CoCoNuT - Complexity - Lecture 3

N.P. Smart

Dept of Computer Science
University of Bristol,
Merchant Venturers Building

April 11, 2016

Outline

Complementation of Languages

Search Problems

Average and Worst Case

co-C Recap

Complimentation of Languages

Let $\mathcal{L}$ be a language, we define $\overline{\mathcal{L}}$ as

$$\overline{\mathcal{L}} = \{x \in \Sigma^* : x \notin \mathcal{L}\}.$$

co Classes

If C is a class of languages then co-C is the set

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

We have $\text{P} = \text{co-P}$

NP and co-NP

Recall NP is the set of languages which have short proofs.

So co-NP is the set of languages which have short refutations.

Our earlier example FACTORING was in NP and co-NP.

It is believed that NP $\cap$ co-NP is larger than P

- e.g. FACTORING $\in$ NP $\cap$ co-NP but we suspect that FACTORING $\notin$ P.

It is unknown whether NP $=$ co-NP.

- Take SAT formulae ϕ.
- Easy to present a proof that ϕ is satisfiable.
- Hard to see how to present a proof that ϕ is not-satisfiable.

Recall Definition of NP

Go back to our witness definition of NP.

NP

$\mathcal{L} \in$ NP means $\exists V\ s.t.: x \in \mathcal{L}$ iff $\exists w \in \{0,1\}^{p(|x|)}$ s.t. $V(x,w) = 1$.

Alternative Definition of co-NP

We can define co-NP via

co-NP: Defn 1

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

or via

co-NP: Defn 2

$$\mathcal{L} \in \text{co-NP means } \exists V\ s.t.: x \in \mathcal{L} \text{ iff } \forall w \in \{0,1\}^{p(|x|)} \text{ s.t. } V(x,w) = 1.$$

We will now show that both definitions are equivalent:

Alternative Definition of co-NP

Defn 2 $\Longrightarrow$ Defn 1

Suppose $\mathcal{L}$ a language such that

$$y \in \mathcal{L} \text{ iff } \forall w \in \{0,1\}^{p(|y|)} \text{ s.t. } V(y,w) = 1.$$

Let $x \in \overline{\mathcal{L}}$

- Implies $\exists w$ such that $V(x,w) = 0$.
- Define the machine $V'(x,w) = 1 - V(x,w)$.
- Then V' accepts the language $\overline{\mathcal{L}}$ thus $\overline{\mathcal{L}} \in$ NP.
- i.e. $\mathcal{L} = \overline{\overline{\mathcal{L}}}$ satisfies Defn 1.

Alternative Definition of co-NP

Defn 1 $\implies$ **Defn 2**

Now suppose $\mathcal{L}$ is such that

$$\overline{\mathcal{L}} \in \mathrm{NP}$$

Let $x \in \mathcal{L}$

- $\exists V'$ and $\forall w$ s.t. $V'(x, w) = 0$.
  - Otherwise V' would accept a false proof.
- Now define $V(x, w) = 1 - V'(x, w)$
- Then $\forall w$ we have $V(x, w) = 1$.

The Polynomial Hierarchy

We can play games with $\exists$ and $\forall$ for a long time

The class Σ_i

$\mathcal{L} \in \Sigma_i$ if there is a verifier V such that $\exists V$ such that

$$x \in \mathcal{L} \text{ iff } \exists w_1 \in \{0,1\}^{p(|x|)}, \forall w_2 \in \{0,1\}^{p(|x|)}, \exists w_3 \in \{0,1\}^{p(|x|)},$$
$$\ldots Q_i w_i \in \{0,1\}^{p(|x|)}, \text{ s.t. } V(x, w_1, w_2, \ldots, w_i) = 1.$$

where $Q_i = \exists$ if i is odd and $Q_i = \forall$ if i is even.

The polynomial hierarchy is defined by

$$\mathrm{PH} = \cup_{i \geq 1} \Sigma_i.$$

Believed that $\Sigma_i \neq \Sigma_{i+1}$.

The Polynomial Hierarchy

Example

Given a graph G and an integer k determine if the largest clique in G has size exactly k.

- In a clique every two vertices are connected by an edge.

Call this problem LARGEST-CLIQUE

A "witness" for this problem would be

- A set of k vertices S s.t. S is a clique
- For all sets of $k+1$ vertices S' we have S' is not a clique

Thus LARGEST-CLIQUE $\in \Sigma_2$.

QSAT$_i$

Given a boolean formulae ϕ in n variables $x_1, \ldots, x_n$ which are partitioned into i sets $X_1, \ldots, X_i$.

Quantified SAT, or QSAT$_i$

QSAT$_i$ is the problem to determine whether the following statement is true.

$$\exists X_1 \forall X_2 \exists X_3 \forall X_4 \ldots Q_i X_i \phi,$$

where $Q_i = \exists$ if i is odd and $Q_i = \forall$ if i is even.

QSAT$_i$ is Σ_i-complete.

Search vs Decision

Up to now we have focused on decision problems.

This can seem a bit artificial, but it is the tradition of complexity theory.

- Nice philosophy about theorems, proofs etc
- Simplifies the discussions a lot
- Generalises to other questions
- Useful notion in cryptography.

In practice we care about search problems

- We want to find the variables which satisfy SAT
- We want to find the factors of a number N
- ...

