Lower Bounds on Circuit Complexity

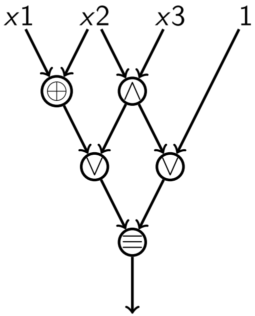
A. Kulikov

Steklov Institute of Mathematics at St. Petersburg

Estonian Theory Days
02 October 2009

Boolean Circuits

- inputs: propositional variables $x_1, x_2, \ldots, x_n$ and constants 0, 1
- gates: binary functions
- fan-out of a gate is unbounded



Random Functions are Complex

Random Functions are Complex

- Shannon counting argument: count how many different Boolean functions in n variables can be computed by circuits with t gates and compare this number with the total number 2^{2^n} of all Boolean functions.

Random Functions are Complex

- Shannon counting argument: count how many different Boolean functions in n variables can be computed by circuits with t gates and compare this number with the total number 2^{2^n} of all Boolean functions.
- The number $F(n,t)$ of circuits of size $\leq t$ with n input variables does not exceed
$$\left(16(t+n+2)^2\right)^t .$$
Each of t gates is assigned one of 16 possible binary Boolean functions that acts on two previous nodes, and each previous node can be either a previous gate ($\leq t$ choices) or a variables or a constant ($\leq n+2$ choices).

Random Functions are Complex

- Shannon counting argument: count how many different Boolean functions in n variables can be computed by circuits with t gates and compare this number with the total number 2^{2^n} of all Boolean functions.
- The number $F(n, t)$ of circuits of size $\leq t$ with n input variables does not exceed
$$\left(16(t+n+2)^2\right)^t .$$
Each of t gates is assigned one of 16 possible binary Boolean functions that acts on two previous nodes, and each previous node can be either a previous gate ($\leq t$ choices) or a variables or a constant ($\leq n+2$ choices).
- For $t = 2^n/(10n)$, $F(n, t)$ is approximately $2^{2^n/5}$, which is $\ll 2^{2^n}$.

Random Functions are Complex

- Shannon counting argument: count how many different Boolean functions in n variables can be computed by circuits with t gates and compare this number with the total number 2^{2^n} of all Boolean functions.
- The number $F(n,t)$ of circuits of size $\leq t$ with n input variables does not exceed
$$\left(16(t+n+2)^2\right)^t .$$
Each of t gates is assigned one of 16 possible binary Boolean functions that acts on two previous nodes, and each previous node can be either a previous gate ($\leq t$ choices) or a variables or a constant ($\leq n+2$ choices).
- For $t = 2^n/(10n)$, $F(n,t)$ is approximately $2^{2^n/5}$, which is $\ll 2^{2^n}$.
- Thus, the circuit complexity of almost all Boolean functions on n variables is exponential in n. Still, we do not know any explicit function with super-linear circuit complexity.

Known Lower Bounds

	circuit size	formula size
full binary basis B_2	$3n - o(n)$ [Blum]	$n^{2-o(1)}$ [Nechiporuk]
basis $U_2 = B_2 \setminus \{\oplus, \equiv\}$	$5n - o(n)$ [Iwama et al.]	$n^{3-o(1)}$ [Hastad]
monotone basis $M_2 = \{\vee, \wedge\}$	exponential [Razborov; Alon, Boppana; Andreev; Karchmer, Wigderson]	

Explicit Functions

Explicit Functions

- We are interested in explicitly defined Boolean functions of high circuit complexity.

Explicit Functions

- We are interested in explicitly defined Boolean functions of high circuit complexity.
- Not explicitly defined function of high circuit complexity: enumerate all Boolean functions on n variables and take the first with circuit complexity at least $2^n/(10n)$.

Explicit Functions

- We are interested in explicitly defined Boolean functions of high circuit complexity.
- Not explicitly defined function of high circuit complexity: enumerate all Boolean functions on n variables and take the first with circuit complexity at least $2^n/(10n)$.
- To avoid tricks like this one, we say that a function f is explicitly defined if $f^{-1}(1)$ is in NP.

Explicit Functions

- We are interested in explicitly defined Boolean functions of high circuit complexity.
- Not explicitly defined function of high circuit complexity: enumerate all Boolean functions on n variables and take the first with circuit complexity at least $2^n/(10n)$.
- To avoid tricks like this one, we say that a function f is explicitly defined if $f^{-1}(1)$ is in NP.
- Usually, under a Boolean function f we actually understand an infinite sequence $\{f_n \mid n = 1, 2, \dots\}$.

