Computational complexity

lecture 7

Complete problems

<u>Previous lecture</u>

NP – SAT, Hamiltonian cycle, clique, subset sum, dominating set, ...

P – HORNSAT

polyL – no complete problems

L – almost every language is complete

NL – reachability in directed graphs

<u>Now</u>

PSPACE - QBF

PSPACE-completeness of QBF

QBF problem

input: boolean formula $\phi(x_1,\ldots,x_n)$ with variables $x_1,\ldots,x_n$

question: is the following sentence true:

$$\exists x_1 \forall x_2 \exists x_3 \forall x_4 \ldots \phi(x_1,\ldots,x_n)$$

Theorem

The QBF problem is **PSPACE**-complete.

(the problem remains **PSPACE**-complete even if we require that ϕ is in the CNF)

PSPACE-completeness of QBF

QBF problem

input: boolean formula $\phi(x_1,...,x_n)$ with variables $x_1,...,x_n$

question: is the following sentence true:

$$\exists x_1 \forall x_2 \exists x_3 \forall x_4 \ldots \phi(x_1,...,x_n)$$

Theorem

The QBF problem is **PSPACE**-complete.

(the problem remains **PSPACE**-complete even if we require that ϕ is in the CNF)

Proof

QBF is in **PSPACE**: we browse all possible valuations in lexico-graphic order... (backtracking)

for a fixed valuation, obviously we can compute the value of ϕ in **PSPACE**

PSPACE-completeness of QBF

Theorem

The QBF problem ($\exists x_1 \forall x_2 \exists x_3 \forall x_4 \ldots \phi(x_1,\ldots,x_n)$) is **PSPACE**-complete.

Proof (**PSPACE**-hardness)

- A similar trick as in the Savitch theorem.
- Let L be a language recognized by a machine M working in polynomial space
- having an input word w of length n, we want to construct a formula

PSPACE-completeness of QBF

Theorem

The QBF problem ($\exists x_1 \forall x_2 \exists x_3 \forall x_4 \ldots \phi(x_1,\ldots,x_n)$) is **PSPACE**-complete.

Proof (**PSPACE**-hardness)

- A similar trick as in the Savitch theorem.
- Let L be a language recognized by a machine M working in polynomial space
- having an input word w of length n, we want to construct a formula
- configurations of M can be encoded in $p(n)$ bits, for some polynomial p
- for every i we will write a formula $\psi_i(x_1,\ldots,x_{p(n)},y_1,\ldots,y_{p(n)})$ saying that from the configuration $x_1,\ldots,x_{p(n)}$ it is possible to reach the configu-ration $y_1,\ldots,y_{p(n)}$ in at most 2^i steps of M
- at the very end, it is enough to check whether the formula $\psi_{p(n)}(x_1,\ldots,x_{p(n)},y_1,\ldots,y_{p(n)})$ is true, where $x_1,\ldots,x_{p(n)}$ encodes the initial configuration, and $y_1,\ldots,y_{p(n)}$ encodes the accepting configuration (we can assume that it is fixed, or we can add some existential quantification)

PSPACE-completeness of QBF

Theorem

The QBF problem ($\exists x_1 \forall x_2 \exists x_3 \forall x_4 \ldots \phi(x_1,\ldots,x_n)$) is **PSPACE**-complete.

Proof (**PSPACE**-hardness)

- for every i we want to write a formula $\psi_i(x_1,\ldots,x_{p(n)},y_1,\ldots,y_{p(n)})$ saying that from the configuration $x_1,\ldots,x_{p(n)}$ it is possible to reach the configuration $y_1,\ldots,y_{p(n)}$ in at most 2^i steps of M
- For $i=0$, either the configurations are equal, or M performs a single step between them – this can be easily written using a formula (as while proving that SAT is **NP**-hard)
- The formula can be easily generated in logarithmic space

PSPACE-completeness of QBF

Theorem

The QBF problem ($\exists x_1 \forall x_2 \exists x_3 \forall x_4 \ldots \phi(x_1,\ldots,x_n)$) is **PSPACE**-complete.

