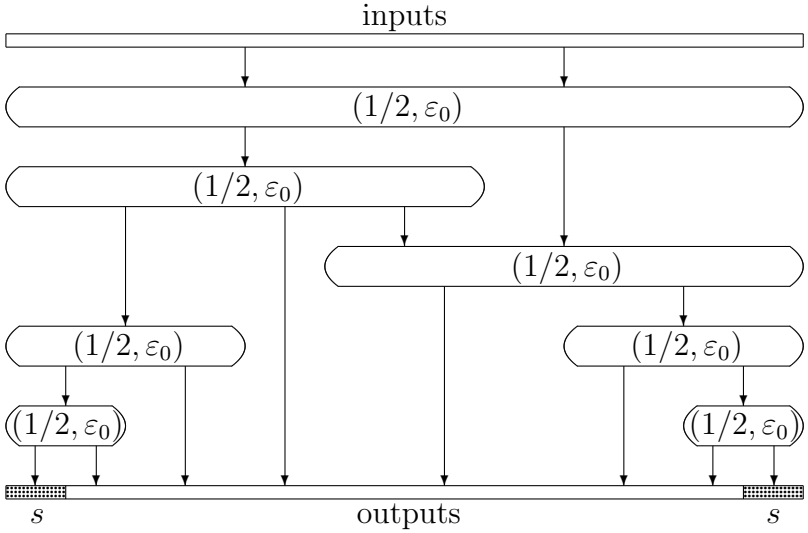


Figure 5.2: Construction of a (λ, ε)-separator

Thus, no more than $k\varepsilon_0 m$ largest (smallest) elements can miss the s extreme right (left) outputs of the circuit. Therefore, it is sufficient to choose $\varepsilon_0 = \varepsilon/k$. Now the circuit depth estimate follows from Lemma 5.5. □

Theorem 5.6 ([4, 5]). *Sorting n elements may be performed by a comparator circuit of depth $O(\log n)$.*

[9] If $2^{k-1}s = n$, then these two separators can be placed in parallel on the same layer.

[10] In any comparator circuit, the set of m largest (smallest) elements moves strictly to the right (left).

► For simplicity of reasoning we assume $n = 2^k$. Let $\mu, \varepsilon > 0$ be parameters to be chosen later. The sorting circuit is constructed from layers of parallel (λ, ε)-separators, where λ is determined individually for each layer, and $\lambda \geqslant \mu$. It is convenient to imagine the circuit functioning in time: transformations of one layer are performed in one time unit. We consider the action of a separator layer as a rearrangement of elements in a structure associated with the complete binary tree of depth k. At each vertex of the tree there is a container for storing elements.

At the initial time $t = 0$ all n elements are in the root container. Then, at any time, the elements of each non-empty container are subjected to approximate sorting by a separator: elements from the outermost intervals are sent to the parent node, the remaining ones are distributed equally between the containers of child nodes, in the direction to the leaves.

We define the *capacity* of a container at depth d at time t as $B_{d,t} = na^d\nu^t$, where the constant parameters $a > 1$ and $\nu < 1$ will be specified later. If the container stores b elements, then in the case $\mu B_{d,t} < b/2$ the container's contents, except for a possible odd element, are ordered by a composition of $(1/2, \varepsilon)$- and (λ, ε)-separators, $\lambda = \mu B_{d,t}/b$, after which $\lfloor \mu B_{d,t} \rfloor$ elements from each end, as well as the odd element (if any), are sent to the parent node above. The remaining elements are moved down one level: the left half — to the container of the left child node, the right half — to the container of the right one. Otherwise, in the case $\mu B_{d,t} \geqslant b/2$, no separation is performed — all elements should be returned to the parent container. The scheme of migration of elements is shown in Fig. 5.3. The root container is an exception — its elements are rearranged by a $(1/2, \varepsilon)$-separator, divided equally into two parts (the number of elements in the container is necessarily even), which are sent to the corresponding containers of the child nodes.

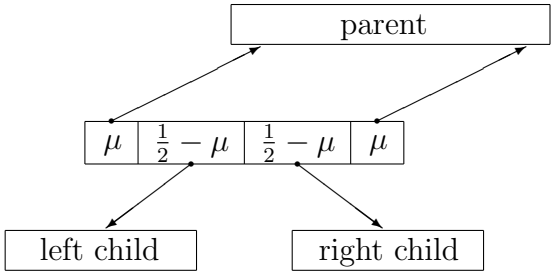


Figure 5.3: Scheme of migration of elements

The process continues until all elements are at the bottom $O(1)$ levels of the tree; the exact termination condition is stated as $B_{k,t} < 1/\mu$. After that, sorting circuits are applied in each subtree of depth $O(1)$. The constructed circuit contains $k \log_{1/\nu}(2a) + O(1) \asymp \log n$ layers of separators and therefore has the desired depth. It remains to check the correctness of the algorithm.

First of all, note that at any time moment, the containers of the same level are filled equally. In particular, this is why the root container always has an even number

of elements. In addition, a leaf container cannot keep more than one element[11]. Hence, the described procedure does not throw elements outside the tree.

It is also clear from the construction that at any given time, all elements are concentrated either at even or at odd levels of the tree.

I. By induction on t we prove that under certain conditions on the parameters a, ε, μ, ν the number of elements in any container never exceeds its capacity. The statement is obvious for $t = 0$.

Consider an arbitrary container at depth d, empty at time $t-1$. The number of elements in it at the next time t in the case $B_{d,t} \geqslant a\nu$ does not exceed

$$2(2\mu B_{d+1,t-1} + 1) + B_{d-1,t-1}/2 = B_{d,t}(4\mu a + 1/(2a))/\nu + 2 < B_{d,t}(4\mu a + 3/a)/\nu,$$

which is less than $B_{d,t}$ subject to

$$\underline{\nu \geqslant 4\mu a + 3/a}. \tag{5.15}$$

In the remaining case $B_{d,t} < a\nu$ at time $t-1$ all containers at higher levels $d' < d$ are empty, because $B_{d',t-1} < 1$. This means that all elements are at levels $d+1$ and lower. In this case $B_{d+1,t-1} < a^2$, and under the additional assumption

$$\underline{a^2 = 1/\mu} \tag{5.16}$$

we conclude that $d + 1 \neq k$ (otherwise the process of constructing the circuit would have already been completed). This means that in each subtree rooted at a node of depth $d+1$ at time $t-1$ there are an even number of elements, therefore, an even number of elements are at the root of the subtree (this applies to the child nodes with respect to the one under consideration). Thus, at most $4\mu B_{d+1,t-1} = 4\mu a B_{d,t}/\nu < B_{d,t}$ elements are sent to a container of depth d at time t according to (5.15).

János Komlós
Rutgers University,
since 1988

II. The proof of the correctness of the algorithm is based on the evaluation of the number of irrelevant elements in each container. We assume that when placed in the tree leaves, elements should obey to ascending order from left to right. For any element, the *native* vertices are the tree leaf in which the element should be located after ordering, as well as all vertices on the path from the tree root to this leaf. During the execution of the algorithm, an element of some container will be called an *r-stranger* if it is at a distance $\geqslant r$ along the tree edges from the nearest native vertex.

By induction on t we check that at time t the number of r-strangers ($r \geqslant 1$) in a container of depth d does not exceed $\varepsilon^{r-1}\mu B_{d,t}$ for an appropriate choice of parameters. The induction base is trivial, since at time $t = 0$ all elements are in the root container, native to them. We are going to prove the induction step.

III. First, let us consider the simpler case $r \geqslant 2$ (then we can assume $d \geqslant 2$). The r-strangers of a container of depth d can include $(r+1)$-strangers from containers of

[11] Such elements should go up. However, in reality, the process of constructing the circuit will finish even earlier.

child nodes and $(r-1)$-strangers from the parent container at the previous moment of time. Taking into account filtering, their number is bounded from above as

$$2\varepsilon^r \mu B_{d+1,t-1} + \varepsilon(\varepsilon^{r-2}\mu B_{d-1,t-1}) = \varepsilon^{r-1}\mu B_{d,t}(2\varepsilon a + 1/a)/\nu,$$

i.e., does not exceed $\varepsilon^{r-1}\mu B_{d,t}$ provided

$$\underline{\nu \geqslant 2\varepsilon a + 1/a}. \tag{5.17}$$

IV. Now consider the case $r = 1$. In a vertex v at time t strangers can arrive from two sources: 2-strangers from child vertices, and from the parent vertex — both strangers and elements native to its other child vertex v' at the previous moment of time. The number of strangers from child vertices can be easily estimated as $2\varepsilon\mu B_{d+1,t-1} = 2\varepsilon\mu a B_{d,t}/\nu$. The parent container requires a more careful analysis.

Let the parent container at time $t-1$ contain q elements, of which q_0 are native elements for v, q_1 are native elements for v', and q_2 are strangers. If $q_0 \geqslant q/2$, then the number of strangers for the vertex v sent to its container is estimated as $\varepsilon(q_1 + q_2) \leqslant \varepsilon q/2$, i.e., as the sum of errors of a (λ, ε)-separator that did not send strangers upward, and a $(1/2, \varepsilon)$-separator that sent elements native to v' to a wrong half. Otherwise, if $q_0 < q/2$, then this number should be estimated as $\varepsilon q/2 + (q/2 - q_0)$, where the second term takes into account native elements of v', which end up in the container of v even after correct sorting. Let us estimate $q/2 - q_0$.

Consider a special hypothetical distribution of elements into containers at time $t-1$ with the same number of elements in each container as in the real distribution. Sort and distribute all elements uniformly among the nodes at level d (where the vertices v and v' reside), and then arbitrarily move elements up and down the tree to fill all containers correctly, but so that the parent node of v and v' receives $\lceil q/2 \rceil$ and $\lfloor q/2 \rfloor$ elements from these child nodes, respectively.

In the considered distribution, the subtree rooted at vertex v contains only elements native to it, and the container of the parent vertex contains $\geqslant q/2$ elements native to v. Let us estimate the maximum number of elements native to v that can be moved from this container to any other. This will yield an estimate for $q/2 - q_0$.

For containers of the same level $d-1$: for one container the specified elements will be 1-strangers, for two — 2-strangers, for four — 3-strangers, etc. The total number of positions available for placement at this level is estimated as

$$\mu\left(1 + 2\varepsilon + (2\varepsilon)^2 + \ldots\right) B_{d-1,t-1} < \mu B_{d-1,t-1}/(1 - 2\varepsilon). \tag{5.18}$$

At an arbitrary higher level $d-h$, for one container the elements in question will be native, for another — 1-strangers, for two more — 2-strangers, for four — 3-strangers, etc. The total number of available positions at these levels (for odd $h \geqslant 3$) is estimated as

$$\left(1 + \mu(1 + 2\varepsilon + (2\varepsilon)^2 + \ldots)\right) \sum_{i=1}^{d/2} B_{d-2i-1,t-1} < \left(1 + \frac{\mu}{1-2\varepsilon}\right) \sum_{i \geqslant 1} a^{-2i} B_{d-1,t-1} = \left(1 + \frac{\mu}{1-2\varepsilon}\right) \frac{B_{d-1,t-1}}{a^2 - 1}. \tag{5.19}$$

Since there are no more free positions to fill in the subtree of the vertex v, at any lower level $d+h$ there are available 2^h containers[12] for which the specified elements will be $(h+1)$-strangers, 2^{h+1} containers for which these elements will be $(h+2)$-strangers, etc. The total number of available positions does not exceed

$$\sum_{i=1}^{(k-d)/2} \mu\left((2\varepsilon)^{2i-1} + (2\varepsilon)^{2i} + \ldots\right) B_{d+2i-1,t-1} < \mu \sum_{i\geqslant 1} \frac{(2\varepsilon)^{2i-1}}{1-2\varepsilon}\, a^{2i-1} B_{d,t-1} = \frac{2\varepsilon a \mu B_{d,t-1}}{(1-2\varepsilon)(1-(2a\varepsilon)^2)}. \quad (5.20)$$

Summing up (5.18), (5.19), (5.20), we obtain

$$q/2 - q_0 < \left(\frac{1}{a^2-1} + \frac{\mu a^2}{(1-2\varepsilon)(a^2-1)} + \frac{2\varepsilon a^2 \mu}{(1-2\varepsilon)(1-(2a\varepsilon)^2)}\right) B_{d-1,t-1}.$$

As a consequence, the total number of strangers in the container of vertex v at time t, including those coming from child vertices, is estimated as

$$\left(2\varepsilon a\mu + \frac{1}{a^3-a} + \frac{\mu a}{(1-2\varepsilon)(a^2-1)} + \frac{2\varepsilon a\mu}{(1-2\varepsilon)(1-(2a\varepsilon)^2)}\right) B_{d,t}/\nu,$$

which does not exceed $\mu B_{d,t}$ provided

$$\nu \geqslant 2\varepsilon a + \frac{1}{\mu(a^3-a)} + \frac{a}{(1-2\varepsilon)(a^2-1)} + \frac{2\varepsilon a}{(1-2\varepsilon)(1-(2a\varepsilon)^2)}. \quad (5.21)$$

V. Now it is easy to see that at the moment t of completion of the circuit construction procedure there are no strangers in any container (assuming that the parameters are chosen correctly). Indeed, in an arbitrary container of depth d, according to what was proved above, there are at most $\mu B_{d,t} \leqslant \mu B_{k,t} < 1$ strangers.

Moreover, by (5.16), $B_{d,t} = a^{d-k} B_{k,t} < a^{d-k}/\mu \leqslant 1$ for $d \leqslant k-2$, which means that all elements are in the lower two layers of the tree.

It remains to specify the choice of parameters that satisfies all the necessary conditions (5.15), (5.16), (5.17), (5.21). For example, $\varepsilon = \mu = 1/100$, $a = 10$, $\nu = 0.7$ will do. ■

Corollary 5.1 ([4, 5]). $\mathsf{D}_{\mathcal{B}_M}(\mathrm{SORT}_n) \asymp \log n$, $\mathsf{C}_{\mathcal{B}_M}(\mathrm{SORT}_n) \asymp n \log n$.

• The multiplicative constant factor in the depth bound of Theorem 5.6 is insanely large. A number of papers have attempted to reduce it. Estimates published with proofs (for various modifications of the algorithm) have an order of several thousands, see, e.g., [66, 112, 241]. Some rough estimations admit the existence of circuits with the depth of approximately $100 \log_2 n$ (mentioned in [282]). In practice, better results are provided by variants of Batcher's circuits [16] of depth $\asymp \log^2 n$.

[12] In the subtree rooted in v'.

Other applications

Fast computation of logarithm and exponential. Unlike ordinary arithmetic operations, fast computation of trigonometric, logarithmic and exponential numerical functions with a given accuracy requires more sophisticated techniques.

Theoretically fast algorithms for computing logarithm (with an accuracy of n digits) have complexity of order $\mathsf{M}(n)\log n$; the first of these were proposed by E. Salamin and R. Brent [48]. They rely on a procedure of finding the *arithmetic-geometric mean* of two numbers $\mathrm{AGM}(a,b)$. By definition,

$$\mathrm{AGM}(a,b)=\lim_{k\to\infty}a_k=\lim_{k\to\infty}b_k,\qquad\text{where}\quad a_0=a,\quad b_0=b,\quad a_{k+1}=(a_k+b_k)/2,\quad b_{k+1}=\sqrt{a_kb_k}.$$

Let $0\leqslant b\leqslant a\leqslant 1$. The sequences $\{a_n\},\{b_n\}$ converge quadratically: for $a_k\gg b_k$ we have

$$\frac{b_{k+1}}{a_{k+1}}=\frac{2\sqrt{b_k/a_k}}{1+b_k/a_k}\approx\frac{2}{\sqrt{b_k/a_k}},$$

and for $a_k\asymp b_k$,

$$0\leqslant a_{k+1}-b_{k+1}=\frac{a_{k+1}^2-b_{k+1}^2}{2a_{k+2}}=\frac{(a_k-b_k)^2}{8a_{k+2}}\leqslant\frac{(a_k-b_k)^2}{8\mathrm{AGM}(a,b)}.$$

Therefore, computing $\mathrm{AGM}(a,b)$ with an accuracy of 2^{-n} requires $\log_2(a/b)+\log_2 n+O(1)$ iterations for small a,b.

The Gauss formula connects the AGM with the value of the elliptic integral of the first kind:

$$\frac{\pi}{2\mathrm{AGM}(a,b)}=I(a,b)=\int_0^{\pi/2}\frac{d\tau}{\sqrt{a^2\cos\tau+b^2\sin\tau}}=\int_0^{+\infty}\frac{dx}{(x^2+a^2)(x^2+b^2)}.\tag{5.22}$$

Here, the role of elliptic integrals is just a justification of the formula

$$|\ln(4/b)-I(1,b)|=O(b^2)\quad\text{as } b\to 0+.\tag{5.23}$$

Formula (5.23) serves as a guide to constructing an algorithm for calculating $\ln X$. By shifting the position of the binary point, the argument can be made sufficiently large: $2^kX\in[2^n,2^{n+1}]$. Then, with high accuracy, $\ln(2^kX)\approx I(1,b)$, where $b=2^{2-k}/X$. Finally, $\ln X$ is calculated as

$$\ln X\approx\frac{\pi}{2\mathrm{AGM}(1,b)}-k\ln 2,$$

where, in the circuit implementation, the constants π and $\ln 2$ (or rather, their approximate values) are considered precomputed. The complexity of the method is determined by $2\log_2 n+O(1)$ iterations of computing $\mathrm{AGM}(1,b)$, which involve arithmetic operations (including root extraction) with $O(n)$-digit numbers. The complexity of one iteration is $O(\mathsf{M}(n))$.

There are numerous variations of the described method, some of which rely on identities different from (5.22), (5.23). An optimized algorithm for calculating the logarithm was proposed by D. Bernstein in [26].

Given a fast way to compute the logarithm, the computation of the exponential e^Y can be implemented by Newton's method of successive approximations, solving the equation $f(x)=Y-\ln x$ via iterations $x_{k+1}=x_k(Y+1-\ln x_k)$. The method also has complexity $\mathsf{M}(n)\log n$.

The calculation of trigonometric and many transcendental functions is grounded on this foundation, for more details see, e.g., [51].

Chapter 6

Algebraic method $\boxed{A}$

> The essence of the algebraic method lies in transferring computations from the original algebraic structure to some other one via a transformation that preserves operations. A gain is obtained if the operations of interest are performed more simply in the second structure, and the transition itself is not very complicated.

Boolean matrix multiplication $\boxed{A}$

Consider the multiplication of matrices over the boolean semiring $(\mathbb{B}, \vee, \wedge)$. A popular application of this operation is constructing of the transitive closure of a graph, that is, determining its connected components. As is known, over the basis of semiring operations the trivial upper bound for the complexity $\mathsf{C}_{\mathcal{B}_M}(MM_n) \leqslant 2n^3 - n^2$ is tight [240]. The situation changes if we allow a computational basis to be extended to a complete one. Immediately after the appearance of Strassen's method [313] of fast matrix multiplication, a number of researchers (e.g., [93, 88]) observed that when performing boolean multiplication it is advantageous to switch to integer multiplication.

Lemma 6.1 ([88])**.** $\mathsf{C}(MM_n^{(\mathbb{B},\vee,\wedge)}) \preccurlyeq \log^2 n \cdot \mathsf{C}_{\mathcal{A}^R}(MM_n^{\mathbb{Z}_{n+1}})$.

$\triangleright$ Having embedded the boolean coefficients of matrices into the ring $\mathbb{Z}_{n+1}$, we perform the multiplication over this ring. At the end, we perform the inverse transition, checking the coefficients of the product for equality to zero. It remains to note that the complexity of arithmetic operations in the ring $\mathbb{Z}_{n+1}$ is, in any case, at most quadratic in the number of digits. $\square$

Corollary 6.1. $\mathsf{C}(MM_n^{(\mathbb{B},\vee,\wedge)}) \preccurlyeq n^{\omega+o(1)}$, *where* $\omega < 2.38$ *is the matrix multiplication exponent.*

- A small advantage in terms of complexity is provided by using a direct product $\mathbb{Z}_{p_1} \times \ldots \times \mathbb{Z}_{p_s}$ of residue rings over several coprime modules instead of $\mathbb{Z}_{n+1}$.

In general, the problem of fast matrix multiplication over monotone semirings has not been solved. For example, the question of the possibility of matrix multiplication over tropical semirings $(\mathbb{R}, \min, +)$, $(\mathbb{R}, \max, +)$ with subcubic complexity $n^{3-\Omega(1)}$ in a nonmonotone basis remains open.

Formula complexity of summation modulo 7 $\boxed{A}\boxed{/_2}$

Let us consider the problem of constructing short formulae over the basis $\mathcal{B}_2$ for the operator $\mathrm{MOD}_n^7(X)$ of summation modulo 7. The application of formula (4.1) leads to the bound $\Phi_{\mathcal{B}_2}(\mathrm{MOD}_n^m) \preccurlyeq \Phi_{\mathcal{B}_0}(\mathrm{MOD}_n^m) \preccurlyeq n^{1+\log_2 m}$. Obviously, this method does not reveal all the capabilities of the basis $\mathcal{B}_2$.

For the binary basis, W. McColl in [216] proposed a slightly more economical formula

$$\mathrm{MOD}_{n_1+n_2}^{m,r}(X) = \bigwedge_{k=1}^{m-1} \left(\mathrm{MOD}_{n_1}^{m,k}(X^1) \sim \mathrm{MOD}_{n_2}^{m,r-k}(X^2)\right), \tag{6.1}$$

where "$\sim$" denotes the boolean equivalence operation and $X = (X^1, X^2)$, $|X^i| = n_i$. Computation by this formula leads to the upper bound

$$\Phi_{\mathcal{B}_2}(\mathrm{MOD}_n^m) \preccurlyeq n^{1+\log_2(m-1)}, \tag{6.2}$$

which for $m = 3$ still remains a record[1)]. For $m = 7$ formula (6.1) provides only the bound $\Phi_{\mathcal{B}_2}(\mathrm{MOD}_n^7) \prec n^{3.59}$.

D. van Leijenhorst [186] proposed in the case $m = 7$ to switch to calculations in the multiplicative group of the field $\mathbb{F}_8$ with the representation of its elements by binary matrices of size 3×3. This group is isomorphic to the group $(\mathbb{Z}_7, +)$. The group operation in the chosen representation of the group $\mathbb{F}_8^*$ is the ordinary matrix multiplication over $\mathbb{F}_2$.

Theorem 6.1 ([186]). $\Phi_{\mathcal{B}_2}(\mathrm{MOD}_n^7) \prec n^{2.59}$.

▶ Let the $(7,4)$-operator $\pi : (\mathbb{Z}_7, +) \to \mathbb{F}_8^*$ perform the transition between the two representations according to the rule $r \to g^r$, where g is a generator of the group $\mathbb{F}_8^*$. Denote $H_n(X) = g^{\sum_{i=1}^n x_i}$. The operator $H_n(X)$ may be computed recursively according to the matrix multiplication rule

$$H_{n_1+n_2}[i,j](X) = \bigoplus_{k=0}^{2} H_{n_1}[i,k](X^1) \cdot H_{n_2}[k,j](X^2), \tag{6.3}$$

from which we obtain $\Phi_{\mathcal{B}_2}(H_n) \preccurlyeq n^{\log_2 6}$. But in view of

$$\mathrm{MOD}_n^7(x) = \pi^{-1}(H_n(\pi(x_1), \ldots, \pi(x_n)))$$

and $\Phi_{\mathcal{B}_2}(\pi, \pi^{-1}) = O(1)$, we establish $\Phi_{\mathcal{B}_2}(\mathrm{MOD}_n^7) \preccurlyeq \Phi_{\mathcal{B}_2}(H_n)$, therefore, $\Phi_{\mathcal{B}_2}(\mathrm{MOD}_n^7) \prec n^{2.59}$. ■

[1)] For $m \geqslant 5$ this bound is superseded by the general complexity bound for the class of symmetric functions $\Phi_{\mathcal{B}_2}(S_n) \prec n^{2.82}$ [306].

• A similar approach to computing sums modulo 3 and 5 consists of switching to calculations in fields $\mathbb{F}_4$ and $\mathbb{F}_{16}$, respectively, and does not improve (6.2). Computation by rules (6.3) leads to the depth bound $\mathsf{D}_{\mathcal{B}_2}(\mathrm{MOD}_n^7) \lesssim 3\log_2 n$, which was slightly improved by the author in [297] to $\mathsf{D}_{\mathcal{B}_2}(\mathrm{MOD}_n^7) \lesssim 2.93\log_2 n$ by partitioning variables into three groups and special encoding.

Integer multiplication via DFT $\boxed{A}\boxed{\varepsilon}$

In 1963, A. L. Toom [320] not only generalized Karatsuba's method, but also proposed a concept that has been followed by all fast multiplication algorithms since then. By partitioning numbers into blocks, integer multiplication turns into multiplication of polynomials over a suitably chosen ring R. Multiplication in the ring $R[x]$ by interpolation reduces to componentwise multiplication in the ring R^N, where N is the number of interpolation points.

According to [209], the first fast algorithm for multiplying integers based on the DFT was constructed by N. S. Bakhvalov — his method had complexity $O(n\log^3 n)$. Then, in 1971, A. Schönhage and V. Strassen [280] published two faster methods of multiplication at once. The first of them exploits the natural idea of transition to calculations over the field of complex numbers $\mathbb{C}$, which admits a DFT of arbitrary order.

Theorem 6.2 ([280]). *For any $d = O(1)$,*

$$\mathsf{C}(M_n) \preccurlyeq n\log n\log\log n\cdot\ldots\cdot\log^{(d-1)} n\cdot(\log^{(d)} n)^2.$$

▶ Let $2n = 2^k q$. Split n-digit numbers A and B to be multiplied into blocks of length q and, via the substitution $2^q \to x$, pass to polynomials:

$$A \to A(x) = \sum_{i=0}^{2^{k-1}-1} a_i x^i, \qquad B \to B(x) = \sum_{i=0}^{2^{k-1}-1} b_i x^i.$$

The desired product AB can be restored from the product of polynomials $C(x) = A(x)B(x)$ by the inverse substitution $x = 2^q$ with complexity of order $2^k(2q+k)$, since the coefficients of $C(x)$ are $(2q+k)$-digit numbers.

We interpret the multiplication of polynomials as multiplication in the ring $\mathbb{C}[x]$, where the operations are performed with such precision that the coefficients of the product polynomial, which are actually integers, are computed with an error $< 1/2$. So, they can then be recovered by rounding.

The product of polynomials is calculated via the DFT as[2)]

$$C(x) = \mathrm{DFT}^{-1}_{2^k,\zeta}\left(\mathrm{DFT}_{2^k,\zeta}(A(x))\odot\mathrm{DFT}_{2^k,\zeta}(B(x))\right),\quad \mathrm{DFT}^{-1}_{2^k,\zeta} = 2^{-k}\cdot\mathrm{DFT}_{2^k,\zeta^{-1}}, \tag{6.4}$$

and the DFTs themselves are performed by the method of Theorem 2.4. Recall that the standard circuit for the DFT of order 2^k consists of k layers, on which DFTs of

[2]Here, polynomial notation is used for the DFT arguments. Recall also that the symbol $\odot$ denotes the componentwise product of vectors.

order 2 are performed in parallel (this is simply addition and subtraction of a pair of complex numbers), between which there are layers of parallel multiplications by roots of unity ζ^j, see Lemma 2.1. Fig. 6.1 shows the circuit for the DFT of order 8.

To perform both scalar and nonscalar multiplications, we use a recursive call of this multiplication algorithm followed by rounding to s digits after the binary point. In this case, complex multiplication is performed by a two-layer circuit: on the first layer — four real multiplications, on the second layer — real addition and subtraction.

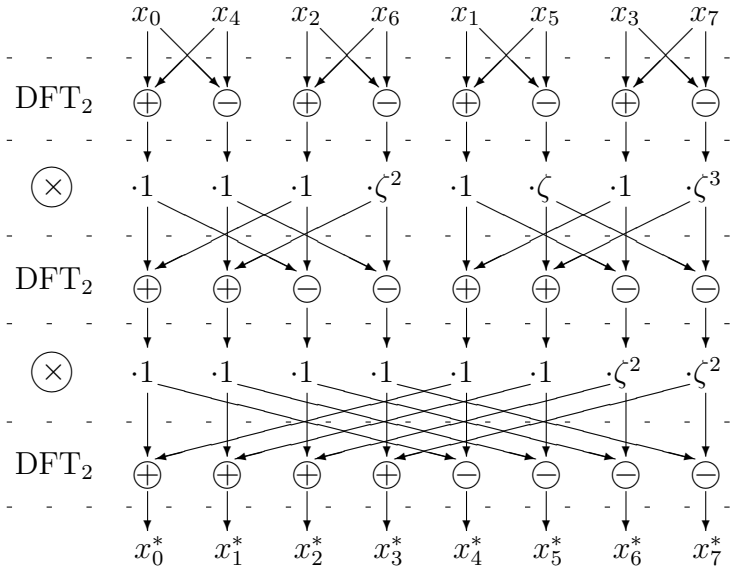


Figure 6.1: Circuit for the DFT of order 8 with primitive root ζ

Note that if the (complex) components of the input vector of the DFT of order 2^k are bounded in absolute value by m, then the coefficients of the intermediate vectors in the process of computing the DFT are obviously bounded by $2^k m$. Consequently, when computing the product $C(x)$ by formula (6.4), there will be no numbers whose absolute value exceed $M = 2^{3k+2q}$ (since $m < 2^q$). Thus, to write the real parts of complex numbers in the process of computation, it is sufficient to use $3k + 2q + 1$ digits before the binary point[3)]. We will determine the number s of digits kept after the binary point later, based on the error estimate.

Let ε_x denote the error in calculating a (real) value x. We will roughly estimate the evolution of the absolute value of the error ε from layer to layer. Since $(a + \varepsilon_a) \pm (b+\varepsilon_b) = (a \pm b)+(\varepsilon_a \pm \varepsilon_b)$, the error estimate doubles at layers of additive operations, $\varepsilon \to 2\varepsilon$. In the case of multiplication, we have

$$\varepsilon_{ab} = (a + \varepsilon_a)(b + \varepsilon_b) - ab + \varepsilon_o = b\varepsilon_a + a\varepsilon_b + \varepsilon_a\varepsilon_b + \varepsilon_o,$$

where ε_o is the error that occurs during rounding, $|\varepsilon_o| \leqslant 2^{-s}$. Therefore, in the case of nonscalar multiplication, we set $\varepsilon \to 3M\varepsilon + 2^{-s}$, and in the case of multiplication by roots of unity[4)] ($|b| < 1$ and $\varepsilon_b \leqslant 2^{-s}$) we set $\varepsilon \to \varepsilon + 2M2^{-s}$.

[3] Taking into account that we do not allow an error greater than 1/2.

[4] These are precomputed constants.

Therefore, when computing the DFT of order 2^k, the error estimate changes from ε to

$$4(2M2^{-s}+4(2M2^{-s}+\ldots+4(2M2^{-s}+2\varepsilon)\ldots))<M2^{2k-s}+2^{2k}\varepsilon.$$

Finally, the calculation error according to formula (6.4) is estimated as

$$0\overset{\mathrm{DFT}}{\longrightarrow}M2^{2k-s}\overset{\odot}{\longrightarrow}3M^2 2^{2k-s}+2^{-s}<M^2 2^{2k+2-s}\overset{\mathrm{DFT}}{\longrightarrow}M^2 2^{4k+2-s}+M2^{2k-s}<M^2 2^{4k+3-s}.$$

When choosing $s=10k+4q+4$, the error is strictly less than $1/2$, which is required.

For $q\asymp\log n$ and $l=s+3k+2q+1$ we obtain the recurrence relation

$$\mathsf{C}(M_n)\leqslant O(2^k k)(\mathsf{C}(M_l)+l),$$

which, if after d steps we apply the standard multiplication method, leads to the bound in the statement of the theorem. ■

• The same complexity bound (and probably by the same method) was previously obtained by A. A. Karatsuba [146], but not published.

Subsequent faster multiplication methods differ primarily in the choice of a ring for performing the DFT. The second Schönhage—Strassen [280] method achieved a bound of $\mathsf{C}(M_n)\preccurlyeq n\log n\log\log n$ that remained the record for 35 years. It involves a reduction to multiplication in the polynomial ring $\mathbb{Z}_{\Phi_m}[x]/(x^{2^{m+1}-1})$, where $\Phi_m=2^{2^m}+1$. The DFT is performed over the ring $\mathbb{Z}_{\Phi_m}$, which is distinguished by the simplicity of multiplications by roots of unity — they are simply powers of two.

M. Fürer's method [92] combines the advantages of both methods mentioned: logarithmic speed of size reduction and simplicity of multiplications by roots of unity. It transfers multiplication to the ring $C_p[y]/(y^{2^{ps}}-1)$, where $C_p=\mathbb{C}[x]/(x^{2^p}+1)$. The DFT of order $2^{s(p+1)}$ is used for multiplication. If (by the method of Lemma 2.1) we represent it as a composition of DFTs of order 2^{p+1}, then multiplications by powers of x will dominate among scalar multiplications. Note that x is a primitive root[5)] of order 2^{p+1} in C_p. The method proves the bound $\mathsf{C}(M_n)\preccurlyeq n\log n\cdot 2^{O(\log^* n)}$.

In the method of D. Harvey and J. van der Hoeven [119] integer multiplication is reduced to multiplication in the ring of polynomials of several variables $C_p[x_1,\ldots,x_d]/(x_1^{n_1}-1,\ldots,x_d^{n_d}-1)$. For multiplication in this ring, a multidimensional DFT is used, which, however, is easily reduced to ordinary DFTs. The transition to a multidimensional space radically solves the problem of the simplicity of primitive roots of unity (it is sufficient to consider monomials of variables x_i), but introduces a number of technical difficulties, overcoming which allowed the authors [119] to obtain a record result $\mathsf{C}(M_n)\preccurlyeq n\log n$.

A similar theory is developed for the multiplication of polynomials over an arbitrary ring R. The method proposed in [120] has a complexity estimate of $O(n\log n)$ in the case of the validity of an unproven conjecture. The unconditionally proven estimate $\mathsf{C}_{\mathcal{A}^R}(M_n^R)\preccurlyeq n\log n\cdot 2^{O(\log^* n)}$ was obtained earlier by the same authors together with G. Lecerf in [121].

For a more detailed acquaintance with the theory of fast multiplication methods, the survey papers of D. Bernstein [25, 28] and a modern survey by S. B. Gashkov and the author [105] are recommended.

Parallel integer division circuits $\boxed{A}\boxed{\varepsilon}$

As is easy to verify, Cook's method of dividing numbers described above (Theorem 5.1) [69] leads to circuits of depth of order $\log^2 n$. An important milestone in

[5)] Here we allow the usual liberty of speech, calling the elements of the factor ring C_p not the classes of equivalent polynomials modulo $x^{2^p}+1$, but the representatives of classes.

the development of fast parallel algorithms was the work of P. Beame, S. Cook, H. Hoover [20], in which the authors constructed circuits of logarithmic depth for division, as well as for related problems. A somewhat simpler and more efficient method was proposed by J. Håstad and T. Leighton [124]. These methods are based on the transition to modular arithmetic — one of the most popular algebraic techniques. As explained above, it is sufficient to show parallel circuits for inversion.

Theorem 6.3 ([20]). $\mathsf{D}(I_n) \asymp \log n$.

Paul Beame
University of Washington, since 1987

▶ Without loss of generality, we may assume that the input of the circuit is an n-digit number $a \in [1/2, 1]$. Let also $\log_2 n \in \mathbb{N}$ and $n \geqslant 16$. Denote $z = 1 - a$. Then

$$a^{-1} = \sum_{k=0}^{\infty} z^k \approx \sum_{k=0}^{2n-1} z^k = \prod_{i=0}^{\log_2 n} (1 + z^{2^i}), \qquad (6.5)$$

where the approximation error does not exceed 2^{1-2n} due to $z \leqslant 1/2$. If each of the factors on the right-hand side of (6.5) is computed with an absolute error of $\leqslant \varepsilon$, then an absolute value of the error in computing the product (6.5) does not exceed

$$\log_2 n \cdot \varepsilon \cdot \sum_{k=0}^{2n-1} z^k \leqslant 2n \log_2 n \cdot \varepsilon \leqslant 2^{1-n}\varepsilon. \qquad (6.6)$$

I. In parallel, we compute all powers of z^{2^i} with an accuracy of 2^{-2n} each, $i \leqslant \log_2 n$.

For this, we use modular arithmetic (this is the central part of the method). Choose coprime numbers $p_1, \ldots, p_s$ satisfying $P = p_1 \cdot \ldots \cdot p_s \geqslant 2^{n^2}$ so that $p_i \leqslant 2n^2$ and $s \leqslant 2n^2$. This can be done according to the law of distribution of prime numbers (see, e.g., [264]).

By assumption, $2^n z$ is an n-digit integer. We raise it to the power 2^i in the ring $\mathbb{Z}_{p_1} \times \ldots \times \mathbb{Z}_{p_s}$ and then recover the result $(2^n z)^{2^i} < 2^{n^2}$ via the Chinese remainder theorem.

The procedure is performed in three stages. First, the remainders $a_j = 2^n z \bmod p_j$ are computed, then the powers $a_j^{2^i} \bmod p_j$, and finally the desired number $(2^n z)^{2^i} \bmod P$. The depth of the first and third stages is estimated in the following two lemmas. Recall that $\mathsf{D}(M_n) \asymp \log n$ and $\mathsf{D}(\Sigma_{m,n}) \asymp \log(mn)$ (see, for example, Corollaries 4.1 and 4.2), and $\mathsf{D}(\mathcal{P}_n) \asymp n$ (see below (11.5) and Corollary 12.1).

Lemma 6.2. *The residue $A \bmod p$, where A is an n-digit number, and p is an m-digit constant, can be computed with depth $O(m + \log n)$.*

▷ Split the dividend into blocks of length m: $A = \sum_{k=0}^{n/m} a_k 2^{km}$. Compute $A \bmod p$ by the formula

$$A \bmod p = \sum_{k=0}^{n/m} a_k (2^{km} \bmod p) \bmod p.$$

Since the factors $2^{km} \bmod P$ can be considered as precomputed constants, the depth is estimated as $\mathsf{D}(M_m) + \mathsf{D}(\Sigma_{2m,\,n/m}) + \mathsf{D}(\mathcal{P}_{2m+\log_2 n}) \preccurlyeq m + \log n$, since the sum of products is $\leqslant (n/m)2^{2m}$ (here we consider the operation of external modulo reduction as a general boolean operator). □

Lemma 6.3. *Recovering a number $A \in [0, P)$ from residues modulo m-digit coprime constants $p_1, \ldots, p_s$, where $P = p_1 \cdot \ldots \cdot p_s$, can be performed with depth $O(\log(ms))$.*

▷ Let $a_i = A \bmod p_i$. Then $A = \sum_{i=1}^{s} u_i a_i \bmod P$, where $u_i < P/p_i$ are suitable constants. The sum of products is computed with depth at most $\mathsf{D}(M_{ms}) + \mathsf{D}(\Sigma_{ms,\,s}) \preccurlyeq \log(ms)$. The final reduction modulo P can be performed by the formula $X \bmod P = X - \lfloor X \cdot (1/P) \rfloor \cdot P$ with depth of order $\mathsf{D}(M_{ms}) \asymp \log(ms)$ (we assume the constant $1/P$ to be precomputed with the required accuracy). □

As a consequence, for the depth of approximate computation of all powers z^{2^i} we obtain an estimate $O(\log n)$, since the operation of raising to a power in $\mathbb{Z}_{p_i}$ at the second stage can be considered as a boolean operator of the general form.

II. Let us compute the final product in (6.5), again via modular arithmetic.

For each of the determined approximate values $\tilde{z}_i \approx z^{2^i}$, we find the remainders of $b_i = \lfloor 2^{2n}(1+\tilde{z}_i) \rfloor$ divided by p_j (Lemma 6.2). The products $\prod_{i=0}^{\log_2 n} b_i$ in $\mathbb{Z}_{p_j}$ (recall that b_i are $O(\log n)$-digit numbers) may be computed by a binary tree of ordinary integer multiplications, and the result is finally reduced modulo p_j. Eventually, the desired product $\prod_i 2^{2n}(1 + \tilde{z}_i)$ is recovered via Lemma 6.3 (exactly, since $2^{(2n+1)(\log_2 n+1)} \leqslant 2^{n^2} < P$ for $n \geqslant 16$). Restoring the correct position of the binary point, we obtain an approximation to a^{-1} with an error of $\leqslant 2^{-n}$ due to (6.6), since the factors in the product are computed with an accuracy of $\varepsilon < 2^{1-2n}$ (this consists of the error in $\tilde{z}_i$ and rounding when going to b_i). The depth of the stage is $\asymp \log n$. ■

• Note that both the estimates and the constructions in Lemmas 6.2, 6.3 are far from optimal. Beame, Cook, and Hoover [20] estimated the complexity of their division circuits of logarithmic depth as $n^{4+o(1)}$. Håstad and Leighton [124] proposed a family of circuits of complexity $n^{1+\varepsilon}$ and depth $\varepsilon^{-2} \log n$. In development of this result, J. Reif and S. Tate [261] showed that division is possible with the depth of order $\log n \log\log n$ and optimal complexity of order $\mathsf{M}_{\log}(n) = O(n \log n)$ (taking into account the result [119]), where the functional $\mathsf{M}_{\log}(n)$ is defined in the same way as $\mathsf{M}(n)$, but assuming the logarithmic depth multiplication algorithm.

Modular composition of polynomials $\boxed{A}\boxed{/_2}$

The operator MC_n^R of modular composition of two polynomials $f, g \in R[x]$ of degree $< n$ modulo a third polynomial h of degree n is defined as $MC_n^R(f, g, h) = f(g) \bmod h$. The composition operation plays an important role in the arithmetic of finite fields, in particular, in the problem of factorization of polynomials over finite fields (see, for example, [144]).

The first subquadratic complexity algorithm for modular composition was invented by R. Brent and H. Kung [50][6]. It exploits reduction to the multiplication of rectangular matrices. Let $rs \geqslant n$. Write $f(x) = f_1(x) + x^r f_2(x) + \ldots + x^{(s-1)r} f_s(x)$, where $\deg f_i < r$. By $r+s-2$ successive multiplications and divisions with remainder, the polynomials $g_i = g^i \bmod h$, $i = 1, 2, \ldots, r, 2r, \ldots, (s-1)r$ are computed. Next, the compositions $\varphi_j = f_j(g) \bmod h$, $j = 1, \ldots, s$, are computed by substituting the polynomials g_i for the powers of x^i. We have

$$\begin{bmatrix} \varphi_1 \\ \varphi_2 \\ \cdots \\ \varphi_s \end{bmatrix}_{s\times n} = \begin{bmatrix} f_1 \\ f_2 \\ \cdots \\ f_s \end{bmatrix}_{s\times r} \cdot \begin{bmatrix} g_0 \\ g_1 \\ \cdots \\ g_{r-1} \end{bmatrix}_{r\times n},$$

where the coefficients of the corresponding polynomials are written out in the matrix rows. Finally, $f(g) \equiv \varphi_1 + g_r\varphi_2 + \ldots + g_{(s-1)r}\varphi_s \bmod h$. When choosing $r \sim s \sim \sqrt{n}$, the method provides a complexity bound

$$\mathsf{C}(MC_n) \prec \mathsf{C}(MM_{r,s,n}) + (r+s)\cdot(\mathsf{C}(M_n) + \mathsf{C}(QR_{n,n})) \prec n^{1.63},$$

if we apply the record complexity bound for the rectangular matrix multiplication [8].

Fundamental progress in the problem of modular composition was achieved by the algebraic approach proposed 30 years later by C. Umans [325]. Umans' algorithm has almost linear complexity for polynomials over a field of small characteristic. We restrict ourselves to a slightly less general case of a field of small order.

Christopher Umans
California Institute of Technology, since 2002

Theorem 6.4 ([325]). *The modular composition of polynomials over the field $\mathbb{F}_p$ for any $m \preccurlyeq \log n$ has complexity $\mathsf{C}_{\mathcal{A}^{\mathbb{F}_p}}(MC_n) \preccurlyeq \left((pm^2)^m n^{1+O(1/m)}\right)^{1+o(1)}$.*

► Umans' method is based on interpolation. It is necessary to reduce the problem dimension first, since a polynomial $f(g(x))$ can have a degree of order n^2. To do this, for $n \leqslant d^m$, the Kronecker substitution $x_i = x^{d^i}$, $i = 0, \ldots, m-1$, is performed, under which the polynomial $f(x)$ turns into $f^*(x_1, \ldots, x_{m-1})$. The polynomials $g_i(x) = g^{d^i} \bmod h$ are computed in a standard way (by successive raisings to the power d) with a total complexity of order $\underline{m \log d \cdot \mathsf{M}(n)}$. Now the degree of the polynomial $\hat{f}(x) = f^*(g_0(x), \ldots, g_{m-1}(x))$ does not exceed $N = dmn$, and it can be computed via interpolation at N points. To provide a reserve of interpolation points, we move from $\mathbb{F}_p$ to $\mathbb{F}_q$, where $q = p^t > N$.

First, we consider the auxiliary problems of evaluation of a polynomial on a set of points and the inverse problem of interpolation. The bisection method leads to the following result, probably first obtained by H. Kung [180].

[6] More precisely, in the paper [50] they considered a special case of the problem with $h(x) = x^n$, but the result is easily generalized.

Lemma 6.4 ([180]). *The following problem are implemented by circuits over $\mathcal{A}^R$ of complexity $\preccurlyeq \mathsf{M}(n)\log n$:*

(i) evaluating of a polynomial $f(x) \in R[x]$ of degree $< n$ at n points $\alpha_0, \ldots, \alpha_{n-1} \in R$;

(ii) recovering a polynomial $f(x) \in R[x]$ of degree $< n$ from given values $f(\alpha_i) = c_i$ at points α_i, $i = 0, \ldots, n-1$.

$\triangleright$ Consider a balanced binary tree T with n leaves, oriented toward the root. The leaves are numbered from 0 to $n-1$ — they correspond to the indices of the interpolation points α_i. Next, each vertex v is assigned a set of numbers $I(v) = I(v') \cup I(v'')$, where v', v'' are vertices preceding v. The root of the tree corresponds to the set $[\![n]\!]$.

Let $p_I(x) = \prod_{i \in I}(x - \alpha_i)$. Since $f(\alpha) = f \bmod (x - \alpha)$, all values $f(\alpha_1), \ldots, f(\alpha_n)$ can be determined by moving along the tree T from the root and calculating at each vertex v the polynomials $f \bmod p_{I(v)}$. At the root of the tree we simply have $f \bmod p_{[\![n]\!]} = f$, at the leaves we obtain $f(\alpha_i)$.

The complexity of computing auxiliary polynomials $p_{I(v)}$ is bounded by

$$\mathsf{C}(M_{n/2+1}) + 2\mathsf{C}(M_{n/4+1}) + \ldots + (n/2)\mathsf{C}(M_2) \preccurlyeq \mathsf{M}(n)\log n.$$

The complexity of divisions with remainder, providing all $f \bmod p_{I(v)}$, is estimated as

$$2\mathsf{C}(QR_{n,n/2+1}) + 4\mathsf{C}(QR_{n/2,n/4+1}) + \ldots + n\mathsf{C}(QR_{3,2}) \preccurlyeq \mathsf{M}(n)\log n.$$

This proves (i).

