Circuit Complexity and Multiplicative Complexity of Boolean Functions

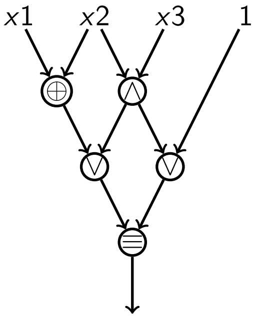
Alexander S. Kulikov
joint work with Arist Kojevnikov

Steklov Institute of Mathematics at St. Petersburg

Franco-Russian workshop on Algorithms, complexity and applications
14 June 2010

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 variable 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 variable 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 variable 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

$2n - c$	[Kloss, Malyshev, 65]
$2n - c$	[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

$2n - c$	[Kloss, Malyshev, 65]
$2n - c$	[Schnorr, 74]
$2.5n - o(n)$	[Paul, 77]
$2.5n - c$	[Stockmeyer, 77]
$3n - o(n)$	[Blum, 84]

This Talk

Known Lower Bounds for Circuits over B_2

Known Lower Bounds

$2n - c$	[Kloss, Malyshev, 65]
$2n - c$	[Schnorr, 74]
$2.5n - o(n)$	[Paul, 77]
$2.5n - c$	[Stockmeyer, 77]
$3n - o(n)$	[Blum, 84]

This Talk

- We 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

$2n - c$	[Kloss, Malyshev, 65]
$2n - c$	[Schnorr, 74]
$2.5n - o(n)$	[Paul, 77]
$2.5n - c$	[Stockmeyer, 77]
$3n - o(n)$	[Blum, 84]

This Talk

- We present a proof of a $7n/3 - c$ lower bound which is as simple as Schnorr's proof of $2n - c$ lower bound.
- The key idea is a combined circuit complexity measure assigning different weight to gates of different types.

Gate Elimination Method

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

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.

The main idea

Gate Elimination Method

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.

The main idea

- Take an optimal circuit for the function in question.

Gate Elimination Method

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.

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

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.

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

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.

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

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.

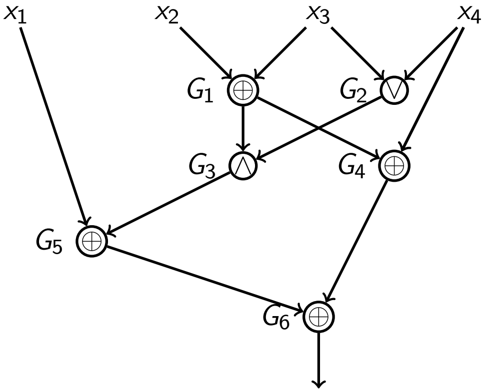
The main idea

- 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.