Proof (**PSPACE**-hardness)

- for every i we want to write a formula $\psi_i(x_1,\ldots,x_{p(n)},y_1,\ldots,y_{p(n)})$ saying that from the configuration $x_1,\ldots,x_{p(n)}$ it is possible to reach the configuration $y_1,\ldots,y_{p(n)}$ in at most 2^i steps of M
- For $i=0$, either the configurations are equal, or M performs a single step between them – this can be easily written using a formula (as while proving that SAT is **NP**-hard)
- The formula can be easily generated in logarithmic space
- A naive idea for $i>0$: $\psi_{i+1}(x,y)=\exists z.(\psi_i(x,z)\wedge\psi_i(z,y))$
- This does not work, since the formula grows exponentially

PSPACE-completeness of QBF

Theorem

The QBF problem ($\exists x_1 \forall x_2 \exists x_3 \forall x_4 \ldots \phi(x_1,\ldots,x_n)$) is **PSPACE**-complete.

Proof (**PSPACE**-hardness)

- for every i we want to write a formula $\psi_i(x_1,\ldots,x_{p(n)},y_1,\ldots,y_{p(n)})$ saying that from the configuration $x_1,\ldots,x_{p(n)}$ it is possible to reach the configuration $y_1,\ldots,y_{p(n)}$ in at most 2^i steps of M
- For $i=0$, either the configurations are equal, or M performs a single step between them – this can be easily written using a formula (as while proving that SAT is **NP**-hard)
- The formula can be easily generated in logarithmic space
- A naive idea for $i>0$: $\psi_{i+1}(x,y)=\exists z.(\psi_i(x,z)\wedge\psi_i(z,y))$
- This does not work, since the formula grows exponentially
- One has to use ψ_i only once:

$$\psi_{i+1}(x,y)=\exists z.\forall r.\forall t.((r=x\wedge t=z)\vee(r=z\wedge t=y)\rightarrow\psi_i(r,t))$$

- This is not in QBF, but quantifiers from ψ_i can be moved to the front of the formula (assuming that variable names are unique)

PSPACE-completeness of QBF

The QBF problem ($\exists x_1 \forall x_2 \exists x_3 \forall x_4 \ldots \phi(x_1,\ldots,x_n)$) is **PSPACE**-complete.

<u>Proof</u> (**PSPACE**-hardness)

- for every i we want to write a formula $\psi_i(x_1,\ldots,x_{p(n)},y_1,\ldots,y_{p(n)})$ saying that from the configuration $x_1,\ldots,x_{p(n)}$ it is possible to reach the configuration $y_1,\ldots,y_{p(n)}$ in at most 2^i steps of M
- For $i=0$, either the configurations are equal, or M performs a single step between them – this can be easily written using a formula (as while proving that SAT is **NP**-hard)
- The formula can be easily generated in logarithmic space
- One has to use ψ_i only once:

$$\psi_{i+1}(x,y)=\exists z.\forall r.\forall t.((r=x \wedge t=z) \vee (r=z \wedge t=y) \rightarrow \psi_i(r,t))$$

- This is not in QBF, but quantifiers from ψ_i can be moved to the front of the formula (assuming that variable names are unique)
- Again, this can be easily created in logarithmic space: first comparisons of appropriate variables, then ψ_0
- Remark: for **PSPACE** one usually relaxes the definition of hardness, and allows for reductions in **P** (instead of "in **L**")

Plan for the nearest future

- **NL=coNL**
- existence of **NP**-intermediate problems
- difficult problems that are not **NP**-hard
- relativisation and the Baker-Gill-Solovay theorem
- decision problems vs search problems
- polynomial hierarchy
- alternating machines
- probabilistic machines

It is enough to solve a complete problem

Fact

If a C-complete problem is in class D (and D is closed under composition with functions computable in **L**), then $C \subseteq D$

Proof – obvious

Corollary:

If reachability in directed graphs is in **coNL**, then **NL**=**coNL**

If SAT is in **P**, then **P**=**NP**

etc.