Let's move on to the interpolation problem. Denote $p^*_{I,k}(x) = \prod_{i \in I \setminus \{k\}}(x - \alpha_i)$. We will carry out computations exploiting the Lagrange interpolation formula

$$f(x) = \sum_{k=1}^{n} c^*_k \cdot p^*_{[\![n]\!],k}(x), \qquad c^*_k = c_k / p^*_{[\![n]\!],k}(\alpha_k).$$

First, compute the values $p^*_{[\![n]\!],k}(\alpha_k)$. Note that $p^*_{[\![n]\!],k}(\alpha_k) = p'_{[\![n]\!]}(\alpha_k)$, where $p'_{[\![n]\!]}$ is the derivative of the polynomial $p_{[\![n]\!]}$. Polynomial $p'_{[\![n]\!]}$ is computed with linear complexity if $p_{[\![n]\!]}$ is known. Then, according to (i), the values of the polynomial $p'_{[\![n]\!]}$ at all points α_k are computed with complexity $\preccurlyeq \mathsf{M}(n)\log n$. Another n division operations allow us to determine all c^*_k.

Denote $f_I(x) = \sum_{k \in I} c^*_k \cdot p^*_{I,k}(x)$. Moving along the tree in the direction from the leaves to the root, we sequentially calculate all the polynomials $f_{I(v)}$ by formulas $f_{I(v)} = f_{I(v')} p_{I(v'')} + f_{I(v'')} p_{I(v')}$, where v' and v'' precede v. At the leaves, the polynomials $f_{I(v)}$ coincide with constants c^*_j. At the root, we obtain the desired polynomial $f(x)$.

All auxiliary polynomials $p_{I(v)}$ are computed with complexity $\preccurlyeq \mathsf{M}(n)\log n$. After this, the complexity of computing all $f_{I(v)}$ may be estimated as

$$2\mathsf{C}(M_{n/2+1}) + 4\mathsf{C}(M_{n/4+1}) + \ldots + n\mathsf{C}(M_2) \preccurlyeq \mathsf{M}(n)\log n.$$

$\square$

According to Lemma 6.4, the values of the polynomials $g_i(x)$ at the interpolation points may be determined with complexity $\preccurlyeq \underline{m\mathsf{M}(N)\log N}$ operations in $\mathbb{F}_q$. It

remains to solve the problem of evaluating of the polynomial f^* of m variables on a set of N vectors $\alpha_i = (\alpha_{i,0}, \ldots, \alpha_{i,m-1}) \in \mathbb{F}_q^m$.

Let $\mathbb{F}_{q_0} \cong \mathbb{F}_p(\beta)$, where $q_0 = p^s \geqslant dm^2$, $s \mid t$, and β is a primitive element of the field $\mathbb{F}_{q_0} \subset \mathbb{F}_q$. Compute polynomials $\varphi_1(x), \ldots, \varphi_N(x) \in \mathbb{F}_q[x]$ of degree $< m$ given their values at points β^j:

$$\varphi_i(\beta^j) = (\alpha_{i,j})^{q_0^{-j}} = (\alpha_{i,j})^{p^{t-sj}}, \qquad j = 0, \ldots, m-1.$$

The complexity of computing the powers of $\alpha_{i,j}$ is $\preccurlyeq \underline{Nm \log q}$, and interpolation according to Lemma 6.4 may be performed with complexity of $\preccurlyeq \underline{N\mathsf{M}(m) \log m}$ operations in $\mathbb{F}_q$.

Consider the polynomial $F(y) = f^*(y, y^{q_0}, \ldots, y^{q_0^{m-1}})$. Its nonzero coefficients coincide with the coefficients of the polynomial $f(x)$, only belong to other terms, while $\deg F < q_0^m$. Formally setting $F \in S[y]$, where $S = \mathbb{F}_q[x]/(x^{q_0-1} - \beta)$, we find the values of $F(y)$ at N points $\varphi_i(x) \in S$ by the method of Lemma 6.4 via $\preccurlyeq \underline{\mathsf{M}(Q) \log Q}$ operations in S, where $Q = \max\{N, q_0^m\}$.

We denote the Frobenius powers of the polynomials $\varphi_i(x) = \sum_{l=0}^{m-1} a_l x^l$ as $\varphi_i^{[j]} = \sum_{l=0}^{m-1} a_l^{q_0^j} x^l$. Now we will show that $f^*(\alpha_i) = F(\varphi_i(x))|_{x=1}$. Indeed,

$$\begin{aligned} F(\varphi_i(x)) = f^*\left(\varphi_i(x), \varphi_i^{q_0}(x), \ldots, \varphi_i^{q_0^{m-1}}(x)\right) = \\ f^*\left(\varphi_i(x), \varphi_i^{[1]}(x^{q_0}), \ldots, \varphi_i^{[m-1]}(x^{q_0^{m-1}})\right) \equiv \\ f^*\left(\varphi_i(x), \varphi_i^{[1]}(\beta x), \ldots, \varphi_i^{[m-1]}(\beta^{m-1} x)\right) \mod x^{q_0-1} - \beta, \end{aligned}$$

where the degree of the last polynomial does not exceed $(d-1)m^2 < q-1$, i.e., this is exactly the result of the reduction modulo $x^{q_0-1} - \beta$. It remains to note that (after the substitution $x = 1$) $\varphi_i^{[j]}(\beta^j) = (\varphi_i(\beta^j))^{q_0^j} = \alpha_{i,j}$, since $\beta \in \mathbb{F}_{q_0}$. The complexity of the substitutions does not exceed $\underline{q_0 N}$ operations in $\mathbb{F}_q$.

Now the polynomial $\hat{f}(x)$ may be recovered by the method of Lemma 6.4 via $\underline{\mathsf{M}(N) \log N}$ operations in $\mathbb{F}_q$. The final reduction of $\hat{f}$ modulo h is performed with complexity $\mathsf{C}(QR_{N,n}) \preccurlyeq \underline{dm \log n \cdot \mathsf{M}(n)}$ over $\mathbb{F}_p$.

The complexity of the described algorithm is bounded by the sum of the complexity estimates of individual steps, which are underlined in the course of the proof. Finally, we obtain

$$\begin{aligned} \mathsf{C}_{\mathcal{A}^{\mathbb{F}_p}}(MC_n) \preccurlyeq m \log d \cdot \mathsf{M}(n) + dm \log n \cdot \mathsf{M}(n) + \mathsf{M}(Q) \log Q \cdot \mathsf{M}(q_0)\mathsf{M}(t) + \\ [\, m\mathsf{M}(N) \log N + Nm \log q + N\mathsf{M}(m) \log m + q_0 N + \mathsf{M}(N) \log N \,]\, \mathsf{M}(t) \preccurlyeq \\ (Nt)^{1+o(1)}(q_0 + \log q) + (q_0^{m+1} t)^{1+o(1)}. \end{aligned}$$

Given $N = dmn$, choosing $d^m \leqslant dn$, $q_0 \leqslant pdm^2$ and $q \leqslant q_0 dmn$ (so that $t \preccurlyeq \log(dmn)$), we conclude

$$\mathsf{C}_{\mathcal{A}^{\mathbb{F}_p}}(MC_n) \preccurlyeq p(dm)^{O(1)} n^{1+o(1)} + (d^2(pm^2)^{m+1} n)^{1+o(1)},$$

from which the stated bound follows. ■

In the case $p = n^{o(1)}$, one can choose $1 \prec m \prec \frac{\log n}{\log\log n}$ such that $p^m = n^{o(1)}$ and $d = n^{o(1)}$, so $\mathsf{C}_{\mathcal{A}^{\mathbb{F}_p}}(\mathcal{MC}_n) = n^{1+o(1)}$.

• A slightly more subtle argument yields roughly the same complexity estimate as in Theorem 6.4, only with char $\mathbb{F}_p$ instead of p [325].

The refined complexity estimates for the algorithms in Lemma 6.4 obtained by A. Bostan and É. Schost in [43] are $\lesssim 1.5\mathsf{M}(n)\log_2 n$ for the problem (i) of evaluating at n points and $\lesssim 2.5\mathsf{M}(n)\log_2 n$ for the problem (ii) of interpolating given values at points, where $n = 2^k$.

An alternative algorithm for performing the central stage of modular composition — multipoint evaluation of a polynomial of several variables — is proposed in [31].

J. van der Hoeven and G. Lecerf [131] by modifying the method of K. Kedlaya and C. Umans [151][7] extended the described algorithm to fields of any cardinality and residue rings, obtaining for the *bit* complexity the estimate $\mathsf{C}_{\mathcal{B}_2}(\mathcal{MC}_n^R) = (n\log q)^{1+o(1)}$, where $R = \mathbb{F}_q$ or $R = \mathbb{Z}_q$.

Other applications

Multiplication in Mersenne fields. Consider the problem of multiplication modulo a Mersenne prime $2^p - 1$, i.e., in fact, multiplication in a prime field $\mathbb{Z}_{2^p-1}$. It can be represented as a cyclic convolution of order p of vectors of binary notation of the numbers being multiplied. Therefore, the multiplication is performed via the DFT of order p with a primitive root $2 \in \mathbb{Z}_{2^p-1}$. But p is also a prime number, and a DFT of such order is usually not implemented very efficiently. However, in this situation, one can use a technique proposed by R. Crandall and B. Fagin [76], which in some cases allows one to reduce a DFT of an "unsuitable" order to a DFT of a "suitable" order.

The technique consists in transition to approximate calculation of real DFT of arbitrary order N. We split the number $X = [x_{p-1}, \ldots, x_0]$ to be multiplied into N blocks of approximately equal length: $X = [X_{N-1}, \ldots, X_0]$. Let B_i be the position of the beginning of the i-th block. Write X as

$$X = \sum_{i=0}^{N-1} X_i \cdot 2^{B_i} = \sum_{i=0}^{N-1} \left(X_i \cdot 2^{B_i - ip/N}\right) 2^{ip/N} = \sum_{i=0}^{N-1} X'_i \cdot 2^{ip/N}.$$

Now the multiplication of X by $Y = \sum_{i=0}^{N-1} Y'_i \cdot 2^{ip/N}$ can be performed via two DFTs of order N with primitive root $2^{p/N} \in (\mathbb{R} \bmod 2^p - 1)$ and one inverse DFT. From the resulting vector $[Z'_{N-1}, \ldots, Z'_0]$ the desired product may be determined as

$$XY = \sum_{i=0}^{N-1} \left(Z'_i \cdot 2^{ip/N - B_i}\right) 2^{B_i} \bmod 2^p - 1.$$

When choosing a block size close to the length of a machine word, the described method is efficiently implemented on standard computers.

Arithmetic in normal bases of finite fields. A finite field of order q^n is isomorphic to the factor ring of polynomials $\mathbb{F}_q[x]/(m_n(x))$ modulo an irreducible over $\mathbb{F}_q$ polynomial $m_n(x)$ of degree n. Thus, operations in a finite field are essentially operations on polynomials. Sometimes, however, alternative representations may be useful.

In general, elements of the field $\mathbb{F}_{q^n}$ are represented by linear combinations of basis elements $\alpha_0, \ldots, \alpha_{n-1}$ over $\mathbb{F}_q$. In the standard (polynomial) representation $\alpha_i = \alpha^i$ (here α is the generator of the basis, that is, a root of an irreducible polynomial). Of the other representations, the most popular is the normal one. Its basis elements have the form $\alpha_i = \beta^{q^i}$, where β is the generating element of the basis (a normal element of the field).

[7] The method [151] is developed for the RAM-program model.

The idea of transitions between standard and normal bases to speed up computations was apparently first put forward by E. Kaltofen and V. Shoup in [144]. They also constructed a circuit of subquadratic complexity $O(n^{1.82})$ (in operations of the field $\mathbb{F}_q$) for the transition from the normal to the standard representation. An analogous circuit for the transition in the opposite direction was constructed by the author in [288][8]. These transition algorithms are similar to the Brent—Kung modular composition algorithm [50].

Let us consider the problem of constructing circuits over the complete arithmetic basis $\mathcal{A}^{\mathbb{F}_q}$ for the transition between two representations of an arbitrary element $y \in \mathbb{F}_{q^n}$:

$$y = a_0 + a_1\beta + \ldots + a_{n-1}\beta^{n-1} = b_0\beta + b_1\beta^q + \ldots + b_{n-1}\beta^{q^{n-1}}.$$

Let $n \leqslant ms$. If the normal representation (coefficients b_i) is given, we write

$$y = \gamma_0 + \gamma_1^{q^m} + \ldots + \gamma_{s-1}^{q^{m(s-1)}}, \quad \text{where} \quad \gamma_k = b_{mk}\beta + b_{mk+1}\beta^q + \ldots + b_{mk+m-1}\beta^{q^{m-1}}.$$

Then

$$\begin{bmatrix} \gamma_0 \\ \gamma_1 \\ \ldots \\ \gamma_{s-1} \end{bmatrix}^A_{s\times n} = \begin{bmatrix} \gamma_0 \\ \gamma_1 \\ \ldots \\ \gamma_{s-1} \end{bmatrix}^B_{s\times m} \cdot \begin{bmatrix} \beta \\ \beta^q \\ \ldots \\ \beta^{q^{m-1}} \end{bmatrix}^A_{m\times n},$$

where the rows of the matrices labeled by A contain the coefficients of the standard representations of the elements, and the matrix labeled by B contains the coefficients of the normal representation at $\beta, \beta^q, \ldots, \beta^{q^{m-1}}$. Further, the powers of the elements γ_i in the standard basis may be computed via modular composition as

$$f(x)^{q^k} \bmod m_n(x) = f(x^{q^k}) \bmod m_n(x) = f(\xi(x)) \bmod m_n(x), \qquad \xi(x) = x^{q^k} \bmod m_n(x).$$

It remains to compute the sum $\sum \gamma_k$. The complexity of the indicated method (here a modification of [144] from [288] is presented) for small q and $s \approx m \approx \sqrt{n}$ is estimated as $\mathsf{C}(MM_{s,m,n}) + s\mathsf{C}(MC_n) + sn \prec n^{1.63}$ when applying known bounds on the complexity of modular composition [325] (see Theorem 6.4) and multiplication of rectangular matrices [8], when the field characteristic is not too large.

The method [288] of the transition in the opposite direction is based on the following observation: normal representations of elements y and y^{q^k} differ by a cyclic shift of k positions. First, we compute the powers of $y^{q^m}, y^{q^{2m}}, \ldots, y^{q^{(s-1)m}}$. Then, from the known partial normal representations for $1, x, \ldots, x^{n-1}$ we derive partial representations of elements $y^{q^m}, y^{q^{2m}}, \ldots, y^{q^{(s-1)m}}$:

$$\begin{bmatrix} y \\ y^{q^m} \\ \ldots \\ y^{q^{(s-1)m}} \end{bmatrix}^B_{s\times m} = \begin{bmatrix} y \\ y^{q^m} \\ \ldots \\ y^{q^{(s-1)m}} \end{bmatrix}^A_{s\times n} \cdot \begin{bmatrix} 1 \\ x \\ \ldots \\ x^{n-1} \end{bmatrix}^B_{n\times m}.$$

Their combination, by the above remark, provides all normal coordinates of y. The complexity of the algorithm does not exceed $s\mathsf{C}(MC_n) + \mathsf{C}(MM_{s,n,m}) \prec n^{1.63}$.

For the sake of completeness, we note that transition from one standard representation to another is performed by the modular composition operation: if $y = f(\beta)$ in the standard basis with generator β, then $y = f(\xi(x)) \bmod m_n(x)$ in the basis with generator α, which is the root of the polynomial $m_n(x)$, where $\beta = \xi(\alpha)$.

The transition between two normal representations is even simpler: it is easy to check that the transition matrix is circulant[9], so the complexity of the procedure does not exceed $\mathsf{C}(M_n^{\mathbb{F}_q})$ [288].

It is clear that when performing usual arithmetic operations, say, multiplication or division, in a normal basis it is advantageous to pass to a standard basis. An opposite example, when it is

[8] In [144] the transition to the normal representation is performed by a probabilistic algorithm.

[9] Recall that all rows of a circulant matrix are generated by cyclic shifts of the same vector.

advisable to pass from a standard basis to a normal one, is apparently provided by the operator computing all Frobenius automorphisms: $y \to (y^q, y^{q^2}, \ldots, y^{q^n})$. In a normal basis this operation is "free of charge", and for a standard basis in [288] an algorithm of complexity $\preccurlyeq n\mathsf{C}(M_n^{\mathbb{F}_q})$ is proposed that exploits the idea of transition to a normal basis. This algorithm turns out to be slightly faster than the algorithm [108] that does not use transition.

In practice, special normal bases with a simple multiplication table (optimal, Gaussian) are often used — such bases can be found in many fields. But even for them, the standard Massey—Omura multiplication algorithm [215] has quadratic complexity. At the same time, as shown by A. A. Bolotov and S. B. Gashkov [39], such bases have low transitive complexity[10)], usually of order $n \log n$, so the order of complexity of multiplication in these bases is close to $\mathsf{C}(M_n^{\mathbb{F}_q})$. A similar result, exploiting the idea of transition implicitly, was obtained a little earlier in [98].

In [110] the problem of constructing (arithmetic) circuits for transition between polynomial and normal bases in extensions of fields of characteristic 0 is considered. The proposed algorithms have complexity $O(n^{1.99})$ with respect to the extension degree n, but they are probabilistic.

[10]Complexity of transition to the standard representation and backward.

Chapter 7

Special encoding $\boxed{e}$

Transition to an alternative encoding (representation) of inputs in which the required function is computed more easily is especially popular in boolean computations. This technique is close in spirit to the idea of the algebraic method, but is free from structural algebraic restrictions.

Circuit complexity of bit counting $\boxed{e}$

By combining the standard FA_3 circuits of three-bit summation, it is easy to construct an n-bit summation (counting) circuit of complexity $5n+O(\log n)$, see Fig. 7.1.

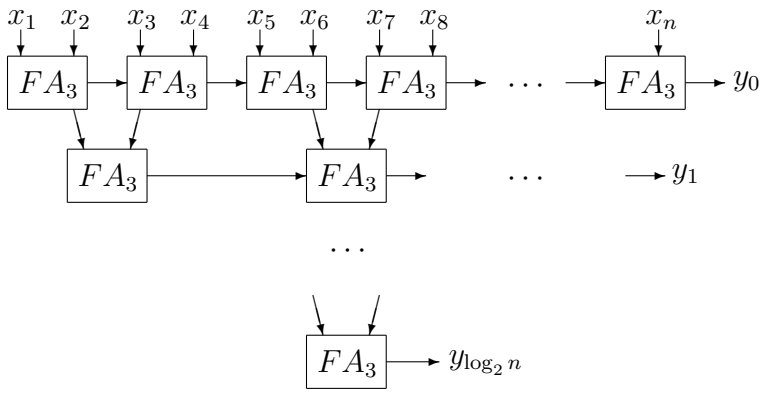


Figure 7.1: Standard n-bit counting circuit

In light of the fact that an n-bit adder composed of FA_3 blocks is optimal [259], it was natural to assume that the same was true for the bit counting operator. Somewhat unexpectedly, in 2010, a group of mathematicians from St. Petersburg [77] discovered that this was not the case.

Theorem 7.1 ([77]). $\mathsf{C}_{\mathcal{B}_2}(\Sigma_{1,n}) \leqslant 4.5n + O(\log n)$.

Alexander Sergeevich Kulikov
St. Petersburg University, 2005 to 2023

► The design exploits the idea of transition from the usual bit notation to encoding two bits x, y by the pair $(x, x \oplus y)$. A special compressor, denoted $MDFA$, performs summation of five bits according to the rule $x_1+x_2+x_3+x_4+x_5 = 2(y_1 + y_2) + y_0$ with complexity 8, if two pairs of inputs are given in the modified encoding, and the pair of outputs (y_1, y_2) is written in the same encoding, see[1)] Fig. 7.2. Now replacing the basic compressor in the original circuit Fig. 7.1, see Fig. 7.3, immediately leads to the required estimate. ■

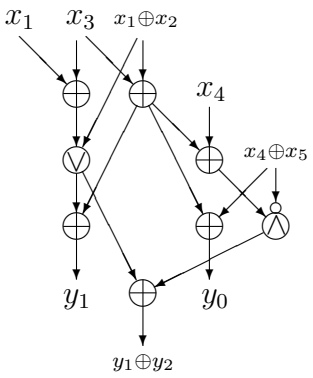


Figure 7.2: $MDFA$ compressor circuit

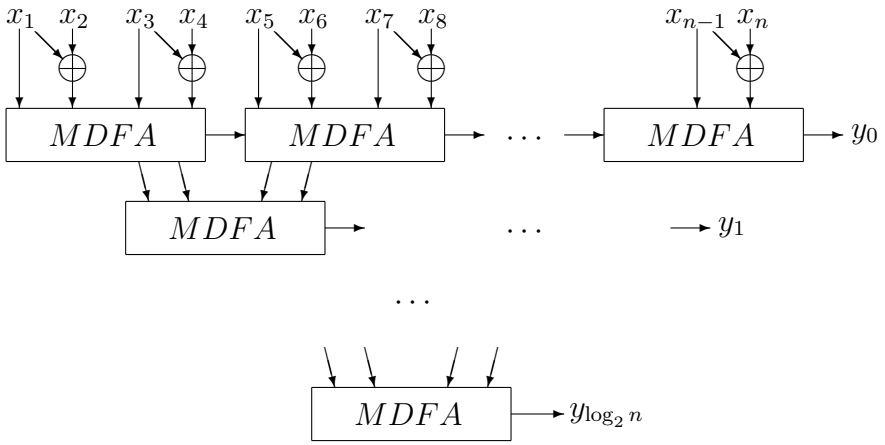


Figure 7.3: n-bit counting circuit composed of $MDFA$ compressors

Corollary 7.1 ([77]). *The complexity of the class $\mathcal{S}_n$ of symmetric boolean functions of n variables satisfies $\mathsf{C}_{\mathcal{B}_2}(\mathcal{S}_n) \leqslant 4.5n + o(n)$.*

[1]Recall that small circles at inputs of elements are invertors.

• The idea of $(x, x \oplus y)$ encoding goes back to the work of L. Stockmeyer [312], who applied it to prove an asymptotically tight bound for the complexity of the summation operator modulo 4: $\mathsf{C}_{\mathcal{B}_2}(\mathrm{MOD}_n^4) = 2.5n \pm O(1)$.

The MDFA compressor also allows to improve the standard bound for the complexity of summing three n-bit numbers. Given that $5n - 3$ operations are required to add two numbers [259], it would be natural to assume that summing three numbers would require about $10n$ operations. However, this is also not the case. By organizing $MDFA$ compressors in a chain (in the same style as FA_3 compressors in the standard adder circuit, see Theorem 1.3), we obtain a triple adder witnessing the complexity bound $\mathsf{C}_{\mathcal{B}_2}(\Sigma_{n,3}) < 9n$.

Real complexity of complex DFT $\boxed{e}$

Theorem 2.4 estimates the complexity of the DFT of order $N = 2^k$ over the field $\mathbb{C}$ asymptotically as $1.5N \log_2 N$ arithmetic operations, and this bound has not yet been improved. From a practical point of view, the complexity bound expressed in real operations is of greater interest. Recall that complex addition or subtraction is performed in 2 real additions or subtractions, and multiplication by a complex constant is performed in 6 real operations (4 scalar multiplications and 2 additions, or 3 multiplications and 3 addition-subtractions). Then from Theorem 2.4 it follows that $\mathsf{C}_{\mathcal{A}_L^{\mathbb{R}}}(\mathrm{DFT}_N[\mathbb{C}]) < 5N \log_2 N$.

A better bound is provided by the so-called "split-radix FFT" algorithm, known since the 1980s, which takes into account that multiplications by 4th roots $\pm\mathbf{i}$ are performed "for free".

Theorem 7.2 ([338]). *Let $N = 2^k$. Then $\mathsf{C}_{\mathcal{A}_L^{\mathbb{R}}}(\mathrm{DFT}_N[\mathbb{C}]) \leqslant 4N \log_2 N$.*

▶ Recall that the components of the DFT of order PQ can be computed by formulas (2.7):

$$x^*_{pQ+q} = \sum_{j=0}^{P-1} (\zeta^Q)^{jp} \cdot \zeta^{jq} \cdot \sum_{i=0}^{Q-1} (\zeta^P)^{iq} x_{iP+j} \tag{7.1}$$

for $p = 0, \ldots, P-1$ and $q = 0, \ldots, Q-1$.

Assuming $P = 2^{k-1}$ and $Q = 2$, we compute the components with even indices, i.e., with $q = 0$, by formula (7.1). For any $p = 0, \ldots, P-1$,

$$x^*_{2p} = \sum_{j=0}^{P-1} (\zeta^2)^{jp} (x_j + x_{P+j}).$$

The computations involve $P = 2^{k-1}$ additions of complex coefficients and performing the DFT of order 2^{k-1}.

To compute the components with odd indices, set $P = 2^{k-2}$, $Q = 4$ and again

apply formula (7.1). For any $p = 0, \ldots, P-1$ we have

$$x^*_{4p+1} = \sum_{j=0}^{P-1} (\zeta^4)^{jp} \cdot \zeta^j \cdot (x_j - x_{2P+j} + \mathbf{i}(x_{P+j} - x_{3P+j})),$$
$$x^*_{4p+3} = \sum_{j=0}^{P-1} (\zeta^4)^{jp} \cdot \zeta^{3j} \cdot (x_j - x_{2P+j} - \mathbf{i}(x_{P+j} - x_{3P+j})).$$

These formulas imply $4P$ complex additions-subtractions, $2P$ multiplications by powers of the primitive root ζ, and two DFTs of order 2^{k-2}.

Finally, we obtain the relation

$$\mathsf{C}_{\mathcal{A}^{\mathbb{R}}_L}(\mathrm{DFT}_N) \leqslant \mathsf{C}_{\mathcal{A}^{\mathbb{R}}_L}(\mathrm{DFT}_{N/2}) + 2\mathsf{C}_{\mathcal{A}^{\mathbb{R}}_L}(\mathrm{DFT}_{N/4}) + 6N,$$

which is resolved as stated, taking into account the initial conditions $\mathsf{C}_{\mathcal{A}^{\mathbb{R}}_L}(\mathrm{DFT}_2) = 4$ and $\mathsf{C}_{\mathcal{A}^{\mathbb{R}}_L}(\mathrm{DFT}_1) = 0$. ■

• The exact complexity of the method is $4N \log_2 N - 6N + 8$. It was published by R. Yavne in [338], but was obtained in a more difficult way. The split-radix FFT method was published almost simultaneously in [81, 214, 328].

For a long time, the bound of Theorem 7.2 seemed impregnable, until J. van Buskirk discovered that it can be improved (published in [199]). The key point in the method is the renormalization of the input vector $X \to \sigma \odot X$, due to which some of the scalar multiplications in the algorithm become simpler. Note that multiplication by a constant of the form $\pm 1 + a\mathbf{i}$ or $a \pm \mathbf{i}$ can be performed via two real additions-subtractions and two multiplications.

Let $\|a\| = \max\{|\Re a|, |\Im a|\}$ denote the l_∞-norm of a complex number $a = \Re a + \Im a \cdot \mathbf{i}$.

Theorem 7.3 ([199]). *Let $N = 2^k$. Then $\mathsf{C}_{\mathcal{A}^{\mathbb{R}}_L}(\mathrm{DFT}_N[\mathbb{C}]) \leqslant 3\frac{7}{9} N \log_2 N + 2N$.*

► Let $\zeta_N = e^{2\pi\mathbf{i}/N}$ be a primitive root of order N in $\mathbb{C}$. For all $j \in \mathbb{Z}$, we define the real coefficients

$$\sigma_{N,j} = \prod_{0 \leqslant l < k/2} \left\| \zeta_N^{j \cdot 4^l} \right\|.$$

Due to $\|\zeta_N^j\| = \left\| \zeta_N^{\pm j \pm N/4} \right\|$ these coefficients satisfy the symmetry properties $\sigma_{N,j} = \sigma_{N,-j}$ and the periodicity properties $\sigma_{N,j} = \sigma_{N,j+N/4}$. Moreover, by construction, $(\sigma_{N/4,j}/\sigma_{N,j})\zeta_N^j = \zeta_N^j / \|\zeta_N^j\|$ has the form $\pm 1 + a\mathbf{i}$ or $a \pm \mathbf{i}$.

Denote the normalized coefficients of the input vector of the DFT by $\dot{x}_j$: $\dot{x}_j = \sigma_{N,j}^{-1} x_j$. We will construct circuits for normalized transforms

$$\mathrm{DFT}'_N(x_0, x_1, \ldots, x_{N-1}) = \mathrm{DFT}_{N,\zeta_N}[\mathbb{C}]\,(\dot{x}_0, \dot{x}_1, \ldots, \dot{x}_{N-1})\,.$$

According to formula (7.1) with the choice of parameters $P = 2^{k-2}$ and $Q = 4$ and the periodicity property of the coefficients $\sigma_{k,j}$, for the components $\dot{x}^*_i$ of the normalized transform DFT'_N the following holds (hereinafter $\zeta = \zeta_N$):

$$\dot{x}^*_{4p+q} = \sum_{j=0}^{P-1} (\zeta^4)^{jp} \cdot \zeta^{jq} \cdot \sigma^{-1}_{N,j} \cdot \gamma_{j,q}, \qquad \gamma_{j,q} = \sum_{r=0}^{3} \mathbf{i}^{rq}\, x_{rP+j}.$$

The inner sums $\gamma_{j,q}$ constitute components of P order-4 DFTs and can be computed via 16 real additions-subtractions each. Further computations for $q = 0,\ 1,\ 3$ are performed according to the formulas

$$\begin{aligned}
\dot{x}^*_{4p} &= \sum_{j=0}^{P-1} (\zeta^4)^{jp} \sigma^{-1}_{N/4,j} \cdot (\sigma_{N/4,j}/\sigma_{N,j}) \cdot \gamma_{j,0}, \\
\dot{x}^*_{4p+1} &= \sum_{j=0}^{P-1} (\zeta^4)^{jp} \sigma^{-1}_{N/4,j} \cdot (\sigma_{N/4,j}/\sigma_{N,j}) \zeta^j \cdot \gamma_{j,1}, \\
\dot{x}^*_{4p+3} &= \sum_{j=0}^{P-1} (\zeta^4)^{j(p+1)} \sigma^{-1}_{N/4,j} \cdot (\sigma_{N/4,j}/\sigma_{N,j}) \zeta^{-j} \cdot \gamma_{j,3}.
\end{aligned} \tag{7.2}$$

(Note the cyclic shift of the vector of coefficients of the outer DFT in the last sum — it allows to reduce multiplications by ζ^{3j} to simpler multiplications by normalized numbers ζ^{-j}.)

Computations by formulas (7.2) involve $N/4$ multiplications by real constants, $N/2$ multiplications by constants of the form $\pm 1 + a\mathbf{i}$ or $a \pm \mathbf{i}$, and three type-$\mathrm{DFT}'_{N/4}$ transforms.

To compute the remaining components $\dot{x}^*_{4p+2}$, we apply formula (7.1) with parameters $P = 2^{k-3}$ and $Q = 8$:

$$\begin{aligned}
\dot{x}^*_{8p+2} &= \sum_{j=0}^{P-1} (\zeta^8)^{jp} \sigma^{-1}_{N/8,j} \cdot (\sigma_{N/8,j}/\sigma_{N/2,j}) (\zeta^2)^j \cdot \alpha_j, \\
\dot{x}^*_{8p+6} &= \sum_{j=0}^{P-1} (\zeta^8)^{j(p+1)} \sigma^{-1}_{N/8,j} \cdot (\sigma_{N/8,j}/\sigma_{N/2,j}) (\zeta^2)^{-j} \cdot \beta_j,
\end{aligned}$$

where

$$\begin{aligned}
\alpha_j &= (\sigma_{N/2,j}/\sigma_{N,j})\gamma_{j,2} + \mathbf{i}(\sigma_{N/2,j+N/8}/\sigma_{N,j+N/8})\gamma_{j+N/8,2}, \\
\beta_j &= (\sigma_{N/2,j}/\sigma_{N,j})\gamma_{j,2} - \mathbf{i}(\sigma_{N/2,j+N/8}/\sigma_{N,j+N/8})\gamma_{j+N/8,2}.
\end{aligned}$$

Recall that $\sigma_{N/2,j} = \sigma_{N/2,j+N/8}$. These computations are performed via $N/4$ multiplications by real or imaginary constants, $N/4$ multiplications by constants of the form $\pm 1 + a\mathbf{i}$ or $a \pm \mathbf{i}$, and two type-$\mathrm{DFT}'_{N/8}$ transforms.

As a result, we obtain the relation

$$\mathsf{C}_{\mathcal{A}^{\mathbb{R}}_L}(\mathrm{DFT}'_N) \leqslant 3\mathsf{C}_{\mathcal{A}^{\mathbb{R}}_L}(\mathrm{DFT}'_{N/4}) + 2\mathsf{C}_{\mathcal{A}^{\mathbb{R}}_L}(\mathrm{DFT}'_{N/8}) + 8.5N,$$

which, in accordance with the initial conditions $\mathsf{C}(\mathrm{DFT}'_4) \leqslant 16$, $\mathsf{C}(\mathrm{DFT}'_2) = 4$ and $\mathsf{C}(\mathrm{DFT}'_1) = 0$, is resolved as $\mathsf{C}_{\mathcal{A}_L^{\mathbb{R}}}(\mathrm{DFT}'_N) \leqslant 3\frac{7}{9} N \log_2 N$. It remains to take into account $2N$ operations for the transition from the coordinates X to $\dot{X}$. ■

• The method of Theorem 7.3 is also explained in [137, 27]. An accurate complexity of the algorithm is given by

$$\frac{34}{9} N \log_2 N - \frac{124}{27} N - 2\left(1 + \frac{(-1)^k}{9}\right) \log_2 N + 8 + \frac{16}{27} \cdot (-1)^k.$$

Additionally applying a trick of reducing DFT to Walsh—Hadamard transforms, J. Alman and K. Rao [10] improved the bound for the real complexity of the complex DFT to $\mathsf{C}_{\mathcal{A}_L^{\mathbb{R}}}(\mathrm{DFT}_N[\mathbb{C}]) \leqslant 3.75 N \log_2 N + O(N)$.

Matrix multiplication. Speeding up Strassen's method $\boxed{e}\boxed{/_2}$

Of the theoretically fast matrix multiplication methods, Strassen's method is of greatest practical interest [313]. Efforts of many researchers have focused on optimizing the algorithm's running time. Recall that in the basic scheme of the Strassen method in Winograd's modification, the multiplication of matrices of size 2×2 is performed via 7 multiplications and 15 additive operations in a ring. Whereas the multiplicative complexity of the basic algorithm determines the exponent of the complexity of multiplication of $n \times n$ matrices, the basic additive complexity is approximately proportional to the multiplicative constant in the complexity estimate of the general multiplication. Thus, according to (5.10) for $n = 2^k$ we obtain $\mathsf{C}_{\mathcal{A}^R}(M\!M_n) \leqslant 6n^{\log_2 7}$.

M. Bodrato [38] observed that the fast 2×2 matrix multiplication can be accomplished in 12 additive operations if an alternative representation $X \to \widehat{X}$ is used (see also [149]). For example [149], let

$$\widehat{X} = \begin{bmatrix} \widehat{x}_{11} & \widehat{x}_{12} \\ \widehat{x}_{21} & \widehat{x}_{22} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} - x_{21} + x_{22} \\ x_{22} - x_{21} & x_{12} + x_{22} \end{bmatrix}. \tag{7.3}$$

Now the product $\widehat{Z} = \widehat{X}\widehat{Y}$ can be calculated by formulas

$$\begin{gathered} u_1 = \widehat{x}_{11}\widehat{y}_{11}, \quad u_2 = \widehat{x}_{12}\widehat{y}_{12}, \quad u_3 = \widehat{x}_{21}\widehat{y}_{21}, \quad u_4 = \widehat{x}_{22}\widehat{y}_{22}, \\ u_5 = (\widehat{x}_{12} - \widehat{x}_{21})(\widehat{y}_{22} - \widehat{y}_{12}), \quad u_6 = (\widehat{x}_{12} - \widehat{x}_{11})(\widehat{y}_{12} - \widehat{y}_{21}), \quad u_7 = (\widehat{x}_{22} - \widehat{x}_{12})(\widehat{y}_{12} - \widehat{y}_{11}), \\ \widehat{z}_{11} = u_1 + u_5, \quad \widehat{z}_{12} = u_2 + u_5 - u_6 + u_7, \quad \widehat{z}_{21} = u_3 + u_7, \quad \widehat{z}_{22} = u_4 - u_6, \end{gathered} \tag{7.4}$$

which involve 7 multiplications and 12 additions-subtractions.

Theorem 7.4 ([38, 149]). *Let $n = 2^k$. Then for a ring R,*

$$\mathsf{C}_{\mathcal{A}^R}(M\!M_n) \leqslant 5n^{\log_2 7} + O(n^2 \log n).$$

► Let us recursively extend the alternative representation to matrices of size $2^k \times 2^k$. For $k = 1$, the representation $\widehat{X}$ is defined by formulas (7.3). If the representation

for matrices of size $n/2 \times n/2$ is already defined, then the representation of an $n \times n$ matrix $X = \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix}$ is defined as[2)]

$$\widehat{X} = \begin{bmatrix} \widehat{X}_{11} & \widehat{X}_{12} - \widehat{X}_{21} + \widehat{X}_{22} \\ \widehat{X}_{22} - \widehat{X}_{21} & \widehat{X}_{12} + \widehat{X}_{22} \end{bmatrix}. \tag{7.5}$$

Lemma 7.1. *For the complexity of conversion between the two representations of an $n \times n$ matrix X, the following bound holds:*

$$\mathsf{C}_{\mathcal{A}}(X \to \widehat{X}), \mathsf{C}_{\mathcal{A}}(\widehat{X} \to X) \leqslant \frac{3}{4} \cdot n^2 \log_2 n.$$

▷ A single application of formulas (7.3) or (7.5) costs 3 additive operations. The same is true for the other direction: for $n = 2$,

$$X = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} = \begin{bmatrix} \widehat{x}_{11} & \widehat{x}_{12} - \widehat{x}_{21} \\ \widehat{x}_{22} - \widehat{x}_{12} & \widehat{x}_{21} - \widehat{x}_{12} + \widehat{x}_{22} \end{bmatrix}.$$

Thus, for the complexity $T(n)$ of the transition between the representations of an $n \times n$ matrix, we have the recurrence relation $T(n) \leqslant 4T(n/2) + 3n^2/4$, which is resolved exactly as stated. □

Let $\widehat{MM}_n$ denote the matrix multiplication operator in the alternative representation. When it is implemented by the method of Theorem 2.3 or 5.3, a recursive call of the multiplication algorithms of smaller and smaller dimensions is performed, and each time the matrices being multiplied have a suitable form for applying formulas (7.4). For the complexity of $\widehat{MM}_n$, one can employ estimate (5.10)[3)], in which $m = 2$, $r = 7$ and $s = 12$. So we obtain

$$\mathsf{C}(MM_n) \leqslant \mathsf{C}(\widehat{MM}_n) + 3T(n) \leqslant 5n^{\log_2 7} - 4n^2 + \frac{9}{4} \cdot n^2 \log_2 n.$$

■

The bound of Theorem 7.4 is somewhat conventional, since in practice for small n it is not the Strassen method that is called, but, say, the standard multiplication algorithm — in this case the multiplicative constant in the complexity estimate turns out to be even smaller (see, e.g., [55]).

• Bodrato [38] also showed that for any representation of 2×2 matrices by minimal length codes, the additive complexity of a Strassen-type multiplication algorithm is at least 12.

E. Karstadt and O. Schwartz [149] also found suitable matrix representations for other potentially interesting algorithms, in particular for Smirnov's algorithm [308] based on fast multiplication of 3×3 and 3×6 matrices (see also [24]). Moreover, even more economical representations exist for rings of characteristic 2, see [148].

[2]It should be noted that the matrix notation for the alternative representation is conditional: the matrix product in the modified representation is determined by different rules than in the standard one.

[3]It is important here that the new encoding preserves the size of matrices, so the complexity of additive matrix operations does not change.

A similar approach for accelerating Strassen's algorithm, based on a different encoding, though redundant, was proposed by M. Cenk and M. Hasan [55]. Their algorithm leads to the same estimate as in Theorem 7.4, $\lesssim 5n^{\log_2 7}$, and under the condition of employing the standard multiplication algorithm for small matrix sizes — to the estimate $\mathsf{C}(MM_n) \lesssim 3.55n^{\log_2 7}$ (for $n = 2^k$).

Formula complexity of summation modulo 5 $\boxed{e}\boxed{U}\boxed{/_2}\boxed{\because}$

Consider the problem of computing the sum of n boolean variables modulo m over the basis $\mathcal{B}_0$. Simple formulas (4.1) lead to the bound $\Phi_{\mathcal{B}_0}(\mathrm{MOD}_n^m) \preccurlyeq n^{\log_2 m+1}$ [206]. For $m = 3$ this bound has not yet been improved, but for larger m shorter formulas may be constructed by the method proposed by the author in [297]. It is based on a specially extended encoding of inputs.

Let $S \subset \mathbb{Z}_m$. We define the functions

$$\mathrm{MOD}_n^{m,S}(X) = \left(\sum_{i=1}^{n} x_i \bmod m \ \in S\right).$$

Clearly, the set of all functions $\mathrm{MOD}_n^{m,S}$, $0 < |S| < m$, defines the value of the sum of variables modulo m.

To each set S we associate a boolean $m \times m$ matrix I_m^S, whose rows and columns are labeled by digits from $\mathbb{Z}_m$, and whose entries are defined as $I_m^S[i,j] = (i+j \in S)$. Consider a covering of the matrix I_m^S by rectangles (all-ones submatrices): let the k-th rectangle be located at the intersection of rows A_k and columns B_k. Then for $X = (X^1, X^2)$, $|X| = n$, $|X^i| = n_i$,

$$\mathrm{MOD}_n^{m,S}(X) = \bigvee_k \mathrm{MOD}_{n_1}^{m,A_k}(X^1) \cdot \mathrm{MOD}_{n_2}^{m,B_k}(X^2). \tag{7.6}$$

From (4.1) we deduce the identity

$$\mathrm{MOD}_n^{m,S}(X) = \bigvee_{k=0}^{m-1} \mathrm{MOD}_{n_1}^{m,k}(X^1) \cdot \mathrm{MOD}_{n_2}^{m,S-k}(X^2), \tag{7.7}$$

corresponding to the trivial covering of the matrix by separate rows (here $S - k$ denotes the set $\{r - k \mid r \in S\}$). The rank of this covering (the number of covering submatrices) is m.

Nontrivial complexity bounds can be obtained from coverings of rank $< m$. Though in the case $m = 3$ for all $S \neq \varnothing, \mathbb{Z}_m$ the matrices I_m^S have full rank, already for $m = 5$ the matrices $I_m^{\mathbb{Z}_m \setminus \{r\}}$ (coinciding with $\overline{I}_m$ up to a permutation of rows and columns) have rank 4, and the corresponding covering is formed by rectangles with sides 2 and 3, see Fig. 7.4.

Theorem 7.5 ([297]). $\Phi_{\mathcal{B}_0}(\mathrm{MOD}_n^5) \prec n^{3.22}$.

▶ So, functions $\mathrm{MOD}_n^{5,S}$ for $|S| = 4$ may be implemented by formulas (7.6) with 4 summands, and for $|S| = 1$ — by dual formulas (conjunctions of disjunctions) of

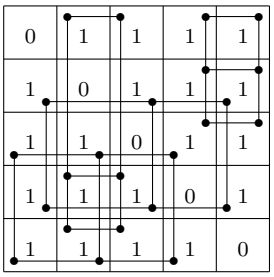


Figure 7.4: Covering of the matrix $\overline{I_5}$

the same size. For the remaining functions $\mathrm{MOD}_n^{5,S}$ we choose expression (7.7), since a matrix I_5^S for $2 \leqslant |S| \leqslant 3$ has full rank.

Applying the potential method, we estimate the complexity of the formulae that this strategy leads to. Let p_n denote the complexity of formulae for $\mathrm{MOD}_n^{5,S}$, $|S| \in \{1, 4\}$, and let q_n denote the complexity of formulae for $\mathrm{MOD}_n^{5,S}$, $|S| \in \{2, 3\}$. Define $r_n = \max\{p_n, q_n/a\}$, where the parameter a will be chosen later. From (7.6) and (7.7) the inequality

$$r_{2n} \leqslant \max\{8q_n, 5(p_n + q_n)/a\}$$

follows. In order to minimize the ratio r_{2n}/r_n, we choose $a = \frac{5+\sqrt{185}}{16}$. As a result we obtain $r_{2n} \leqslant 8ar_n$. Therefore, $\Phi_{\mathcal{B}_0}(\mathrm{MOD}_n^5) \preccurlyeq n^{3+\log_2 a} \prec n^{3.22}$. ■

• A minor refinement of the bound of Theorem 7.5 is possible with an additional selection of the optimal ratio n_1/n_2. The above method also leads to the bounds [297]

$$\Phi_{\mathcal{B}_0}(\mathrm{MOD}_n^7) \prec n^{3.63}, \qquad \mathsf{D}_{\mathcal{B}_0}(\mathrm{MOD}_n^5) \lesssim 3.35 \log_2 n, \qquad \mathsf{D}_{\mathcal{B}_0}(\mathrm{MOD}_n^7) \lesssim 3.87 \log_2 n.$$

For larger m, the general bounds on the depth and complexity of the operator $\Sigma_{1,n}$ have priority, see (7.9).

Fast exponentiation of polynomials $\boxed{e}$

Consider the problem of fast exponentiation of a polynomial $f(x)^m$ in a quotient ring $R[x]/(p(x))$, where $\deg f < \deg p = n$. When implementing finite field arithmetic, one has to deal with such operations regularly.

A straightforward approach is to perform successive multiplications with reduction modulo $p(x)$. If we compute powers of $f(x)$ following a minimal addition chain for m, then a total of $\mathsf{L}(m) \sim \log_2 m$ steps of complexity $\mathsf{C}_{\mathcal{A}^R}(M_n^R) + \mathsf{C}_{\mathcal{A}^R}(QR_{2n,n}^R)$ each are required. Even in the favorable case (Theorem 5.2), division with remainder "costs" about two multiplications, and usually — somewhat more. P. Montgomery [220] observed that multiplication in a quotient ring can be simplified by switching to a special representation of elements. An adaptation of Montgomery's integer method for polynomials is presented below.

Let $b(x) = x^{n-1}$ and $q(x) = b^{-1}(x) \bmod p(x)$ (assuming that $p(x)$ has a nonzero

constant term, the polynomials b and p are coprime[4]). Consider the transform $f \to f^* = fb \bmod p$. It establishes a correspondence which is in fact a homomorphism if the operation of addition of images is defined in a natural way and the multiplication is defined as $f \star g = fgq \bmod p$ (Montgomery multiplication). Then $(f \pm g)^* = f^* \pm g^*$ and $(fg)^* = f^* \star g^*$.

