Combinatorial Methods in Circuit Complexity

S. Jukna

This text contains some additional topics with application of combinatorial methods in the theory of computing. Together with applications presented in the book, these topics may be used, for example, in a course focused on *Concrete Complexity* (proving lower bounds on the size of different circuit models). The text will grow. The topics appear in oder the corresponding text is appended. This is the version of: `September 24, 2002`.

Johann Wolfgang Goethe-Universität, Frankfurt am Main, D-60054, Germany

Email: jukna@thi.informatik.uni-frankfurt.de

Contents

TOPIC 1
Boolean Formulas: The Classics

(Related sections: 15.2.2., 20)

Summary: Every formula can be made balanced without increasing its size. Random restrictions can sustantially reduce the formula size. An $\Omega(n^{5/2})$ lower bound on the formula size. Nechiporuk's lower bound for general formulas.

Recall that a *DeMorgan formula* (or just a *formula*) is a circuit with And, Or and Not gates, whose underlying graph is a tree. Such formulas can be defined inductively as follows: (i) every boolean variable x_i and its negation $\overline{x}_i$ is a formula of size 1 (these formulas are called *leaves*); (ii) if F_1 and F_2 are formulas of size l_1 and l_2, then both $F_1 \wedge F_2$ and $F_1 \vee F_2$ are formulas of size $l_1 + l_2$.

Let $|F|$ denote the size of a formula F (i.e. the number of leaves in F). By $L(f)$ we denote the minimal size and by $D(f)$ the minimal depth[1] of a DeMorgan formula computing f.

1.1. Size versus depth

Let F be a DeMorgan formula. Since its underlying graph is a binary tree, we have $|F| \leq 2^{D(F)}$. This implies that, for every function f,

$$D(f) \geq \log L(f).$$

In fact, we have also a converse inequality:

Theorem 1.1 (Spira 1971). *For any f, $D(f) = O(\log L(f))$.*

Proof. We must show how, given a formula F of size ℓ, we can reduce its depth to $O(\log \ell)$. The idea is to separate the leaves of a formula into two formulas each of which has at most $\frac{2}{3}$ of the original size. Then recurse on the subtrees to get log depth.

Let $D(\ell)$ denote the minimal number such that every formula F of size at most ℓ has an equivalent[2] formula F' of depth at most $D(\ell)$. We will argue by induction on ℓ to show that

$$D(\ell) \leq 2\log_{3/2} \ell = O(\log_2 \ell). \qquad (*)$$

[1] The depth of a formula is, of course, the depth of its underlying tree.

[2] Two formulas are *equivalent* if they compute the same boolean function.

Let F be an arbitrary formula of size ℓ. By Exercise 4.14 from the book, we can find a node v in the underlying graph of F such that

$$\frac{\ell}{3} \leq |F_v| < \frac{2\ell}{3}.$$

Replacing the subformula F_v by contants 0 and 1 we obtain two formulas $F_{v\to 0}$ and $F_{v\to 1}$, each of size

$$|F| - |F_v| \leq \ell - \frac{\ell}{3} = \frac{2\ell}{3}.$$

Consider the following formula

$$F' = \mathrm{sel}(F_v, F_{v\to 0}, F_{v\to 1})$$

where $\mathrm{sel}(x,y,z) = \overline{x}y \vee xz$ is the *selector* function, which outputs y if $x = 0$, and outputs z if $x = 1$. The formula F' computes the same boolean function as F and has depth at most

$$2 + \max\{D(F_v), D(F_{v\to 0}), D(F_{v\to 1})\} \leq 2 + D\left(\frac{2\ell}{3}\right),$$

where by the induction hypothesis (*),

$$D\left(\frac{2\ell}{3}\right) \leq 2\log_{3/2}\left(\frac{2\ell}{3}\right) = 2\left(\log_{3/2}\ell - 1\right).$$

Thus,

$$D(\ell) \leq 2 + D\left(\frac{2\ell}{3}\right) \leq 2\log_{3/2}\ell.$$

□

1.2. The effect of random restrictions

Already in 1961, Subbotovskaya has found an argument to show that some boolean functions require boolean formulas of super-linear size. Her idea was, given a formula F computing some function f, to set randomly some of the variables to constants and show that this restriction reduces the size of F considerably whereas the resulting subfunction of f is not much "easier."

Let us recall some notation. A *partial assignment* (or *restriction*) is a function $\rho : [n] \to \{0, 1, *\}$, where we understand $*$ to mean that the corresponding variable is unassigned. Given a boolean function $f(x_1, \ldots, x_n)$, $f|_\rho$ indicates the resulting *restriction* of f. Let $\mathcal{R}_k$ be the set of all partial assignments which leave exactly k variables unassigned. What we will be interested in is the *random* restrictions $f|_\rho$ that results from choosing a *random* partial assignment from $\mathcal{R}_k$.

The probability distribution of restrictions in $\mathcal{R}_k$ we will use is the following: randomly assign k variables to be $*$, and assign all other variables to be 0 or 1 randomly and independently.

Intuitively, a random restriction should reduce considerably the size of a formula. The following lemma makes this intuition precise.

Lemma 1.2 (Subbotovskaya 1961). *Let f be a boolean function of n variables, and let ρ be a random restriction from $\mathcal{R}_k$. Then, with probability at least $\frac{3}{4}$,*

$$L(f|_\rho) \leq 4 \cdot \left(\frac{k}{n}\right)^{3/2} \cdot L(f).$$

Proof. Let F be an optimal DeMorgan formula for the function f of size $s = L(f)$. Construct the restriction ρ in $n - k$ stages as follows:

- At any stage, choose a variable randomly from the remaining ones, and assign it 0 or 1 randomly.

We analyse the effect of this restriction to the formula F, stage-by-stage.

Suppose the first stage chooses the variable x_i. When this variable is set to a constant, then all the input gates $e \in F$, labeled by the literals x_i and $\overline{x}_i$ will disappear from the formula F. By averaging, the expected number of such literals is s/n.

In fact, the formula is likely to be reduced even further. For each of the input gates e, labeled by x_i or $\overline{x}_i$, consider the gate which e feeds into. For example, suppose the gate is $e \wedge G$ for some subformula G. We may assume w.l.o.g. that G does not contain the literals x_i or $\overline{x}_i$.

Ex. 1.3. Why? *Hint*: Use the minimality of F.

Now, if the variable x_i is assigned 0, then the subformula G will disappear from the formula F, thereby erasing at least one more input gate. Since x_i is assigned 0 or 1 randomly (with probability 1/2), we expect at least $\frac{1}{2} \cdot \frac{s}{n}$ input gates to disappear because of these secondary effects. In total, we thus expect at least

$$\frac{s}{n} + \frac{s}{2n} = \frac{3s}{2n}$$

input gates $e \in F$ to disappear in the first stage, yielding a new formula with expected size at most [3]

$$s - \frac{3s}{2n} = s \cdot \left(1 - \frac{3}{2n}\right) \leq s \cdot \left(1 - \frac{1}{n}\right)^{3/2}.$$

The succeeding stages of the restriction can be analysed in the same way. After each stage the number of variables decrements by one. Hence, the expected size $E[L(f|_\rho)]$ of the final formula is is at most

$$s \cdot \left(1 - \frac{1}{n}\right)^{3/2} \cdot \left(1 - \frac{1}{n-1}\right)^{3/2} \cdot \ldots \cdot \left(1 - \frac{1}{k+1}\right)^{3/2} = s \cdot \left(\frac{k}{n}\right)^{3/2}.$$

By Markov's inequality, the probability that $L(f|_\rho)$ is more than 4 times its expected value is less than $\frac{1}{4}$, which completes the proof. □

Corollary 1.4. *Let f be a boolean function of n variables, and $1 \leq k \leq n$ be an integer. Then there exists a restriction $\rho \in \mathcal{R}_k$ such that*

$$L(f|_\rho) \leq 4 \cdot \left(\frac{k}{n}\right)^{3/2} \cdot L(f).$$

Example 1.5. Let $f = x_1 \oplus x_2 \oplus \cdots \oplus x_n$. Applying Corollary 1.4 with $k = 1$ we get that

$$1 \leq L(f|_\rho) \leq 4 \cdot \left(\frac{1}{n}\right)^{3/2} \cdot L(f),$$

which gives the lower bound $L(f) = \Omega(n^{3/2})$.

[3] Because $\left(1 - \frac{3}{2n}\right)^2 = 1 - \frac{3}{n} + \frac{9}{4n^2}$ whereas $\left(1 - \frac{1}{n}\right)^3 = 1 - \frac{3}{n} + \frac{3}{n^2} - \frac{1}{n^3}$.

1.3. An $\Omega\left(n^{5/2}\right)$ lower bound

Andreev (1987) used Subbotovskaja's argument to prove the first super-quadratic lower bound for formula size.

Arrange the variables $x_1, \ldots, x_n$ into a $b \times m$ matrix

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ & & \cdots & \\ x_{b1} & x_{b2} & \cdots & x_{bm} \end{pmatrix}$$

where $b = \log n$ and $m = n/b$. Given a boolean function $\varphi : \{0,1\}^b \to \{0,1\}$ on b variables, let $f_\varphi(X)$ denote the following boolean function on bm variables X

$$f_\varphi(X) \rightleftharpoons \varphi\left(\bigoplus_{j=1}^{m} x_{1j}, \bigoplus_{j=1}^{m} x_{2j}, \ldots, \bigoplus_{i=j}^{m} x_{bj}\right).$$

Lemma 1.6. *Let ρ be a random restriction from $\mathcal{R}_k$ where $k = \lceil b \ln(4b) \rceil$. Then with probability at least $3/4$, the restriction ρ assigns at least one $*$ to each of the b rows of X.*

Proof. Observe that for every fixed i, $\Pr[\rho(i) = *] = \binom{n-1}{k-1} / \binom{n}{k} = \frac{k}{n}$. Hence,

$$\begin{aligned} & \Pr[\rho \text{ assigns } * \text{ to every row of } X] \\ \geq\ & 1 - \sum_{i=1}^{b} \Pr[\rho(x_{ij}) \neq *, \forall j = 1, \ldots, m] \\ \geq\ & 1 - b \cdot \left(1 - \frac{k}{n}\right)^m \geq 1 - b \cdot e^{-\frac{km}{n}} \\ =\ & 1 - b \cdot e^{-\ln(4b)} = 1 - 1/4 = 3/4. \end{aligned}$$

□

By this lemma, we have that with probability at least $3/4$, φ is a subfunction of $f_\varphi|_\rho$, and hence,

$$\Pr[L(f_\varphi|_\rho) \geq L(\varphi)] \geq \frac{3}{4}.$$

On the other hand, by Lemma 1.4,

$$\Pr\left[L(f_\varphi|_\rho) \leq 4 \cdot \left(\frac{k}{bm}\right)^{3/2} \cdot L(f_\varphi)\right] \geq \frac{3}{4}.$$

Thus, there must be a restriction $\rho \in \mathcal{R}_k$ for which both these events happen, implying that

$$L(f_\varphi) \geq \frac{1}{4} \cdot \left(\frac{bm}{k}\right)^{3/2} \cdot L(f_\varphi|_\rho) \geq \frac{1}{4} \cdot \left(\frac{bm}{k}\right)^{3/2} \cdot L(\varphi). \tag{1.7}$$

Using an easy counting, Shannon has proved that, for almost all boolean functions φ in b variables,

$$L(\varphi) = \Omega\left(\frac{2^b}{\log b}\right).$$

Taking any of these "most complicated" functions φ we get from (1.7) that

$$L(f_\varphi) = \Omega\left(\frac{bm}{k}\right)^{3/2} \cdot \left(\frac{2^b}{\log b}\right) = \Omega\left(\frac{n^{5/2}}{\log n \cdot \log\log n}\right). \tag{1.8}$$

Strictly speaking, this lower bound is not "explicit" since the function φ is not specified. But the construction can be easily adopted to get the same lower bound for the following *explicit* boolean function $A_{b,m}$. This function has $n = 2^b + bm$ variables; the first 2^b variables of $A_{b,m}$ will specify a boolean function φ of b variables by listing its truth table. Then the value of $A_{b,m}$ is defined to be the value of $f_\varphi(X)$ where X is the set of remaining bm variables. From (1.8) we immediately get the following lower bound for $A_{b,m}$.

Theorem 1.9 (Andreev 1987). *Let $b = \lceil \log_2 n - 1 \rceil$ and $m = \lceil n/(2b) \rceil$. Then the function $A_{b,m}$ (of n variables) requires DeMorgan formulas of size $\Omega\left(n^{5/2-\varepsilon}\right)$ for every fixed constant $\varepsilon > 0$.*

1.4. Nechiporuk's theorem

We have seen that Subbotovskaja's method of random restrictions can help us to show that some functions require large formulas. In particular - that some special functions (like the Andreev's function above) require size $\Omega\left(n^{5/2-\varepsilon}\right)$. However, for "natural" functions (like the parity function $x_1 \oplus x_2 \oplus \cdots \oplus x_n$) this method yields only $\Omega\left(n^{3/2}\right)$ lower bounds, which are far from the upper bound $O(n^2)$. Fortunately, we have other methods (due to Nechiporuk and Khrapchenko) which can be applied to many explicit functions, and give almost optimal bounds.

Khrapchenko's method is described in Sect. 15.2.2 of the book (Theorem 15.13). In this section we will describe the Nechiporuk's argument and show how it works for explicit functions. Nechiporuk's method gives slightly worse bounds but works also in the case when we allow *all* binary boolean operations (including $\wedge, \vee, \neg, \oplus$, etc.) as gates.

Let $L(f)$ denote the minimum size of a boolean formula (over the basis of all binary boolean operations) computing f.

A *subfunction* of f on $Y \subseteq X$ is a function obtained from f by setting all the variables of $X \setminus Y$ to constants. Let $N_Y(f)$ be the number of different non-constant subfunctions of f on Y. Intuitively, if f has many subfunctions, then it is complicated and hence should require large formulas. This intuition was made precise by Nechiporuk (1966) who proved the following theorem with a weaker constant factor; the version bellow is due to Paterson (unpublished) and Zwick (1991).

Theorem 1.10 (Nechiporuk 1966). *Let f be a boolean function on a variable set X, and let $Y_1, Y_2, \ldots, Y_k$ be disjoint subsets of X. Then*

$$L(f) \geq \sum_{i=1}^{k} \log_5\left(2N_{Y_i}(f) + 1\right).$$

Proof. Let $S_Y(f)$ be the collection of functions g on a variable set Y for which either g or $\neg g$ is a subfunction of f. Clearly, $N_Y(f) \leq |S_Y(f)|$. Let $size_Y(F)$ be the number of occurrences of variables of Y in a boolean formula F. The theorem follows directly from the following claim.

Claim 1.11. *For every binary formula F and every variable set Y, we have*

$$2|S_Y(F)| + 1 \leq 5^{size_Y(F)}.$$

The claim is proved by the induction on the size of F.

The base case $F = x_i$ divides into two subcases $i \in Y$ or $i \notin Y$. If $i \in Y$ then $S_Y(x_i) = \{x_i\}$; if $i \notin Y$ then $S_Y(x_i) = \{0, 1\}$; thus, both subcases satisfy the claim.

Assume now by induction that $F = F_1 * F_2$, where F_1 and F_2 satisfy the claim, and $*$ is a binary operation. For brevity, let $S_1 = S_Y(F_1)$ and $S_2 = S_Y(F_2)$. Consider the following two collections of boolean functions on Y:

$$\begin{aligned} T &\rightleftharpoons \{g_1 * g_2 : g_1 \in S_1 \text{ and } g_2 \in S_2\}, \\ \widetilde{T} &\rightleftharpoons \{\neg g : g \in T\}. \end{aligned}$$

It is straightforward to check that

$$S_Y(F) \subseteq T \cup \widetilde{T} \cup S_1 \cup S_2,$$

and therefore,[4]

$$\begin{aligned} 2|S_Y(f)| + 1 &\leq 2 \cdot (|T| + |\widetilde{T}| + |S_1| + |S_2|) + 1 \\ &\leq 4|S_1||S_2| + 2|S_1| + 2|S_2| + 1 \\ &= (2|S_1| + 1) \cdot (2|S_2| + 1) \\ &\leq 5^{size_Y(F_1)} \cdot 5^{size_Y(F_2)} = 5^{size_Y(F)}. \end{aligned}$$

This completes the proof of the claim and hence the proof of the theorem. □

Ex. 1.12. Show that Nechiporuk's theorem cannot yield lower bounds larger than $\Omega(\frac{n^2}{\log n})$.

1.4.1. Applications

To apply Nechiporuk's theorem, the function must have many subfunctions. Below we describe two such functions.

The block equality function. Let $n = (k+1) \cdot b$ where $b = \lceil \log k \rceil + 1$ and $k \geq 3$. Break the variables $x_1, \ldots, x_n$ into $(k+1)$ blocks $Y_0, \ldots, Y_k$ of size b, where

$$Y_i = \{x_{i \cdot b+1}, \ldots, x_{(i+1) \cdot b}\}$$

is the i-th block, $i = 0, 1, \ldots, k$. Define [5]

$$f(x_1, \ldots, x_n) := \begin{cases} 1 & \text{if } \exists i \neq j \text{: } Y_i = Y_j \\ 0 & \text{otherwise} \end{cases}$$

Claim 1.13. *For every $i = 0, 1, \ldots, k$,*

$$N_f(Y_i) \geq \frac{2^{2k}}{k}.$$

To show this, assume for simplicity that $i = 0$, i.e. let us consider the first block Y_0. Take arbitrary assignments $\alpha_1, \ldots, \alpha_k \in \{0, 1\}^b$ of constants to the remaining blocks $Y_1, \ldots, Y_k$. If some two of these assignments α_i and α_j coincide, then the resulting subfunction $f(Y_0, \alpha_1, \ldots, \alpha_k)$ is constant 1. So, to get non-constant subfunctions we have to restrict ourselves to assignments $\alpha = (\alpha_1, \ldots, \alpha_k)$

[4] Because $size_Y(F_1) + size_Y(F_2) = size_Y(F)$.

[5] Here $Y_i = Y_j$ means that these blocks coincide as strings.

where all α_i's are different. There are at least $\binom{2^b}{k}$ such assignments. Moreover, all of them induce different subfunctions.

Ex. 1.14. Verify this last claim.

Thus,

$$N_f(Y_i) \quad \geq \quad \binom{2^b}{k} \quad \geq \quad \binom{2k}{k} = \frac{(2k)!}{(k!)^2} \approx \frac{2^{2k}}{\sqrt{\pi k}} > \frac{2^{2k}}{k}.$$

By Nechiporuk,

$$L(f) \quad \geq \quad \sum_{i=0}^{m} \log_5(N_f(Y_i)) \geq (k+1) \cdot \log_5 \frac{2^{2k}}{k} = \Omega(k^2) = \Omega\left(\frac{n^2}{\log^2 n}\right),$$

because $k = \Omega(n/\log n)$.

Indirect Addressing Function The "indirect addressing function" $f_n(z, Y, x)$ is defined as follows. It has n variables arranged into three groups. Let $n = k + 2m$ where $k = \log b$ and $b = \log m$. The first group z consists of k variables, the second group Y consists of m variables arranged into an $(m/b) \times b$ matrix, and the third group x consists of the remaining m variables.

The value of this function is computed as follows. First we compute the number $i = int(z)$ whose binary code is the input z. Then we take the ith row Y_i of Y, compute the number $j = int(Y_i)$ and let

$$f_n(z, Y, x) = x_j.$$

Ex. 1.15. Prove that $L(f_n) = \Omega(\frac{n^2}{\log n})$. *Hint*: Apply Nechiporuk's theorem with $Y_1, \ldots, Y_{m/b}$ being the rows of Y. For each row Y_i fix the input z so that $int(z) = i$, and observe that after that each assignment of constants to x leads to a different subfunction of f_n on Y_i.

Ex. 1.16. Show that f_n can be computed by a DeMorgan *circuit* of size $O(n)$.

TOPIC 2
Decision Trees: Their Size

(Related sections: 10.4, 14.4)

Summary: If a boolean function f *and* its negation $\neg f$ have a DNF of size N, then f can be computed a decision tree of size $n^{O(\log^2 N)}$. This is almost optimal: there are *explicit* boolean functions requiring decision trees of size $2^{O(\log^2 N)}$. The proof employs spectral arguments.

The *size* of a decision tree[1] is the number of all its leaves. Let $\mathrm{dt}(f)$ denote the minimum size of a decision tree computing f.

We already know (see Theorem 10.22 in the book) that that P=NP∩co-NP for decision trees if we consider their *depth*. In this lecture we will show that the situation changes drastically if we consider the *size* instead of the depth. It turns out that then there are explicit boolean functions f such that *both* f and its negation $\overline{f}$ have *nondeterministic* decision trees of small size, whereas the size of any *deterministic* decision tree for f is super-polynomial.[2]

Let $D(f)$ denote the minimal number of monomials in a DNF of f. That is, $D(f)$ is the minimal number t such that f can be written as an Or of t monomials.

Ex. 2.1. Let D_1 be a DNF of a boolean function f, and D_2 be a DNF of its negation $\overline{f}$. Show the following *cross-intersection property*: if K is a monomial in D_1 then every monomial in D_2 contains at least one literal which is contradictory to at least one literal in K.

Let f be a Boolean function, and suppose we know that $D(f)$ is small. Is then the decision tree also small? The following examples show that it may be *not* the case.

Ex. 2.2. Consider the following function $f(X)$ on $n = m^2$ boolean variables $X = \{x_{ij} : 1 \le i, j \le m\}$:

$$f = \bigvee_{i=1}^{m} \bigwedge_{j=1}^{m} x_{ij}.$$

Show that $\mathrm{dt}(f) \ge 2^{\sqrt{D(f)}}$. *Hint*: Observe that all the minterms and maxterms of f have length m. Show that every such function requires a decision tree of size at least 2^m.

---

[1] See Sect. 10.4 of the book for the definition.

[2] We have not defined what a "nondeterministic decision tree" actually is, because we do not what to introduce a formal notion of "nondeterministic computation" (we don't need it later). Informally, a tree is nondeterministic if it has additional "guessing" nodes from which the computation can proceed in both directions. Thus, on one input we may have many computation paths from the root to a leaf. A tree accepts the inputs if at least one of these paths reach a leaf labeled by 1.

The function from this example has very small DNF (of size m) but the DNF of its negation

$$\overline{f} = \bigwedge_{i=1}^{m} \bigvee_{j=1}^{m} \overline{x}_{ij}$$

is huge—it has 2^m monomials. It is therefore natural to ask what happens if *both* the function f *and* its negation $\overline{f}$ have small DNFs? Put otherwise, does P=NP∩co-NP for decision trees if we consider the *size* as their complexity measure? This question was open for a long time, and was recently answered *negatively* in [**18**]. Below we describe this solution.