Known Lower Bounds for Circuits over B_2

Known Lower Bounds

$2n - c$	[Schnorr, 74]
$2.5n - o(n)$	[Paul, 77]
$2.5n - c$	[Stockmeyer, 77]
$3n - o(n)$	[Blum, 84]

Known Lower Bounds for Circuits over B_2

Known Lower Bounds

$2n - c$	[Schnorr, 74]
$2.5n - o(n)$	[Paul, 77]
$2.5n - c$	[Stockmeyer, 77]
$3n - o(n)$	[Blum, 84]

This Talk

In this talk, we will present a proof of a $7n/3 - c$ lower bound which is as simple as Schnorr's proof of $2n - c$ lower bound.

Known Lower Bounds for Circuits over B_2

Known Lower Bounds

$2n - c$	[Schnorr, 74]
$2.5n - o(n)$	[Paul, 77]
$2.5n - c$	[Stockmeyer, 77]
$3n - o(n)$	[Blum, 84]

This Talk

In this talk, we will present a proof of a $7n/3 - c$ lower bound which is as simple as Schnorr's proof of $2n - c$ lower bound.

Gate Elimination

All the proofs are based on the so-called gate elimination method. This is essentially the only known method for proving lower bounds on circuit complexity.

Gate Elimination Method

The main idea

Gate Elimination Method

The main idea

- Take an optimal circuit for the function in question.

Gate Elimination Method

The main idea

- Take an optimal circuit for the function in question.
- Setting some variables to constants obtain a subfunction of the same type (in order to proceed by induction) and eliminate several gates.

Gate Elimination Method

The main idea

- Take an optimal circuit for the function in question.
- Setting some variables to constants obtain a subfunction of the same type (in order to proceed by induction) and eliminate several gates.
- A gate is eliminated if it computes a constant or a variable.

Gate Elimination Method

The main idea

- Take an optimal circuit for the function in question.
- Setting some variables to constants obtain a subfunction of the same type (in order to proceed by induction) and eliminate several gates.
- A gate is eliminated if it computes a constant or a variable.
- By repeatedly applying this process, conclude that the original circuit must have had many gates.

Gate Elimination Method

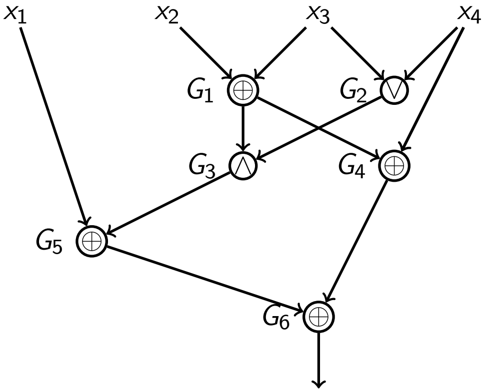
The main idea

- Take an optimal circuit for the function in question.
- Setting some variables to constants obtain a subfunction of the same type (in order to proceed by induction) and eliminate several gates.
- A gate is eliminated if it computes a constant or a variable.
- By repeatedly applying this process, conclude that the original circuit must have had many gates.

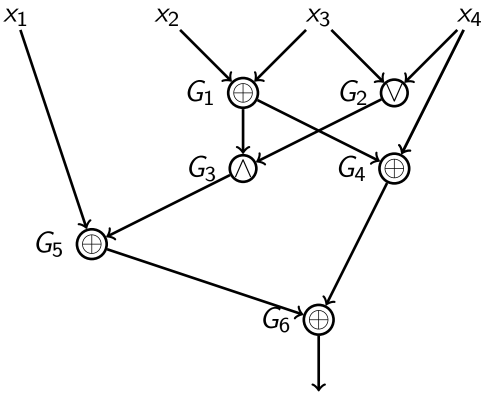
Remark

This method is very unlikely to produce non-linear lower bounds.