Let us denote $h(x) = -p^{-1}(x) \bmod b(x)$. The following lemma suggests a simple way to implement Montgomery multiplication.

Lemma 7.2. *If* $\deg f \leqslant 2n-2$, *then*

$$\frac{f + ((f \bmod b)h \bmod b)p}{b} = fq \bmod p. \tag{7.8}$$

▷ The numerator of the fraction is divisible by b, since $b \mid f(1+hp)$. Therefore, the fraction is a polynomial of degree $< n$, which, as we see when multiplying by bq, belongs to the same equivalence class modulo p as the polynomial fq. Thus, the two polynomials simply coincide. □

Formula (7.8), if f is replaced by the product fg of polynomials of degree $< n$, means that the complexity of Montgomery multiplication $f \star g$ does not exceed $3\mathsf{C}_{\mathcal{A}^R}(M_n^R) + O(n)$. The transition from f to $f^* = f \star (b^2 \bmod p)$ costs three multiplications, and in the opposite direction — two.

Theorem 7.6. *Let* $p(x) \in R[x]$, $\deg p = n$, *and* $x \nmid p(x)$. *Raising to a fixed power* m *in the ring* $R[x]/(p(x))$ *can be performed by a circuit of complexity* $(3\mathsf{L}(m)+3)(\mathsf{C}_{\mathcal{A}^R}(M_n^R) + O(n))$ *over the basis* $\mathcal{A}^R$.

► Guided by the shortest addition chain $a_0 = 1, a_1, \ldots, a_k = m$ for m, we sequentially compute the polynomials $\varphi_i = f^{a_i} q^{a_i - 1} \bmod p$ according to rule (7.8), starting with $f = f^{a_0} q^{a_0 - 1} \bmod p$. Finally, $f^m \bmod p = \varphi_k \star a$, where $\varphi_k = f^m q^{m-1} \bmod p$, and the polynomial $a = b^m \bmod p$ is assumed to be precomputed. ■

A more elegant scheme computing $f \to f^* \to (f^m)^* \to f^m$ (modulo p) makes full use of the auxiliary encoding, but requires two more multiplications. However, this method is efficient in the case of a non-fixed polynomial p or when the number m is also an input to the algorithm and is given by a binary code (which often occurs in practice; then a binary addition chain for m is applied). Of course, Montgomery encoding is suitable for a wide range of arithmetic problems in quotient rings, not only for exponentiation.

• Almost without changes, the technique of modular computations may be adapted to number rings $\mathbb{Z}_p$ — it was originally developed for them [220]. Usually p is a prime n-digit number, and $b = 2^n$. Lemma 7.2 is refined as follows: for $f < p^2$, the left-hand side (7.8) either coincides with the right-hand side or exceeds it by p.

An alternative to Montgomery's method is P. Barrett's [15] method. The method is based on substitution of the divisor: the quotient $\lfloor f/p \rfloor$ is approximately $\lfloor \lfloor f/b \rfloor \cdot \lfloor b^2/p \rfloor / b \rfloor$. If we precompute

[4] If $p(x) = p_0(x)x^k$, where $x \nmid p_0(x)$, then the problem becomes simpler, since the computations can be performed separately modulo x^k and modulo $p_0(x)$. In applications (for example, in finite field arithmetic), the polynomial p is usually irreducible, in particular, $x \nmid p(x)$.

$\lfloor b^2/p \rfloor$, then multiplication modulo p may be performed via three ordinary multiplications and several linear complexity operations.

Other applications

Formulae for symmetric boolean functions. An arbitrary symmetric boolean function of n variables has the form $h(\Sigma_{1,n}(X))$, and the standard method for computing it is as follows: first, calculate the arithmetic sum of the inputs, and then implement the function h of $\log_2 n$ variables. In this case, since different digits of the arithmetic sum, generally speaking, have different complexity and depth, in the second step, as shown in [156, 242], it is advantageous to employ the cascade method[5]. Thus, in the case of standard $(3,2)$-compressors, the upper bounds for the depth of the counting operator $\Sigma_{1,n}$ and the class of symmetric functions $\mathcal{S}_n$ are related as [242]

$$\mathsf{D}_{\mathcal{B}_2}(\Sigma_{1,n}) \lesssim 3.71 \log_2 n, \qquad \mathsf{D}_{\mathcal{B}_2}(\mathcal{S}_n) \lesssim 3.81 \log_2 n$$

(see Corollary 4.1 for the proof of the former bound).

In the author's method [296, 306] the sum $\Sigma = \Sigma_{1,n}(X)$ is encoded by the set $(\{\Sigma_q | q = 2, 3, [\ldots]\}, \Sigma')$, where $\Sigma_q = \Sigma \bmod q^{k_q}$ and $|\Sigma' - \Sigma| < E$ for $\prod q^{k_q} \geqslant 2E$. The value of Σ_2 is computed by the modification of the compressor method proposed in [242], and in the general case Σ_q is computed similarly, only in the q-ary number system with the use of special q-ary compressors. The approximate sum Σ' may be computed by L. Valiant's method [326], see below on p. 106. Then, an absolute value of the sum $\Sigma = \Sigma_{1,n}(X)$ may be recovered from its code via a simple arithmetic procedure in the spirit of the Chinese remainder theorem. A general symmetric function can be computed directly as $h(\{\Sigma_q\}, \Sigma')$, without restoring Σ. The efficiency of the approach relies on the fact that the least significant digits of sums in the compressor method are easier to compute than the most significant ones. On this path, record estimates to date have been obtained [306]:

$$\begin{gathered}\Phi_{\mathcal{B}_0}(\Sigma_{1,n}), \Phi_{\mathcal{B}_0}(\mathcal{S}_n) \preccurlyeq n^{3.77}, \qquad \Phi_{\mathcal{B}_2}(\Sigma_{1,n}) \preccurlyeq n^{2.82}, \qquad \Phi_{\mathcal{B}_2}(\mathcal{S}_n) \preccurlyeq n^{2.85}, \\ \mathsf{D}_{\mathcal{B}_0}(\Sigma_{1,n}), \mathsf{D}_{\mathcal{B}_0}(\mathcal{S}_n) \lesssim 3.96 \log_2 n, \qquad \mathsf{D}_{\mathcal{B}_2}(\Sigma_{1,n}), \mathsf{D}_{\mathcal{B}_2}(\mathcal{S}_n) \lesssim 2.98 \log_2 n.\end{gathered} \tag{7.9}$$

Slightly better estimates were obtained for threshold symmetric functions, in particular, for the majority function. However, the above estimates are non-constructive due to the method of computing Σ'. A constructive version of the method, basing on a shortened encoding (Σ_2, Σ_3), was proposed by the author earlier in [293, 294].

Almost monotone complexity of boolean functions. In monotone or mostly monotone computations, preliminary sorting of input vectors often allows one to construct simpler circuits. Let us consider a well-known example. In 1957, A. A. Markov [213] discovered a fundamental fact: any system of boolean functions of n variables can be implemented by a circuit over the basis $\mathcal{B}_0$ including only $b(n) = \lceil \log_2(n+1) \rceil$ negation elements[6]. Any circuit over $\mathcal{B}_0$ can be transformed quite efficiently into a circuit with $b(n)$ negations. First, negations are lowered to the input level according to De Morgan's rules, while the complexity of the circuit at most doubles. After this, it remains to implement the operator $V_n = (\overline{x}_1, \ldots, \overline{x}_n)$, using negation elements sparingly.

The record upper bound $O(n \log n)$ for the complexity of computing V_n by a circuit with $b(n)$ negations was obtained by R. Beals [18] (see also [19]). The method is based on M. Fischer's observation [87] that on an ordered set of inputs the operator V_n can be implemented simply, with linear complexity.

Indeed, for $n = 3$ and $x_1 \geqslant x_2 \geqslant x_3$ a circuit with $b(3) = 2$ negations is constructed as follows. First, compute $\overline{x}_2$, then $x_1\overline{x}_2 \vee x_3$, then $\overline{(x_1\overline{x}_2 \vee x_3)} = (\overline{x}_1 \vee x_2)\overline{x}_3$. It remains to note that

$$\overline{x}_2((\overline{x}_1 \vee x_2)\overline{x}_3) = \overline{x}_1\overline{x}_2\overline{x}_3 = \overline{x}_1 \quad \text{and} \quad \overline{x}_2 \vee ((\overline{x}_1 \vee x_2)\overline{x}_3) = \overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 = \overline{x}_3.$$

[5] The method consists of successively decomposing the function in a selected variable: $f(x_1, x_2, \ldots, x_n) = x_1 f(1, x_2, \ldots, x_n) \vee \overline{x}_1 f(0, x_2, \ldots, x_n)$, see, e.g., [207].

[6] Moreover, he indicated a rule that allows one to determine the inversion complexity (the minimum number of negations in a circuit) of a function from the table of its values. For an arbitrary function of n variables, the inversion complexity does not exceed $\lceil \log_2(n+1) \rceil$.

The method is generalized to the case of an arbitrary $n = 2k-1$: the described calculations are performed with triplets $x_i \geqslant x_k \geqslant x_{k+i}$, and in the middle an algorithm is called that computes V_{k-1}.

It is easy to check that the overall complexity of the circuit does not exceed $4n$. The benefit of ordering the input set is also demonstrated by the lower bound proved by K. Tanaka and T. Nishino [316]: in the general case, the complexity of implementing V_n by circuits with $b(n)$ negations cannot be less than $5n$ (for $n \geqslant 3$).

A downside of the monotone encoding is a nonlinear complexity of the transition to it and back[7]. The transition $x_1, \ldots, x_n \overset{\mathrm{SORT}_n}{\longrightarrow} x_{\pi(1)}, \ldots, x_{\pi(n)}$ is performed by a comparator circuit[8] with complexity $\preccurlyeq n \log n$, if we apply some AKS-like method. The key observation of Beals [18] is that to return to the original order of variables $\overline{x}_{\pi(1)}, \ldots, \overline{x}_{\pi(n)} \longrightarrow \overline{x}_1, \ldots, \overline{x}_n$ one can reverse the original sorting circuit. Indeed, if a comparator in the circuit implements $(f, g) \to (f \vee g, f \cdot g)$, then, given $\overline{f \vee g} = \overline{f} \cdot \overline{g}$ and $\overline{f \cdot g} = \overline{f} \vee \overline{g}$, the functions $\overline{f}$ and $\overline{g}$ may be computed by formulas

$$\overline{f} = \overline{f} \cdot \overline{g} \vee (\overline{f} \vee \overline{g})g, \qquad \overline{g} = \overline{f} \cdot \overline{g} \vee (\overline{f} \vee \overline{g})f.$$

As a consequence, the transition to the original order of variables is also performed with complexity $\preccurlyeq n \log n$. Thus, information about the structure of the sorting circuit complements the monotone encoding of the input set.

Instead of sorting, one can use the median selection in the above method; the circuits are simpler, but the order of complexity does not change[9]. The question of the existence of a circuit of linear complexity for V_n remains open. H. Morizumi and G. Suzuki [223] constructed a circuit of linear complexity with $\log^{1+o(1)} n$ negation elements.

Interpolation and evaluation at points of arithmetic progression. Recall that the evaluation of a polynomial of degree $< n$ on a set of n points, as well as the inverse problem of reconstructing the coefficients of a polynomial from the values at these points, may be performed with complexity $\preccurlyeq \mathsf{M}(n) \log n$ (Lemma 6.4): refined estimates in [43] have the form $\lesssim 1.5\mathsf{M}(n) \log_2 n$ and $\lesssim 2.5\mathsf{M}(n) \log_2 n$ for $n = 2^k$, respectively. These operations can be implemented even faster if the set of points has a special structure. In particular, for the case of an arithmetic progression, J. Gerhard [109] proposed a slightly more efficient algorithm. It is based on the transition from the standard polynomial notation $f(x) = \sum f_i x^i$ to Newtonian form.

Let $\alpha_0, \ldots, \alpha_{n-1} \in R$ denote the set of interpolation points. The Newtonian representation of a polynomial $f(x)$ of degree $< n$ is characterized by a coefficient vector $(g_0, \ldots, g_{n-1})$, where

$$f(x) = g_0 + g_1(x - \alpha_0) + g_2(x - \alpha_0)(x - \alpha_1) + \ldots + g_{n-1}(x - \alpha_0) \cdot \ldots \cdot (x - \alpha_{n-2}).$$

Let $\alpha_i = \alpha_0 + ih$, where h is the difference of the arithmetic progression. By the identity

$$V(x) = G(x)S(x) \bmod x^n, \qquad \text{where} \quad V(x) = \sum_{i=0}^{n-1} \frac{f(\alpha_i)}{i!h^i} x^i, \quad G(x) = \sum_{i=0}^{n-1} g_i x^i, \quad S(x) = \sum_{i=0}^{n-1} \frac{1}{i!h^i} x^i,$$

all values $f(\alpha_i)$ may be computed from given g_i with complexity $\mathsf{C}_{\mathcal{A}}(M_n^R) + O(n)$. The inverse problem has the same complexity, since

$$G(x) = V(x)S^{-1}(x) \bmod x^n, \qquad \text{where} \quad S^{-1}(x) = \sum_{i=0}^{n-1} \frac{(-1)^i}{i!h^i} x^i.$$

The transition between the standard and Newtonian representations of a polynomial is actually performed by the method of Lemma 6.4 using an auxiliary tree. A careful estimate of the complexity

[7] We are talking, of course, about the monotone part of the code used for the input set.

[8] Recall that a comparator is a circuit that performs the transform $(x, y) \to (x \vee y, xy)$. A comparator circuit is actually a formula composed of comparator elements, in the sense that branching of circuit inputs and outputs of comparators is prohibited. Such circuits implement various partial orders on the set of variables.

[9] The lower bound for the complexity of comparator circuits for selecting median is $\Omega(n \log n)$ [7].

of the transition in either direction, provided by A. Bostan and É. Schost [43], is $\lesssim \mathsf{M}(n) \log_2 n$ for $n = 2^k$. Consequently, the same bound holds for the complexity of evaluation at the points of arithmetic progression and the inverse interpolation problem.

Note that in the case when the points $\alpha_0, \ldots, \alpha_{n-1}$ form a geometric progression, the complexity of evaluation and interpolation if of order $\mathsf{M}(n)$ in both the standard and Newtonian representations of polynomials [43].

Chapter 8

Duality principles

The concept of duality is an efficient tool in synthesis theory. In general, it allows one to prove the closeness of the complexity of solving two problems that are dual in a certain sense. Thus, having constructed a fast algorithm for a dual problem, one can obtain an equally efficient solution (or establish its existence) for the original one.

Complexity of universal matrices. Transposition principle

Consider the problem of computing a linear mapping with a boolean $k \times 2^k$ matrix Υ_k composed of all possible different columns of height k.

$$\Upsilon_3 = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

Universal matrices play an important role in synthesis theory and are also check matrices for Hamming codes.

Theorem 8.1. $\mathsf{L}(\Upsilon_k) = 2^{k+1} - 2k - 2$.

▶ The result of the theorem follows from a more general fact known as the *transposition principle*. It was probably first formulated by B. S. Mityagin and B. N. Sadovskii [217] in 1965.

Lemma 8.1 ([217])**.** *In any commutative semigroup $(G, +)$ for any $m \times n$ matrix A without zero rows and columns,*

$$\mathsf{L}_+(A) + m = \mathsf{L}_+(A^T) + n. \tag{8.1}$$

▷ It is convenient to represent a circuit that performs the transform $X \to AX$ as a graph oriented from inputs to outputs. At each node, intermediate sums received from the incoming edges are added. If a node has q input edges, then the sum is

computed in $q-1$ operations. Note that the equivalent complexity of the circuit in terms of binary additions is equal to the difference between the number of edges and the number of nodes, not counting the inputs.

Further, note that an entry $A[i,j]$ of the matrix A contains the number of oriented paths between the j-th input and the i-th output (the paths are counted in accordance with the group operation).

Now, if in an arbitrary circuit S we reverse the orientation of edges, the resulting circuit will compute the transposed matrix A^T (by the property of preserving the number of paths), and its equivalent complexity will be $\mathsf{L}(S)+m-n$ (the total number of vertices in the circuit graph is preserved, only the outputs and inputs interchange). □

Now the theorem is proved quite simply. The transposed matrix Υ_k^T contains 2^k-k-1 distinct rows of weight $\geqslant 2$. According to Lemma 3.1, this lower complexity estimate is also attainable from above. Hence, $\mathsf{L}(\Upsilon_k^T)=2^k-k-1$. It remains to apply Lemma 8.1, not forgetting to take into account the presence of a zero column in Υ_k. ■

The upper bound of Theorem 8.1 can also be proved by the bisection method, but applying Lemma 8.1 simultaneously provides a proof of the optimality of the circuit. On the other hand, although the transposition principle is constructive, circuits obtained with its help can have an intricate structure.

• Theorem 8.1 in particular means that $\mathsf{L}_{\oplus}(\Upsilon_k)=2^{k+1}-2k-2$. Moreover, A. V. Chashkin [56] (see also [57]) showed that adding nonlinear operations to the basis does not lead to an improvement: $\mathsf{C}_{\mathcal{B}_2}(L_{\Upsilon_k})=2^{k+1}-2k-2$. Most likely, this is typical for linear operators in general. On the other hand, without linear operations things are difficult: as A. S. Kulikov, O. Melanich and I. Mikhailin [178] proved, $\mathsf{C}_{\mathcal{U}_2}(L_{\Upsilon_k})\geqslant 5(2^k-k-1)$.

In an extended linear basis, an analogue of Lemma 8.1 holds, see Lemma 8.6 below.

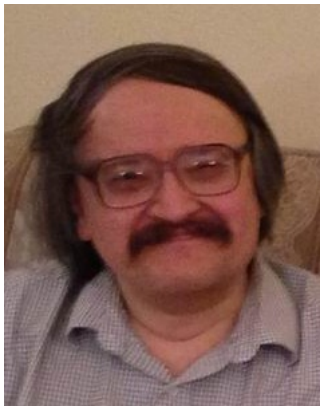
The transposition principle allows to establish that bound (3.4) of the additive complexity of the set of numbers $n_1,\ldots,n_s$ also holds for the complexity of computing the vector $(n_1,\ldots,n_s)$, in other words, for the complexity of the linear transform $x_1,\ldots,x_s\to n_1x_1+\ldots+n_sx_s$.

In general, Lemma 8.1 turns out to be useful if not in obtaining fundamentally new results, then in simplifying proofs. For example, Lemma 3.3 may be proved a bit more simply via the transposition principle, and the estimate is somewhat more accurate.

Parallel adders. Grinchuk's method đ /2

The result of Theorem 2.6 on the complexity of computing a system of carries cannot be improved without additional assumptions about the semiring R in which the computations are performed. But in the most interesting case $R=(\mathbb{B},\vee,\wedge)$ M. I. Grinchuk [114] obtained a stronger estimate, essentially exploiting the duality of disjunction and conjunction.

Recall that the dual of a boolean function f is denoted by f^* and is defined as $f^*(X)=\overline{f(\overline{X})}$. A circuit for f turns into a circuit for f^* when all elements are replaced by their duals (this is the *duality principle*). In particular, $\mathsf{C}_{\mathcal{B}_M}(f)=\mathsf{C}_{\mathcal{B}_M}(f^*)$. For more details, see, e.g., [335].



Mikhail Ivanovich Grinchuk
Moscow University, since 1980s to 2013

In Grinchuk's method, the computation of a function of one type is reduced to the computation of a function of the dual type. The duality principle allows one to ignore the difference between types and to apply recursive reasoning. A trivial illustration of this approach is the proof of Theorem 1.1.

So, the task of constructing a parallel adder is to minimize the depth of functions

$$F_n(X,Y) = y_{n-1} \vee x_{n-1}(y_{n-2} \vee \ldots \vee x_2(y_1 \vee x_1 y_0)\ldots). \quad (8.2)$$

Theorem 8.2 ([114]). $\mathsf{D}_{\mathcal{B}_M}(F_n) \leqslant \log_2 n + \log_2 \log n + O(1)$.

▶ As in the proof of Theorem 2.6, consider a wider family of functions:

$$F_{r,2k-1}(X,Y) = x_{r+k-1} \cdot \ldots \cdot x_k(y_{k-1} \vee x_{k-1}(\ldots(y_1 \vee x_1 y_0)\ldots)),$$
$$F_{r,2k}(X,Y) = x_{r+k-1} \cdot \ldots \cdot x_k(y_{k-1} \vee x_{k-1}(\ldots(y_1 \vee x_1(y_0 \vee x_0))\ldots)).$$

(The second index in the function notation indicates the number of variables in the part of alternating operations.) By construction, $F_i = F_{0,2i-1}$. Denote $d(r,m) = \mathsf{D}_{\mathcal{B}_M}(F_{r,m})$.

Lemma 8.2.

$$d(r+s,m) \leqslant \max\{\lceil \log_2 r \rceil,\, d(s,m)\} + 1, \quad (8.3)$$
$$d(r,2s+m) \leqslant \max\{d(r,2s),\, d(s+1,m-1)\} + 1. \quad (8.4)$$

▷ The first bound is trivial. The second follows from a decomposition generalizing the identity $y_1 \vee x_1 y_0 = (y_1 \vee x_1)(y_1 \vee y_0)$.

Let X^k and Y^k denote (sub)sets of variables X and Y with indices counted upward from k. Put $t = \lceil m/2 \rceil$, and also $P = x_{r+s+t-1} \cdot \ldots \cdot x_{s+t}$ and $Q = y_{s+t-1} \vee \ldots \vee y_t$. Then we can write

$$F_{r,2s+m}(X,Y) = P \cdot F_{0,2s+m}(X,Y) = P \cdot F_{0,2s}(X^t,Y^t) \cdot (Q \vee F_{0,m}(X,Y)) = F_{r,2s}(X^t,Y^t) \cdot F^*_{s+1,m-1}(Y^{(m \bmod 2)-1}, X^{m \bmod 2}). \quad (8.5)$$

By the duality principle, functions f and f^* have the same depth over the basis $\mathcal{B}_M$, which immediately implies (8.4). □

For $h \geqslant 2$ and $0 \leqslant r \leqslant 2^h - 1$ define auxiliary quantities $\nu(h,r)$ by the rules

$$\nu(2,0) = \nu(2,1) = \nu(2,2) = 2, \qquad \nu(2,3) = 0,$$

and for $h > 2$ recursively as

$$\nu(h+1,r) = \begin{cases} \nu(h, r-2^h), & 2^h \leqslant r < 2^{h+1}, \\ \nu(h,r) + \nu(h, 1 + \nu(h,r)/2), & 0 \leqslant r < 2^h. \end{cases} \quad (8.6)$$

Division by 2 is always possible, since the function $\nu(h,r)$ takes only even values.

The quantity $\nu(h,r)$ has the meaning of a lower bound for the largest number m such that the function $F_{r,m}$ is implemented with depth h. This is formally proved by the following lemma.

Lemma 8.3. *For any $h \geqslant 2$ and $r < 2^h$, we have $d(r, \nu(h,r)) \leqslant h$.*

▷ For $h = 2$ the statement can be verified directly. Let us prove the induction step from h to $h+1$.

If $2^h \leqslant r < 2^{h+1}$, then by (8.3), (8.6) and the induction hypothesis,

$$d(r, \nu(h+1,r)) = d(r, \nu(h, r-2^h)) \leqslant \max\{h,\, d(r-2^h, \nu(h, r-2^h))\} + 1 = h+1.$$

Otherwise, if $r < 2^h$, then, applying(8.6), (8.4) and the induction hypothesis, we obtain

$$d(r, \nu(h+1,r)) = d(r, \nu(h,r) + \nu(h, 1+\nu(h,r)/2)) \leqslant$$
$$\max\{d(r, \nu(h,r)),\, d(1+\nu(h,r)/2, \nu(h, 1+\nu(h,r)/2))\} + 1 \leqslant h+1.$$

□

Lemma 8.4. $\nu(h,r) \geqslant \frac{2^h - r - 1}{h}$.

▷ For $h \leqslant 3$ the inequality is verified directly. Consider the induction step from h to $h+1$.

If $r \geqslant 2^h$, then

$$\nu(h+1,r) = \nu(h, r-2^h) \geqslant \frac{2^h - (r-2^h) - 1}{h} > \frac{2^{h+1} - r - 1}{h+1}.$$

If $r < 2^h$, then

$$\nu(h+1,r) = \nu(h,r) + \nu(h, 1+\nu(h,r)/2) \geqslant$$
$$\frac{2^h - 2}{h} + \left(1 - \frac{1}{2h}\right)\nu(h,r) \geqslant \frac{2^h - 2}{h} + \left(1 - \frac{1}{2h}\right)\frac{2^h - r - 1}{h}.$$

The obtained estimate $l_1(r)$ depends on r linearly with the coefficient $a = -\frac{2h-1}{2h^2}$. Therefore, in order to verify that $l_1(r)$ is at least $l_2(r) = \frac{2^{h+1}-r-1}{h+1}$ (a linear function of r with the coefficient $-\frac{1}{h+1} > a$) on the interval $[0, 2^h - 1]$, it suffices to compare the values of the functions at the right end of the interval, $r = 2^h - 1$. For $h \geqslant 3$, we have

$$l_1(2^h - 1) = \frac{2^h - 2}{h} \geqslant \frac{2^h}{h+1} = l_2(2^h - 1),$$

which completes the proof of the induction step. □

For $r = 0$, Lemma 8.4 yields the estimate $\nu(h,0) \geqslant (2^h - 1)/h$. Then, applying Lemma 8.3, we derive $d(0,n) \leqslant \log_2 n + \log_2 \log n + O(1)$. ■

We emphasize that the key point of the proof is formula (8.5), which is valid due to the duality of the basis functions.

Corollary 8.1 ([114]). $\mathsf{D}_{\mathcal{B}_0}(\Sigma_n) \leqslant \log_2 n + \log_2 \log n + O(1)$.

• The method of Theorem 8.2 is optimal up to an additive constant in the depth bound, since B. Commentz-Walter had previously proved [67] the bound $\mathsf{D}_{\mathcal{B}_M}(F_n) \geqslant \log_2 n + \log_2 \log n - O(1)$. It should not be surprising that the duality principle also plays a role in the proof of the lower bound. Together with J. Sattler [68] they also obtained a bound for the complete basis

$$\mathsf{D}_{\mathcal{B}_0}(F_n) \geqslant \log_2 n + (1 - o(1)) \log_2 \log \log n,$$

from which, as V. M. Khrapchenko [159] noted, taking into account $\mathsf{D}_{\mathcal{B}_0}(F_n) \leqslant \mathsf{D}_{\mathcal{B}_0}(\Sigma_n) + O(1)$ there follows a similar lower bound for the depth of addition $\mathsf{D}_{\mathcal{B}_0}(\Sigma_n)$.

Using Grinchuk's construction, A. Hermann [127] constructed an n-bit adder of linear complexity and depth $\log_2 n + \log_2 \log n + \log_2 \log \log n + O(1)$.

Multiplication of rectangular and square matrices $\boxed{d}$

One of the most important tools in the theory of fast matrix multiplication algorithms is the trilinear identity discovered by V. Ya. Pan [236], which allows, in particular, to reduce the multiplication of rectangular matrices to the multiplication of square ones.

Lemma 8.5 ([236]). *In a commutative ring R, for any permutation π of indices m, p, q,*

$$\operatorname{rk}^R MM_{m,p,q} = \operatorname{rk}^R MM_{\pi(m),\pi(p),\pi(q)}, \qquad \underline{\operatorname{rk}}^R MM_{m,p,q} = \underline{\operatorname{rk}}^R MM_{\pi(m),\pi(p),\pi(q)}.$$

Victor Yakovlevich Pan
City University of New York,
since 1988

▷ We will prove only the second equation. Assume that the operator $MM_{m,p,q}$ admits a (d, r)-representation of type (5.11):

$$u^d XY = \sum_{l=1}^{r} C_l(u) X_l(u) Y_l(u) \bmod u^{d+1}. \tag{8.7}$$

Transposing it[1)], we obtain

$$u^d Y^T X^T = \sum_{l=1}^{r} C_l^T(u) Y_l(u) X_l(u) \bmod u^{d+1}.$$

Hence,

$$\underline{\operatorname{rk}}\, MM_{q,p,m} \leqslant \underline{\operatorname{rk}}\, MM_{m,p,q}. \tag{8.8}$$

By scalar multiplication (8.7) by $m \times q$ matrix Z^T (or, equivalently, by tensor convolution[2)] with Z^T), we derive *Pan's trilinear identity* for (d, r)-representations:

$$u^d \sum_{1 \leqslant i \leqslant m,\ 1 \leqslant j \leqslant p,\ 1 \leqslant k \leqslant q} x_{ij} y_{jk} z_{ki} = \sum_{l=1}^{r} X_l(u) Y_l(u) Z_l(u) \bmod u^{d+1}, \tag{8.9}$$

[1] Here we use the commutativity of the ring.
[2] With a suitable indexing, e.g., $x_i^j y_j^k z_k^i$.

where $Z_l(u)$ are linear combinations of variables z_{ik}.

Substituting $x_{ij} = 1$ and $x_{i'j'} = 0$ into (8.9) for all $(i', j') \neq (i, j)$, we obtain a (d, r)-representation for an arbitrary sum $\sum_{k=1}^{q} y_{jk} z_{ik}$. All such representations can be combined in the formula

$$u^d Y Z^T = \sum_{l=1}^{r} C'_l(u) Y_l(u) Z_l(u) \bmod u^{d+1},$$

where $C'_l(u) \in R[u]^{p \times m}$. Thus,

$$\underline{\mathrm{rk}}\, MM_{m,p,q} \leqslant \underline{\mathrm{rk}}\, MM_{p,q,m}. \tag{8.10}$$

Together (8.8) and (8.10) imply the assertion of the lemma. □

Lemma 8.5 establishes duality between bilinear algorithms for multiplying matrices that are identical up to a permutation of linear dimensions. Applying Lemma 5.4, we immediately obtain

Corollary 8.2. *For a commutative rink R,*

$$\mathrm{rk}^R MM_{mpq} \leqslant (\mathrm{rk}^R MM_{m,p,q})^3, \qquad \underline{\mathrm{rk}}^R MM_{mpq} \leqslant (\underline{\mathrm{rk}}^R MM_{m,p,q})^3.$$

• Employing this technique, Italian mathematicians [33] derived from $\underline{\mathrm{rk}}\, MM_{2,3,2} \leqslant 10$ the inequality $\underline{\mathrm{rk}}\, MM_{12} \leqslant 1000$, which, due to the observation of D. Bini [32] (Theorem 5.4), entailed $\mathsf{C}_{\mathcal{A}}(MM_n) \preccurlyeq n^{\log_{12} 1000 + o(1)} \prec n^{2.78}$ — at that time this was the record upper bound for the complexity of matrix multiplication.

Of practical interest is A. V. Smirnov's bound $\mathrm{rk}\, MM_{3,3,6} \leqslant 40$ [308]. In practice, the algorithms of Theorem 5.3 based on estimates for ranks have priority over the algorithms of Theorem 5.4 based on estimates for border ranks. Thus, the matrix multiplication method with complexity of order $n^{3 \log_{54} 40} \prec n^{2.775}$ has application prospects almost at the level of Strassen's algorithm. In terms of practical application, the results of V. Ya. Pan, obtained by directly constructing economical trilinear decompositions, also deserve attention. In particular, in [237] he showed that $\mathrm{rk}\, MM_{44} \leqslant 36133$, whence $C_{\mathcal{A}}(MM_n) \preccurlyeq n^{\log_{44} 36133} \prec n^{2.774}$ (this is currently a record result for the complexity of matrix multiplication that does not employ bounds on the border ranks). For more details, see [239]. A practically significant speedup of Pan's algorithms (roughly in the spirit of Theorem 7.4) was obtained by T. Hadas and O. Schwartz [118].

Complexity of a rational function and its gradient $\boxed{d}$

An important role in the complexity theory of arithmetic circuits is played by the fact discovered by W. Baur and V. Strassen on the equivalence of the arithmetic complexity of a rational function (in particular, a polynomial) and its gradient [17]. Recall that the gradient of a function $f(x_1, \ldots, x_n)$ is defined as $\bigtriangledown f = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right)$. An elegant way of proving the Baur—Strassen result, based on the application of the transposition principle for linear circuits, was proposed by the brothers S. B. and I. B. Gashkov in [102]. First, we need the following generalization of Lemma 8.1.

Lemma 8.6 ([289]). *For any $m \times n$ matrix A over a ring R with unity*[3)]*,*

$$\mathsf{C}_{\mathcal{A}_L^R}(L_{A^T}) \leqslant \mathsf{C}_{\mathcal{A}_L^R}(L_A) + m - 1, \tag{8.11}$$

and this inequality is achieved on circuits (for L_A and L_{A^T}) with the same set of elements of scalar multiplication, not counting multiplications by -1.

▷ Here we prove a slightly weaker bound $\mathsf{C}_{\mathcal{A}_L^R}(L_{A^T}) \leqslant \mathsf{C}_{\mathcal{A}_L^R}(L_A) + m$. The proof is very similar to the proof of Lemma 8.1. Consider the (directed) graph of a circuit computing L_A and assign weights to the edges of the graph: weight 1 for the edges entering addition elements, weights 1 and -1 for the edges entering subtraction elements, and weight a for the edges entering elements of multiplication by a. The weighted graph contains complete information about the original circuit.

As an illustration, Fig. 8.1a shows a circuit that implements the transform $y_1 = x_1 - x_2 - ax_3$, $y_2 = x_2 - ax_3$. (The operation of multiplication by constant is denoted by $\bullet$, negative inputs of subtraction elements are marked with circles.) Fig. 8.1b shows the graph of the circuit.

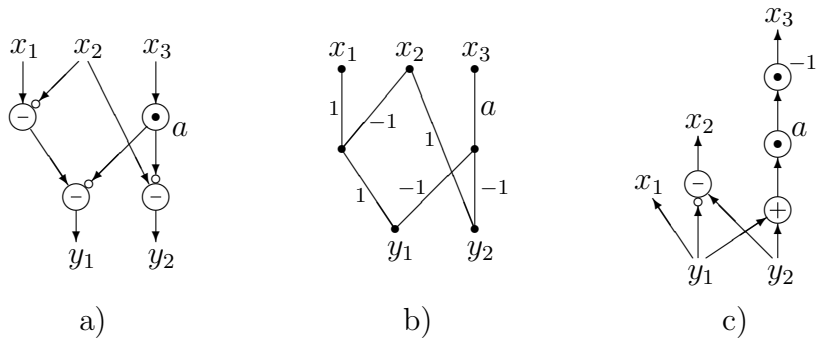


Figure 8.1: Initial circuit (a), its graph (b), and the transposed circuit (c)

The weight of an oriented path in a graph is defined as the product of weights of edges along the path. By construction, a matrix entry $A[i, j]$ is equal to the sum of the weights of all paths connecting the j-th input and the i-th output. In this sense, the graph implements the matrix A.

When reversing the orientation of the graph, we obtain a graph that computes the matrix A^T in the sense specified above. It remains to transform it into a circuit over the basis $\mathcal{A}_L$.

To do this, first we attach elements of scalar multiplication to all edges with weights different from ± 1[4)] (thereby, we restore the set of scalar multiplications of the original circuit). After this, only edges with weights ± 1 remain in the circuit. The remaining vertices of the graph should be replaced by trees of additive elements.

We will perform the procedure of "lifting" negative labels from inputs to outputs. If not all edges entering a node have weight -1, then summation at this node is

[3]Recall that L_A denotes the linear operator with a matrix A.

[4]More precisely, we replace an edge of weight a with an element of multiplication by a together with incoming and outgoing edges of weight 1.

implemented by a tree of binary additions and subtractions[5]. Otherwise, we invert the labels of the edges entering and leaving the node, with the exception of the edges leading to elements of scalar multiplication — in this case, inversion is applied to the edges leaving the specified elements of multiplication. The process is terminated at the outputs of the circuit.

If all edges connected to an output of the circuit have weights -1, then summation at this output is implemented by a tree of addition elements followed by a multiplication element by -1. Fig. 8.1c shows the resulting circuit for the considered example.

According to the argument from Lemma 8.1, the number of binary elements in the resulting circuit and the original one differs by $m-n$ (it is equal to the difference between the number of edges and the number of vertices in the graph, excluding inputs; the set of inputs changes during transposition). The set of elements of multiplication by nontrivial constants is the same in both circuits. In addition, at most n elements of multiplication by -1 are included (according to the number of outputs).

In fact, as can be verified, the procedure of lifting negative labels results in at least one output receiving an incoming edge of weight 1. Therefore, at most $n-1$ additional multiplications by -1 are required. □

- The bound of Lemma 8.6 is tight, witnessing by the transform

$$(x, y_1, \ldots, y_m) \to (x - y_1, \ldots, x - y_m).$$

Theorem 8.3 ([17]). *For an arbitrary rational function $f \in R(X)$, where R is a ring with unity, $\mathsf{C}_{\mathcal{A}_D^R}(f, \bigtriangledown f) \leqslant 4\mathsf{C}_{\mathcal{A}_D^R}(f)$.*

► Consider a minimal circuit S for the function $f(x_1, \ldots, x_n)$. Let us introduce new formal variables $dx_1, \ldots, dx_n$ — differentials of variables — and construct a circuit S' for the differential of the function f,

$$df = \frac{\partial f}{\partial x_1} dx_1 + \ldots + \frac{\partial f}{\partial x_n} dx_n.$$

Volker Strassen
Universität Zürich,
1968 to 1988

The circuit S' is constructed parallel to the circuit S inductively from inputs to outputs. A variable x_i corresponds to the differential dx_i (induction base). Now, if the next element of the circuit S performs the operation $u \circ v$, and the differentials du, dv have already been computed, then $d(u \circ v)$ is computed according to the rules:

$$d(u \pm v) = du \pm dv, \quad d(uv) = u \cdot dv + v \cdot du, \quad d(u/v) = (du - (u/v) \cdot dv)/v. \quad (8.12)$$

[5]In a degenerate case, when a node has a single entering edge of weight 1, an empty tree is inserted, i.e., the node together with the edge entering it are simply removed from the circuit.

In the case of scalar multiplication by a, we simply have $d(au) = a \cdot du$.

If we consider S' as a circuit on the inputs $dx_1, \ldots, dx_n$ over the basis $\mathcal{A}_D^{R(X)}$ (i.e., all possible functions from $R(X)$ are allowed as constants), then by construction its complexity does not exceed $3\mathsf{C}(S)$ due to (8.12).

Since the differential and gradient as linear operators over $R(X)$ turn into each other under transposition[6)], by Lemma 8.6 we obtain

$$\mathsf{C}_{\mathcal{A}_L^{R(X)}}(\bigtriangledown f) \leqslant \mathsf{C}_{\mathcal{A}_L^{R(X)}}(df) \leqslant 3\mathsf{C}(S),$$

where the circuit for the gradient employs exactly the same "constants" as the circuit S'. All the necessary constants are provided by the circuit S, see (8.12), therefore after connecting circuit S to the constructed circuit we obtain the required result: $\mathsf{C}_{\mathcal{A}_D^R}(f, \bigtriangledown f) \leqslant 4\mathsf{C}(S)$. ■

Method [17] allows to establish a connection between the complexity of matrix inversion and the computation of its determinant.

Corollary 8.3 ([17]). *Let $A = (a_{ij})$ be a non-singular $n \times n$ matrix. Then the complexities of computing the inverse matrix A^{-1} and the determinant* $\det A$ *as functions of the coefficients a_{ij} are related as*

$$\mathsf{C}_{\mathcal{A}_D^R}(A^{-1}) \leqslant 4\mathsf{C}_{\mathcal{A}_D^R}(\det A) + n^2.$$

▷ Let $A^{-1} = (b_{ij})$. Then, according to Cramer's rule,

$$b_{ij} = \frac{1}{\det A} \frac{\partial \det A}{\partial a_{ji}}.$$

Thus, by Theorem 8.3

$$\mathsf{C}(A^{-1}) \leqslant \mathsf{C}(\det A, \bigtriangledown \det A) + n^2 \leqslant 4\mathsf{C}(\det A) + n^2.$$

□

• In fact, a method for fast gradient computation was proposed earlier by S. Linnainmaa [188]. Independently, the result of Theorem 8.3 with a less precise complexity estimate was obtained in [160].

The fact that both problems – matrix inversion and determinant computation – has complexity at most of order of that of matrix multiplication was proved by Strassen in [313].

The author in [289] obtained (generally speaking) a stronger bound than in Theorem 8.3

$$\mathsf{C}_{\mathcal{A}_D^R}(f, \bigtriangledown f) \leqslant 3\mathsf{C}_{\mathcal{A}_D^R}(f) + n, \tag{8.13}$$

where n is the number of arguments of the function f. The proof is based on a more symmetric calculation of differentials of multiplicative operations:

$$d(uv) = (du/u + dv/v) \cdot (uv), \qquad d(u/v) = (du/u - dv/v) \cdot (u/v).$$

In this case, unlike the method [17], in the circuits constructed by the method [289], the set of divisor functions may differ from the similar set in the original circuit computing the function f.

[6)] The constant $1 \in R$ is fed to the only input of the transposed circuit.

Inequality (8.13) is also valid in the case of different weights of additive and multiplicative operations [289].

Theorem 8.3 also holds over the basis without division $\mathcal{A}^R$. Naturally, we are talking about the complexity of polynomials. Bound (8.13) is not applicable in this situation.

E. Kaltofen and M. Singer [145] noted that the complexity bound of Theorem 8.3 is achieved while preserving the order of depth of the original circuit. The same is true for bound (8.13), see [289].

Chapter 9

Probabilistic method $\boxed{P}$

This chapter will discuss methods of synthesis based on probabilistic arguments. It is usually proved that among a set of circuits of a certain type there is one that implements the desired function.

Monotone formulae for symmetric threshold functions $\boxed{P}$

Recall that T_n^k denotes the symmetric threshold-k function of n variables: $T_n^k(x_1,\ldots,x_n) = (x_1 + \ldots + x_n \geqslant k)$. In the case $k = 1$, $\Phi_{\mathcal{B}_M}(T_n^1) = n$ trivially holds. A function with threshold 2 is easily computed by the bisection method. Let $X = (X_1, X_2)$, $|X_i| = n_i$. The following formula is valid [172]:

$$T^2_{n_1+n_2}(X) = T^1_{n_1}(X_1)\cdot T^1_{n_2}(X_2) \vee T^2_{n_1}(X_1) \vee T^2_{n_2}(X_2). \tag{9.1}$$

Theorem 9.1 ([177]). $\Phi_{\mathcal{B}_M}(T_n^2) \leqslant n\lfloor\log_2 n\rfloor + 2(n - 2^{\lfloor\log_2 n\rfloor}) \sim n\log_2 n$.

▶ Apply (9.1) recursively, splitting the set of variables in half. ■

• R. E. Krichevskii [177] proved the lower bound $\Phi_{\mathcal{B}_M}(T_n^2) \succcurlyeq n\log n$ along with the upper bound of Theorem 9.1. In fact, as was later established by J. Radhakrishnan [255] and S. A. Lozhkin [194] (modulo Krichevskii's structural result $\Phi_{\mathcal{B}_0}(T_n^2) = \Phi_{\mathcal{B}_M}(T_n^2)$ [177]), the bound of Theorem 9.1 is tight even in the class of formulae over the basis $\mathcal{B}_0$.

For constant $k \geqslant 3$, constructive methods of synthesis are not so accurate, but it is possible to prove the existence of comparatively simple circuits. The following result is due to L. S. Khasin [153].

Theorem 9.2 ([153]). *For constant $k \geqslant 2$, we have $\Phi_{\mathcal{B}_M}(T_n^k) \preccurlyeq n\log n$.*

▶ Let $k \mid n$ and $(X_i^1,\ldots,X_i^k)$ be a random partition of the set of variables X into

equal-sized subsets[1] X_i^j, $i \in \mathbb{N}$. Consider a random function

$$f(X) = \bigvee_{i=1}^{t} H_i, \qquad H_i = T^1_{n/k}(X_i^1) \cdot T^1_{n/k}(X_i^2) \cdot \ldots \cdot T^1_{n/k}(X_i^k). \tag{9.2}$$

The function does not have implicants of length $< k$, so[2] $f(X) \geqslant T_n^k(X)$. The probability that H_i contains some implicant $I = x_{j_1} \cdot \ldots \cdot x_{j_k}$ is estimated as

$$\mathbf{P}(H_i \geqslant I) = k!\,\mathbf{P}(x_{j_1} \in X_i^1, \ldots, x_{j_k} \in X_i^k) = k!\,\frac{n/k}{n} \cdot \frac{n/k}{n-1} \cdot \ldots \cdot \frac{n/k}{n-k+1} \geqslant \frac{k!}{k^k} > \frac{1}{e^k}.$$

Then, the probability that f does not contain an implicant I is

$$\mathbf{P}(f \not\geqslant I) = \prod_{i=1}^{t} \mathbf{P}(H_i \not\geqslant I) \leqslant (1 - e^{-k})^t.$$

For $t \approx e^k k \ln n$ due to the inequality $(1 - 1/x)^x < 1/e$ valid for $x > 1$, we have

$$\mathbf{P}(f \neq T_n^k) = \mathbf{P}(\exists_I f \not\geqslant I) \leqslant C_n^k\,\mathbf{P}(f \not\geqslant I) < C_n^k/n^k \leqslant 1.$$

As a consequence, $\mathbf{P}(f = T_n^k) > 0$. Therefore, some formula of form (9.2) implements the function T_n^k and has complexity $tn \asymp n \log n$. ■

• The bound of Theorem 9.2 is tight in order, $\Phi_{\mathcal{B}_M}(T_n^k) \asymp n \log n$, for example, in view of the simple lower bound $\Phi_{\mathcal{B}_M}(T_n^k) \geqslant \Phi_{\mathcal{B}_M}(T^2_{n-k+2})$ [153].

For thresholds functions with small thresholds, an upper bound $\Phi_{\mathcal{B}_M}(T_n^k) \preccurlyeq n \log n (k^2/2)^{\log^* n}$ was proved constructively by M. Kleiman and N. Pippenger [161].

Monotone formulae for the majority function $\boxed{P}\boxed{\varepsilon}$

From the proof of Theorem 9.2 it is evident that with the growth of k the complexity of the method grows exponentially, and for the implementation of threshold functions with large thresholds, in particular, for the majority function $\mathrm{maj}_n = T_n^{n/2}$, the method does not fit. An elegant solution to the problem was found by L. Valiant [326]. Let $\alpha = \frac{3-\sqrt{5}}{2} \approx 0.38$.

Theorem 9.3 ([326]). $\mathsf{D}_{\mathcal{B}_M}(\mathrm{maj}_n) \lesssim 2(1 + \log_{4\alpha} 2) \log_2 n < 5.28 \log_2 n$.

► In fact, we will prove a slightly weaker bound $\mathsf{D}_{\mathcal{B}_M}(\mathrm{maj}_n) \lesssim 2(1 + \log_\beta 2) \log_2 n$ for an arbitrary constant $\beta \in (1,\, 4\alpha)$.

[1] More formally, we consider the uniform distribution on the set of all permutations $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ and choose

$$X^j = (x_{\pi((j-1)n/k+1)},\, x_{\pi((j-1)n/k+2)},\, \ldots,\, x_{\pi(jn/k)}), \quad 1 \leqslant j \leqslant k.$$

[2] $f(X) \geqslant g(X)$ means that $f(\alpha) \geqslant g(\alpha)$ in every point α.