2.1. The upper bound

The sum $N(f) := D(f) + D(\overline{f})$ will be called the *weight* of f. It is clear that $N(f) \leq \mathrm{dt}(f)$ (just because every decision tree represents *both* the function and its negation). It was long unknown if $\mathrm{dt}(f)$ is polynomial in $N(f)$, i.e. if $\mathrm{dt}(f) \leq N(f)^c$ for some absolute constant c.

The following result is due to Ehrenfeucht and Haussler (1989) and says that the decision tree size of any Boolean function is quasi-polynomial in its weight. In the next section we will show that this upper bound is almost optimal: there are explicit functions f for which $\mathrm{dt}(f) = 2^{\Omega\left(\log^2 N\right)}$. Here and throughout, $\log x$ stands for $\log_2 x$.

Theorem 2.3 (Ehrenfeucht–Haussler 1989). *Let f be a Boolean function in n variables and $N = D(f) + D(\overline{f})$ be its weight. Then*

$$\mathrm{dt}(f) \leq \left(\frac{n}{\log^2 N}\right)^{O(\log^2 N)} \leq n^{O\left(\log^2 N\right)}.$$

We give a simpler and more intuitive proof of this result.

Proof. The idea is to employ the following simple "greedy" strategy: given DNFs for f and $\overline{f}$, let the decision tree *always* test the "most popular" literal first.

Assume, we have DNFs for both f and $\overline{f}$, and let N be the total number of monomials in these two DNFs. Since the disjunction of these two DNFs is a tautology (i.e., outputs 1 on all inputs), there must a monomial of length at most $\log N$.

> **Ex. 2.4.** Show this. *Hint*: Every monomial of length k accepts only 2^{n-k} of the inputs.

Select one of such monomials and denote its length by k. The selected monomial belongs to one of the two DNFs. By the cross-intersection property of monomials (see Exercise 2.1), every monomial in the other DNF contains at least one literal which is contradictory to at least one literal in the selected monomial. Hence, there is a literal in the selected monomial, which is contradictory to at least a $1/k$-portion of the monomials in the other DNF. Thus, if we evaluate this literal to 1, then all these monomials will get the value 0 and so will disappear from the DNF.

Test this variable first and apply this strategy recursively to both restrictions which arise. By the observation we just made, for each node v, at least one of its two successors is such that at least one of the two DNFs in it decreases by a factor of $(1 - 1/k)$. Let us call the corresponding outgoing edge(s) *decreasing*. Now, if v is a node (not a leaf) such that the path from the source to v contains s decreasing

edges, at least one of the two initial DNFs was decreased at least $s/2$ times, and each time it was decreased by a factor of $1-1/k \geq 1-1/\log N$. If s would be at least $2\log^2 N$ then at least one of the DNFs at v would have only

$$\begin{aligned} N\left(1-\frac{1}{\log N}\right)^{s/2} &< N\cdot e^{-s/(2\log N)} \\ &\leq N\cdot e^{-\log N} \\ &= N^{1-\log e} < 1 \end{aligned}$$

monomials, which is impossible (because v is not a leaf). Thus, *every* path to a leaf has at most n edges, and among them at most $s := 2\log^2 N$ can be decreasing. Recall that for *every* node at least one of the out-going edges was decreasing. Assume w.l.o.g. that every node has *exactly* one decreasing edge (if there were two, we simply ignore one of them). Mark decreasing edges by 1 and the remaining edges by 0. Then every leaf corresponds to a 0-1 string of length at most n with at most s ones. The number of such strings (and hence, the total number of leaves) does not exceed $L(n,s/2)$, where $L(n,t)$ denotes the maximal possible number of leaves in a decision tree of depth n such that *every* path from the root to a leaf has at most t 1-edges.

It remains to estimate $L(n,t)$ for $t = s/2$. Clearly, we have the following recurrence:

$$L(n,t) \leq L(n-1,t) + L(n-1,t-1),$$

with $L(0,t) = L(n,0) = 1$. Using the identity $\binom{n-1}{k-1} = \binom{n}{k} - \binom{n-1}{k}$, this recurrence implies

$$L(n,t) \leq \sum_{i=0}^{t} \binom{n}{i} \leq \left(\tfrac{ne}{t}\right)^t.$$

Thus,

$$\mathrm{dt}(f) \leq L(n,s/2) = L(n,\log^2 N) \leq \left(\tfrac{n}{\log^2 N}\right)^{O(\log^2 N)},$$

as desired. □

Ex. 2.5. (Research project). Is it possible to improve the upper bound $\mathrm{dt}(f) \leq n^{O(\log^2 N)}$ in Ehrenfeucht-Haussler theorem to $\mathrm{dt}(f) \leq 2^{O(\log^2 N)}$?

We have proved that every function has a decision tree of size quasi-polynomial in its weight. Can this upper bound be improved? The negative answer was recently given in [**18**]: there are explicit Boolean functions f for which

$$\mathrm{dt}(f) \geq 2^{\Omega(\log^2 N)} = N^{\Omega(\log N)}.$$

That is, there exist boolean functions f whose DNF and CNF *both* have size at most N, but any decision tree computing f requires size super-polynomial in N. This implies that, unlike for the depth of decision trees, for their size we have an inequality P≠NP∩co-NP.

2.2. Lower bounds

We already know how to prove that a Boolean function cannot be computed by a decision tree of small size: it is enough that its minterms and maxterms are long enough (see Exercise 2.2). However, this trivial argument cannot help us to establish trade-off between the weight of a Boolean function and its decision tree

size: we want to show that the decision tree size is super-polynomial in the total number of minterms *and* maxterms, and this number is exponentially large in the length of minterms/maxterms.

Example: Let $\mathcal{A} = \{A_0, A_1\}$ be disjoint k-element subsets and let $\mathcal{B}$ be the family of all k-element subsets B of $\{1, \ldots, n\}$ such that $B \cap A_0 \neq \emptyset$ and $B \cap A_1 \neq \emptyset$. Let f be a (monotone) Boolean function whose minterms are all the monomials $\bigwedge_{i \in B} x_i$ over all $B \in \mathcal{B}$. This function has only two maxterms $\bigvee_{i \in A_0} x_i$ and $\bigvee_{i \in A_1} x_i$. Thus, the weight $N(f)$ of f is $|\mathcal{B}| + 2 \geq \binom{n-2}{k-2} \geq n^{\Omega(k)}$. Since minterms and maxterms have length k, the (trivial) argument, mentioned above, yields a lower bound 2^k on the size of any decision tree computing f. But this bound is much smaller than $N(f)$.

So, to disprove the P=NP∩co-NP conjecture for the size of decision trees we have to argue in some more subtle way. For this purpose we will use an argument which has many applications in engineering. The argument is based on harmonic analysis of Boolean functions, and is known as the *spectral argument.*'

2.2.1. Fourier transforms for boolean functions

Roughly speaking, the main idea of what is known as "spectral argument" is to estimate how far is a given Boolean function apart from the parity function. For this it will be convenient to switch to $(-1,+1)$-notation, i.e., to consider Boolean functions as mappings from $\{-1,+1\}^n$ to $\{-1,+1\}$, where the correspondence $1 \to -1$ and $0 \to +1$ is assumed. Explicitely, this correspondence is given by the transformation $x' = 1 - 2x$ which transforms the value $x \in \{0,1\}$ into the value $x' \in \{+1,-1\}$.

Ex. 2.6. Show that in $(-1,+1)$ notation the $\mathrm{OR}(x,y)$ turns to the function $\mathrm{OR}(x,y) = (x + y - x \cdot y + 1)/2$. What about the $\mathrm{AND}(x,y)$? What about the parity function $x \oplus y$?

We treat the functions $f : \{-1,+1\}^n \to \{-1,+1\}$ as elements of 2^n-dimensional vector space with an inner product defined by

$$\langle f, g \rangle := 2^{-n} \sum_{x \in \{-1,+1\}^n} f(x)g(x).$$

The set of all monomials

$$X_S := \prod_{i \in S} x_i\,, \qquad S \subseteq [n] := \{1, \ldots, n\}$$

forms an orthonormal basis for this space. That is

- Every function $f : \{-1,+1\}^n \to \{-1,+1\}$ can be uniquely expressed as $\sum_S \hat{f}(S) X_S$, where $\hat{f}(S)$ are real constants;
- If $S \neq T$ then $\langle X_S, X_T \rangle = 0$, and for every S, $\langle X_S, X_S \rangle = 1$.

The *Fourier transform* of f is just the expansion of f as a linear combination of the X_S's. Since the X_S's form an orthonormal basis, the Sth Fourier coefficient $\hat{f}(S)$ of f is found via:

$$\hat{f}(S) := \langle f, X_S \rangle = 2^{-n} \sum_x f(x) X_S(x)$$

Ex. 2.7. Check the last statement. *Hint*: Suppose that $f = \sum_S \hat{f}(S) \cdot X_S$. To find $\hat{f}(T)$ take the scalar product of f with X_T.

Ex. 2.8. Let $f = x_i$ be a single variable. Show that then $\hat{f}(\{i\}) = 1$.

Ex. 2.9. Suppose that the value of f does not depend on the ith variable, i.e. that

$$f(x_1, \dots, x_{i-1}, +1, x_{i+1}, \dots, x_n) = f(x_1, \dots, x_{i-1}, -1, x_{i+1}, \dots, x_n)$$

Show that then $\hat{f}(S) = 0$ for every S with $i \in S$.

If $f(X)$ is a boolean function in n variables $X = \{x_1, \dots, x_n\}$, its *leading Fourier coefficient* (or LFC, for short) is $\hat{f}(S)$ for $S = \{1, \dots, n\}$.

Ex. 2.10. What is the LFC of the parity function $x_1 \oplus x_2 \oplus \cdots \oplus x_n$?

If $f(X)$ is an arithmetic combination of some functions with disjoint sets of variables, then the LFC of f can be easily commuted form the LFCs of these functions.

Ex. 2.11. Let $X = X_1 \cup X_2$ be a partition of X into two nonempty blocks, $X_1 \cap X_2 = \emptyset$. Let $f(x) = f_1(X_1) + f_2(X_2) + f_1(X_1) \cdot f_2(X_2)$ where $f_i(X_i)$ is an arbitrary boolean function depending only on the variables in X_i, $i = 1, 2$. Prove that the LFC of f is just the product of the LFCs of f_1 and f_2. *Hint*: Observe that the summand $f_1(X_1) + f_2(X_2)$ does not contribute to the LFC of f (cf. Exercise 2.9), and use the disjointness of X_1 and X_2 to compute the LCF of $f_1(X_1) \cdot f_2(X_2)$.

2.3. The spectral lower bound

The following lemma estimates the size of a decision tree for f in terms of Fourier coefficients of f.

Lemma 2.12. *For every Boolean function f in n variables and every subset of indices $S \subseteq \{1, \dots, n\}$ we have the bound*

$$\mathrm{dt}(f) \geq 2^{|S|} \cdot \sum_{T \supseteq S} |\hat{f}(T)|. \tag{2.13}$$

Proof. Take a decision tree for f of size $\mathrm{dt}(f)$. For a leaf ℓ, let $\mathrm{val}(\ell) \in \{-1, +1\}$ be its label (recall that we are in $(-1, +1)$-notation), and let I_ℓ be the set of indices of those variables, which are tested on the path to ℓ. Let $B_\ell \subseteq \{-1, +1\}^n$ be the set of all the inputs that reach leaf ℓ; hence, $|B_\ell| = 2^{n-|I_\ell|}$.

Since each input reaches a *unique* leaf, the sets B_ℓ are mutually disjoint. Hence,

$$\hat{f}(T) = 2^{-n} \sum_x f(x) \cdot X_T(x) = 2^{-n} \sum_\ell \sum_{x \in B_\ell} f(x) \cdot X_T(x) = \sum_\ell \mathrm{val}(\ell) \cdot \Delta(T, \ell)$$

for every $T \subseteq [n] := \{1, \dots, n\}$, where

$$\Delta(T, \ell) := 2^{-n} \sum_{x \in B_\ell} X_T(x).$$

Now, if $T \not\subseteq I_\ell$ (i.e., if some variable x_i with $i \in T$ is not tested along the path from the root to the leaf ℓ) then $X_T(x) = +1$ for exactly half of the inputs $x \in B_\ell$, and hence, $\Delta(T, \ell) = 0$. If $T \subseteq I_\ell$ then the value of X_T is fixed on B_ℓ to either $+1$ or -1, and hence, $|\Delta(T, \ell)| = 2^{-n} \cdot |B_\ell| = 2^{-|I_\ell|}$. Thus, in both cases, $|\Delta(T, \ell)| \leq 2^{-|I_\ell|}$. Since for any $S \subseteq [n]$ there are only $2^{|I_\ell| - |S|}$ sets T satisfying $S \subseteq T \subseteq I_\ell$, we

conclude that

$$\begin{aligned}\sum_{T:T\supseteq S}|\hat{f}(T)| &\leq \sum_{T:T\supseteq S}\sum_{\ell}|\Delta(T,\ell)| = \sum_{\ell}\sum_{T:T\supseteq S}|\Delta(T,\ell)|\\ &\leq \sum_{\ell}2^{-|S|} = 2^{-|S|}\cdot \mathrm{dt}(f),\end{aligned}$$

and the desired bound (2.13) follows. □

2.3.1. A super-polynomial trade-off

Recall that our goal is to exhibit a boolean function f which requires decision tree of size super-polynomial in its weight $N = D(f) + D(\overline{f})$. In this section we will prove a super-polynomial lower bound of the form $N^{\Omega(\sqrt{\log N})}$ for the Iterated Majority function. In the next section we will use a bit more subtle argument to prove the lower bound $N^{\Omega(\log N)}$ which (in view of Ehrenfeucht–Haussler's Theorem 2.3) is almost the best possible.

The majority of three Boolean variables is given by

$$MAJ_3(x_1,x_2,x_3) = x_1x_2 \vee x_1x_3 \vee x_2x_3.$$

Note that in the $(-1,+1)$-representation (i.e., when the correspondence $1 \to -1$ and $0 \to +1$ is assumed) we have that

$$MAJ_3(x_1,x_2,x_3) = (x_1+x_2+x_3-x_1x_2x_3)/2.$$

Consider the monotone function F_h in $n = 3^h$ variables which is defined inductively as follows: $F_0 = x$, $F_1 = MAJ_3(x_1,x_2,x_3)$ and for $h \geq 2$

$$F_h = MAJ_3(F_{h-1}^{(1)}, F_{h-1}^{(2)}, F_{h-1}^{(3)})$$

where $F_{h-1}^{(\nu)}$ are three copies of F_{h-1} with disjoint(!) sets of variables.

> **Ex. 2.14.** Show that F_h has $n = 3^h$ variables and its weight is $2\cdot 3^{2^h-1}$.
> *Hint*: Observe that: (1) $D(F_0) = 1$ and $D(F_h) = 3\cdot D(F_{h-1})^2$, and (2) the minimal DNF of the negation $\overline{F}_h$ coincides with the DNF of F_h with all the variables negated.

Theorem 2.15. *Let F_h be the iterated majority function and $N = D(f) + D(\overline{f})$ be its weight. Then*

$$\mathrm{dt}(F_h) \geq N^{\Omega(\sqrt{\log N})}.$$

Proof. By the previous exercise, the function $F_h(x_1,x_2,\ldots,x_n)$ has $n = 3^h = 2^{c\cdot h}$ variables, where $c = \log 3 > 3/2$, and has the weight $N = 2\cdot 3^{2^h-1} = 2^{\Theta(2^h)} = 2^{\Theta(n^{2/3})}$. Since $2^{\Omega(n)} \geq 2^{\Omega(\log^{3/2} N)} = N^{\Omega(\sqrt{\log N})}$, it is enough to prove the lower bound

$$\mathrm{dt}(F_h) \geq 2^{\Omega(n)}.$$

To prove this we will apply Lemma 2.12 with $S = [n] = \{1,\ldots,n\}$. Letting a_h to denote the absolute value $\left|\hat{F}_h([n])\right|$ of the leading Fourier coefficient of F_h, this lemma yields

$$\mathrm{dt}(F_h) \geq a_h\cdot 2^n.$$

It remains therefore to prove an appropriate lower bound on a_h. We proceed by induction on h.

Clearly, $a_0 = 1$, since F_0 is a variable (cf. Exercise 2.9).

For the inductive step recall that in the $(-1,+1)$-representation,

$$MAJ_3(x_1,x_2,x_3) = \frac{1}{2}\left(\sum_{i=1}^{3} x_i - \prod_{i=1}^{3} x_i\right)$$

Thus,

$$F_h = \frac{1}{2}\left(\sum_{\nu=1}^{3} F_{h-1}^{(\nu)} - \prod_{\nu=1}^{3} F_{h-1}^{(\nu)}\right)$$

Using the property of leading Fourier coefficients, established in Exercise 2.11, we obtain that

$$a_h = \tfrac{1}{2} a_{h-1}^3.$$

Together with the condition $a_0 = 1$, this recursion resolves to

$$a_h \quad = \quad 2^{-3^0} \cdot a_{h-1}^3 = 2^{-3^0-3^1} \cdot a_{h-2}^3 = 2^{-3^0-3^1-3^2} \cdot a_{h-3}^3 = \ldots = 2^{-\Delta},$$

where

$$\Delta = 3^0 + 3^1 + 3^2 + \cdots + 3^{h-1} = \frac{3^h - 1}{3-1} = (3^h - 1)/2 = (n-1)/2.$$

Thus

$$\mathrm{dt}(F_h) \geq a_h \cdot 2^n \geq 2^{-(n-1)/2} \cdot 2^n = 2^{\Omega(n)},$$

as desired. □

2.3.2. An almost optimal trade-off (optional section)

In (0,1)-notation,

$$\textsc{nand}(x,y) = \overline{x} \vee \overline{y} = \begin{cases} 0 & \text{if } x = y = 1 \\ 1 & \text{otherwise} \end{cases}$$

In (+1,-1)-notation (via the correspondence $1 \to -1$ and $0 \to +1$),

$$\textsc{nand}(x,y) = \tfrac{1}{2}(xy - x - y - 1) = \begin{cases} +1 & \text{if } x = y = -1 \\ -1 & \text{otherwise} \end{cases}$$

Iterated NAND is a function G_h on $n = 2^h$ variables which is computed by a read-once formula of depth h with NAND-gates. Put otherwise, this function is defined by the following recursion: $G_0 = x$ and

$$G_h = \tfrac{1}{2}\left(G_{h-1}^{(1)} \cdot G_{h-1}^{(2)} - G_{h-1}^{(1)} - G_{h-1}^{(2)} - 1\right), \tag{2.16}$$

where $G_{h-1}^{(1)}$, $G_{h-1}^{(2)}$ are two copies of G_{h-1} with disjoint sets of variables. For this function we have that

$$N_h := N(G_h) \leq 2^{2^{(h/2)+1}} \approx 2^{\sqrt{n}}.$$

By Ehrenfeucht-Haussler theorem,

$$\mathrm{dt}(G_h) \leq \left(\tfrac{ne}{\log^2 N_h}\right)^{\log^2 N_h} \leq N_h^{O(\log N_h)}.$$

The next result shows that this bound is optimal.

Theorem 2.17. $\mathrm{dt}(G_h) \geq N_h^{\Omega(\log N_h)}$.

Proof. It is enough to prove that $\mathrm{dt}(G_h) \geq \exp(\Omega(n))$, because $n \approx (\log N_h)^2$.

Say that a Fourier coefficient $\hat{G}_h(T)$ is *dense* if for every subtree of height 2, T contains the index of at least one of the four variables in that subtree. Let

$$c_h := \left| \sum_{T \text{ dense}} \hat{G}_h(T) \right|.$$

By direct computation, $c_2 = 27/8 = 3.375$. Indeed, in order to compute c_2, we use the following transformation. Let $f_1 = G_1^{(1)} + 1/2$ and $f_2 = G_1^{(2)} + 1/2$. Then it follows from (2.16) that

$$G_2 = \tfrac{1}{2} f_1 f_2 - \tfrac{3}{4} f_1 - \tfrac{3}{4} f_2 + \tfrac{1}{8}.$$

Since each monomial in f_1 and f_2 contains at least one variable and the sets of variables of f_1 and f_2 are disjoint, there are no common monomials in the four terms in the above expression of G_2. Hence, it is easy to calculate the sum of the absolute values of the coefficients in the non-constant monomials, which is $c_2 = 1/2 \cdot r_1 \cdot r_2 + 3/4 \cdot (r_1 + r_2) = 27/8 = 3.375$, where $r_1 = r_2 = 3/2$ is the sum of the absolute values of the coefficients in f_1 and f_2.

For $h > 2$ we use the recursion (2.16) directly. Only the first term $G_{h-1}^{(1)} \cdot G_{h-1}^{(2)}$ can contribute to dense coefficients. Hence,

$$c_h = \tfrac{1}{2} c_{h-1}^2.$$

This resolves to $c_h = 2(c_2/2)^{2^{h-2}}$ which is $\exp(\Omega(n))$ since $c_2 > 2$.

Applying Lemma with $S = \emptyset$, we obtain that

$$\mathrm{dt}(G_h) \geq 2^0 \cdot \sum_T |\hat{f}(T)| \geq c_h = \exp(\Omega(n)),$$

thus completing the proof of the theorem. □

TOPIC 3
A Time-Space Tradeoff

(Related sections: 2)

Summary: Proof that every nondeterministic branching program computing an explicit function $f : D^n \to \{0,1\}$ in linear time must have exponential size.

In this lecture we will consider branching programs computing boolean functions $f : D^n \to \{0,1\}$ with $|D| > 2$. Our goal is to prove that some explicit functions f require branching programs of exponential size if the time is required to be linear in n, i.e. does not exceed kn for some constant k. Since the logarithm of the size of a branching program is of the same oder as the space used by a Turing machine, this gives a non-trivial tradeoff between space and time.

A celebrated recent result of Miklós Ajtai (1999) gives such a lower bound for the boolean case (when $|D| = 2$), but the proof in this case is based on very subtle combinatorial and probabilistic arguments. On the other hand, for larger domains such lower bounds can be obtained using a simple averaging argument, and we present it below.

We stress that, although we allow the domain D be large, the argument we describe below works also in the case of *nondeterministic* branching programs. The argument of Ajtai used for the boolean case $D = \{0,1\}$ essentially employs the fact that the underlying branching program is *deterministic*: in this case input vectors are in one-to-one correspondence with the computations on them. Proving nontrivial lower bounds for nondeterministic time kn branching programs over $D = \{0,1\}$ remains an open problem, even for $k = 1+\varepsilon$ with an arbitrary small constant $\varepsilon > 0$.