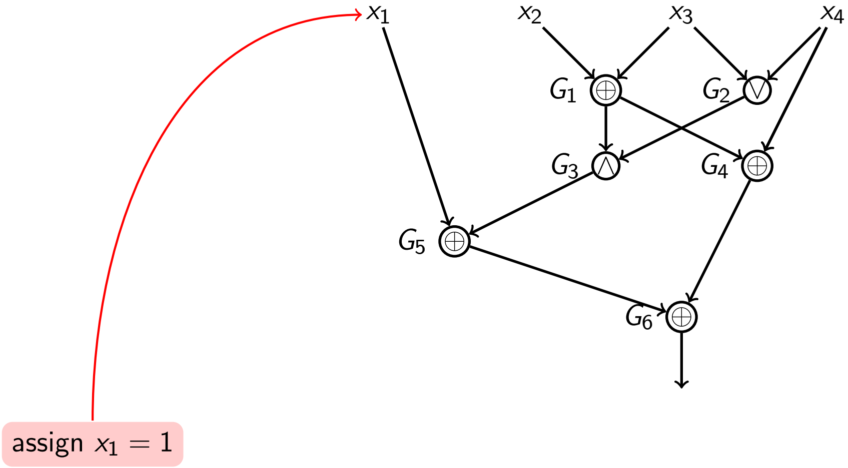
Remark

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

Example

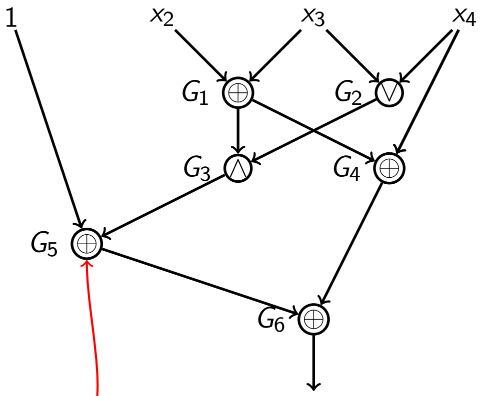


Example



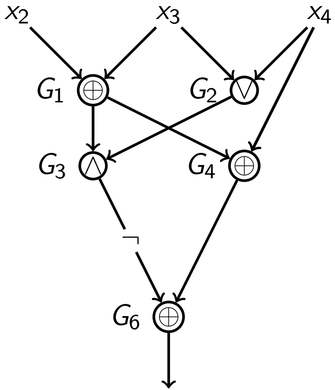
x_1
x_2
x_3
x_4
G_1 $\oplus$
G_2 $\vee$
G_3 $\wedge$
G_4 $\oplus$
G_5 $\oplus$
G_6 $\oplus$
assign $x_1 = 1$

