CoCoNuT - Complexity - Lecture 2

N.P. Smart

Dept of Computer Science
University of Bristol,
Merchant Venturers Building

March 19, 2015

Outline

Non Deterministic Turing Machines

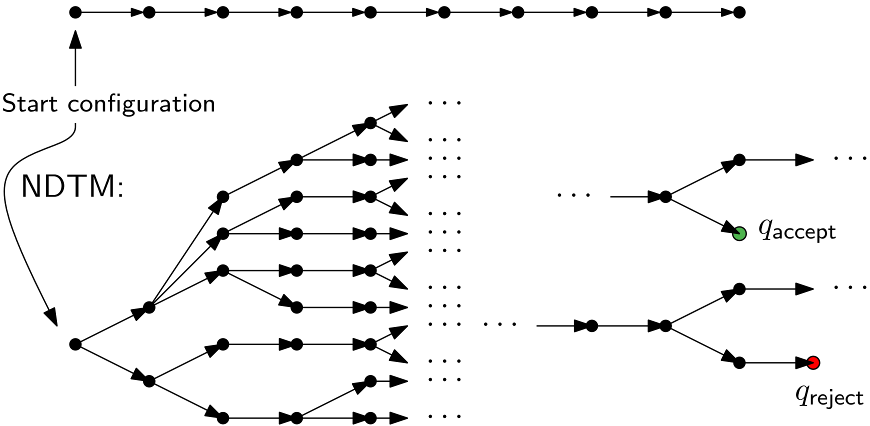
Cook-Levin Proof

More NP-complete Languages

Non-deterministic Turing machines

A node • is a configuration

DTM:



Time complexity for NTMs

Definition. Let N be a NTM which is a decider. The **running time** of N is $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that N uses *on any branch of its computation* on *any* input of length n.

Theorem. *(Time-complexity of NTM simulation.)* Every $t(n)$-time NTM N has an *equivalent* $2^{O(t(n))}$-time TM M.

(equivalent: N and M decide the same language.)

Nondeterministic polynomial-time

Definitions

Let $t : \mathbb{N} \to \mathbb{R}^+$ be a function. Then NTIME($t(n)$) is the set of all languages which are decidable by a nondeterministic Turing machine running in time $O(t(n))$.

- We say that an NDTM runs in time $O(t(n))$ if all of its computational paths halt in time $O(t(n))$.

We will now see that there is a close connection between nondeterministic Turing machines and verification of proofs.

This will explain the name NP, which stands for "Nondeterministic Polynomial-time" (and not "non-polynomial time").

Nondeterministic polynomial-time

Theorem

$$\mathrm{NP} = \bigcup_{k \geq 0} \mathrm{NTIME}(n^k).$$

Proof (sketch) We first show that $\mathcal{L} \in \mathrm{NP}$ implies that $\mathcal{L}$ is decided by an NDTM running in polynomial time (i.e. $\mathrm{NP} \subseteq \bigcup_{k \geq 0} \mathrm{NTIME}(n^k)$).

- From the definition of NP, there exists a verifier V for $\mathcal{L}$ running in time $O(n^k)$ for some fixed k.
- We define an NDTM N which behaves as follows:
  1. Nondeterministically guess a string c of length at most $O(n^k)$.
  2. Run V on input x, c. If V accepts, accept; otherwise, reject.

Nondeterministic polynomial-time

Theorem

$$\mathrm{NP} = \bigcup_{k \geq 0} \mathrm{NTIME}(n^k).$$

Proof (sketch) We first show that $\mathcal{L} \in \mathrm{NP}$ implies that $\mathcal{L}$ is decided by an NDTM running in polynomial time (i.e. $\mathrm{NP} \subseteq \bigcup_{k \geq 0} \mathrm{NTIME}(n^k)$).

- From the definition of NP, there exists a verifier V for $\mathcal{L}$ running in time $O(n^k)$ for some fixed k.
- We define an NDTM N which behaves as follows:
  1. Nondeterministically guess a string c of length at most $O(n^k)$.
  2. Run V on input x, c. If V accepts, accept; otherwise, reject.

Nondeterministic polynomial-time

Theorem

$$\mathrm{NP} = \bigcup_{k \geq 0} \mathrm{NTIME}(n^k).$$

Proof (sketch) We first show that $\mathcal{L} \in \mathrm{NP}$ implies that $\mathcal{L}$ is decided by an NDTM running in polynomial time (i.e. $\mathrm{NP} \subseteq \bigcup_{k \geq 0} \mathrm{NTIME}(n^k)$).

- From the definition of NP, there exists a verifier V for $\mathcal{L}$ running in time $O(n^k)$ for some fixed k.
- We define an NDTM N which behaves as follows:
  1. Nondeterministically guess a string c of length at most $O(n^k)$.
  2. Run V on input x, c. If V accepts, accept; otherwise, reject.