Example

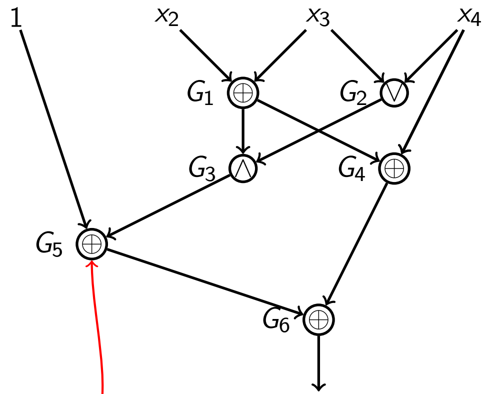


Example



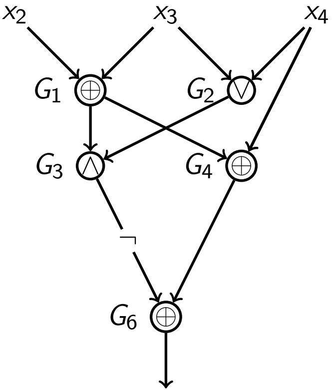
assign $x_1 = 1$

Example

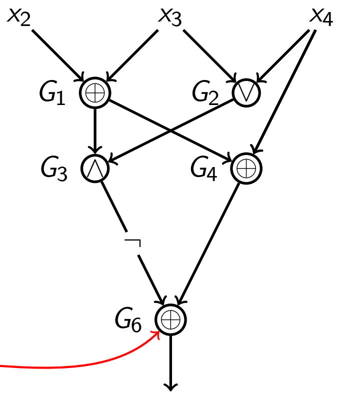


G_5 now computes $G_3 \oplus 1 = \neg G_3$

Example

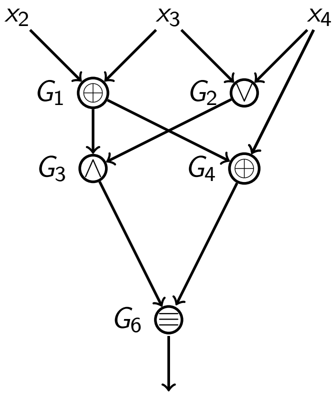


Example

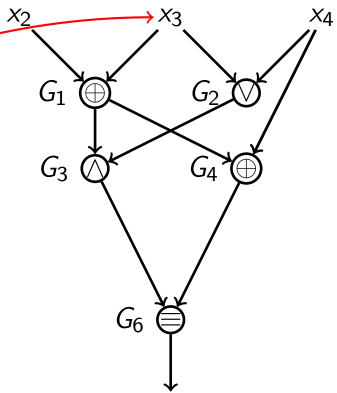


now we can change the binary function assigned to G_6

Example

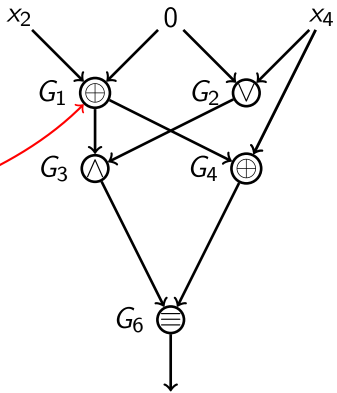


Example



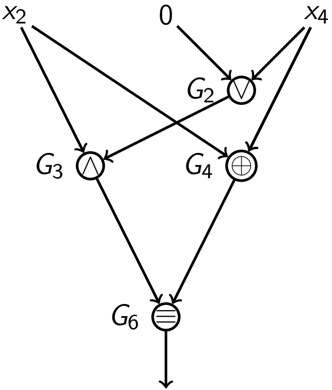
now assign $x_3 = 0$

Example

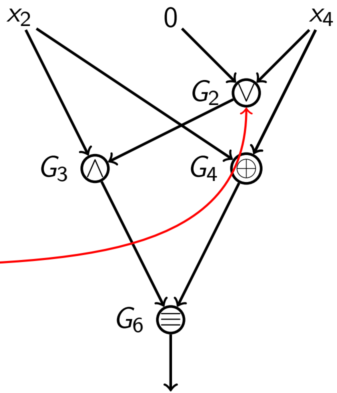


G_1 then is equal to x_2

Example

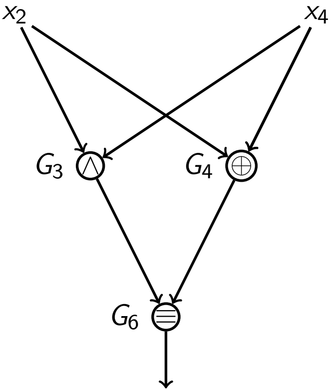


Example



$G_2 = x_4$

Example



The Class $Q_{2,3}^n$

Definition

A function $f\colon \{0,1\}^n \to \{0,1\}$ belongs to the class $Q_{2,3}^n$ if

The Class $Q^n_{2,3}$

Definition

A function $f\colon \{0,1\}^n \to \{0,1\}$ belongs to the class $Q^n_{2,3}$ if

1. for all different $i, j \in \{1, \ldots, n\}$, one obtains at least three different subfunctions by replacing x_i and x_j by constants;

The Class $Q_{2,3}^n$

Definition

A function $f\colon \{0,1\}^n \to \{0,1\}$ belongs to the class $Q_{2,3}^n$ if

1. for all different $i, j \in \{1, \dots, n\}$, one obtains at least three different subfunctions by replacing x_i and x_j by constants;
2. for all $i \in \{1, \dots, n\}$, one obtains a subfunction in $Q_{2,3}^{n-1}$ (if $n \geq 4$) by replacing x_i by any constant.