We can build up virtually the same theory for search problems:

- See Goldreich's book for an extensive study on this.

Search Problems

Let R denote a (poly-bounded) relation

- Set of values $(x, y) \in \{0, 1\}^* \times \{0, 1\}^*$
- Such that $|y| < p(|x|)$ for some poly p.

Think of x as the problem and y the solution.

The class P_{search} is the set of relations R such that

- There is an algorithm A (depending on R)
- For all x the algorithm $A(x)$ outputs y
- Such that $(x, y) \in R$ or $y = \perp$ and there is no such pair in R.
- The algorithm A runs in poly-time

Clearly $\mathrm{P}_{search} \subset \mathrm{P}$.

- In some vague undefined sense we will not worry about

Search Problems

Just as NP denotes the class of efficiently checkable decision problems

We can define NP_{search} as the class of efficiently checkable search problems.

The class NP_{search} is the set of relations R such that

- There is an algorithm A (depending on R)
- We have $(x, y) \in R$ iff $A(x, y)$ outputs 1
- The algorithm A runs in poly-time

$P_{search} \subset NP_{search}$?

Surprisingly no...

Let $R = \{(0,0),(0,b) | b = 1 \text{ if God exists}, b = 0 \text{ otherwise}\}$.

Then R is a polynomially-bounded relation.

R is in P_{search} since

- On input 0, output 0
- On input $x \neq 0$, output fail

is a polynomial-time search algorithm for R.

But R is not in NP_{search}, assuming I cannot (dis)prove the existence of God by computer, since any algorithm that could verify whether or not $(0,1) \in R$ would be such a proof.

Search To Decision Reductions

We can relate some search problems to other decision problems.

Example:
Given a group G of prime order q and two elements g and $h = g^x$ for some (unknown) x

Write down an algorithm to find x (This is called the Discrete Log Problem)

- Suppose you are given an algorithm which on input of $(g', h') \in G$ finds the least significant bit of the discrete log y' such that $h' = g^{y'}$,
    - i.e. it decides if y is odd or even.

Search To Decision Reductions

Some search problems are believed to be much harder than their associated decision ones.

- Bit esoteric, but in Crypto one has instances for which search Diffie–Hellman is hard but decision Diffie–Hellman is easy.

Given a group G of prime order g

Computational/Search Diffie–Hellman:
Given g, g^x and g^y (but not x or y) determine g^{xy}.

Decision Diffie–Hellman:
Given g, g^x and g^y and either ($Z = g^z$ or $Z = g^{xy}$ with 50 percent probability) (but not x, y or z) determine whether $Z = g^z$ or $Z = g^{xy}$.

In general it is believed that for most groups if the DLP is hard, then these two problems are equivalent.

- But there are some groups for which this does not hold.

Search To Decision Reductions

We can relate some search problems to the associated decision problems.

Example: Consider the two problems

SEARCH-KNAPSACK:

Given $x_1, \ldots, x_n$ and an S find $b_i \in \{0, 1\}$ such that

$$\sum b_i \cdot x_i = S.$$

DECISION-KNAPSACK:

Given $x_1, \ldots, x_n$ and an S find if there exists $b_i \in \{0, 1\}$ such that

$$\sum b_i \cdot x_i = S.$$

Show that the two problems are poly-time equivalent.

Average and Worst Case

Another problem with the traditional view of complexity theory is that it only deals with worst case problems.

Consider the problem

$$\text{FACTORING} = \{\langle x, y\rangle \mid x \text{ is an integer with a prime factor lower than } y\}.$$

We know FACTORING $\in$ NP $\cap$ co-NP.

But we do not know whether FACTORING $\in$ P.

The reason FACTORING is hard is because some numbers are hard to factor!

- The worst case is hard

Average and Worst Case

But on average FACTORING is easy.

Fifty percent of the time we read in x and y, and as long as $y > 2$ we output 1.

- Since fifty percent of all random numbers x are even!

So FACTORING is easy on average.

Indeed most advanced factoring algorithms use a trick of factoring the hard number by finding lots of factors of easy numbers.

Average and Worst Case

Even NP-complete problems are not hard on average

In industry most SAT problems one encounters can be easily solved.

Graph 3-colourability is NP-complete

- In worst case colouring the nodes such that no edge connects two nodes with the same colour is very hard.
- On average (with a specific definition of what is a random graph) can solve this in constant time.
- Most graphs are not 3-colourable, and we can quickly determine they are not.

Random Self-Reductions

Are there some problems which are as bad in the worst case as they are on average?

Consider the *DDH* problem considered earlier

Suppose DDH is easy on average, i.e. given a random DDH instance $(A, B, C, D) = (g, g^x, g^y, Z)$ there is an algorithm A which will solve the DDH problem.

Now suppose we are given a hard instance (A', B', C', D'), we can turn it into a random instance by setting

$$A = A'^{r_1}, \quad B = B'^{r_1 \cdot r_2}, \quad C = C'^{r_1 \cdot r_3}, \quad D = D'^{r_1 \cdot r_2 \cdot r_3}.$$

such (A', B', C', D') is a DDH tuple iff (A, B, C, D) is a DDH tuple.

- Pick $r_1, r_2, r_3 \in \{0, \ldots, |G| - 1\}$.

Thus the worst case problem can be reduced to the average case