Example

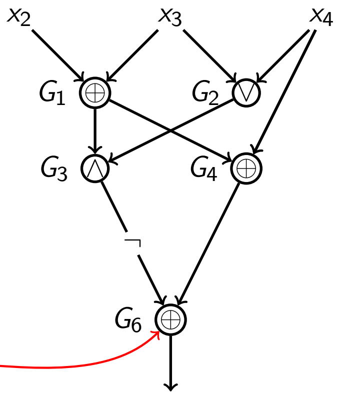


G_5 now computes $G_3 \oplus 1 = \neg G_3$

Example

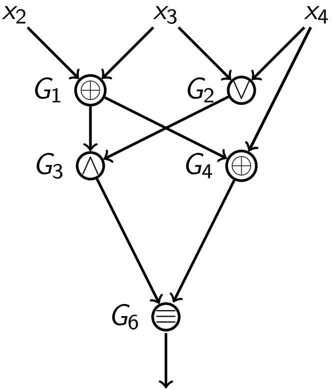


Example

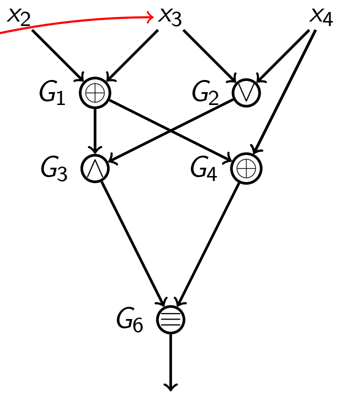


now we can change the binary function assigned to G_6

Example

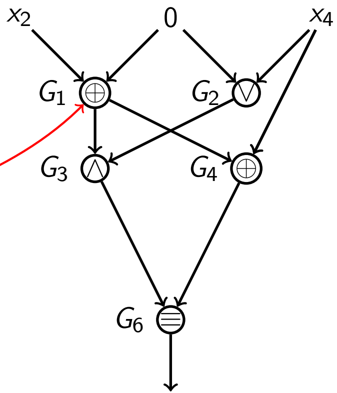


Example



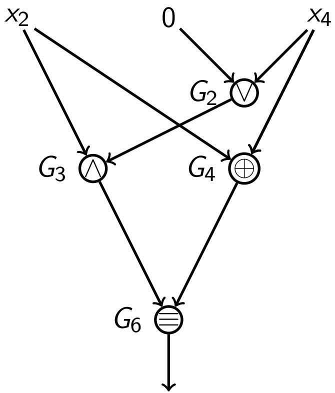
now assign $x_3 = 0$

Example

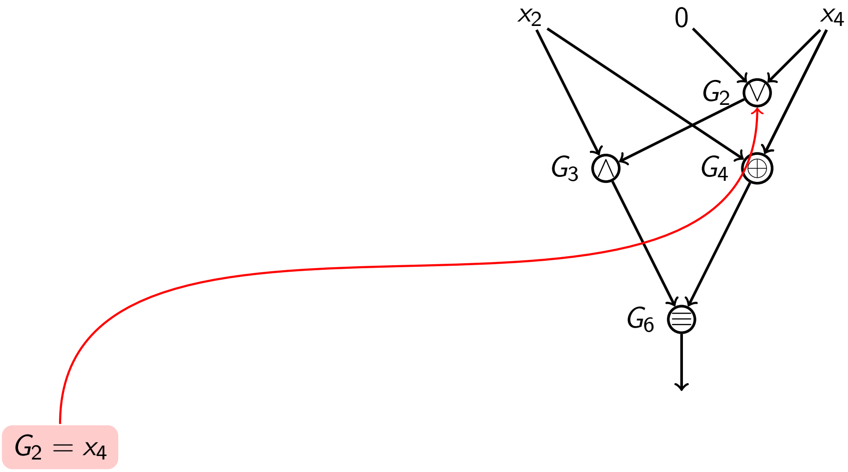


G_1 then is equal to x_2

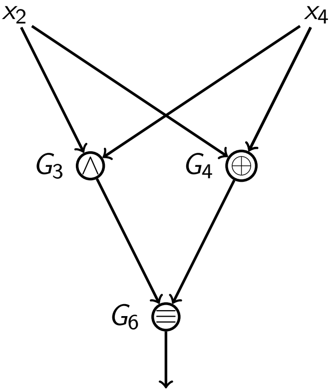
Example



Example



Example



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

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 $i, j \in \{1, \ldots, n\}$, 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.

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

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;
2. for all $i \in \{1, \ldots, n\}$, one obtains a subfunction in $Q^{n-1}_{2,3}$ (if $n \geq 4$) by replacing x_i by any constant.

Modular functions

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

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;
2. for all $i \in \{1, \ldots, n\}$, one obtains a subfunction in $Q^{n-1}_{2,3}$ (if $n \geq 4$) by replacing x_i by any constant.

Modular functions

- Let $\mathrm{MOD}^n_{m,r}(x_1, \ldots, x_n) = 1$ iff $\sum_{i=1}^n x_i \equiv r \pmod{m}$.
- Then $\mathrm{MOD}^n_{3,r}, \mathrm{MOD}^n_{4,r} \in Q^n_{2,3}$, but $\mathrm{MOD}^n_{2,r} \notin Q^n_{2,3}$ (as one can only get $\mathrm{MOD}^{n-2}_{2,0}$ and $\mathrm{MOD}^{n-2}_{2,1}$ from $\mathrm{MOD}^n_{2,r}$ by fixing two constants).

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 $Q^{n-1}_{2,3}$. □

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 $\bar{x}_j$.
- This 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 $\bar{x}_j$.
- This 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.

Multiplicative Complexity of Boolean Functions

Multiplicative Complexity of Boolean Functions

- The minimal possible number of AND-type gates in a circuit computing f is called the multiplicative complexity of f.

Multiplicative Complexity of Boolean Functions

- The minimal possible number of AND-type gates in a circuit computing f is called the multiplicative complexity of f.
- The multiplicative complexity of almost all Boolean function is about $2^{n/2}$.

Multiplicative Complexity of Boolean Functions

- The minimal possible number of AND-type gates in a circuit computing f is called the multiplicative complexity of f.
- The multiplicative complexity of almost all Boolean function is about $2^{n/2}$.
- The best known lower bound is $n-1$ (holds even for the conjunction of n variables).

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., $\tau(\mathrm{MOD}^3_{3,0}) = x_1x_2x_3 \oplus (1 \oplus x_1)(1 \oplus x_2)(1 \oplus x_3)$.

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., $\tau(\mathrm{MOD}^3_{3,0}) = x_1x_2x_3 \oplus (1 \oplus x_1)(1 \oplus x_2)(1 \oplus x_3)$.
- 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., $\tau(\mathrm{MOD}^3_{3,0}) = x_1x_2x_3 \oplus (1 \oplus x_1)(1 \oplus x_2)(1 \oplus x_3)$.
- Note that $\tau(f)$ is multi-linear.

Lemma (Degree lower bound)

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

Proof of the Degree Lower Bound

Proof of the Degree Lower Bound

- We show that a circuit with d AND-type gates cannot compute a function of degree more than $d+1$ by induction.