Leslie Gabriel Valiant
Harvard University, since 1982

For simplicity, we assume that n is odd, $n = 2m - 1$. We define a series of probability distributions $\Delta_0, \Delta_1, \ldots$ on special sets of monotone formulae. The distribution Δ_k covers formulae of depth $2k$. The distribution Δ_0 includes literals x_i and the constant 0, while for a formula $G \in \Delta_0$,

$$\mathbf{P}(G \equiv 0) = 1 - 2\alpha, \qquad \mathbf{P}(G \equiv x_i) = \frac{2\alpha}{n}, \quad i = 1, \ldots, n.$$

For $k \geqslant 1$, the distribution Δ_k includes all possible formulae of the form $(G_1 \vee G_2)(G_3 \vee G_4)$, where the subformulas G_i are chosen from Δ_{k-1} independently.

Consider arbitrary sets of variables $X_0 \in \mathrm{maj}_n^{-1}(0)$, $X_1 \in \mathrm{maj}_n^{-1}(1)$ with weights $\|X_0\| = m - 1$ and $\|X_1\| = m$. Let us introduce notations for the probability of formula error:

$$p_k = \mathbf{P}(G(X_0) = 1 \mid G \in \Delta_k), \qquad q_k = \mathbf{P}(G(X_1) = 0 \mid G \in \Delta_k).$$

By construction,

$$p_{k+1} = (1 - (1 - p_k)^2)^2, \qquad q_{k+1} = 1 - (1 - q_k^2)^2. \tag{9.3}$$

It is easy to verify that the sequences $\{p_k\}$ and $\{q_k\}$ are monotonically decreasing on the intervals $(0,\, \alpha)$ and $(0,\, 1 - \alpha)$, respectively (the ends of these intervals are the roots of the equations $x = (1 - (1 - x)^2)^2$ and $x = 1 - (1 - x^2)^2$).

Lemma 9.1. *For some constant $\gamma > 0$ and some $t \leqslant \log_\beta n$,*

$$p_t < \alpha - \gamma, \qquad q_t < 1 - \alpha - \gamma.$$

▷ At $t = 0$ the error probabilities are equal to[3)]

$$p_0 = \mathbf{P}(G \equiv x_i \mid x_{0,i} = 1,\, G \in \Delta_0) = \alpha - \frac{\alpha}{n},$$
$$q_0 = \mathbf{P}(G \equiv 0 \mid G \in \Delta_0) + \mathbf{P}(G \equiv x_i \mid x_{1,i} = 0,\, G \in \Delta_0) = 1 - \alpha - \frac{\alpha}{n}.$$

Assuming $p_k = \alpha - \varepsilon_k$, and taking into account $(1-\alpha)^2 = \alpha$, from (9.3) we obtain

$$p_{k+1} = (1 - (1 - \alpha + \varepsilon_k)^2)^2 = \\ \alpha - 4\alpha\varepsilon_k + \big((2(1-\alpha) + \varepsilon_k)^2 - 2(1-\alpha)\big)\,\varepsilon_k^2 \leqslant \alpha - \beta\varepsilon_k \tag{9.4}$$

provided that $\varepsilon_k \leqslant \gamma_1$. As a consequence, for some $t_1 = \log_\beta n - O(1)$ we have $\varepsilon_{t_1} \geqslant \beta^{t_1}\varepsilon_0 > \gamma_1$.

Similarly, denoting $q_k = 1 - \alpha - \delta_k$, we obtain

$$q_{k+1} = 1 - (1 - (1 - \alpha - \delta_k)^2)^2 = \\ 1 - \alpha - 4\alpha\delta_k - \big((2(1-\alpha) - \delta_k)^2 - 2(1-\alpha)\big)\,\delta_k^2 \leqslant 1 - \alpha - \beta\delta_k \tag{9.5}$$

[3)] Here $x_{0,i}$ and $x_{1,i}$ denote the components of the vectors X_0 and X_1.

provided that $\delta_k \leqslant \gamma_2$. Consequently, for some $t_2 = \log_\beta n - O(1)$, we have $\delta_{t_2} \geqslant \beta^{t_2}\varepsilon_0 > \gamma_2$.

It remains to choose $t = \max\{t_1, t_2\}$ and $\gamma = \min\{\gamma_1, \gamma_2\}$. □

Lemma 9.2. *Let $p_t < \alpha - \gamma$ and $q_t < 1 - \alpha - \gamma$ for a constant $\gamma > 0$. Then for some $u = \log_2 n + O(1)$ we have $p_{t+u}, q_{t+u} < 1/2^{n+1}$.*

▷ *I.* First, we show that $p_{t+u'}, q_{t+u'} \leqslant 1/8$ for a suitable $u' = O(1)$. We can assume that γ is sufficiently small, say, $\gamma \leqslant 0.3$. From (9.3) it follows

$$p_{k+1} = p_k^2(2 - p_k)^2, \qquad q_{k+1} = q_k^2(2 - q_k^2). \tag{9.6}$$

It is easy to check that the functions $p(x) = x - x^2(2-x)^2$ and $q(x) = x - x^2(2-x^2)$ on the intervals $I_p = [1/8,\, \alpha - \gamma]$ and, correspondingly, $I_q = [1/8,\, 1 - \alpha - \gamma]$ are convex upwards, and therefore take minimum values at the ends (these values are positive). Hence, there exists $\tau > 0$ such that $\tau \leqslant \min_{x \in I_p} p(x), \min_{x \in I_q} q(x)$. Then $p_{k+1} \leqslant p_k - \tau$ as soon as $p_k \in I_p$, and $q_{k+1} \leqslant q_k - \tau$ as soon as $q_k \in I_q$. Therefore, we can choose $u' = (1 - \alpha - \gamma - 1/8)/\tau$.

II. From (9.6) it is clear that $p_{k+1} \leqslant 4p_k^2$ and $q_{k+1} \leqslant 2q_k^2$. Therefore, starting from $p_{t+u'}, q_{t+u'} \leqslant 1/8$, we derive

$$p_{t+u'+i} \leqslant 2^{-(2^i+2)}, \qquad q_{t+u'+i} \leqslant 2^{-(2^{i+1}+1)}.$$

Finally, putting $i = \lceil \log_2 n \rceil$ we obtain the required estimates. □

Combining Lemmas 9.1 and 9.2, we find that with probability $< 1/2^{n+1}$ a formula from Δ_{t+u} incorrectly computes the function maj_n on the input X_0, and the same for the input X_1. As a consequence, with probability $> 1/2$ a formula from Δ_{t+u} (having depth $2(t+u)$) implements the function maj_n. ■

Corollary 9.1. $\Phi_{\mathcal{B}_M}(\mathrm{maj}_n) \prec n^{5.28}$.

• To prove exactly what Theorem 9.3 states, a strengthening of Lemma 9.1 is required. For example, it can be shown that for $t \leqslant \log_{4\alpha}(n/8)$,

$$p_t < \alpha - \frac{(4\alpha)^t}{4n}, \qquad q_t < 1 - \alpha - \frac{(4\alpha)^t}{3n}. \tag{9.7}$$

▷ Indeed, from (9.4) in view of $\varepsilon_k \leqslant \alpha$ we deduce

$$4\alpha\varepsilon_k \geqslant \varepsilon_{k+1} \geqslant 4\alpha\varepsilon_k - \big((2-\alpha)^2 - 2(1-\alpha)\big)\,\varepsilon_k^2 > 4\alpha(1-\varepsilon_k)\varepsilon_k.$$

Hence,

$$\varepsilon_t \geqslant 4\alpha(1-\varepsilon_{t-1})\varepsilon_{t-1} \geqslant (4\alpha)^t\varepsilon_0 \prod_{i=0}^{t-1}(1-\varepsilon_i) \geqslant (4\alpha)^t\varepsilon_0 \prod_{i=0}^{t-1}(1-(4\alpha)^i\varepsilon_0). \tag{9.8}$$

Employing the inequality $\ln(1-x) \geqslant -2x$ valid for $0 \leqslant x \leqslant 1/2$, and taking into account $\varepsilon_0 = \alpha/n$ and $(4\alpha)^t \leqslant n/8$, the product in (9.8) can be bounded as

$$\prod_{i=0}^{t-1}(1-(4\alpha)^i\varepsilon_0) \geqslant e^{-2\varepsilon_0(1+(4\alpha)+\ldots+(4\alpha)^{t-1})} \geqslant e^{-2\varepsilon_0(4\alpha)^t/(4\alpha-1)} \geqslant e^{-\alpha/(16\alpha-4)}.$$

This yields the first inequality in (9.7): $\varepsilon_t \geqslant (4\alpha)^t \varepsilon_0 e^{-\alpha/(16\alpha-4)} > (4\alpha)^t/(4n)$.

Similarly, from (9.5) we obtain

$$\delta_{k+1} \leqslant 4\alpha\delta_k + \left((2(1-\alpha))^2 - 2(1-\alpha)\right)\delta_k^2 < 4\alpha(1+\delta_k/4)\delta_k, \tag{9.9}$$

and under the additional assumption $\delta_k \leqslant \alpha/4$ — also

$$\delta_{k+1} \geqslant 4\alpha\delta_k + \left((2(1-\alpha)-\alpha/4)^2 - 2(1-\alpha)\right)\delta_k^2 > 4\alpha\delta_k.$$

Thus, the second inequality in (9.7) holds: $\delta_t \geqslant (4\alpha)^t\delta_0 > (4\alpha)^t/(3n)$, if only $\delta_{t-1} \leqslant \alpha/4$.

Let us prove by induction that $\delta_k < 2(4\alpha)^k\delta_0$ for $k \leqslant t$. As a corollary, we obtain the required inequality $\delta_t \leqslant \alpha/4$. For $k = 0$ there is nothing to prove. Let us verify the induction step from $k-1$ to k. Indeed, according to (9.9), the induction hypothesis, the inequalities $1 + x \leqslant e^x$ and $(4\alpha)^k \leqslant n/8$,

$$\delta_k \leqslant 4\alpha(1+\delta_{k-1}/4)\delta_{k-1} \leqslant (4\alpha)^k\delta_0 \prod_{i=0}^{k-1}(1+\delta_i/4) \leqslant (4\alpha)^k\delta_0 \prod_{i=0}^{k-1}(1+(4\alpha)^i\delta_0/2) \leqslant$$
$$(4\alpha)^k\delta_0 e^{(1+4\alpha+\ldots+(4\alpha)^{k-1})\delta_0/2} \leqslant (4\alpha)^k\delta_0 e^{\delta_0(4\alpha)^k/(8\alpha-2)} \leqslant (4\alpha)^k\delta_0 e^{\alpha/(64\alpha-16)} < 2(4\alpha)^k\delta_0.$$

□

R. Boppana [41] proved the bound $\Phi_{\mathcal{B}_M}(T_n^k) \preccurlyeq k^{4.28} n \log n$ for an arbitrary threshold function by modifying Valiant's method (this is better than in the method of Theorem 9.2). He also established that the generating formula $(G_1 \vee G_2)(G_3 \vee G_4)$ used in the proof is optimal among read-once formulae; this statement was proved in a more rigorous form by M. Dubiner and U. Zwick [80]. For the ternary monotone basis $\{\mathrm{maj}_3\}$, A. Gupta and S. Mahajan [116] obtained by a similar method the bound $\Phi_{\{\mathrm{maj}_3\}}(\mathrm{maj}_n) \preccurlyeq n^{4.3}$.

Valiant's method is provably efficient in constructing approximations of threshold functions. Let Ψ_n^k denote the class of monotone functions that take value 0 on inputs of weight $\leqslant k$ and value 1 on inputs of weight $\geqslant n-k$. By the method of Theorem 9.3 for any constant $\alpha \in (0,\, 1/2)$ formula of complexity $O(n^2)$ implementing some function from $\Psi_n^{\alpha n}$ may be constructed. Its complexity is optimal in order due to the classical lower bound $\Phi_{\mathcal{B}_M}(f) \geqslant (k+1)^2$ proved by E. Moore and C. Shannon [222] for an arbitrary function $f \in \Psi_n^k$.

Note that the known lower bounds for the complexity of the majority function are $\Phi_{\mathcal{B}_M}(\mathrm{maj}_n) \succcurlyeq n^2$ [222] (which also follows from $\Phi_{\mathcal{B}_0}(\mathrm{maj}_n) \succcurlyeq n^2$ [155]) and $\Phi_{\{\mathrm{maj}_3\}}(\mathrm{maj}_n) \succcurlyeq n^{\log_2 3}$ [263].

Several papers have attempted to derandomize the method. For example, S. A. Lozhkin and A. A. Semenov [198] pointed out the possibility of constructing a formula of complexity $L \asymp k^{6.28} n \log n$ for the function T_n^k in time of order $L \cdot \log n$.

In complete binary bases, the best known upper bounds [306]

$$\Phi_{\mathcal{B}_0}(\mathrm{maj}_n) \prec n^{3.64}, \quad \mathsf{D}_{\mathcal{B}_0}(\mathrm{maj}_n) \lesssim 3.81 \log_2 n, \quad \Phi_{\mathcal{B}_2}(\mathrm{maj}_n) \prec n^{2.77}, \quad \mathsf{D}_{\mathcal{B}_2}(\mathrm{maj}_n) \lesssim 2.91 \log_2 n$$

were obtained by a method that includes the considered probabilistic procedure as an ingredient.

Parallel circuits for the logical permanent $\boxed{P}\boxed{A}$

The *logical permanent* of order n is a function of n^2 boolean variables:

$$PERM_n(X) = \bigvee_{\pi} x_{1,\pi(1)} \cdot x_{2,\pi(2)} \cdot \ldots x_{n,\pi(n)}, \tag{9.10}$$

where the disjunction is taken over all permutations $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$. The permanent expresses the existence of a perfect matching in a bipartite graph with parts of n vertices each.

In essence, the permanent is a monotone version of the determinant (of the matrix of variables X). It is not surprising that the method for computing the permanent presented below exploits a reduction to computing the determinant. Let us take as a starting point a simple circuit for the determinant proposed by A. L. Chistov [64] (see also [34]), in a modification due to N. Pippenger [250]. Let $+_n$ denote the operation of summation of n arguments. Consider the basis $\mathcal{A}_n = \{*, -, +_{n+1}\}$, in which the complexity of the operation $+_{n+1}$ is set equal to n.

Theorem 9.4 ([64, 34, 250]). *The determinant of order n over a field F can be computed by a circuit over the basis $\mathcal{A}_n^F$ of complexity $O(n^4 \log n)$ and depth $O(\log n)$.*

As a consequence, we obtain a circuit over the standard arithmetic basis $\mathcal{A}^F$ of complexity $O(n^4 \log n)$ and depth $O(\log^2 n)$.

▶ To compute the determinant efficiently, it is sufficient to select a suitable algebraic expression. Let A_k be the principal $k \times k$ submatrix[4)] of the matrix A. According to Cramer's rule, if the matrix A_k is invertible, then

$$\det A_{k-1} / \det A_k = (A_k^{-1})[k,k],$$

whence for a matrix $A \in F^{n\times n}$ we obtain

$$1/\det A = \prod_{k=1}^{n} (A_k^{-1})[k,k]. \tag{9.11}$$

Note that the determinant $\det A$ is, up to a factor of $(-1)^n$, the coefficient at x^n of the determinant of the matrix polynomial[5)] $P(x) = I_n - xA$. From (9.11) and the identity for formal power series $(1-x)^{-1} = \sum_{i=0}^{\infty} x^i$ together with its matrix version $(I_k - xA_k)^{-1} = \sum_{i=0}^{\infty} (xA_k)^i$, we derive the formula [250]

$$\det P(x) = \left(\prod_{k=1}^{n} (P(x)_k^{-1})[k,k] \right)^{-1} = \prod_{k=1}^{n} \left(1 + \sum_{i=1}^{\infty} (A_k)^i [k,k] x^i \right)^{-1} = 1 + \sum_{j=1}^{\infty} \left(1 - \prod_{k=1}^{n} \left(1 + \sum_{i=1}^{\infty} (A_k)^i [k,k] x^i \right) \right)^j . \tag{9.12}$$

Since we are interested in the coefficient at x^n, the sums on the right-hand side (9.12) can be restricted to the first n terms.

The computations are performed according to (9.12). Let us first note that in the basis $\mathcal{A}_n$ the standard multiplication of $n \times n$ matrices may be performed with complexity $O(n^3)$ and depth 2, matrix-vector multiplication — with complexity $O(n^2)$ and depth 2, and multiplication of polynomials modulo x^{n+1} — with complexity $O(n^2)$ and depth 2.

Let $v_k = (0, \ldots, 0, 1)$ be a vector of length k. The result of multiplying v_k by a $k \times k$ matrix B is the bottom row of the matrix.

[4] The principal is the upper left corner submatrix.

[5] $\det P(x)$ is the reversed characteristic polynomial of the matrix A.

1) For any k, we compute $v_k(A_k)^i$ for all $i \leqslant n$.

To do this, we sequentially compute $(A_k)^2, (A_k)^4, \ldots, (A_k)^{2^{\lfloor \log_2 n \rfloor}}$. In parallel, as the factors are ready, for $i = 2^h + j$ we calculate the rows $v_k(A_k)^i = (v_k(A_k)^j)(A_k)^{2^h}$. As a result, in particular, all the entries $(A_k)^i[k,k]$ are computed.

The circuit has (in total over all k) complexity $O(n^4 \log n)$ and depth $O(\log n)$.

2) The internal product of n polynomials modulo x^{n+1} in (9.12) may be computed with complexity $O(n^3)$ and depth $O(\log n)$.

3) Finally, a sum of the form $\sum_{j=0}^{n} p^j(x)$ is computed as $\prod_{j=0}^{\log_2 n} \left(1 + p^{2^j}(x)\right)$ modulo x^{n+1}. These calculations reduce to $2\log_2 n$ polynomial multiplications and are therefore implemented by a circuit of complexity $O(n^2 \log n)$ and depth $O(\log n)$. ■

The connection between the permanent $PERM_n(X)$ and the determinant may be established by a simple observation [83]. Define the entries of an $n \times n$ matrix $T(Y)$ as $T[i,j] = x_{i,j} y_{i,j}$, where $y_{i,j}$ are formal variables. (This matrix is called the *Tutte matrix*.) Then

$$PERM_n(X) = 1 \quad \Longleftrightarrow \quad \det T(Y) \not\equiv 0. \tag{9.13}$$

L. Lovázs [189] observed that this criterion allows one to establish the existence of simple parallel circuits for the permanent, since the equality of the determinant to zero can be verified by its values in a random set of points. A convenient tool for justifying the correctness of the method is a lemma due to J. Schwartz [281], which we will prove in a weakened form.

Lemma 9.3 ([281]). *Any multilinear polynomial $f \in \mathbb{Z}[x_1, \ldots, x_n]$ that is not identically zero has at most nm^{n-1} roots in the set $[\![m]\!]^n$.*

▷ Proof by induction. The case $n = 1$ is trivial. Let us prove the induction step from $n-1$ to n. Without loss of generality, let f essentially depend on the variable x_1. Write $f(X) = x_1 q(x_2, \ldots, x_n) + r(x_2, \ldots, x_n)$, where $q \not\equiv 0$. By the induction hypothesis, the polynomial q takes zero values on at most $(n-1)m^{n-2}$ inputs from $[\![m]\!]^{n-1}$. For any other input, there is at most one value x_1 that turns the polynomial f into zero. The total number of roots of the polynomial f in $[\![m]\!]^n$, therefore, does not exceed $(n-1)m^{n-1} + m^{n-1}$. □

Probably the first to point out the existence of parallel circuits of polynomial complexity for the permanent were A. Borodin, J. von zur Gathen and J. Hopcroft [42]. We present a more elementary proof.

Theorem 9.5. *The function $PERM_n$ is implemented by a boolean circuit over a finite complete basis with complexity $O(n^7 \log^2 n)$ and depth $O(\log^2 n)$.*

► For different values of inputs X, there are less than 2^{n^2} different Tutte matrices $T(Y)$. Set $m = n^2 2^{n^2}$. As follows from Lemma 9.3, there exists an integer matrix $Q \in [\![m]\!]^{n \times n}$ such that $\det T(Q) \neq 0$ if $\det T(Y) \not\equiv 0$.

Thus, $PERM_n(X) = (\det T(Q) \neq 0)$. To reduce the complexity of the computations, we use modular arithmetic. Note that $|\det T(Q)| < n!m^n < n^{3n}2^{n^3}$ for any Tutte matrix T. According to the asymptotic distribution of prime numbers (see, e.g., [264]), the product of the first s primes $p_1 \cdot \ldots \cdot p_s$ exceeds $2n^{3n}2^{n^3}$ for some $s \asymp n^3/\log n$. Therefore, $\det T(Q) = 0$ is equivalent to $\det T(Q) \equiv 0 \mod p_i$ for all $i = 1, \ldots, s$.

In the standard way, multiplication in the field $\mathbb{F}_p$ is implemented with complexity $O(\log^2 p)$ and depth $O(\log\log p)$ as $(ab \bmod p) = ab - p\lfloor abq \rfloor$, where q is a suitable approximation of the number $1/p$. The sum of n terms in the field $\mathbb{F}_p$ may be computed with complexity $O(n\log p + \log^2 p)$ and depth $O(\log(n\log p))$, say, by the compressor method followed by reduction modulo p.

Therefore, by Theorem 9.4 the value $r_i = \det T(Q) \bmod p_i$ is determined by a circuit of complexity $O(n^4\log^3 n)$ and depth $O(\log^2 n)$ for any $i \leqslant s$. The final check of all r_i being equal to zero is trivially implemented with complexity $O(s\log n)$ and depth $O(\log n)$. ■

• A careful analysis of the method of Theorem 9.4 yields a circuit of depth $O(\log^2 n)$ and complexity of order $n^{\omega+1+o(1)}$ over the basis $\mathcal{A}^F$, where $\omega < 2.38$ is the exponent of matrix multiplication, see [34]. The advantage of the method is the absence of division operations and weak requirements on the coefficient ring (instead of a field, a commutative ring with unity can be taken). In a basis with division, the complexity of computing the determinant of matrices over number fields (while preserving the order of depth) can be reduced to $O(n^{2.78})$ by the method of Z. Galil and V. Pan [97], if we apply current bounds on the complexity of rectangular matrix multiplication [8]. Recall that when depth restrictions are lifted, the determinant can be computed with complexity $n^{\omega+o(1)}$ [313].

The problem of implementing the determinant and the permanent with depth $o(\log^2 n)$ is still open. Explicit constructions of circuits for the permanent of complexity $n^{O(1)}$ and depth $\log^{O(1)} n$ are also unknown. Below (Theorem 11.4) we will show an alternative approach to computing a permanent by circuits without depth restrictions.

Complexity of linear boolean operators with dense matrices $\boxed{P}\boxed{/_2}$

For any boolean matrix A, $\mathsf{L}(A) \leqslant |A|$ trivially holds. In particular, an $n \times n$ matrix with a small number of ones, $|A| \asymp n$, has linear complexity $\mathsf{L}(A) \asymp n$. Less trivial is the question of the complexity of matrices with a small number of zeros. Is it true that $\mathsf{L}(A) \preccurlyeq |\overline{A}|$? The method of Russian mathematicians [179] actually gives an affirmative answer to this question.

Theorem 9.6. *For any boolean $n \times n$ matrix A of weight $n^2 - qn$, where $1 \leqslant q \leqslant n\log^{-4} n$, we have $\mathsf{L}(A) \preccurlyeq qn$.*

▶ The zeros split each row of the matrix into *intervals* — by interval we mean a substring of ones in successive columns. By interval sum we mean the sum of variables with successive indices $x_i + x_{i+1} + \ldots + x_j$. We will need an auxiliary lemma on the complexity of computing interval sums, which was proved by N. Alon and B. Schieber [11].

Lemma 9.4 ([11])**.** *Any interval sum on a set of n variables can be written as $u + v$, where all necessary sums u, v are computed by an additive circuit of complexity $n\log n$.*

▷ Divide the set of variables in half, with lower and higher indices. With complexity $O(n)$ compute prefix sums of variables with higher indices and suffix sums of variables with lower indices. Any interval sum with terms from both halves can be represented as the sum of some suffix and prefix from what has already been computed. To implement sums that lie entirely within each of the halves, we use recursion. The complexity $T(n)$ of the circuit satisfies $T(n) \leqslant 2T(n/2) + O(n)$, hence, $T(n) \leqslant n \log n$. □

With a suitable permutation of the rows, the matrix A can be represented as $A = \begin{bmatrix} A_0 \\ A_1 \end{bmatrix}$, where each row of the submatrix A_0 contains $\leqslant q \log n$ zeros, and each row of the submatrix A_1 contains more than $q \log n$ zeros. By assumption, the matrix A_1 has size $O(n/\log n) \times n$.

The rows of the matrix A_1^T split into $O(qn)$ intervals. Therefore, by Lemma 9.4, $\mathsf{L}(A_1^T) \preccurlyeq qn$, consequently, by the transposition principle (Lemma 8.1), $\mathsf{L}(A_1) \preccurlyeq qn$.

Consider the matrix A_0. The rows of the matrix A_0 also split into $O(qn)$ intervals. We divide the matrix into vertical strips of width $l = \log n$. The intervals that lie entirely inside one strip are called *simple*, the rest are *composite*.

I. Let us show that any composite interval sum can be written as $t+u+v+w$, where all intermediate sums t, u, v, w are computed by an additive circuit of complexity $(n/l)\log(n/l) + O(n)$.

To do this, in each strip we compute prefix and suffix sums with a total complexity of $O(n)$, including the sums of all variables of the strip (group sums). We feed the group sums (there are n/l) to the inputs of the circuit from Lemma 9.4: so we obtain the components u, v for the interval group sums. The complexity of the circuit is $(n/l)\log(n/l)$. It remains to note that an arbitrary composite interval sum, different from a group sum, inevitably crosses several strips and is therefore represented as the sum of a suffix t, a prefix w, and an interval group sum $u + v$.

As a consequence, all composite interval sums for the matrix A_0 may be computed with complexity $(n/l)\log(n/l) + O(qn)$. If we denote by s the total number of strips with simple intervals over all rows of the matrix A_0, then we have $\mathsf{L}(A_0) \leqslant (n/l)\log(n/l) + O(qn) + sl$.

II. The central point of the proof is the following observation: there is always a permutation of the columns of the matrix A_0 for which the number s is sufficiently small. We argue by the probabilistic method.

Let a row contain r zeros. The number of permutations of the row elements for which a certain strip contains a simple interval is roughly estimated from above as $C_{l+2}^2 C_n^{r-2}$ (the first factor estimates the number of ways to specify the ends of the interval, the second — the number of ways to place the remaining $r-2$ zeros).

Then the mathematical expectation of the number of strips with simple intervals in one row by order does not exceed

$$(n/l)\frac{C_{l+2}^2 C_n^{r-2}}{C_n^r} \preccurlyeq \frac{nlr(r-1)}{(n-r)(n-r+1)} \preccurlyeq \frac{q^2 \log^3 n}{n}$$

taking into account $l = \log n$ and $r \leqslant q \log n$. Consequently, $\mathbf{E}[s] \preccurlyeq q^2 \log^3 n$. Thus,

with nonzero probability $s \preccurlyeq q^2 \log^3 n$. Finally, we obtain

$$\mathsf{L}(A) \leqslant \mathsf{L}(A_0) + \mathsf{L}(A_1) \preccurlyeq qn + q^2 \log^4 n \preccurlyeq qn.$$

■

In particular, in the most interesting case $q = O(1)$ the following holds (this is the main result of the paper [179]):

Corollary 9.2 ([179]). *For any boolean $n \times n$ matrix A of weight $n^2 - O(n)$, we have $\mathsf{L}(A) \asymp n$.*

• The cardinality lower bound shows that the result of Theorem 9.6 is tight in order for $\log(n/q) \asymp \log n$. The authors [179] also proposed a constructive but longer way of proving part *II*.

The general problem of estimating the additive complexity of a class of boolean $m \times n$ matrices of a given weight αmn, $0 < \alpha < 1$, was posed by E. I. Nechiporuk in [226, 228]. He established the order of complexity (more asymptotically accurate than Theorem 9.6 does) for some relations between α, m, and n.

Chapter 10

Mass production method $\boxed{m}$

The idea of mass production works when the cost (complexity) of a single unit of output can be reduced by producing several similar units. In this chapter we consider, on the one hand, situations in which this is possible in principle, and, on the other hand, situations in which mass production brings useful results.

Group linear transforms $\boxed{m}$

Consider the problem of applying a linear transform AX to several disjoint variable vectors $X_1, \ldots, X_m$. In the model of monotone additive circuits, $\mathsf{L}(AX_1, \ldots, AX_m) = m \cdot \mathsf{L}(AX)$ is obviously satisfied. However, when performing calculations in the field, one can often obtain a better bound exploiting fast matrix multiplication. The following example is taken from the work of W. Paul [246][1)].

Theorem 10.1. *Let $\mathbb{F}$ be a field, $A \in \mathbb{F}^{n\times n}$ and S be a bilinear circuit for multiplication of $n \times n$ and $n \times m$ matrices over $\mathbb{F}$. Then $\mathsf{C}_{\mathcal{A}_L^{\mathbb{F}}}(AX_1, \ldots, AX_m) \leqslant \mathsf{C}(S)$.*

▶ The proof is trivial. Vectors $X_1, \ldots, X_m$ are organized into a matrix of variables $\mathbf{X}$. To compute the product $A\mathbf{X}$, we employ the circuit S, in which, since one of the inputs (matrix A) is constant, nonscalar multiplications turn into scalar ones (it is important that the circuit S is bilinear). ■

Applying the known complexity bound for matrix multiplication [75, 8], we obtain

Corollary 10.1. *Let $A \in \mathbb{B}^{n\times n}$. Then $\mathsf{L}_{\oplus}(I_n \otimes A) \preccurlyeq n^{2.38}$.*

• The corollary indicates the existence of a boolean matrix of the form $A = \begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix}$, which can be computed easier than its constituent matrices A_1 and A_2: $\mathsf{L}_{\oplus}(A) < \mathsf{L}_{\oplus}(A_1) + \mathsf{L}_{\oplus}(A_2)$. It suffices to consider a matrix $I_n \otimes A$ satisfying the condition $\mathsf{L}_{\oplus}(A) \sim n^2/\log_2 n$. Employing results on fast multiplication by narrow rectangular matrices, for example, [72], one can verify that the savings in joint implementation of matrices can reach two times: $\sup_{\mathsf{L}(A)>0} \frac{\mathsf{L}(A_1)+\mathsf{L}(A_2)}{\mathsf{L}(A)} = 2$.

[1)] In [246] it is formulated with respect to the implementation of linear transforms over $\{\mathbb{B}, \vee\}$ by circuits over a complete basis.

Computing a boolean function on multiple inputs $\boxed{m}\boxed{c}$

In light of the fact that the path to the optimal implementation of a random boolean function lies through the computation of a linear operator (Theorem 3.2), it is natural to expect that it is also possible to speed up the computation of function values on several inputs. A result by D. Uhlig [322] confirms this assumption (we prove it in a weakened form).

Theorem 10.2 ([322]). *Let X, Y be disjoint groups of n boolean variables. Then for any boolean function f,*

$$\mathsf{C}_{\mathcal{B}_2}(f(X), f(Y)) \lesssim 2^n/n.$$

▶ Decompose the function $f(X)$ in the first r variables X_1:

$$f(X_1, X_2) = \bigoplus_{\sigma\in\mathbb{B}^r} X_1^\sigma \cdot f_\sigma(X_2), \qquad X_1^\sigma = \prod_{i=1}^{r} x_{1,i}^{\sigma_i}.$$

Assume we have circuits that independently compute $2^r + 1$ functions

$$g_0 = f_0,\ \ g_1 = f_0 \oplus f_1,\ \ \ldots,\ \ g_i = f_{i-1} \oplus f_i,\ \ \ldots,\ \ g_{2^r-1} = f_{2^r-2} \oplus f_{2^r-1},\ \ g_{2^r} = f_{2^r-1}$$

(here the indices $i = \sigma$ are interpreted as binary numbers). Note that

$$g_0 \oplus \ldots \oplus g_i = f_i = g_{i+1} \oplus \ldots \oplus g_{2^r}.$$

Thus, if $X_1 = i \leqslant j = Y_1$, then we can determine $f_i(X_2)$ and $f_j(Y_2)$ by feeding the variables X to the inputs of the circuits computing $g_0, \ldots, g_i$, and the variables Y to the inputs of the circuits computing $g_{j+1}, \ldots, g_{2^r}$. In the case $i > j$, the variables X and Y interchange. In accordance with the specified rule, we introduce indicator functions

$$\eta_i^X = (i \leqslant X_1 \leqslant Y_1) \vee (i > X_1 > Y_1), \qquad \eta_i^Y = (i > Y_1 \geqslant X_1) \vee (i \leqslant Y_1 < X_1),$$

that choose whether to use a circuit for g_i to compute $f(X)$ or to compute $f(Y)$. Finally, the true values of $f(X)$ and $f(Y)$ are determined by the formulas

$$f(X) = \bigoplus_{i=0}^{2^r} \eta_i^X g_i(Z_i), \qquad f(Y) = \bigoplus_{i=0}^{2^r} \eta_i^Y g_i(Z_i), \qquad Z_i = \eta_i^X X_2 \vee \eta_i^Y Y_2.$$

The complexity of each indicator function and operator Z_i is linear, so

$$\mathsf{C}(f(X), f(Y)) \leqslant \sum_{i=0}^{2^r} \mathsf{C}(g_i) + O(2^r n) \lesssim (2^r + 1)\frac{2^{n-r}}{n-r} + O(2^r n)$$

according to Theorem 3.2. It suffices to set $r \sim \log n$. ■

We see that when calculating several values of a function, the function table (specially prepared) is actually used only once, which leads to a saving in complexity.

• The method of Theorem 10.2 leaves a huge reserve for increasing the number of values that can be computed asymptotically without increasing complexity. Uhlig showed that $(1 + o(1))2^r$ auxiliary functions g_i are sufficient to determine the values of f on $s = 2^{o(r/\log r)}$ independent inputs (this result can be obtained by recursively applying the method of Theorem 10.2, see [324]). As a consequence [323][2] (see also [324, 208]),

$$\mathsf{C}_{\mathcal{B}_2}(f(X_1), \ldots, f(X_s)) \lesssim 2^n/n.$$

The problem of computing multiple values is related to the problem of implementing multiplexor functions with independent address variables and common data variables. Uhlig's method automatically implies

$$\mathsf{C}_{\mathcal{B}_0}(\mu_n(X_1; Y), \ldots, \mu_n(X_s; Y)) \sim 2^{n+1},$$

as in the case $s = 1$ (Theorem 3.5). In this version of the problem, J. Holmgren and R. Rothblum [132] increased the bound on the number of multiplexors in the system that preserves the linear complexity $O(2^n)$ to $s \asymp 2^n/n^3$.

G. Galbiati and M. Fischer [96] showed that the analogue of Theorem 10.2 in a monotone basis does not hold:

$$\mathsf{C}_{\mathcal{B}_M}(f(X), g(Y)) = \mathsf{C}_{\mathcal{B}_M}(f) + \mathsf{C}_{\mathcal{B}_M}(g), \tag{10.1}$$

if X and Y share at most one variable.

Matrix multiplication. Direct sum method $\boxed{m}\boxed{\varepsilon}$

Arnold Schönhage
Universität Tübingen,
1972 to 1989

If in the previous sections it was shown that economy of complexity in mass production is possible, then in this one we will discuss the application of the principle of mass production to one of the central problems of computational theory — fast matrix multiplication.

A. Schönhage's direct sum method [278] serves as a nontrivial generalization of Corollary 8.2. It allows one to efficiently transform circuits for several independent matrix multiplications into circuits for multiplying square matrices.

By $T_1 \oplus T_2$ we denote the union of systems of bilinear forms $T_1(X, Y)$ and $T_2(X', Y')$ on disjoint sets of variables, $X \cap X' = Y \cap Y' = \varnothing$ (the new system has the meaning of a direct sum of tensors). Let $T^{\oplus s}$ denote for brevity the direct sum of s instances of T.

What benefit can direct sums bring? Obviously, $\operatorname{rk}(T_1 \oplus T_2) \leqslant \operatorname{rk} T_1 + \operatorname{rk} T_2$ and $\underline{\operatorname{rk}}(T_1 \oplus T_2) \leqslant \underline{\operatorname{rk}} T_1 + \underline{\operatorname{rk}} T_2$. Moreover, if there is a hypothesis (not yet refuted) regarding the first inequality that equality actually takes place, then this is not the case for the border ranks. An elegant example was found by Schönhage [278].

Theorem 10.3 ([278]). *For a ring R,*

$$\underline{\operatorname{rk}}^R(MM_{m,1,p} \oplus MM_{1,(m-1)(p-1),1}) = mp + 1.$$

[2] Cited from [331].

► To write a representation that proves the upper bound, it is convenient to employ trilinear identity (8.9). Let

$$\sum_{i=1}^{m}\sum_{j=1}^{p} x_i y_j z_{j,i}, \qquad \sum_{i=1}^{m-1}\sum_{j=1}^{p-1} x_{i,j} y_{i,j} z$$

be trilinear forms that need to be expressed. We introduce additional notation

$$x_{m,j} = -\sum_{i=1}^{m-1} x_{i,j}, \qquad y_{m,j} = 0, \qquad x_{i,p} = 0, \qquad y_{i,p} = -\sum_{j=1}^{p-1} y_{i,j}.$$

A suitable representation is

$$u^2 \sum_{i=1}^{m}\sum_{j=1}^{p}(x_i y_j z_{j,i} + x_{i,j} y_{i,j} z) = \sum_{i=1}^{m}\sum_{j=1}^{p}(x_i + u x_{i,j})(y_j + u y_{i,j})(z + u^2 z_{j,i}) - \left(\sum_{i=1}^{m} x_i\right)\left(\sum_{j=1}^{p} y_j\right) z \mod u^3.$$

The accuracy of the bound is obvious due to the fact that the system contains $mp+1$ linearly independent forms. ■

• In this case $\underline{\mathrm{rk}}\, \mathbb{MM}_{m,1,p} = mp$ and $\underline{\mathrm{rk}}\, \mathbb{MM}_{1,(m-1)(p-1),1} = (m-1)(p-1)$ (due to the dimensions of the systems; in the second case, taking into account Lemma 8.5 — commutativity is not required for the proof).

The following result is often called Schönhage's τ-theorem.

Theorem 10.4 ([278]). *Let* $\underline{\mathrm{rk}}^R \bigoplus_{i=1}^{k} \mathbb{MM}_{m_i,p_i,q_i} = r > 2$ *and* $m_i p_i q_i \geqslant 2$ *hold in a commutative ring* R*, and let* τ *be determined from* $\sum_i (m_i p_i q_i)^{\tau/3} = r$*. Then*

$$\mathsf{C}_{\mathcal{A}^R}(\mathbb{MM}_n) \preccurlyeq n^{\tau+o(1)}.$$

► First, we note that the tensor product operation is distributive with respect to the direct sum:

$$(T_1 \oplus T_2) \otimes T = (T_1 \otimes T) \oplus (T_2 \otimes T), \qquad T \otimes (T_1 \oplus T_2) = (T \otimes T_1) \oplus (T \otimes T_2).$$

In particular, Lemma 5.4 as applied to the product of direct sums is formulated as follows: if a system $\bigoplus_{i=1}^{I} T_i$ has a (d_1, r_1)-representation, and $\bigoplus_{j=1}^{J} T'_j$ has a (d_2, r_2)-representation over the ring R, then the system $\bigoplus_{i=1}^{I}\bigoplus_{j=1}^{J}(T_i \otimes T'_j)$ admits a $(d_1 + d_2, r_1 r_2)$-representation.

Let us prove one more lemma on the border rank of tensor products.

Lemma 10.1 ([278]). *Let a system* T_1 *have a* (d, r)*-representation and* $T_2^{\oplus r}$ *have a* (d', r')*-representation over* R*. Then* $T_1 \otimes T_2$ *admits a* $(d + d', r')$*-representation.*

▷ Using the given (d,r)-representation for T_1, write

$$u^d(T_1\otimes T_2)=\sum_{l=1}^{r}C_l(u)\otimes(X_l(u)Y_l(u))\bmod u^{d+1},$$

where $C_l(u)\in R[u]^{\dim T_1}$, and $X_l(u)$, $Y_l(u)$ are linear combinations of the vectors of variables X^i and Y^j, respectively. Considering the r internal products X_lY_l to be independent, we express them by the given (d',r')-representation (multiplying by $u^{d'}$):

$$u^{d+d'}(T_1\otimes T_2)=\sum_{l=1}^{r}C_l(u)\otimes\left(\sum_{s=1}^{r'}C'_{l,s}(u)X'_s(u)Y'_s(u)\right)\bmod u^{d+d'+1}=$$
$$\sum_{l=1}^{r}\sum_{s=1}^{r'}(C_l(u)\otimes C'_{l,s}(u))X'_s(u)Y'_s(u)\bmod u^{d+d'+1},$$

where $C'_{l,s}(u)\in R[u]^{\dim T_2}$, and $X'_s(u)$, $Y'_s(u)$ are linear combinations of the components of the vectors $X_l(u)$ and $Y_l(u)$, respectively, i.e., ultimately, simply linear combinations of the variables. The required representation is constructed. □

Let us proceed directly to the proof of the theorem, which we will carry out only for the case $k=2$ (the general case does not principally differ). By h-fold application of Lemma 5.4 from a (d,r)-representation for $MM_{m_1,p_1,q_1}\oplus MM_{m_2,p_2,q_2}$ we obtain an (hd,r^h)-representation for the system $\bigoplus_{s=0}^{h}MM_{m_1^sm_2^{h-s},p_1^sp_2^{h-s},q_1^sq_2^{h-s}}^{\oplus C_h^s}$ in view of (5.13). From

$$r^h=((m_1p_1q_1)^{\tau/3}+(m_2p_2q_2)^{\tau/3})^h=\sum_{s=0}^{h}C_h^s(m_1p_1q_1)^{\tau s/3}(m_2p_2q_2)^{\tau(h-s)/3}$$

it follows that for some $s=s(h)$,

$$C_h^s(m_1p_1q_1)^{\tau s/3}(m_2p_2q_2)^{\tau(h-s)/3}\geqslant\frac{r^h}{h+1}.\tag{10.2}$$

Lemma 10.2. *Let $g(h)=\lceil\gamma^{h^\alpha}r^{3h/\tau}\rceil$. Then, for a suitable choice of constants $0<\gamma,\alpha<1$, there exists an (hb,r^{3h})-representation for $MM_{g(h)}$, where $b=\lceil\frac{3\log_2 r}{\log_2 r-1}d\rceil$.*

▷ *I*. We prove the induction step, assuming that the statement holds for all values $<h$ of the parameter.

Let $r^{3h_0}\leqslant C_h^{s(h)}<r^{3h_0+3}$, i.e., $h_0=\lfloor(\log_r C_h^{s(h)})/3\rfloor$. Set $n=g(h_0)$ and $s=s(h)$. Thus, based on the (hd,r^h)-representation constructed above for the system $MM_{m_1^sm_2^{h-s},p_1^sp_2^{h-s},q_1^sq_2^{h-s}}^{\oplus r^{3h_0}}$ and the assumed (h_0b,r^{3h_0})-representation for MM_n, by Lemma 10.1 we obtain an $(hd+h_0b,r^h)$-representation for $MM_{nm_1^sm_2^{h-s},np_1^sp_2^{h-s},nq_1^sq_2^{h-s}}$. Therefore, by Lemma 5.4 we have a $(3hd+3h_0b,r^{3h})$-representation for $MM_{n^3(m_1p_1q_1)^s(m_2p_2q_2)^{h-s}}$. By the choice of s, from (10.2) follows

$$n^3(m_1p_1q_1)^s(m_2p_2q_2)^{h-s}\geqslant\left(\frac{r^hn^\tau}{(h+1)C_h^s}\right)^{3/\tau}.$$

To justify the induction step, it suffices to show that the left-hand side of the inequality is not less than $g(h)$.

Let $f(h) = \gamma^{h^\alpha}$. Note that in view of $h_0 < (h/3)\log_r 2$,

$$\frac{n^\tau}{C_h^s} > \frac{g^\tau(h_0)}{r^{3h_0+3}} \geqslant \frac{f^\tau(h_0)}{r^3} > \frac{f^\tau(\frac{h}{3\log_2 r})}{r^3}.$$

If we ensure the inequality

$$\frac{f^3(\frac{h}{3\log_2 r})}{(r^3(h+1))^{3/\tau}} \geqslant f(h), \tag{10.3}$$

then we obtain the required relation (taking into account rounding to the nearest integer from above)

$$\left(\frac{r^h n^\tau}{(h+1)C_h^s}\right)^{3/\tau} > \frac{r^{3h/\tau} f^3(\frac{h}{3\log_2 r})}{(r^3(h+1))^{3/\tau}} \geqslant f(h) r^{3h/\tau}.$$

To ensure (10.3), we choose $\alpha < 1$ arbitrarily from the condition $(3\log_2 r)^\alpha > 3$. Then (10.3) holds for any $\gamma \in (0,\, 1)$ for sufficiently large h, say, for $h \geqslant h_1$ for any $\gamma \leqslant \gamma_1$. Finally, note that by the choice of b, we have $hb \geqslant 3hd + 3h_0 b$.

II. It remains to specify the induction base. Let $r^3 \leqslant C_h^{s(h)}$ hold for $h \geqslant h_2$. We include in the induction base all $h \leqslant \max\{h_1, h_2\}$ (from $h \leqslant h_2$ it follows that $h_0 \geqslant 1$ in the induction step). Finally, we choose γ small enough, $\gamma \leqslant \gamma_1$, so that the desired representation for $MM_{g(h)}$ exists for all $h \leqslant \max\{h_1, h_2\}$. For example, for this it is sufficient to ensure that $g(h) \leqslant r^h$ for such h. □

Let $g^{k-1}(h) < n \leqslant g^k(h)$. From Lemma 10.2 and Lemma 5.3 it follows that $\operatorname{rk} MM_{g(h)} \leqslant ch^2 r^{3h}$, where $c = O(1)$. Then by Theorem 5.3,

$$\mathsf{C}_{\mathcal{A}^R}(MM_n) \leqslant \mathsf{C}_{\mathcal{A}^R}(MM_{g^k(h)}) \preccurlyeq g^2(h)(ch^2 r^{3h})^{k+2} \preccurlyeq$$
$$g^2(h) r^{9h} (ch^2)^{k+2} \left(\gamma^{-h^\alpha} g(h)\right)^{(k-1)\tau} \preccurlyeq r^{c_1 h} c_2^{kh^\alpha} \cdot n^\tau$$

for suitable constants c_1, c_2. The assertion of the theorem follows, for example, when choosing $h \asymp \log_r^{1/2} n$ (then $k \preccurlyeq h$). ■

Applying Theorem 10.4 to the example from Theorem 10.3 with the choice of parameters $m = p = 4$, we obtain

Corollary 10.2 ([278]). *If R is a commutative ring, then*

$$\mathsf{C}_{\mathcal{A}^R}(MM_n) \prec n^{2.548}.$$

• Probably the strongest bound directly using the method of Theorem 10.4 was obtained by D. Coppersmith and S. Winograd in [74], $\mathsf{C}_{\mathcal{A}^R}(MM_n) \prec n^{2.496}$. More modern theoretically fast methods of matrix multiplication employ the direct sum method as a working tool.