The function $f : D^n \to \{0,1\}$, for which we are going to show that any nondeterministic time kn branching program must be large, is easy to describe. The domain D is the Galois field $\mathrm{GF}(q)$ where q is a sufficiently large with respect to k prime power.[1] The function $f(Y,x)$ has $n = m^2 + m$ variables, the first m^2 of which are arranged in an $m \times m$ matrix Y. Then $f(Y,x) = 1$ if and only if $Y \cdot x = 0$, i.e., if x is orthogonal (over $\mathrm{GF}(q)$) to all the rows of Y.

3.1. The model

We consider functions $f : D^n \to \{0,1\}$ where D is a finite domain. A standard model to compute such functions $f(x_1, \ldots, x_n)$ is that of (deterministic) branching programs. Such a program is a directed acyclic graph whose nodes are labeled by

[1] "Sufficiently large" here means that $q \geq k^{\Omega(k)}$. The functions considered in Beame et. al. (1998) require $q \geq 2^{2^{\Omega(k)}}$.

variables x_i and have out-degree d; the outgoing edges are labeled by tests "$x_i = \delta$" for all $\delta \in D$. Sinks (i.e. out-degree 0 nodes) are labeled by constants 0 and 1. Such a program computes f in a natural way. We declare one node as a *source*. Given an input string $a = (a_1, \ldots, a_n) \in D^n$, start the computation at the source; if the computation reaches a node labeled by x_i, then follow the outgoing edge labeled by "$x_i = a_i$". That is, we follow the path, the tests along which are consistent with the actual values of the corresponding coordinates of the input string. Proceeding this way, the computation reaches a sink; its label is the value $f(a)$.

We can introduce *nondeterminism* by allowing so-called guessing nodes. These nodes are unlabeled and have an arbitrary out-degree. If a computation reaches such a node then it can proceed further by following any of the outgoing edges. Such a program accepts an input string if and only if at least one path from the source to a 1-sink is consistent with this input. In this lecture we consider only nondeterministic programs.

The *size* of a program P, size(P), is the number of non-guessing nodes.

We say that a program computes a given function f in *time* T if for every input $a \in f^{-1}(1)$ there is a path from the source to a 1-sink which is consistent with a (i.e. is an accepting path) and along which at most T tests are made. Note that we do not require that other paths accepting this input a, must be short—it is enough that at least one of them is such!

3.2. Main lemma

In this section we prove that if a function f can be computed by a small program, then there is an input $a \in f^{-1}(1)$ and two disjoint subsets Z_1, Z_2 of its bits such that *simultaneously* changing these bits in many ways does not change the value of f.

We will consider branching programs working in time $T \leq kn$. Throughout this section, let

$$K := 2\sum_{j=0}^{k} \binom{3k^2}{k};$$

hence, $K \leq k^{O(k)}$.

Lemma 3.1. *Let $f : D^n \to \{0,1\}$, $F \subseteq f^{-1}(1)$ and $\mu(F) = |F|/|D|^n$ be the density of F. Suppose that f can be computed by a nondeterministic branching program P in time kn. Then for every $s \leq n/(2K)$ there exist two disjoint s-element subsets of variables Z_1, Z_2, an assignment $w : X \setminus (Z_1 \cup Z_2) \to D$, and sets A_j, $j = 1,2$ of assignments $a : Z_i \to D$ such that*

(i) *$f(a_1, w, a_2) = 1$ for all $(a_1, a_2) \in A_1 \times A_2$;*
(ii) *$|A_j| \geq \frac{1}{\Delta} \cdot |D|^s$ for both $j = 1,2$;*[2]
(iii) *$|A_j| \geq \frac{1}{\sqrt{\Delta}} \cdot |D|^s$ for at least one $j = 1,2$,*

where

$$\Delta = \frac{K^2 \binom{n}{s}^2 \cdot \mathrm{size}(P)^{3k^2}}{2\mu(F)}.$$

[2]In this note we will not use this (stronger) property to derive explicit lower bounds. We include it just because the proof of this property for the boolean case ($|D| = 2$) with Δ much closer to *size*(P) is the key of the whole Ajtai's argument. If time allows, we will try to discuss also this proof later.

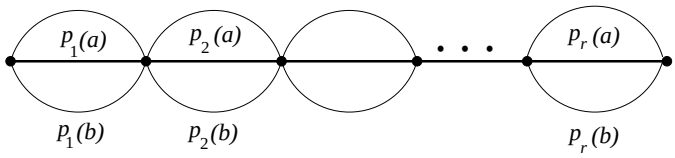

Figure 1. One factor-class of computations

Proof. We first prove the lemma under the assumption that, for every input $a \in F$, there is a computation $comp(a)$ along which no variable is tested more than k times; such programs are known as nondeterministic read-k branching programs (k-n.b.p.). We then show that essentially the same argument works for the programs of length kn.

Let P be a k-n.b.p. computing f. We may assume w.l.o.g. that each variable is tested along $comp(a)$ at least once. This can be achieved by adding a sequence of n "dummy" tests of all variables at the source; the size of a b.p, increases only by an additive factor of n, moreover the resulting b.p. is $(k+1)$-n.b.p.

For each input $a \in F$, fix one accepting read-k computation path $comp(a)$. Take $l := 3k$ and split each such path into $r \leq lk$ sub-paths

$$comp(a) = \big(p_1(a), \ldots, p_r(a)\big)$$

of length $\lceil n/l \rceil$ each (except, perhaps, the last sub-path); the length of a sub-path $p_i(a)$ is the number of different variables tested along it. That is, we have r *time intervals*, and in the i-th of them the computation on a follows the sub-path $p_i(a)$.

Factorize the (selected) computations $comp(a) = \big(p_1(a), \ldots, p_r(a)\big)$, $a \in F$, according to the following equivalence relation: two paths are equivalent if the starting nodes of the corresponding sub-paths $p_1(a), \ldots, p_r(a)$ coincide (see Fig. 1).

Since we have only size(P) nodes and each factor class corresponds to a sequence of at most $r \leq 3k^2$ nodes, the number of factor-classes does not exceed

$$S := \text{size}(P)^{3k^2}.$$

Thus, there is a subset $A \subseteq F$ of inputs such that $|A| \geq |F|/S$ and all the computations $comp(a)$ on inputs $a \in A$ belong to one factor-class. From now on we concentrate only on these computations.

By our assumption, no variable can appear in more than k time intervals. (Recall that we have $r \leq kl$ such intervals, each of length $\lceil n/l \rceil$.) Define the *trace* $T(a,x)$ of a variable x along the computation $comp(a)$ as the set of those (at most k) time intervals along which this variable is tested.

Say that a pair Z_1, Z_2 of disjoint s-element subsets of variables is *good* for an input $a \in A$ if there exit two disjoint traces T_1, T_2 such that for both $j = 1, 2$ all the variables in Z_j have the same trace T_j along the computation $comp(a)$.

Claim 3.2. [3] *Every input $a \in A$ has at least one good pair.*

Proof. Take an input $a \in A$ and partite the set of all variables according to their traces along the computation $comp(a)$. Since we have only $\sum_{j=0}^{k} \binom{r}{j} \leq K/2$ possible traces, the average size of one block in this partition is at least $2n/K$. Hence at least $n/2$ of the variables belong to blocks of size at least $n/K \geq s$.

[3] Cf. Exercise 2.12 in the book.

Let Y be the union of these (large) blocks; hence, $|Y| \geq n/2$. The fact that the variables from different blocks have different traces alone does not yield a good pair – we need these traces be *disjoint*. To obtain such a pair we will use the fact that each sub-path can test at most n/l variables.

Take any of large blocks Z_1. By the definition, $|Z_1| \geq s$ and all the variables in Z_1 have the same trace T_1, say, $T_1 = \{1, \ldots, k\}$. The corresponding to this trace sub-paths $p_1(a), \ldots, p_k(a)$ can test at most $k \cdot \frac{n}{l} < \frac{n}{2} \leq |Y|$ different variables. Hence, there must be a variable $y \in Y$ which is tested in none of these sub-paths. This means that the trace $T(a, y)$ of y is disjoint from the trace T_1 of the variables in Z_1. Let Z_2 be the (unique) large block containing y. Since all the variables in Z_2 have the same trace $T_2 = T(a, y)$, the pair Z_1, Z_2 is good for a, as desired. □

We have shown that every input $a \in A$ has at least one good pair Z_1, Z_2. The number of all possible pairs of disjoint s-element subsets Z_1, Z_2 of variables does not exceed

$$\binom{n}{s}\binom{n-s}{s} \leq \binom{n}{s}^2,$$

and the number of possible pairs T_1, T_2 of (corresponding to these subsets of variables Z_1, Z_2) traces does not exceed K^2. Hence,[4] some pair must be good for many inputs: there exists a pair Z_1, Z_2 of disjoint s-element subsets of variables, a pair T_1, T_2 of disjoint traces, and a subset of inputs $B \subseteq A$ such that

$$|B| \geq |A| \cdot \binom{n}{s}^{-2} \cdot K^{-2} \geq |F| \cdot S^{-1} \cdot \binom{n}{s}^{-2} \cdot K^{-2} = \frac{2}{\Delta} \cdot |D|^n.$$

and the following holds for every computation $comp(a) = (p_1(a), \ldots, p_r(a))$ on an input $a \in B$ and every time interval $i = 1, \ldots, r$:

(*) *If $i \in T_j$ then no variable from Z_{3-j} is tested along $p_i(a)$.*

We will use this property of these inputs to "cut-and-paste" them. Let $b_1, c_1 : Z_1 \to \{0,1\}$ be an arbitrary assignments to the variables in Z_1, $b_2, c_2 : Z_1 \to \{0,1\}$ an arbitrary assignments to the variables in Z_2, and $w : X \setminus (Z_1 \cup Z_2) \to \{0,1\}$ an assignment to the remaining variables (see Fig. 2).

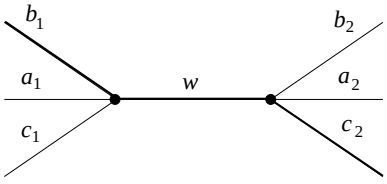

Figure 2

Claim 3.3. *If both inputs $b = (b_1, w, b_2)$ and $c = (c_1, w, c_2)$ belong to B, then also the combined inputs (b_1, w, c_2) and (c_1, w, b_2) belong to B.*

[4]This is the step which makes the Lemma useless in the case of boolean programs. At this place Ajtai argues in a much more subtle way: he picks the time intervals at random and shows that the obtained pairs Z_1, Z_2 for different inputs $a \in A$ are "not very different."

Proof. We will make use of the fact that the computations $comp(a) = (p_1(a), \ldots, p_r(a))$ on all inputs $a \in B$ belong to the same factor-class. That is, the starting nodes of the corresponding sub-paths $p_1(a), \ldots, p_r(a)$ are the same for all inputs $a \in B$.

Take the combined input $e = (b_1, w, c_2)$, and consider the computation

$$comp(e) = (p_1(e), \ldots, p_r(e))$$

on it. In oder to prove that e belongs to B, it is enough to show that during each time interval $i = 1, \ldots, r$, the i-th sub-path $p_i(e)$ coincides with the i-th sub-path of the computation on *at least one* input from B; then the pair Z_1, Z_2 is also good for e, implying that input e must belong to B (recall that B is a *maximal* subset of A for which this pair is good).

If $i \in T_1$ then, by (*), only the variables from $X \setminus Z_2$ can be tested along $p_i(c)$. Since on these variables e coincides with c, we have that in this case $p_i(e) = p_i(b)$.

If $i \notin T_1$ then, again by (*), only the variables from $X \setminus Z_1$ can be tested along $p_i(b)$. Since on these variables e coincides with b, we have that in this case $p_i(e) = p_i(b)$. □

Let $B_j(a)$ denote the set of all inputs $b \in B$ such that b coincides with a on $X \setminus Z_j$. We will use the fact that the set of inputs B is large, $|B| \geq \frac{2}{\Delta} \cdot |D|^n$, to show that for at least one input $a \in B$ *both* sets $B_1(a)$ and $B_2(a)$ must be large.

Claim 3.4. *There exists an $a \in B$ such that $|B_j(a)| \geq \frac{1}{\Delta} \cdot |D|^s$ for both $j = 1, 2$.*

Proof. Given an $j \in \{1, 2\}$, we can partition the set B into at most $|D|^{n-s}$ classes: two inputs are in one class if they coincide outside Z_j. Each class of this partition can have at most $|D|^s$ inputs. Say that a class is *small* if it has $< \frac{1}{\Delta} \cdot |D|^s$ inputs. The average size of a class is at least $N = |B|/|D|^{n-s}$. Hence, for every $0 \leq \gamma \leq 1$, at most $\gamma \cdot |B|$ inputs from B can belong to classes of size at most γN (see Exercise 2.11 in the book). Taking

$$\gamma := |B|^{-1} \cdot \frac{1}{\Delta} \cdot |D|^n$$

we obtain that at most

$$\gamma \cdot |B| = \frac{1}{\Delta} \cdot |D|^n < |B|/2$$

inputs can belong to small classes.[5] Therefore, there must be an input $a \in B$ which belongs to large classes in *both* partitions (w.r.t. Z_1 and Z_2). □

Fix now an input $a \in B$, guaranteed by Claim 3.4, and let A_j $(j = 1, 2)$ be the projection of $B_j(a)$ onto the set Z_j. Since all the inputs from $B_1(a) \cup B_2(a)$ coincide on the set $X \setminus (Z_1 \cup Z_2)$, Claim 3.3 gives us the first condition (i) of Lemma 3.1. The second condition (ii) is also fulfilled by Claim 3.4, because $|A_j| \geq |B_j(a)|$ for both $j = 1, 2$. To ensure the third condition (iii), we can use the following version of Claim 3.4.

Claim 3.5. *There exists an $a \in B$ such that $|B_j(a)| \geq \frac{1}{\sqrt{\Delta}} \cdot |D|^s$ for at least one $j = 1, 2$.*

Proof. Arguing as in the previous claim with $Z = Z_1 \cup Z_2$ instead of Z_i, we obtain a subset $B' \subseteq B$ of $|B'| \geq \frac{1}{\Delta} \cdot |D|^{2s}$ inputs such that all they coincide on $X \setminus Z$. Let A_j be the projection of these inputs onto Z_j. Since, by Claim 3.3, $|B'| = |A_1| \cdot |A_2|$, the claim follows. □

[5] Because $\gamma N = \frac{1}{\Delta} \cdot |D|^s$ and $|B| > \frac{1}{\Delta} |D|^n$.

This completes the proof of Lemma 3.1 in the case of k-n.b.p.

In the (more general) case of branching programs of length at most kn the argument is the same: just observe that if a path has length kn then at least $n/2$ variables must be tested along it at most $2k$ times. Hence, if $comp(a) = (p_1, \ldots, p_r)$ then at least $n/2$ of the variables appear in at most $2k$ of the intervals $p_1, \ldots, p_r$; call these variables *legal* for a, and consider the traces of only these variables. The rest is the same with n replaced by $n/2$ and k replaced by $2k$. □

3.3. Application to linear codes

Let $q \geq 2$ be a prime power and consider the domain $D = GF(q)$. Given an $n \times n$ matrix A over $GF(q)$, let $f_A : D^n \to \{0,1\}$ denote the function such that $f_A(x) = 1$ if and only if $A \cdot x = 0$, i.e. if the vector x is orthogonal (over $GF(q)$) to all the rows of A.

Theorem 3.6. ([17]) *There exists an absolute constant $c > 0$ such that for every positive integer k and for every prime power $q \geq k^{ck}$ there exists an $n \times n$ matrix A over $GF(q)$ such that every nondeterministic branching program computing $f_A(x)$ in time kn has size exponential in $n \cdot k^{-O(k)}$.*

Although the matrix A is not explicitly given, we can take it as an input an obtain an *explicit* function $f'(A, x) = f_A(x)$ in $N = n^2 + n$ variables such that every nondeterministic branching program computing this function in time kn has size at least $2^{\Omega(\sqrt{N}/k^{O(k)})}$. Moreover, in this case the time spent to test the variables in A may be arbitrarily long.

Proof. Take $\beta := 1/(2K)$ and $\varepsilon := \beta + 1/n$. From the Gilbert-Varshamov bound (see any textbook in Coding Theory), we know that for all sufficiently large n there exist linear codes $F \subseteq GF(q)^n$ of minimum relative distance ε and rate[6] $1 - H_q(\varepsilon)$; that is, $\log_q |F| \geq (1 - H_q(\varepsilon))n$ and every two vectors from F differ in at least εn coordinates. Let A be the parity-check matrix of F, that is,

$$F = \{x \in D^n : A \cdot x = 0\}.$$

Consider the function $f(x) := f_A(x)$. This function accepts $|F| \geq 2^{(1-H_q(\varepsilon))n}$ vectors, and the Hamming distance between any two of them is at least εn.

Let P be a nondeterministic branching program computing f in time kn, and set $s = \lceil \beta n \rceil$. By Lemma 3.1, there exist two disjoint s-element subsets of variables Z_1, Z_2; an assignment $w : X \setminus (Z_1 \cup Z_2) \to D$, and sets A_i, $i = 1, 2$ of assignments $a : Z_i \to D$ such that $f(a_1, w, a_2) = 1$ for all $(a_1, a_2) \in A_1 \times A_2$ and $|A_j| > \frac{1}{\sqrt{\Delta}} \cdot q^s$ for at least one $j = 1, 2$, where

$$\Delta = \frac{K^2 \binom{n}{s}^2 \cdot \text{size}(P)^{3k^2}}{2\mu(F)}.$$

But since each two codewords must differ in at least $\varepsilon n > s$ coordinates, none of the sets A_1, A_2 can have more than one assignment. Hence,

$$\Delta > q^{2s}.$$

---

[6] Here $H_q(x)$ is the q-ary entropy function defined by $H_q(0) = 0$ and

$$H_q(x) = x \log_q(q-1) - x \log_q x - (1-x) \log_q(1-x)$$

for $0 < x \leq (q-1)/q$.

Since $\mu(F) \geq q^{-nH_q(\varepsilon)}$, this implies that $\text{size}(P)^{3k^2}$ must be at least

$$\frac{2\mu(F)\cdot\Delta}{K^2\binom{n}{s}^2} \geq \frac{2q^{2s-H_q(\varepsilon)}}{K^2\binom{n}{s}^2} \geq 2^{\alpha n},$$

where (remembering that $s = \lceil \beta n \rceil$ and $\binom{n}{s} \leq (en/s)^s$)

$$\alpha = (2\beta - H_q(\varepsilon))\cdot \log q - 2\beta\log(e/\beta) - (2\log K)/n.$$

Let us estimate $H_q(x)$. Using $\log_q x = (\log_2 x)/(\log_2 q)$ we obtain

$$\begin{aligned} H_q(x) &= x\frac{\log(q-1)}{\log q} - x\frac{\log x}{\log q} - (1-x)\frac{\ln(1-x)}{\ln q} \leq x + x\frac{\log(1/x)}{\log q} + x\frac{1-x}{\ln q} \\ &\leq x\cdot\left(1 + \frac{1+\log(1/x)}{\log q}\right). \end{aligned}$$

In our case $x = \varepsilon = \beta + 1/n \sim \beta = 1/(2K) = 1/k^{\Theta(k)}$. Hence, if $\log q \geq ck\log k$ for a sufficiently large constant c, then $H_q(\varepsilon) \leq (3/2)\beta$. If the constant c is even larger, then

$$\alpha \geq (1/2)\beta\cdot\log q - 2\beta\log(e/\beta) - (2\log K)/n \geq \beta.$$

Thus, if $\log q \geq ck\log k$ and $c > 0$ is a sufficiently large constant, then $\alpha \geq \beta = k^{-\Theta(k)}$ and the desired lower bound $\text{size}(P) \geq 2^{\alpha n/(3k^2)} \geq 2^{\Omega(n/k^{O(k)})}$ follows. □

Ex. 3.7. Let $f : D^n \to \{0,1\}$. A *Boolean version* of a function $f : D^n \to \{0,1\}$ is a Boolean function f' on $N = n\cdot\lceil\log_2|D|\rceil$ variables obtained by encoding the elements of D by strings in $\{0,1\}^m$ with $m = \lceil\log_2|D|\rceil$. Show that every k'-n.b.p. for f' can be transformed to a $|D|$-way k-n.b.p. for f of the same size with $k \leq k'\cdot\lceil\log_2|D|\rceil$. Why the lower bound for the function f_A given by Theorem 3.6 does not yield a non-trivial lower bound on the size of (boolean) k'-n.b.p. computing f'_A, even for $k' = 2$?

TOPIC 4
Monotone Circuits With Real-Valued Gates

(Related sections: 10.6)

Summary: Proof that the lower bounds criterion for monotone *boolean* circuits (Theorem 10.16) holds also for monotone circuits with *real-valued* gates.

In this lecture we consider monotone circuits where, besides boolean AND and OR gates, one may use arbitrary non-decreasing *real-valued* functions $\varphi : \mathbb{R}^2 \to \mathbb{R}$ as gates. Such a function φ is *non-decreasing* if $\varphi(x_1, x_2) \leq \varphi(y_1, y_2)$ whenever $x_1 \leq y_1$ and $x_2 \leq y_2$. The corresponding circuits are called *monotone real circuit.*

As in boolean circuits, inputs for such circuits also are binary strings $x \in \{0,1\}^n$; the output is also a binary bit 0 and 1. But each intermediate gate may compute any monotone function $f : \{0,1\}^n \to \mathbb{R}$. Hence, unlike in boolean case, here we have uncountable number possible gates $\varphi : \mathbb{R}^2 \to \mathbb{R}$, and one may expect that at least some monotone boolean functions can be computed much more efficiently by such circuits. Exercise 10.15 of the book shows that this intuition is correct: every slice function can be computed by a very small monotone circuit with real-valued gates, but easy counting shows that most of slice functions cannot be computed by boolean circuits of polynomial size (even if NOT gates are allowed!).

It is therefore somewhat surprising that the criterion, stated (in the book) for boolean circuits remains true also for circuits with real-valued gates. The only difference from the boolean case is that now in the definition of t-simplicity we allow larger CNFs and DNFs.

Definition A monotone boolean function f is *weakly t-simple* if for every pair of integers $2 \leq s, r \leq n/2$ there exist a $(c+1)$-CNF C consisting of at most $t \cdot (2d)^{c+1}$ clauses, a $(d+1)$-DNF D consisting of at most $t \cdot (2c)^{d+1}$ monomials, and a subset $I \subseteq \{1, \ldots, n\}$ of $|I| \leq c$ indices such that either $C \leq f$ or $f \leq D \vee \left(\bigvee_{i \in I} x_i\right)$ (or both) hold.