NL=coNL

Theorem Immerman-Szelepcseny (1987)
Unreachability in directed graphs is in **NL**.

Thus **NL**=**coNL**, since reachability in directed graphs is **NL**-complete.

Remark
Reachability in undirected graphs is in **L** (Reingold, 2004)
(this is a rather difficult theorem)

NL=coNL

Theorem Immerman-Szelepcseny (1987)

Unreachability in directed graphs is in **NL**.

Proof

- Idea: in **NL** we can not only check reachability, but also count reachable nodes

NL=coNL (∗)

Theorem Immerman-Szelepcseny (1987)

Unreachability in directed graphs is in **NL**.

Proof

- Idea: in **NL** we can not only check reachability, but also count reachable nodes
- First consider such an algorithm in **NL**: given two numbers k and q, output q different nodes reachable from node s in $\leq k$ steps, and accept (if there are less such nodes, reject)
- Solution: a loop – set a counter to 0, then for every node v in the graph, nondeterministically: either ignore v, or guess a path of length $\leq k$ from s to v, output v, and increase the counter

NL=coNL (∗)

<u>Theorem</u> Immerman-Szelepcseny (1987)

Unreachability in directed graphs is in **NL**.

<u>Proof</u>

- We can: given k and q, output q different nodes reachable from s in $\leq k$ steps, and accept (if there are less such nodes, reject)
- Main trick: using this algorithm, we will compute (by induction) q_k – a number of nodes reachable from s in $\leq k$ steps
- $q_0=1$
- Given q_k we compute q_{k+1} as follows:
  - ➔ set q_{k+1} to 1 (we include s)
  - ➔ for every other node v, output q_k nodes reachable in $\leq k$ steps from s; if among them there is a node u such that (u,v) is an edge, then increase q_{k+1} (we do not store the whole list of q_k nodes; we rather check the condition on-the-fly)
- It is now easy to finish: compute q_n, output all q_n nodes reachable in $\leq n$ steps, and check that the target node does not appear

NL=coNL

Question: why cannot we prove in a similar way that **NP=coNP**?
E.g., that SAT is in **coNP**?

NL=coNL

Question: why cannot we prove in a similar way that **NP=coNP**? E.g., that SAT is in **coNP**?

- The proof is based on counting: in **NL** we can not only check reachability, but also count (and enumerate) reachable nodes.
- However, in polynomial time, even nondeterministically, we cannot count all valuations satisfying a given formula – there are exponentially many of them, so if we would like to count them “one-by-one”, polynomial time is not enough.

NL=coNL

Corollary from the Immerman-Szelepcseny theorem:
for every space-constructible function $S(n) \geq log(n)$
NSPACE($S(n)$)=**coNSPACE**($S(n)$)

Proof: on tutorials
We use a technique called *padding*

Ladner's theorem

Theorem (Ladner, 1975) – existence of NP-intermediate problems:

If **P≠NP**, then there is a problem, which is in **NP\P**, but is not **NP**-hard with respect to polynomial-time reductions (so even more with respect to logarithmic-space reductions).

Ladner's theorem

Theorem (Ladner, 1975) – existence of NP-intermediate problems:

If **P≠NP**, then there is a problem, which is in **NP\P**, but is not **NP**-hard with respect to polynomial-time reductions (so even more with respect to logarithmic-space reductions).

Proof:

Supposing that SAT∉**P** we will give a language L∈**NP** such that:

- L is not in **P**, and
- SAT does not reduce to L in polynomial time

Ladner's theorem

Theorem (Ladner, 1975) – existence of NP-intermediate problems:

If **P≠NP**, then there is a problem, which is in **NP\P**, but is not **NP**-hard with respect to polynomial-time reductions (so even more with respect to logarithmic-space reductions).

Proof:

Supposing that SAT∉**P** we will give a language L∈**NP** such that:

- L is not in **P**, and
- SAT does not reduce to L in polynomial time

We create L as a variant of SAT with an appropriate amount of padding. In general, with padding we can change a problem into a simpler one. We want to add enough padding so that the SAT problem stops to be **NP**-complete, but not too much, so that still it is not in **P**.