Matrix multiplication. Laser method $\boxed{m}\boxed{\varepsilon}$

The idea of mass production turned out to be extremely popular in the problem of fast matrix multiplication. In the "laser" method of V. Strassen [315] several variants of this idea are combined at once.

The further presentation follows closely the work of D. Coppersmith and S. Winograd [75]. The method allows one to construct algorithms for matrix multiplication from identities for systems of bilinear forms that are not matrix products. For example, the system $T_q = \{x_0y_i + x_iy_0 \mid i = 1, \dots, q\}$ can be (approximately) computed as

$$u(x_0y_i + x_iy_0) = (x_0 + ux_i)(y_0 + uy_i) - x_0y_0 \mod u^2 \tag{10.4}$$

(an example from [315]). Therefore $\underline{\mathrm{rk}}\, T_q \leqslant q + 1$. The system T_q is obtained by summing the components of the matrix products $MM_{1,1,q}$ and $MM_{q,1,1}$, but this sum is not direct, so Theorem 10.4 cannot be applied immediately. Strassen's method explains how to go from (10.4) to an identity with a direct sum on the left-hand side.

Theorem 10.5 ([315]). *If R is a commutative ring, then*

$$\mathsf{C}_{\mathcal{A}^R}(MM_n) \prec n^{2.48}.$$

▶ The first step is to reduce identity (10.4) to a more symmetric form. Consider the tensor product $T_q \otimes T'_q \otimes T''_q$, where the systems T'_q, T''_q are obtained from T_q by a cyclic shift of the groups of variables (x, y, z) in the trilinear form $\sum_{i=1}^q (x_0y_i + x_iy_0)z_i$ corresponding to T_q, see (8.9). Thus,

$$T'_q = \left\{ x_0y_1, \dots, x_0y_q, \sum_{j=1}^q x_jy_j \right\}, \qquad T''_q = \left\{ x_1y_0, \dots, x_qy_0, \sum_{k=1}^q x_ky_k \right\}.$$

By construction, $\underline{\mathrm{rk}}\, T_q = \underline{\mathrm{rk}}\, T'_q = \underline{\mathrm{rk}}\, T''_q$ (in fact, this is proved in Lemma 8.5 for the special case of matrix multiplication). In particular, the system T'_q has a $(1, q+1)$-representation of the form

$$ux_0y_j = u(x_0 + ux_j)y_j \mod u^2, \qquad u\sum_{j=1}^q x_jy_j = \sum_{j=1}^q (x_0 + ux_j)y_j - x_0\sum_{j=1}^q y_j.$$

The system $T_q \otimes T'_q \otimes T''_q$ consists of bilinear forms $x_{00k}y_{ij0} + x_{i0k}y_{0j0}$ (q^3 pieces), $\sum_{k=1}^q x_{00k}y_{ijk} + x_{i0k}y_{0jk}$ (q^2 pieces), $\sum_{j=1}^q x_{0jk}y_{ij0} + x_{ijk}y_{0j0}$ (q^2 pieces), and $\sum_{j,k=1}^q x_{0jk}y_{ijk} + x_{ijk}y_{0jk}$ (q pieces). In it one can detect a structure of the block product of 2×2 matrices:

$$\begin{bmatrix} x_{00k}y_{ij0} + x_{i0k}y_{0j0} & x_{00k}y_{ijk} + x_{i0k}y_{0jk} \\ x_{0jk}y_{ij0} + x_{ijk}y_{0j0} & x_{0jk}y_{ijk} + x_{ijk}y_{0jk} \end{bmatrix} = \begin{bmatrix} x_{00k} & x_{i0k} \\ x_{0jk} & x_{ijk} \end{bmatrix} \cdot \begin{bmatrix} y_{ij0} & y_{ijk} \\ y_{0j0} & y_{0jk} \end{bmatrix} \tag{10.5}$$

(summation is performed over repeated indices; symbols of summation are omitted for brevity). The complication here is that the matrix blocks are essentially 3-dimensional

tensors, which cannot be consistently represented by ordinary (2-dimensional) matrices. However, individual block products *are* matrix products: for example, $x_{00k}y_{ij0}$ represents the operator $MM_{q,1,q^2}$, and $x_{ijk}y_{0jk}$ represents the operator $MM_{q,q^2,1}$.

Strassen's key observation is that from an ordinary product of (large enough) matrices one can extract many individual products of components (or blocks).

Lemma 10.3. *Let the system MM_n have a (d,r)-representation. Then the system*

$$B_{n,s} = \{x_{ij}y_{jk} \mid 1 \leqslant i,j,k \leqslant n,\ i+j+k = s\}$$

admits a $(d+h,r)$-representation, $h \preccurlyeq n^2$.

▷ Keeping in mind (8.9), we can assume that we are given a (d,r)-representation for the trilinear form $\sum_{1\leqslant i,j,k\leqslant n} x_{ij}y_{jk}z_{ki}$. After the change of variables

$$x_{ij} := u^{i^2+2ij} \cdot x_{ij}, \qquad y_{jk} := u^{j^2+2j(k-s)} \cdot y_{jk}, \qquad z_{ki} := u^{(k-s)^2+2(k-s)i} \cdot z_{ki}$$

taking into account $i^2+2ij+j^2+2j(k-s)+(k-s)^2+2(k-s)i = (i+j+k-s)^2$ we obtain an almost (d,r)-representation for $\sum_{i+j+k=s} x_{ij}y_{jk}z_{ki}$, which may differ from a correct representation by occurrences of negative powers of u. Negative powers can be eliminated by multiplying by u^h with exponent $h \preccurlyeq n^2$. □

It is important that any variable x_{ij} or y_{jk} has at most one occurrence in $B_{n,s}$ (a pair of indices from i,j,k uniquely determine the third), so $B_{n,s}$ is a direct sum of individual forms. For $s \sim 3n/2$, the system $B_{n,s}$ contains $(3/4-o(1))n^2$ forms.

By Lemma 5.4, the system $T_q \otimes T'_q \otimes T''_q$ has a $(3,(q+1)^3)$-representation. Then the tensor power $(T_q \otimes T'_q \otimes T''_q)^{\otimes m}$ has a $(3m,(q+1)^{3m})$-representation, also by Lemma 5.4. As a consequence of (10.5), the system $(T_q \otimes T'_q \otimes T''_q)^{\otimes m}$ has a structure of the product of $2^m \times 2^m$ matrices composed from blocks X_{ij} and Y_{jk}, where the product of two blocks $X_{ij}Y_{jk}$ represents the multiplication of matrices MM_{q^a,q^b,q^c}, $a+b+c = 3m$. Therefore, according to Lemma 10.3, the border rank of the system $B_{2^m,s} \otimes \{X_{ij}Y_{jk} \mid 1 \leqslant i,j,k \leqslant 2^m\}$ does not exceed $(q+1)^{3m}$. But this system is the direct sum of matrix products MM_{q^a,q^b,q^c}. Therefore, applying Theorem 10.4, when choosing $s \sim (3/2)2^m$ we deduce $\mathsf{C}_{\mathcal{A}^R}(MM_n) \preccurlyeq n^{\tau+o(1)}$, where τ is determined from the condition $(3/4)2^{2m}q^{\tau m} = (q+1)^{3m}$, or, after taking the root, from $4q^\tau = (q+1)^3$. The statement of the theorem is obtained for $q = 5$. ■

• The method was further developed by D. Coppersmith and S. Winograd in [75], where, using more complex basic representations than (10.4), they obtained the bound $\mathsf{C}_{\mathcal{A}^R}(MM_n) \prec n^{2.376}$, which remained a record for 20 years. The current best[3)] bound is $\mathsf{C}_{\mathcal{A}^R}(MM_n) \prec n^{2.372}$ [8]. For a systematic exposition of the theory of fast methods of matrix multiplication, see also [6, 53, 340, 35].

[3] April 2024

Other applications

Monotone circuits for slice-functions. An interesting application of the mass production method was found by A. Hiltgen and M. Paterson [128]. It concerns the joint computation of similar slice-functions. A *k-slice function* is a monotone function of the form $T^k_{|X|}(X)f(X) \vee T^{k+1}_{|X|}(X)$ — it takes value 0 on all inputs of weight $< k$ and equal to 1 on all inputs of weight $> k$ (here f is an arbitrary boolean function).

Let $X = (X_1, \ldots, X_r)$, $|X_1| = \ldots = |X_r| = m$, $n = rm$. Consider the problem of computing a family of k-slice functions $f_i(X) = X_i^{\alpha_1} \vee \ldots \vee X_i^{\alpha_s} \vee T_n^{k+1}(X)$, where α_j are distinct vectors of weight $k \leqslant m$, and $X_i^\sigma = \prod_{\sigma_j=1} x_{i,j}$ are monomials of the variables X_i. The presence of a common part $T_n^{k+1}(X)$ takes this problem out of the scope of rule (10.1).

In the notation $\dot{x}_{i,j} = x_{i,j} \cdot T_m^k(X_i)$, $\dot{x}_j = \dot{x}_{1,j} \vee \ldots \vee \dot{x}_{r,j}$, $\dot{X} = (\dot{x}_1, \ldots, \dot{x}_m)$, each of the functions f_i can be computed as

$$f_i(X) = T_m^k(X_i)(\dot{X}^{\alpha_1} \vee \ldots \vee \dot{X}^{\alpha_s}) \vee T_n^{k+1}(X),$$

hence, $\mathsf{C}_{\mathcal{B}_M}(f_1, \ldots, f_r) < n + ks + r\mathsf{C}_{\mathcal{B}_M}(T_m^k) + \mathsf{C}_{\mathcal{B}_M}(T_n^{k+1}) = ks + O(n \log n)$. For sufficiently large r, this method of computation is more economical than independent implementation of functions f_i or sums $X_i^{\alpha_1} \vee \ldots \vee X_i^{\alpha_s}$.

Applying this technique in a more subtle way, the authors of [128] obtained an asymptotically tight estimate $\mathsf{C}_{\mathcal{B}_M}(\{f_{a,b} \mid 0 \leqslant a, b < p\}) \sim 3n$ of the complexity of the family of Nechiporuk's 1-slice functions, where $n = p^2$ and $p \in \mathbb{P}$. Here $X = (x_{i,j})$ and $f_{a,b}(X) = \bigvee_{i=0}^{p-1} x_{i,ai+b} \vee T_n^2(X)$ (index operations are performed modulo p). The construction of the functions follows the incidence matrix of points and lines of a finite affine plane over $\mathbb{F}_p$. As E. I. Nechiporuk established in [229], a linear operator over $(\mathbb{B}, \vee)$ with such a matrix[4)] has monotone complexity $(p-1)n^2 \sim n^{3/2}$. Actually, the conjecture [330] that the slice version of the operator should also have complexity of order $n^{3/2}$ was refuted in [128].

[4]The components of the operator are boolean sums $\bigvee_{i=0}^{p-1} x_{i,ai+b}$ — any two such sums have at most one common variable.

Chapter 11

Combinatorial methods

In a number of problems in the theory of fast computing, it is convenient to organize variables or intermediate data in a structure with certain combinatorial properties.

Monotone circuits for the symmetric threshold-2 function

For the function T_n^2, the lower complexity bound $\mathsf{C}_{\mathcal{B}_2}(T_n^2) \geqslant 2n - O(1)$ was one of the first to be obtained (by B. M. Kloss in [165]), and immediately in a complete basis. The question of the possibility of computing the function with such complexity remained open until L. Adleman in the 1970s found an elegant way[1)], and immediately for a monotone basis. In Adleman's method, the variables correspond to nodes of a two-dimensional lattice. It is essential that any two nodes are either in different rows or in different columns of the lattice.

Leonard Max Adleman

University of Southern California, since 1980

Theorem 11.1. $\mathsf{C}_{\mathcal{B}_M}(T_n^2) \leqslant 2n + O(\sqrt{n})$.

▶ Let a set of n variables be numbered by two indices $x_{i,j}$, $i = 1, \ldots, p$, $j = 1, \ldots, q$, where $pq \geqslant n$. Denote $u_i = \bigvee_j x_{i,j}$ and $v_j = \bigvee_i x_{i,j}$. Then

$$T_n^2(X) = T_p^2(u_1, \ldots, u_p) \vee T_q^2(v_1, \ldots, v_q),$$

therefore,

$$\mathsf{C}(T_n^2) \leqslant 2n - p - q + 1 + \mathsf{C}(T_p^2) + \mathsf{C}(T_q^2),$$

since all sums u_i and v_j may be computed with complexity $2n - p - q$. Now it suffices to choose $p = q \approx \sqrt{n}$. ■

[1] Not published by the author, cited from [37, 331].

• More precisely, the estimate extracted from Theorem 11.1 is $\mathsf{C}_{\mathcal{B}_M}(T_n^2) \leqslant 2n + (2 + o(1))\sqrt{n}$. The lower estimate proved by the author in [301] is $\mathsf{C}_{\mathcal{B}_M}(T_n^2) \geqslant 2n + \sqrt{2n/3} - O(1)$.

Adleman's method can be generalized and allows one to obtain the bound $\mathsf{C}_{\mathcal{B}_M}(T_n^k) \leqslant kn + o(n)$ for any constant k, see [331], however, in this case the adapted Yao's method has priority, see [301].

Asymptotic complexity of formulae ∵ c

In the theory of asymptotically optimal synthesis, various coverings of the boolean cube are applied, which have useful properties within the framework of a given computational model. The method of covering the cube with spheres was proposed by O. B. Lupanov [202] for constructing economical switching circuits and subsequently found application in a number of other problems. We will consider an application of the method to the synthesis of asymptotically minimal formulae over the basis $\mathcal{B}_0$ — Lupanov's result [203].

A *sphere* (of radius 1) in the space $\mathbb{B}^r$ is defined as a set of boolean vectors of length r that differ from a given vector (the center of the sphere) in exactly one position. The most important property of a sphere is the possibility to select any its point using one variable in the following sense. Let $\varphi_\alpha(X)$ be the characteristic function[2)] of a sphere S with center α. Then for an arbitrary vector $\sigma \in S$, we have $X^\sigma = \varphi_\alpha(X) x_i^{\sigma_i}$, where i is the number of the position that distinguishes σ from α.

Starting from the perfect Hamming code with distance 3, it is easy to check that for $r = 2^k$ the boolean cube $\mathbb{B}^r$ decomposes into $2^r/r$ spheres, see, e.g., [207].

• For $r = 2^k$, the Hamming code generates a partition of the cube $\mathbb{B}^{r-1}$ into $s = 2^{r-1}/r$ disjoint balls of radius 1 with centers $\alpha_1, \ldots, \alpha_s$. As a consequence, the set of spheres with centers $\{(\alpha_i, \beta) \mid 1 \leqslant i \leqslant s, \beta \in \mathbb{B}\}$ forms a partition of the cube $\mathbb{B}^r$. In the general case $r \neq 2^k$ the cube $\mathbb{B}^r$ can be covered by $(1 + o(1))2^r/r$ spheres [143].

Before proceeding to the proof of the main result, we obtain an auxiliary bound.

Lemma 11.1 ([203]). *Let* $|X| = p$, $|Y| = r$, *and*

$$f(X, Y) = y_1 f_1(X) \vee y_2 f_2(X) \vee \ldots \vee y_r f_r(X). \tag{11.1}$$

Then $\Phi_{\mathcal{B}_0}(f) \leqslant 2^p \left(\frac{r}{s} + p2^s\right)$ *for any* $s \in \mathbb{N}$.

▷ Let us divide the boolean vectors of length p into groups $I_1, \ldots, I_q$ of s in each (for simplicity, we assume $2^p = qs$). Let $\chi_{j,\tau}(X)$ be the characteristic function of a subset of the group I_j, defined by the vector[3)] $\tau \in \mathbb{B}^s$. Then

$$f(X, Y) = \bigvee_{k=1}^{q} \bigvee_{\tau \in \mathbb{B}^s} \chi_{k,\tau}(X) D_{k,\tau}(Y), \qquad D_{k,\tau}(Y) = \bigvee_{f_i(I_k) = \chi_{k,\tau}(I_k)} y_i. \tag{11.2}$$

[2]The characteristic function of a set S takes the value 1 at points of S, and the value 0 — outside the set.

[3]The i-th vector belongs to the set if $\tau_i = 1$.

Since $\sum_\tau \Phi(D_{k,\tau}) = r$ for any k, and $\Phi(\chi_{j,\tau}) \leqslant ps$ (e.g., consider a perfect DNF), formula (11.2) has complexity

$$\Phi(f) \leqslant qr + q2^s ps = 2^p \left(\frac{r}{s} + p2^s\right).$$

□

Theorem 11.2 ([203]). *For an arbitrary boolean function f of n variables,*

$$\Phi_{\mathcal{B}_0}(f) \leqslant \left(1 + O\left(\frac{\log\log n}{\log n}\right)\right) \frac{2^n}{\log_2 n}.$$

▶ Let $X = (X_1, X_2, X_3)$, $|X_1| = p$, $|X_2| = r = 2^k$, $|X_3| = n - p - r$. Employing a partition of the cube $\mathbb{B}^r$ into spheres with characteristic functions $\varphi_i(X_2)$, we write

$$f(X) = \bigvee_{\sigma\in\mathbb{B}^{n-p-r}} f_\sigma(X_1, X_2)\cdot X_3^\sigma = \bigvee_{i=1}^{2^r/r} \varphi_i(X_2) \bigvee_{\sigma\in\mathbb{B}^{n-p-r}} f_{i,\sigma}(X_1, X_2)\cdot X_3^\sigma, \qquad (11.3)$$

where each of the functions $f_{i,\sigma}(X_1, X_2)$ has the form (11.1) (the variables X_2 or their negations act as the variables Y).

Estimating roughly $\Phi(\varphi_i) \leqslant r^2$ and applying Lemma 11.1, for the complexity of computing the function by formula (11.3) we derive

$$\Phi(f) \leqslant \frac{2^r}{r}\left(r^2 + 2^{n-p-r}\left(n - p - r + 2^p\left(\frac{r}{s} + p2^s\right)\right)\right) < 2^r r + 2^{n-p}\cdot\frac{n}{r} + \frac{2^n}{s} + 2^{n+s}\cdot\frac{p}{r}.$$

The required bound is obtained by choosing parameters $n/4 \leqslant r \leqslant n/2$, $p \approx 2\log_2\log n$, $s \approx \log_2 n - 2\log_2\log n$. Note that the dominant complexity comes from the variables that select the points of the spheres. ■

That the result of the theorem is asymptotically tight was shown by J. Riordan and C. Shannon in [262] (this was the first application of the cardinality argument to obtaining lower complexity bounds in synthesis theory).

- In [193] S. A. Lozhkin improved the accuracy of the formula complexity bound to

$$\Phi_{\mathcal{B}_0}(\mathcal{P}^n) = \left(1 \pm O\left(\frac{1}{\log n}\right)\right)\frac{2^n}{\log_2 n}. \qquad (11.4)$$

The asymptotic estimate of the complexity of formulae automatically implies a bound on the depth $\mathsf{D}_{\mathcal{B}_0}(\mathcal{P}^n) \geqslant n - \log_2\log_2 n - o(1)$. Using a modification of the representation (11.3), also based on a partition of the boolean cube into spheres, S. B. Gashkov [99] obtained an upper bound $\mathsf{D}_{\mathcal{B}_0}(\mathcal{P}^n) \leqslant \lceil n - \log_2\log_2 n + o(1)\rceil + 2$. Essentially ultimate result

$$\mathsf{D}_{\mathcal{B}_0}(\mathcal{P}^n) \leqslant \lceil n - \log_2\log_2 n + o(1)\rceil \qquad (11.5)$$

was established by Lozhkin [192] via a partition of the cube into special sphere-like sets. A simple way to derive a slightly weaker bound is given below, see Corollary 12.1. Lozhkin also showed [195] that the complexity bound (11.4) and the depth bound (11.5) up to an additive term $O(1)$ are achieved on a single formula.

Gashkov [100] adapted Lupanov's method to the computation of polynomials with coefficients 0 and 1 by formulae in the arithmetic basis $\{*, \pm, 1\}$. The complexity of the class of polynomials with individual constraints $d_i - 1$ on the exponents of each variable x_i is asymptotically equal to $d_1\cdot\ldots\cdot d_n/\log_2 n$. The proof employs coverings of parallelepipeds in $\mathbb{N}^n$ by hemispheres instead of coverings by spheres.

Multiplicative complexity of polynomials ∵

Recall that a degree-d polynomial of one variable can be computed by an arithmetic circuit with $\asymp \sqrt{d}$ nonscalar multiplications, and this bound is best possible [244] (see p. 33). Consider a more general problem of implementing polynomials of n variables. It is known that in the case $d \geqslant 2$ about $\sqrt{C_{n+d}^d}$ (up to an order) multiplications are necessary [59][4]. S. Lovett [190] showed that this bound is almost achievable. The method is based on the idea of covering the set of vector exponents of monomials by the sum of two sets of approximately half the dimension.

Theorem 11.3 ([190]). *Any polynomial $f \in R[x_1, \ldots, x_n]$ of degree d can be computed by a circuit over $\mathcal{A}^R$ with nonscalar multiplicative complexity $(dn)^{O(1)}\sqrt{C_{n+d}^d}$.*

▶ The proof is based on the observation that any monomial[5] X^e of degree $\leqslant d$ can be represented as $X^a X^b$, where $a \in A$, $b \in B$, given a suitable choice of sets A and B whose cardinality is close to $\sqrt{C_{n+d}^d}$.

It suffices to consider the case where n is odd and d is even. Let $A = B \subset \mathbb{N}_0^n$ be the set of all vectors of weight[6] $\leqslant d/2$ with zero components in some $m = (n-1)/2$ cyclically adjacent[7] positions $i+1, i+2, \ldots, i+m \subset \mathbb{Z}_n$.

Lemma 11.2 ([190]). *Any vector $e = (e_1, \ldots, e_n) \in \mathbb{N}_0^n$ of weight d belongs to the Minkowski sum[8] $A + A$.*

▷ Let w_i^- and w_i^+ denote the sums of the components of the vector e in cyclically adjacent positions $i-1, i-2, \ldots, i-m$ and, respectively, $i+1, i+2, \ldots, i+m$. Note that $w_i^- + w_i^+ + e_i = d$.

First, we show that for some i we have $w_i^-, w_i^+ \leqslant d/2$. Suppose that $w_j^+ > d/2$ for some j. Therefore, $w_j^- = w_{j+m}^+ \leqslant d/2$. As a consequence, for some i we have $w_{i-1}^+ > d/2$ and $w_i^+ \leqslant d/2$. But then $w_i^- \leqslant d - w_{i-1}^+ \leqslant d/2$.

Now $e = a + b$, where

$$\begin{aligned} &a_{i+1} = e_{i+1},\ \ldots,\ a_{i+m} = e_{i+m}, \quad a_{i-1} = \ldots = a_{i-m} = 0, \quad a_i = d/2 - w_i^+, \\ &b_{i-1} = e_{i-1},\ \ldots,\ b_{i-m} = e_{i-m}, \quad b_{i+1} = \ldots = b_{i+m} = 0, \quad b_i = d/2 - w_i^-. \end{aligned}$$

Hence, $a, b \in A$. □

As follows from the lemma, a polynomial $f = \sum f_e X^e$ can be represented as

$$f(X) = \sum_{a \in A} X^a \cdot \sum_{b \in A} c_{a,b} X^b, \qquad c_{a,b} \in \{0,\ f_{a+b}\}. \tag{11.6}$$

[4] At least, if we consider polynomials over a field.
[5] As usual, by X^e we understand $\prod_i x_i^{e_i}$.
[6] Here the weight of a vector is defined as the sum of its components.
[7] That is, consecutive in the numbering $\ldots, n, 1, 2, \ldots, n, 1, 2, \ldots$
[8] The Minkowski sum $A + B$ is defined as $\{a + b \mid a \in A,\ b \in B\}$.

All monomials X^a are computed sequentially with the use of $|A|$ multiplications, another $|A|$ multiplications are performed by the inner sums in formula (11.6). It remains to note[9)] that $|A| \leqslant nC^{d/2-1}_{(n+d-1)/2} \leqslant n\sqrt{C^{d-2}_{n+d-1}} \leqslant d\sqrt{C^d_{n+d}}$ due to the simple inequality $C^{2k}_{2n} \geqslant \left(C^k_n\right)^2$.

The cases of even n and odd d may be reduced to the one considered (at the cost of an additional factor in the complexity estimate). ■

• The method of Theorem 11.3 obviously allows one to compute monotone polynomials in the monotone basis $\mathcal{A}_+$. The question of whether one can approach closer to the lower bound $\sqrt{C^d_{n+d}}$ is still open. As can be seen from the proof of the theorem, this bound is achievable (up to an order) in some special cases, for example, for even constant d (which is also easily verified directly). The author [304] showed that for constant odd d the order of complexity is $n^{\lceil d/2 \rceil} \asymp \sqrt{nC^d_{n+d}}$ for fields of characteristic 0. The question of the complexity of the considered classes of polynomials over finite fields is open.

Circuit complexity of the logical permanent ∵ s

Above, a probabilistic construction of parallel circuits for the logical permanent function $PERM_n$ of polynomial complexity was presented (Theorem 9.5). It was based on the algebraic properties of the permanent. Further, we will see that algorithms for finding a maximal matching in a graph help explicitly build simpler, although no longer parallel, circuits.

Let $G = (U, V, E)$ be a bipartite undirected graph with parts $U = \{u_1, \ldots, u_n\}$, $V = \{v_1, \ldots, v_n\}$ and an edge set E. As usual, we introduce boolean variables $x_{i,j} = [(u_i, v_j) \in E]$. Recall that $PERM_n(X) = 1$ iff the graph G defined by X has a perfect matching[10)].

Consider some matching $M \subset E$. A path in G is called *M-alternating* if it alternates edges from M and from $E \setminus M$. A vertex of G is called *M-free* if it is not adjacent to edges from M. An alternating path is called *M-augmenting* if both its end vertices are free.

It is easy to check that a matching M is maximal in a graph G if there are no M-augmenting paths in G. Moreover, we have

Claim 11.1 ([134]). *If a bipartite graph G has a matching N of n edges, then for an arbitrary matching $M \subset G$ of $m < n$ edges there exists an M-augmenting path of length $\leqslant 2m/(n-m) + 1$.*

▷ Consider the union of matchings M and N. In each connected component of the graph $M \cup N$, any vertex is adjacent to at most one edge from M and at most one edge from N. Therefore: (a) the difference between the number of edges from M and

[9] Fixing an extreme nonzero position of the vector $a \in A$ and estimating the number of ways to partition $d/2 - 1$ into $m + 2$ summands.

[10] A *matching* is a set of edges that don't have common adjacent vertices. A perfect matching covers all vertices of the graph.

from N in a connected component does not exceed 1 in absolute value; (b) all edges of a connected component form an M-alternating path.

As a consequence, there are $\geqslant n-m$ connected components with a predominance of edges from N, and hence, the same number of non-adjacent M-augmenting paths. Thus, there exists a path with at most $m/(n-m)$ edges from M. □

It remains to note that the symmetric difference[11)] of a matching M and an M-augmenting path is a matching of size $|M|+1$.

Thus, the problem of finding a maximal matching can be solved in n successive steps of searching for augmenting paths. At each step, we construct a partition of the graph vertices into layers: on the zero layer — free vertices U_0 of the part U, and then on any d-th layer — vertices at distance d from U_0 along alternating paths. And so on until a free vertex from V is covered. Then we restore an alternating path between this vertex and a vertex from U_0.

This is a simple algorithm with the running time[12)] of order n^3, which can be found, e.g., in [2]. Representing the algorithm as a circuit requires additional elaboration (approximately as in the case of GCD algorithms, see Theorem 1.4).

Theorem 11.4. $\mathsf{C}(PERM_n) \preccurlyeq n^4 \log^3 n$.

▶ The circuit computes a sequence of matchings $M_1, M_2, \ldots, M_{n+1}$. Transformations $M_k \to M_{k+1}$ are performed by similar subcircuits S_k:

$$\varnothing = M_1 \xrightarrow{S_1} M_2 \xrightarrow{S_2} \ \ldots \ \xrightarrow{S_n} M_{n+1}.$$

The circuit S_k searches for an M_k-augmenting path and, if successful, increases the cardinality of the current matching with its help. By construction, $|M_k| \leqslant k-1$.

A matching M is characterized by a set of attributes $p_{i,j} = [(u_i, v_j) \in M]$, which are recomputed when M changes during the calculations. Along the way, we update the auxiliary attributes $p^u_i = \bigvee_j p_{i,j}$ and $p^v_j = \bigvee_i p_{i,j}$ expressing non-freedom of vertices u_i and v_j, as well as the attributes $t[u,v]$ of legality[13)] of edges in M-alternating paths:

$$t[u_i, v_j] = x_{i,j} \cdot \overline{p_{i,j}}, \qquad t[v_j, u_i] = p_{i,j}.$$

This means that from the part U to the part V the paths should pass along edges from $E \setminus M$, and in the opposite direction — along edges from M.

1) Let us describe the circuit S_k. By Claim 11.1, if a graph G has a perfect matching, then it contains an M_k-augmenting path of length at most $l_k = 2\lfloor k/(n-k) \rfloor + 1$.

Following the dynamic programming technique, we construct the shortest M_k-alternating paths of length $\leqslant l_k$ connecting every pair of vertices from G. The corresponding circuit consists of $s = \lceil \log_2 l_k \rceil$ layers-subcircuits $L_1, \ldots, L_s$. The next i-th layer adds information about alternating paths of length $\leqslant 2^i$.

[11)] The symmetric difference of sets A and B is defined as $(A \setminus B) \cup (B \setminus A)$.

[12)] The running time of an algorithm is usually understood to be the number of instructions executed by a program implementing it on a general-purpose computer.

[13)] The concept of legality introduces the orientation of edges in a graph so that all oriented paths are M-alternating. In doing so, the set of alternating paths does not change.

In what follows, $\mathsf{code}(\pi)$ denotes the code (i.e., a boolean vector) of an alternating path π. We choose a natural way of encoding a path by a sequence of its vertex numbers. Say, the path

$$u_{i_0} \to v_{i_1} \to u_{i_2} \to \ldots \to v_{i_r}$$

is encoded by the string $(i_0, i_1, i_2, \ldots, i_r, 0, \ldots, 0)$. Circuits L_i operate on paths encoded by a string of $2^{i-1}+1$ numbers, and the codes of the paths they compute contain 2^i+1 numbers. The numbers are written in $b = \lceil \log_2(n+1) \rceil$ digits.

Let $\pi_{u,v}$ denote the found alternating path between u and v. Denote by $\delta_{u,v}$ the indicator that some path $\pi_{u,v}$ has been found, and let $\lambda_{u,v}$ be its length.

1.0) The specified quantities are initialized as $\lambda_{u,v} = \delta_{u,v} = t[u,v]$. Initialization of paths of length 1 is performed with complexity $O(n^2 \log n)$:

$$\mathsf{code}(\pi_{u_i,v_j}) = t[u_i, v_j] \cdot (i,\ j), \qquad \mathsf{code}(\pi_{v_j,u_i}) = t[v_j, u_i] \cdot (j,\ i).$$

1.1) On a layer L_i, for each ordered pair of nodes (u,v), the following calculations are performed.

If $\delta_{u,v} = 0$, then for all other nodes $w \in G$, we compute the indicators of the presence and the length of a path from u to v through w:

$$\delta_{u,w,v} = \delta_{u,w}\delta_{w,v}, \qquad \lambda_{u,w,v} = \lambda_{u,w} + \lambda_{w,v}.$$

Some paths $\pi_{u,w} \cup \pi_{w,v}$ among those considered may contain cycles and, thus, not be the shortest. Therefore, among all the paths found, i.e., when $\delta_{u,w,v} = 1$, one should select a path of the minimum length $\lambda_{u,w,v}$ — it does not contain cycles.

The circuit first computes $\lambda_{u,v} = \min_{\delta_{u,w,v}=1} \lambda_{u,w,v}$, setting $\lambda_{u,v} = 0$ if no paths are found. Next, it computes the indicators I_w of the first position w for which $\lambda_{u,v} = \lambda_{u,w,v} \neq 0$. The complexity of these computations for one pair of nodes u, v is trivially $O(n \log n)$. Finally, the desired path may be determined as

$$\mathsf{code}(\pi_{u,v}) = \bigvee_w I_w \cdot \mathsf{code}(\pi_{u,w} \cup \pi_{w,v}). \tag{11.7}$$

The code of the combined path in (11.7) is computed as

$$\mathsf{code}(\pi_{u,w} \cup \pi_{w,v}) = \mathsf{code}(\pi_{u,w}) \vee (\mathsf{code}(\pi_{w,v}) \gg q\lambda_{u,w}),$$

where $x \gg k$ is the operation of shifting a string x by k positions to the right. Shifting a string of length L by a variable value from 0 to $L-1$ positions is performed by a simple circuit of complexity $O(L \log L)$ (Lemma 1.1). The multiplications and disjunction in formula (11.7) have complexity $\preccurlyeq n2^i b$. Therefore,

$$\mathsf{C}(L_i) \preccurlyeq n^3 \log n + n^3 b 2^i i \preccurlyeq 2^i n^3 \log^2 n. \tag{11.8}$$

1.2) At the output of the last subcircuit L_s we obtain descriptions of the shortest alternating paths $\pi_{u,v}$, characterized by attributes $\delta_{u,v}$, $\lambda_{u,v}$, $\mathsf{code}(\pi_{u,v})$.

We distinguish paths connecting free vertices from U and V by indicators $I_{i,j} = \delta_{u_i,v_j} \cdot \overline{p_i^u} \cdot \overline{p_j^v}$. Next, we mark by indicators $J_{i,j}$ the position of just one augmenting

path π, if such a path is found. These calculations are trivially performed[14] with complexity $O(n^2)$. The path itself is computed as

$$\mathsf{code}(\pi) = \bigvee_{i,j} J_{i,j} \cdot \mathsf{code}(\pi_{u_i,v_j})$$

with complexity $\preccurlyeq n^3 \log n$.

1.3) Finally, the current matching should be updated: $M_k \to M_{k+1}$. The attributes $q_{i,j}$ of the presence of (undirected) edges (u_i, v_j) in the path π are trivially computed from its code[15)] with complexity $\preccurlyeq n^3$. Now we set $p_{i,j} := p_{i,j} \oplus q_{i,j}$. The recomputation of the remaining attributes p_i^u, p_j^v, $t[u,v]$ is then performed with a total complexity of $O(n^2)$. By (11.8), we obtain

$$\mathsf{C}(S_k) = \sum_{i=1}^{s} \mathsf{C}(L_i) + O(n^3 \log n) \preccurlyeq l_k n^3 \log^2 n. \tag{11.9}$$

2) In the case where G has a perfect matching, M_{n+1} is such. Therefore, $PERM_n = \bigwedge_i p_i^u$. Finally, in view of $\sum_{k=1}^{n} l_k \asymp n\left(1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}\right) \sim n \ln n$, (11.9) implies

$$\mathsf{C}(PERM_n) \leqslant \mathsf{C}(S_1) + \ldots + \mathsf{C}(S_n) + O(n) \preccurlyeq n^4 \log^3 n.$$

■

• In Theorem 11.4 we deliberately sacrificed complexity for the sake of simplicity of the circuit design. The best known upper bounds on the circuit complexity of the permanent were obtained by transferring algorithms for finding maximum matchings in graphs to deterministic Turing machines and then to circuits. The best known running time $O(n^{2.5}/\log n)$ among deterministic algorithms (also in the model of Turing machines) provides the algorithm of T. Feder and R. Motwani [85][16)]. It is known that a program with execution time T on a Turing machine can be simulated by a boolean circuit of complexity $O(T \log T)$, see, e.g., [331]. From this we can conclude that $\mathsf{C}(PERM_n) \preccurlyeq n^{2.5}$. The corresponding circuit, however, may have a very intricate structure.

It is noteworthy that as A. A. Razborov [256] showed, $\mathsf{C}_{\mathcal{B}_M}(PERM_n) = n^{\Omega(\log n)}$, i.e., the permanent is an example of a (monotone) boolean function of polynomial complexity in a complete basis and superpolynomial — in the monotone basis. However, É. Tardos [317] proposed an analogous example of a function with almost exponential monotone complexity $2^{n^{\Theta(1)}}$.

Monotone complexity of the cyclic walk polynomial ∵ s

Like the previous section, this section demonstrates the connection (this time, not too trivial) between combinatorial properties of graphs and methods for fast computing graph-related polynomials.

[14] Prefix circuits can be employed here.

[15] We can start by computing the indicators $\chi_{i,j}$ of the equality of the i-th component of $\mathsf{code}(\pi)$ to the number j.

[16] The result [85] slightly improves the complexity estimate $O(n^{2.5})$, which is provided by the well-known Dinitz—Karzanov [79, 150] and Hopcroft—Karp [134] algorithms.

Let us associate with an arbitrary edge $e = (i, j)$ of the complete undirected graph K_n on n vertices a variable, for which we will use the notation $x_e = x_{i,j} = x_{j,i}$. Consider the polynomial

$$CW_{k,n}(X) = \sum_{i_1 \neq i_2 \neq \ldots \neq i_k \neq i_1} x_{i_1,i_2} \cdot x_{i_2,i_3} \cdot \ldots \cdot x_{i_{k-1},i_k} \cdot x_{i_k,i_1},$$

whose monomials correspond to all possible cyclic walks of length k in the graph K_n (the definition allows multiple traversals along the same edges).

The complexity of computing polynomials of this type by monotone arithmetic circuits turns out to be related to the treewidth of graphs.

A *tree decomposition* of a graph G on k vertices numbered from 0 to $k-1$ is a tree T, each vertex u of which is associated with a certain set $V_u \subset [\![k]\!]$ so that the following conditions are met:

0) $\bigcup_{u \in T} V_u = [\![k]\!]$;

1) for any edge $(i, j) \in G$ there is a vertex $u \in T$, satisfying the property $i, j \in V_u$;

2) for any $i \in [\![k]\!]$ the set of vertices $\{u \in T \mid i \in V_u\}$ is a connected subtree.

The number $\max_{u \in T} |V_u| - 1$ is called the *width* of a decomposition T. The *treewidth* $\mathrm{tw}(G)$ of a graph G is the minimum width of its decomposition.

Let us consider a more general problem of computing a polynomial whose monomials correspond to images of a k-vertex graph G in the complete graph K_n^* on n vertices $[\![n]\!]$ with allowed loops under the action of all possible mappings $\varphi : [\![k]\!] \to [\![n]\!]$. The image of a vertex i is the vertex $\varphi(i)$. The image of an edge $e = (i,\ j)$ is the edge $\varphi(e) = (\varphi(i),\ \varphi(j))$, which can be a loop for $\varphi(i) = \varphi(j)$. The images of vertices and edges define the image $\varphi(G)$ of the graph G.

Let $\chi_G = \prod_{e \in G} x_e$ denote the characteristic function of a graph — the product of variables corresponding to its edges. As usual, we set $\chi_G = 1$ if the graph has no edges.

For an arbitrary graph G on vertices $[\![k]\!]$ its *homomorphism polynomial* is defined as

$$\mathrm{Hom}_{G,n}(X) = \sum_{\varphi : [\![k]\!] \to [\![n]\!]} \chi_{\varphi(G)}.$$

The cyclic walk polynomial defined above is a special case of the homomorphism polynomial when G is a cycle of length k (denoted by $\mathcal{C}_k$), i.e., $CW_{k,n} = \mathrm{Hom}_{\mathcal{C}_k,n}|_{x_{0,0} = \ldots = x_{n-1,n-1} = 0}$ (loops are not allowed in walks).

When computing a polynomial, it is more convenient to use a special type of tree decomposition.

A *nice tree decomposition* of a graph G is a tree decomposition that is a binary tree oriented towards the root and contains vertices of only the following types:

a) a leaf u with no incoming edges, satisfying $|V_u| = 1$;

b) an *attaching* node u with one input (from a vertex z), for which $V_z \subset V_u$ and $|V_u| = |V_z| + 1$.

c) an *excluding* node u with one input (from a vertex z), for which $V_u \subset V_z$ and $|V_u| = |V_z| - 1$.

d) a *uniting* node u with two inputs (from vertices z_1, z_2), where $V_u = V_{z_1} = V_{z_2}$.

The root r of a nice decomposition is assumed to be an excluding node and has label $V_r = \varnothing$.

It can be verified that any tree decomposition can be transformed into a nice tree decomposition of the same width and with $O(1)$ times more nodes, see also [164]. The following result is actually contained in [78].

Theorem 11.5. *Let a graph G on k vertices have a nice tree decomposition T of width $\mathrm{tw}(G)$ with t vertices. Then $\mathsf{C}_{\mathcal{A}^R_+}(\mathrm{Hom}_{G,n}) \preccurlyeq tk^2 n^{\mathrm{tw}(G)+1}$.*

▶ Distribute the edges of the graph G between suitable vertices of the tree T without repetitions, which means: an edge (i, j) must be assigned to some vertex $u \in T$ for which $i, j \in V_u$. By E_u we denote the set of edges assigned to a vertex $u \in T$.

Let T_u be a subtree of the tree T rooted at u. Set $W_u = \bigcup_{z \in T_u} V_z$. By G_u we denote the subgraph of the graph G formed by the vertices W_u and edges assigned to the vertices of the subtree T_u.

Further, by $\{\varphi, \psi\}$ we denote the union of mappings φ, ψ with disjoint domains.

Sequentially scanning the tree T from the leaves to the root, at each vertex $u \in T$ we compute the family of polynomials

$$P_{u,\psi}(X) = \sum_{\varphi : W_u \setminus V_u \to [\![n]\!]} \chi_{\{\varphi,\psi\}(G_u)} \tag{11.10}$$

for all $\psi : V_u \to [\![n]\!]$, essentially via the dynamic programming method.

a) If u is a leaf of the tree, and $V_u = \{i\}$, then $P_{u,\, i \to j} = 1$ for all $j \in [\![n]\!]$.

b) If u is an attaching vertex with z being its preceding vertex, and $V_u = \{i\} \cup V_z$, then for each $\psi : W_z \to [\![n]\!]$ and $\psi_j = \{i \to j, \psi\} : W_u \to [\![n]\!]$ we set

$$P_{u,\psi_j} = P_{z,\psi} \cdot \prod_{e \in E_u} x_{\psi_j(e)}.$$

It is easy to see that definition (11.10) is satisfied in this case: it suffices to note that the set of mappings φ over which the summation is performed coincides for P_{u,ψ_j} and $P_{z,\psi}$. Moreover, $i \notin W_z$ by property 2) of tree decomposition, hence, the mapping $\{\varphi, \psi_j\}$ is well defined.

c) If u is an excluding vertex with z being its preceding vertex, and $V_z = \{i\} \cup V_u$, then for each $\psi : W_u \to [\![n]\!]$ we set

$$P_{u,\psi} = \sum_{j=0}^{n-1} P_{z,\{i \to j,\, \psi\}} \cdot \prod_{e \in E_u} x_{\psi(e)}.$$

In this case (11.10) is satisfied directly.

d) Let u be a uniting vertex, and z_1, z_2 be its preceding nodes. Recall that $V_u = V_{z_1} = V_{z_2}$ and $W_{z_1}, W_{z_2} \subset W_u$. Then for each $\psi : V_u \to [\![n]\!]$ we set

$$P_{u,\psi} = P_{z_1,\psi} \cdot P_{z_2,\psi} \cdot \prod_{e \in E_u} x_{\psi(e)}.$$

To justify this formula we again apply the connectivity property 2) of tree decomposition, which yields $(W_{z_1} \setminus V_{z_1}) \cap (W_{z_2} \setminus V_{z_2}) = \varnothing$. Denote $Q_i = W_{z_i} \setminus V_{z_i}$. Then

$$P_{z_1,\psi} \cdot P_{z_2,\psi} = \sum_{\varphi_1 : Q_1 \to [\![n]\!]} \chi_{\{\varphi_1,\psi\}(G_{z_1})} \cdot \sum_{\varphi_2 : Q_2 \to [\![n]\!]} \chi_{\{\varphi_2,\psi\}(G_{z_2})} =$$

$$= \sum_{\varphi_1,\varphi_2} \chi_{\{\varphi_1,\varphi_2,\psi\}(G_{z_1} \cup G_{z_2})} = \sum_{\varphi : W_u \setminus V_u \to [\![n]\!]} \chi_{\{\varphi,\psi\}(G_{z_1} \cup G_{z_2})}.$$

e) At the root of the tree, the polynomial $P_{r,\varnothing \to [\![n]\!]} = \mathrm{Hom}_{G,n}$ is computed.

By construction, no more than $O(k^2 n^{\mathrm{tw}(G)+1})$ operations are performed in the computations at each node. ■

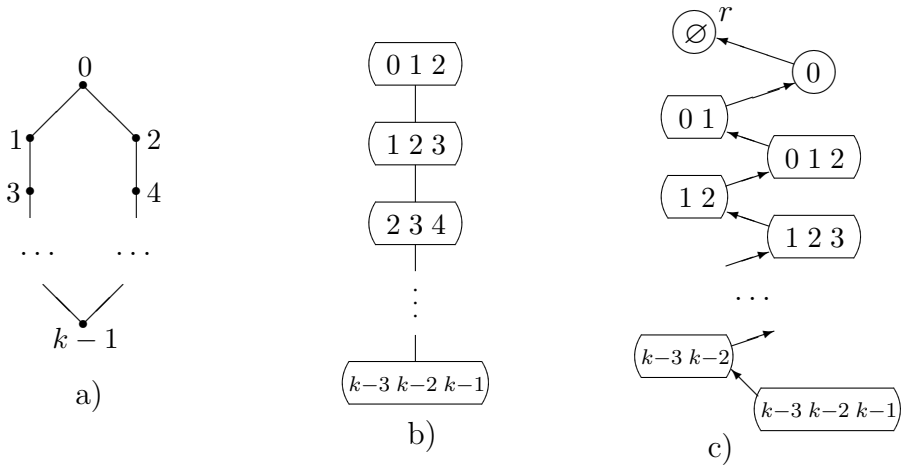


Figure 11.1: Graph $\mathcal{C}_k$ (a), its tree decomposition (b) and its nice tree decomposition (c)

The treewidth of the cycle $\mathcal{C}_k$ for $k \geqslant 3$ is[17] 2. A minimal treewidth decomposition and a nice decomposition of $\mathcal{C}_k$ are shown in Fig. 11.1. They contain $O(k)$ vertices. Setting $x_{i,i} = 0$ for all $i \in [\![n]\!]$, i.e., excluding loops, we obtain

Corollary 11.1. *If $k \geqslant 3$, then $\mathsf{C}_{\mathcal{A}^R_+}(CW_{k,n}) \preccurlyeq (kn)^3$.*

• In the definition of the homomorphism polynomial, loops are generally prohibited. To eliminate them, it is sufficient to set the variables associated with loops to zero.