Proof of the Degree Lower Bound

- We show that a circuit with d AND-type gates cannot compute a function of degree more than $d+1$ by induction.
- Note that it is important that we are working in GF(2). To compute x^k one needs only about $\log k$ binary multiplications.

Proof of the Degree Lower Bound

- We show that a circuit with d AND-type gates cannot compute a function of degree more than $d+1$ by induction.
- Note that it is important that we are working in GF(2). To compute x^k one needs only about $\log k$ binary multiplications.
- Consider the first (in a topological sorting of all the gates) AND-type gate. Note that it computes a function of degree at most 2.

Proof of the Degree Lower Bound

- We show that a circuit with d AND-type gates cannot compute a function of degree more than $d + 1$ by induction.
- Note that it is important that we are working in GF(2). To compute x^k one needs only about $\log k$ binary multiplications.
- Consider the first (in a topological sorting of all the gates) AND-type gate. Note that it computes a function of degree at most 2.
- Replace this gate by a new variable. We now have a circuit with $d - 1$ AND-type gates and hence, by induction, it computes a function of degree at mots d.

Proof of the Degree Lower Bound

- We show that a circuit with d AND-type gates cannot compute a function of degree more than $d + 1$ by induction.
- Note that it is important that we are working in GF(2). To compute x^k one needs only about $\log k$ binary multiplications.
- Consider the first (in a topological sorting of all the gates) AND-type gate. Note that it computes a function of degree at most 2.
- Replace this gate by a new variable. We now have a circuit with $d - 1$ AND-type gates and hence, by induction, it computes a function of degree at mots d.
- Now return back the removed gate. Since we are in GF(2), this increases the degree by at most 1. □

Combined Complexity Measure

Idea

In a typical bottleneck case we usually have only XOR-type gates, however we are given several AND-type gates in advance.

Combined Complexity Measure

Idea

In a typical bottleneck case we usually have 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

In a typical bottleneck case we usually have 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.

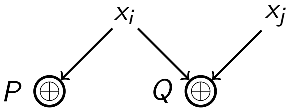
An Improved Lower Bound

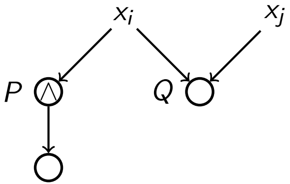
Lemma

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:





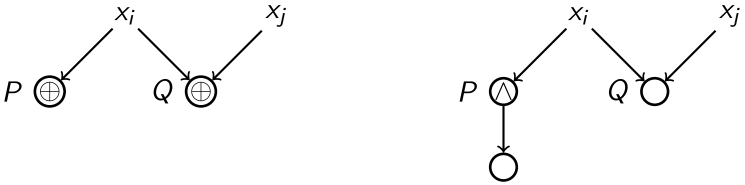
An Improved Lower Bound

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 the former case, by assigning x_i a constant one eliminates both XOR-type gates. In the latter, by assigning x_i the right constant one eliminates P, Q and all successors of P. In both cases, μ is reduced by at least 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.

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

□

Improving the Degree Lower Bound

Improving the Degree Lower Bound

- Consider the function $\mathrm{MOD}^3_{3,2}$:

$$\tau(\mathrm{MOD}^3_{3,2}) \quad = \quad x_1x_2(1 \oplus x_3) \oplus x_1(1 \oplus x_2)x_3 \oplus (1 \oplus x_1)x_2x_3 =$$

Improving the Degree Lower Bound

- Consider the function $\mathrm{MOD}^3_{3,2}$:

$$\begin{aligned}\tau(\mathrm{MOD}^3_{3,2}) &= x_1x_2(1\oplus x_3)\oplus x_1(1\oplus x_2)x_3\oplus(1\oplus x_1)x_2x_3 = \\ &= x_1x_2x_3\oplus x_1x_2\oplus x_2x_3\oplus x_1x_3\end{aligned}$$

Improving the Degree Lower Bound

- Consider the function $\mathrm{MOD}^3_{3,2}$:

$$\begin{aligned}\tau(\mathrm{MOD}^3_{3,2}) &= x_1x_2(1\oplus x_3)\oplus x_1(1\oplus x_2)x_3\oplus(1\oplus x_1)x_2x_3 = \\ &= x_1x_2x_3\oplus x_1x_2\oplus x_2x_3\oplus x_1x_3\end{aligned}$$

- The Degree Lower Bound Lemma says that one needs at least two binary multiplications to compute $\mathrm{MOD}^3_{3,2}$, just because $\tau(\mathrm{MOD}^3_{3,2})$ contains a monomial of degree 3.

