Lower bounds
in Boolean and algebraic complexity

Sylvain Perifel

LIAFA, Université Paris 7

Journées nationales d'informatique mathématique
Paris, January 21, 2010

Lower bounds

- Lower bounds on resources needed to solve problems (time, space, etc.): central questions in complexity
- What problems are not efficiently computable?

Lower bounds

- Lower bounds on resources needed to solve problems (time, space, etc.): central questions in complexity
- What problems are not efficiently computable?

- Foundation of computational complexity theory: hierarchy theorems (Hartmanis & Stearns 1965)

  with more time one can decide more things

  = lower bound on certain ad-hoc problems
- More challenging problem: proving a superpolynomial lower bound on the time of any algorithm for SAT
- Even modest lower bounds appear hard to prove

Lower bounds

- Lower bounds on resources needed to solve problems (time, space, etc.): central questions in complexity
- What problems are not efficiently computable?

- Foundation of computational complexity theory: hierarchy theorems (Hartmanis & Stearns 1965)
  with more time one can decide more things
  = lower bound on certain ad-hoc problems
- More challenging problem: proving a superpolynomial lower bound on the time of any algorithm for SAT
- Even modest lower bounds appear hard to prove

- Links with derandomization, cryptography, etc.

Algebraic vs Boolean

- $\{0, 1\}$: poor structure $\rightarrow$ extension to other finite fields, or to $\mathbb{R}$, $\mathbb{C}$, etc. (More mathematical tools)
- "Arithmetization": e.g. $\mathsf{IP} = \mathsf{PSPACE}$ (Shamir 1992)

Algebraic vs Boolean

- $\{0, 1\}$: poor structure $\rightarrow$ extension to other finite fields, or to $\mathbb{R}$, $\mathbb{C}$, etc. (More mathematical tools)
- "Arithmetization": e.g. $\mathsf{IP} = \mathsf{PSPACE}$ (Shamir 1992)

- Algebraic models: natural models to deal with algebraic problems (polynomials), operations $+$ and $\times$.
  Rich structure.
- Same questions as in the Boolean world (P vs NP, etc.).
  Many links (transfer theorems): e.g. on some structures, equivalent "P vs NP" questions

Outline

1. Boolean circuits

2. Derandomization

3. Lower bounds on the computation of polynomials

4. Permanent

Outline

1. Boolean circuits

2. Derandomization

3. Lower bounds on the computation of polynomials

4. Permanent

Why circuits?

- Easier model than Turing machines
  (→ easier to study? "Look inside circuits")
- If NP does not have polynomial-size circuits
  then $\mathsf{NP} \neq \mathsf{P}$

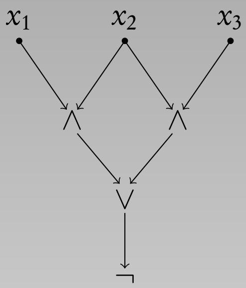
Why circuits?

- Easier model than Turing machines
  (→ easier to study? "Look inside circuits")
- If NP does not have polynomial-size circuits
  then NP $\neq$ P

- Idea behind circuits: one algorithm for one input size
  → possibly different algorithms for different input sizes
- Example
  adder for 32 bit integers vs adder for 33 bit integers:
  why use the same algorithm?

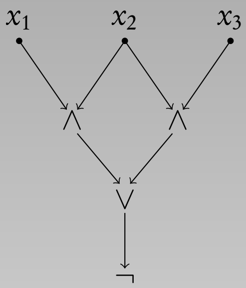
Boolean circuits: definition

- Directed acyclic graph with nodes labeled by $\neg$ (fan-in 1), $\vee, \wedge$ (fan-in 2). Inputs $x_1, \ldots, x_n$.