In [171] it is shown that the bound of Theorem 11.5 is best possible in order for constant k. Tropical $(\min, +)$-analogs of homomorphism polynomials are degenerate for many G, in particular, for bipartite graphs, which include cycles of even length. For odd k, the tropical version of the polynomial $CW_{n,k}$ is informative. For $k = 3$, it coincides with the polynomial of finding the minimum weight triangle and has complexity $\Theta(n^3)$, for instance, due to C. Schnorr's lower bound from [275]. Following the method of M. Jerrum and M. Snir [135] the lower complexity bound $\Omega(n^3)$ may be proved for any constant odd k. Thus, the result of Corollary 11.1 in this case is tight up to an order.

[17] It is easy to verify that among connected graphs only trees have width 1.

Chapter 12

Miscellaneous

Orthogonal systems method. Multiplicative complexity of boolean functions $\boxed{/_2}$

The method of decomposition in orthogonal systems of functions (in boolean algebra) was proposed by E. I. Nechiporuk in [225] and successfully applied to several problems (minimization of the number of switches in swithing-rectifier circuits, minimization of the number of negations in circuits over the basis $\{\vee, {}^{-}\}$). To illustrate the idea, we consider the problem of minimizing the number of multiplications when computing boolean functions of n variables by circuits over the Zhegalkin basis $\mathcal{B}_1 = \{\oplus, \wedge, 1\}$. We denote the corresponding complexity functional by C^{μ}.

Theorem 12.1 ([225]). $\mathsf{C}^{\mu}(\mathcal{P}^n) \lesssim (2 + o(1))2^{n/2}$.

▶ If n is even, we divide the set of variables X into two groups X_1, X_2 equally. Represent a function $f \in \mathcal{P}^n$ as a Zhegalkin polynomial in the first group of variables:

$$f(X) = \bigoplus_{\sigma \in \mathbb{B}^{n/2}} X_1^{\sigma} \cdot f_{\sigma}(X_2), \qquad X_1^{\sigma} = \prod_{\sigma_i = 1} x_i.$$

First, compute all subfunctions f_σ as all possible linear combinations of monomials of the variables X_2. The multiplicative complexity of computing all monomials in $n/2$ variables is at most[1)] $2^{n/2}$ by Lemma 3.1. Then, following Horner's scheme with respect to variables X_1, compute f as

$$f(X) = x_1 f_1 \oplus f_0 = x_1(x_2 f_{11} \oplus f_{10}) \oplus (x_2 f_{01} \oplus f_{00}) = \ldots \tag{12.1}$$

using another $2^{n/2} - 1$ multiplications.

In the case of odd n, the above method requires approximately $2^{\lceil n/2 \rceil} + 2^{\lfloor n/2 \rfloor} \approx 2.12 \cdot 2^{n/2}$ multiplications. However, the method of decomposition in orthogonal systems allows us to obtain the same form of asymptotics as for even n.

Divide the set of variables X into three groups X_1, X_2, Y, where $|X_1| = |X_2| = m$. On the set of monomials of variables Y, we introduce a numbering with two indices, $Y^{i,j}$, $0 \leqslant i, j < 2^{|Y|/2} + 1$.

[1)] In fact, it is simply $2^{n/2} - n/2 - 1$.

Set $h_i(Y) = \bigvee_j Y^{i,j}$ and $g_j(Y) = \bigvee_i Y^{i,j}$. The introduced set of functions allows us to express any monomial as $Y^{i,j} = h_i(Y)g_j(Y)$ and satisfies the orthogonality conditions $h_i(Y)h_{i'}(Y) = 0$ for $i \neq i'$, and $g_j(Y)g_{j'}(Y) = 0$ for $j \neq j'$. Then

$$f(X) = \bigoplus_{\sigma,\tau\in\mathbb{B}^m} X_1^\sigma \cdot X_2^\tau \cdot f_{\sigma,\tau}(Y) = \bigoplus_i h_i(Y) \cdot \left[\bigoplus_\sigma X_1^\sigma \cdot \bigoplus_{\tau,j:\, Y^{i,j}\in f_{\sigma,\tau}(Y)} X_2^\tau \cdot g_j(Y) \right], \tag{12.2}$$

where $Y^{i,j} \in f_{\sigma,\tau}(Y)$ means that the monomial $Y^{i,j}$ appears in the Zhegalkin polynomial of the function $f_{\sigma,\tau}(Y)$.

Let us estimate the complexity of computations by formula (12.2). To compute all monomials of the variables X_2 and Y and, as a consequence, the functions h_i and g_j, $2^m + 2^{|Y|}$ multiplications are sufficient. Another $2^m(2^{|Y|/2}+1)$ multiplications are required to compute all possible products $X_2^\tau g_j(Y)$. Further, the computation of each of the sums in square brackets (12.2) is performed in the spirit of (12.1) in $2^m - 1$ multiplications, i.e., using $(2^{|Y|/2}+1)(2^m-1)$ multiplications for all sums. It remains to perform $2^{|Y|/2}+1$ multiplications by $h_i(Y)$. The required bound is achieved, say, for $m \sim |Y| \sim n/3$. ■

Applying a much more complicated synthesis method — a combination of multilevel and triangular representations of boolean functions, as well as a number of other ideas, — Nechiporuk [225] established an asymptotically tight result[2] $\mathsf{C}^\mu(\mathcal{P}^n) \sim 2^{n/2}$ (a detailed proof is published in [227]).

• In view of the extreme hardness of the asymptotically optimal Nechiporuk method, it is of interest to search for elementary and at the same time sufficiently economical methods of synthesis. One of them is presented in Theorem 12.1. A faster method was proposed by S. N. Selezneva in [285]. It is based on the representation

$$f(X_1, X_2, X_3, x) = \bigoplus_{\sigma\in\mathbb{B}^m} X_1^\sigma \cdot \bigoplus_{\tau\in\mathbb{B}^p} X_2^\tau (x \cdot f_{\sigma,\tau,1}(X_3) \vee \overline{x} \cdot f_{\sigma,\tau,0}(X_3)) =$$
$$= \bigoplus_{\sigma\in\mathbb{B}^m} X_1^\sigma \cdot \bigoplus_{\tau\in\mathbb{B}^p} \left((xX_2^\tau \oplus f_{\sigma,\tau,0}(X_3))(\overline{x}X_2^\tau \oplus f_{\sigma,\tau,1}(X_3)) \oplus g_{\sigma,\tau}(X_3) \right),$$

where $|X_1| = m$, $|X_2| = p$, and $g_{\sigma,\tau} = f_{\sigma,\tau,0} \cdot f_{\sigma,\tau,1}$. For a suitable choice of parameters m, p and odd n, the method has complexity $\sim \sqrt{2} \cdot 2^{n/2}$. In the case of even n, the estimate from [285] is $\sim (3/2) \cdot 2^{n/2}$, but an additional decomposition in orthogonal systems allows us to obtain the same asymptotic complexity as for odd n.

Tree balancing method. Depth of boolean functions $\boxed{U}$

Recall that formulae over finite bases, as a rule, admit parallel reconstruction: the existence of a formula of complexity L implies the existence of a formula of depth $\asymp \log L$ for the same function (see Theorems 4.4 and 4.5). But it turns out that a wide class of formulae admits almost ideal parallelization: say, with depth close to $\log_2 L$, in the case of binary bases. This property is possessed by formulae in which operations rarely alternate when moving from inputs to outputs.

[2] The lower bound is proved elementarily by the cardinality argument.

Sergey Andreevich Lozhkin
Moscow University, since 1978

The *alternation depth* of a path in a circuit or formula is the number of chains of identical multi-input (two- or more) operations into which it is decomposed[3]. The alternation depth of a circuit or formula is the maximum alternation depth of a path from input to output.

S. A. Lozhkin [191] observed that formulae of small alternation depth are parallelized almost perfectly.

Theorem 12.2 ([191]). *If a function f is implemented by a formula over the basis $\mathcal{B}_0$ of complexity L and alternation depth h, then $\mathsf{D}_{\mathcal{B}_0}(f) \leqslant \lceil \log_2 L \rceil + h - 1$.*

▶ Taking into account the possibility of lowering negations to the level of inputs, the alternation depth of a formula over $\mathcal{B}_0$ is the maximum number of series of conjunctions or disjunctions in an input-output path.

We will argue by induction on h. For $h = 1$ the statement is obvious. Consider the induction step from $h-1$ to h.

Let a formula F have complexity L and alternation depth h. We select in it the maximal external subformula G consisting of identical operations (gates); let $F_1, \ldots, F_s$ be subformulas attached to the inputs of G. Thus, $F = F_1 \circ F_2 \circ \ldots \circ F_s$ up to the placement of brackets, where $\circ \in \{\vee, \wedge\}$ is an operation defining the subformula G. By construction, $\sum \Phi(F_i) = L$, and the alternation depth of formulae F_i does not exceed $h-1$.

The induction hypothesis allows us to replace each formula F_i by an equivalent formula F_i' of depth $d_i \leqslant \lceil \log_2 \Phi(F_i) \rceil + h - 2$. It remains to note (this is the key point of the proof) that the $\circ$-sum $F_1' \circ F_2' \circ \ldots \circ F_s'$ can be computed by a balanced tree with root vertex depth

$$d = \left\lceil \log_2 \sum_{i=1}^{s} 2^{d_i} \right\rceil \leqslant \left\lceil \log_2 \sum_{i=1}^{s} 2^{h-1} \Phi(F_i) \right\rceil \leqslant \lceil \log_2 L \rceil + h - 1.$$

■

A similar statement is true in any basis in which binary operations are associative and commutative, for example, in $\mathcal{B}_1$ or $\mathcal{A}_+$.

Corollary 12.1 ([191]). $\mathsf{D}_{\mathcal{B}_0}(\mathcal{P}_n) \leqslant n - \log_2 \log n + O(1)$.

▷ By analyzing the construction of the formula from Theorem 11.2, it is easy to verify (taking into account lowering of negations) that the formula has a constant alternation depth. □

• Based on O. B. Lupanov's construction [204] of formulae of asymptotically optimal complexity and alternation depth 3, Lozhkin in [191] obtained a bound of Corollary 12.1 in the form $\mathsf{D}_{\mathcal{B}_0}(\mathcal{P}_n) \leqslant \lceil n - \log_2 \log_2 n + o(1) \rceil + 3$. This is only slightly weaker than his own record bound (11.5).

[3] Chains of unary operations are not counted.

Independently of Lozhkin, H. Hoover, M. Klawe and N. Pippenger [133] proposed a method for minimizing the depth of formulae when restricting the fan-out of elements. This method turns out to be dual to Lozhkin's method (essentially, the same transformation is applied to a reversed formula), which is explained in detail by S. B. Gashkov in [101].

Gradient method. Depth of circuits for multiple addition $\boxed{\nabla}\boxed{U}$

The exposition would be incomplete without mentioning the gradient method, which plays a significant role in discrete optimization problems. Unfortunately, the chosen model (circuits and formulae of unbounded depth) does not allow us to adequately illustrate the diversity of variations and applications of the method. So, we restrict ourselves to a single example, another example will be considered below for the model of bounded-depth circuits, see p. 143.

Let us return to the problem of minimizing the depth of a circuit of compressors in the multiple addition problem. The method of Theorem 4.3 is optimal in the asymptotic sense, but the additional constant $O(1)$ hidden in the depth estimate is very large (the circuit includes several times more compressors than minimally required).

But what if we simply construct a circuit, sequentially connecting compressors each time to the minimum possible depth (this is the gradient method)? Will the constructed circuit be efficient? At least in the case of the most popular compressor FA_3 the answer is affirmative, as shown by the author in [287]. Recall that the compressor FA_3 is characterized by input depths $(0,0,1)$ and output depths $(2,3)$, and its characteristic polynomial (4.10) has a root $\lambda \approx 1.2056$. An example of a gradient circuit for summing eight numbers is shown in Fig. 12.1 (compare with the circuit in Fig. 4.1).

Theorem 12.3. $D_{FA_3}(n) \leqslant \log_\lambda n + 4.6$.

▶ Consider a gradient compressor circuit S on n inputs, denote its depth by D. We consider a compressor to be located at level d if its outputs have depths $d{+}2$ and $d{+}3$. For an arbitrary set with repetitions $T \subset \mathbb{N}_0$, we denote $\sigma(T) = \sum_{t\in T} \lambda^t$ (the sum of potentials). Let S_r denote the circuit constituted by compressors of the circuit S located at levels less than r. By $T(S_r)$ we denote the set (with repetitions) of depths of outputs of a circuit S_r, in which numbers less than r are replaced by r (the set contains only numbers r, $r+1$, and $r+2$). For brevity, we set $\sigma(S_r) = \sigma(T(S_r))$. Due to the choice of λ,

$$n = \sigma(S_0) \leqslant \sigma(S_1) \leqslant \ldots \leqslant \sigma(S_{D-2}) = \lambda^D + \lambda^{D-1}. \tag{12.3}$$

If all compressors could be connected without gaps, then the chain (12.3) would consist entirely of equalities, but this is, of course, impossible. Already at the first step, a third of the inputs have to be "lowered" to the depth 1. The more such gaps occur at subsequent depths, the greater the magnitude of differences $\sigma(S_{r+1}) - \sigma(S_r)$, and the further the sum of the potentials of the intermediate sums deviates from n. Fortunately, it turns out that level 0 of the circuit S is special: at each of the subsequent levels, no more than two intermediate sums are lowered.

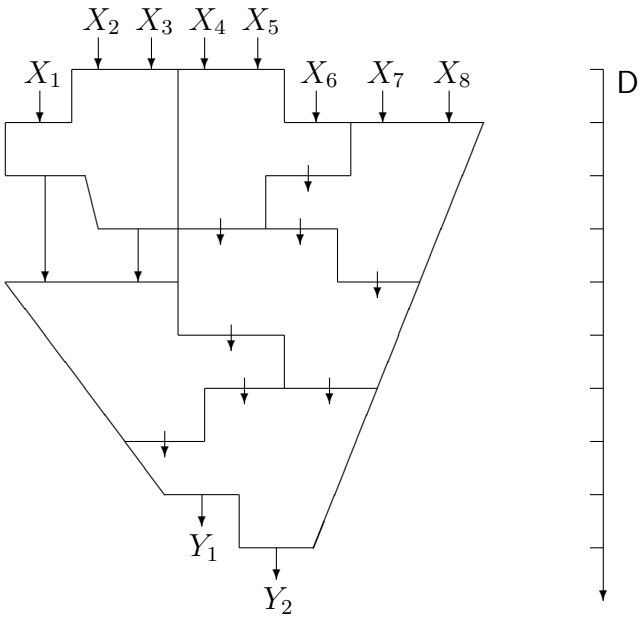


Figure 12.1: Gradient circuit of $(3,2)$-compressors

Lemma 12.1 ([287]). *Let $r > 0$, and $m_0(r)$, $m_1(r)$, $m_2(r)$ be the number of occurences of r, $r+1$, $r+2$ in the set $T(S_r)$, respectively. Then*

$$m_0(r) \leqslant 2m_1(r) + 2, \quad m_1(r) \leqslant 3m_0(r)/2 + 2m_2(r), \quad m_2(r) \leqslant m_1(r). \tag{12.4}$$

$\triangleright$ We argue by induction on r. The circuit contains $k = \lfloor n/3 \rfloor$ compressors of level 0, so

$$m_0(1) = n \bmod 3, \quad m_1(1) = m_2(1) = k,$$

and the inequalities (12.4) hold for $r = 1$. To prove the induction step from r to $r+1$, consider the two cases. Further, m_i denotes $m_i(r)$.

In the case $m_0 \leqslant 2m_1$, $k = \lfloor m_0/2 \rfloor$ compressors are placed at level r, so

$$m_0(r+1) = m_1 - k + (m_0 \bmod 2), \quad m_1(r+1) = m_2 + k, \quad m_2(r+1) = k.$$

In the case $m_0 \geqslant 2m_1$, at level r there are $k = \lfloor (m_0 + m_1)/3 \rfloor$ compressors, hence,

$$m_0(r+1) = (m_0 + m_1) \bmod 3, \quad m_1(r+1) = m_2 + k, \quad m_2(r+1) = k.$$

It is easy to check that in both cases the validity of inequalities (12.4) for $m_i(r+1)$ automatically follows from the validity of these inequalities for m_i. $\square$

To estimate the depth of the circuit S, only the first inequality of Lemma 12.1 is essential. It means that at any level $r \geqslant 1$ the circuit contains $\min\{\lfloor m_0(r)/2 \rfloor, m_1(r)\}$ compressors, hence, $\sigma(S_{r+1}) - \sigma(S_r) \leqslant 2(\lambda^{r+1} - \lambda^r)$. Therefore,

$$\lambda^D + \lambda^{D-1} = \sigma(S_{D-2}) = \sum_{r=1}^{D-3} (\sigma(S_{r+1}) - \sigma(S_r)) + (\sigma(S_1) - \sigma(S_0)) + \sigma(S_0) \leqslant$$
$$2(\lambda^{D-2} - \lambda) + (\lambda - 1)(n/3 + 2) + n < 2\lambda^{D-2} + (\lambda + 2)n/3.$$

Finally, we obtain $\lambda^{D-2} \leqslant \frac{\lambda+2}{3(\lambda^2+\lambda-2)}\, n$, whence $D \leqslant \log_\lambda n + 4.6$. ■

It is informative to compare the indicated upper bound with the lower bound $D_{FA_3}(n) > \log_\lambda n - 3.8$, which follows from Lemma 4.1. Note that the complexity of the gradient circuit is approximately $5mn + O(n)$ matching the complexity of a trivial circuit composed of standard adders. The term $O(n)$ here corresponds to the growth of the length of intermediate sums as the depth increases. The smallness of this term follows from the fact that the number of compressors decreases from level to level in geometric progression, while the length increases linearly.

• Reasoning a little more carefully, the author in [287] obtained an upper bound for the depth of the gradient circuit $D_{FA_3} < \log_\lambda n - 0.8$. Together with the refined lower bound $D_{FA_3}(n) > \log_\lambda n - 2.7$ this means that the gradient method is obviously inferior in depth to a hypothetically optimal method by no more than one, and for many n it is already provably optimal. The question of the efficiency of gradient circuits for other types of compressors has apparently not been studied.

Chapter 13

Bounded-depth circuits

Introduction

Bounded-depth circuits are circuits over an infinite basis that include functions with an unbounded number of arguments[1)]. Such circuits have been intensively studied since about the 1980s, primarily in the direction of obtaining high lower complexity bounds. Already from the pioneering works [319, 3, 94] it became clear that the model allows one to obtain superpolynomial lower bounds for the complexity of specific functions over complete boolean bases[2)].

The most popular models of bounded-depth circuits are *AC-circuits* — circuits over the basis $\{\bigvee, \bigwedge, ^{-}\}$ (unbounded fan-in disjunctions and conjunctions)[3)]. They generalize circuits over the standard basis $\mathcal{B}_0$. It can be assumed that all negations in an AC-circuit are applied only to the inputs of variables and are not taken into account in counting the depth. *$AC[\oplus]$-circuits* are circuits in a wider basis $\{\bigvee, \bigwedge, \bigoplus, ^{-}, 1\}$ — a generalization of circuits over the basis $\mathcal{B}_2$. There are also studied circuits that additionally include elements of modular summation, symmetric, and threshold functions. The complexity of a boolean operator F when implemented by AC-circuits and, respectively, $AC[\oplus]$-circuits of depth d will be denoted by $\mathsf{C}_d^{AC}(F)$ and $\mathsf{C}_d^{AC[\oplus]}(F)$. For the complexity of circuits with alternating layers of operations, for example, $\bigvee\bigwedge\bigvee$-circuits[4)], we use a natural notation like $\mathsf{C}_{\bigvee\bigwedge\bigvee}(F)$.

An unbounded fan-in analogue of arithmetic circuits are circuits over the basis $\{\Sigma, \Pi\}$, where Σ are linear combinations over a ring R, and Π are products of an unbounded number of variables. We introduce a notation for the complexity of computing an operator F by such circuits in the form $\mathsf{C}^R_{\Sigma\Pi\Sigma}(F)$ (an example for depth-3 circuits).

---

[1]Sometimes, bounded-depth circuits also mean circuits over finite bases that have a bounded alternation depth (the number of alternations of an operation in an input-output computation path).

[2]Soon, similar results were obtained for the complexity of bounded-depth arithmetic circuits over finite fields, and more recently in [187] — for circuits over infinite fields.

[3]The name refers to the fact that such circuits appear in defining complexity classes AC^k.

[4]These are monotone circuits of depth 3 with disjunctors on the first and third layers and conjunctors on the middle layer.

The simplest and historically the first[5] type of circuits of unbounded fan-in elements are *linear* (*rectifier*) *circuits*. In such circuits, a single addition operation is used (in some commutative semigroup). Since the number of gates as a measure of complexity of linear circuits does not make sense, the number of edges in the graph of a circuit is counted. The complexity of a linear operator with matrix A when implemented by depth-d linear circuits over the basis $\{+\}$ is denoted by $\mathsf{W}_d^+(A)$. The notation $\mathsf{W}_d(A)$ is used for the complexity of universal circuits that correctly compute a matrix[6] A regardless of the choice of semigroup.

The linear circuit model is equivalent to the additive circuit model: an additive circuit may be transformed into a linear one while maintaining the order of complexity: for an arbitrary $m \times n$ matrix A, we have $\mathsf{W}(A) \asymp \mathsf{L}(A) + m + n$, where the notation $\mathsf{W}(A)$ refers to the complexity of circuits with unrestricted depth. It is the depth restriction that distinguishes the linear circuit model.

In more general boolean and arithmetic models, depth-2 circuits represent a degenerate case corresponding to normal forms (DNF, CNF) or standard notation of a polynomial. But for linear circuits, nontrivial problems start already at depth 2. In the boolean case, computing a matrix by a depth-2 circuit is interpreted as constructing a covering of the matrix by *rectangles* (i.e. all-1 submatrices). The *weight of a rectangle* is the sum of the number of rows and columns, the *weight of a covering* is the sum of the weights of its rectangles. Then the complexity of computing a boolean matrix with depth 2 corresponds to the minimal weight of a covering. Coverings as is correspond to $\bigvee$-circuits, and universal circuits correspond to partitions, i.e., coverings consisting of disjoint rectangles. For more details on the complexity of linear circuits, see [142].

Below we present several results illustrating general synthesis methods and reflecting the features of the model of bounded-depth circuits, and also consider another general method, the cut method.

Depth-2 linear circuits for the Sierpiński matrix. Gradient method /2 ▽

Let us begin with two results on the complexity of computing Sierpiński matrices by depth-2 linear circuits. The first exploits the idea of halving to construct efficient universal circuits, and the second involves a fairly general combinatorial version of the gradient method and produces nearly optimal linear $\bigvee$-circuits.

The sequence of *Sierpiński* (*disjointness*) *matrices* is recursively defined as

$$S_1 = [1], \qquad S_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \qquad S_{2n} = \begin{bmatrix} S_n & 0 \\ S_n & S_n \end{bmatrix}. \tag{13.1}$$

Sierpiński matrices are a popular object in combinatorics and complexity theory, for some of their properties see, e.g., in [142].

The minimal linear circuit is constructed directly by definition (13.1): $\mathsf{W}(S_n) = \mathsf{W}^{\vee}(S_n) = n\log_2(n/2)$. The validity of this estimate was proved by S. N. Se-

[5] Introduced by O. B. Lupanov [200] in 1956.

[6] As in the case of additive circuits, it is convenient to assume that a linear circuit computes the matrix itself.

lezneva [283] and independently by J. Boyar and M. Find [44]. A simple method for constructing depth-2 circuits is proposed in the work of S. Jukna and the author [142].

Theorem 13.1 ([142]). $\mathsf{W}_2(S_n) \preccurlyeq n^{\log_2(\sqrt{2}+1)} \prec n^{1.28}$.

▶ The purpose is to present a matrix decomposition into rectangles of small total weight. Following (13.1), we construct a covering of the matrix S_{2n} from (three identical) coverings of submatrices S_n. The covering will consist of rectangles with the ratio of side lengths $1:1$ (squares) and $1:2$ (bricks). In each triple of such rectangles, we merge two along the long side, see Fig. 13.1[7].

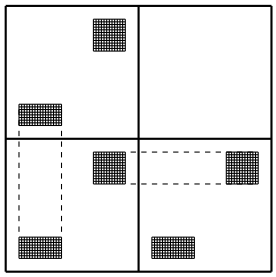


Figure 13.1: Constructing coverings for Sierpiński matrices

Thus, an $u \times u$ square of a covering of S_n generates the same-size square and an $u \times 2u$ brick in the covering of S_{2n}. A $v \times 2v$ brick of a covering of S_n generates the same-size brick and a $2v \times 2v$ square in the covering of S_{2n}. Let u_n and v_n denote the sum of the lengths of the sides of the squares and the sum of the lengths of the short sides of the bricks from the covering of S_n. Starting from $u_2 = v_2 = 1$, for $n = 2^r$ we obtain

$$\begin{bmatrix} u_{2n} \\ v_{2n} \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} u_n \\ v_n \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 1 & 1 \end{bmatrix}^r \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} = P \cdot \begin{bmatrix} 1+\sqrt{2} & 0 \\ 0 & 1-\sqrt{2} \end{bmatrix}^r \cdot P^{-1} \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

where P is some invertible 2×2 matrix, since the matrix $\begin{bmatrix} 1 & 2 \\ 1 & 1 \end{bmatrix}$ has eigenvalues $1 \pm \sqrt{2}$. As a consequence, $u_n + v_n \asymp n^{\log_2(\sqrt{2}+1)}$. ■

• In [9], a very nontrivial generalization of the method of Theorem 13.1 was proposed, which allowed to reduce the complexity bound to $\mathsf{W}_2(S_n) \prec n^{1.26}$; see also [303], where a somewhat more accurate estimate was obtained.

The authors of [63] observed that in the model of depth-2 linear $\bigvee$-circuits, the upper bound of Theorem 13.1 can be improved via the gradient method.

Consider a family $\mathcal{F}$ of subsets of a finite set X. By definition, a family $\mathcal{F}$ contains a covering of size t if $X = \bigcup_{i=1}^{t} F_i$ for some $F_i \in \mathcal{F}$ (i.e., all elements of X

[7] Of course, the rectangles do not necessarily consist of adjacent rows and columns.

are covered by some t sets from $\mathcal{F}$). The following result was principally proved by A. A. Sapozhenko in [270], but is better known as the Lovász—Stein lemma. It is more convenient for us to use the version of S. Stein [311], see also [138].

Lemma 13.1 ([311]). *If $|F| \leqslant s$ for any $F \in \mathcal{F}$, and any element $x \in X$ belongs to at least r sets in $\mathcal{F}$, then the family $\mathcal{F}$ contains a covering of size $t \leqslant (1+\ln s)|\mathcal{F}|/r$.*

▷ Consider the gradient method of constructing a covering. At each successive step, we add to the covering a set $F \in \mathcal{F}$ that covers the maximum number of elements that have not yet been covered.

First, set $C_0 = \varnothing$ and $X_0 = X$. If at i-th step a set F_i is chosen, then we set $C_i = C_{i-1} \cup \{F_i\}$ and $X_i = X_{i-1} \setminus F_i$. The process ends at step t under the condition $X_t = \varnothing$. Then C_t is the desired covering.

Note that $X_j = \bigcup_{i=j+1}^{t}(F_i \cap X_{i-1})$. By assumption, the number of new elements covered at each step does not increase:

$$s \geqslant |F_1 \cap X_0| \geqslant \ldots \geqslant |F_i \cap X_{i-1}| \geqslant \ldots$$

Denote by t_k the number of sets $F_i \cap X_{i-1}$ of cardinality k. Then $t = \sum_{k=1}^{s} t_k$. We introduce the notation $w_k = |X_{j_k}|$, where $j_k = t_s + t_{s-1} + \ldots + t_k$ and formally $j_{s+1} = 0$. Since none of the sets $F \in \mathcal{F}$ contains more than $k-1$ elements from X_{j_k},

$$rw_k \leqslant (k-1)|\mathcal{F}|. \tag{13.2}$$

Since $t_k = (w_{k+1} - w_k)/k$, applying (13.2), we finally obtain

$$t = \sum_{k=1}^{s} t_k = \sum_{k=1}^{s} \frac{w_{k+1} - w_k}{k} = \frac{w_{s+1}}{s} + \sum_{k=2}^{s} \frac{w_k}{(k-1)k} - w_1 \leqslant \frac{|\mathcal{F}|}{r}\left(1 + \frac{1}{2} + \ldots + \frac{1}{s}\right) \leqslant \frac{|\mathcal{F}|}{r}(1+\ln s).$$

The last transition relies on the well-known relation between harmonic numbers and natural logarithms[8]. □

The lemma shows that the gradient method guarantees a result not too much weaker than an optimistic estimate $|\mathcal{F}|/r$.

Further, we prefer to use the definition of the Sierpiński matrix as a disjointness matrix. The rows and columns of the $n \times n$ matrix S_n, $n = 2^k$, are indexed by all possible subsets of the set X of k elements. Set $S_n[A, B] = (A \cap B = \varnothing)$. It is easy to verify that this definition is equivalent to (13.1).

Theorem 13.2 ([63]). $\mathsf{W}_2^{\vee}(S_n) \preccurlyeq n^{\log_2(9/4)} \log^{7/2} n \prec n^{1.17}$.

► Split the matrix S_n into $(k+1)^2$ submatrices $S_n^{a,b}$, $0 \leqslant a, b \leqslant k$, where $S_n^{a,b}$ is formed by rows labeled by sets of cardinality a, and columns labeled by sets of cardinality b.

[8] Namely, $1 + \frac{1}{2} + \ldots + \frac{1}{s} \leqslant \ln s + \gamma + \frac{1}{2s}$, where $\gamma = 0.577\ldots$ is Euler's constant.

We construct coverings of matrices $S_n^{a,b}$ independently. Next, we assume $a+b \leqslant k$, since in other cases the submatrices are entirely zero.

Consider the family $\mathcal{F} = \{F_R\}$ of rectangles in $S_n^{a,b}$ formed by rows $A \subset R$ and columns $B \subset \overline{R}$, where $|R| = (a-b+k)/2$ and $\overline{R} = X \setminus R$.

Any rectangle F_R has size $C^a_{(a-b+k)/2} \times C^b_{(b-a+k)/2}$, and each element of the matrix $S_n^{a,b}$ is covered by $r = C^{(k-a-b)/2}_{k-a-b}$ rectangles from $\mathcal{F}$. Moreover, $|\mathcal{F}| = C_k^{(a-b+k)/2}$. Then, according to Lemma 13.1,

$$t \preccurlyeq \ln\left(C^a_{(a-b+k)/2} \cdot C^b_{(b-a+k)/2}\right) \cdot |\mathcal{F}|/r \preccurlyeq k^{3/2} \cdot C_k^{(a-b+k)/2} \cdot 2^{a+b-k}$$

rectangles form a covering of the matrix $S_n^{a,b}$. Thus, for $a \geqslant b$,

$$\mathsf{W}_2^\vee(S_n^{a,b}) \preccurlyeq k^{3/2} \cdot C_k^{(a-b+k)/2} \cdot 2^{a+b-k} \cdot C^a_{(a-b+k)/2} = k^{3/2} \cdot 2^{-2u} \cdot C_k^u \cdot C^a_{k-u}, \qquad (13.3)$$

where $u = (k-a-b)/2$.

Let us remind a well-known relation $C_n^m \leqslant 2^{nH(m/n)}$, where $H(x) = -x\log_2 x - (1-x)\log_2(1-x)$ is the *binary entropy function*, defined on the segment $x \in [0,\,1]$[9]. After introducing the notation $\alpha = u/k$, estimate (13.3) continues as

$$\mathsf{W}_2^\vee(S_n^{a,b}) \preccurlyeq k^{3/2} \cdot 2^{(H(\alpha)-2\alpha)k} \cdot C_{k-u}^{(k-u)/2} \preccurlyeq k^{3/2} \cdot 2^{(1+H(\alpha)-3\alpha)k}.$$

It is easy to check that the function $H(x) - 3x$ takes the maximum value $\log_2(9/8)$ at $x = 1/9$. ■

• The upper bound of Theorem 13.2 is extremely close to the lower bound $\mathsf{W}_2^\vee(S_n) \succ n^{1.16}$ from [142]. Moreover, the methods of proof of both bounds allow to establish the complexity as $\mathsf{W}_2^\vee(S_n) \approx n^\alpha$ up to a factor of $(\log n)^{O(1)}$, where the exponent α is defined as the solution to the problem of finding the extremum of a certain function [63].

$\oplus\wedge\oplus$-circuits for boolean functions ∵ c

The path to fast computation often lies through constructing economical coverings (for example, of a boolean cube). But in the model of bounded-depth circuits, due to its inherent parallelism, this happens much more often. As an example, consider the problem of synthesizing $\oplus\wedge\oplus$-circuits solved by S. N. Selezneva [286] in the sense of determining the order of complexity of the class of boolean functions of n variables. The lower bound $\Omega(2^n/n^2)$ is established by a simple cardinality argument [284]. The proof of the upper bound involves the construction of a covering of a boolean cube by fragments of radius-2 spheres.

Recall that a unit sphere centered at α in a boolean cube is the set of vectors that differ from α exactly at one position. Those of the vectors that are obtained by replacing 0 with 1 form a *positive hemisphere*, the rest form a *negative*. As usual, a family T of subsets $Q \subset \mathbb{B}^n$ is called a *covering* of a (sub)set $S \subset \mathbb{B}^n$ if $S \subset \bigcup_{Q\in T} Q$.

J. Cooper, R. Ellis, and A. Kahng [71] constructed coverings of a boolean cube by hemispheres of order-optimal cardinality. Let $\mathbb{B}^n_+$ (respectively $\mathbb{B}^n_-$) denote the boolean cube $\mathbb{B}^n$ excluding all-0 (all-1) vector.

[9] At the ends of the segment, by continuity, $H(0) = H(1) = 0$.

Lemma 13.2 ([71]). *There exists a covering T_n of the boolean cube $\mathbb{B}^n_+$ by positive hemispheres of cardinality $|T_n| \asymp 2^n/n$.*

▷ First, we establish the existence of a partial covering $T \subset \mathbb{B}^n$ of the boolean cube with slightly worse characteristics. Denote by $\overline{T}$ the set of points not covered by a partial covering T. It is convenient to introduce a special measure, the *δ-size* of a partial covering: $|T|_\delta = |T| + \delta|\overline{T}|$ for an arbitrary $\delta \geqslant 1/n$.

I. Let us prove the existence of a partial covering T for which

$$|T|_\delta \leqslant (2\ln(\delta n) + 1)2^n/n. \tag{13.4}$$

We define a random set of points T_0 by the condition: the probability that a point of the j-th layer of the cube belongs to T_0 is $p_j = \min\{\ln(\delta n)/(j+1), 1\}$. Consider a covering T consisting of hemispheres with centers in T_0.

By construction, all points of the $\ln(\delta n)$ lower layers of the cube, except for zero point, are covered by the covering T. For $j+1 > \ln(\delta n)$, the probability that a point of the j-th layer of the cube does not belong to any of the hemispheres is estimated as $(1-p_{j-1})^j \leqslant 1/(\delta n)$ due to the inequality $(1-x)^{1/x} \leqslant 1/e$ valid for $x < 1$. Then for the mathematical expectation of the δ-size of the covering, we have

$$\mathbf{E}[|T|_\delta] = \mathbf{E}[|T|] + \delta\,\mathbf{E}[|\overline{T}|] \leqslant \sum_{j=0}^{n} p_j C_n^j + \delta\, 2^n \frac{1}{\delta n} \leqslant$$

$$\frac{\ln(\delta n)}{n+1} \cdot \sum_{j=0}^{n} C_{n+1}^{j+1} + \frac{2^n}{n} \leqslant (2\ln(\delta n) + 1)\frac{2^n}{n}.$$

II. Note that if T_1 is a covering of the cube $\mathbb{B}^{n_1}_+$, and T_2 is a partial covering of the cube $\mathbb{B}^{n_2}_+$, then the set $T_1 \triangle T_2 := (\mathbb{B}^{n_1} \times T_2) \cup (T_1 \times \overline{T_2})$ is a covering of the cube $\mathbb{B}^{n_1+n_2}_+$.

Guided by this rule, we construct inductively a covering of the cube $\mathbb{B}^n_+$ of cardinality $\leqslant b2^n/n$, where $b \geqslant 1$ is a suitable constant. The induction base $n = 1$ is trivially satisfied. We prove the induction step from $n-1$ to n.

Let $n_1 = \lfloor n/2 \rfloor$ and $n_2 = \lceil n/2 \rceil$. Consider the covering $T_1 \triangle T_2$, where T_1 is a covering of the cube $\mathbb{B}^{n_1}_+$ that exists by the induction hypothesis, and T_2 is a partial covering of the cube $\mathbb{B}^{n_2}_+$ that satisfies (13.4) for $\delta = b/n_1$. Then

$$|T_1 \triangle T_2| = 2^{n_1}|T_2| + |T_1||\overline{T_2}| \leqslant 2^{n_1}|T_2|_\delta \leqslant (2\ln(\delta n_2) + 1)\frac{2^n}{n_2} \leqslant 2(2\ln(2b) + 1)\frac{2^n}{n}.$$

When choosing $b = 16$, we obtain $2(2\ln(2b) + 1) < b$, hence, the induction step is proved. □

Let $S^+(\alpha)$ and $S^-(\alpha)$ denote the positive and negative hemispheres of radius 1 centered at α, respectively.

Svetlana Nikolaevna Selezneva
Moscow University, since 1998

Let $t = \lfloor n/2 \rfloor$. Represent a boolean cube as $\mathbb{B}^n = \mathbb{B}^t \times \mathbb{B}^{n-t}$. For any vector $\alpha \in \mathbb{B}^n$, write $\alpha = (\alpha^0, \alpha^1)$, where $\alpha^0 \in \mathbb{B}^t$ and $\alpha^1 \in \mathbb{B}^{n-t}$. Denote $Q(\alpha) = S^-(\alpha^0) \times S^+(\alpha^1)$; call this set the *quadrant* centered at $\alpha \in \mathbb{B}^n$ (essentially, it is a quarter of a radius-2 sphere). From Lemma 13.2 it follows

Corollary 13.1. *There exists a covering T_n of the boolean cube $\mathbb{B}^n_\pm = \mathbb{B}^t_- \times \mathbb{B}^{n-t}_+$ by quadrants of cardinality $|T_n| \asymp 2^n/n^2$.*

▷ The desired covering is obtained as the direct product of a covering of the cube $\mathbb{B}^t_-$ by negative hemispheres and a covering of the cube $\mathbb{B}^{n-t}_+$ by positive hemispheres. □

Theorem 13.3 ([286]). $\mathsf{C}_{\oplus\wedge\oplus}(\mathcal{P}_n) \preccurlyeq 2^n/n^2$.

► Represent an arbitrary boolean function $f(X,Y)$, $|X| = t$, $|Y| = n-t$, basing on its Zhegalkin polynomial as

$$f(X,Y) = \bigoplus_{\sigma\in\mathbb{B}^t} \bigoplus_{\tau\in\mathbb{B}^{n-t}} f_{\sigma,\tau} X^\sigma Y^\tau, \qquad X^\sigma = \prod_{\sigma_i=1} x_i, \quad Y^\tau = \prod_{\tau_i=1} y_i.$$

Let f^* be the part of the function f consisting of monomials whose vector exponents belong to $\mathbb{B}^n_\pm$: $f^*(X,Y) = \bigoplus_{\sigma\in\mathbb{B}^t_-,\tau\in\mathbb{B}^{n-t}_+} f_{\sigma,\tau} X^\sigma Y^\tau$.

Consider a covering T of the boolean cube $\mathbb{B}^n_\pm$ by quadrants guaranteed by Corollary 13.1. Let $\{\alpha_1, \ldots, \alpha_{|T|}\}$ be the set of quadrant centers numbered in order of decreasing weight, and, if weights are equal, in order of decreasing weight of the first part α^0_i.

Next, we perform $|T|$ steps of transforming the representation of the function f^*. After each j-th step, we have $f^* = P_j \oplus R_j$, where P_j is the sum of $2j$ multi-affine functions[10], and the "remainder" R_j has the form

$$R_j = \bigoplus_{\gamma=(\gamma^0,\gamma^1)\in\mathbb{B}^n_\pm \setminus \bigcup_{i=1}^j Q(\alpha_j)} c_{j,\gamma} X^{\gamma^0} Y^{\gamma^1}. \tag{13.5}$$

(Note that for all sets γ over which the summation is performed, $|\gamma| \leqslant |\alpha_j|$ holds.) Initially set $P_0 = 0$ and $R_0 = f^*$.

The next j-th step is as follows. Starting from the representation $f^* = P_{j-1} \oplus R_{j-1}$, we select from the polynomial R_{j-1} monomials whose vector exponents belong to $Q(\alpha_j)$, and set

$$R'_j = \bigoplus_{\gamma\in Q(\alpha_j)} c_{j-1,\gamma} X^{\gamma^0} Y^{\gamma^1}.$$

[10] *Multi-affine functions* are products of affine boolean functions, i.e. exactly those functions that admit a $\bigwedge\bigoplus$-representation.

For any $\sigma \in S_-(\alpha_j^0)$ define a linear function

$$g_\sigma(Y) = \bigoplus_{\tau \in S_+(\alpha_j^1)} c_{j-1,(\sigma,\tau)} Y^{\tau \oplus \alpha_j^1}.$$

Then we obtain

$$R'_j = Y^{\alpha_j^1} \bigoplus_{\sigma \in S_-(\alpha_j^0)} X^\sigma g_\sigma(Y).$$

Set

$$A_j = X^{\alpha_j^0} Y^{\alpha_j^1} \oplus Y^{\alpha_j^1} \cdot \prod_{\sigma \in S_-(\alpha_j^0)} (X^{\sigma \oplus \alpha_j^0} \oplus g_\sigma(Y)), \qquad \begin{matrix} P_j = P_{j-1} \oplus A_j \\ R_j = R_{j-1} \oplus A_j \end{matrix}. \tag{13.6}$$

By construction, $A_j = R'_j \oplus \bigoplus_\gamma X^{\gamma^0} Y^{\gamma^1}$ for some set of vectors γ satisfying the conditions $|\gamma| \leqslant |\alpha_j|$ and $|\gamma^0| \leqslant |\alpha_j^0| - 2$. As a consequence, $\gamma \notin \bigcup_{i=1}^j Q(\alpha_j)$. Therefore, P_j is a sum of $2j$ multi-affine functions, and R_j has form (13.5).

After step $|T|$ we obtain $f^* = P_{|T|}$. By adding to a $\bigoplus\bigwedge\bigoplus$-representation of f^* the monomials of $f \oplus f^*$ (there are at most $2^t + 2^{n-t}$ of them), we obtain a $\bigoplus\bigwedge\bigoplus$-circuit for f containing $\leqslant 2|T| + 2^t + 2^{n-t} \asymp 2^n/n^2$ multiplication gates on the second layer and $\leqslant t2^{n-t}$ summation gates on the first layer by (13.6). ■

• The order of layers in bounder-depth circuits matters. In particular, the complexity of $\bigwedge\bigoplus\bigwedge$-circuits for n-variable functions cannot be estimated better than 2^n (example: disjunction of n variables).

Starting from depth 4, the complexity of circuits over the basis $\{\bigwedge, \bigoplus, 1\}$ is of order $2^{n/2}$ (as is easy to verify). In the model of AC-circuits, due to the duality of the operations $\wedge$ and $\vee$, the effect of the exceptionality of depth 3 does not take place: the asymptotics $2 \cdot 2^{n/2}$ of the complexity of the class of n-variable functions is achieved immediately on circuits of depth 3, as shown by the author in [299].

Depth-4 $AC[\oplus]$-circuits for the majority function $\boxed{P}\boxed{\varepsilon}$

It is known that minimal AC-circuits of depth d for the majority function of n variables have complexity $2^{n^{1/(d-1)\pm o(1)}}$: the lower bound is proved by J. Håstad [122], and the upper bound is achieved on simple monotone circuits constructed by R. Boppana [40]. Recently, I. Oliveira, R. Santhanam, and S. Srinivasan [232] discovered that the $AC[\oplus]$-circuit model, although not improving the result in depth 3, already demonstrates (somewhat unexpectedly) higher computational power on circuits of depth 4. Before we proceed to the presentation of the method [232], which combines two probabilistic arguments with the idea of approximate computations, let us recall for comparison the construction of optimal depth-3 circuits.

Theorem 13.4 ([40]). $\mathsf{C}_{\bigvee\bigwedge\bigvee}(\mathrm{maj}_n) \preccurlyeq \sqrt{n} \cdot 2^{\sqrt{n \log n}}$.

▶ Divide the set of variables into r groups $X_1, \ldots, X_r$ of the same size $|X_i| = m = \lceil n/r \rceil$. We can write

$$\mathrm{maj}_n(X) = \bigvee_{k_1 + \ldots + k_r = n/2} T_m^{k_1}(X_1) \cdot \ldots \cdot T_m^{k_r}(X_r). \tag{13.7}$$

The circuit is constructed by formula (13.7), in which the threshold functions T_m^k are expressed via CNF. At the first layer of the $\bigvee\bigwedge\bigvee$-circuit, all possible disjunctions of variables in each of the groups X_i are computed: a total of $\leqslant r2^m$ pieces. At the second layer, internal products in (13.7) are computed — there are at most $C_{n/2+r-1}^{r-1} < n^r$ of them. The required estimate is obtained for $r \approx \sqrt{n/\log n}$ and $m \approx \sqrt{n \log n}$. ■

Theorem 13.5 ([232]). $\mathsf{C}_4^{AC[\oplus]}(\mathrm{maj}_n) \leqslant 2^{n^{1/4+o(1)}}$.

Srikanth Srinivasan
Indian Institute of Technology, Mumbai, 2012 to 2020

▶ The synthesis method combines ideas familiar from the formula implementation of the majority function: calculating arithmetic sums of variables (for this, $\oplus$ operations are used) and constructing monotone approximations. Only in contrast to Valiant's method [326] the increase in the accuracy of the approximation is achieved not by sequential, but by parallel steps.

Due to

$$\mathrm{maj}_n = \bigvee_{k \geqslant n/2} E_n^k, \qquad E_n^k = T_n^k \cdot \overline{T_n^{k+1}}, \tag{13.8}$$

the computation of the function maj_n is reduced to the computation of elementary symmetric functions E_n^k by depth-4 circuits with an output disjunction element. It is sufficient to describe a circuit for[11] $E_n^{n/2}$.

I. First, we construct a monotone $\bigwedge\bigvee\bigwedge$-circuit that approximately computes the function maj_m. For convenience, we assume that m is even. The following result is essentially due to K. Amano [12].

Lemma 13.3 ([232]). *Let $0 < \delta \leqslant 1/(4 \ln m)$ and m be sufficiently large. For some probability distribution Δ on the set of $\bigwedge\bigvee\bigwedge$-formulae of m variables of complexity $2^{\sqrt{\log m/\delta}}$, a random formula $G(X) \in \Delta$ satisfies the following conditions:*

(i) For any input $\sigma \in \mathbb{B}^m$ of weight $\leqslant (1/2 - \delta)m$, we have $G(\sigma) = 0$;

(ii) For any input $\sigma \in \mathbb{B}^m$ of weight $m/2$, we have $\mathbf{P}(G(\sigma) = 1) \geqslant 1/2$.