Nondeterministic polynomial-time

Theorem

$$\mathrm{NP} = \bigcup_{k \geq 0} \mathrm{NTIME}(n^k).$$

Proof (sketch) We first show that $\mathcal{L} \in \mathrm{NP}$ implies that $\mathcal{L}$ is decided by an NDTM running in polynomial time (i.e. $\mathrm{NP} \subseteq \bigcup_{k \geq 0} \mathrm{NTIME}(n^k)$).

- From the definition of NP, there exists a verifier V for $\mathcal{L}$ running in time $O(n^k)$ for some fixed k.
- We define an NDTM N which behaves as follows:
  1. Nondeterministically guess a string c of length at most $O(n^k)$.
  2. Run V on input x, c. If V accepts, accept; otherwise, reject.

Theorem

$$\mathrm{NP} = \bigcup_{k \geq 0} \mathrm{NTIME}(n^k).$$

Proof (sketch) For the other direction, we show that if $\mathcal{L}$ is decided by an NDTM running in polynomial time, $\mathcal{L} \in \mathrm{NP}$ (i.e. $\bigcup_{k \geq 0} \mathrm{NTIME}(n^k) \subseteq \mathrm{NP}$).

- From the definition of NTIME, there exists an NDTM N which decides $\mathcal{L}$ and runs in time $O(n^k)$ for some fixed k.
- We define a verifier V which takes as input a string x and a witness c. c is the description of a sequence of computational path choices made by N.
- V simulates the computation of N on input x according to c, checking whether each transition made is valid.
- If N accepts in the end, V accepts; otherwise, V rejects.

Theorem

$$\mathrm{NP} = \bigcup_{k \geq 0} \mathrm{NTIME}(n^k).$$

Proof (sketch) For the other direction, we show that if $\mathcal{L}$ is decided by an NDTM running in polynomial time, $\mathcal{L} \in \mathrm{NP}$ (i.e. $\bigcup_{k \geq 0} \mathrm{NTIME}(n^k) \subseteq \mathrm{NP}$).

- From the definition of NTIME, there exists an NDTM N which decides $\mathcal{L}$ and runs in time $O(n^k)$ for some fixed k.
- We define a verifier V which takes as input a string x and a witness c. c is the description of a sequence of computational path choices made by N.
- V simulates the computation of N on input x according to c, checking whether each transition made is valid.
- If N accepts in the end, V accepts; otherwise, V rejects.

Theorem

$$\text{NP} = \bigcup_{k \geq 0} \text{NTIME}(n^k).$$

Proof (sketch) For the other direction, we show that if $\mathcal{L}$ is decided by an NDTM running in polynomial time, $\mathcal{L} \in \text{NP}$ (i.e. $\bigcup_{k \geq 0} \text{NTIME}(n^k) \subseteq \text{NP}$).

- From the definition of NTIME, there exists an NDTM N which decides $\mathcal{L}$ and runs in time $O(n^k)$ for some fixed k.
- We define a verifier V which takes as input a string x and a witness c. c is the description of a sequence of computational path choices made by N.
- V simulates the computation of N on input x according to c, checking whether each transition made is valid.
- If N accepts in the end, V accepts; otherwise, V rejects.

Theorem

$$\mathrm{NP} = \bigcup_{k \geq 0} \mathrm{NTIME}(n^k).$$

Proof (sketch) For the other direction, we show that if $\mathcal{L}$ is decided by an NDTM running in polynomial time, $\mathcal{L} \in \mathrm{NP}$ (i.e. $\bigcup_{k \geq 0} \mathrm{NTIME}(n^k) \subseteq \mathrm{NP}$).

- From the definition of NTIME, there exists an NDTM N which decides $\mathcal{L}$ and runs in time $O(n^k)$ for some fixed k.
- We define a verifier V which takes as input a string x and a witness c. c is the description of a sequence of computational path choices made by N.
- V simulates the computation of N on input x according to c, checking whether each transition made is valid.
- If N accepts in the end, V accepts; otherwise, V rejects.

Theorem

$$\mathrm{NP} = \bigcup_{k \geq 0} \mathrm{NTIME}(n^k).$$

Proof (sketch) For the other direction, we show that if $\mathcal{L}$ is decided by an NDTM running in polynomial time, $\mathcal{L} \in \mathrm{NP}$ (i.e. $\bigcup_{k \geq 0} \mathrm{NTIME}(n^k) \subseteq \mathrm{NP}$).

- From the definition of NTIME, there exists an NDTM N which decides $\mathcal{L}$ and runs in time $O(n^k)$ for some fixed k.
- We define a verifier V which takes as input a string x and a witness c. c is the description of a sequence of computational path choices made by N.
- V simulates the computation of N on input x according to c, checking whether each transition made is valid.
- If N accepts in the end, V accepts; otherwise, V rejects.