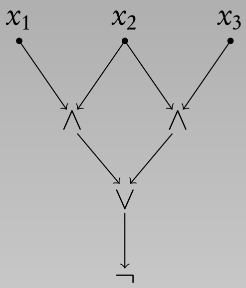
Boolean circuits: definition

- Directed acyclic graph with nodes labeled by $\neg$ (fan-in 1), $\vee$, $\wedge$ (fan-in 2).
  Inputs $x_1, \ldots, x_n$.
- Size = number of nodes
- Fixed number of inputs
  $\rightarrow$ words of same length



Boolean circuits: definition

- Directed acyclic graph with nodes labeled by $\neg$ (fan-in 1), $\vee$, $\wedge$ (fan-in 2). Inputs $x_1, \ldots, x_n$.
- Size = number of nodes
- Fixed number of inputs $\rightarrow$ words of same length
- A word x is accepted iff $C(x) = 1$
- C accepts a subset of $\{0, 1\}^n$
  $\rightarrow$ family of circuits (C_n) to recognize a language
  (C_n has n inputs)



Circuits vs TM

- A Turing machine running in time $t(n)$ can be simulated by a family of circuits (C_n) of size $t^2(n)$

Circuits vs TM

- A Turing machine running in time $t(n)$ can be simulated by a family of circuits (C_n) of size $t^2(n)$
- If a family of circuits (C_n) can be described efficiently (uniformity), then it can be simulated by a Turing machine running in time $t(n) \simeq |C_n|$
- If the family of circuits cannot be described (nonuniformity), the simulation is impossible: circuits can do more than TM (undecidable problems recognized by nonuniform circuits)

Circuits vs TM

- A Turing machine running in time $t(n)$ can be simulated by a family of circuits (C_n) of size $t^2(n)$

- If a family of circuits (C_n) can be described efficiently (uniformity), then it can be simulated by a Turing machine running in time $t(n) \simeq |C_n|$
- If the family of circuits cannot be described (nonuniformity), the simulation is impossible: circuits can do more than TM (undecidable problems recognized by nonuniform circuits)

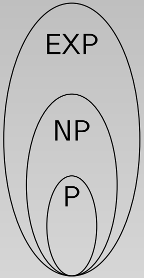
- Measure of complexity: size of the smallest circuits

Nonuniform complexity classes

- $\mathsf{P/poly}$ = languages recognized by families of polynomial-size circuits
- Uncountably many languages…

Nonuniform complexity classes

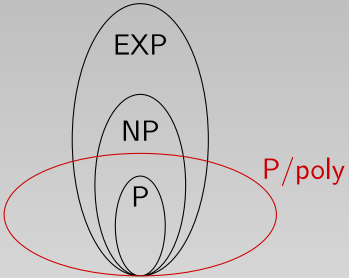
- P/poly = languages recognized by families of polynomial-size circuits
- Uncountably many languages…



$$\mathsf{EXP} = \mathsf{DTIME}(2^{n^{O(1)}})$$
$$\mathsf{NP} = \mathsf{NTIME}(n^{O(1)})$$
$$\mathsf{P} = \mathsf{DTIME}(n^{O(1)})$$

Nonuniform complexity classes

- P/poly = languages recognized by families of polynomial-size circuits
- Uncountably many languages…



$$\mathsf{EXP} = \mathsf{DTIME}(2^{n^{O(1)}})$$
$$\mathsf{NP} = \mathsf{NTIME}(n^{O(1)})$$
$$\mathsf{P} = \mathsf{DTIME}(n^{O(1)})$$
$$\mathsf{P/poly} = \mathsf{SIZE}(n^{O(1)})$$

Diagonalization

- Rooted in Cantor's diagonal argument
- Diagonalization against Turing machines:
  definition of a language L not recognized
  by any TM working in time $t(n)$:
  - (M_i) = enumeration of TMs working in time $t(n)$
  - $\mathtt{0}^n \in L \iff M_n(\mathtt{0}^n)$ does not accept

  $\rightarrow$ every M_n makes a mistake.

Diagonalization

- Rooted in Cantor's diagonal argument
- Diagonalization against Turing machines:
  definition of a language L not recognized
  by any TM working in time $t(n)$:
  - (M_i) = enumeration of TMs working in time $t(n)$
  - $0^n \in L \iff M_n(0^n)$ does not accept

  $\rightarrow$ every M_n makes a mistake.