The Class $Q_{2,3}^n$

Definition

A function $f\colon \{0,1\}^n \to \{0,1\}$ belongs to the class $Q_{2,3}^n$ if

1. for all different $i, j \in \{1, \dots, n\}$, one obtains at least three different subfunctions by replacing x_i and x_j by constants;
2. for all $i \in \{1, \dots, n\}$, one obtains a subfunction in $Q_{2,3}^{n-1}$ (if $n \geq 4$) by replacing x_i by any constant.

Modular functions

The Class $Q_{2,3}^n$

Definition

A function $f\colon \{0,1\}^n \to \{0,1\}$ belongs to the class $Q_{2,3}^n$ if

1. for all different $i, j \in \{1, \ldots, n\}$, one obtains at least three different subfunctions by replacing x_i and x_j by constants;
2. for all $i \in \{1, \ldots, n\}$, one obtains a subfunction in $Q_{2,3}^{n-1}$ (if $n \geq 4$) by replacing x_i by any constant.

Modular functions

- Let $\mathrm{MOD}_{m,r}^n(x_1, \ldots, x_n) = 1$ iff $\sum_{i=1}^n x_i \equiv r \pmod{m}$.

The Class $Q_{2,3}^n$

Definition

A function $f\colon \{0,1\}^n \to \{0,1\}$ belongs to the class $Q_{2,3}^n$ if

1. for all different $i, j \in \{1, \dots, n\}$, one obtains at least three different subfunctions by replacing x_i and x_j by constants;
2. for all $i \in \{1, \dots, n\}$, one obtains a subfunction in $Q_{2,3}^{n-1}$ (if $n \geq 4$) by replacing x_i by any constant.

Modular functions

- Let $\mathrm{MOD}_{m,r}^n(x_1, \dots, x_n) = 1$ iff $\sum_{i=1}^n x_i \equiv r \pmod{m}$.
- Then $\mathrm{MOD}_{3,r}^n, \mathrm{MOD}_{4,r}^n \in Q_{2,3}^n$, but $\mathrm{MOD}_{2,r}^n \notin Q_{2,3}^n$.

Schnorr's $2n$ Lower Bound

Theorem

If $f \in Q^n_{2,3}$, then $C(f) \geq 2n - 8$.

Schnorr's $2n$ Lower Bound

Theorem

If $f \in Q^n_{2,3}$, then $C(f) \geq 2n - 8$.

Proof

Schnorr's $2n$ Lower Bound

Theorem

If $f \in Q^n_{2,3}$, then $C(f) \geq 2n - 8$.

Proof

- Induction on n. If $n \leq 4$, then the statement is trivial.

Schnorr's $2n$ Lower Bound

Theorem

If $f \in Q^n_{2,3}$, then $C(f) \geq 2n - 8$.

Proof

- Induction on n. If $n \leq 4$, then the statement is trivial.
- Consider an optimal circuit and its top gate Q which is fed by different variables x_i and x_j (they are different, since the circuit is optimal).

Schnorr's $2n$ Lower Bound

Theorem

If $f \in Q^n_{2,3}$, then $C(f) \geq 2n - 8$.

Proof

- Induction on n. If $n \leq 4$, then the statement is trivial.
- Consider an optimal circuit and its top gate Q which is fed by different variables x_i and x_j (they are different, since the circuit is optimal).
- Note that $Q = Q(x_i, x_j)$ can only take two values, 0 and 1, when x_i and x_j are fixed.

Schnorr's $2n$ Lower Bound

Theorem

If $f \in Q^n_{2,3}$, then $C(f) \geq 2n - 8$.

Proof

- Induction on n. If $n \leq 4$, then the statement is trivial.
- Consider an optimal circuit and its top gate Q which is fed by different variables x_i and x_j (they are different, since the circuit is optimal).
- Note that $Q = Q(x_i, x_j)$ can only take two values, 0 and 1, when x_i and x_j are fixed.
- Thus, either x_i or x_j fans out to another gate P.

Schnorr's $2n$ Lower Bound

Theorem

If $f \in Q^n_{2,3}$, then $C(f) \geq 2n - 8$.

