CS 341: Algorithms

Lecture 20: Reductions, P, NP, co-NP

Éric Schost

based on lecture notes by many other CS341 instructors

David R. Cheriton School of Computer Science, University of Waterloo

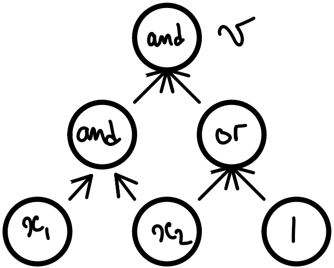
Fall 2024

More examples of Karp reductions

Circuit satisfiability

CircuitSAT.

- instance: a **circuit** = DAG with labels on the vertices
- inputs labelled by boolean variables $x_1, \ldots, x_n$ or $0, 1$
- internal vertices labelled by **and**, **or**, **not**
- there is a marked vertex v for the output
- **problem:** is there a choice of boolean x_i that makes v **true**?



k-terms conjonctive formula satisfiability

kSAT.

- instance: a **boolean formula** in n variables $x_1, \dots, x_n$ in **CNF**

$$(y_{\mathbf{1,1}} \vee \cdots \vee y_{\mathbf{1,k_1}}) \quad \wedge \quad \cdots \quad \wedge \quad (y_{\boldsymbol{\ell},\mathbf{1}} \vee \cdots \vee y_{\boldsymbol{\ell},\boldsymbol{k_\ell}})$$

  with literals $y_{i,j}$ of the form x_m, $\overline{x_m}$, 1 or 0 and $k_i \leq k$
- **problem:** is there a choice of the variables that makes it true?

Remark 1: in clause i, can have repeated $y_{i,j}$ (then we only write them once)

$$(\overline{z} \vee x) \wedge (\overline{z} \vee y) \wedge (z \vee \overline{x} \vee \overline{y}) \qquad\qquad k = 3$$

Remark 2: can assume there are no constants 1 or 0

- if $y_{i,j} = 0$, remove the literal, if $y_{i,j} = 1$ remove the clause

Remark 3: key cases are $\boldsymbol{k = 2}$ and $\boldsymbol{k = 3}$

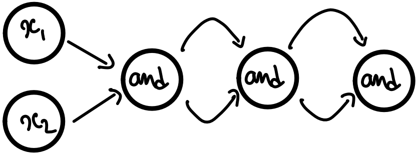
CircuitSAT $\leq_P$ 3SAT

Reduction:

- **given:** circuit $\boldsymbol{C}$ with $\boldsymbol{s}$ gates, variables $x_1, \ldots, x_n$, output v
- **build:** 3-CNF formula $\boldsymbol{F}$ with $\boldsymbol{O(s)}$ clauses
- **ensure:** C satisfiable $\iff$ F satisfiable

Remark:

- easy to build a formula: do it for all vertices bottom-up
- not polynomial, not 3CNF



$$((x_1 \wedge x_2) \wedge (x_1 \wedge x_2)) \wedge ((x_1 \wedge x_2) \wedge (x_1 \wedge x_2))$$

CircuitSAT $\leq_P$ 3SAT

Reduction:

- **given:** circuit $\boldsymbol{C}$ with $\boldsymbol{s}$ gates, variables $x_1, \ldots, x_n$, output v
- **build:** 3-CNF formula $\boldsymbol{F}$ with $\boldsymbol{O(s)}$ clauses
- **ensure:** C satisfiable $\iff$ F satisfiable

Key idea: introduce one new variable y_i per non-input gate and use

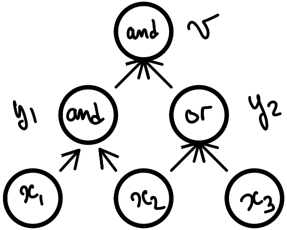
$$\boldsymbol{y_i = z \iff (z \implies y_i) \wedge (y_i \implies z) \iff (y_i \vee \overline{z}) \wedge (\overline{y_i} \vee z)}$$

- **and gate:** $z = t \wedge u$, and so $\overline{z} = \overline{t} \vee \overline{u}$

$$(y_i \vee \overline{t} \vee \overline{u}) \wedge (\overline{y_i} \vee (t \wedge u)) = \boldsymbol{(y_i \vee \overline{t} \vee \overline{u}) \wedge (\overline{y_i} \vee t) \wedge (\overline{y_i} \vee u)}$$

- **or gate:** $z = t \vee u$ gives $\boldsymbol{(y_i \vee \overline{t}) \wedge (y_i \vee \overline{u}) \wedge (\overline{y_i} \vee t \vee u)}$
- **not gate:** $z = \overline{t}$ gives $\boldsymbol{(y_i \vee t) \wedge (\overline{y_i} \vee \overline{t})}$