▷ We define a sequence of distributions Δ_k of depth-k formulae, parameterized by numbers $l_k \in \mathbb{N}$. On the set of variables we introduce a uniform distribution Δ_0: for $G \in \Delta_0$ we set $\mathbf{P}(G \equiv x_i) = 1/m$. The distribution Δ_k contains formulae $G_1 \circ \ldots \circ G_{l_k}$, where G_i are randomly chosen from Δ_{k-1}, and $\circ = \wedge$ for odd k and $\circ = \vee$ for even k.

[11] Any function E_n^k is a subfunction of the function E_{2n}^n.

Let $|\sigma| = m/2$. Then, since $1 - x \leqslant e^{-x}$ for $x \in \mathbb{R}$,

$$p_1 = \mathbf{P}(G(\sigma) = 1 \mid G \in \Delta_1) = 2^{-l_1},$$
$$p_2 = \mathbf{P}(G(\sigma) = 0 \mid G \in \Delta_2) = (1 - p_1)^{l_2} \leqslant e^{-l_2 p_1},$$
$$p_3 = \mathbf{P}(G(\sigma) = 0 \mid G \in \Delta_3) \leqslant l_3 p_2.$$

Now let $|\sigma| \leqslant (1/2 - \delta)m$. Under the assumption $\delta l_1 \leqslant 1/2$,

$$q_1 = \mathbf{P}(G(\sigma) = 1 \mid G \in \Delta_1) \leqslant ((1 - 2\delta)/2)^{l_1} \leqslant (1 - \delta l_1)p_1,$$
$$q_2 = \mathbf{P}(G(\sigma) = 0 \mid G \in \Delta_2) = (1 - q_1)^{l_2} \geqslant e^{-l_2(q_1 + q_1^2)},$$
$$q_3 = \mathbf{P}(G(\sigma) = 1 \mid G \in \Delta_3) = (1 - q_2)^{l_3} \leqslant e^{-q_2 l_3}.$$

When estimating q_1, we used the inequality $(1-x)^a \leqslant 1 - ax/2$ valid as far as $ax \leqslant 1$, and when estimating q_2, we used the inequality $1 - x \geqslant e^{-x-x^2}$ valid for $0 \leqslant x \leqslant 1/2$.

Let[12] $l_1 = \sqrt{\ln m/\delta}$, $l_2 = cl_1 2^{l_1}$, $l_3 = e^{cl_1 - 2}$, where $c \in \mathbb{R}$. In this case, $\delta l_1 = \sqrt{\delta \ln m} \leqslant 1/2$.

Then $p_2 \leqslant e^{-cl_1}$ and $p_3 \leqslant e^{-2} < 1/4$. Further,

$$q_2 \geqslant e^{-l_2(q_1+q_1^2)} \geqslant e^{-l_2 p_1(1-\delta l_1 + (1-\delta l_1)^2 p_1)} \geqslant e^{-cl_1(1+p_1/4)+c\delta l_1^2} \geqslant m^c e^{-cl_1(1+p_1/4)},$$

$$q_3 \leqslant e^{-q_2 l_3} \leqslant e^{-m^c e^{-cl_1 p_1/4 - 2}} < e^{-m^c e^{-c/4-2}},$$

since $x2^{-x} < 1$ when $x \geqslant 1$. If we choose $c = 2$, then $q_3 \leqslant e^{-m^2/13}$.

Now the probability that $G(\sigma) = 1$ for some vector σ of weight $\leqslant (1/2 - \delta)m$ can be estimated as $q_4 < 2^m q_3 \leqslant 2^m e^{-m^2/13}$. This value does not exceed $1/4$ for $m \geqslant 11$.

We define the distribution Δ as the restriction of Δ_3 to the set of formulae for which condition (i) is satisfied. Then for any vector $\sigma \in \mathbb{B}^m$ of weight $m/2$ we obtain $\mathbf{P}(G(\sigma) = 1 \mid G \in \Delta) \geqslant 1 - p_3 - q_4 \geqslant 1/2$. □

II. Now the function $E_m^{m/2}(X)$ can be calculated approximately as

$$\tilde{E} = \Psi_m(X) \cdot \Psi'_m(\overline{X}) \cdot \mathrm{MOD}_m^{s,r}(X),$$

where $r = m/2 \bmod s$, Ψ_m, Ψ'_m are random functions implemented by circuits from Lemma 13.3, and $\overline{X}$ is the vector of negations of variables. Under the condition $s \geqslant \delta m$, the approximating (random) function $\tilde{E}(X)$ is 0 outside the middle layer of the boolean cube $\mathbb{B}^m$, and at any point of the middle layer it is 1 with probability $\geqslant 1/4$. In the case $s = 2^k$, the function $\mathrm{MOD}_m^{s,r}$ has a simple representation as a Zhegalkin polynomial.

Lemma 13.4. *For $n \geqslant 2^k$ and any r, we have $\mathsf{C}_{\oplus\wedge}(\mathrm{MOD}_n^{2^k,r}) \leqslant n^{2^k}$.*

▷ Let $X^S = \prod_{i \in S} x_i$. Note that $C_n^{2^k-1}$ is odd only for $n \equiv -1 \bmod 2^k$. As a consequence, $\mathrm{MOD}_n^{2^k,-1}(X) = \bigoplus_{|S|=2^k-1} X^S$. Then an arbitrary function $\mathrm{MOD}_n^{2^k,r}(X)$ can be represented as $\mathrm{MOD}_{n+t}^{2^k,-1}(X, 1^t)$, where 1^t is the all-1 vector of length $t = 2^k - r - 1$.

[12] We ignore rounding in the following (rather rough) calculations.

A polynomial of degree 2^k-1 expressing the function definitely contains no more than $C_n^{2^k-1}+C_n^{2^k-2}+\ldots+C_n^0<n^{2^k}$ monomials. □

(If we stop right here and implement the function E_{2m}^m as a disjunction of a suitable number of independent random functions $\tilde{E}$, then with an appropriate choice of parameters δ and s we end with a circuit of complexity $2^{m^{1/3+o(1)}}$. But such a result can be obtained much more simply by generalizing the method of Theorem 13.4 even without using $\oplus$-operations [40].)

III. Perhaps the key idea of the method [232] is based on the observation that if a set consisting of an equal number of elements of two types is randomly divided into subsets of even size, then each of the subsets will also contain an equal number of elements of both types with a fairly high probability.

Let $\tilde{X}_1,\ldots,\tilde{X}_r$ be a random partition of a set of n variables into groups of m variables[13]. Consider a random function $\Phi=\prod_{i=1}^r\tilde{E}_i(\tilde{X}_i)$, in which the partition and all inner functions $\tilde{E}_i$ are chosen independently; the functions $\tilde{E}_i$ have the form $\tilde{E}$. By construction, $\Phi\leqslant E_n^{n/2}$. In this case

$$p=\mathbf{P}(\Phi(\sigma)=1\mid|\sigma|=n/2)\geqslant\frac{1}{2^{2r}}\cdot\frac{(C_m^{m/2})^r}{C_n^{n/2}}\geqslant\frac{1}{4^r}\cdot\frac{(2^m/m)^r}{2^n}=\frac{1}{(4m)^r}.$$

Then the disjunction $\bigvee_{i=1}^{n/p}\Phi_i$ of independent random functions of type Φ takes value 0 at least on one set of weight $n/2$ with probability $\leqslant C_n^{n/2}(1-p)^{n/p}<2^ne^{-n}<1$. Therefore, some depth-4 formula of the form $\bigvee_{i=1}^{n/p}\Phi_i$ computes the function $E_n^{n/2}$.

After choosing parameters $r,s\asymp\sqrt[4]{n/\log n}$, $\delta=s/m$, the complexity of the constructed circuit is estimated as $(n/p)r\left(m^s+2^{\sqrt{\log m/\delta}}\right)\asymp 2^{\sqrt[4]{n\log^3 n}}$. ■

- For $AC[\oplus]$-circuits of arbitrary depth d, the authors [232] obtained bounds

$$2^{n^{1/(2d-4)}-o(1)}\leqslant\mathsf{C}_d^{AC[\oplus]}(\mathrm{maj}_n)\leqslant 2^{n^{2/(3d-12)}+o(1)}.$$

As noted above, for $d=3,4$ the lower bound is tight; from the proof [232] follows a slightly better upper bound, which strengthens the standard estimate for the complexity of AC-circuits [40] for the remaining $d\geqslant 5$.

Cut method. Linear circuits for Sylvester matrices ⫽

The cut method is used in parallel computation models, which include bounded-depth circuits. The idea is to make a cut (one or more) in an original, generally speaking, non-parallel circuit, guided by an appropriate rule, and then perform parallel reconstruction of the subcircuits before and after the cut independently.

Next, we will consider two examples of the application of this method, a simple and a more complex one.

The boolean version of the *Sylvester matrix* is defined as

$$H_1=[0],\qquad H_2=\begin{bmatrix}0&0\\0&1\end{bmatrix},\qquad H_{2n}=\begin{bmatrix}H_n&H_n\\H_n&\overline{H_n}\end{bmatrix}. \tag{13.9}$$

[13]For simplicity, we assume that n is divisible by $2r$.

This matrix is the binary analogue of the DFT matrix and has many remarkable properties (see, e.g., [142]).

The recursive definition of the Sylvester matrix is consistent with the modified definition of the Kronecker product of boolean matrices $A\overline{\otimes}B$, in which the matrices B are substituted into the positions of zeros of the matrix A, and the matrices $\overline{B}$ are substituted into the positions of ones. Then

$$H_{n_1 n_2} = H_{n_1}\overline{\otimes}H_{n_2}, \qquad \overline{H}_{n_1 n_2} = \overline{H}_{n_1}\overline{\otimes}H_{n_2}. \tag{13.10}$$

The standard linear circuit for the matrix H_n has complexity $\asymp n\log n$ and is constructed directly by definition (13.9). The circuit consists of $\log_2 n$ layers: at the next layer, pairs of matrices H_{2m}, $\overline{H_{2m}}$ for groups of $2m$ variables are assembled from matrices of size $m\times m$ for the subgroups of m variables.

Theorem 13.6 ([142]). *For $d\geqslant 2$ and $n=2^k$, we have $\mathsf{W}_d(H_n)\preccurlyeq dn^{1+1/d}$.*

► In fact, we will construct a circuit for the pair of matrices H_n, $\overline{H}_n$. To do this, take the standard circuit and split its layers into d groups of approximately equal size. Then glue the layers inside each group, i.e., replace them with a trivial depth-1 circuit. For $d=1$ we have $\mathsf{W}_1(H_n,\overline{H}_n)=|H_n|+|\overline{H}_n|=n^2$.

In view of (13.10),

$$\mathsf{W}_{d+1}(H_{n_1n_2},\overline{H}_{n_1n_2})\leqslant 2n_1^2n_2+n_1\mathsf{W}_d(H_{n_2},\overline{H}_{n_2}).$$

Hence, for $n=n_1\cdot\ldots\cdot n_d$, where $n_i\in\{2^{\lfloor k/d\rfloor},2^{\lceil k/d\rceil}\}$, we obtain

$$W_d(H_n,\overline{H}_n)\leqslant 2n(n_1+\ldots+n_d)=2n[(1+x)2^{-x}]d2^{k/d},\qquad x=\frac{k}{d}-\left\lfloor\frac{k}{d}\right\rfloor. \tag{13.11}$$

The function in square brackets on the interval $[0,\,1]$ takes its maximum value $2/(e\ln 2)\approx 1.06$ at $x=\log_2(e/2)$. ■

• The result of the theorem is order-tight, since the lower bound proved in [142] is $\mathsf{W}_d(H_n)\geqslant \mathsf{W}_d^{\vee}(H_n)\gtrsim dn(n/2)^{1/d}$. If we note that no additional matrix needs to be computed at the last layer, the bound of the theorem can be established in a more accurate form $\mathsf{W}_d(H_n)\leqslant 2(d-1)n^{1+1/d}$, using the relation $W_d(H_n)\leqslant n(n_1+2n_2+\ldots+2n_{d-1}+n_d)$ instead of (13.11).

Reconstruction of arithmetic circuits into ΣΠΣΠ-circuits [//]

In [1] M. Agrawal and V. Vinay obtained in some way an unexpected and resonant result: it turns out that arithmetic circuits over sufficiently general rings can be modeled by arithmetic circuits of depth 4 and of slightly greater complexity. The complexity bound was subsequently refined and took its final form after the work of S. Tavenas [318].

Manindra Agrawal
Indian Institute of Technology, Kanpur, since 1996

The idea of the method is very simple: a circuit of depth 4 is constructed by cutting the original circuit approximately in half, and each half is reduced to depth 2. As a result, a polynomial is written as a composition of polynomials. Essentially, the task is only to select a suitable cut and carefully estimate the complexity of the new circuit.

The following statements are formulated for rings of characteristic 0: actually, it is required that the basic ring R satisfy the condition $\deg(fg) = \deg f + \deg g$ for any polynomials $f, g \in R[x]$. An arithmetic circuit is called *homogeneous* if only homogeneous polynomials are computed at the outputs of its elements.

Lemma 13.5 ([314]). *Let R be a ring of characteristic 0. If a polynomial $f \in R[X]$ of degree d is computed by an arithmetic circuit with s nonscalar multiplication elements, then all its homogeneous components can be computed by a homogeneous arithmetic circuit with sd^2 nonscalar multiplication elements.*

▷ Split any polynomial obtained in the process of computations into homogeneous components of degree $\leqslant d$ (components of higher degrees are not required). Replace the multiplications of the initial circuit with multiplications of homogeneous components of degree from 1 to $d-1$. So we obtain a circuit that computes all homogeneous components of the polynomial f. □

The following lemma describes the transition from a usual (homogeneous) arithmetic circuit to a $\Sigma\Pi\Sigma\Pi$-circuit. The proof is carried out by cutting the original circuit in half: the cut line passes through the elements in which the degrees of the intermediate polynomials overcome a given threshold h. The reasoning scheme is close to [327]. By $\operatorname{mon} f$ we denote the set of monomials of a polynomial f.

Lemma 13.6. *Let a (homogeneous) polynomial $f \in R[x_1, \ldots, x_n]$ of degree d be computed by a homogeneous arithmetic circuit S with s multiplication elements, where R is a ring of characteristic 0. Then for any $h \leqslant d$ it can be represented as*

$$f(x_1, \ldots, x_n) = p(q_1, \ldots, q_r), \quad p \in R[y_1, \ldots, y_r], \quad q_j \in R[x_1, \ldots, x_n], \tag{13.12}$$

where $\deg q_j \leqslant h$, $\deg p < 6d/h$, $|\operatorname{mon} p| \leqslant s^{\deg p}$, and $r \leqslant s^2 + n$.

▷ We classify the edges of the circuit S that are inputs to the multiplication elements. An incoming edge that delivers a factor of higher degree to the element of multiplication is called *strong*, another edge is called *weak*. In the case of equality of the degrees of the polynomials being multiplied, we appoint the strong and weak edges in a pair arbitrarily. A directed path connecting two vertices in a circuit is called *legal* if it does not contain weak edges.

Denote by $g(v)$ the polynomial evaluated at a vertex $v \in S$. For a multiplication element v, let $g(v) = g_1(v) \cdot g_2(v)$, where $g_1(v)$ is the factor coming along the strong

edge, $g_2(v)$ — along the weak edge. For an addition element v, let $g_1(v) = g(v)$ and $g_2(v) = 1$. For a legal path $\rho = (v_1, \ldots, v_l)$, let $g(\rho) = g_2(v_2) \cdot \ldots \cdot g_2(v_l)$. In the case $l = 1$, we formally set $g(\rho) = 1$. Finally, we define

$$g(v, w) = \sum_{\rho=(v,\ldots,w)} g(\rho),$$

where the summation is over all legal paths from v to w; if there are no such paths, we set $g(v, w) = 0$.

Let $V_t = \{v \in S \mid \deg g(v) \geqslant t > \deg g_1(v)\}$ be the set of multiplication elements in which the degree of polynomials being computed first overcomes a threshold t. Obviously, the set V_t is an antichain. Due to the homogeneity of the circuit, any its input-output path passes through an element from V_t (of course, if $t \leqslant d$).

Claim 13.1.
(i) Let $\deg g(w) \geqslant t$. *Then* $g(w) = \sum_{v \in V_t} g(v) \cdot g(v, w)$.
(ii) Let $\deg g(u, w) \geqslant t - \deg g(u) > 0$. *Then* $g(u, w) = \sum_{v \in V_t} g(u, v) \cdot g(v, w)$.

▷ The proof of (i) is carried out by induction. We will sequentially examine the vertices of the circuit, starting from the layer V_t, according to the order of calculations[14)].

Base of induction: if $w \in V_t$, then there are no other vertices in V_t preceding w, so the equality being verified is the identity $g(w) = g(w) \cdot g(w, w)$.

Otherwise, if w is a multiplication element, then for the vertex w_1 preceding w along a strong edge, $\deg g(w_1) \geqslant t$ holds. Therefore, by the induction hypothesis, $g(w_1) = \sum_{v \in V_t} g(v) \cdot g(v, w_1)$. Since all legal paths to w pass through w_1, we have $g(v, w) = g(v, w_1) \cdot g_2(w)$. So, we obtain

$$g(w) = g(w_1) \cdot g_2(w) = \sum_{v \in V_t} g(v) \cdot g(v, w_1) \cdot g_2(w) = \sum_{v \in V_t} g(v) \cdot g(v, w).$$

If w is an addition element, and it is preceded by elements w_1, w_2, then trivially

$$g(w) = g(w_1) + g(w_2) = \sum_{v \in V_t} g(v) \cdot (g(v, w_1) + g(v, w_2)) = \sum_{v \in V_t} g(v) \cdot g(v, w).$$

The proof of (ii) is completely analogous. Essentially, it suffices to formally put $g(v) := g(u, v)$. □

To represent polynomials $g(v)$ and $g(u, w)$, we employ the formulas provided by Claim 13.1:

$$g(w) = \sum_{v \in V_t} g_1(v) \cdot g_2(v) \cdot g(v, w), \tag{13.13}$$

$$g(u, w) = \sum_{v \in V_t} g(u, v') \cdot g_2(v) \cdot g(v, w), \tag{13.14}$$

[14)] We mean an easily verifiable fact that on the set of vertices of a directed acyclic graph, one can introduce a *natural numbering* by consecutive natural numbers, in which any edge goes from a vertex with a lower number to a vertex with a higher number.

where v' precedes v along a strong edge. In formula (13.13) we choose $t = (\deg g(w))/2$, then the degrees of all factors in it do not exceed t. In formula (13.14) we choose $t = \deg g(u) + (\deg g(u,w))/2$ (hence, $u \notin V_t$), then $\deg g(u,v'), \deg g(v,w) \leqslant (\deg g(u,w))/2$.

Now we describe the transformation of the circuit to form (13.12). Sequentially, in descending order of degrees, we express all polynomials $g(v)$ and $g(u,w)$ according to rules (13.13), (13.14) up to polynomials of degree $\leqslant h$, which we consider as formal variables $y_1, \ldots, y_r$. By construction, $r \leqslant s + C_s^2 + n \leqslant s^2 + n$. Denote by $\deg^*$ the degree of a polynomial $g(v)$ or $g(u,w)$ as a polynomial of variables y_i.

Claim 13.2.
(i) If $d' = \deg g(v) > h$, *then* $\deg^* g(v) \leqslant 6\lfloor d'/h \rfloor - 3$.
(ii) If $d' = \deg g(u,w) > h$, *then* $\deg^* g(u,w) \leqslant 6\lfloor d'/h \rfloor - 1$.

▷ Inequalities (i), (ii) are proved jointly by induction on d', relying on (13.13), (13.14). The induction base $d' \leqslant 4h$ is verified directly.

In view of the simple inequality $\lfloor a+b \rfloor \geqslant \lfloor a \rfloor + \lfloor b \rfloor$, it is only necessary to deal with situations when factors of degree $\leqslant h$ occur in formulas (13.13), (13.14).

Note that for $d' > 4h$ at most one factor in formula (13.13) and at most two factors (if two, then the leftmost and the rightmost ones) in formula (13.14) have degrees $\leqslant h$. The induction step immediately follows from this. □

Claim 13.3. *For* $g(y_1, \ldots, y_r) \in \{g(v),\ g(u,v)\}$, *we have* $|\operatorname{mon} g| \leqslant s^{\deg^* g - 1}$.

▷ Trivial proof by induction. For $\deg^* g = 1$ the statement is obvious. The induction step is provided by rules (13.13), (13.14), since by the choice of the threshold t at least two factors in each internal product differ from 1. □

Claims 13.2 and 13.3 provide the required bounds on $\deg p$ and $|\operatorname{mon} p|$. The lemma is proved. □

• The number of monomials in Claim 13.3 can be estimated more accurately as $s^{3\lfloor d'/h \rfloor - 1}$, where $d' = \deg g(x_1, \ldots, x_n) \geqslant h$ (the proof is similar to that of Claim 13.2). Then the conclusion of Lemma 13.6 can be refined to $|\operatorname{mon} p| \leqslant s^{3d/h}$.

Theorem 13.7 ([318]). *Let* R *be a ring of characteristic* 0. *If a polynomial* $f \in R[x_1, \ldots, x_n]$ *of degree* d *has multiplicative complexity* s, *then* $\mathsf{C}^R_{\Sigma\Pi\Sigma\Pi}(f) = 2^{O\left(\sqrt{d \log(sd) \log n}\right)}$.

▶ In the proof we use the simple inequality $C^k_{n+k} \leqslant (k+1)n^k$.

Consider the case of a homogeneous polynomial f. We apply Lemma 13.6 to the circuit transformed via Lemma 13.5 with the choice $h \approx \sqrt{\frac{d \log(sd)}{\log(3n)}}$. Note that this ensures that $h \leqslant d$, since it is known that $sd \leqslant dC^d_{n+d} \leqslant d(d+1)n^d \leqslant (3n)^d$.

The circuit constructed according to formula (13.12) has at most $C^h_{n+h} \leqslant (h+1)n^h = 2^{O\left(\sqrt{d \log(sd) \log n}\right)}$ elements on the first layer (all possible monomials of

degree $\leqslant h$ of variables x_i), at most $s^2d^4 + n = 2^{O(\log(sd))}$ elements on the second layer, and at most $(sd^2)^{6d/h} = 2^{O\left(\sqrt{d\log(sd)\log n}\right)}$ elements on the third layer.

When implementing an arbitrary polynomial, just combine the circuits that calculate its homogeneous components (by the final addition element). ■

Note that the transformation to depth-4 circuits is performed by rebuilding the circuit topology, so the result of Theorem 13.7 is valid for computations in rings R of a fairly general form, even, for example, for circuits over the tropical semiring $(\mathbb{R}, \min, +)$.

Theorem 13.7 shows that if the complexity s of a polynomial is not very large, then its complexity in the model of depth-4 circuits is not too large either. For example, if $s = 2^{o(d\log n)}$, then the depth-4 complexity will also be $2^{o(d\log n)}$. Various consequences of this result are provided in [1, 115, 318]. One of the most interesting is record-breaking simple circuits for the determinant of an $n\times n$ matrix. Recall that the ordinary complexity of computing the determinant is polynomial, see Theorem 9.4.

Corollary 13.2 ([318]). *The determinant of an $n \times n$ matrix over a ring of characteristic 0 can be computed by a $\Sigma\Pi\Sigma\Pi$-circuit of complexity $2^{O(\sqrt{n}\cdot\log n)}$.*

It was not possible to construct such circuits in any other way. Before the works [1, 115] appeared, even circuits of bounded depth and complexity $2^{o(n)}$ were not known for the determinant.

Reconstruction of arithmetic circuits into $\Sigma\Pi\Sigma$-circuits

The results of the previous section received a further development, and this time it was truly unexpected. A group of Indian mathematicians [115] have managed to reconstruct depth-4 circuits into depth-3 circuits with virtually no change in the complexity estimate if the calculations are performed over the fields $\mathbb{Q}, \mathbb{R}, \mathbb{C}$. The transformation into a depth-3 circuit exploits two algebraic (and partly number-theoretic) techniques: at an intermediate step, a circuit of depth 5 is obtained, in which all multiplications are exponentiations.

The algebraic tools narrow the set of admissible rings, so the subsequent presentation is limited to the case of complex polynomials.

Neeraj Kayal
Microsoft Research Lab, Bengaluru, since 2008

Lemma 13.7. *In a field of characteristic 0,*

$$2^{n-1}n!\,x_1\cdot\ldots\cdot x_n = \sum_{\varepsilon_2,\ldots,\varepsilon_n=\pm1} \varepsilon_2\cdot\ldots\cdot\varepsilon_n\,(x_1+\varepsilon_2x_2+\ldots+\varepsilon_nx_n)^n.$$

▷ The proof is carried out by "expansion the brackets" on the right-hand side and reducing similar terms. It is easy to verify that the monomial $x_1 \cdot \ldots \cdot x_n$ appears in any term under the summation sign with coefficient $n!$. In any other monomial of degree n, some variable x_i occurs in an even power (possibly in zero). Then this monomial appears in any

pair of terms of the sum that differ only in the value of ε_i, with opposite coefficients. As a consequence, its coefficient in the total sum is 0. □

• The formula of the lemma has been known for a long time. According to [106], its various versions were proposed at school and student olympiads in the USSR in the 1980s. The formula was published by I. Fischer in [86], and S. B. Gashkov and E. T. Shavgulidze [106] proved its optimality: it is impossible to get by with fewer number of linear forms on the right-hand side than 2^{n-1}. R. Saptharishi [272] observed that an alternative representation in the form of a sum of 2^n powers of linear forms is also provided by the well-known formula of H. Ryser [269] for the algebraic permanent:

$$n!\,x_1 \cdot \ldots \cdot x_n = \operatorname{per} \begin{pmatrix} x_1 & x_2 & \cdots & x_n \\ x_1 & x_2 & \cdots & x_n \\ \vdots & \vdots & \ddots & \vdots \\ x_1 & x_2 & \cdots & x_n \end{pmatrix} = \sum_{T \subset [\![n]\!]} (-1)^{n-|T|} \left(\sum_{i \in T} x_i \right)^n .$$

The following lemma allows to pass from a ΣΠΣΠ-representation of a polynomial f (13.12) to a ΣEΣEΣ-representation, where E denotes a layer of exponentiation elements.

Lemma 13.8 ([115, 318]). *Under the conditions of Lemma 13.6, a polynomial $f \in \mathbb{C}[x_1, \ldots, x_n]$ of degree d for any $h \leqslant d$ can be represented as*

$$f(X) = \sum_{i=1}^{s_1} \left(\sum_{j=1}^{s_2} (l_{i,j}(X))^{h_{i,j}} \right)^{d_i}, \qquad d_i \leqslant 6d/h, \quad h_{i,j} \leqslant h, \quad \deg l_{i,j} \leqslant 1, \quad (13.15)$$

where $s_1 \leqslant (2s)^{6d/h}$ and $s_2 \leqslant h \cdot (2n)^h$.

▷ In formula (13.12), we express each monomial of the polynomial p by Lemma 13.7 as $\sum_{i=1}^{2^{k-1}} \lambda_i^k(y_1, \ldots, y_r)$, where k is the degree of the monomial, $\deg \lambda_i = 1$.

In each polynomial λ_i, we replace the variables Y with the corresponding polynomials of the variables X (i.e., make a substitution $y_j = q_j(X)$), and then, applying Lemma 13.7 again, we rewrite each monomial of the variables X as $\sum_{i=1}^{2^{k-1}} l_{i,j}^k(x_1, \ldots, x_n)$, where k is the degree of the monomial.

By construction, $s_1 \leqslant 2^{\deg p} \cdot |\operatorname{mon} p| \leqslant (2s)^{\deg p}$ and $s_2 \leqslant 2^{h-1} C_{h+n}^h \leqslant h(2n)^h$. □

The key to the synthesis of depth-3 circuits is an elegant result of N. Saxena [273], which allows one to express a power of a linear form as a sum of a relatively small number of products of polynomials of one variable.

Lemma 13.9 ([273]). *For any $d, n > 0$ and distinct numbers $a_1, \ldots, a_{dn+1} \in \mathbb{C}$,*

$$(x_1 + \ldots + x_n)^d = \sum_{i=1}^{dn+1} b_i \prod_{j=1}^{n} E_d(a_i x_j), \qquad E_d(x) = 1 + x + \frac{x^2}{2} + \ldots + \frac{x^d}{d!},$$

with some $b_i \in \mathbb{C}$.

▷ Let $u = x_1 + \ldots + x_n$. Since

$$e^{uz} = 1 + uz + \ldots + (uz)^d/d! + \ldots,$$

$u^d/d!$ is the coefficient at z^d of the power series $e^{uz} \in \mathbb{C}[[z]]$. Due to

$$e^{uz} = e^{x_1 z} \cdot \ldots \cdot e^{x_n z} \equiv F(z) = E_d(x_1 z) \cdot \ldots \cdot E_d(x_n z) \mod z^{d+1},$$

the desired value u^d is the coefficient of the polynomial $F(z)$ at z^d up to a constant factor.

According to the (Lagrange) interpolation formula, the coefficients of a polynomial of degree m can be expressed as linear combinations of the values of the polynomial at $m+1$ points, in this case, $F(a_1), \ldots, F(a_{dn+1})$. □

Theorem 13.8 ([115, 318]). *If a polynomial $f \in \mathbb{C}[x_1, \ldots, x_n]$ of degree d has multiplicative complexity s, then $\mathsf{C}^{\mathbb{C}}_{\Sigma\Pi\Sigma}(f) = 2^{O\left(\sqrt{d \log(sd) \log n}\right)}$.*

▶ By Lemma 13.9, any term of the outer sum in (13.15) can be rewritten as

$$\left(\sum_{j=1}^{s_2} (l_{i,j}(X))^{h_{i,j}}\right)^{d_i} = \sum_{k=1}^{s_2 d_i + 1} b_k \prod_{j=1}^{s_2} E_{d_i}(a_k (l_{i,j}(X))^{h_{i,j}}).$$

Over the field $\mathbb{C}$, a polynomial $E_m(ax^q)$ can be represented as a product of linear factors $\prod_{i=1}^{qm} \lambda_{a,m,q,i}(x)$. Finally, we obtain

$$f(X) = \sum_{i=1}^{s_1} \sum_{k=1}^{s_2 d_i + 1} b_k \prod_{j=1}^{s_2} \prod_{u=1}^{d_i h_{i,j}} \lambda_{a_k, d_i, h_{i,j}, u}(l_{i,j}(X)).$$

The formula contains (according to the bounds of Lemma 13.8 and taking into account Lemma 13.5) of order

$$s_1 s_2^2 d^2/h \leqslant (2sd^2)^{6d/h} (2n)^{2h} d^2 h$$

linear factors. By choosing $h \approx \sqrt{\frac{d \log(sd)}{\log(3n)}}$ (recall that in this case $h \leqslant d$) we obtain the required bound. ■

Nitin Saxena
Indian Institute of Technology, Kanpur, since 2013

Now Corollary 13.2 can be strengthened.

Corollary 13.3 ([318]). *The determinant of a complex $n \times n$ matrix can be computed by a $\Sigma\Pi\Sigma$-circuit of complexity $2^{O(\sqrt{n} \cdot \log n)}$.*

• The authors [115] proved that the result of Theorem 13.8 is also valid over the field $\mathbb{Q}$. It follows from the paper [187] that the bound of Theorem 13.8 is close to optimal: an example is given of a degree-d polynomial f of n variables, for which $\mathsf{C}_{\Sigma\Pi\Sigma}(f) = 2^{\Omega(\sqrt{d} \cdot \log n)}$, although under the constraint $d \prec \log n$.

An analogue of Theorem 13.8 for finite fields does not hold. Indeed, D. Grigoriev and A. A. Razborov [113] obtained, in essence, an extremely possible lower bound $2^{\Omega(n)}$ for the complexity of computing symmetric functions MOD_q^n over $\mathbb{F}_p$ by depth-3 circuits under the condition $p, q \in \mathbb{P}$, and $q \neq p$, as well as the majority function maj_n over $\mathbb{F}_2$. As is known [45], the multiplicative complexity of any nonlinear symmetric function of n variables over $\mathbb{F}_2$ is $\Theta(n)$. It is rather easy to verify that for $p = O(1)$ we have $\mathsf{C}_{\mathcal{A}^{\mathbb{F}_p}}(\mathrm{MOD}_2^n) = n^{O(1)}$.

Theorem 13.8 cannot be extended to tropical circuits either. In particular, in [210] it is shown that the tropical version of the polynomial $CONN_n$ of degree n in C_n^2 variables (corresponding to the problem of finding the shortest path in a graph) has complexity $2^{\Theta(n \log n)}$ when implemented by depth-3 circuits over the semiring $(\mathbb{R}, \min, +)$.

Bibliography

[1] Agrawal M., Vinay V. *Arithmetic circuits: A chasm at depth four.* Proc. FOCS (Philadelphia, 2008). Los Alamitos: IEEE, 2008, 67–75. 152, 156

[2] Aho A. V., Hopcroft J. E., Ullman J. D. *Data structures and algorithms.* Reading: Addison-Wesley, 1983. 129

[3] Ajtai M. Σ_1^1*-formulae on finite structures.* Annals of Pure and Applied Logic. 1983. **24**(1), 1–48. 141

[4] Ajtai M., Komlós J., Szemerédi E. *Sorting in* $c \log n$ *parallel steps.* Combinatorica. 1983. **3**(1), 1–19. 32, 60, 63, 67

[5] Ajtai M., Komlós J., Szemerédi E. *An* $O(n \log n)$ *sorting network.* Proc. STOC (Boston, 1983). NY: ACM, 1983, 1–9. 32, 60, 61, 63, 67

[6] Alekseev V. B. *Complexity of matrix multiplication.* In: Kiberneticheskii Sbornik[1)]. Vol. 25. Moscow: Mir, 1988, 189–236. (in Russian) 122

[7] Alekseev V. E. *Sorting algorithms with minimum memory.* Cybernetics. 1969. (5), 642–648. 32, 93

[8] Alman J., Duan R., Vassilevska Williams V., Xu Y., Xu Z., Zhou R. *More asymmetry yields faster matrix multiplication.* 2024. arXiv:2404.16349v1. 76, 80, 112, 115, 122

[9] Alman J., Guan Y., Padaki A. *Smaller low-depth circuits for Kronecker powers.* Proc. SODA (Florence, 2023). SIAM, 2023, 4159–4187. 143

[10] Alman J., Rao K. *Faster Walsh-Hadamard and Discrete Fourier transforms from matrix non-rigidity.* Proc. STOC (Orlando, Florida, 2023). NY: ACM, 2023, 455–462. 87

[11] Alon N., Schieber B. *Optimal preprocessing for answering on-line product queries.* Tech. report, 1987. http://www.math.tau.ac.il/~haimk/adv-ds-2008 112

[12] Amano K. *Bounds on the size of small depth circuits for approximating majority.* Proc. ICALP (Rhodes, Greece, 2009). LNCS. 2009. **5555**, 59–70. 149

[13] Amano K. *Integer complexity and mixed binary-ternary representation.* Proc. ISAAC (Seoul, 2022). LIPIcs. 2022. **248**, Art. 29. 16

[14] Andreev A. E. *On the synthesis of circuits of functional elements in complete monotone bases.* In: Matematicheskie Voprosy Kibernetiki[2)]. Vol. 1. Moscow: Fizmatlit, 1988, 114–139. (in Russian) 39, 41

[15] Barrett P. *Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor.* Advances in Cryptology. Proc. CRYPTO (Santa Barbara, 1986). LNCS. 1987. **263**, 311–323. 91

[16] Batcher K. E. *Sorting networks and their applications.* Proc. AFIPS spring joint comput. conf. (Atlantic City, 1968). **32**. NY: AFIPS, 1968, 307–314. 32, 60, 67

[1] Cybernetic Collection.

[2] Mathematical Problems of Cybernetics.

[17] Baur W., Strassen V. *The complexity of partial derivatives.* Theor. Comput. Sci. 1983. **22**, 317–330. 100, 102, 103

[18] Beals R. *Improved construction of negation-limited circuits.* DIMACS Tech. report 95-31. Princeton Univ., 1995. 92, 93

[19] Beals R., Nishino T., Tanaka K. *On the complexity of negation-limited boolean networks.* SIAM J. Comput. 1998. **27**(5), 1334–1347. 92

[20] Beame P. W., Cook S. A., Hoover H. J. *Log depth circuits for division and related problems.* SIAM J. Comput. 1986. **15**(4), 994–1003. 74, 75

[21] Belaga E. G. *On the computation of polynomials of one variable with preliminary processing of coefficients.* In: Problemy Kibernetiki[3)]. Vol. 5. Moscow: Fizmatlit, 1961, 7–16. (in Russian) 32

[22] Bellman R. *On a routing problem.* Quarterly of Appl. Math. 1958. **16**, 87–90. 17

[23] Bellman R. *Dynamic programming treatment of the travelling salesman problem.* J. ACM. 1962. **9**(1), 61–63. 18

[24] Beniamini G., Cheng N., Holtz O., Karstadt E., Schwartz O. *Sparsifying the operators of fast matrix multiplication algorithms.* 2020. arXiv:2008.03759v1. 88

[25] Bernstein D. J. *Multidigit multiplication for mathematicians.* 2001. http://cr.yp.to/papers.html#m3 73

[26] Bernstein D. J. *Computing logarithm intervals with the arithmetic-geometric-mean iteration.* 2003. http://cr.yp.to/papers.html#logagm 68

[27] Bernstein D. J. *The tangent FFT.* Proc. AAECC (Bangalore, 2007). LNCS. 2007. **4851**, 291–300. 87

[28] Bernstein D. J. *Fast multiplication and its applications.* Algorithmic Number Theory, MSRI Publ. 2008. **44**, 325–384. 73

[29] Bernstein D. J. *Batch binary Edwards.* Advances in Cryptology. Proc. CRYPTO (Santa Barbara, 2009). LNCS. 2009. **5677**, 317–336. 24

[30] Bernstein D. J., Yang B.-Y. *Fast constant-time gcd computation and modular inversion.* IACR TCHES. 2019. **3**, 340–398. 31

[31] Bhargava V., Ghosh S., Kumar M., Mohapatra C. K. *Fast, algebraic multivariate multipoint evaluation in small characteristic and applications.* Proc. SODA (Rome, 2022). NY: ACM, 2022, 403–415. 79

[32] Bini D. *Relations between exact and approximate bilinear algorithms.* Applications. Calcolo. 1980. **17**, 87–97. 58, 60, 100

[33] Bini D., Capovani M., Lotti G., Romani F. $O(n^{2.7799})$ *complexity for* $n \times n$ *approximate matrix multiplication.* Inform. Process. Lett. 1979. **8**(5), 234–235. 58, 100

[34] Bini D., Pan V. Y. *Polynomial and matrix computations. Vol.* 1. Boston: Birkhäuser, 1994. 2, 110, 112

[35] Bläser M. *Fast matrix multiplication.* Theory of Computing Library Graduate Surveys. 2013. **5**. 122

[36] Blelloch G. E. *Prefix sums and their applications.* in: Synthesis of parallel algorithms. San Francisco: Morgan Kaufmann, 1993, 35–60. 28

[37] Bloniarz P. A. *The complexity of monotone Boolean functions and an algorithm for finding shortest paths in a graph.* Ph.D. thesis. Tech. report no. 238. Lab. for Computer Science, MIT, 1979. 124

[38] Bodrato M. *A Strassen-like matrix multiplication suited for squaring and higher power computation.* Proc. ISSAC (Munich, 2010). NY: ACM, 2010, 273–280. 87, 88

[3] Problems of Cybernetics.

[39] Bolotov A. A., Gashkov S. B. *On fast multiplication in normal bases of finite fields.* Discrete Math. and Appl. 2001. **11**(4), 327–356. 81

[40] Boppana R. B. *Threshold functions and bounded depth monotone circuits.* Proc. STOC (Washington, 1984). NY: ACM, 1984, 475–479. 148, 151

[41] Boppana R. B. *Amplification of probabilistic Boolean formulas.* Proc. FOCS (Portland, 1985). Los Alamitos: IEEE, 1985, 20–29. 109

[42] Borodin A., von zur Gathen J., Hopcroft J. *Fast parallel matrix and GCD computations.* Inform. and Control. 1982. **52**, 241–256. 111

[43] Bostan A., Schost É. *Polynomial evaluation and interpolation on special sets of points.* J. Complexity. 2005. **21**, 420–446. 79, 93, 94

[44] Boyar J., Find M. *Cancellation-free circuits in unbounded and bounded depth.* Theor. Comput. Sci. 2015. **590**, 17–26. 143

[45] Boyar J., Peralta R. *Tight bounds for the multiplicative complexity of symmetric functions.* Theor. Comput. Sci. 2008. **396**, 223–246. 159

[46] Brauer A. *On addition chains.* Bull. AMS. 1939. **45**, 736–739. 34

[47] Brent R. P. *The parallel evaluation of general arithmetic expressions in logarithmic time.* J. ACM. 1974. **21**(2), 201–206. 49

[48] Brent R. P. *Multiple-precision zero-finding methods and the complexity of elementary function evaluation.* in: Analytic Computational Complexity. NY: Academic Press, 1975, 151–176. 54, 68

[49] Brent R. P., Kuck D. J., Maruyama K. *The parallel evaluation of arithmetic expressions without division.* IEEE Trans. Comp. 1973. C-**22**, 532–534. 49, 50

[50] Brent R. P., Kung H. T. *Fast algorithms for manipulating formal power series.* J. ACM. 1978. **25**(4), 581–595. 76, 80

[51] Brent R. P., Zimmermann P. *Modern computer arithmetic.* Cambridge University Press, 2010. 2, 68

[52] Bshouty N. H. *On the additive complexity of* 2×2 *matrix multiplication.* Inform. Proc. Letters. 1995. **56**(6), 329–335. 24

[53] Bürgisser P., Clausen M., Shokrollahi M. A. *Algebraic complexity theory.* Berlin–Heidelberg: Springer-Verlag, 1997. 122

[54] Cayley A. *A theorem on trees.* Quart. J. Pure Appl. Math. 1889. **23**, 376—378. 20

[55] Cenk M., Hasan M. A. *On the arithmetic complexity of Strassen-like matrix multiplications.* J. Symb. Comput. 2017. **80**, 484–501. 88, 89

[56] Chashkin A. V. *On the complexity of Boolean matrices, graphs and their corresponding Boolean functions.* Discrete Math. and Appl. 1994, **4**(3), 229–257. 96

[57] Chashkin A. V. *Discrete mathematics.* Moscow: Akademija, 2012. (in Russian) 32, 96

[58] Chashkin A. V. *On the computation of monotone Boolean functions.* Lecture notes of Youth Scientific Schools on Discrete Math. and its Appl. Iss. VIII. Moscow: Izd. IPM RAN, 2016, 30–44. (in Russian) 39

[59] Chen X., Kayal N., Wigderson A. *Partial derivatives in arithmetic complexity and beyond.* Found. Trends in Theor. Comput. Sci. 2011. **6**(1–2), 1–138. 127

[60] Cherukhin D. Yu. *Realization of linear functions by formulas in various bases.* Vestnik Mosk. Univ. Ser. 1. Matematika. Mekhanika[4)]. 2001. (6), 15–19. (in Russian) 44

[4]Moscow University Bulletin. Ser. 1. Mathematics. Mechanics.

[61] Cherukhin D. Yu. *On the problem of logical representation of the parity counter.* Neformal'naya Nauka[5]. 2008. (2), 14—23. (in Russian) 22

[62] Chin A. *On the depth complexity of the counting functions.* Inform. Proc. Letters. 1990. **35**, 325–328. 43, 44

[63] Chistikov D., Iván S., Lubiw A., Shallit J. *Fractional coverings, greedy coverings, and rectifier networks.* Proc. STACS (Hannover, 2017). LIPIcs. 2017. **66**, Art. 23. 143, 144, 145

[64] Chistov A. L. *Fast parallel calculation of the rank of matrices over a field of arbitrary characteristic.* Fundamentals of computation theory. LNCS. 1985. **199**, 63–69. 110

[65] Chockler H., Zwick U. *Which bases admit non-trivial shrinkage of formulae?* Comput. Complexity. 2001. **10**, 28–40. 44

[66] Chvátal V. *Lecture notes on the new AKS sorting network.* Tech. report DCS-TR-94. Rutgers Univ., 1992. 67

[67] Commentz-Walter B. *Size-depth tradeoff in monotone Boolean formulae.* Acta Inf. 1979. **12**, 227–243. 99

[68] Commentz-Walter B., Sattler J. *Size-depth tradeoff in non-monotone Boolean formulae.* Acta Inf. 1979. **14**, 257–269. 99

[69] Cook S. *On the minimum computation time of functions.* Ph. D. Thesis, Harvard Univ., 1966. 30, 53, 73

[70] Cooley J. W., Tukey J. W. *An algorithm for the machine calculation of complex Fourier series.* Math. Comp. 1965. **19**, 297–301. 25, 26

[71] Cooper J. N., Ellis R. B., Kahng A. B. *Asymmetric binary covering codes.* J. Comb. Theory, Ser. A. 2002. **100**, 232–249. 145, 146

[72] Coppersmith D. *Rapid multiplication of rectangular matrices.* SIAM J. Comput. 1982. **11**(3), 467–471. 115

[73] Coppersmith D., Schieber B. *Lower bounds on the depth of monotone arithmetic computations.* J. Complexity. 1999. **15**(1), 17–29. 51

[74] Coppersmith D., Winograd S. *On the asymptotic complexity of matrix multiplication.* SIAM J. Comput. 1982. **11**(3), 472–492. 120

[75] Coppersmith D., Winograd S. *Matrix multiplication via arithmetic progressions.* J. Symb. Comput. 1990. **9**, 251–280. 115, 121, 122

[76] Crandall R., Fagin B. *Discrete weighted transforms and large-integer arithmetic.* Math. Comp. 1994. **62**(205), 305–324. 79

[77] Demenkov E., Kojevnikov A., Kulikov A., Yaroslavtsev G. *New upper bounds on the Boolean circuit complexity of symmetric functions.* Inform. Proc. Letters. 2010. **110**(7), 264–267. 82, 83

[78] Díaz J., Serna M., Thilikos D. M. *Counting H-colorings of partial k-trees.* Theor. Comput. Sci. 2002, **281**, 291–309. 133

[79] Dinic E. A. *Algorithm for solution of a problem of maximum flow in networks with power estimation.* Soviet Math. Doklady. 1970. **11**, 1277–1280. 131

[80] Dubiner M., Zwick U. *Amplification and percolation (probabilistic Boolean functions).* Proc. FOCS (Pittsburgh, 1992). Los Alamitos: IEEE, 1992, 258–267. 109

[81] Duhamel P., Hollmann H. *Split-radix FFT algorithm.* Electronics Letters. 1984. **20**, 14–16. 85

[82] Dunne P. E. *The complexity of Boolean networks.* San Diego: Academic Press, 1988. 2

[5] Informal Science.