Theorem 4.1 (The Criterion for Real Circuits). *Let f be a monotone boolean function. If f can be computed by a monotone real circuit of size t then f is weakly t-simple.*

The criterion was proved in [**15**]. Here we will give another, more direct proof.

Proof. The idea (due to Avi Wigderson) is to consider thresholds of real gates and approximate the thresholded values. For a real-valued function $f : \{0,1\}^n \to \mathbb{R}$ and a real number t, let $f^{(t)}$ denote the boolean function that outputs 1 if $f(x) \geq t$, and outputs 0 otherwise.

For a gate $f = \varphi(g,h)$ with two input gates g and h the threshold function $f^{(t)}$ can be determined from the thresholds of its input gates as

$$f^{(t)} = \bigvee_{\varphi(s,r)\geq t} g^{(s)} \wedge h^{(r)} \tag{4.2}$$

or

$$f^{(t)} = \bigwedge_{\varphi(s,r)<t} g^{(s)} \vee h^{(r)}. \tag{4.3}$$

Proof of (4.2). We have to show that, for every $x \in \{0,1\}^n$,

$$\varphi(g(x),h(x)) \geq t \iff \exists s,r:\ g(s) \geq s,\ h(x) \geq r \ \text{ and } \ \varphi(s,r) \geq t.$$

The $\Rightarrow$ direction is trivial: just take $s = g(x)$ and $r = h(x)$. For the other ($\Leftarrow$) direction we use the monotonicity of φ: $\varphi(g(x),h(x)) \geq \varphi(s,r) \geq t$. □

Ex. 4.4. Prove (4.3).

It is convenient to think these expressions as an infinite AND and an infinite OR, respectively. However, since the number of settings $x \in \{0,1\}^n$ for input variables is finite, the real gates take only finite number of possible values, and therefore, we really need finite expressions.

As before, every threshold $f^{(t)}$ is approximated by two functions: a c-And-Or $f_0^{(t)}$ and a d-Or-And $f_1^{(t)}$. The approximators for the thresholds of the input variables are 0, 1, or the variable itself, depending on the value of the threshold; they can always represented by at most one literal and thus newer fail.

Let now $f = \varphi(g,h)$ be an intermediate gate with two input gates g and h, and suppose that, for all (finitely many!) reals s,r, the approximators for threshold functions $g^{(s)}$ and $h^{(r)}$ of its input gates are already constructed.

To construct the approximator $f_0^{(t)}$ from the approximators $g_0^{(s)}$ and $h_0^{(r)}$ of its two input gates g and h, we first consider the representation

$$f^{(t)} = \bigvee_{\varphi(s,r)\geq t} g_1^{(s)} \wedge h_1^{(r)}.$$

Since the monomials in the DNFs $g_1^{(s)}$ and $h_1^{(r)}$ have length at most d, all the subexpressions $g_1^{(s)} \wedge h_1^{(r)}$ can be turned into a single $2d$-Or-And D_t. After that we use the same procedure as before to convert this Or-And into a c-And-Or $f_0^{(t)}$ (Claim 10.17 from the book).

When forming $f_0^{(t)}$, we introduce errors as we throw away clauses that become longer than c. We want to count the number of inputs $x \in \{0,1\}^n$ such that $f^{(t)}(x) = 0$ while $f_0^{(t)}(x) = 1$ for some t, i.e., the union over t of the errors introduced in a gate by $f_0^{(t)}$. To do this, let us list in the increasing order $t_1 \leq t_2 \leq \ldots \leq t_N$ all the $N \leq 2^n$ possible values $f(x)$ the gate f can output when the input vector x ranges over $\{0,1\}^n$. Hence,

$$D := D_{t_1} \vee D_{t_2} \vee \cdots \vee D_{t_N}$$

is a $2d$-Or-And, and this Or-And makes no error on x, i.e., $D(x) = f(x)$.

By the definition,

$$D_{t_1} \geq D_{t_2} \geq \cdots \geq D_{t_N}.$$

Hence, if $\mathrm{tr}(D)$ denotes the family of all transversals of D (i.e., the family of all subsets of variables, each of which intersects all the monomials of D) then

$$\mathrm{tr}(D_{t_1}) \subseteq \mathrm{tr}(D_{t_2}) \subseteq \cdots \subseteq \mathrm{tr}(D_{t_N}),$$

implying that

$$\mathrm{tr}(D) = \mathrm{tr}(D_{t_N}).$$

This means that all the clauses (=transversals), which we throw away (because they are longer than c) when forming a c-And-Or f_0 from the Or-And D, are precisely those clauses, which we would throw away when converting the $2d$-Or-And D_{t_N} into a c-And-Or. Thus, by Claim 10.17 from the book, all the errors that may appear during the construction of the left approximator f_0, can be corrected by a $(c+1)$-CNF C of size $|C| \leq (2d)^{c+1}$. That is, for every input x such that $f(x) = 0$ but $f_0(x) = 1$, we have that $C(x) = 0$.

A dual argument can be used to bound the number of errors introduced when constructing the right approximator f_1. In this case we use the expression (4.3) instead of (4.2), and all the introduced errors can be corrected by a $(d+1)$-DNF D of size $|D| \leq (2c)^{d+1}$.

The rest of the proof is the same as that of Theorem 10.16 from the book. □

Using Theorem 4.1 we can, for example, immediately extend the exponential lower bound of Alon and Boppana (see Theorem 10.20 in the book) to the case of monotone real circuits.

Theorem 4.5. *If $s \leq \sqrt{q/2 \ln q}$ then any monotone circuit computing* $\mathrm{POLY}(q, s)$ *has size at least $q^{\Omega(s)}$.*

Remark. As we already mentioned at the beginning, there *exists* (not explicitly given) monotone boolean functions which can be computed by a small monotone circuits with real gates, but requires boolean (even non-monotone!) circuits of exponential size. On the other hand, in [**15**] an explicit monotone boolean function is exhibited which has a small non-monotone boolean circuit, but requires monotone real circuits of exponential size. The overall conclusion is that non-monotone boolean circuits and monotone real circuits are incomparable in their power.

TOPIC 5
Resolution Lower Bounds for the Weak Pigeonhole Principle

(Related sections: 4.8, 18, 19)

Summary: Proof that even the weak version of the pigeonhole principle PHP_n^m, where the number $m \geq n+1$ of pigeons may be *arbitrary large*, has no resolution refutation proof shorter than $2^{\Omega(n^{1/4})}$. This resolves an old and most intriguing open problem about the power of resolution.

The pigeonhole principle PHP_n^m asserts that if $m \geq n+1$ then m pigeons cannot sit in n holes so that every pigeon is alone in its hole. In terms of 0-1 matrices, this principle asserts that, if $m \geq n+1$ then every $m \times n$ 0-1 matrix satisfies *precisely one* of the following two conditions:[1]

1. Every row has at least one 1.
2. Every column has at most one 1.

Exponential lower bounds on the length of the resolution resolution refutation proof of $\neg PHP_n^m$ were long known for the case when the number m of pigeons is not too large: $m \leq n^2/\log n$. The larger m is, the more true the principle PHP_n^m is, and it could be that $\neg PHP_n^m$ could be refuted by much shorter resolution refutation proof. And indeed, all attempts to overcome this "n^2 barrier" in last 15 years failed.

The problem was recently solved by Ran Raz.

Theorem 5.1 (Raz 2001). *For every $m \geq n+1$, every resolution refutation proof of $\neg PHP_n^m$ has length at least $2^{\Omega(n^{1/10})}$.*

Shortly after, Razborov (2001) has found a new, considerably simpler argument which even leads to a higher lower bound $2^{\Omega(n^{1/3})}$. This is already quite close to the best known upper bound $2^{\Omega(\sqrt{n \log n})}$ due to Buss and Pitassi (1996).

As an intermediate step, Razborov (2001) gives an even simpler proof of a slightly weaker lower bound $2^{\Omega(n^{1/4})}$, which already illustrated all major ideas, and we present this proof bellow.

Theorem 5.2 (Razborov 2001). *For every $m \geq n+1$, every resolution refutation proof of $\neg PHP_n^m$ has length at least $2^{\Omega(n^{1/4})}$.*

---

[1] Note that Condition 1 allows that a row has more than one 1. In terms of pigeons this means that one pigeon can be splitted between several holes. In a more friendly (with respect to the pigeons) version of PHP_n^m, this condition is replaced by: every row has exactly one 1 (so that each pigeon sits comfortably in some one hole). This is the *functional version* of the PHP_n^m, and exponential lower bounds on the length of the resolution proofs of it were obtained by Razborov, via an appropriate modification of the "filtering" argument described in this lecture. For more information on the proof complexity of different versions of the pigeonhole principle, visit Razborov's home page at http://genesis.mi.ras.ru/~razborov/.

Our first goal is to show that, in the case of the pigeonhole principle, we can restrict our attention to so-called *monotone* resolution refutation proof.

5.1. Monotone refutation proofs

Recall that the negation $\neg PHP_n^m$ of the pigeonhole principle asserts that m pigeons can be arranged into n pigeonholes so that no two pigeons sit in one hole. If we let

$$x_{i,j} = 1 \iff \text{the } i\text{-th pigeon sits in the } j\text{-th hole},$$

then $\neg PHP_n^m$ can be written as an unsatisfiable CNF,[2] which is a conjunction of the following clauses (we call them *axioms*):[3]

- **Pigeon Axioms:** each of the m pigeon sits in at least one of n holes:

$$x_{i,1} \vee x_{i,2} \vee \cdots \vee x_{i,n} \quad \text{for all } i = 1, \ldots, m.$$

- **Hole Axioms:** no two pigeons sit is one hole:

$$\overline{x}_{i_1,j} \vee \overline{x}_{i_2,j} \quad \text{for all } i_1 \neq i_2 \text{ and } j = 1, \ldots, n.$$

Resolution is a proof system which operates with clauses and has one rule of inference:[4]

$$\frac{C_0 \vee x_{i,j} \qquad C_1 \vee \overline{x}_{i,j}}{C} \ (C_0 \vee C_1 \leq C).$$

A *resolution refutation* of a CNF $\mathcal{F}$ is a resolution proof of the empty clause 0 from the clauses appearing in $\mathcal{F}$. The *length* (or *size*) of a resolution refutation proof P is the overall number $|P|$ of clauses in it.

A clause is *positive* if it has no negated variables. Monotone refutation for $\neg PHP_n^m$ is a proof system which operates with positive clauses and has one rule of inference:

$$\frac{C_0 \vee X_{I_0,j} \qquad C_1 \vee X_{I_1,j}}{C} \ (C_0 \vee C_1 \leq C; I_0 \cap I_1 = \emptyset),$$

where $X_{I,j} = \bigvee_{i \in I} x_{i,j}$. Note that, in general, this rule *is not* sound: there are assignments[5] satisfying both assumptions but falsifying the conclusion.[6] Still, the rule *is* sound if we consider only assignments satisfying all hole axioms; we call such assignments "legal." That is, an assignment a is *legal* if it sends no two pigeons to the same hole.

By a *monotone refutation* of $\neg PHP_n^m$ we will mean a monotone refutation of the set of pigeon axioms

$$X_{i,[n]} = x_{i,1} \vee x_{i,2} \vee \cdots x_{i,n} \quad i = 1, \ldots, m.$$

The following theorem reduces the lower bounds problem for the resolution refutation of $\neg PHP_n^m$ to the lower bounds problem of the monotone refutation of $\neg PHP_n^m$.

---

[2]Recall that a CNF is a set of clauses, each clause being an Or of literals (variables or their negations). A CNF is unsatisfiable if none of the assignments of 0-1 values to the variables can satisfy all its clauses.

[3]Note that in the book, for ease of notation, this set of axioms was denoted by PHP_n^m.

[4]Here and throughout, $C \leq C'$ means that every literal appearing in C also appears in C'.

[5]An assignment is a mapping a which gives each variable $x_{i,j}$ a value $a_{i,j} \in \{0,1\}$. Thus, $a_{i,j} = 1$ means that a sends the i-th pigeon to the j-th hole.

[6]Just take $a_{i_0,j} = a_{i_1,j} = 1$ for some $i_0 \in I_0, i_1 \in I_1$, and let $C_0 = C_1 = C = 0$ (the empty clause). Then $X_{I_0,j}(a) = X_{I_1,j}(a) = 1$ but $0(a) = 0$.

Theorem 5.3 (Buss–Pitassi 1996). [7] *If $\neg PHP_n^m$ has a resolution refutation of length L, then $\neg PHP_n^m$ also has a monotone refutation of length at most L.*

Ex. 5.4. Prove this theorem. *Hint*: Replace each $\overline{x}_{i,j}$ by $\bigvee_{l \neq i} x_{l,j}$.

Ex. 5.5. (Buss–Pitassi 1996) A resolution refutation P is *tree-like* if no clause $C \in P$ participates as an assumption in more than one application of the inference rule (i.e., the underlying graph of P is a binary tree). Prove that, for any $m \geq n+1$, any tree-like monotone refutation P of $\neg PHP_n^m$ has length at least 2^n. *Hint*: Induction on n. For the induction step, take the clauses $C_0, C_1 \in P$ from which 0 is derived by the last application of the inference rule; consider the corresponding subproofs P_0, P_1, and apply the induction hypothesis.

Thus, Theorem 5.2 is a direct consequence of the following

Theorem 5.6 (Razborov 2001). *For every $m \geq n+1$, every monotone refutation of $\neg PHP_n^m$ has length at least $2^{\Omega(n^{1/4})}$.*

The rest of this note is devoted to the proof of this theorem, due to Razborov.

5.2. Proof idea: the pseudo-width of refutation proofs

Suppose we have a monotone refutation proof P of the pigeon axioms

$$X_{i,[n]} = \bigvee_{j=1}^{n} x_{i,j}, \qquad i = 1, \ldots, m.$$

To analyze the refutation P, we are going to allow also axioms [8] of the form

$$X_{i,J} := \bigvee_{j \in J} x_{i,j}, \qquad i \in \{1, \ldots, m\}$$

where $|J|$ exceeds a certain threshold d_i; the threshold will depend on the pigeon i. In this way we will be able to simplify the refutation P by "filtering out" of it all clauses C containing at least one such axiom (we just replace each such clause by the corresponding axiom). For this purpose we consider the "degree of freedom" $d_i(C)$ of each pigeon i in a clause C: this is the number of holes offered by C to this pigeon, that is, the number of holes j such that $x_{i,j} \in C$. The clause C is "filtered out" from the proof (i.e. can be replaced by an axiom) if $d_i(C) \geq d_i$ for at least one pigeon i.

Our first task (Fact 1 below) will be to show that if the thresholds d_i are chosen in a clever way, then in every clause C passing this filter *almost all*, namely, at least $m - O(\log |P|)$ *pigeons pass it safely*, i.e., their "degree of freedom" in C is well below the corresponding threshold d_i (is $\leq d_i - \delta$ for particular δ). Thus, the number of pigeons who *narrowly* (= non-safely) pass the filter $(d_1, \ldots, d_m)$ is at most $O(\log |P|)$. The second task (Fact 2 below) will be to show that the number of pigeons who *narrowly* passed the filter must be at least $\Omega\left(n/(\log |P|)^3\right)$. This implies $\log |P| \geq \Omega(n^{1/4})$, as desired.

Now let us formulate these two tasks more precisely.

---

[7] Buss and Pitassi proved that the converse also holds: If $\neg PHP_n^m$ has a monotone refutation of length L, then $\neg PHP_n^m$ also has a resolution refutation of length at most $L^{O(1)}$.

[8] This general and often very useful trick is known as the "Contrivances Paradox:" stronger statements are sometimes easier to prove, because their analysis becomes more robust. (Proof of Theorem 27.5 from the book gives another illustration of this trick.)

For a clause C let $J_i(C)$ be the set of holes *offered* by C to the i-th pigeon,

$$J_i(C) := \{j : x_{i,j} \text{ occurs in } C\}.$$

and let

$$d_i(C) := |J_i(C)|$$

be the number of these holes, the *degree of freedom* of the i-th pigeon in the clause C.

Given a vector $d = (d_1, \ldots, d_m)$ of positive integers, a *d-axiom* is an arbitrary clause of the form $X_{i,J}$ where $|J| \geq d_i$; we will also say that $X_{i,J}$ is the axiom for the i-th pigeon. Note that every monotone refutation of $\neg PHP_n^m$ is a monotone refutation of the set of d-axioms for $d_1 = \ldots = d_m = n$.

> **Ex. 5.7.** Let $m \geq n+1$ and $\mathcal{A}$ be a set of d-axioms. Suppose that it has at least one axiom for every pigeon. Show that $\mathcal{A}(a) = 0$ for all legal assignments a.

Let δ be another parameter. Say that the degree of freedom $d_i(C)$ of the i-th pigeon in a clause C is *large* if $d_i(C) \geq d_i - \delta$. We let [9]

$$I_{d,\delta}(C) := \{i : d_i(C) \geq d_i - \delta\}$$

to denote the set of those pigeons whose degree of freedom in C is not far away from the threshold d_i.

Definition 5.8. [10] The *pseudo-width* $w_{d,\delta}(P)$ of the refutation P is the maximum number of pigeons whose degree of freedom is large in at least one clause $C \in P$, that is, $w_{d,\delta}(P)$ is the maximum, over all $C \in P$, of

$$w_{d,\delta}(C) := |I_{d,\delta}(C)|.$$

Razborov's proof of Theorem 5.6 consists of two steps, establishing the following two facts.

Fact 1. *If $\neg PHP_n^m$ has a monotone refutation of length L then there exist a vector $d = (d_1, \ldots, d_m)$ with all $d_i > n/\log m$, and a set $\mathcal{A}$ of d-axioms such that $|\mathcal{A}| \leq L$, and $\mathcal{A}$ has a monotone refutation P of length at most L and pseudo-width*

$$w_{d,n/\log m}(P) \leq O(\log L).$$

Fact 2. *Let $\mathcal{A}$ be an arbitrary set of d-axioms and $\delta < d_i$ for all i. If P is a monotone refutation of $\mathcal{A}$, then*

$$w_{d,\delta}(P) \geq \Omega\left(\frac{\delta^2}{n \cdot \log|\mathcal{A}|}\right).$$

> **Ex. 5.9.** Show that these two fact imply that if $\neg PHP_n^m$ has a monotone refutation of length L, then $\log L = \Omega(n^{1/4})$. *Hint*: We may assume that $m \leq 2L$. (Why?)

---

[9] Note that the universes of $J_i(C)$ and $I_{d,\delta}(C)$ are different! The first consist of *holes* whereas the second consists of *pigeons*.

[10] Usually, the width of a refutation is defined as the maximum length of (=the number of literals in) its clauses. However, in the case of pigeonhole principle, such (natural) notion of width does not work (see Final Remark bellow).

5.2.1. Short proofs have small pseudo-width

To prove Fact 1 we have to somehow "filter out" clauses of large pseudo-width. For this we need the following combinatorial lemma (we give its proof in Sect. 5.3):

Lemma 5.10. (Pigeon filter lemma) *Let $R = \{r_{i,k}\}$ be an $m \times L$ matrix with integer entries. If L is sufficiently large, then there exists a sequence $r_1, \ldots, r_m$ of integers such that $r_i < \lfloor \log m \rfloor$ and for every column k at least one of the following two events happen:*

(i) *$r_{i,k} \le r_i$ for at least one row i;*
(ii) *$r_{i,k} \le r_i + 1$ for at most $O(\log L)$ rows i.*

Suppose now that $\neg PHP_n^m$ has a monotone refutation P' of length L. Set $\delta := n/\log m$, and consider the $m \times L$ matrix $R = \{r_{i,C}\}$ whose rows are pigeons i and columns are clauses $C \in P'$, and

$$r_{i,C} := \left\lfloor \frac{n - d_i(C)}{\delta} \right\rfloor + 1.$$

Let $r_1, \ldots, r_m$ be a sequence of integers guaranteed by Lemma 5.10. Set

$$d_i := \lfloor n - \delta r_i \rfloor + 1,$$

and note that $d_i > \delta$ because $r_i < \log m$.

Now take an arbitrary clause $C \in P'$. Our goal is to show that either or C contains a d-axiom (and C can be replaced by that axiom which reduces its pseudo-with $w_{d,\delta}(C)$ to 1) or $w_{d,\delta}(C) \le O(\log L)$. To show this, consider the column of M, corresponding to the clause C.

If the first case (i) in Lemma 5.10 takes place, then

$$\left\lfloor \frac{n - d_i(C)}{\delta} \right\rfloor + 1 = r_{i,C} \le r_i$$

for some pigeon i. This implies $d_i(C) \ge d_i$, because d_i is the minimal integer for which $\lfloor \frac{n-d_i}{\delta} \rfloor + 1 \le r_i$. Thus, in this case, C contains a subclause $X_{i,J}$ which is a d-axiom, and can be replaced by this axiom.

In the second case (ii) the number of pigeons i, for which $r_{i,C} \le r_i + 1$, does not exceed $O(\log L)$. By the definition, $w_{d,\delta}(C)$ is the number of those pigeons i for which $d_i(C) \ge d_i - \delta$. For each such pigeon i we have $\left\lfloor \frac{n-d_i(C)}{\delta} \right\rfloor \le \lfloor \frac{n-d_i+\delta}{\delta} \rfloor \le r_i$ or, equivalently, $r_{i,C} \le r_i + 1$. Thus, the number of such pigeons i, and hence, the pseudo-width $w_{d,\delta}(C)$ of C, cannot exceed $O(\log L)$. □

5.2.2. Pigeonhole proofs have large pseudo-width

Now we prove Fact 2.

Let $d = (d_1, \ldots, d_m)$ be an integer vector, $\mathcal{A}$ be an arbitrary set of d-axioms $\mathcal{A}$, and δ be any number such that $\delta < d_i$ for all i. Let

$$w_0 := \frac{\varepsilon \delta^2}{n \log |\mathcal{A}|}$$

where $\varepsilon > 0$ is a sufficiently small constant. Take an arbitrary monotone refutation P of $\mathcal{A}$. We will show that $w_{d,\delta}(P) > w_0$.

Suppose the opposite, i.e., that $w_{d,\delta}(C) \le w_0$ for all clauses $C \in P$. Our goal is to show that then $0 \notin P$ (i.e., that P is *not* a refutation of $\mathcal{A}$).

For an assignment a to the variables $x_{i,j}$, let

$$J_i(a) := \{j \ : a_{i,j} = 1\}$$

and

$$\ell := \lfloor \frac{\delta}{4w_0} \rfloor.$$

Let D be the set of all legal assignments[11] a such that $|J_i(a)| \leq \ell$ for all pigeons i. Note that legality of a implies that [12] $J_{i_1}(a) \cap J_{i_2}(a) = \emptyset$ for all $i_1 \neq i_2$.