CircuitSAT $\leq_P$ 3SAT



gives

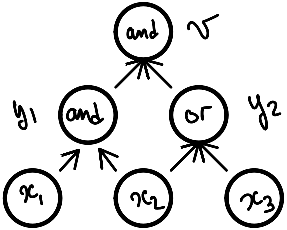
$$(y_1 = (x_1 \wedge x_2)) \ \wedge \ (y_2 = (x_2 \vee x_3)) \ \wedge \ (v = (y_1 \wedge y_2)) \ \wedge \ v$$

and

$$\begin{aligned}(y_1 \vee \overline{x_1 \wedge x_2}) \ \wedge \ (\overline{y_1} \vee (x_1 \wedge x_2)) \wedge \\ (y_2 \vee \overline{x_2 \vee x_3}) \ \wedge \ (\overline{y_2} \vee (x_2 \vee x_3)) \wedge \\ (v \vee \overline{y_1 \wedge y_2}) \ \wedge \ (\overline{v} \vee (y_1 \wedge y_2)) \ \wedge \ v\end{aligned}$$

given C, F can be constructed in polynomial time

CircuitSAT $\leq_P$ 3SAT



gives

$$(y_1 = (x_1 \wedge x_2)) \;\wedge\; (y_2 = (x_2 \vee x_3)) \;\wedge\; (v = (y_1 \wedge y_2)) \;\wedge\; v$$

and

$$\begin{aligned} F = &(y_1 \vee \overline{x_1} \vee \overline{x_2}) \;\wedge\; (\overline{y_1} \vee x_1) \;\wedge\; (\overline{y_1} \vee x_2) \;\wedge \\ &(y_2 \vee \overline{x_2}) \;\wedge\; (y_2 \vee \overline{x_3}) \;\wedge\; (\overline{y_2} \vee x_2 \vee x_3) \;\wedge \\ &(v \vee \overline{y_1} \vee \overline{y_2}) \;\wedge\; (\overline{v} \vee y_1) \;\wedge\; (\overline{v} \vee y_2) \;\wedge\; v \end{aligned}$$

given C, F can be constructed in polynomial time

Aside: polynomial-time Turing reductions

A stronger form of reduction

Consider two problems PROB1, PROB2, **not necessarily decision problems**

Definition

PROB1 is **polynomial-time Turing reducible** to PROB2 if there is an algorithm that solves PROB1 using

- a polynomial number of operations
- a polynomial number of calls to a solver (oracle) for PROB2

Remark:

- inputs/output transfers to/from the oracle count as "operations"
- so all inputs to the oracle have polynomial size

Notation:

- PROB1 $\leq_P^T$ PROB2

Examples and key property

Example 1

- reducing an optimization problem to its decision version (if optimal is an integer of polynomial size)

Example 2

- Karp reductions for decision problems (only one oracle call, at the end)

Claim

if Prob1 $\leq_P^T$ Prob2 and Prob2 can be solved in polynomial time, then it's also the case for Prob1

Proof: same as for Karp reductions

Example: factoring

Effective version: FACTOR

- **input:** integer M — input size $\Theta(\log M)$
- **output:** the prime factors of M

Decision version: HASFACTOR

- **input:** integers M and $0 \leq k \leq M$ — input size $\Theta(\log M)$
- **output:** **yes** iff M has a prime factor $\leq k$

Remark: polynomial time = $\boldsymbol{\log(M)^{O(1)}}$

Claim 1:

HASFACTOR $\leq_P^T$ FACTOR

Proof: factor M and check

Example: factoring

Claim 2:

$\textsc{Factor} \leq_P^T \textsc{HasFactor}$

1. Find the first ℓ such that M has a prime factor between 2^ℓ and $2^{\ell+1} - 1$
   - test all $\ell = 1, 2, 3, \ldots, \log(M)$ — $O(\log M)$ calls to $\textsc{HasFactor}$ with inputs $\leq M$
   - if all **no**, M is prime, done
2. Find the smallest factor between 2^ℓ and $2^{\ell+1} - 1$
   - binary search — $O(\log M)$ calls to $\textsc{HasFactor}$ with inputs $\leq M$
3. We found one prime factor P. Repeat on M/P
   - $\log M$ prime factors at most

Conclusion: if $\textsc{HasFactor}$ can be solved in polynomial time, we can factor integers in polynomial time.

P, NP, co-NP

The classes P and NP

Definition

P is the set of decision problems that can be solved in polynomial time
NP is the set of decision problems where **yes**-instances can be **certified** in polynomial time.

Precisely, a **decision problem** PROB is in **NP** if

- there exists an algorithm B (**a certifier**) that takes as input an instance x and an extra input y (**a certificate**) and outputs "yes" or "no" in polynomial time in size(x)+size(y)
- x **yes**-instance for PROB if and only if there exists y of size polynomial in size(x), such that $B(x, y) =$"yes"

Remarks

1. if we can solve Prob in polynomial time, we can certify it as well (with an empty certificate) so

$$\mathbf{P} \subset \mathbf{NP}$$

\$1,000,000 question: $\mathbf{P} = \mathbf{NP}$?