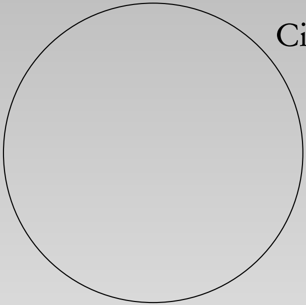
- If M_n makes a mistake on a single input (0^n among an infinite number of words), then it does not recognize the language
  $\rightarrow$ an infinite number of words can be used to diagonalize

Diagonalization against circuits

- For circuits : all circuits with n inputs must make a mistake on some input of size n
  (only 2^n inputs, but $\simeq 2^{n^k}$ circuits of size n^k).

Diagonalization against circuits

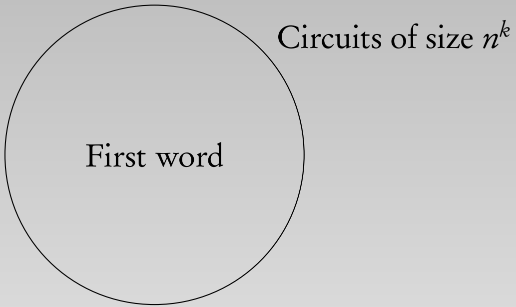
- For circuits : all circuits with n inputs must make a mistake on some input of size n
(only 2^n inputs, but $\simeq 2^{n^k}$ circuits of size n^k).
- An idea: eliminate half of the circuits at each step (take the answer of the minority of the circuits).



Circuits of size n^k

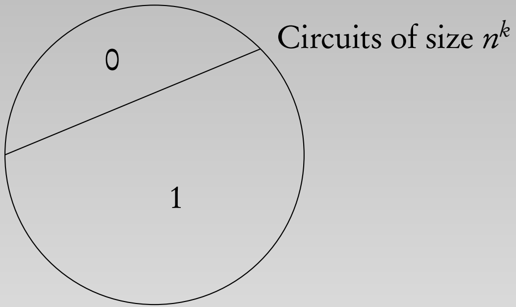
Diagonalization against circuits

- For circuits : all circuits with n inputs must make a mistake on some input of size n
  (only 2^n inputs, but $\simeq 2^{n^k}$ circuits of size n^k).
- An idea: eliminate half of the circuits at each step (take the answer of the minority of the circuits).



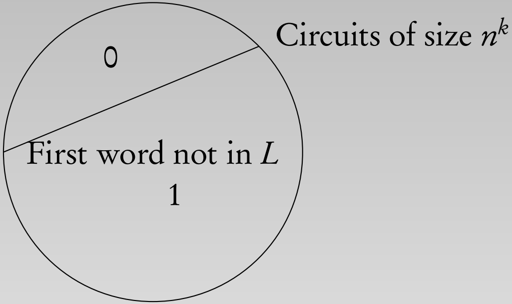
Diagonalization against circuits

- For circuits : all circuits with n inputs must make a mistake on some input of size n (only 2^n inputs, but $\simeq 2^{n^k}$ circuits of size n^k).
- An idea: eliminate half of the circuits at each step (take the answer of the minority of the circuits).



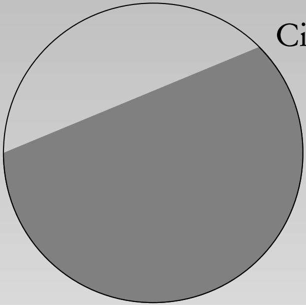
Diagonalization against circuits

- For circuits : all circuits with n inputs must make a mistake on some input of size n
(only 2^n inputs, but $\simeq 2^{n^k}$ circuits of size n^k).
- An idea: eliminate half of the circuits at each step (take the answer of the minority of the circuits).



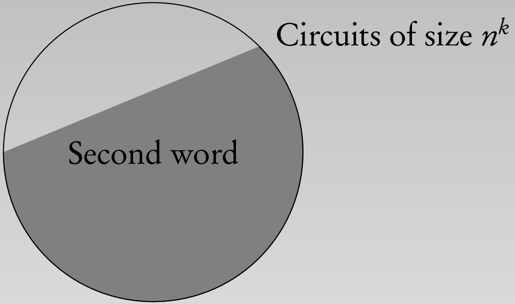
Diagonalization against circuits

- For circuits : all circuits with n inputs must make a mistake on some input of size n
(only 2^n inputs, but $\simeq 2^{n^k}$ circuits of size n^k).
- An idea: eliminate half of the circuits at each step (take the answer of the minority of the circuits).