Proof

- Induction on n. If $n \leq 4$, then the statement is trivial.
- Consider an optimal circuit and its top gate Q which is fed by different variables x_i and x_j (they are different, since the circuit is optimal).
- Note that $Q = Q(x_i, x_j)$ can only take two values, 0 and 1, when x_i and x_j are fixed.
- Thus, either x_i or x_j fans out to another gate P.
- By assigning this variable, we eliminate at least two gates and get a subfunction from $Q^{n-1}_{2,3}$. $\square$

AND-type Gates vs XOR-type Gates

Binary functions

The set B_2 of all binary functions contains 16 functions $f(x, y)$:

AND-type Gates vs XOR-type Gates

Binary functions

The set B_2 of all binary functions contains 16 functions $f(x, y)$:

1. 2 constants: 0, 1

AND-type Gates vs XOR-type Gates

Binary functions

The set B_2 of all binary functions contains 16 functions $f(x, y)$:

1. 2 constants: 0, 1
2. 4 degenerate functions: x, $\bar{x}$, y, $\bar{y}$.

AND-type Gates vs XOR-type Gates

Binary functions

The set B_2 of all binary functions contains 16 functions $f(x, y)$:

1. 2 constants: 0, 1
2. 4 degenerate functions: x, $\bar{x}$, y, $\bar{y}$.
3. 2 XOR-type functions: $x \oplus y \oplus a$, where $a \in \{0, 1\}$.

AND-type Gates vs XOR-type Gates

Binary functions

The set B_2 of all binary functions contains 16 functions $f(x,y)$:

1. 2 constants: 0, 1
2. 4 degenerate functions: x, $\bar{x}$, y, $\bar{y}$.
3. 2 XOR-type functions: $x \oplus y \oplus a$, where $a \in \{0,1\}$.
4. 8 AND-type functions: $(x \oplus a)(y \oplus b) \oplus c$, where $a, b, c \in \{0,1\}$.

AND-type Gates vs XOR-type Gates

Binary functions

The set B_2 of all binary functions contains 16 functions $f(x, y)$:

1. 2 constants: 0, 1
2. 4 degenerate functions: x, $\bar{x}$, y, $\bar{y}$.
3. 2 XOR-type functions: $x \oplus y \oplus a$, where $a \in \{0, 1\}$.
4. 8 AND-type functions: $(x \oplus a)(y \oplus b) \oplus c$, where $a, b, c \in \{0, 1\}$.

Remark

Optimal circuits contain AND- and XOR-type gates only, as constant and degenerate gates can be easily eliminated.

AND-type Gates vs XOR-type Gates

AND-type Gates vs XOR-type Gates

AND-type Gates vs XOR-type Gates

AND-type Gates vs XOR-type Gates

- AND-type gates are easier to handle than XOR-type gates.

AND-type Gates vs XOR-type Gates

AND-type Gates vs XOR-type Gates

- AND-type gates are easier to handle than XOR-type gates.
- Let $Q(x_i, x_j) = (x_i \oplus a)(x_j \oplus b) \oplus c$ be an AND-type gate. Then by assigning $x_i = a$ or $x_j = b$ we make this gate constant. That is, we eliminate not only this gate, but also all its direct successors!

AND-type Gates vs XOR-type Gates

AND-type Gates vs XOR-type Gates

- AND-type gates are easier to handle than XOR-type gates.
- Let $Q(x_i, x_j) = (x_i \oplus a)(x_j \oplus b) \oplus c$ be an AND-type gate. Then by assigning $x_i = a$ or $x_j = b$ we make this gate constant. That is, we eliminate not only this gate, but also all its direct successors!
- While by assigning any constant to x_i, we obtain from $Q(x_i, x_j) = x_i \oplus x_j \oplus c$ either x_j or $\bar{x}_j$.

AND-type Gates vs XOR-type Gates

AND-type Gates vs XOR-type Gates

- AND-type gates are easier to handle than XOR-type gates.
- Let $Q(x_i, x_j) = (x_i \oplus a)(x_j \oplus b) \oplus c$ be an AND-type gate. Then by assigning $x_i = a$ or $x_j = b$ we make this gate constant. That is, we eliminate not only this gate, but also all its direct successors!
- While by assigning any constant to x_i, we obtain from $Q(x_i, x_j) = x_i \oplus x_j \oplus c$ either x_j or $\bar{x}_j$.
- That is why, in particular, the current record bounds for circuits over $U_2 = B_2 \setminus \{\oplus, \equiv\}$ are stronger than the bounds over B_2.