2. **NP** means **N**on-deterministic **P**olynomial time

- nothing to do with randomized algorithms
- non-deterministic Turing machines have several transitions available each step
- existence of one accepting path $\simeq$ existence of a certificate

Examples

Independent set

- **instance**: graph G, integer K
- **certificate:** a set S of vertices
- **certification:** test if $|S| \geq K$ and S independent

Vertex cover

- **instance**: graph G, integer K
- **certificate:** a set S of vertices
- **certification:** test if $|S| \leq K$ and S covers all edges

Clique

- **instance**: graph G, integer K
- **certificate:** a set S of vertices
- **certification:** test if $|S| \geq K$ and S clique

Examples

Circuit sat

- **instance**: boolean circuit C
- **certificate:** a sequence x of bits
- **certification:** test if $C(x) = \mathsf{true}$

3SAT

- **instance**: a boolean formula F in 3CNF
- **certificate:** a sequence x of bits
- **certification:** test if $F(x)$ is true

SAT

- **instance**: a boolean formula F
- **certificate:** a sequence x of bits
- **certification:** test if $F(x)$ is true

Examples

Hamiltonian cycle

- **instance**: graph G
- **certificate:** a sequence S of vertices
- **certification:** test if S is a Hamiltonian cycle in G

Hamiltonian path

- **instance**: graph G
- **certificate:** a sequence S of vertices
- **certification:** test if S is a Hamiltonian path in G

Factors

- **instance**: integers M and $0 \leq k \leq M$
- **certificate:** integer P
- **certification:** test if P is prime, P divides M and $P \leq k$

co-NP

Definition

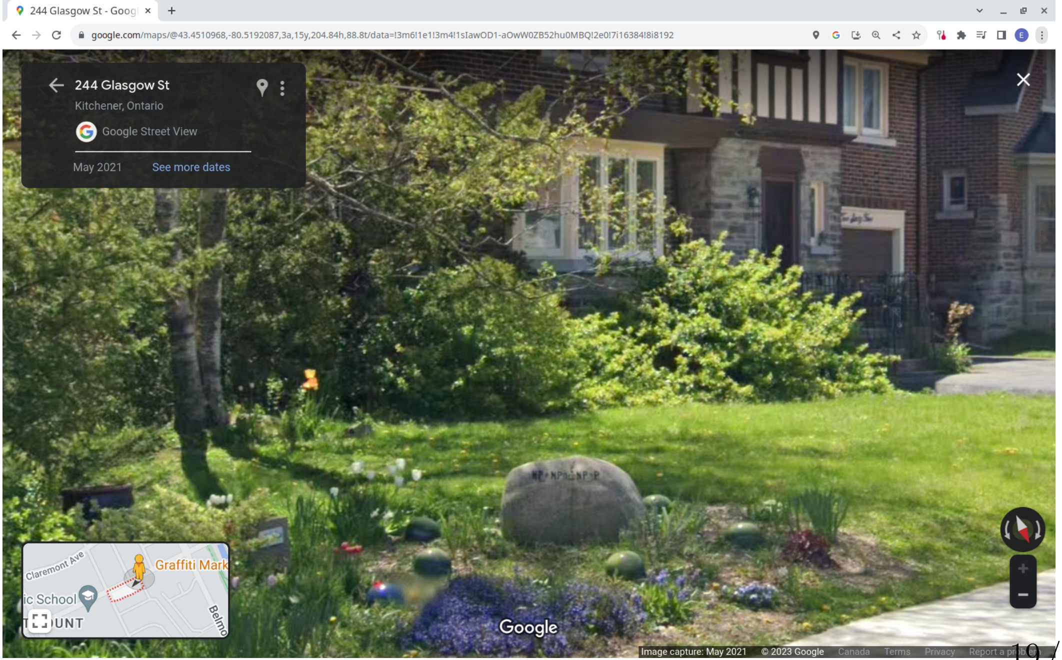
co-NP is the set of decision problems whose **no**-instances can be certified in polynomial time.

Remark: most problems so far are thought to not be in **co-NP**

- certify that a formula not satisfiable?
- certify that a graph has no Hamiltonian path?
- but HasFactor is in **co-NP** (certificate = all prime factors)

Exercise (after we see NP-completeness)

If a single NP-complete problem is in **co-NP**, **NP=co-NP**
(so doubtful that HasFactor is NP-complete)


244 Glasgow St - Googl
google.com/maps/@43.4510968,-80.5192087,3a,15y,204.84h,88.8t/data=!3m6!1e1!3m4!1sIawOD1-aOwW0ZB52hu0MBQ!2e0!7i16384!8i8192
244 Glasgow St
Kitchener, Ontario
Google Street View
May 2021
See more dates
Claremont Ave
Graffiti Mark
ic School
UNT
Google
Image capture: May 2021
© 2023 Google
Canada
Terms
Privacy