For a set Γ of positive clauses and another positive clause C, let $\Gamma \models C$ mean that every assignment $a \in D$ satisfying all clauses from Γ also satisfies C. In this case we say that Γ *semantically implies* C.

Recall that each clause in $\mathcal{A}$ has the form $X_{i,J} := \bigvee_{j \in J} x_{i,j}$ for some pigeon i and some set J of $|J| \geq d_i$ holes; $X_{i,J}$ is the axiom for the pigeon i. Let $\mathcal{A}_i$ denote the set of all such axioms in $\mathcal{A}$, and let $\mathcal{A}_I := \bigcup_{i \in I} \mathcal{A}_i$. For a clause C, let $\mathcal{A}_C := \mathcal{A}_{I_{d,\delta}(C)}$ where, as before,

$$I_{d,\delta}(C) = \{i \ : d_i(C) \geq d_i - \delta\}$$

is the set of those pigeons i for which $d_i(C) \geq d_i - \delta$. That is,

$$\mathcal{A}_C = \{X_{i,J} \ : |J| \geq d_i \text{ and } i \text{ is such that } d_i(C) \geq d_i - \delta\}$$

contains axioms of only those pigeons who are "free enough" in the clause C.

For a clause $C \in P$ we will show, by induction on the number of steps in the derivation of C, that

$$\mathcal{A}_C \models C.$$

This gives the desired contradiction, because $\delta < d_i$ for all i implies that $I_{d,\delta}(0) = \emptyset$; hence, $\mathcal{A}_0 \not\models 0$ because $\mathcal{A}_0 = \emptyset$, implying that $0 \notin P$ (i.e., P is not a refutation of $\mathcal{A}$).

Base case $C \in \mathcal{A}$ is obvious since then $C \in \mathcal{A}_C$.

Inductive step. Suppose that $\mathcal{A}_{C_0} \models C_0$, $\mathcal{A}_{C_1} \models C_1$ and C is obtained from C_0, C_1 by a single application of the monotone refutation rule.

Since the rule is sound on D, $\{C_0, C_1\} \models C$. Hence, if we take

$$I = I_{d,\delta}(C_0) \cup I_{d,\delta}(C_1),$$

then $|I| \leq 2w_0$ and $\mathcal{A}_I \models C$. Let us choose a minimal $I \subseteq \{1, \ldots, m\}$ such that $\mathcal{A}_I \models C$; then still $|I| \leq 2w_0$. We will show that, in fact, $I \subseteq I_{d,\delta}(C)$; this will obviously imply [13] $\mathcal{A}_C \models C$.

Assume the contrary, and pick an arbitrary $i_0 \in I \setminus I_{d,\delta}(C)$. Since I is minimal,

$$\mathcal{A}_{I \setminus \{i_0\}} \not\models C.$$

Hence, there is an assignment $a \in D$ which satisfies all clauses in $\mathcal{A}_{I \setminus \{i_0\}}$ but falsifies C. We may assume that $a_{i,j} = 0$ for all j and all $i \notin I \setminus \{i_0\}$ (because C is positive and none of such variables $x_{i,j}$ appears in $\mathcal{A}_{I \setminus \{i_0\}}$). Let now $J_1 := \{1, \ldots, n\} \setminus J_0$ where[14]

$$J_0 = \bigcup_{i \in I \setminus \{i_0\}} J_i(a) \cup J_{i_0}(C).$$

[11] That is, D is the set of all $m \times n$ matrices with at most ℓ 1's in each row.
[12] No two pigeons are sent by a to the same hole.
[13] Because $\Gamma \models C$ and $\Gamma' \supseteq \Gamma$ implies $\Gamma' \models C$.
[14] Recall that $J_{i_0} = \{j : x_{i_0,j} \in C\}$.

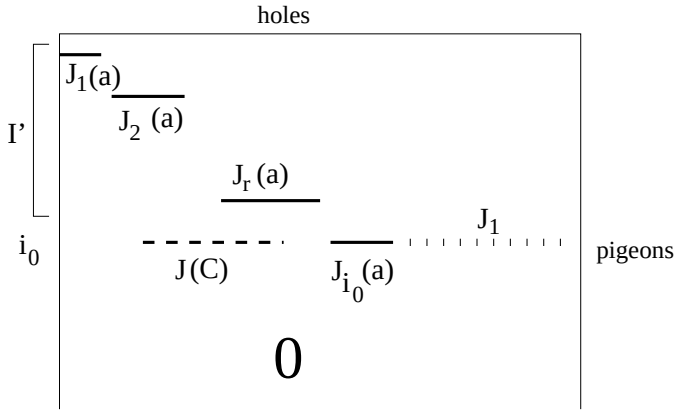

Figure 1. $I = I' \cup \{i_0\}$ with $I' = \{1, \ldots, r\}$

Since $i_0 \notin I_{d,\delta}(C)$, we know that $|J_{i_0}(C)| = d_{i_0}(C) < d_{i_0} - \delta$. Hence,

$$|J_1| \geq n - \big(|I| \cdot \ell + |J_{i_0}(C)|\big) \geq n - \big(2w_0\ell + (d_{i_0} - \delta)\big) \geq n - d_{i_0} + \delta/2. \tag{5.11}$$

By its definition, J_1 is the set of holes "permissible" for the pigeon i_0 (see Fig. 1): if we pick an arbitrary ℓ-element subset J of J_1 and change the assignment a by letting $a_{i_0,j} = 1$ iff $j \in J$, then we will get an assignment a' in D which still satisfies all clauses in $\mathcal{A}_{I\setminus\{i_0\}}$ (we have not touched other pigeons) but falsifies C. We want to show that J can be chosen in such a way that this new assignment will also satisfy all clauses in $\mathcal{A}_{i_0}$; this will give the desired contradiction with $\mathcal{A}_I \not\models C$.

For this purpose, pick $\mathbf{J}$ uniformly and at random among all ℓ-element subsets of J_1, and let $\mathbf{a}$ be the random assignment resulting from the assignment a by setting to 1 all $a_{i_0,j}$ with $j \in \mathbf{J}$.

Take an arbitrary axiom $A \in \mathcal{A}_{i_0}$. Since $|J_{i_0}(A)| \geq d_{i_0}$, by (5.11) we have

$$|J_{i_0}(A) \cap J_1| \geq \delta/2.$$

A fixed element appears in $J_{i_0}(A) \cap \mathbf{J}$ with probability

$$p = \frac{|\mathbf{J}|}{|J_1|} \leq \frac{\ell}{n - d_{i_0} + \delta/2}.$$

Hence, applying Chernoff's inequality we obtain

$$\Pr[A(\mathbf{a}) = 1] = \Pr[J_{i_0}(A) \cap \mathbf{J} \neq \emptyset] \geq 1 - e^{-\Omega(p\cdot\delta)} \geq 1 - e^{-\Omega(\delta\ell/n)}.$$

Since

$$\frac{\delta\ell}{n} \geq \frac{\delta^2}{4w_0 n} = \frac{\delta^2}{4n} \cdot \frac{n \cdot \log|\mathcal{A}|}{\varepsilon\delta^2} = \frac{\log|\mathcal{A}|}{4\varepsilon},$$

we obtain that

$$\Pr[A(\mathbf{a}) = 1] \geq 1 - |\mathcal{A}|^{-2},$$

if the constant ε is sufficiently small. Since clearly, $|\mathcal{A}_{i_0}| < |\mathcal{A}_{i_0}|^2 \leq |\mathcal{A}|^2$, this implies that, for at least one choice a' of $\mathbf{a}$, *all* axioms in $\mathcal{A}_{i_0}$ will be satisfied. Since (as we observed above) the assignment a' also satisfies all axioms in $\mathcal{A}_{I\setminus\{i_0\}}$ but falsifies C, we obtained a contradiction with $\mathcal{A}_I \not\models C$.

This completes the proof of Fact 2, and thus, the proof of Theorem 5.6 and Theorem 5.2. □

Final Remark. As we have seen, the whole proof is based on a concept of *pseudo-width* $w_{d,\delta}(P)$ of (monotone) resolution refutation P. This concept (as it was defined) makes sense only for resolution refutations of $\neg PHP_n^m$. On the other hand, one may define the *width* $w(P)$ of a resolution refutation P of an *arbitrary* CNF as the maximum number of literals in a clause from P. Ben-Sasson and Wigderson (1999) established a very general trade-off between the length $|P|$ of P and its width $w(P)$. For a CNF $\mathcal{F}$, let $w(\mathcal{F})$ (resp., $L(\mathcal{F})$) be the minimum width (resp., length) of a resolution refutation of $\mathcal{F}$. Then, for every unsatisfiable CNF $\mathcal{F}$,

$$w(\mathcal{F}) \leq O\left(\sqrt{N \cdot \log L(\mathcal{F})}\right) \qquad (*)$$

where N is the number of variables in $\mathcal{F}$. It is much easier to bound the width $w(\mathcal{F})$ than the length $L(\mathcal{F})$ and, remarkably, Ben-Sasson and Wigderson pointed out that (apparently) *all* lower bounds on $L(\mathcal{F})$ known at that time can be viewed [15] as lower bounds on $w(\mathcal{F})$ followed by applying the inequality (*). Unfortunately, this "width method" seemed to fail for CNFs $\mathcal{F}$ with a huge number of variables N, as is in the case of $\mathcal{F} = \neg PHP_n^m$ where $N = n \cdot m$ and m is much larger than n. To get a super-polynomial lower bound on the length $L(\mathcal{F})$ using (*) we would need a to show that the width $w(\mathcal{F})$ is at least the square root $\sqrt{N}$ of the *total* number of variables $n \cdot m$. Using pseudo-width instead of width for this CNF was crucial: in this case a lower bound for the pseudo-width of oder $n/\log m$ was enough!

5.3. Proof of the pigeon filter lemma

The lemma follows from the following property of randomly chosen numbers (show this!). Let m and L be positive integers. Set $t := \lfloor \log m \rfloor - 1$, and let $\mathbf{r}$ be a random variable taking its values in $\{1, \ldots, t\}$ with probabilities

$$\Pr[\mathbf{r} = t] = \frac{2}{2^t}$$

and

$$\Pr[\mathbf{r} = s] = \frac{1}{2^s} \quad \text{for each} \quad s = 1, 2, \ldots, t-1.$$

Ex. 5.12. Why we cannot take just $\Pr[\mathbf{r} = s] = 2^{-s}$ for *all* $s \leq t$? *Hint*: $\sum_{i=1}^{n} a^i = (a^{n+1} - a)/(a-1)$.

Lemma 5.13. *Let $x = (x_1, \ldots, x_m)$ be a sequence of integers from $\{1, \ldots, t\}$. Let $\mathbf{r}_1, \ldots, \mathbf{r}_m$ be m independent copies of $\mathbf{r}$. Then, with probability at least $1 - O(L^{-2})$ at least one of the following two events happens:*

(A_x) $\mathbf{r}_i \geq x_i$ *for at least one integer x_i;*
(B_x) $\mathbf{r}_i < x_i - 1$ *for all but at most $O(\log L)$ integers x_i.*

Proof. Our goal is to show that at least one of $\Pr[\neg A_x]$ and $\Pr[\neg B_x]$ is at most $O(L^{-2})$. Define the "weight" of x as $W(x) := \sum_{i=1}^{m} 2^{-x_i}$. We consider two cases depending on whether $W(x) \leq 2 \ln L$ or not.

Case 1. $W(x) \geq 2 \ln L$.

[15] Note that the proof of Haken's lower bound for $\neg PHP_n^{n+1}$ in the book also follows this frame.

Let $I = \{i : x_i \le t\}$ and note that $\sum_{i \notin I} 2^{-x_i} \le m2^{-(t+1)} \le 2$. Therefore, $\sum_{i \in I} 2^{-x_i} \ge W(x) - 2 \ge 2 \ln L - 2$. On the other hand, for every $i \in I$ we have

$$\Pr[\mathbf{r}_i < x_i] = \sum_{s=1}^{x_i - 1} \Pr[\mathbf{r} = s] = \sum_{s=1}^{x_i - 1} 2^{-s} \le 1 - 2^{-x_i}.$$

Since the events "$\mathbf{r}_i < x_i$" are independent,[16] and $\Pr[\mathbf{r}_i < x_i] = 1$ for all $i \notin I$, we have that in this case

$$\Pr[\neg A_x] = \Pr[\forall i : \mathbf{r}_i < x_i] = \prod_{i \in I} \left(1 - 2^{-x_i}\right) \le \exp\left(-\sum_{i \in I} 2^{-x_i}\right) \le O(L^{-2}).$$

Case 2. $W(x) \le 2 \ln L$.

We first show that $\Pr[\mathbf{r} \ge x_i - 1] \le 2^{2-x_i}$ for every i. Indeed, if $x_i > t$ then either $x_i = t + 1$ and $\Pr[\mathbf{r} \ge x_i - 1] = \Pr[\mathbf{r} = t] = 2^{1-t} = 2^{2-x_i}$, or $x_i \ge t + 2$ and $\Pr[\mathbf{r} \ge x_i - 1] = 0$. If $x_i \le t$ then

$$\Pr[\mathbf{r} \ge x_i - 1] = \sum_{s=x_i-1}^{t} 2^{-s} \le 2^{1-x_i} \sum_{j=0}^{\infty} 2^{-j} \le 2^{2-x_i}.$$

Hence, the expected number of i for which $\mathbf{r}_i \ge x_i - 1$ does not exceed

$$\sum_{i=1}^{m} 2^{2-x_i} = 4 \sum_{i=1}^{m} 2^{-x_i} = 4W(x) \le 8 \ln L.$$

Since these events "$\mathbf{r}_i \ge x_i - 1$" are independent, we may apply Chernoff's inequality and conclude that, for any sufficiently large constant C,

$$\Pr[\neg B_x] = \Pr[|\{i : \mathbf{r}_i \ge x_i - 1\}| \ge C \log L] \le L^{-2}.$$

□

Let now $R = \{r_{i,k}\}$ be an $m \times L$ matrix with integer entries. If $x = (x_1, \ldots, x_m)$ is a column of R, the say that the (random) sequence $\mathbf{r}_1, \ldots, r_m$ is *bad* for this column if neither of the events A_x and B_x happens. By Lemma 5.13, the sequence is bad for one (fixed) row of R with probability at most $p = O(L^{-2})$. Since we have only L columns, the sequence is good (not bad) for *all* columns with probability at least $1 - L \cdot p = 1 - O(1/L)$. If L is large enough, this probability is nonzero. Hence, there is a particular choice of the sequence $r_1, \ldots, r_n$ which is good for all columns of R.

This completes the proof of Lemma 5.10.

[16] Each r_i is an *independent* copy of $\mathbf{r}$.

TOPIC 6
The Mystery of Negations

(Related sections: 9, 10.6)

Summary: Monotone boolean functions can be computed by circuits with no negation gates. At the cost of a small increase in size, *every* boolean function on n variables can be computed using only $\log n$ negation gates. There are explicit *monotone* functions that *can* be computed by polynomial-size circuit using $\log n$ negations, but every circuit with $(1-\varepsilon)\log n$ negations must have exponential size.

Let us consider boolean circuits with And, Or and Not gates. Such a circuit C is just a sequence $C = (g_1, \ldots, g_t)$ of boolean functions $g_i : \{0,1\}^n \to \{0,1\}$ where each g_i is either a constant 0 or 1, or one of the input variables $x_1, \ldots, x_n$, or is obtained from the previous functions by applying an Or, And or Not operation. The length t of this sequence is the *size* of the circuit. Given a multi-output function $f : \{0,1\}^n \to \{0,1\}^k$, we may consider it as the family of boolean (i.e., single-output) functions $f_1, \ldots, f_k$ such that, for every input $a \in \{0,1\}^n$, $f_i(a)$ is the i-th bit of the output $f(a)$. A circuit computes such a function f if all the functions $f_1, \ldots, f_k$ belong to it.

The main difficulty in proving non-trivial lower bounds on the size of circuits is the presence of Not gates. The effect of such gates on circuit size remains to a large extent a mystery. It is, therefore, worth to recall what we actually know about this "ghost." Among the basic questions concerning the role of Not gates are the following:

- Given a function f, what is the minimum number $M(f)$ of Not gates in a circuit computing f?
- Given a circuit, to what extend can we decrease the number of Not gates in it without a substantial increase in its size?
- Suppose that a function f in n variables *can* be computed by a circuit of size polynomial in n, but every circuit with $M(f)$ negations computing f requires superpolynomial size. What is then the minimal number $R(f)$ of negations sufficient to compute f in polynomial size? In other words, how many Not gates do we need in oder to achieve superpolynomial savings in circuit size?

In this lecture we give some (partial) answers to these questions.

6.1. Markov's theorem

More than 30 years ago, Markov (1957) has made an intriguing observation that *every* Boolean function on n variables can be computed by a circuit with only

$\lceil \log(n+1) \rceil$ negations.[1] In fact, he announced a stronger result. A *multivalued* function on n variables is just a function $f : \{0,1\}^n \to \{0,1\}^k$. We can consider such a (mult-output) function as the sequence $f = (f_1, \ldots, f_k)$ of Boolean (i.e., single-output) functions such that, for every input $a \in \{0,1\}^n$, $f_i(a)$ is the i-th bit of the output $f(a)$. If $k = 1$, then the function is *boolean*. For two binary vectors $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ we write $x \leq y$ if $x_i \leq y_i$ for all i. We also write $x < y$ if $x \leq y$ and $x_i < y_i$ for at least one i. A function f is *monotone* if $x \leq y$ implies $f(x) \leq f(y)$.

A *chain* in the binary n-cube is an increasing sequence $y^1 < y^2 < \cdots < y^k$ of vectors in $\{0,1\}^n$. The *decrease* $d_Y(f)$ of a function $f = (f_1, \ldots, f_k)$ on a chain Y is the number of indices i such that $f(y^i) \not\leq f(y^{i+1})$. [2] The *decrease* $d(f)$ of f is the maximum of $d_Y(f)$ over all chains Y. In [**20**] Markov announced the following surprisingly tight result.

Theorem 6.1. (Markov 1957) *For every function f, the minimum number of Not gates contained in a circuit computing f is precisely $M(f) := \lceil \log(d(f)+1) \rceil$.*

Proof. Let $\mathrm{neg}(f)$ denote the minimum number of negation gates contained in a circuit computing f. Our goal is to prove that, for every function f,

$$\mathrm{neg}(f) = \lceil \log(d(f)+1) \rceil.$$

For simplicity, we consider only boolean functions f (the case of multi-output functions is similar and we leave it as an exercise). We first prove the **lower bound**:

$$\mathrm{neg}(f) \geq \lceil \log(d(f)+1) \rceil.$$

Fix a chain $Y = \{y^1 < y^2 < \cdots < y^k\}$ for which $d_Y(f) = d(f)$, and let $I(f) = \{i : f(y^i) > f(y^{i+1})\}$ be the set of the (indices of) *inverses* made by function f on this chain; hence, $|I(f)| = d_Y(f) = d(f)$. Take an arbitrary circuit C computing f, and let r be the number of negation gates in it. Our goal is to show that $r \geq \lceil \log(|I(f)|+1 \rceil$.

To show this, let g be the function computed on the output of the first negation gate of C. Since the function computed at the input of this gate is monotone, $d_Y(g) \leq 1$, i.e., g can make at most one inverse on Y. If $d_Y(g) = 0$ then $g \equiv 0$ or $g \equiv 1$, and we can replace this negation gate by a constant. Otherwise, there is an i_0 such that $g(y^i) = 1$ for all $i \in I_1 = \{1, \ldots, i_0\}$, and $g(y^i) = 0$ for all $i \in I_0 = \{i_0+1, \ldots, k\}$. Depending on whether $|I_1 \cap I(f)| \geq |I(f)|/2$ or not, replace the gate g by constant 1 or by constant 0. In both cases the resulting circuit has one negation gate fewer, and computes a function f_1 for which $d_Y(f_1) \geq |I(f)|/2 = d(f)/2$.

Now, if the original circuit C would have $r \leq \lceil \log(|I(f)|+1 \rceil - 1$ negations, then repeating this argument r times we would obtain a circuit without any negations which computes a function f_r for which $d_Y(f_r) \geq 1$. But the function f_r is monotone, a contradiction.

Let us now prove[3] the **upper bound**:

$$\mathrm{neg}(f) \leq \lceil \log(d(f)+1) \rceil.$$

[1] Here and in what follows, all logarithms are base two; hence, $\lceil \log(n+1) \rceil$ is the number of bits in the binary representation of n.

[2] That is, the number of indices i such that for some j, $f_j(y^i) > f_j(y^{i+1})$.

[3] We follow the argument due to Fischer (1996).

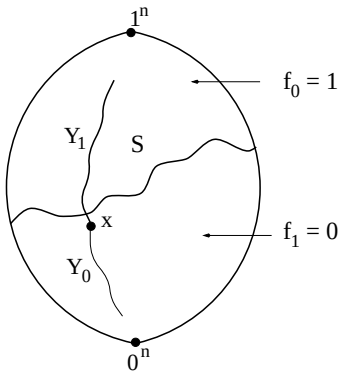

Figure 1. Chain Y_0 ends in x, and Y_1 starts with x.

We will prove the upper bound by induction on

$$M(f) := \lceil \log(d(f) + 1) \rceil.$$

Base: $M(f) = 0$. Then $d(f) = 0$, so f is monotone and $\mathrm{neg}(f) = 0$.

Induction Step: Suppose $\mathrm{neg}(f') \leq M(f')$ for all boolean functions f' such that $M(f') \leq M(f) - 1$.

Let S be the set of all vectors $x \in \{0,1\}^n$ such that for every chain Y *starting* with x we have

$$d_Y(f) < 2^{M(f)-1}. \tag{1}$$

We can show that for every chain Y *ending* in a vector outside the set S we also have

$$d_Y(f) < 2^{M(f)-1}. \tag{2}$$

To show this, assume that there is a chain Y_0 ending in a vector $x \notin S$ and such that $d_{Y_0}(f) \geq 2^{M(f)-1}$ (see Fig. 1). The fact that x does not belong to S means that there must be a chain Y_1 starting in x for which $d_{Y_1}(f) \geq 2^{M(f)-1}$. But then the decrease $d_{Y_0 \cup Y_1}(f)$ of f on the combined chain $Y_0 \cup Y_1$ is

$$d_{Y_0 \cup Y_1}(f) = d_{Y_0}(f) + d_{Y_1}(f) \geq 2^{M(f)} = 2^{\lceil \log(d(f)+1) \rceil} > d(f),$$

a contradiction with the definition of $d(f)$.