The definition will be:

$$L=\{w01^{f(|w|)} : w\in \mathrm{SAT}\}$$

for an appropriate function f

Ladner's theorem (∗)

$L=\{w01^{f(|w|)} : w \in \text{SAT}\}$ for an appropriate function f.

We now define f

- Fix a computable enumeration $M_1, M_2, M_3, \ldots$ of Turing machines, such that M_i works in time $O(n^i)$, and every language in **P** is recognized by some M_i
- To this end, we take a list $M'_1, M'_2, M'_3, \ldots$ on which every Turing machine appears infinitely often. To M'_i we add a counter, which stops the machine after n^i steps – this results in M_i

Ladner's theorem (∗)

$L=\{w01^{f(|w|)} : w \in \text{SAT}\}$ for an appropriate function f.

We now define f

- Fix a computable enumeration $M_1,M_2,M_3,\ldots$ of Turing machines, such that M_i works in time $O(n^i)$, and every language in **P** is recognized by some M_i

The function f is defined by the following algorithm:

(a) take $i=1$, $n=1$

(b) put $f(n)=n^i$

(c) if there is a word v of length $\leq log(n)$ such that M_i incorrectly recognizes whether v belongs to L, then increase i by 1

(d) increase n by 1, go back to (b)

Ladner's theorem (∗)

M_i works in time $O(n^i)$, every lang. in **P** is recognized by some M_i

$L=\{w01^{f(|w|)} : w\in \text{SAT}\}$ for f defined by:

(a) take $i=1$, $n=1$

(b) put $f(n)=n^i$

(c) if there is a word v of length $\leq log(n)$ such that M_i incorrectly recognizes whether v belongs to L, then increase i by 1

(d) increase n by 1, go back to (b)

Fact 1: It can be checked in polynomial time whether a word is of the proper form (i.e., if the number of ones is appropriate).

- In order to compute $f(n)$ we repeat the loop n times, in every repetition we check polynomially many words v (of logarithmic length)
- On every word v we run M_i, which works in time $O(log^i n)$
- We can spend this time, as the input should have length $\geq f(n)\geq n^i$ (we interrupt the loop as soon as there are not enough ones)
- Remark: i is not a constant (time $O(log^i n)$ by itself is not polynomial)
- Remark 2: the simulation time depends on $|M_i|$, but $|M_i|=|i|=log(i)\leq log(n)$, so this is OK

Ladner's theorem (∗)

M_i works in time $O(n^i)$, every lang. in **P** is recognized by some M_i

$L=\{w01^{f(|w|)} : w \in \text{SAT}\}$ for f defined by:

(a) take $i=1$, $n=1$

(b) put $f(n)=n^i$

(c) if there is a word v of length $\leq log(n)$ such that M_i incorrectly recognizes whether v belongs to L, then increase i by 1

(d) increase n by 1, go back to (b)

Fact 1: It can be checked in polynomial time whether a word is of the proper form (i.e., if the number of ones is appropriate).

- In order to compute $f(n)$ we repeat the loop n times, in every repetition we check polynomially many words v (of logarithmic length)
- On every word v we run M_i, which works in time $O(log^i n)$
- We can spend this time, as the input should have length $\geq f(n) \geq n^i$ (we interrupt the loop as soon as there are not enough ones)
- We also need to check whether $v \in L$ (where $|v| \leq log\ n$)
  - ➔ we check the number of ones in v by the induction assumption
  - ➔ we check whether prefix∈SAT in time exponential in $log(n)$

Ladner's theorem (∗)

M_i works in time $O(n^i)$, every lang. in **P** is recognized by some M_i

$L=\{w01^{f(|w|)} : w\in$ SAT$\}$ for f defined by:

(a) take $i=1$, $n=1$

(b) put $f(n)=n^i$

(c) if there is a word v of length $\leq log(n)$ such that M_i incorrectly recognizes whether v belongs to L, then increase i by 1

(d) increase n by 1, go back to (b)

Fact 1: It can be checked in polynomial time whether a word is of the proper form (i.e., if the number of ones is appropriate).