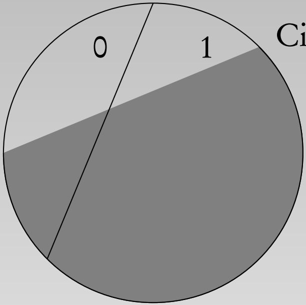
Diagonalization against circuits

- For circuits : all circuits with n inputs must make a mistake on some input of size n (only 2^n inputs, but $\simeq 2^{n^k}$ circuits of size n^k).
- An idea: eliminate half of the circuits at each step (take the answer of the minority of the circuits).



Diagonalization against circuits

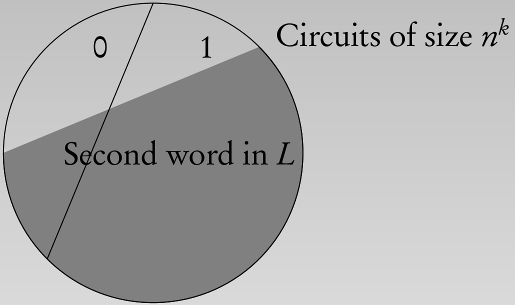
- For circuits : all circuits with n inputs must make a mistake on some input of size n
  (only 2^n inputs, but $\simeq 2^{n^k}$ circuits of size n^k).
- An idea: eliminate half of the circuits at each step (take the answer of the minority of the circuits).



Circuits of size n^k

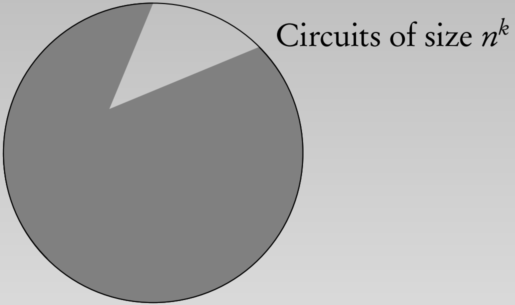
Diagonalization against circuits

- For circuits : all circuits with n inputs must make a mistake on some input of size n
(only 2^n inputs, but $\simeq 2^{n^k}$ circuits of size n^k).
- An idea: eliminate half of the circuits at each step (take the answer of the minority of the circuits).



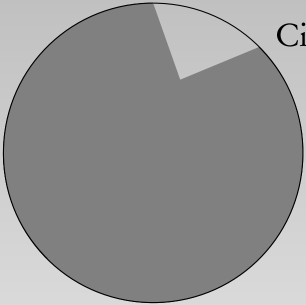
Diagonalization against circuits

- For circuits : all circuits with n inputs must make a mistake on some input of size n
(only 2^n inputs, but $\simeq 2^{n^k}$ circuits of size n^k).
- An idea: eliminate half of the circuits at each step (take the answer of the minority of the circuits).



Diagonalization against circuits

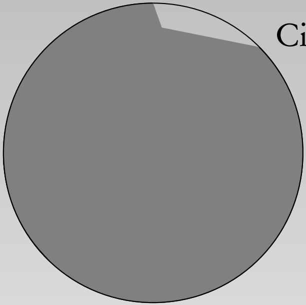
- For circuits : all circuits with n inputs must make a mistake on some input of size n
  (only 2^n inputs, but $\simeq 2^{n^k}$ circuits of size n^k).
- An idea: eliminate half of the circuits at each step (take the answer of the minority of the circuits).



Circuits of size n^k

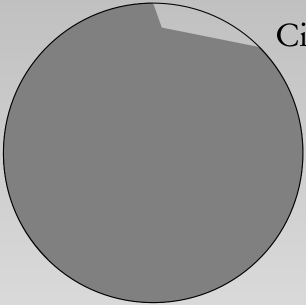
Diagonalization against circuits

- For circuits : all circuits with n inputs must make a mistake on some input of size n
(only 2^n inputs, but $\simeq 2^{n^k}$ circuits of size n^k).
- An idea: eliminate half of the circuits at each step (take the answer of the minority of the circuits).



Diagonalization against circuits

- For circuits : all circuits with n inputs must make a mistake on some input of size n
  (only 2^n inputs, but $\simeq 2^{n^k}$ circuits of size n^k).
- An idea: eliminate half of the circuits at each step (take the answer of the minority of the circuits).