AND-type Gates vs XOR-type Gates

AND-type Gates vs XOR-type Gates

- AND-type gates are easier to handle than XOR-type gates.
- Let $Q(x_i, x_j) = (x_i \oplus a)(x_j \oplus b) \oplus c$ be an AND-type gate. Then by assigning $x_i = a$ or $x_j = b$ we make this gate constant. That is, we eliminate not only this gate, but also all its direct successors!
- While by assigning any constant to x_i, we obtain from $Q(x_i, x_j) = x_i \oplus x_j \oplus c$ either x_j or $\bar{x}_j$.
- That is why, in particular, the current record bounds for circuits over $U_2 = B_2 \setminus \{\oplus, \equiv\}$ are stronger than the bounds over B_2.
- Usually, the main bottleneck of a proof based on gate elimination is a circuit whose top contains many XOR-type gates.

Polynomials over $\mathrm{GF}(2)$

Polynomials over $\mathrm{GF}(2)$

Polynomials over $\mathrm{GF}(2)$

Polynomials over $\mathrm{GF}(2)$

- Let $\tau(f)$ denote the unique polynomial over $\mathrm{GF}(2)$ representing f.

Polynomials over $\mathrm{GF}(2)$

Polynomials over $\mathrm{GF}(2)$

- Let $\tau(f)$ denote the unique polynomial over $\mathrm{GF}(2)$ representing f.
- E.g., $\tau(\mathrm{MOD}^3_{3,0}) = x_1x_2x_3 + (1-x_1)(1-x_2)(1-x_3)$.

Polynomials over $\mathrm{GF}(2)$

Polynomials over $\mathrm{GF}(2)$

- Let $\tau(f)$ denote the unique polynomial over $\mathrm{GF}(2)$ representing f.
- E.g., $\tau(\mathrm{MOD}^3_{3,0}) = x_1x_2x_3 + (1-x_1)(1-x_2)(1-x_3)$.
- Note that $\tau(f)$ is multi-linear.

Polynomials over $\mathrm{GF}(2)$

Polynomials over $\mathrm{GF}(2)$

- Let $\tau(f)$ denote the unique polynomial over $\mathrm{GF}(2)$ representing f.
- E.g., $\tau(\mathrm{MOD}^3_{3,0}) = x_1x_2x_3 + (1 - x_1)(1 - x_2)(1 - x_3)$.
- Note that $\tau(f)$ is multi-linear.
- It can be easily shown that, for any r, $\deg(\tau(\mathrm{MOD}^n_{4,r})) \leq 3$, while $\deg(\tau(\mathrm{MOD}^n_{3,r})) \geq n - 1$.

Polynomials over $\mathrm{GF}(2)$

Polynomials over $\mathrm{GF}(2)$

- Let $\tau(f)$ denote the unique polynomial over $\mathrm{GF}(2)$ representing f.
- E.g., $\tau(\mathrm{MOD}^3_{3,0}) = x_1x_2x_3 + (1-x_1)(1-x_2)(1-x_3)$.
- Note that $\tau(f)$ is multi-linear.
- It can be easily shown that, for any r, $\deg(\tau(\mathrm{MOD}^n_{4,r})) \leq 3$, while $\deg(\tau(\mathrm{MOD}^n_{3,r})) \geq n-1$.

Lemma (Degree lower bound)

Any circuit computing f contains at least $\deg(\tau(f)) - 1$ AND-type gates.

Combined Complexity Measure

Idea

Thus, in a bottleneck case we see only XOR-type gates, however we are given several AND-type gates in advance.

Combined Complexity Measure

Idea

Thus, in a bottleneck case we see only XOR-type gates, however we are given several AND-type gates in advance. Let us increase the weight of a XOR-type gate.

Combined Complexity Measure

Idea

Thus, in a bottleneck case we see only XOR-type gates, however we are given several AND-type gates in advance. Let us increase the weight of a XOR-type gate.

Definition

For a circuit C, let $A(C)$ and $X(C)$ denote the number of AND- and XOR-type gates in C, respectively. Let also $\mu(C) = 3X(C) + 2A(C)$.

An Improved Lower Bound

Lemma

For any circuit C computing $f \in Q^n_{2,3}$, $\mu(C) = 3X(C) + 2A(C) \geq 6n - 24$.

An Improved Lower Bound

Lemma

For any circuit C computing $f \in Q^n_{2,3}$, $\mu(C) = 3X(C) + 2A(C) \geq 6n - 24$.

Proof

An Improved Lower Bound

Lemma

For any circuit C computing $f \in Q^n_{2,3}$, $\mu(C) = 3X(C) + 2A(C) \geq 6n - 24$.

Proof

- As in the previous proof, we consider a top gate $Q(x_i, x_j)$ and assume wlog that x_i feeds also another gate P.