Consider now two functions f_0 and f_1 defined as follows:

$$f_1(x) = \begin{cases} f(x) & \text{if } x \in S \\ 0 & \text{if } x \notin S \end{cases}$$

and

$$f_0(x) = \begin{cases} 1 & \text{if } x \in S \\ f(x) & \text{if } x \notin S \end{cases}$$

By (1) and (2) we have $d(f_i) \leq 2^{M(f)-1} - 1$ for both $i = 0, 1$. Hence,

$$M(f_i) = \lceil \log(d(f_i) + 1) \rceil \leq \lceil \log 2^{M(f)-1} \rceil = M(f) - 1.$$

By the induction hypothesis, $\mathrm{neg}(f_i) \leq M(f_i) \leq M(f) - 1$ for both $i = 0, 1$. It therefore remains to show that

$$\mathrm{neg}(f) \leq 1 + \max\{\mathrm{neg}(f_0), \mathrm{neg}(f_1)\}. \tag{3}$$

For this we need one auxiliary result. Let $\mu(y, y', x)$ be a boolean function in $n+2$ variables $y, y', x_1, \ldots, x_n$. Say that μ is a *connector* of two boolean functions $f_0(x)$ and $f_1(x)$ if for $i = 0, 1$

$$\mu(i, \neg i, x) = f_i(x),$$

that is, if $\mu(0, 1, x) = f_0(x)$ and $\mu(1, 0, x) = f_1(x)$.

Claim 6.2. *Every pair of functions $f_0(x)$, $f_1(x)$ has a connector μ such that* $\mathrm{neg}(\mu) \leq \max\{\mathrm{neg}(f_0), \mathrm{neg}(f_1)\}$.

Proof of Claim. We argue by induction on $r = \max\{\mathrm{neg}(f_0), \mathrm{neg}(f_1)\}$. If $r = 0$ then both functions f_i are monotone, and we can take $\mu(y, y', x) = (y \wedge f_1) \vee (y' \wedge f_0)$. *Induction Step.* Let $C_i(x)$ be a circuit with $\mathrm{neg}(f_i)$ negations computing $f_i(x)$. Replacing the first Not gate in C_i by a new variable z we obtain a circuit $C_i'(z, x)$ on $n+1$ variables which contains one Not gate fewer. Let $f_i'(z, x)$ be the function computed by this circuit; hence, $\mathrm{neg}(f_i') \leq r - 1$. Moreover, if $h_i(x)$ is the monotone function computed immediately before the first Not gate (in C_i) then

$$f_i(x) = f_i'(\neg h_i(x), x).$$

By the induction hypothesis, there is a boolean function $\mu'(y, y', z, x)$ (the connector of the pair f_0', f_1') such that $\mathrm{neg}(\mu') \leq \max\{\mathrm{neg}(f_0'), \mathrm{neg}(f_1')\} \leq r-1$ and for $i = 0, 1$

$$\mu'(i, \neg i, z, x) = f_i'(z, x).$$

It is easy to see that then the function $\mu(y, y', x)$ obtained from $\mu'(y, y', z, x)$ by replacing the variable z by the function[4]

$$Z(y, y', x) := \neg\big((y \wedge h_1(x)) \vee (y' \wedge h_0(x))\big)$$

is a connector of f_0 and f_1. Since the functions h_0 and h_1 are monotone, we have[5] $\mathrm{neg}(\mu) \leq 1 + \mathrm{neg}(\mu') \leq r$, as desired. □

Now we can finish the proof of the theorem as follows. Let $s(x)$ be the characteristic function of S. Note that $s(x)$ is monotone (show this!). Let μ be a connector of f_0 and f_1. Then

$$f(x) = \mu\big(s(x), \neg s(x), x\big)$$

and by Claim,

$$\mathrm{neg}(f) \leq 1 + \mathrm{neg}(\mu) \leq 1 + \max\{\mathrm{neg}(f_0), \mathrm{neg}(f_1)\}.$$

This completes the proof of (3), and thus, the proof of the theorem. □

Ex. 6.3. Show that the maximum of $M(f)$ over all functions on n variables is:
(i) $\lceil \log(n+1) \rceil$ in the case of multi-output functions, and
(ii) $\lfloor \log(n+1) \rfloor$ in the case of single-output (i.e., boolean) functions.

[4] $Z(0, 1, x) = \neg h_0(x)$ and $Z(1, 0, x) = \neg h_1(x)$

[5] In the first inequality we use the following simple fact: If $f = g(h_1(x), \ldots, h_m(x))$ then $\mathrm{neg}(f) \leq \mathrm{neg}(g) + \max_i \mathrm{neg}(h_i)$.

Ex. 6.4. Let $f : \{0,1\}^n \to \{0,1\}^k$ be a k-output function, and $f_1, \ldots, f_k$ be the corresponding family of boolean functions. Show that $d(f) \leq \sum_{i=1}^{k} d(f_i)$.

6.2. Fischer's theorem

According to Markov's theorem, every function in n variables *can* be computed by a circuit with at most

$$M(n) := \lceil \log(n+1) \rceil$$

Not gates. Fischer (1974) has shown that restricting the number of negations to $M(n)$ entails only a polynomial blowup in circuit size!

Theorem 6.5. (Fischer 1974)[6] *If a function on n variables can be computed by a circuit of size t, then it can be computed by a circuit of size at most $2t+O(n^2 \log^2 n)$ using at most $M(n)$ Not gates.*

Proof. It is easy to show (do this!) that every circuit of size t can be transformed to a circuit of size at most $2t$ such that all negations are placed only on the input variables. Hence, it is enough to show how to compute the (multi-output) function

$$NEG(x_1, \ldots, x_n) = (\neg x_1, \ldots, \neg x_n)$$

by a circuit of size $O(n^2 \log^2 n)$ using $M(n)$ negations.

A simple (but crucial) observation is that[7]

$$\neg x_i = f_i(x)$$

where[8]

$$f_i(x) := \bigwedge_{k=0}^{n} \left(\neg T_k^n(x) \vee T_{k,i}^n(x)\right)$$

and

$$T_{k,i}^n(x_1, \ldots, x_n) := T_k^{n-1}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n).$$

It can be shown (we will not do this) that all the functions T_k^n and $T_{k,i}^n$ $(0 \leq k \leq n, 1 \leq i \leq n)$ can be computed by a *monotone* circuit of size $O(n^2 \log^2 n)$. It remains to compute the function $\neg T(x) := \left(\neg T_1^n(x), \neg T_2^n(x), \ldots, \neg T_n^n(x)\right)$ using at most $\lceil \log(n+1) \rceil$ negations. To do this, we first take a monotone circuit $C_1(x)$ computing the function $T(x) := \left(T_1^n(x), T_2^n(x), \ldots, T_n^n(x)\right)$. Observe that the outputs of this circuit belong to the set A_{sort} of all inputs $y \in \{0,1\}^n$ whose bits are *sorted* in decreasing oder $y_1 \geq y_2 \geq \cdots \geq y_n$. Using only $\lceil \log(n+1) \rceil$ negations it is possible to construct a circuit $C_2(y)$ of size $O(n)$ which computes $NEG(y)$ correctly on all inputs in A_{sort} (see Exercise 6.6). The resulting circuit $C(x) = C_2(C_1(x))$ computes $\neg T(x)$. □

Ex. 6.6. Let $n = 2^r - 1$. Construct a circuit C_n of size $O(n)$ which has at most r Not gates such that $C_n(y) = NEG(y)$ for all inputs $y \in A_{sort}$. *Hint*: Let $y = (y_1, \ldots, y_n) \in A_{sort}$. Take the middle bit y_m $(m = n/2)$ and show that the output of C_n can be obtained from the output of $C_{n/2}$ and the output of

[6] In fact, the term $O(n^2 \log^2 n)$ here can be replaced by $O(n \log n)$.

[7] Let $a \in \{0,1\}^n$. If $\neg x_i(a) = 1$ then $a_i = 0$, implying that in this case $T_k^n(a) = T_{k,i}^n(a)$ for all $k = 1, \ldots, n$, and hence, $f_i(a) = 1$. If $\neg x_i(a) = 0$ then $a_i = 1$. So, if we take $k := |a|$ (= the number of 1's in a) then $T_k^n(a) = 1$ and $T_{k,i}^n(a) = 0$, implying that $f_i(a) = 0$.

[8] Here, as before, $T_k^n(x_1, \ldots, x_n)$ is the k-th threshold function: $T_k^n(x) = 1$ iff x has at least k 1's.

$\neg y_m$. For this observe that $\neg y_m = 1$ implies $\neg y_j = \neg y_m$ for all $j > m$, whereas $\neg y_m = 0$ implies $\neg y_j = \neg y_m$ for all $j < m$.

Ex. 6.7. Fischer's theorem holds for every function f in n variables, but requires $\lceil \log(n+1) \rceil$ negations. Suppose that f outputs 1 on all inputs with at least k 1's, and outputs 0 on all inputs with at most l 1's. Show that then Fischer's simulation requires only $\lceil \log(k-l) \rceil$ negations.

6.3. Can Fischer's simulation be improved?

Markov's theorem motivates the following question.

> Suppose that a function f in n variables *can* be computed by a circuit of size polynomial in n (in this case we say that f is *feasible*), but every circuit with $M(f)$ negations computing f requires superpolynomial size. What is then the minimal number $R(f)$ of negations sufficient to compute f in polynomial size?

Fischer's result only implies that this number lies somewhere between the Markov's lower bound $M(f)$ and the generic upper bound $\lceil \log(n+1) \rceil$. The discrepancy between these two bounds is especially evident for functions with $M(f) = 0$, i.e., for monotone functions. Berkowitz [**7**] and Valiant [**33**] have shown that for so-called *slice* functions (these are monotone boolean functions which are non-trivial only on one slice of the n-cube) negations are powerless, i.e., cannot lead to a superpolynomial savings.[9] So, the first natural question was whether using Not gates can lead to superpolynomial savings in size *at all*?

This (long standing) problem was resolved by Razborov [**26**]: there exist explicit feasible monotone boolean functions f such that $R(f) > 0$. The function here is the characteristic function of bipartite graphs containing a perfect matching. Another example of such a function (requiring even larger monotone circuits) was given by Tardos [**32**]. The next natural question was: *how large does* $R(f)$ *actually is*? Or, in other words:

> How many Not gates do we actually need to achieve superpolynomial savings in circuit size?

This last question was considered by many authors under additional restrictions on the topology and/or on the use of Not gates. In particular, Okolnishnikova [**22**], and Ajtai and Gurevich [**4**] have shown that there exists monotone functions that can be computed with polynomial size, *constant* depth circuits, but can not be computed with *monotone*, polynomial size, constant depth circuits. This implies that $R(f) > 0$ in the class of AC^0 circuits. Moreover, it was shown by Satha and Wilson [**29**] that in this class we need much more than $\lceil \log(n+1) \rceil$ negations: there is a (multi-output) function computable in constant depth that cannot be computed in constant depth with $o(n/\log^{1+\varepsilon} n)$ negations. (Note that this result does not contradict with the Markov–Fischer upper bound: their simulation requires logarithmic depth.) Another line of research was to restrict the use of Not gates. For circuits of logarithmic depth, a lower bound $R(f) = \Omega(n)$ was proved by Raz and Wigderson [**25**] under the restriction that *all* the negations are placed on the input variables: there is an explicit monotone function (corresponding to the connectivity problem for graphs) that can be computed with polynomial size, depth $O(\log^2 n)$

[9] See Exercises 10.12 and 10.13 i the book.

circuits, but can not be computed with polynomial size, depth $k \log n$ circuits using only $o(n/2^k)$ negated variables.

However, if we put restrictions neither on the depth nor on the use of Not gates, then the only we knew so far was that some feasible monotone functions f_n require $R(f_n) > 0$ negations. It was even not clear[10] whether there are feasible monotone functions for which $R(f_n) \to \infty$ as $n \to \infty$.

In the next section we show that, somewhat surprisingly, the results of Razborov and Tardos can be used to move the threshold $R(f)$ (of the first superpolynomial decrease in size) much nearer to the Markov–Fischer bound $\lceil \log(n+1) \rceil$: there are (explicit) feasible monotone functions $f : \{0,1\}^n \to \{0,1\}^n$ such that $R(f) \geq \log n - c \log \log n$.

6.4. Feasible functions requiring many Not gates

The idea is very simple, and is based on the following observation. Let $C = C(Y_0, Y_1)$ be a circuit which computes two Boolean functions $f_0 = f_0(Y_0)$ and $f_1 = f_1(Y_1)$ on disjoint sets of variables, $Y_0 \cap Y_1 = \emptyset$. A *minterm* of a monotone boolean function is a minimal set of variables which, if assigned the value 1, forces the function to take the value 1 regardless of the values assigned to the remaining variables.

Observation 6.8. *If C has only one Not gate, then at least one of the two functions f_0, f_1 can be computed by a* monotone *circuit of the same (or smaller) size.*

Proof. Let g be the monotone(!) Boolean function computed at the input to the Not gate (i.e., immediately before this gate). Since g is monotone, all its minterms are positive (no negated literals). Hence, we have only two possibilities: either some minterm of g lies entirely in Y_1, or not. In the first case we assign constant 1 to all the variables in Y_1, whereas in the second case we assign constant 0 to all the variables in Y_0. As the function g is monotone, in both cases it turns into a constant function (constant 1 in the first case, and constant 0 in the second), and the subsequent Not gate can be eliminated. Since $Y_0 \cap Y_1 = \emptyset$, the setting $Y_{\overline{\varepsilon}} \to \varepsilon$ does not affect the function $f_{1-\varepsilon}$. Hence, we obtain a circuit which computes either f_0 or f_1 and contains no Not gates, i.e., is monotone. □

Observation 6.8 gets rid of only one Not gate. However, it can be directly extended to the case of many such gates by taking more copies of boolean functions.

Let $f = f(X)$ be a Boolean function in m variables $X = \{x_1, \ldots, x_m\}$, and $n = km$. A function $f_n : \{0,1\}^n \to \{0,1\}^k$ is a *k-fold extension* of f if it computes k copies of f on disjoint copies $X_1, \ldots, X_k$ of X. That is, given an input $(a^1, \ldots, a^k)$ with $a^i \in \{0,1\}^{X_i}$, the function outputs the sequence $\big(f(a^1), \ldots, f(a^k)\big)$. Important here is that the i-th output bit $f(a^i)$ is independent of inputs a^j for $j \neq i$. Note also that the extension of monotone functions are monotone.

Iterating the argument used in the proof of Fact 6.8 immediately yields the following

[10] A step in this direction was made by Amano and Maruoka [?] who proved that the clique function cannot be computed in polynomial size using only $(1/6) \log \log n$ negations. Unfortunately, this does not answer the question, because the clique function itself is not feasible (unless $P = NP$).

Observation 6.9. *Let f be a monotone Boolean function, and k be a nonnegative integer. If f cannot be computed by a monotone circuit of size t, then its k-fold extension cannot be computed by a circuit of size t using $\lceil \log(k+1) \rceil$ negations.*

As a corollary we obtain

Theorem 6.10. ([**16**]) *There exist explicit feasible monotone functions $f_n : \{0,1\}^n \to \{0,1\}^n$ such that $R(f_n) \geq \log n - 9 \log\log n$.*

Proof. Take an explicit monotone Boolean function T_m in m variables considered in [**32**]. As shown in [**32**], this function is feasible (can be computed by a non-monotone circuit of size $m^{O(1)}$) but every monotone circuit computing it requires size is exponential in $\Omega(m^{1/8})$. Let $n = km$ where $k = 2^r$ and

$$r = \left\lceil \log \frac{n}{(\log n)^9} \right\rceil$$

and consider the k-fold extension f_n of T_m. Then f_n can be computed by a (non-monotone) circuit of size at most $k \cdot m^{O(1)}$, which is, of course, polynomial in n. On the other hand, by Fact 7.9, every circuit with at most r Not gates computing f_n must have size exponential in $\Omega(m^{1/8}) = \Omega\left((2^{-r}n)^{1/8}\right) = \Omega\left((\log n)^{9/8}\right)$. Thus, $R(f_n) > r \geq \log n - 9 \log\log n$. □

One may ask what happens if besides Not gates we will allow, say, any monotone *real-valued* functions $\varphi : \mathsf{R}^2 \to \mathsf{R}$ as gates—does then the use of negations can still lead to a drastic decrease of size? The question makes sense because it is known[11] (see [**?**]) that there exist a lot of monotone boolean functions (slice functions) that can be computed by monotone real circuits (no Not gates) of linear size $O(n)$, but require boolean circuits with And, Or and Not gates of size $2^{\Omega(n)}$. On the other hand, it is shown in [**15**] that Tardos' function T_m requires also monotone real circuits of size $2^{\Omega(m^{1/8})}$. This means that the function f_n considered in Theorem 6.10 captures the role of negations in a quite strong sense: this function is feasible but cannot be computed by a circuit of polynomial size with fewer than $\log n - 9 \log\log n$ Not gates, even if we allow *all* monotone real-valued functions as gates.

The message of this theorem is that, in the context of the P vs. NP problem, it is important to understand the role of Not gates when their number r is *indeed* very close to the Markov–Fisher upper bound of $\lceil \log(n+1) \rceil$: moving by more than $\log\log n$ apart from this bound we essentially move by $\log n$ and reach the world of monotone circuits ($r = 0$) where Not gates play no role—they don't exist in this world at all!

> **Ex. 6.11.** (*Research problem*) Unlike Tardos' function T_m, the function f_n in Theorem 6.10 has many output bits. Find an explicit monotone *boolean* (i.e., single-output) function f_n for which $R(f_n) = \Omega(\log n)$.

When approaching this problem the following observation may be (apparently) useful.

Observation 6.12. *Let f be a boolean function and suppose that it can be computed by a circuit of size t with at most r negations. Then for any $A \subseteq f^{-1}(0)$ and $B \subseteq f^{-1}(1)$, there is a monotone boolean function g such that g can be computed*

[11] See Exercises 10.14 and 10.15 in the book.

by a monotone circuit of size at most t and either g or its negation $\overline{g}$ rejects a 2^{-r} fraction of inputs from A and accepts a 2^{-r} fraction of inputs from B.

Proof. We argue by induction on r. If $r = 0$, then we choose the circuit itself. If $r \geq 1$, then consider the first negation gate and the function g that is computed at the gate immediately before this negation gate. Let $\varepsilon \in \{0,1\}$ be such that $g(a) = \varepsilon$ for at least one half of the inputs $a \in A$. If also one half of the inputs $b \in B$ have $g(b) = \varepsilon \oplus 1$, then either g or $\overline{g}$ has the property stated in the lemma. Otherwise (if $g(b) = \varepsilon$ for one half of the inputs $b \in B$) we replace the negation gate by the constant $\varepsilon \oplus 1$ and use the induction hypothesis for the sets $A_1 \subseteq A$ and $B_1 \subseteq B$ of inputs on which g outputs the constant ε. □

Now if a boolean function f can be computed by a circuit with small number of negations then, according to Observation 7.1, there must be a *monotone* boolean function g such that either g or its negation $\overline{g}$ separates some two large enough subsets of inputs $A_0 \subseteq f^{-1}(0)$ and $A_1 \subseteq f^{-1}(1)$. If this pair is separated by the function g itself, then we can try to argue in a standard way to show that g cannot be t-simple if t is too small, and then apply the lower bounds criterion (Theorem 10.16 in the book). The problem, however, is when the pair A_0, A_1 is separated by $\overline{g}$. Then g also separates this pair, but does this in the "wrong direction:" $g(A_0) = 1$ and $g(A_1) = 0$. This is a problem because for the original function to be not t-simple the 0/1-terms of f must be scattered enough, meaning that they must be short (much shorter than $n/2$); hence, their complements must be long, implying that the corresponding terms of g may be no more scattered well enough.

TOPIC 7
Communication Complexity and Monotone Depth

(Related sections: 15.2, 18)

Summary: Minterm-maxterm games. Proof that s-t connectivity of directed n-vertex graph cannot be solved by monotone circuits of depth $o(\log^2 n)$. The amplification argument.

We will consider games between two players, Alice and Bob, where communication between players is allowed. However, we will assume that the players are far away from each other, so that each communicated bit costs a money. The goal is to pay (and hence to communicate) as few as possible. The players are not adversaries; they help and trust each other.

7.1. The Karchmer–Wigderson game

We will restrict ourselves with one particular type of games related to the depth of boolean circuits. Let A, B be two disjoint subsets of $\{0,1\}^n$. The Karchmer–Wigderson game on this pair is the following game.

- Alice gets a vector $a \in A$.
- Bob gets a vector $b \in B$.
- The goal is to find an i such that $a_i \neq b_i$.

Before the game starts, the players agree on a "protocol" for exchanging messages. After that, given an input pair (a, b), the protocol dictates to each player what messages to send at each point, based on his/her input and the messages be received so far. It also dictates when to stop, and how to determine the answer from the information received. There is no limit on the computational complexity of these decisions, which are free of charge. The cost of the protocol is the number of bits they have to exchange on the *worst case* choice of input pair (a, b). The *communication complexity* $C(A, B)$ of the game on the pair A, B is the cost of the best protocol for this game.

More formally, this measure can be defined as follows.

A *communication matrix* for this game is an $|A| \times |B|$ matrix M whose rows are indexed by vectors from A and columns are indexed by vectors from B. The entry, corresponding to the row a and column b, is the set $M(a, b) = \{i : a_i \neq b_i\}$. By sending bits 0 and 1, the players actually split the rows (if this bit is send by Alice) or columns (if this bit is send by Bob) into two disjoint parts.

A *communication protocol* (or a *communication tree*) T for this game is a binary tree, each inner node of which correspond to a decision made by one of the players at this node. The tree has the following properties:

1. The root is labeled by the whole communication matrix M.

2. If a node u is labeled by a matrix N, then the sons of u are labeled by the corresponding submatrices N_0 and N_1 of N. Moreover, these submatrices are obtained from N by splitting the rows of N (if u is Alice's node) or by splitting the columns of N (if u is Bob's node).
3. If w is a leaf and N is its label, then the entries of N must share a common element.[1]

Since at each node, the rows (or columns) of the corresponding submatrix are splitted into *disjoint* parts, the protocol is *deterministic*: each pair (a,b) will reach precisely one leaf. The depth of a tree is the maximum number of edges from the root to a leaf. The minimum depth of a communication tree is the *communication complexity* $C(A,B)$ of the game on the pair A,B. The *communication complexity* $C(f)$ of a boolean function f is $C(A,B)$ with $A=f^{-1}(0)$ and $B=f^{-1}(1)$.