- Not efficient enough for polynomial-size circuits

Diagonalization (2)

- Diagonalization possible on "small pieces" of circuits
  $\rightarrow$ ok if we can glue the pieces together
- Kolmogorov complexity "Symmetry of Information":
  $C(x, y) \simeq C(x) + C(y|x)$

Theorem

"Polynomial-time (SI)" implies $\mathsf{EXP} \not\subset \mathsf{P/poly}$

Proof idea

Diagonalize on pieces of circuits (by eliminating half of the circuits at each step). Glue pieces together thanks to (SI). ∎

Outline

Derandomization

- Can every probabilistic algorithm be turned into a deterministic one without slowdown? (or with "only" polynomial slowdown)
- In other words: does randomness speed up computation more than polynomially?

Derandomization

- Can every probabilistic algorithm be turned into a deterministic one without slowdown? (or with "only" polynomial slowdown)
- In other words: does randomness speed up computation more than polynomially?

Probabilistic algorithm:

- algorithm endowed with a generator of random bits,
- deciding a language with error $< 1/3$.

Derandomization

- Can every probabilistic algorithm be turned into a deterministic one without slowdown? (or with "only" polynomial slowdown)
- In other words: does randomness speed up computation more than polynomially?

Probabilistic algorithm:

- algorithm endowed with a generator of random bits,
- deciding a language with error $< 1/3$.

Examples:

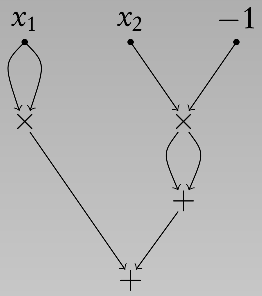
- derandomization of primality testing (AKS 2002) ($O(n^6)$ vs $O(n^2)$ probabilistically)
- PIT: testing if an arithmetic circuit computes the identically 0 polynomial

Arithmetic circuits and PIT

Arithmetic circuits:

- indeterminates and constants as input,
- gates $+$, $\times$ (fan-in 2)

$\rightarrow$ computes a polynomial

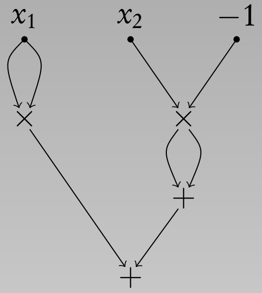


Arithmetic circuits and PIT

Arithmetic circuits:

- indeterminates and constants as input,
- gates $+$, $\times$ (fan-in 2)

$\rightarrow$ computes a polynomial



Problem PIT (Polynomial Identity Test):
decide whether an arithmetic circuit computes the identically zero polynomial :

- easy polynomial-time probabilistic algorithm
- no known polynomial-time deterministic algorithm

Randomized algorithms and circuits

Theorem (Adleman 1978)

Probabilistic polynomial-time algorithms have polynomial-size circuits

Proof idea

If the error probability is small ($< 2^{-n}$), then there is a sequence of random choices that works for all inputs of size n
→ these good random choices can be "hard wired" into the circuits. ∎

Derandomization and lower bounds

1. Lower bounds imply derandomization.

Impagliazzo, Nisan, Wigderson, … (1993–1997):

Theorem

If exponential-time problems do not have polynomial-size circuits, then derandomization is possible

1. Lower bounds imply derandomization.

Impagliazzo, Nisan, Wigderson, … (1993–1997):

THEOREM

If exponential-time problems do not have polynomial-size circuits, then derandomization is possible

Proof idea

use a hard problem in EXP to produce a pseudo-random generator that fools polynomial-time algorithms

Pseudo-random generator

Pseudo-random generator:

function $f : \{0, 1\}^{\log n} \rightarrow \{0, 1\}^n$ such that

for any circuit C of size $\leq s$:

$$|\Pr_x(C(x) = 1) - \Pr_a(C(f(a)) = 1)| < 1/10.$$

Pseudo-random generator

Pseudo-random generator:

function $f : \{0,1\}^{\log n} \to \{0,1\}^n$ such that
for any circuit C of size $\leq s$:

$$|\Pr_x(C(x) = 1) - \Pr_a(C(f(a)) = 1)| < 1/10.$$

A PRG f can be used to derandomize
a probabilistic algorithm A:

- run through all $a \in \{0,1\}^{\log n}$, compute $f(a)$ and simulate A with random bits $f(a)$,
- take the majority answer of all the simulations.