Corollary: $L\in$ **NP**

Ladner's theorem (∗)

M_i works in time $O(n^i)$, every lang. in **P** is recognized by some M_i

$L=\{w01^{f(|w|)} : w \in \text{SAT}\}$ for f defined by:

(a) take $i=1$, $n=1$

(b) put $f(n)=n^i$

(c) if there is a word v of length $\leq log(n)$ such that M_i incorrectly recognizes whether v belongs to L, then increase i by 1

(d) increase n by 1, go back to (b)

Fact 2: if SAT$\notin$**P** then $L\notin$**P**

- If $L\in$**P**, then some M_i recognizes L, so from some moment on (i.e. for $n \geq n_0$ for some n_0) we have that $f(n)=n^i$
- Then it is easy to solve SAT in **P** (a contradiction):
  - ➔ if $|w| \geq n_0$ we append $|w|^i$ ones at the end, and we start M_i
  - ➔ for w shorter than n_0 the results can be hardcoded
- BTW, we have shown that f is unbounded (it is also nondecreasing)

Ladner's theorem (∗)

M_i works in time $O(n^i)$, every lang. in **P** is recognized by some M_i

$L=\{w01^{f(|w|)} : w\in \text{SAT}\}$ for an appropriate f.

<u>Fact 3</u>: if SAT$\notin$**P** then L is not **NP**-hard

- Suppose that SAT reduces to L through a function g computable in time n^k. We will show a polynomial algorithm for SAT.

Ladner's theorem (∗)

M_i works in time $O(n^i)$, every lang. in **P** is recognized by some M_i

$L=\{w01^{f(|w|)} : w \in \text{SAT}\}$ for an appropriate f.

<u>Fact 3</u>: if SAT$\notin$**P** then L is not **NP**-hard

- Suppose that SAT reduces to L through a function g computable in time n^k. We will show a polynomial algorithm for SAT.
- We know that there is n_0 such that for $n \geq n_0$ it holds that $f(n)>n^k$
- For formulas w shorter than n_0 the results can be hardcoded

Ladner's theorem (∗)

M_i works in time $O(n^i)$, every lang. in **P** is recognized by some M_i

$L=\{w01^{f(|w|)} : w\in\text{SAT}\}$ for an appropriate f.

<u>Fact 3</u>: if SAT$\notin$**P** then L is not **NP**-hard

- Suppose that SAT reduces to L through a function g computable in time n^k. We will show a polynomial algorithm for SAT.
- We know that there is n_0 such that for $n\geq n_0$ it holds that $f(n)>n^k$
- For formulas w shorter than n_0 the results can be hardcoded
- For $|w|\geq n_0$ we consider the word $g(w)$; it has length $\leq|w|^k$.

  If $g(w)$ is not of the form $w'01^{f(|w'|)}$, then it is not in L, we reject (by fact 1, this can be checked in **P**). Otherwise $w\in\text{SAT} \Leftrightarrow w'\in\text{SAT}$

Ladner's theorem (∗)

M_i works in time $O(n^i)$, every lang. in **P** is recognized by some M_i

$L=\{w01^{f(|w|)} : w\in\text{SAT}\}$ for an appropriate f.

<u>Fact 3</u>: if SAT$\notin$**P** then L is not **NP**-hard

- Suppose that SAT reduces to L through a function g computable in time n^k. We will show a polynomial algorithm for SAT.
- We know that there is n_0 such that for $n\geq n_0$ it holds that $f(n)>n^k$
- For formulas w shorter than n_0 the results can be hardcoded
- For $|w|\geq n_0$ we consider the word $g(w)$; it has length $\leq|w|^k$.

  If $g(w)$ is not of the form $w'01^{f(|w'|)}$, then it is not in L, we reject (by fact 1, this can be checked in **P**). Otherwise $w\in\text{SAT} \Leftrightarrow w'\in\text{SAT}$

  Moreover, either $|w'|<n_0$, or we have that $|w|^k\geq|g(w)|>f(|w'|)>|w'|^k$, thus the new formula is shorter at least by 1.
- We repeat this in a loop; after a linear number of steps the input length decreases below n_0, and we obtain a result.