7.1.1. Communication complexity and the cover number

Let M be the communication matrix of the KW game on a pair A,B. Recall that the entry, corresponding to the row $a\in A$ and column $b\in B$, is the set $M(a,b)=\{i : a_i\neq b_i\}$. A *rectangle* is a submatrix R of M such that all its entries share a common element. The cover number $\alpha(A,B)$ of a pair A,B is the minimum number of mutually disjoint rectangles whose union cover whole communication matrix M of this pair.

Proposition 7.1. *For every pair A,B, $C(A,B)\geq \log\alpha(A,B)$.*

> **Ex. 7.2.** Prove this proposition. *Hint*: Note that each leaf of a communication protocol gives us a rectangle.

7.1.2. Communication complexity and the depth of circuits

Let $D(f)$ be the minimum depth of a formula with AND, OR and NOT gates computing f. The following theorem connects communication with computation.

Theorem 7.3 (Karchmer–Wigderson 1988). *For every Boolean function f,*

$$D(f)=C(f).$$

We prove lower and upper bounds on $D(f)$ separately.

Lemma 7.4. $C(f)\leq D(f)$.

Proof. We may assume that Alice and Bob have agreed on a formula of depth $D(f)$ computing f. Further, we may assume, using de Morgan's laws, that negations are applied only to the variables (leaves) and, thus, that all gates are ANDs and ORs.

Now suppose Alice gets an input a such that $f(a)=0$, and Bob gets an input b such that $f(b)=1$. In oder to find an i such that $a_i\neq b_i$, the players use the information provided by the underlying formula. At AND gates speaks Alice, and at OR gates speaks Bob.

Suppose the output gate is an AND-gate, that is, we can write $f=f_0\wedge f_1$. Then Alice sends a bit i corresponding to a function f_i such that $f_i(a)=0$; if both $f_0(a)$ and $f_1(a)$ output 0, then Alice send 0. We know that we must have $f_i(b)=1$. We can then repeat this step at the gate corresponding to the output gate of f_i, where Alice sends a bit if the gate is an AND gate and Bob sends a bit if the gate is an OR gate (he sends a bit corresponding to a function which outputs 1). Alice and

[1] Recall that entries are *subsets* of $\{1,\ldots,n\}$.

Bob repeat this process until they reach a leaf. This leaf is labeled by some variable x_i or its negation $\overline{x}_i$. Hence, $a_i \neq b_i$ implying that i is a correct answer. □

Lemma 7.5. *If $A, B \subseteq \{0,1\}^n, A \cap B = \emptyset$, and $C(A,B) \leq c$, then there exists a function $f : \{0,1\}^n \to \{0,1\}$ such that $f(A) = 0, f(B) = 1$, and $D(f) \leq c$. In particular, $D(f) \leq C(f)$.*

Proof. We prove the lemma by induction on c. Suppose $c = 0$. Then we must have, for some index i, that $a_i \neq b_i$ for all pairs (a,b) in $A \times B$. Thus we may choose either $f = x_i$ or $f = \overline{x_i}$ according to which function satisfies $f(A) = 0$ and $f(B) = 1$.

Next, we prove the claim is true for c assuming it is true for $c-1$. Consider a protocol for the communication game on A, B that uses at most c bits. Let us assume Alice sends the first bit. Then there is a partition $A = A_0 \cup A_1, A_0 \cap A_1 = \emptyset$, such that for $a \in A_0$, Alice sends the bit 0 and for $a \in A_1$, Alice sends the bit 1. We then know that $C(A_0, B) \leq c-1$ and $C(A_1, B) \leq c-1$. Applying our induction hypothesis, we find there exists a function f_0 such that $f_0(A_0) = 0, f_0(B) = 1$, and $D(f_0) \leq c-1$, and there exists a function f_1 such that $f_1(A_1) = 0, f_1(B) = 1$, and $D(f_1) \leq c-1$. We define $f = f_0 \wedge f_1$. Then $f(A) = 0, f(B) = 1$, and

$$D(f) \leq 1 + \max\{D(f_0), D(f_1)\} \leq c$$

as desired. Note that, if Bob had sent the first bit, we would have partitioned B and defined $f = f_1 \vee f_2$. □

In a *monotone version* of Karchmer–Wigderson game on a pair A, B, given an input pair (a,b) in $A \times B$, the players must find an i such that $a_i = 0$ and $b_i = 1$. Note that, in general, this game may be not well defined: if a is the all-1 vector and b is the all-0 vector, then no valid answer exists. However, if $A \subseteq f^{-1}(0)$, $B \subseteq f^{-1}(1)$ and if f is a *monotone* boolean function, then every pair (a,b) has a valid answer.

For a monotone boolean function f, let $C_+(f)$ be the communication complexity of a monotone version of the Karchmer–Wigderson game on the pair A, B with $A = f^{-1}(0)$ and $B = f^{-1}(1)$. Let $D_+(f)$ be the minimum depth of a boolean formula with AND and OR gates computing f.

Theorem 7.6. *For every monotone boolean function f, $C_+(f) = D_+(f)$.*

Ex. 7.7. Prove this theorem.

7.2. The st-connectivity problem

We consider directed graphs on $m+2$ vertices with two exhibited vertices s (the source) and t (the target). The *st-connectivity problem* $STCON_n$ is, given a directed graph, to determine whether it contains a path from s to t. Obviously, $STCON_m$ is a monotone function: if we add edges we cannot disconnect an existing path from s to t.

In this lecture we will show that any monotone circuit solving this problem has depth $\Omega((\log m)^2)$. We will do this by proving this lower bound on the communication complexity of the corresponding Karchmer–Wigderson game: [2]

[2]For convenience, we have reversed the roles of players (now Alice gets inputs from $f^{-1}(1)$, not from $f^{-1}(0)$). This does not change anything: since we consider only *deterministic* games, a function and its complement have the same complexity.

- Alice gets a graph G with $STCON(G) = 1$.
- Bob gets a graph H with $STCON(H) = 0$.
- The goal is to find an edge (u, v) which is present in G but is absent in H.

Since we are interested in proving *lower* bounds on the communication complexity of this game, we can restrict our attention to special inputs.

- Alice gets a directed path p form s to t.
- Bob gets a coloring c of vertices by the colors 0 and 1 such that $c(s) = 0$ and $c(t) = 1$.
- The goal is to find an edge $(u, v) \in p$ such that $c(u) = 0$ and $c(v) = 1$.

Let $C(STCON_m)$ denote the communication complexity of this game. Note that every protocol for the KW game can be used to solve this (restricted) game: given a coloring c, Bob converts it into a graph H in which (u, v) is an edge iff $c(u) = c(v)$.

Ex. 7.8. Prove that $C(STCON_m) = O((\log m)^2))$. *Hint*: Use binary search; in fact one of the players may do most of the talking, with the other player communicating only $O(\log n)$ bits overall.

7.3. The fork game

For positive integers n and l, let $[n]^l$ denote the set of all strings $a = (a_1, \ldots, a_l)$ with $a_i \in [n] = \{1, \ldots, n\}$. A *fork position* of two strings $a = (a_1, \ldots, a_l)$ and $b = (b_1, \ldots, b_l)$, a *fork point* is a position i such that $a_i \neq b_i$ and, if $i > 1$ then $a_{i-1} = b_{i-1}$.

The fork game on a subset $S \subseteq [n]^l$ is the following game:

- Alice gets a string $a = (a_1, \ldots, a_l) \in S$.
- Bob gets a string $b = (b_1, \ldots, b_l) \in S$.
- The goal is to find a fork position i of a and b. If $a_l = b_l$ then $i = l$ is also a legal answer.

Given a pair of strings $a, b \in [n]^l$, the players follow some agreed-upon deterministic protocol of communicating bits back and forth until in the end they have agreed on an answer $i \in \{0, 1, \ldots, l\}$. The communication complexity of a protocol is the total number of communicated bits on the worth-case input pair a, b.

Let $C(FORK_{n,l})$ denote the communication complexity of the fork game game on the whole set $S = [n]^l$.

7.3.1. Reducing *FORK* to *STCON*

We can relate this game to the previous (s-connectivity) game as follows. Look at $[n]^l$ as an $n \times l$ grid whose nodes are arranged into levels $V_1, \ldots, V_l$, each with n nodes. Add two new levels $V_0 = \{s\}$ and $V_{l+1} = \{t\}$. and each other level consists of n vertices. We restrict our attention to graphs on $m = n \cdot l$ vertices, where only edges, connecting nodes from adjacent levels, are allowed.

Lemma 7.9. $C(FORK_{n,l}) \leq C(STCON_{n \cdot l})$.

Proof. Suppose we have a protocol Π for the Karchmer–Wigderson game corresponding to $STCON_{n \cdot l}$. We will show that this protocol can be used for the game $FORK_{n,l}$. To use the protocol Π, the players must convert their inputs $a = (a_1, \ldots, a_l)$ and $b = (b_1, \ldots, b_l)$ (for the fork game) to inputs for the st-connectivity game.

The players convert their inputs in the following way. Alice converts her input $(a_1, \ldots, a_l)$ into a path $p = (u_0, u_1, \ldots, u_l, u_{l+1})$ where $u_0 = s$, $u_{l+1} = t$, and $u_i = a_i$ for $1 \leq i \leq l$. Bob converts his input $(b_1, \ldots, b_l)$ into a coloring c by assigning color 0 to all vertices $s, b_1, \ldots, b_l$, and assigning color 1 to the remaining vertices; hence, $c(s) = 0$ and $c(t) = 1$ (because $t \notin \{s, b_1, \ldots, b_l\}$).

The players now can use the protocol Π to get an edge $(u_{i-1}, u_i) \in p$ such that $c(u_{i-1}) = 0$ and $c(u_i) = 1$. This means that u_{i-1} is in $(s, b_1, \ldots, b_l)$ and u_i is not. We claim that i is a valid answer for the fork game on the pair a, b. Indeed, if $i = 1$ then $u_{i-1} = s$, and hence, $a_1 \neq b_1$. If $1 < i \leq l$ then $u_{i-1} = a_{i-1}$ and $u_i = a_i$; hence, $a_{i-1} = b_{i-1}$ and $a_i \neq b_i$. Finally, if $i = l + 1$ then $a_l = b_l = u_l$, and l is a legal answer for the fork game. □

7.4. The lower bound

By Lemma 7.9 and Exercise 7.8 we know that $C(FORK_{n,n}) = O((\log n)^2)$. We will show that this trivial upper bound is almost optimal.

Theorem 7.10 (Grigni–Sipser 1991). *Any communication protocol for a string game on the set $[n]^l$ requires at least $\Omega((\log n) \cdot (\log l))$ bits of communication. That is,*

$$C(FORK_{n,l}) = \Omega((\log n) \cdot (\log l)).$$

Proof. Call a two-player protocol an (α, l)-*protocol* if there is some subset $S \subseteq [n]^l$ such that $|S| \geq \alpha n^l$ and for all $a, b \in S$ the protocol finds a fork position of a and b or outputs l if $a_l = b_l$. Denote by $C(\alpha, l)$ the minimum communication complexity of an (α, l)-protocol. That is, if $C(\alpha, l) = d$ then there exists a subset $S \subseteq [n]^l$ of $|S| \geq \alpha n^l$ strings and a protocol Π of communication complexity d such that Π works correctly on S. In particular, $C(1, n) = C(FORK_{n,l})$.

We start with two simple claims.

Claim 7.11. *For any $l \geq 1$ and any $\alpha > 1/n$, $C(\alpha, l) > 0$.*

Proof. Suppose that $C(\alpha, l) = 0$. Thus, there exists a subset of strings $S \subseteq [n]^l$ such that $|S| \geq \alpha n^l > n^{l-1} \geq 1$ such that the players know the answer $i \in \{0, 1, \ldots, l\}$ without any communication. Since $|S| \geq 2$, the set S contains at least two different strings $a \neq b$. But on input (a, a) the answer must be $i = l$, whereas on input (a, b) the answer must be some $i < l$, a contradiction. □

Claim 7.12. *For every $l \geq 1$ and α, if $C(\alpha, l) > 0$ then $C(\alpha, l) \geq C(\alpha/2, l) + 1$.*

Proof. Let $d = C(\alpha, l)$. Thus, there exists a subset $S \subseteq \{1, \ldots, n\}^l$ such that $|S| \geq \alpha n^l$ and there is a protocol Π such that for all $a, b \in S$, the protocol correctly solves the game on these inputs. Assume w.l.o.g. that Alice speaks first (the case when Bob speaks first is similar). Hence Alice sends either 0 or 1. After this (first) bit is communicated, the set S is splitted into two parts S_0 and S_1. Assume w.l.o.g. that $|S_0| \geq |S_1|$. Let Π_0 be the rest of the protocol Π, after assuming that the first bit send by Alice was 0. That is, Π_0 works exactly like Π, but without sending the first bit, and continuing as if the value of the first bit was 0. The communication complexity of Π_0 is at most $d-1$. Obviously, Π_0 must work correctly on S_0, because Π does this on the whole set $S = S_0 \cup S_1$. Hence, Π_0 is an $(\alpha/2, l)$-protocol. Thus, $C(\alpha/2, l) \leq d - 1 = C(\alpha, l)$. □

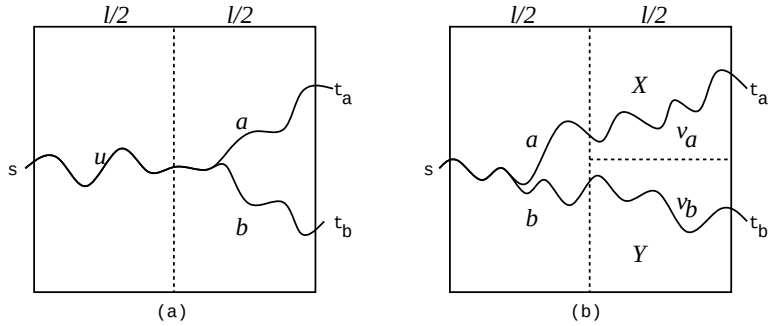

Figure 1. Two cases for constructing a protocol for strings a and b of length $l/2$.

Applying Claim 8.7 $t = (\log n)/2$ times, we obtain that $C(1,n) \geq C(\alpha,n) + t$ with $\alpha = 1/\sqrt{n}$. Since $\alpha > 1/n$, Claim 8.6 yields $C(1,n) = \Omega(\log n)$. This lower bound is, however, too weak.

We now turn to an amplification step: given an (α,l)-protocol (with $l \geq 2$ and α not too small), we convert it to a $(\sqrt{\alpha}/2, l/2)$-protocol. Thus α may be amplified greatly[3] while l is cut in half. By amplifying α after every $\Theta(\log n)$ steps of the traversal strategy, we may keep $\alpha > 1/n$ until l reaches 1, showing the protocol has a path of depth at least $\Omega(\log^2 n)$.

Ex. 7.13. Let $S \subseteq [n]^k$ and $|S| \geq \alpha \cdot n^k$. Show that then the projection of S onto some set of $k/2$ coordinates contains at least $\sqrt{\alpha} \cdot n^{k/2}$ strings.

Lemma 7.14. (Amplification Lemma) *For every $l \geq 2$ and $\alpha \geq 1/\sqrt{n}$,*

$$C(\alpha,l) \geq C(\frac{\sqrt{\alpha}}{2}, \frac{l}{2}).$$

Proof. We are given an (α,l)-protocol working correctly on some set $S \subseteq [n]^l$ of $|S| \geq \alpha n^l$ strings. Consider a bipartite graph G with parts U and V where U consists of all $n^{l/2}$ possible strings on the first $l/2$ levels, and V consists of all $n^{l/2}$ possible strings on the last $l/2$ levels. We connect u and v if their concatenation uv is a string in S; we say that v is an *extension* of u. Hence, G is a bipartite graph with parts of size $n^{l/2}$ and $|S| \geq \alpha n^l$ edges.

By Lemma 2.7 from the book (re-formulated in terms of bipartite graphs), at least one of the following two must hold:

(a) Some $u \in U$ has at least $\sqrt{\frac{\alpha}{2}}\,|V|$ extensions.

(b) There is an $U' \subseteq U$ such that $|U'| \geq \sqrt{\frac{\alpha}{2}}\,|U|$ and each $u \in U'$ has at least $\frac{\alpha}{2}\,|V|$ extensions.

In both cases (a) and (b), we show how to construct a $(\sqrt{\alpha}/2, l/2)$-protocol (see Fig. 1).

Case (a) In this case, we have one string u on the left that has many extensions v on the right such that $uv \in S$. Thus we can recover a $(\sqrt{\alpha/2}, l/2)$-protocol as follows: let S' be the set of extensions of u. Given two strings $v_a, v_b \in S'$, the two players can play the S'-game on these inputs by following the S-protocol for the

[3]Note that $\sqrt{\alpha}/2$ is at least twice larger than α, if $\alpha \leq 1/16$.

strings uv_a, uv_b. Since these strings are identical on the first $l/2$ coordinates, the answer i must correspond to a point where (the paths corresponding to) v_a and v_b diverge. [4]

Case (b) In this case, we take a random partition of the $nl/2$ nodes in the right $l/2$ levels. More precisely, take $n/2$ nodes at random from each of the right $l/2$ levels, and call their union X; call the set of remaining $nl/4$ right nodes Y.

Claim 7.15. *Given case (b) holds and we take a string $u \in U'$, then with probability at least $1 - 2e^{-\alpha n/4}$ there is an extension $v_X(u)$ of u entirely in set X and another extension $v_Y(u)$ of u entirely in set Y.*

Proof of Claim 7.15. We prove that such an extension $v_X(u)$ exists with probability $1 - e^{-\alpha n/4}$, and then add failure probabilities. We know that u has a set $S \subseteq [n]^{l/2}$ of $|S| \geq \frac{\alpha}{2} |V| = \frac{\alpha}{2} n^{l/2}$ extensions. Since the probability that a string $v \in S$ lies entirely in the set X is at least[5] $2^{-l/2}$, the probability that *none* of the strings from S lies entirely in X does not exceed

$$(1 - 2^{-l/2})^{|S|} < e^{-|S|/2^{l/2}} \leq e^{-(\alpha/2)\,(n/2)^{l/2}} \leq e^{-\alpha n/4}$$

Hence, the probability that X contains no extension of u does not exceed $e^{-\alpha n/4}$. Since the set of points Y has the same distribution as X, the same argument yields that Y contains no extension of u with probability at most $e^{-\alpha n/4}$. Adding these failure probabilities we obtain that, with probability at least $1 - 2e^{-\alpha n/4}$ there is an extension of u entirely in set X *and* another extension of u entirely in set Y. □

Now if we have $\alpha \gg 1/n$, then $1 - o(1)$ of strings $u \in U'$ will have extensions in both X and Y.

> **Ex. 7.16.** Prove this. *Hint*: Say that $u \in U'$ is *good* if u has extensions in both X and Y. Use Markov's inequality to show that with probability at least $1/2$, at least $|U'|\,(1 - 4e^{-\alpha n/4})$ nodes of U' are good.

In particular if we always keep $\alpha \geq n^{-1/2}$, then (for n large enough) there exists some choice of X and Y such that[6] $\sqrt{\alpha}/2$ of all possible left-half strings have extensions in both X and Y. This set of extendable strings is our new S'.

This yields a $(\sqrt{\alpha}/2, l/2)$-protocol as follows. Given strings $u_a, u_b \in S'$, the players follow the S-protocol on the inputs $u_a v_X(u_a), u_b v_Y(u_b)$. Since the S-protocol is correct on these strings, and since they share no vertices in the right $l/2$ levels, the protocol must return an answer i in the first $l/2$ levels, hence the answer is in fact valid for u_a and u_b.

This completes the proof of Lemma 7.14. □

Now we can finish the proof of Theorem 7.10 as follows.

By $t = O(\log n)$ applications of Claim 8.7 and one application of the Amplification Lemma (Lemma 7.14), we obtain that

$$C(\frac{2}{\sqrt{n}}, l) \geq C(\frac{16}{n}, n) + \Omega(\log n) \geq C(\frac{2}{\sqrt{n}}, l) + \Omega(\log n).$$

---

[4] Strictly speaking, we must consider strings v_a, v_b as elements of $[n]^{l/2}$. Hence, if i is the answer of the S-protocol on uv_a, uv_b, then the answer of the S'-protocol is $i - l/2$.

[5] Because each v_i belongs to X with probability $1/2$, and this probability is the same for each i.

[6] Since $\frac{1}{2} \leq \frac{1}{\sqrt{2}}(1 - o(1))$ (we are quite generous here).

Applying the last inequality $O(\log l)$ times and using Claim 8.6, we obtain

$$C(\frac{2}{\sqrt{n}}, l) \geq C(\frac{2}{\sqrt{n}},\ 1) + \Omega(\log n) \cdot \Omega(\log l).$$

Hence,

$$C(FORK_{n,l}) = C(1, n) \geq C(\frac{2}{\sqrt{n}}, n) = \Omega((\log n)^2).$$

□

TOPIC 8
More on Rank Arguments

(Related sections: 15.2, 16)

Summary: A simple rank lower bounds argument for monotone formulas and monotone span programs. Proof that no self-avoiding family of subsets of an n-element set can have more than $n^{O(\log n)}$ members.

We already know (see Theorem 16.8 in the book) that if the family $\mathcal{A}$ of minterms of a monotone boolean function f is self-avoiding, then f cannot be computed by a monotone span program of size $|\mathcal{A}| - 1$. In this lecture we will discuss a recent observation, made by Pudlák, that this (and even more general) result can be obtained by a direct and very simple rank argument.

So far, it was not known whether self-avoiding families can have exponentially many members (see Problem 1 on p. 216 of the book). Combining Pudlák's argument with some known results in communication complexity, Gál has recently shown that no self-avoiding family of subsets of an n-element set can have more than $n^{O(\log n)}$ members.

8.1. Locally intersecting families are hard to separate

Let $f(x_1, \ldots, x_n)$ be a monotone boolean function. Recall that a *minterm* (resp. *maxterm*) of f is a minimal subset $A \subseteq \{1, \ldots, n\}$ such that the constant 1 (resp. constant 0) to all variables x_i with $i \in A$ forces the function to output 1 (resp. 0), independent of the values of the remaining variables. Hence, if $\mathcal{A}$ is a subset of minterms and $\mathcal{B}$ a subset of maxterms of f, then the pair $\mathcal{A}, \mathcal{B}$ is *cross-intersecting*, that is $A \cap B \neq \emptyset$ for all $A \in \mathcal{A}$ and $B \in \mathcal{B}$.

Say that the pair $\mathcal{A}, \mathcal{B}$ is *locally intersecting* if we can divide each set $B \in \mathcal{B}$ into two parts B_0 and B_1 so that every $A \in \mathcal{A}$ has a nonempty intersection with exactly one of these parts.