Pseudo-random generator

Pseudo-random generator:

function $f : \{0,1\}^{\log n} \to \{0,1\}^n$ such that

for any circuit C of size $\leq s$:

$$|\Pr_x(C(x) = 1) - \Pr_a(C(f(a)) = 1)| < 1/10.$$

A PRG f can be used to derandomize a probabilistic algorithm A:

- run through all $a \in \{0,1\}^{\log n}$, compute $f(a)$ and simulate A with random bits $f(a)$,
- take the majority answer of all the simulations.

f needs only be computed in exponential time

since its input is of size $\log n$.

If $A \in \mathsf{EXP}$ does not have polynomial-size circuits,

it can be turned into a PRG.

Derandomization and lower bounds (2)

2. Derandomization implies lower bounds.

THEOREM (Kabanets & Impagliazzo 2004)

If derandomization is possible, then NEXP does not have polynomial-size circuits (boolean or arithmetic)

For the idea

We shall only look at a special case of derandomization:

"black-box derandomization"

Black-box derandomization

PIT: decide whether an arithmetic circuit computes the polynomial 0.

- Black-box derandomization of PIT: construct witness points, i.e. s^k points $a_1, \ldots, a_{s^k} \in \mathbb{Z}$ such that for all arithmetic circuit C of size s,

$$C(x) \equiv 0 \iff \forall i, C(a_i) = 0.$$

- Such points exist (e.g., by Schwartz-Zippel lemma).

Black-box derandomization

PIT: decide whether an arithmetic circuit computes the polynomial 0.

- Black-box derandomization of PIT: construct witness points, i.e. s^k points $a_1, \ldots, a_{s^k} \in \mathbb{Z}$ such that for all arithmetic circuit C of size s,

$$C(x) \equiv 0 \iff \forall i, C(a_i) = 0.$$

- Such points exist (e.g., by Schwartz-Zippel lemma).
- The polynomial $p(x) = \prod_i (X - a_i)$ is nonzero but vanishes on all witness points: thus it does not have circuits of size s.

Black-box derandomization

PIT: decide whether an arithmetic circuit computes the polynomial 0.

- Black-box derandomization of PIT: construct witness points, i.e. s^k points $a_1, \dots, a_{s^k} \in \mathbb{Z}$ such that for all arithmetic circuit C of size s,

$$C(x) \equiv 0 \iff \forall i, C(a_i) = 0.$$

- Such points exist (e.g., by Schwartz-Zippel lemma).
- The polynomial $p(x) = \prod_i (X - a_i)$ is nonzero but vanishes on all witness points: thus it does not have circuits of size s.
- If the witness points a_i can be constructed efficiently, this implies that the algebraic versions of P and NP differ.

Outline

Lower bound on ad-hoc polynomials

Problem:
find a polynomial that does not have circuits of size s

Lower bound on ad-hoc polynomials

Problem:
find a polynomial that does not have circuits of size s

Examples:

- $f(x_1, \ldots, x_{s+1}) = \sum_{i=1}^{s+1} x_i$
- $f(x) = x^{2^{s+1}}$

Lower bound on ad-hoc polynomials

Problem:
find a polynomial that does not have circuits of size s

Examples:

- $f(x_1, \ldots, x_{s+1}) = \sum_{i=1}^{s+1} x_i$
- $f(x) = x^{2^{s+1}}$

Not satisfactory??

Lower bound on ad-hoc polynomials

Problem:
find a polynomial that does not have circuits of size s

Examples:

- $f(x_1, \ldots, x_{s+1}) = \sum_{i=1}^{s+1} x_i$
- $f(x) = x^{2^{s+1}}$

Not satisfactory??
Trivial lower bounds: degree, number of variables, etc.