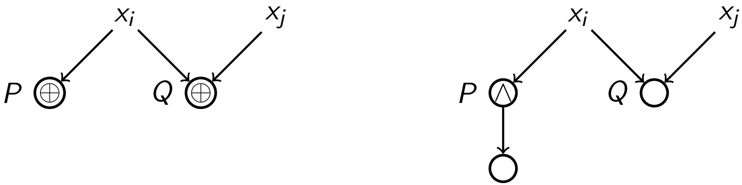
An Improved Lower Bound

Lemma

For any circuit C computing $f \in Q^n_{2,3}$, $\mu(C) = 3X(C) + 2A(C) \geq 6n - 24$.

Proof

- As in the previous proof, we consider a top gate $Q(x_i, x_j)$ and assume wlog that x_i feeds also another gate P.
- There are two cases:



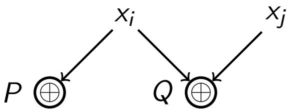
An Improved Lower Bound

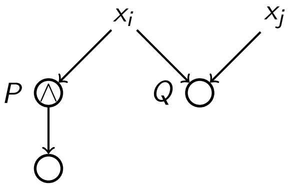
Lemma

For any circuit C computing $f \in Q^n_{2,3}$, $\mu(C) = 3X(C) + 2A(C) \geq 6n - 24$.

Proof

- As in the previous proof, we consider a top gate $Q(x_i, x_j)$ and assume wlog that x_i feeds also another gate P.
- There are two cases:





- In both cases, we can assign x_i a constant such that μ is reduced at least by 6. □

$7n/3$ Lower Bound

Lemma

Let $f \in Q^n_{2,3}$ and $\deg(\tau(f)) \geq n - c$, then $C(f) \geq 7n/3 - c'$.

$7n/3$ Lower Bound

Lemma

Let $f \in Q^n_{2,3}$ and $\deg(\tau(f)) \geq n - c$, then $C(f) \geq 7n/3 - c'$.

proof

Let C be an optimal circuit computing f.

$7n/3$ Lower Bound

Lemma

Let $f \in Q^n_{2,3}$ and $\deg(\tau(f)) \geq n - c$, then $C(f) \geq 7n/3 - c'$.

proof

Let C be an optimal circuit computing f.

$$\begin{array}{r}
3X(C)+2A(C) \geq 6n - 24 \\
A(C) \geq n - c - 1 \\
\hline
C(f) = 3X(C)+3A(C) \geq 7n - 25 - c
\end{array}$$

$\square$

Further Improvements

Further Improvements

- Prove a stronger lower bound on μ. A more involved case analysis is needed.

Further Improvements

- Prove a stronger lower bound on μ. A more involved case analysis is needed.
- Prove stronger lower bound on $A(C)$.

Further Improvements

- Prove a stronger lower bound on μ. A more involved case analysis is needed.
- Prove stronger lower bound on $A(C)$.
  - Remind that $\tau(\mathrm{MOD}_3^n) = x_1 x_2 \dots x_n + \dots$, so any circuit computing MOD_3^n must have at least $(n-1)$ AND-type gates just in order to compute this monomial.

Further Improvements

- Prove a stronger lower bound on μ. A more involved case analysis is needed.
- Prove stronger lower bound on $A(C)$.
  - Remind that $\tau(\mathrm{MOD}_3^n) = x_1x_2\ldots x_n + \ldots$, so any circuit computing MOD_3^n must have at least $(n-1)$ AND-type gates just in order to compute this monomial.
  - Probably, more AND-type gates are needed to compute all the other monomials?

Further Improvements

- Prove a stronger lower bound on μ. A more involved case analysis is needed.
- Prove stronger lower bound on $A(C)$.
  - Remind that $\tau(\mathrm{MOD}_3^n) = x_1x_2 \dots x_n + \dots$, so any circuit computing MOD_3^n must have at least $(n-1)$ AND-type gates just in order to compute this monomial.
  - Probably, more AND-type gates are needed to compute all the other monomials?
  - No, there is a circuit computing MOD_3^n of size $3n$ containing exactly n AND-type gates.

Further Improvements

- Prove a stronger lower bound on μ. A more involved case analysis is needed.
- Prove stronger lower bound on $A(C)$.
  - Remind that $\tau(\mathrm{MOD}_3^n) = x_1 x_2 \dots x_n + \dots$, so any circuit computing MOD_3^n must have at least $(n-1)$ AND-type gates just in order to compute this monomial.
  - Probably, more AND-type gates are needed to compute all the other monomials?
  - No, there is a circuit computing MOD_3^n of size $3n$ containing exactly n AND-type gates.
  - Moreover, any symmetric function can be computed using only n AND-type gates.