Given a locally intersecting pair $\mathcal{A}, \mathcal{B}$, define its *disjointness matrix* to be an $|\mathcal{A}|$ by $|\mathcal{B}|$ matrix R, with its rows indexed by sets $A \in \mathcal{A}$ and its columns indexed by sets $B \in \mathcal{B}$, such that the entry $R_{A,B}$ is defined by

$$R_{A,B} = \begin{cases} 0 & \text{if } A \cap B_0 \neq \emptyset, \\ 1 & \text{if } A \cap B_1 \neq \emptyset. \end{cases}$$

Recall that a monotone span program (over some field $\mathbb{F}$) is given by a matrix M over $\mathbb{F}$ with its rows labeled by variables $x_1, \ldots, x_n$. Such a matrix *accepts* a subset $A \subseteq \{1, \ldots, n\}$ if and only if the rows of M, labeled by variables x_i with $i \in A$, span the all-1 vector (or any other fixed in advance vector). A span program *separates* $\mathcal{A}, \mathcal{B}$ if it accepts all members of $\mathcal{A}$ and rejects the complements $\overline{B}$ of all members $B \in \mathcal{B}$.

Theorem 8.1 (Pudlák 2002). *If the pair $\mathcal{A}, \mathcal{B}$ is locally intersecting, then any monotone span program over $\mathbb{F}$ separating this pair has size at least* $\mathrm{rank}_{\mathbb{F}}(R)$.

Proof. Let M be a monotone span program separating $\mathcal{A}, \mathcal{B}$. Let d be the number of rows in M. The idea is to show that the intersection matrix R of $\mathcal{A}, \mathcal{B}$ is a matrix of scalar products of vectors of dimension at most d; this yields $\mathrm{rank}_{\mathbb{F}}(R) \leq d$ (see Exercise 8.2).

For every $A \in \mathcal{A}$, let $v_A \in \mathbb{F}^d$ be a vector witnessing the fact that A must be accepted, which means that $v_A \cdot M = v_0$, where v_0 is the all-1 vector and v_A is a vector which is nonzero only in coordinates corresponding to elements of A. Let $B \in \mathcal{B}$. Since $\overline{B}$ cannot be accepted, no linear combination of the rows of $M_{\overline{B}}$ can give v_0. Hence (see Exercise 8.3) for each $B \in \mathcal{B}$ there is a vector u_B such that $\langle v_0, u_B \rangle = 1$ and $M_{\overline{B}} \cdot u_B = 0$ (the all-0 vector). Let $w_B \in \mathbb{F}^d$ be the vector obtained from the vector $M \cdot u_B$ by replacing to 0 all its elements, corresponding to the rows labeled by elements of B_0; note that $w_B(i) \neq 0$ only if $i \in B_1$: the elements $w_B(i)$ with $i \in \overline{B}$ are zero because $M_{\overline{B}} \cdot u_B = 0$, and the elements $w_B(i)$ with $i \in B_0$ are zero by the definition of w_B. We claim that

$$R_{A,B} = \langle v_A, w_B \rangle.$$

Indeed, if $A \cap B_0 \neq \emptyset$, then $A \cap B_1 = \emptyset$, and hence, the vectors v_A and w_B have no element on which they both are nonzero; hence, in this case $\langle v_A, w_B \rangle = 0$. If $A \cap B_1 \neq \emptyset$, then $A \cap B_0 = \emptyset$, and hence, in this case, $\langle v_A, w_B \rangle = \langle v_A, M \cdot u_B \rangle$, implying that

$$\langle v_A, w_B \rangle = \langle v_A, M \cdot u_B \rangle = \langle v_A \cdot M, u_B \rangle = \langle v_0, u_B \rangle = 1.$$

This shows that R is a matrix of scalar products of vectors of dimension d, implying that $\mathrm{rank}_{\mathbb{F}}(R) \leq d$. □

Ex. 8.2. Let $M : X \times Y \to \mathbb{F}$ be an $n \times m$ matrix with rows labeled by elements of X and columns labeled by elements of Y. Suppose that there exists a sequence of functions $a(x) = (a_1(x), \ldots, a_d(x))$ from X to $\mathbb{F}$ and a sequence of functions $b(y) = (b_1(y), \ldots, b_d(y))$ from Y to $\mathbb{F}$ such that

$$M(x,y) = \langle a(x), b(y) \rangle = \sum_{i=1}^{d} a_i(x) b_i(y).$$

Show that then $\mathrm{rank}_{\mathbb{F}}(M) \leq d$. *Hint*: Let M_i be the matrix defined by $M_i(x,y) := a_i(x)b_i(y)$. Note that the x-th row of M_i is $a_i(x)$ times one and the same vector r whose y-th entry is $b_i(y)$. Use the sub-additivity of rank (see (14.1) in the book)

Ex. 8.3. (Extension of Ex. 16.1 from the book to arbitrary fields.) Let $A \subseteq \mathbb{F}^n$ and 1^n be the all-1 vector in $\mathbb{F}^n$. Show that $1^n \in \mathrm{span}(A) \iff A \cdot u = 0$ for some $u \in \mathbb{F}^n$ such that $\langle 1^n, u \rangle = 1$. *Hint*: Let $E := \{u \in \mathbb{F}^n : \langle 1^n, u \rangle \neq 1\}$ be the set of all "even" vectors. Then $1^n \in \mathrm{span}(A)$ iff $\mathrm{span}(A)^{\perp} \subseteq E$.

Since, as shown in the book (see Sect. 16.2), monotone span programs are not weaker than switching networks (and hence, are not weaker than monotone formulas) Theorem 8.1 directly yields the following

Theorem 8.4. *If the pair $\mathcal{A}, \mathcal{B}$ is locally intersecting, then any monotone formula separating this pair has size at least* $\mathrm{rank}_{\mathbb{F}}(R)$.

8.2. Self-avoiding families

Let $\mathcal{A}$ be a family of minterms of some monotone boolean function $f(X)$. For a subset $Y \subseteq X$, let $S(Y)$ be the union of all members of $\mathcal{A}$ intersecting Y. Recall that a family $\mathcal{A}$ is *self-avoiding* if one can associate a subset $T_A \subseteq A$ with each $A \in \mathcal{A}$, called the *core* of A, such that

1. T_A determines A in the family. That is, no other set in the family $\mathcal{A}$ contains T_A as a subset.
2. For every $A \in \mathcal{A}$ and every subset $Y \subseteq T_A$, the set $S(Y) \setminus Y$ does not contain any member of $\mathcal{A}$.

In Sect. 16.4 of the book we have proved the following general lower bounds criterion for monotone span programs.

Theorem 8.5. *Let f be a monotone boolean function and $\mathcal{A}$ be the family of its minterms. If $\mathcal{A}$ is self-avoiding then for every field $\mathbb{F}$, every monotone span program over $\mathbb{F}$ computing f has size at least $|\mathcal{A}|$.*

Here we will show that this theorem is a special case of Theorem 8.1.

Proof. Define $\mathcal{B}$ to be the family of all sets $B = B_0 \cup B_1$ where $B_0 \subseteq T_A$ for some $A \in \mathcal{A}$ and $B_1 = X \setminus S(B_0)$. The theorem follows from Theorem 8.1 and the following two claims.

Claim 8.6. *The pair $\mathcal{A}, \mathcal{B}$ is locally intersecting.*

Proof of Claim 8.6. Take an arbitrary $A' \in \mathcal{A}$ and an arbitrary $B \in \mathcal{B}$; hence, $B = B_0 \cup B_1$ where $B_0 \subseteq T_A$ for some $A \in \mathcal{A}$ and $B_1 = X \setminus S(B_0)$.

If $A' \cap B_0 \neq \emptyset$, then $A' \subseteq S(B_0)$, and hence, $A' \cap B_1 = \emptyset$. Thus, A' intersects at most one of the sets B_0 and B_1.

If $A' \cap B_0 = \emptyset$, then $A' \cap B_1 \neq \emptyset$, because otherwise we would have that $A' \subseteq S(B_0)$ and, since $A' \cap B_0 = \emptyset$, also that $A' \subseteq S(B_0) \setminus B_0$. Thus, A' intersects exactly one of the sets B_0 and B_1, implying that the pair $\mathcal{A}, \mathcal{B}$ is locally intersecting. □

Claim 8.7. *The disjointness matrix R of the pair $\mathcal{A}, \mathcal{B}$ has full row-rank over $\mathbb{F}$.*

Proof of Claim 8.7. Recall that the rows of R are labeled by the members A of $\mathcal{A}$, the columns are labeled by the members B of $\mathcal{B}$; moreover, each B consists of two parts B_0 and B_1 where $B_0 \subseteq T_{A'}$ for some $A' \in \mathcal{A}$ and $B_1 = X \setminus S(B_0)$. The entry $R_{A,B}$ is 0 if $A \cap B_0 \neq \emptyset$, and is 1 otherwise.

Fix an $A \in \mathcal{A}$ and let M be the submatrix of R consisting of all columns indexed by subsets Y of T_A. Since, by our assumption, *every* subset of T_A appears as a column of M, the rows of M are rows[1] of the full disjointness matrix[2] D_m with $m = |T_A|$, some of them repeated. Observe that the row indexed by A occurs in M only once. This is because this row has 0's in all columns indexed by one element subsets $\{x\}$ with $x \in T_A$. Hence, if for some $A' \in \mathcal{A}$, the row of M indexed by A' would be the same then we would have that $T_A \subseteq A'$ and the first property of self-avoiding families would imply that $A' = A$.

We know that full disjointness matrix D_m has full row-rank (see Exercise 16.3 in the book). Since the row indexed by A occurs in M only once, this implies that

[1] That is, each row of M is a row of D_m, although some rows of D_m may not appear in M.

[2] Recall that D_m is a 0-1 matrix whose rows and columns are indexed by subsets of $\{1, \ldots, m\}$, and the entry corresponding to A and B is 1 precisely when $A \cap B = \emptyset$.

this row cannot be a linear combination of others. Thus, the corresponding row of the entire matrix R cannot be a linear combination of others, as well. □

Claims 8.6 and 8.7 also imply the following

Lemma 8.8. *If $\mathcal{A}$ is self-avoiding then there is a family $\mathcal{B}$ such that the pair $\mathcal{A}, \mathcal{B}$ is locally intersecting and the disjointness matrix R of this pair has rank $|\mathcal{A}|$ (over any field).*

8.3. The limits of the approach

The highest lower bound on the size of monotone span programs, obtained by using Theorem 8.5 have had the form $n^{\Omega(\log n)}$ (see Sect. 16.5 in the book). A natural question was whether one can prove larger lower bounds when using the more general Theorem 8.1.

Unfortunately, as observed by Anna Gál, Theorem 8.1 (and hence, Theorem 8.5) *cannot* yield lower bounds larger than $n^{O(\log n)}$. This observation also implies that every self-avoiding family can have at most so many members, thus answering the first of four open problems formulated at the end of Sect. 16.5 of the book.

Proposition 8.9 (Gál 2002). *Let $\mathcal{A}, \mathcal{B}$ be a locally intersecting pair of families, and R be its disjointness matrix over a field $\mathbb{F}$. Then*

$$\mathrm{rank}_{\mathbb{F}}(R) \leq n^{O(\log n)}.$$

Proof. Let $\delta(R)$ be the minimum number of mutually disjoint monochromatic submatrices of R whose union covers all entries of R. Hence, $\mathrm{rank}_{\mathbb{F}}(R) \leq \delta(R)$. On the other hand, Aho, Ullman and Yannakakis (1983) has proved that, for every matrix R, $\log \delta(R) \leq O((\log \alpha(R))^2$ where $\alpha(R)$ is the minimum number of monochromatic (but not necessarily disjoint!) submatrices whose union covers all entries of R. (This is an interesting result in itself, and we will prove it in the next section.)

Each entry $R_{A,B}$ of R is either 0 or 1 depending on whether A intersects B in its first block B_0 or in the second one B_1. Hence, all entries of R can be covered by $2n$ submatrices $M_i^{\varepsilon} = \{(A,B) : i \in A \cap B_{\varepsilon}\}$, $i = 1, \ldots, n$ and $\varepsilon = 0, 1$. It is clear that these submatrices cover all entries of R. Moreover, since the pair $\mathcal{A}, \mathcal{B}$ is locally intersecting, we have that all the entries of M_i^{ε} are equal to ε. Hence, we have found a covering of R by $2n$ monochromatic submatrices M_i^{ε}, implying that $\alpha(R) \leq 2n$. By the above mentioned result of Aho et al, $\log \mathrm{rank}_{\mathbb{F}}(R) \leq \log \delta(R) \leq O((\log \alpha(R))^2 \leq O((\log n)^2$. □

The proposition above, together with Lemma 8.8, yield the following

Corollary 8.10. *Let $\mathcal{A}$ be a family of subsets of some n-element set. If $\mathcal{A}$ is self-avoiding then $|\mathcal{A}| \leq n^{O(\log n)}$.*

8.4. Disjoint vs. non-disjoint covers of matrices

Let M be an n by n 0-1 matrix. For $\varepsilon \in \{0,1\}$, let $\alpha_{\varepsilon}(M)$ denote the minimum number of all-ε submatrices of M whose union covers all entries ε of M. Let $\delta(M)$ be the minimum number of *mutually disjoint* monochromatic submatrices of M whose union covers all entries of M.

Theorem 8.11 (Aho–Ullman–Yannakakis 1983). *Let M be a 0-1 matrix with $\alpha_0(M) = r$ and $\alpha_1(M) = s$. Then $\delta(M) \leq r^{O(\log s)}$.*

Proof. We can look at the matrix M as a function $M : X \times Y \to \{0,1\}$ and consider the communication game in which Alice and Bob are trying to compute this function (see lecture "Communication Games"). Let $C(M)$ be the deterministic communication complexity of this game. It is easy to show (do this!) that $\delta(M) \leq 2^{C(M)}$.

Let $M_1, \ldots, M_r$ be the $r = \alpha_0(M)$ all-0 submatrices of M covering all 0's of M. For every matrix M_i, consider the matrix A_i formed by the rows of M intersecting M_i, and B_i be the matrix formed by the columns of M intersecting M_i. Since M_i consists only of 0's, we have that

$$\alpha_1(A_i) + \alpha_1(B_i) \leq \alpha_1(M) \qquad (*)$$

The protocol consists of $\log s = \log \alpha_1(M)$ rounds. In each round, the players do the following. First, Alice checks whether there is an index i such that her row intersects M_i and $\alpha_1(A_i) \leq \frac{1}{2}\alpha_1(M)$. If yes, then (using $1 + \log r$ bits) she sends "1" and the index i to Bob. If not, then she sends "0". Now Bob checks whether there is an index i such that his column intersects M_i and $\alpha_1(B_i) \leq \frac{1}{2}\alpha_1(M)$. If yes, then (using $1 + \log r$ bits) he sends "1" and the index i to Bob. If not, then she sends "0".

If either Alice or Bob find a suitable index i in this round then, by communicating at most $2 + \log r$ bits, they have restricted the problem to a matrix M' ($= A_i$ or B_i) for which $\alpha_1(M') \leq \frac{1}{2}\alpha_1(M)$. Hence, in this case, the theorem follows by induction.

If both players have sent "0" in this round, then they can finish the protocol: the answer is "1". Indeed, if there would be a 0 in the intersection of Alice's row and Bob's column, then this 0 would belong to some submatrix M_i. However, for these submatrices we have on the one hand $\alpha_1(A_i) > \frac{1}{2}\alpha_1(M)$ (since they did not suit Alice), on the other hand $\alpha_1(B_i) > \frac{1}{2}\alpha_1(M)$ since they did not suit Bob. But this contradicts (*).

Thus, we have shown that $C(M) \leq (2 + \log r) \cdot \log s = O((\log r) \cdot (\log s))$. Since $\delta(M) \leq 2^{C(M)}$, the desired upper bound $\delta(M) \leq r^{O(\log s)}$ follows. □

Remark. Note that the measure $\alpha_1(M)$ in this theorem can be replaced by any measure $\mu(M)$ satisfying (*).

> **Ex. 8.12.** Let $\mathcal{A}, \mathcal{B}$ be a locally intersecting pair of families. Suppose that each member of $\mathcal{A}$ has at most a elements, and each member of $\mathcal{B}$ has at most b elements. Show that then the disjointness matrix R of this pair contains an $N \times M$ submatrix of rank 1 with $N \geq |\mathcal{A}|/2b$ and $M \geq |\mathcal{B}|/2a$.
> *Hint*: Use Theorem 18.6 from the book.

BIBLIOGRAPHY

1. A. Aho, J. Ullman and M. Yannakakis (1983): On notions of information transfer in VLSI circuits. In: *Proc. of 15th ACM STOC*, pp. 133–139.
2. M. Ajtai (1998): Determinism versus non-determinism for linear time RAMs with memory restrictions, in: Proc. of 31-th STOC '98, 632–641. Available at ECCC http://www.eccc.uni-trier.de/eccc
3. ——— (1999): A non-linear time lower bound for boolean branching programs, in: Proc. of 40th FOCS '99, 60–70. Available at ECCC http://www.eccc.uni-trier.de/eccc
4. M. Ajtai and Y. Gurevich (1987): Monotone versus positive, *J. of the ACM* **34**, 1004–1015.
5. A.E. Andreev (1987), On a method for obtaining more than quadratic effective lower bounds for the complexity of π-schemes, *Vestnik Moskovskogo Universiteta, Matematika*, **42**:1, 70–73 (in Russian). English translation in *Moscow University Mathematics Bulletin* **42**:1,63–66.
6. P.W. Beame, M. Saks, and J.S. Thathachar (1998) Time-space tradeoffs for branching programs, in: Proc. of 39th FOCS, 254–263. Available at ECCC http://www.eccc.uni-trier.de/eccc
7. S.J. Berkowitz (1982): On some relationships between monotone and non-monotone circuit complexity. Technical Report, University of Toronto.
8. A. Borodin, A. Razborov and R. Smolensky (1993), On lower bounds for read-k times branching programs, *Computational Complexity* **3**, 1-18.
9. S. Buss and T. Pitassi (1996): Resolutions and the weak pigeonhole principle. Manuscript. Available at http://www.cs.arizona.edu/people/toni/
10. A. Ehrenfeucht and D. Haussler (1989): Learning decision trees from random examples. *Information and Computation* **82**, 231–246.
11. M.J. Fischer (1974): The complexity of negation-limited networks–a brief survey, in: Springer Lect. Notes in Comput. Sci., vol. 33, 71–82.
12. ——— (1996): Lectures on network complexity. Technical Report TR-1104, Department of Computer Science, Yale University.
13. A. Gál (2002): Answer to the open problem of ECCC TR02-007. *Electronic Colloquium on Computational Complexity*, Comment 1 on Report No. 007. Available at: http://www.eccc.uni-trier.de/eccc
14. M. Grigni and M. Sipser (1991): Monotone separation of Logspace from NC^1, *FOCS*, 294–298.
15. S. Jukna (1999): Combinatorics of monotone computations, *Combinatorica*, **19**:1 (1999), 1–21.
16. ——— (2001): A note on the minimum number of negations leading to superpolynomial savings, *ECCC Report Nr. 58*. Also available at: http://www.thi.informatik.uni-frankfurt.de/~jukna
17. ——— (2002): A space-time trade-off for nondeterministic branching programs. Available at http://www.thi.informatik.uni-frankfurt.de/~jukna

18. S. Jukna, A. Razborov, P. Savický and I. Wegener (1999): On P versus NP∩co-NP for decision trees and read-once branching programs, *Computational Complexity* **8**:4, 357–370.
19. M. Karchmer and A. Wigderson (1988): Monotone circuits for connectivity require super-logarithmic depth, in: *Proc. of 20th Ann. ACM Symp. on the Theory of Computing*, 539–550. Journal version: *SIAM J. on Discrete Math.*, **3** (1990), 255-265.
20. A.A. Markov (1957): On the inversion complexity of systems of Boolean functions, *Doklady Academii Nauk SSSR*, **116**, 917–919 (in Russian). English translation in: *J. of ACM*, **5**:4 (1958), 331–334, and in: *Soviet Math. Doklady* **4** (1963), 694–696.
21. E.I. Nechiporuk (1966), On a Boolean function, *Doklady Akademii Nauk SSSR*, **169**:4, 765–766 (in Russian). English translation in: *Soviet Mathematics Doklady* **7**:4, 999–1000.
22. E.A. Okolnishnikova (1982): On the influence of negation on the complexity of realization of monotone Boolean functions by formulas of bounded depth, in: *Metody Diskretnogo Analiza* **38**, 74–80 (in Russian)
23. P. Pudlák (2002): Monotone complexity and the rank of matrices, *Electronic Colloquium on Computational Complexity*, Report No. 7.
24. R. Raz (2001): Resolution lower bounds for the weak pigeonhole principle. ECCC Report Nr. 21. Available at http://www.eccc.uni-trier.de/eccc
25. R. Raz and A. Wigderson (1989): Probabilistic communication complexity of Boolean relations, in: Proc. of 30th Ann. IEEE Symp. on Foundations of Comput. Sci., 562–567.
26. A.A. Razborov (1985): A lower bound on the monotone network complexity of the logical permanent, *Matematicheskie Zametki*, **37**:6, 887–990 (in Russian). English translation in: Math. Notes Acad. of Sci. USSR, **37**:6 (1985), 485–493.
27. _____ (2001): Improved resolution lower bounds for the weak pigeonhole principle. ECCC Report Nr. 55. Available at http://www.eccc.uni-trier.de/eccc and at http://genesis.mi.ras.ru/ razborov/
28. A. Rosenbloom (1997): Monotone circuits are more powerful than monotone boolean circuits, *Information Processing Letters* **61**:3, pp. 161–164.
29. M. Santha and Ch. Wilson (1993): Limiting negations in constant depth circuits, *SIAM J. Comput.* **22**:2, 294–302.
30. P.M. Spira (1971), On time–hardware complexity tradeoffs for Boolean functions, *Proceedings of 4th Hawaii Symposium on System Sciences*, Western Periodicals Company, North Hollywood, 525–527.
31. B.A. Subbotovskaja (1961), Realizations of linear functions by formulas using $+, ., -$, *Doklady Akademii Nauk SSSR*, **136**:3, pp. 553–555 (in Russian). English translation in: *Soviet Mathematics Doklady* **2**, 110–112.
32. É. Tardos (1987): The gap between monotone and non-monotone circuit complexity is exponential, *Combinatorica*, **7**:4, 141–142.
33. L.G. Valiant (1986): Negation is powerless for Boolean slice functions, *SIAM J. Comput.* **15**, 531–535.
34. U. Zwick (1991), An extension of Khrapchenko's theorem, *Information Processing Letters*, **37**, 215–217.