THEOREM (Schnorr, 1978)

There is a univariate polynomial of degree s^2 and coefficients in $\{0, 1\}$ that does not have arithmetic circuits of size s (with arbitrary constants)

Proof of Schnorr's result

Lemma

There exist polynomials $g_i \in \mathbb{Z}[y_1, \ldots, y_{s^2}]$ such that if $p(x) = \sum_{i=1}^{k} a_i x^i$ is computed by a circuit of size s, then there are $\alpha_1, \ldots, \alpha_{s^2} \in \mathbb{C}$ satisfying $a_i = g_i(\alpha_1, \ldots, \alpha_{s^2})$.

Proof idea

The coefficients of the polynomial p can be expressed as a function of the "parameters" of the circuit (the constants and the type of gates). ∎

Proof of Schnorr's result (2)

Corollary

There is a nonzero polynomial $h \in \mathbb{Z}[y_1, \ldots, y_{s^2}]$ of degree $\leq s^3$ such that if $p(x) = \sum_{i=1}^{s^2} a_i x^i$ is computed by a circuit of size s, then $h(a_1, \ldots, a_{s^2}) = 0$.

Proof idea

Take a polynomial h such that $h(g_1, \ldots, g_k) \equiv 0$. $\square$

Proof of Schnorr's result (2)

Corollary

There is a nonzero polynomial $h \in \mathbb{Z}[y_1, \ldots, y_{s^2}]$ of degree $\leq s^3$ such that if $p(x) = \sum_{i=1}^{s^2} a_i x^i$ is computed by a circuit of size s, then $h(a_1, \ldots, a_{s^2}) = 0$.

Proof idea

Take a polynomial h such that $h(g_1, \ldots, g_k) \equiv 0$. ∎

Corollary

If $a_1, \ldots, a_{s^2} \in \mathbb{Z}$ are not roots of h, then $p(x) = \sum_{i=1}^{s^2} a_i x^i$ does not have circuits of size s.

Question: how efficiently can we construct a_i?

Outline

Permanent

- Deciding the existence of a perfect matching in a bipartite graph: polynomial-time algorithm
- Counting the number of perfect matchings in a bipartite graph: hard!
- Permanent: counts the number of perfect matchings in a bipartite graph

$$\operatorname{per}(x_{1,1}, \ldots, x_{n,n}) = \sum_{\sigma \in S_n} x_{1,\sigma(1)} \cdots x_{n,\sigma(n)}$$

- Determinant: same with signature of the permutation – easy to compute! (Gaussian elimination)
- Permanent of 0-1-matrices: #P-complete (Valiant 1979) (#P: counting class — big class: well above NP)

Small depth circuits

Depth: length of the longest path from a leaf to the root
("parallel" time complexity)

Circuits of restricted depth: arbitrary fan-in allowed.

Small depth circuits

THEOREM

The permanent cannot be computed by uniform polynomial-size circuits of depth $o(\log \log n)$ (boolean or arithmetic).

Proof idea: if the permanent had uniform polynomial-size circuits of depth $o(\log \log n)$ then:

1. the value of a circuit of size s and depth d could be computed in time $(\log s)^{2^d}$;
2. every language in EXP would have uniform circuits of size $2^{n^{O(1)}}$ and depth $o(\log n)$.

Small depth circuits

Theorem

The permanent cannot be computed by uniform polynomial-size circuits of depth $o(\log \log n)$ (boolean or arithmetic).

Proof idea: if the permanent had uniform polynomial-size circuits of depth $o(\log \log n)$ then:

1. the value of a circuit of size s and depth d could be computed in time $(\log s)^{2^d}$;
2. every language in EXP would have uniform circuits of size $2^{n^{O(1)}}$ and depth $o(\log n)$.

Hence: every language in EXP would have subexponential-time algorithms,
→ a contradiction with the time hierarchy theorem. ■

Conclusion

- Proving lower bounds: central (and challenging!) task
- Importance of nonuniform lower bounds

Conclusion

- Proving lower bounds: central (and challenging!) task
- Importance of nonuniform lower bounds

- Interconnected Boolean and algebraic worlds
- Many links with other domains (e.g. derandomization)

Conclusion

- Proving lower bounds: central (and challenging!) task
- Importance of nonuniform lower bounds

- Interconnected Boolean and algebraic worlds
- Many links with other domains (e.g. derandomization)

- Study of restricted circuits: small depth, uniform, no negations, multilinear…