Computational Complexity; slides 6, HT 2022 variants of NP

Prof. Paul W. Goldberg (Dept. of Computer Science, University of Oxford)

HT 2022

Pseudo-Polynomial Time

KNAPSACK can be solved in time $\mathcal{O}(n\ell)$ using dynamic programming
(recall ℓ is weight limit, n is number of items)
... but, ℓ can be exponential in the input description length!

Pseudo-Polynomial Time: Algorithms polynomial in the maximum of the input length and the value of numbers occurring in the input.

If KNAPSACK is restricted to instances with $\ell \leq p(n)$ for some polynomial p, then we obtain a problem in P.

Equivalently: KNAPSACK is in polynomial time for unary encoding of numbers.

Pseudo-Polynomial Time

KNAPSACK can be solved in time $\mathcal{O}(n\ell)$ using dynamic programming
(recall ℓ is weight limit, n is number of items)
... but, ℓ can be exponential in the input description length!

Pseudo-Polynomial Time: Algorithms polynomial in the maximum of the input length and the value of numbers occurring in the input.

If KNAPSACK is restricted to instances with $\ell \leq p(n)$ for some polynomial p, then we obtain a problem in P.

Equivalently: KNAPSACK is in polynomial time for unary encoding of numbers.

Strong NP-completeness: Problems (e.g. CLIQUE, SAT) which remain NP-complete even if all numbers are bounded by a polynomial in the input length
(equivalently, for unary encoding of numbers).

digression: dealing with NP-hardness

- Maybe a pseudo-polynomial time algorithm is OK
- Move from exact to approximate optimisation: it may be hard to find optimal solution, but finding one within fact 2 (say) of optimal of optimal, is in P.
- fixed-parameter tractability
- model data as noisy (e.g. in smoothed analysis)

NP and co-NP

Notation. For a language $\mathcal{L} \subseteq \Sigma^*$ let $\overline{\mathcal{L}} := \Sigma^* \setminus \mathcal{L}$ be its complement.

Definition.
If $\mathcal{C}$ is a complexity class, we define

$$\text{co-}\mathcal{C} := \{\mathcal{L} : \overline{\mathcal{L}} \in \mathcal{C}\}.$$

coNP: In particular, $\text{co-NP} := \{\mathcal{L} : \overline{\mathcal{L}} \in \text{NP}\}$

A problem belongs to co-NP, if no-instances have short certificates.

co-NP-completeness

Examples of problems in co-NP:

NO HAMILTONIAN CYCLE
Given: Graph G
Question: Is it true that G contains no Hamiltonian cycle?

TAUTOLOGY
Given: Formula φ
Question: Is φ a tautology, i.e. satisfied by all assignments?

Examples of problems in co-NP:

NO HAMILTONIAN CYCLE

Given: Graph G

Question: Is it true that G contains no Hamiltonian cycle?

TAUTOLOGY

Given: Formula φ

Question: Is φ a tautology, i.e. satisfied by all assignments?

Definition. A language $\mathcal{C} \in$ co-NP is *co-NP-complete*, if $\mathcal{L} \leq_p \mathcal{C}$ for all $\mathcal{L} \in$ co-NP.

TAUTOLOGY is co-NP-complete: any reduction from an NP problem to SAT can convert to a reduction from a co-NP problem to TAUTOLOGY

co-NP-complete decision problems are as hard as NP-complete ones.

P, NP, and co-NP

Proposition.

1. $\mathrm{P} = \text{co-P}$
2. Hence, $\mathrm{P} \subseteq \mathrm{NP} \cap \text{co-NP}$

Question:

- $\mathrm{NP} = \text{co-NP}$?

  Most people do not think so (c.f. research in propositional proof theory).

- $\mathrm{P} = \mathrm{NP} \cap \text{co-NP}$?

  Again, most people do not think so.

Later: Ladner's theorem: assuming $\mathrm{P} \neq \mathrm{NP}$, there are "NP-intermediate" problems.

Search versus decision

Problem 1: boolean formula φ — is φ satisfiable?

Problem 2: Given a formula φ, find a satisfiable assignment, or answer "no".

Problem 2 is at least as hard.
But — we can say: "problem 2 is no harder than NP"; solve problem 2 with **oracle** for problem 1.

"oracle": an imaginary black-box that supports queries to a computational problem: given an input, will (in one step) tell you correct output.

FNP, reducing search to decision

FNP: problems of computing a function that can be checked in polynomial time: find a certificate, not just answer "yes"

An FNP problem comprises a *polynomially balanced relation* R for which $R(x, y)$ can be checked in time polynomial in $|x|$, $|y|$.
Given x, search for y with $R(x, y)$.
Note: it's asking too much to solve a NP search problem X using a single decision oracle (why?).
But can solve X using multiple oracle calls to corresponding decision problem.

So, "FNP is as hard as NP"

Reducibility amongst FNP problems

Informally, $X \leq_p Y$ means: given an oracle for problem Y, can reconstruct a solution to X.

In detail, reduction needs 2 functions f, g, where f maps instances of X to instances of Y, and g maps solutions of Y to solutions of X.
If X and Y correspond with relations R_1 and R_2 respectively, want

$$(x, g(z)) \in R_1 \quad \text{iff} \quad (f(x), z) \in R_2.$$

I'll come back to this in the final lecture. Meanwhile, think about how to compare FACTORING in base-2 with FACTORING in base-10.

FSAT problem: given a boolean formula, compute a satisfying assignment.
FSAT is FNP-complete. FACTORING seems to be hard, but is unlikely to be FNP-complete!

Optimisation

"Is there a k-clique" is (in a sense) equally hard as "find a k-clique"

"What's the size of the largest clique?" is (in a sense) harder!

If we told the answer is some value k, an NP machine can verify k-clique(s) exist

Need also a co-NP machine to verify: no $k + 1$-clique exists!

NP, or co-NP alone, don't seem to be sufficient, more later.

In exercises later, will make a start at classifying problems like this

Definition: For complexity classes A and B let A^B denote problems solved by an A-machine with oracle access to B.

As a start, we can put the problem in P^{NP}