Ladner's theorem

We have thus proved:

Theorem (Ladner 1975)

If **P**≠**NP**, then there is a problem, which is in **NP\P**, but is not **NP**-hard with respect to polynomial-time reductions (so even more with respect to logarithmic-space reductions).

CSP problems and the dichotomy conjecture

The CSP problem

<u>Input</u>: variables $x_1,\ldots,x_n$, domains $D_1,\ldots,D_n$, constraints $C_1,\ldots,C_m$ of the form (t,R), where t is a tuple of k variables, and R is a k-ary relation

<u>Question</u>: are there $x_1 \in D_1,\ldots,x_n \in D_n$ satisfying $C_1,\ldots,C_m$?

(a constraint (t,R) is satisfied if the tuple of variables t belong to the relation R)

Clearly CSP∈**NP**

Most natural **NP**-complete problems can be easily reduced to CSP (written as CSP).

CSP problems and the dichotomy conjecture

The CSP problem

Input: variables $x_1,\ldots,x_n$, domains $D_1,\ldots,D_n$, constraints $C_1,\ldots,C_m$ of the form (t,R), where t is a tuple of k variables, and R is a k-ary relation

Question: are there $x_1 \in D_1,\ldots,x_n \in D_n$ satisfying $C_1,\ldots,C_m$?

(a constraint (t,R) is satisfied if the tuple of variables t belong to the relation R)

Clearly CSP∈**NP**

Most natural **NP**-complete problems can be easily reduced to CSP (written as CSP).

Problem CSP(Γ) – like CSP, but only relations from a set Γ can be used

Conjecture: for every set Γ we either have CSP(Γ)∈**P**, or CSP(Γ) is **NP**-complete

Berman's theorem

Is it the case that every problem not in **NP** is **NP**-hard?

Intuitively, **NP**-hard means hardest in **NP**, or even harder (so problems harder than **NP** should be **NP**-hard).

Berman's theorem

Is it the case that every problem not in **NP** is **NP**-hard?

Intuitively, **NP**-hard means hardest in **NP**, or even harder
(so problems harder than **NP** should be **NP**-hard).

But the definition is: L is **NP**-hard if we can reduce every problem from **NP** to L.
So: can we reduce every problem from **NP**, to every (more difficult) problem not in **NP**?

Berman's theorem

Is it the case that every problem not in **NP** is **NP**-hard?

Intuitively, **NP**-hard means hardest in **NP**, or even harder
(so problems harder than **NP** should be **NP**-hard).

But the definition is: L is **NP**-hard if we can reduce every problem from **NP** to L.
So: can we reduce every problem from **NP**, to every (more difficult) problem not in **NP**?

The answer is **no** – we have the following theorem:

<u>Theorem</u> (Berman 1978)
If **P≠NP**, then no language over a single-letter alphabet is **NP**-hard wrt. polynomial-time reductions (so even more wrt. logarithmic-space reductions).

Berman's theorem

Is it the case that every problem not in **NP** is **NP**-hard?

No – we have the following theorem:

Theorem (Berman 1978)
If **P≠NP**, then no language over a single-letter alphabet is **NP**-hard.

Notice that there is a language language over a single-letter alphabet that requires doubly-exponential running time (i.e., surely is not in **NP**): take any language L over $\{0,1\}$ requiring triple-exponential running time, and take $\{1^{|1w|_2} : w \in L\}$, where $|1w|_2$ is the number encoded in binary as $1w$.

There is also an undecidable language over a single-letter alphabet: $\{1^k : M_k$ halts on empty input$\}$

These languages are not **NP**-hard, and not in **NP** (assuming **P≠NP**).

Berman's theorem (∗)

Theorem (Berman 1978)

If **P≠NP**, then no language over a single-letter alphabet is **NP**-hard.

Proof

Next week...