Further Improvements

- Prove a stronger lower bound on μ. A more involved case analysis is needed.
- Prove stronger lower bound on $A(C)$.
  - Remind that $\tau(\mathrm{MOD}_3^n) = x_1x_2\ldots x_n + \ldots$, so any circuit computing MOD_3^n must have at least $(n-1)$ AND-type gates just in order to compute this monomial.
  - Probably, more AND-type gates are needed to compute all the other monomials?
  - No, there is a circuit computing MOD_3^n of size $3n$ containing exactly n AND-type gates.
  - Moreover, any symmetric function can be computed using only n AND-type gates.
  - No lower bound better than $n-1$ is known! Though the multiplicative complexity of almost all functions is exponential.

MOD_3 vs MOD_4

MOD_3 is not simpler than MOD_4

For circuits and formulas over B_2 and U_2, it is known that MOD_3 is not simpler than MOD_4. The exact complexity of MOD_4 is known for some of these models: $C_{B_2}(\mathrm{MOD}_4^n) = 2.5n - c$, $L_{B_2}(\mathrm{MOD}_4^n) = \Theta(n \log n)$. The exact complexity of MOD_3 is known for none of these models.

MOD_3 vs MOD_4

MOD_3 is not simpler than MOD_4

For circuits and formulas over B_2 and U_2, it is known that MOD_3 is not simpler than MOD_4. The exact complexity of MOD_4 is known for some of these models: $C_{B_2}(\mathrm{MOD}_4^n) = 2.5n - c$, $L_{B_2}(\mathrm{MOD}_4^n) = \Theta(n \log n)$. The exact complexity of MOD_3 is known for none of these models.

Why MOD_3 must be harder than MOD_4?

MOD_3 vs MOD_4

MOD_3 is not simpler than MOD_4

For circuits and formulas over B_2 and U_2, it is known that MOD_3 is not simpler than MOD_4. The exact complexity of MOD_4 is known for some of these models: $C_{B_2}(\mathrm{MOD}_4^n) = 2.5n - c$, $L_{B_2}(\mathrm{MOD}_4^n) = \Theta(n \log n)$. The exact complexity of MOD_3 is known for none of these models.

Why MOD_3 must be harder than MOD_4?

- 4 is a power of 2, 3 is not. To compute MOD_4^n, compute the bit representation of $\sum x_i$ and check the last two bits.

MOD_3 vs MOD_4

MOD_3 is not simpler than MOD_4

For circuits and formulas over B_2 and U_2, it is known that MOD_3 is not simpler than MOD_4. The exact complexity of MOD_4 is known for some of these models: $C_{B_2}(\mathrm{MOD}_4^n) = 2.5n - c$, $L_{B_2}(\mathrm{MOD}_4^n) = \Theta(n \log n)$. The exact complexity of MOD_3 is known for none of these models.

Why MOD_3 must be harder than MOD_4?

- 4 is a power of 2, 3 is not. To compute MOD_4^n, compute the bit representation of $\sum x_i$ and check the last two bits.
- MOD_3 survives under substitutions like $x_i = x_j$.

MOD_3 vs MOD_4

MOD_3 is not simpler than MOD_4

For circuits and formulas over B_2 and U_2, it is known that MOD_3 is not simpler than MOD_4. The exact complexity of MOD_4 is known for some of these models: $C_{B_2}(\mathrm{MOD}_4^n) = 2.5n - c$, $L_{B_2}(\mathrm{MOD}_4^n) = \Theta(n \log n)$. The exact complexity of MOD_3 is known for none of these models.

Why MOD_3 must be harder than MOD_4?

- 4 is a power of 2, 3 is not. To compute MOD_4^n, compute the bit representation of $\sum x_i$ and check the last two bits.
- MOD_3 survives under substitutions like $x_i = x_j$.
- $C_{B_2}(\mathrm{MOD}_3^n)$ for $n \leq 5$ "grows like" $3n$.

Open Problems

Open Problems

1. Close the gaps:

$$2.5n \leq C_{B_2}(\mathrm{MOD}_3^n) \leq 3n$$

$$4n \leq C_{U_2}(\mathrm{MOD}_4^n) \leq 5n$$

Open Problems

1. Close the gaps:

$$2.5n \leq C_{B_2}(\mathrm{MOD}_3^n) \leq 3n$$

$$4n \leq C_{U_2}(\mathrm{MOD}_4^n) \leq 5n$$

2. Prove a cn lower bound (for a constant $c > 1$) on the multiplicative complexity of an explicit Boolean function.

Thank you for your attention!