Improving the Degree Lower Bound

- Consider the function $\mathrm{MOD}^3_{3,2}$:

$$\begin{aligned}\tau(\mathrm{MOD}^3_{3,2}) &= x_1x_2(1\oplus x_3)\oplus x_1(1\oplus x_2)x_3\oplus(1\oplus x_1)x_2x_3 = \\ &= x_1x_2x_3\oplus x_1x_2\oplus x_2x_3\oplus x_1x_3\end{aligned}$$

- The Degree Lower Bound Lemma says that one needs at least two binary multiplications to compute $\mathrm{MOD}^3_{3,2}$, just because $\tau(\mathrm{MOD}^3_{3,2})$ contains a monomial of degree 3.
- It however contains other monomials, so probably one needs more binary multiplications in the worst case to compute $\mathrm{MOD}^3_{3,2}$?

Improving the Degree Lower Bound

- Consider the function $\mathrm{MOD}^3_{3,2}$:

$$\begin{aligned}\tau(\mathrm{MOD}^3_{3,2}) &= x_1x_2(1\oplus x_3)\oplus x_1(1\oplus x_2)x_3\oplus(1\oplus x_1)x_2x_3 = \\ &= x_1x_2x_3\oplus x_1x_2\oplus x_2x_3\oplus x_1x_3\end{aligned}$$

- The Degree Lower Bound Lemma says that one needs at least two binary multiplications to compute $\mathrm{MOD}^3_{3,2}$, just because $\tau(\mathrm{MOD}^3_{3,2})$ contains a monomial of degree 3.
- It however contains other monomials, so probably one needs more binary multiplications in the worst case to compute $\mathrm{MOD}^3_{3,2}$?
- No, two multiplications are enough:

$$\tau(\mathrm{MOD}^3_{3,2}) = (x_1\oplus x_2\oplus x_1x_2)(x_1\oplus x_2\oplus x_3\oplus 1)\,.$$

Improving the Degree Lower Bound

- Consider the function $\mathrm{MOD}^3_{3,2}$:

$$\begin{aligned}\tau(\mathrm{MOD}^3_{3,2}) &= x_1x_2(1\oplus x_3)\oplus x_1(1\oplus x_2)x_3\oplus(1\oplus x_1)x_2x_3 = \\ &= x_1x_2x_3\oplus x_1x_2\oplus x_2x_3\oplus x_1x_3\end{aligned}$$

- The Degree Lower Bound Lemma says that one needs at least two binary multiplications to compute $\mathrm{MOD}^3_{3,2}$, just because $\tau(\mathrm{MOD}^3_{3,2})$ contains a monomial of degree 3.
- It however contains other monomials, so probably one needs more binary multiplications in the worst case to compute $\mathrm{MOD}^3_{3,2}$?
- No, two multiplications are enough:

$$\tau(\mathrm{MOD}^3_{3,2}) = (x_1\oplus x_2\oplus x_1x_2)(x_1\oplus x_2\oplus x_3\oplus 1)\,.$$

- Moreover, the multiplicative complexity of any symmetric function is at most $n+o(n)$ [Boyar, Peralta, Pochuev, 2000].

Open Problems

Open Problems

1. Improve the lower bounds $3n - o(n)$ and $5n - o(n)$ for the circuit complexity over B_2 and U_2, respectively.

Open Problems

1. Improve the lower bounds $3n - o(n)$ and $5n - o(n)$ for the circuit complexity over B_2 and U_2, respectively.
2. 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. Improve the lower bounds $3n - o(n)$ and $5n - o(n)$ for the circuit complexity over B_2 and U_2, respectively.
2. Close the gaps:

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

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

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

Thank you for your attention!