[83] Edmonds J. *Systems of distinct representatives and linear algebra.* J. Research Nat. Bureau Stand. — B. Math. and Math. Phys. 1967. **71B**(4), 241–245. 111

[84] Erdös P. *Remarks on number theory III: On addition chains.* Acta Arithm. 1960. **6**, 77–81. 35

[85] Feder T., Motwani R. *Clique partitions, graph compression and speeding-up algorithms.* J. Comp. System Sci. 1995. **51**, 261–272. 131

[86] Fischer I. *Sums of like powers of multivariate linear forms.* Mathematics Magazine. 1994. **67**(1), 59–61. 157

[87] Fischer M. J. *The complexity of negation-limited networks — a brief survey.* Proc. Automata Theory and Formal Lang. Conf. (Kaiserslautern, 1975). LNCS. 1975. **33**, 71–82. 92

[88] Fischer M. J., Meyer A. R. *Boolean matrix multiplication and transitive closure.* Proc. SWAT (East Lansing, 1971). Washington: IEEE, 1971, 129–131. 69

[89] Floyd R. W. *Algorithm 97, shortest path.* Comm. ACM. 1962. **5**, 345. 21

[90] Fomin S., Grigoriev D., Koshevoy G. *Subtraction-free complexity, cluster transformations, and spanning trees.* Found. Comput. Math. 2016. **15**, 1–31. 19

[91] Ford L. R. *Network flow theory.* The Rand Corp., 1956, Report P-923. 17

[92] Fürer M. *Faster integer multiplication.* SIAM J. Comput. 2009, **39**(3), 979–1005. 73

[93] Furman M. E. *Application of a method of rapid multiplication of matrices to the problem of finding the transitive closure of a graph.* Doklady AN SSSR[6]. 1970. **194**(3), 524. (in Russian) 69

[94] Furst M., Saxe J., Sipser M. *Parity, circuits, and the polynomial time hierarchy.* Math. Syst. Theory. 1984. **17**, 13–27. 141

[95] Gabber O., Galil Z. *Explicit constructions of linear size superconcentrators.* Proc. FOCS (San Juan, Puerto-Rico, 1979). Los Alamitos: IEEE, 1979, 364–370. 62

[96] Galbiati G., Fischer M. J. *On the complexity of 2-output Boolean networks.* Theor. Comput. Sci. 1981. **16**, 177–185. 117

[97] Galil Z., Pan V. *Parallel evaluation of the determinant and of the inverse of a matrix.* Inform. Proc. Letters. 1989. **30**, 41–45. 112

[98] Gao S., von zur Gathen J., Panario D., Shoup V. *Algorithms for exponentiation in finite fields.* J. Symb. Comput. 2000. **29**, 879–889. 81

[99] Gashkov S. B. *On the depth of Boolean functions.* In: Problemy Kibernetiki[7]. Vol. 34. Moscow: Nauka, 1978, 265–268. (in Russian) 126

[100] Gashkov S. B. *On parallel evaluation of certain classes of polynomials with an increasing number of variables.* Moscow Univ. Math. Bulletin. 1990. **45**(2), 64–67. 126

[101] Gashkov S. B. *Remark on minimization of depth of Boolean circuits.* Moscow Univ. Math. Bulletin. 2007. **62**(3), 87–89. 138

[102] Gashkov S. B., Gashkov I. B. *On the complexity of the computation of differentials and gradients.* Discrete Math. and Appl. 2005. **15**(4), 327–350. 100

[103] Gashkov S. B., Grinchuk M. I., Sergeev I. S. *Circuit design of an adder of small depth.* J. Applied and Industrial Math. 2008. **2**(2), 167–178. 30

[104] Gashkov S. B., Kochergin V. V. *On addition chains of vectors, gate circuits, and the complexity of computations of powers.* Siberian Advances in Math. 1994. **4**(4), 1–16. 37

[6] Reports of Academy of Sciences USSR.

[7] Problems of Cybernetics.

[105] Gashkov S. B., Sergeev I. S. *Multiplication.* Chebyshevskii Sbornik[8]. 2020. **21**(1), 101–134. (in Russian) 73

[106] Gashkov S. B., Shavgulidze E. T. *Representation of monomials as a sum of powers of linear forms.* Moscow Univ. Math. Bulletin. 2014. **69**(2), 51–55. 157

[107] von zur Gathen J., Gerhard J. *Modern computer algebra.* Cambridge Univ. Press, 1999. 2

[108] von zur Gathen J., Shoup V. *Computing Frobenius maps and factoring polynomials.* Comput. Complexity. 1992. **2**, 187–224. 81

[109] Gerhard J. *Modular algorithms in symbolic summation and symbolic integration.* Berlin, Heidelberg: Springer–Verlag, 2004. 93

[110] Giesbrecht M., Jamshidpey A., Schost É. *Subquadratic-time algorithms for normal bases.* Comput. Complexity. 2021. **30**, Art. 5. 81

[111] Good I. J. *The interaction algorithm and practical Fourier analysis.* J. R. Statist. Soc. B. 1958. **20**(2), 361–372; 1960. **22**(2), 372–375. 25

[112] Goodrich M. T. *Zig-zag sort: a simple deterministic data-oblivious sorting algorithm running in $O(n \log n)$ time.* Proc. STOC (New York, 2014). NY: ACM, 2014, 684–693. 67

[113] Grigoriev D., Razborov A. A. *Exponential complexity lower bounds for depth 3 arithmetic circuits in algebras of functions over finite fields.* Applicable Algebra in Eng. Comm. Comput. 2000. **10**(6), 465–487. 159

[114] Grinchuk M. I. *Sharpening an upper bound on the adder and comparator depths.* J. Applied and Industrial Math. 2009. **3**(1), 61–67. 29, 96, 97, 99

[115] Gupta A., Kamath P., Kayal N., Saptharishi R. *Arithmetic circuits: A chasm at depth 3.* SIAM J. Comput. 2016. **45**(3), 1064–1079. 156, 157, 158

[116] Gupta A., Mahajan S. *Using amplification to compute majority with small majority gates.* Comput. Complexity. 1996. **6**, 46–63. 109

[117] Grove E. *Proofs with potential.* Ph.D. thesis. Univ. of California, Berkeley, 1993. 49

[118] Hadas T., Schwartz O. *Towards practical fast matrix multiplication based on trilinear aggregation.* Proc. ISSAC (Tromsø, 2023). NY: ACM, 2023, 289–297. 100

[119] Harvey D., van der Hoeven J. *Integer multiplication in time $O(n \log n)$.* Annals of Math. 2021. **193**(2), 563–617. 30, 73, 75

[120] Harvey D., van der Hoeven J. *Polynomial multiplication over finite fields in time $O(n \log n)$.* J. ACM. 2022. **69**(2), Art. 12. 73

[121] Harvey D., van der Hoeven J., Lecerf G. *Faster polynomial multiplication over finite fields.* J. ACM. 2017. **63**(6), Art. 52. 73

[122] Håstad J. *Computational limitations of small-depth circuits.* MIT Press, 1986. 148

[123] Håstad J. *Notes for the course advanced algorithms.* 2000. http://www.csc.kth.se/~johanh/algnotes.pdf 54

[124] Hastad J., Leighton T. *Division in $O(\log n)$ depth using $O(n^{1+\epsilon})$ processors.* 1986. http://www.csc.kth.se/~johanh/paraldivision.pdf 74, 75

[125] Heideman M. T. *Applications of multiplicative complexity theory to convolution and the discrete Fourier transform.* Ph.D. Thesis. Rice Univ., 1986. 26

[126] Held M., Karp R. M. *A dynamic programming approach to sequencing problems.* SIAM J. on Appl. Math. 1962. **10**, 196–210. 18

[127] Hermann A. *Faster circuits for And-Or paths and binary addition.* PhD. Thesis, Univ. Bonn, 2020. 99

[8] Chebyshev's Collection.

[128] Hiltgen A. P., Paterson M. S. *PI_k mass production and an optimal circuit for the Nečiporuk slice.* Comput. Complexity. 1995. **5**(2), 132–154. 123

[129] van der Hoeven J. *Newton's method and FFT trading.* J. Symb. Comput. 2010. **45**(8), 857–878. 55

[130] van der Hoeven J. *Optimizing the half-gcd algorithm.* 2022. arXiv:2212.12389v1. 31

[131] van der Hoeven J., Lecerf G. *Fast multivariate multi-point evaluation revisited.* J. Complexity. 2020. **56**, Art. 101405. 79

[132] Holmgren J., Rothblum R. *Linear-size Boolean circuits for multiselection.* Proc. CCC (Ann Arbor, USA, 2024). LIPIcs. 2024. **300**. Art. 11. 117

[133] Hoover H., Klawe M., Pippenger N. *Bounding fan-out in logical networks.* J. ACM. 1984. **31**(1), 13–18. 138

[134] Hopcroft J. E., Karp R. M. *An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs.* SIAM J. Comput. 1973. **2**(4), 225–231. 128, 131

[135] Jerrum M., Snir M. *Some exact complexity results for straight-line computations over semirings.* J. ACM. 1982. **29**(3), 874–897. 17, 18, 134

[136] Jimbo S., Maruoka A. *A method of constructing selection networks with $O(\log n)$ depth.* SIAM J. Comput. 1996. **25**(4), 709–739. 32

[137] Johnson S. F., Frigo M. *A modified split-radix FFT with fewer arithmetic operations.* IEEE Trans. Signal Process. 2007. **55**(1), 111–119. 87

[138] Jukna S. *Extremal combinatorics: with applications in computer science.* Berlin, Heidelberg: Springer–Verlag, 2011. 39, 144

[139] Jukna S. *Boolean function complexity: advances and frontiers.* Berlin, Heidelberg: Springer–Verlag, 2012. 2, 23

[140] Jukna S. *Tropical circuits complexity: limits of pure dynamic programming.* Berlin, Heidelberg: Springer–Verlag, 2023. 18

[141] Jukna S., Seiwert H. *Greedy can beat pure dynamic programming.* Inform. Proc. Letters. 2019. **142**, 90–95. 19, 21

[142] Jukna S., Sergeev I. *Complexity of linear boolean operators.* Foundations and Trends in Theor. Comput. Sci. 2013. **9**(1), 1–123. 142, 143, 145, 152

[143] Kabatiansky G. A., Panchenko V. I. *Unit sphere packings and coverings of the Hamming space.* Problems of Information Transmission. 1988. **24**(4), 261–272. 125

[144] Kaltofen E., Shoup V. *Subquadratic-time factoring of polynomials over finite fields.* Math. Comput. 1998. **67**(223), 1179–1197. 75, 80

[145] Kaltofen E., Singer M. F. *Size efficient parallel algebraic circuits for partial derivatives.* Proc. Intern. Conf. on Computer Algebra in Psysical Research (Dubna, 1990). Singapore: World Scientific, 1991, 133–145. 104

[146] Karatsuba A. A. *Comments to my works, written by myself.* Proc. Steklov Inst. Math. 2013. **282**, suppl. 1, S1–S23. 73

[147] Karatsuba A. A., Ofman Yu. P. *Multiplication of multidigit numbers on automata.* Soviet Physics Doklady. 1963. **7**, 595–596. 23

[148] Karppa M., Kaski P. *Engineering Boolean matrix multiplication for multiple-accelerator shared-memory architectures.* 2019. arXiv:1909.01554v1. 88

[149] Karstadt E., Schwartz O. *Matrix multiplication, a little faster.* J. ACM. 2020. **67**(1), Art. 1. 87, 88

[150] Karzanov A. V. *An exact time bound for a max-flow finding algorithm applied to the "representatives" problem.* In: Voprosy Kibernetiki[9]. Proc. Seminar on Combinatorial Math. (Moscow, 1971). Moscow: Soviet Radio, 1973, 66–70. (in Russian) 131

[151] Kedlaya K. S., Umans C. *Fast polynomial factorization and modular composition.* SIAM J. Comput. 2011. **40**(6), 1767–1802. 79

[152] Kerr L. R. *The effect of algebraic structure on the computational complexity of matrix multiplication.* PhD. Thesis, Cornell Univ., Ithaca, N.Y., 1970. 24

[153] Khasin L. S. *Complexity bounds for the realization of monotonic symmetrical functions by means of formulas in the basis* $\vee, \&, \neg$. Doklady AN SSSR[10]. 1969. **189**(4), 752–755. (in Russian) 105, 106

[154] Khrapchenko V. M. *Asymptotic estimation of addition time of a parallel adder.* In: Problemy Kibernetiki[11]. Vol. 19. Moscow: Nauka, 1967, 107–120. (in Russian) 29, 30, 52

[155] Khrapchenko V. M. *Method of determining lower bounds for the complexity of P-schemes.* Mathematical Notes. 1971. **10**(1), 474–479. 22, 23, 109

[156] Khrapchenko V. M. *The complexity of the realization of symmetrical functions by formulae.* Mathematical Notes. 1972. **11**(1), 70–76. 92

[157] Khrapchenko V. M. *On a relation between the complexity and the depth of formulas.* Metody Diskr. Analiza v Sinteze Upr. Sistem[12]. Vol. 32. Novosibirsk: Inst. Matem. SO AN SSSR, 1978, 76–94. (in Russian) 52

[158] Khrapchenko V. M. *On a relation between the complexity and the depth of formulas in the basis containing median.* Metody Diskr. Analiza v Izuch. Bul. Funktsii i Grafov[13]. Vol. 37. Novosibirsk: Inst. Matem. SO AN SSSR, 1981, 77–84. (in Russian) 49

[159] Khrapchenko V. M. *On one of the possibilities of sharpening estimates for the delay of a parallel adder.* J. Applied and Industrial Math. 2008. **2**(2), 211–214. 99

[160] Kim K. V., Nesterov Yu. E., Cherkasskii B. V. *An estimate of the effort in computing the gradient.* Soviet Math. Doklady. 1984. **29**, 384–387. 103

[161] Kleiman M., Pippenger N. *An explicit conctructing of short monotone formulae for the monotone symmetric functions.* Theor. Comput. Sci. 1978. **7**(3), 325–332. 106

[162] Klein P., Paterson M. S. *Asymptotically optimal circuit for a storage access function.* IEEE Trans. on Computers. 1980. C-**29**(8), 737–738. 41

[163] Kleitman D. *On Dedekind's problem: the number of monotone Boolean functions.* Proc. AMS. 1969. **21**(3), 677–682. 39

[164] Kloks T. *Treewidth. Computations and approximations.* LNCS. **842**. Berlin: Springer, 1994. 133

[165] Kloss B. M., Malyshev V. A. *Complexity bounds for some classes of functions.* Vestnik Mosk. Univ. Ser. 1. Matematika. Mekhanika[14]. 1965. (4), 44–51. (in Russian). 124

[166] Knuth D. E. *The analysis of algorithms.* Actes du Congrès international des mathèmaticiens. Nice, 1970, **3**, 269–274. 30

[9] Problems of Cybernetics.
[10] Reports of Academy of Sciences USSR.
[11] Problems of Cybernetics.
[12] Methods of discrete analysis in synthesis of control systems.
[13] Methods of discrete analysis in studying Boolean functions and graphs.
[14] Moscow University Bulletin. Ser. 1. Mathematics. Mechanics.

[167] Knuth D. E. *The art of computer programming. Vol.* 2. *Seminumerical algorithms.* Reading: Addison-Wesley, 1997. 2, 24, 30

[168] Knuth D. E. *The art of computer programming. Vol.* 3. *Sorting and searching.* Reading: Addison-Wesley, 1998. 32, 60

[169] Kochergin V. V. *Bellman, Knuth, Lupanov, Pippenger problems and their variations.* In: Matematicheskie Voprosy Kibernetiki[15]. Vol. 20. Moscow: Fizmatlit, 2022, 119–256. (in Russian) 37

[170] Kochergin V. V., Kochergin D. V. *Improvement of the lower bound for the complexity of exponentiation.* Prikladnaya Diskretnaya Matematika[16]. 2017. (38), 119–132. 35

[171] Komarath B., Pandey A., Rahul C. S. *Monotone arithmetic complexity of graph homomorphism polynomials.* Proc. ICALP (Paris, 2022). LIPIcs. 2022. **229**, Art. 83. 134

[172] Korobkov V. K. *Implementation of symmetric functions in the class of π-circuits.* Doklady AN SSSR[17]. 1956. **109**(2), 260–263. (in Russian) 105

[173] Korovin V. V. *On the complexity of the realization of a universal function by circuits consisting of functional elements.* Discrete Math. and Appl. 1995. **5**(3), 249–255. 42

[174] Korshunov A. D. *On the number of monotone Boolean functions.* In: Problemy Kibernetiki[18]. Vol. 38. Moscow: Nauka, 1981, 5–108. (in Russian) 39

[175] Korshunov A. D. *Monotone Boolean functions.* Russian Math. Surveys. 2003. **58**(5), 929–1001. 39

[176] Kosaraju S. R. *Parallel evaluation of division-free arithmetic equations.* Proc. STOC (Berkeley, 1986). NY: ACM, 1986, 231–239. 29, 51

[177] Krichevskii R. E. *On the complexity of parallel-sequential switching circuits implementing a sequence of Boolean functions.* In: Problemy Kibernetiki[19]. Vol. 12. Moscow: Nauka, 1964, 45–55. (in Russian) 105

[178] Kulikov A. S., Melanich O., Mihajlin I. *A $5n - o(n)$ lower bound on the circuit size over U_2 of a linear Boolean function.* Proc. Conf. on Computability in Europe (Cambridge, 2012). LNCS. 2012. **7318**, 432–439. 96

[179] Kulikov A. S., Mikhailin I., Mokhov A., Podolskii V. *Complexity of linear operators.* Proc. ISAAC (Shanghai, 2019). LIPIcs. 2019. **149**, Art. 17. 112, 114

[180] Kung H. T. *Fast evaluation and interpolation.* Tech. report. Carnegie-Mellon Univ., Pittsburgh, 1973. 76, 77

[181] Laderman J. D. *A noncommutative algorithm for multiplying 3×3 matrices using* 23 *multiplications.* Bull. AMS. 1976, **82**, 126–128. 60

[182] Ladner R. E., Fischer M. J. *Parallel prefix computation.* J. ACM. 1980. **27**(4), 831–838. 26, 27

[183] Lamagna E. A., Savage J. E. *Combinational complexity of some monotone functions.* Proc. SWAT (New Orleans, 1974). New Orleans: IEEE, 1974, 140–144. 60

[184] Landsberg J. M. *The border rank of the multiplication of 2×2 matrices is seven.* J. AMS. 2006. **19**(2), 447–459. 60

[185] Lehmer D. H. *Euclid's algorithm for large numbers.* Amer. Math. Monthly. 1938. **45**, 227–233. 31

[15] Mathematical Problems of Cybernetics.
[16] Applied Discrete Mathematics.
[17] Reports of Academy of Sciences USSR.
[18] Problems of Cybernetics.
[19] Problems of Cybernetics.

[186] van Leijenhorst D. C. *A note on the formula size of the "mod k" functions.* Inform. Proc. Letters. 1987. **24**, 223–224. 70

[187] Limaye N., Srinivasan S., Tavenas S. *Superpolynomial lower bounds against low-depth algebraic circuits.* Proc. FOCS (virtually, 2022). Los Alamitos: IEEE, 2022, 804–814. 141, 158

[188] Linnainmaa S. *Taylor expansion of the accumulated rounding error.* Nordisk Tidskr. Informationsbehandling (BIT). 1976. **16**(2), 146–160. 103

[189] Lovász L. *On determinants, matchings, and random algorithms.* Proc. Conf. Algebr. Arithm. Categ. Methods in Comput. Theory (Berlin/Wendisch-Rietz, 1979). Berlin: Akademie, 1979, 565–574. 111

[190] Lovett S. *Computing polynomials with few multiplications.* Theory of Computing. 2011. **7**, 185–188. 127

[191] Lozhkin S. A. *On a relationship between the depth and complexity of equivalent formulas and on the depth of monotone functions of Boolean algebra.* In: Problemy Kibernetiki[20]. Vol. 38. Moscow: Nauka, 1981, 269–271. (in Russian) 137

[192] Lozhkin S. A. *On the depth of functions of Boolean algebra in some bases.* Annales Univ. Budapest. Sec. Computatorica. 1983. IV, 113–125. (in Russian) 126

[193] Lozhkin S. A. *High-precision estimates for the complexity of control systems from some classes.* In: Matematicheskie Voprosy Kibernetiki[21]. Vol. 6. Moscow: Fizmatlit, 1996, 189–214. (in Russian) 37, 126

[194] Lozhkin S. A. *On minimal π-circuits of closing contacts for symmetric functions with threshold* 2. Discrete Math. and Appl. 2005. **15**(5), 475–477. 105

[195] Lozhkin S. A. *Synthesis of formulas whose complexity and depth do not exceed the asymptotically best estimates of high degree of accuracy.* Moscow Univ. Math. Bulletin. 2007. **62**(3), 101–107. 126

[196] Lozhkin S. A. *Refined bounds on Shannon's function for complexity of circuits of functional elements.* Moscow Univ. Math. Bulletin. 2022. **77**(3), 144–153. 37

[197] Lozhkin S. A., Vlasov N. V. *On multiplexer function complexity in the π-schemes class.* Kazan. Gos. Univ. Uchen. Zap. Ser. Fiz.-Mat. Nauki[22]. 2009. **151**(2), 98–106. (in Russian) 42

[198] Lozhkin S. A., Semenov A. A. *On a method for compressing information and on the complexity of the realization of monotone symmetric functions.* Soviet Math. (Iz. VUZ). 1988. **32**(7), 73–85. 109

[199] Lundy T. J., van Buskirk J. *A new matrix approach to real FFTs and convolutions of length 2^k.* Computing. 2007. **80**, 23–45. 85

[200] Lupanov O. B. *On rectifier and switching-and-rectifier schemes.* Doklady AN SSSR[23]. 1956. **111**(6), 1171–1174. (in Russian) 35, 36, 142

[201] Lupanov O. B. *A method of circuit synthesis.* Izv. VUZ. Radiofizika[24]. 1958. **1**(1), 120–140. (in Russian) 35, 36, 37

[202] Lupanov O. B. *On the synthesis of contact schemes.* Doklady AN SSSR[25]. 1958. **119**(1), 23–26. (in Russian) 42, 125

[20] Problems of Cybernetics.
[21] Mathematical Problems of Cybernetics.
[22] Scientific Notes of Kazan State University. Ser. Physical and Mathematical Sciences.
[23] Reports of Academy of Sciences USSR.
[24] News of universities. Radiophysics.
[25] Reports of Academy of Sciences USSR.

[203] Lupanov O. B. *On the complexity of realization of functions of propositional calculus by formulas.* In: Problemy Kibernetiki[26]. Vol. 3. Moscow: Fizmatgiz, 1960, 61–80. (in Russian) 125, 126

[204] Lupanov O. B. *On realization of functions of propositional calculus by formulas from finite classes (formulas of bounded depth) over the basis* $\&, \vee, \bar{}$. In: Problemy Kibernetiki[27]. Vol. 6. Moscow: Fizmatlit, 1961, 5–14. (in Russian) 137

[205] Lupanov O. B. *On a certain approach to the synthesis of control systems — the principle of local coding.* In: Problemy Kibernetiki[28]. Vol. 14. Moscow: Nauka, 1965, 31–110. (in Russian) 14, 39, 41

[206] Lupanov O. B. *On computing symmetric functions of propositional calculus by switching networks.* In: Problemy Kibernetiki[29]. Vol. 15. Moscow: Nauka, 1965, 85–99. (in Russian) 43, 89

[207] Lupanov O. B. *Asymptotic estimates of the complexity of control systems.* Moscow: Izd. MGU, 1984. (in Russian) 92, 125

[208] Lupanov O. B. *On the complexity of modeling powers of Boolean* (n, n)*-functions.* In: Matematicheskie Voprosy Kibernetiki[30]. Vol. 12. Moscow: Fizmatlit, 2003, 179–216. (in Russian) 117

[209] Lupanov O. B. *A. N. Kolmogorov and the theory of circuit complexity.* In: Matematicheskie Voprosy Kibernetiki[31]. Vol. 17. Moscow: Fizmatlit, 2008, 5–12. (in Russian) 71

[210] Mahajan M., Nimbhorkar P., Tawari A. *Shortest path length with bounded-alternation* $(\min, +)$ *formulas.* Int. J. Advances in Engin. Sci. and Appl. Math. 2019. **11**, 68–74. 159

[211] Mahler K., Popken J. *On a maximum problem in arithmetic.* Nieuw Archief voor Wiskunde (ser. 3). 1953. **1**(3), 1–15. 15

[212] Margulis G. A. *Explicit constructions of concentrators.* Problems Inform. Transmission. 1973. **9**(4), 325–332. 62

[213] Markov A. A. *On the inversion complexity of systems of Boolean functions.* J. ACM. 1958. **5**(4), 331–334. 92

[214] Martens J.-B. *Recursive cyclotomic factorization — a new algorithm for calculating the discrete Fourier transform.* IEEE Trans. on Acoustics, Speech, and Signal Processing. 1984. **32**, 750–761. 85

[215] Massey J. L., Omura J. K. *Apparatus for finite fields computation.* US patent application. 1986. No. 4587627. 81

[216] McColl W. F. *Some results on circuit depth.* Theory of computation. Report no. 18. Coventry, Univ. of Warwick, 1977. 70

[217] Mitiagin B. S., Sadovskii B. N. *On linear Boolean operators.* Doklady AN SSSR[32]. 1965. **165**(4), 773–776. (in Russian) 95

[218] Moenck R. T. *Fast computation of GCDs.* Proc. STOC (Austin, 1973). NY: ACM, 1973, 142–151. 31

[26] Problems of Cybernetics.
[27] Problems of Cybernetics.
[28] Problems of Cybernetics.
[29] Problems of Cybernetics.
[30] Mathematical Problems of Cybernetics.
[31] Mathematical Problems of Cybernetics.
[32] Reports of Academy of Sciences USSR.

[219] Möller N. *On Schönhage's algorithm and subquadratic integer GCD computation.* Math. Comp. 2008. **77**, 589–607. 31

[220] Montgomery P. L. *Modular multiplication without trial division.* Math. Comp. 1985. **44**, 519–521. 90, 91

[221] Moore E. F. *The shortest path through a maze.* Proc. Intern. Sympos. on Theory of Switching (Cambridge, USA, 1957). Part II. Cambridge: Harvard Univ. Press, 1959, 285–292. 17

[222] Moore E. F., Shannon C. E. *Reliable circuits using less reliable relays.* J. of the Franklin Institute. 1956. **262**(3), 191–208; **262**(4), 281–297. 109

[223] Morizumi H., Suzuki G. *Negation-limited inverters of limited size.* IEICE Trans. Inf. Syst. 2010. E**93**-D(2), 257–262. 93

[224] Muller D. E. *Complexity in electronic switching circuits.* IRE Trans. Comput. 1956. EC-**5**(1), 15–19. 35

[225] Nechiporuk E. I. *On the complexity of schemes in some bases containing nontrivial elements with zero weights.* In: Problemy Kibernetiki[33]. Vol. 8. Moscow: Fizmatlit, 1962, 123–160. (in Russian) 135, 136

[226] Nechiporuk E. I. *Rectifier networks.* Soviet Physics Doklady. 1963. **8**, 5–7. 37, 38, 114

[227] Nechiporuk E. I. *On the synthesis of logical networks in incomplete and degenerate bases.* In: Problemy Kibernetiki[34]. Vol. 14. Moscow: Nauka, 1965, 111–160. (in Russian) 136

[228] Nechiporuk E. I. *On topological principles of self-correction.* In: Problemy Kibernetiki[35]. Vol. 21. Moscow: Nauka, 1969, 5–102. (in Russian) 37, 114

[229] Nechiporuk E. I. *On a Boolean matrix.* In: Problemy Kibernetiki[36]. Vol. 21. Moscow: Nauka, 1969, 237–240. (in Russian) 123

[230] Nigmatullin R. G. *The complexity of Boolean functions.* Moscow: Nauka, 1991. 2

[231] Ofman Yu. P. *On the algorithmic complexity of discrete functions.* Soviet Physics Doklady. 1963. **7**, 589–591. 28

[232] Oliveira I. C., Santhanam R., Srinivasan S. *Parity helps to compute majority.* Proc. Comput. Compl. Conf. (New Brunswick, 2019). LIPIcs. 2019. **137**, Art. 23. 148, 149, 151

[233] Pan V. Ya. *Some schemes for evaluation of polynomials with real coefficients.* Doklady AN SSSR[37]. 1959. **127**(2), 266–269. (in Russian) 32

[234] Pan V. Ya. *Some schemes for evaluation of polynomials with real coefficients.* In: Problemy Kibernetiki[38]. Vol. 5. Moscow: Fizmatlit, 1961, 17–30. (in Russian) 32

[235] Pan V. Ya. *On some ways of evaluation polynomials.* In: Problemy Kibernetiki[39]. Vol. 7. Moscow: Fizmatlit, 1962, 21–30. (in Russian) 32

[236] Pan V. Ya. *Computation schemes for a product of matrices and for the inverse matrix.* Uspekhi Matem. Nauk[40]. 1972. **27**(5), 249–250. (in Russian) 99

[33] Problems of Cybernetics.
[34] Problems of Cybernetics.
[35] Problems of Cybernetics.
[36] Problems of Cybernetics.
[37] Reports of Academy of Sciences USSR.
[38] Problems of Cybernetics.
[39] Problems of Cybernetics.
[40] Advances in Mathematical Sciences.

[237] Pan V. Ya. *Trilinear aggregating with implicit canceling for a new acceleration of matrix multiplication.* Comput. Math. Appl. 1982. **8**(1), 23–34. 100

[238] Pan V. Y. *Structured matrices and polynomials: unified superfast algorithms.* Boston: Birkhäuser, 2001. 2

[239] Pan V. Ya. *Fast matrix multiplication and its algebraic neighbourhood.* Sbornik Mathematics. 2017. **208**(11), 1661–1704. 100

[240] Paterson M. S. *Complexity of monotone networks for Boolean matrix product.* Theor. Comput. Sci. 1975. **1**, 13–20. 24, 69

[241] Paterson M. S. *Improved sorting networks with $O(\log N)$ depth.* Algorithmica. 1990. **5**(1), 75–92. 60, 62, 67

[242] Paterson M. S., Pippenger N., Zwick U. *Faster circuits and shorter formulae for multiple addition, multiplication and symmetric Boolean functions.* Proc. FOCS (St. Louis, 1990). Washington: IEEE, 1990, 642–650. 92

[243] Paterson M. S., Pippenger N., Zwick U. *Optimal carry save networks.* LMS Lecture Notes Series. Boolean function complexity. Vol. 169. Cambridge Univ. Press, 1992, 174–201. 45, 46, 47, 49

[244] Paterson M. S., Stockmeyer L. J. *On the number of nonscalar multiplications necessary to evaluate polynomials.* SIAM J. Comput. 1973. **2**, 60–66. 33, 127

[245] Paterson M., Zwick U. *Shallow circuits and concise formulae for multiple addition and multiplication.* Comput. Complexity. 1993. **3**, 262–291. 49

[246] Paul W. J. *Realizing Boolean functions on disjoint sets of variables.* Theor. Comput. Sci. 1976. **2**, 383–396. 115

[247] Paul W. J. *A 2.5n-lower bound on the combinational complexity of Boolean functions.* SIAM J. Comput. 1977. **6**(3), 427–443. 42

[248] Pippenger N. *The minimum number of edges in graphs with prescribed paths.* Math. Systems Theory. 1979. **12**, 325–346. 38

[249] Pippenger N. *On the evaluation of powers and monomials.* SIAM J. Comput. 1980. **9**(2), 230–250. 37, 39

[250] Pippenger N. *A formula for the determinant.* 2022. arXiv:2206.00134v1. 110

[251] Preparata F. P., Muller D. E. *Efficient parallel evaluation of Boolean expressions.* IEEE Trans. Comp. 1976. C-**25**(5), 548–549. 51

[252] Probert R. L. *On the additive complexity of matrix multiplication.* SIAM J. Comput. 1976. **5**(2), 187–203. 24

[253] Prüfer H. *Neuer Beweis eines Satzes über Permutationen.* Arch. Math. Phys. 1918. **27**, 742–744. 21

[254] Rabin M. O., Winograd S. *Fast evaluation of polynomials by rational preparation.* IBM Res. Rep. RC 3645. N.Y.: Yorktown Heights, 1971. 32, 33

[255] Radhakrishnan J. *Entropy and counting.* IIT Kharagpur, Golden Jubilee Vol. on Comput. Math., Modelling and Algorithms. New Delhi: Narosa Publ., 2001. 105

[256] Razborov A. A. *Lower bounds on monotone complexity of the logical permanent.* Mathematical Notes. 1985. **37**(6), 485–493. 131

[257] Red'kin N. P. *Proof of minimality of circuits consisting of functional elements.* In: Problemy Kibernetiki[41]. Vol. 23. Moscow: Nauka, 1970, 83–101. (in Russian) 12

[258] Red'kin N. P. *On the implementation of monotone functions by contact circuits.* In: Problemy Kibernetiki[42]. Vol. 35. Moscow: Nauka, 1979, 87–110. (in Russian) 39, 40

[41] Problems of Cybernetics.

[42] Problems of Cybernetics.

[259] Red'kin N. P. *On the minimal implementation of a binary adder.* In: Problemy Kibernetiki[43]. Vol. 38. Moscow: Nauka, 1981, 181–216. (in Russian) 13, 82, 84

[260] Redkin N. P. *The generalized complexity of linear Boolean functions.* Discrete Math. and Appl. 2020. **30**(1), 39–44. 12

[261] Reif J., Tate S. *Optimal size integer division circuits.* SIAM J. Comput. 1990. **19**(5), 912–925. 75

[262] Riordan J., Shannon C. *The number of two-terminal series-parallel networks.* J. of Math. and Phys. 1942. **21**(2), 83–93. 126

[263] Rokhlina M. M. *Circuits that increase reliability.* In: Problemy Kibernetiki[44]. Vol. 23. Moscow: Nauka, 1970, 295–301. (in Russian) 109

[264] Rosser J. B., Schoenfeld L. *Approximate formulas for some functions of prime numbers.* Illinois J. Math. 1962. **6**, 64–94. 74, 112

[265] Roy B. *Transitivité et connexité.* C. R. Acad. Sci. Paris. 1959. **249**, 216–218. 21

[266] Rumyantsev P. V. *On the complexity of implementing a multiplexor function by circuits of functional elements.* Proc. XIV Conf. "Problems of Theoretical Cybernetics" (Penza, 2005). Moscow: Izd. Mech.-Mat. Fac. MGU, 2005, 133. (in Russian) 42

[267] Rychkov K. L. *Lower bounds on the complexity of parallel-sequential switching circuits that realize linear Boolean functions.* Sibirsk. Zh. Issled. Oper.[45] 1994. **1**(4), 33–52. (in Russian) 22

[268] Rychkov K. L. *On the lower bounds of formula complexity of the linear Boolean function.* Sibirsk. Elektr. Matem. Izv.[46] 2014. **11**, 165–184. (in Russian) 22

[269] Ryser H. J. *Combinatorial Mathematics.* NY: Wiley, 1963. 157

[270] Sapozhenko A. A. *On the complexity of disjunctive normal forms obtained with a gradient algorithm.* In: Diskretnyj Analiz[47]. Vol. 21. Novosibirsk: Inst. Matem. SO AN SSSR, 1972, 62–71. 144

[271] Sapozhenko A. A. *Dedekind's problem and the method of border functionals.* In: Matematicheskie Voprosy Kibernetiki[48]. Vol. 9. Moscow: Fizmatlit, 2000, 161–220. (in Russian) 39

[272] Saptharishi R. et al. *A survey of lower bounds in arithmetic circuit complexity.* Github survey. 2014–2021. ver. 9.0.3. https://github.com/dasarpmar/lowerbounds-survey 157

[273] Saxena N. *Diagonal circuit identity testing and lower bounds.* Proc. ICALP (Reykjavik, 2008). LNCS. **5126**. Springer, 2008, 60–71. 157

[274] Schnorr C. P. *Zwei lineare untere Schranken für die Komplexität Boolescher Funktionen.* Computing. 1974. **13**(2), 155–171. 12

[275] Schnorr C. P. *A lower bound on the number of additions in monotone computations.* Theor. Comput. Sci. 1976, **2**, 305–315. 134

[276] Schönhage A. *Schnelle Berechnung von Kettenbruchentwicklungen.* Acta Inf. 1971. **1**, 139–144. 30

[277] Schönhage A. *A lower bound for the length of addition chains.* Theor. Comput. Sci. 1975, **1**, 1–12. 35

[43] Problems of Cybernetics.
[44] Problems of Cybernetics.
[45] Siberian Journal of Operations Research.
[46] Siberian Electronic Mathematical News.
[47] Discrete Analysis.
[48] Mathematical Problems of Cybernetics.

[278] Schönhage A. *Partial and total matrix multiplication.* SIAM J. Comput. 1981. **10**(3), 434–455. 59, 60, 117, 118, 120

[279] Schönhage A. *Fast reduction and composition of binary quadratic forms.* Proc. ISSAC (Bonn, 1991). NY: ACM, 1991, 128–133. 31

[280] Schönhage A., Strassen V. *Schnelle Multiplikation großer Zahlen.* Computing. 1971. **7**(3–4), 271–282. 71, 73

[281] Schwartz J. T. *Fast probabilistic algorithms for verification of polynomial identities.* J. ACM. 1980. **27**(4), 701–717. 111

[282] Seiferas J. *Sorting networks of logarithmic depth, further simplified.* Algorithmica. 2009. **53**(3), 374–384. [see also: Seiferas J. *AKS sorting networks.* Manuscript, 2017. available at http://www.researchgate.net/profile/Joel-Seiferas] 60, 67

[283] Selezneva S. N. *Lower bound on the complexity of finding polynomials of Boolean functions in the class of circuits with separated variables.* Computational Math. and Modeling. 2013. **24**(1), 146–152. 143

[284] Selezneva S. N. *On the length of Boolean functions in the class of exclusive-OR sums of pseudoproducts.* Moscow Univ. Comput. Math. and Cybern. 2014. **38**(2), 64–68. 145

[285] Selezneva S. N. *On the multiplicative complexity of Boolean functions.* Fundamenta Informaticae. 2016. **145**, 399–404. 136

[286] Selezneva S. N. *Order of the length of Boolean functions in the class of exclusive-OR sums of pseudoproducts.* Moscow Univ. Comput. Math. and Cybern. 2016. **40**(3), 123–127. 145, 147

[287] Sergeev I. S. *On the depth of circuits for multiple addition and multiplication of numbers.* Proc. Youth Scientific School on Discrete Math. and its Appl. (Moscow, 2007). Part II. Moscow: Izd. IPM RAN, 2007, 40–45. (in Russian) 138, 139, 140

[288] Sergeev I. S. *On constructing circuits for transforming the polynomial and normal bases of finite fields from one to the other.* Discrete Math. and Appl. 2007. **17**(4), 361–373. 80, 81

[289] Sergeev I. S. *On the complexity of the gradient of a rational function.* J. Applied and Industrial Math. 2008. **2**(3), 385–396. 101, 103, 104

[290] Sergeev I. S. *Fast algorithms for elementary operations on complex power series.* Discrete Math. and Appl. 2010. **20**(1), 25–60. 57

[291] Sergeev I. S. *Minimal parallel prefix circuits.* Moscow Univ. Math. Bulletin. 2011. **66**(5), 215–218. 28

[292] Sergeev I. S. *On the complexity of parallel prefix circuits.* ECCC report TR13–041. 2013. 28

[293] Sergeev I. S. *Upper bounds on the depth of symmetric Boolean functions.* Moscow Univ. Comput. Math. and Cybern. 2013. **37**(4), 195–201. 92

[294] Sergeev I. S. *Upper bounds for the formula size of symmetric Boolean functions.* Russian Mathematics. 2014. **58**(5), 30–42. 92

[295] Sergeev I. S. *On the circuit complexity of the standard and the Karatsuba methods of multiplying integers.* 2016. arXiv:1602.02362. 24

[296] Sergeev I. S. *Complexity and depth of formulas for symmetric Boolean functions.* Moscow Univ. Math. Bulletin. 2016. **71**(3), 127–130. 92

[297] Sergeev I. S. *Upper bounds for the size and the depth of formulae for MOD-functions.* Discrete Math. and Appl. 2017. **27**(1), 15–22. 43, 44, 71, 89, 90

[298] Sergeev I. S. *Rectifier circuits of bounded depth.* J. Applied and Industrial Math. 2018. **12**(1), 153–166. 38

[299] Sergeev I. S. *On the complexity of bounded-depth circuits and formulas over the basis of [unbounded] fan-in gates.* Discrete Math. and Appl. 2019. **29**(4), 241–254. 148

[300] Sergeev I. S. *On a relation between the depth and complexity of monotone Boolean functions.* J. Applied and Industrial Math. 2019. **13**(4), 746–752. 52

[301] Sergeev I. S. *On the complexity of monotone circuits for threshold symmetric Boolean functions.* Discrete Math. and Appl. 2021. **31**(5), 345–366. 32, 125

[302] Sergeev I. S. *Formula complexity of a linear function in a k-ary basis.* Mathematical Notes. 2021. **109**(3), 445–458. 45

[303] Sergeev I. S. *Notes on the complexity of coverings for Kronecker powers of symmetric matrices.* 2022. arXiv:2212.01776v1. 143

[304] Sergeev I. S. *On the multiplicative complexity of polynomials.* Discrete Math. and Appl. 2024. **34**(1), 29–32. 128

[305] Sergeev I. S. *On the additive complexity of some integer sequences.* Mathematical Notes. 2024. **115**(3), 378–389. 39

[306] Sergeev I. S. *Economical formulae for symmetric boolean functions.* 2025. https://igorssergeev.github.io/papers/symm2025eng.pdf 70, 92, 109

[307] Shannon C. E. *The synthesis of two-terminal switching circuits.* Bell Systems Technical J. 1949. **28**(1), 59–98. 35

[308] Smirnov A. V. *The bilinear complexity and practical algorithms for matrix multiplication.* Comput. Math. and Math. Physics. 2013. **53**(12), 1781–1795. 60, 88, 100

[309] Stehlé D., Zimmermann P. *A binary recursive GCD algorithm.* Proc. ANTS (Burlington, USA, 2004). LNCS. 2004. **3076**, 411–425. 31

[310] Stein J. *Computational problems associated with Racah algebra.* J. Comput. Physics. 1967. **1**, 397–405. 13

[311] Stein S. K. *Two combinatorial covering problems.* J. Combin. Theory (A). 1974. **16**, 391–397. 144

[312] Stockmeyer L. J. *On the combinational complexity of certain symmetric Boolean functions.* Math. Systems Theory. 1977. **10**, 323–336. 84

[313] Strassen V. *Gaussian elimination is not optimal.* Numer. Math. 1969. **13**(4), 354–356. 24, 69, 87, 103, 112

[314] Strassen V. *Die Berechnungskomplexität von elementarsymmetrischen Funktionen und von Interpolationskoeffizienten.* Numer. Math. 1973. **20**, 238–251. 54, 55, 153

[315] Strassen V. *Relative bilinear complexity and matrix multiplication.* J. für die reine und angewandte Math. 1987. **1987**(375–376), 406–443. 121

[316] Tanaka K., Nishino T. *On the complexity of negation-limited Boolean networks.* Proc. STOC (Montreal, 1994). NY: ACM, 1994, 38–47. 93

[317] Tardos É. *The gap between monotone and non-monotone circuit complexity is exponential.* Combinatorica. 1987. **7**(4), 141–142. 131

[318] Tavenas S. *Improved bounds for reduction to depth* 4 *and depth* 3. Inform. and Comput. 2015. **240**, 2–11. 152, 155, 156, 157, 158

[319] Tkachov G. A. *On complexity of realization of a sequence of Boolean functions by circuits of functional elements and π-schemes under additional restrictions on the structure of the circuits.* In: Kombinatorno-Algebraicheskie Metody v Prikladnoj Matematike[49]. Gorky: Izd. Gork. Univ., 1980, 161–207. 141

[49] Combinatorial-Algebraic Methods in Applied Mathematics.

[320] Toom A. L. *The complexity of a scheme of functional elements simulating the multiplication of integers.* Soviet Math. Doklady. 1963. **3**, 714–716. 24, 71

[321] Ugol'nikov A. B. *On the implementation of monotone functions by circuits of functional elements.* In: Problemy Kibernetiki[50]. Vol. 31. Moscow: Nauka, 1976, 167–185. (in Russian) 39

[322] Uhlig D. *On the synthesis of self-correcting schemes from functional elements with a small number of reliable elements.* Mathematical Notes. 1974. **15**(6), 558–562. 116

[323] Uhlig D. *Zur Parallelberechnung Boolescher Funktionen.* TR Ing. Hochschule Mittweida, 1984. 117

[324] Uhlig D. *Networks computing Boolean functions for multiple input values.* LMS Lecture Notes Series. Boolean function complexity. Vol. 169. Cambridge Univ. Press, 1992, 165–173. 117

[325] Umans C. *Fast polynomial factorization and modular composition in small characteristic.* Proc. STOC (Victoria, Canada, 2008). NY: ACM, 2008, 481–490. 76, 79, 80

[326] Valiant L. G. *Short monotone formulae for the majority function.* J. Algorithms. 1984. **5**, 363–366. 92, 106, 149

[327] Valiant L. G., Skyum S., Berkowitz S., Rackoff C. *Fast parallel computation of polynomials using few processors.* SIAM J. Comput. 1983. **12**(4), 641–644. 153

[328] Vetterli M., Nussbaumer H. J. *Simple FFT and DCT algorithms with reduced number of operations.* Signal Processing. 1984, **6**, 262–278. 85

[329] Warshall S. *A theorem on boolean matrices.* J. ACM. 1962. **9**, 11–12. 21

[330] Wegener I. *More on the complexity of slice functions.* Theor. Comput. Sci. 1986. **43**, 201–211. 123

[331] Wegener I. *The complexity of Boolean functions.* Stuttgart: Wiley–Teubner, 1987. 2, 23, 117, 124, 125, 131

[332] Winograd S. *On multiplication of* 2×2 *matrices.* Linear Algebra and Appl. 1971. **4**, 381–388. 24, 57

[333] Winograd S. *On the multiplicative complexity of the discrete Fourier transform.* Advances in Math. 1979. **32**(2), 83–117. 26

[334] Yablonskii S. V. *Realisation of the linear function in the class of* Π*-schemes.* Doklady AN SSSR[51]. 1954. **94**(5), 805–806. (in Russian) 22

[335] Yablonskii S. V. *Introduction to discrete mathematics.* Moscow: Nauka, 1986. (in Russian) 96

[336] Yablonskii S. V., Kozyrev V. P. *Mathematical problems of cybernetics.* In: Information Materials of Scientific Council of Acad. Sci. SSSR on Complex Problem "Kibernetika"[52]. Vol. 19a. Moscow, 1968, 3–15. (in Russian) 49

[337] Yao A. C. *A study of concrete computational complexity.* Ph.D. thesis. Tech. Report UIUCDCS-R-75-716. Urbana-Champaign, Univ. Illinois, 1975. 32

[338] Yavne R. *An economical method for calculating the discrete Fourier transform.* Proc. Fall Joint Computer Conf. (San Francisco, 1968). Part I. NY: ACM, 1968, 115–125. 84, 85

[339] Zelinsky J. *Upper bounds on integer complexity.* 2022. arXiv:2211.02995v1. 15

[50] Problems of Cybernetics.
[51] Reports of Academy of Sciences USSR.
[52] Cybernetics.

[340] Zhdanovich D. V. *The matrix capacity of a tensor.* J. of Math. Sciences. 2012. **186**(4), 599–643. 122