Cook-Levin Proof

We are now going to prove the major theorem in Complexity Theory; that *SAT* is NP-complete.

Let L be an language in NP.

We need to show this can be encoded as a SAT problem.

Then if we have an algorithm which solves SAT we can use it to decide L.

Hence SAT is NP-complete

Cook-Levin

Think of NP as NDTM's

There is an execution path, i.e. a DTM execution, which contains an ACCEPT state.

Let $\mathbb{T}$ be the DTM corresponding to this path.

- The input $x_1, \ldots, x_n$ is the problem instance
- Assume to be bits.
- The alphabet is $\{\triangleright, \Box, 0, 1\}$
- The state space is Σ, which is of constant size S.
- We assume that the states $\mathrm{ACCEPT}, \mathrm{START} \in \Sigma$.
- Let $\overline{\Sigma} = \Sigma \cup \{\triangleright, \Box, 0, 1\}$.

Cook-Levin

Since $L \in$ NP there is a path of the NDTM:

- Whose run time is bounded by some constant $T = n^c$
- Which accepts the input

We can describe the execution of the NDTM via a tableau
Which for a *specific execution path* could look like:

$\triangleright$	START	x_1	x_2	$\ldots$	x_n	$\square$	$\ldots$	$\square$
$\triangleright$	0	s	x_2	$\ldots$	x_n	$\square$	$\ldots$	$\square$
				$\vdots$				
$\triangleright$	0	1	ACCEPT	$\ldots$	1	0	$\ldots$	$\square$

where $s \in \Sigma$.

Tableau

$\triangleright$	START	x_1	x_2	$\dots$	x_n	$\square$	$\dots$	$\square$
$\triangleright$	0	s	x_2	$\dots$	x_n	$\square$	$\dots$	$\square$
				$\vdots$				
$\triangleright$	0	1	ACCEPT	$\dots$	1	0	$\dots$	$\square$

The tableau will have at most T rows, and $T + 2$ columns.

The total number of possible tableau is given by

$$T \times (T + 2) \times S \times 3^T$$

since each row can have at most one state symbol which comes from Σ and all the other symbols are on the tapes and so (bar the first column) are from $\{0, 1, \square\}$.

The 3^T term looks like a problem....

- But we know at least one such tableau accepts.

Obtaining a CNF

Let the variables for the CNF be $x_{i,j,s}$, where $s \in \overline{\Sigma}$ variable for each cell.

- The variable $x_{i,j,s}$ is true if cell (i, j) contains s.
- Total number of variables is $T \cdot (T + 1) \cdot (S + 4)$, i.e. polynomial in n.

We want a (poly-size) CNF which is satisfiable iff *there is a* tableau which is valid.

Obtaining a CNF

Define four sub-formulae:

- ϕ_{accept} evaluates to true if the tableau contains an accepting state.
- ϕ_{start} evaluates to true if the first row is the correct starting configuration;
- ϕ_{cell} evaluates to true if all squares in the tableau are uniquely filled;
- ϕ_{move} evaluates to true if the tableau is a valid computational path of M (according to its transition rules);

Then our desired CNF is ϕ,

$$\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}.$$

ϕ_{accept}

This is the easiest one we just require at least one cell to contain an ACCEPT symbol.

So

$$\phi_{\text{accept}} = \bigvee_{i,j} c_{i,j,\text{ACCEPT}}.$$

Note this is a poly-size CNF.

ϕ_{start}

Need the first row to contain:

- $\triangleright$ in the first column.
- START in the second column.
- Then the *n* input values.
- Then blank values.

$$\phi_{\text{start}} = c_{1,1,\triangleright} \wedge c_{1,2,\text{START}} \wedge c_{1,3,x_1} \wedge c_{1,4,x_2} \wedge \cdots \wedge c_{1,n+2,x_n} \\ \wedge c_{1,n+3,\square} \wedge \cdots \wedge c_{1,T+2,\square}.$$

Note this is a poly-size CNF.

- So the CNF depends on the problem instance

ϕ_{cell}

We need to encode

- Every cell has a value
- No cell has two values

So we have

$$\phi_{\text{cell}} = \bigwedge_{i,j} \left(\left(\bigvee_{s \in \overline{\Sigma}} c_{i,j,s} \right) \bigwedge_{t \neq u, t, u \in \overline{\Sigma}} (\neg c_{i,j,t} \vee \neg c_{i,j,u}) \right).$$

Note this is a poly-size CNF.

ϕ_{move}

This is the big problem and where we need to cope with the 3^T term above.

We use the *locality* of Turing machines.

In particular that a TM can only affect cells around the head tape.

Consider a 2×3 “window” (submatrix) in a tableau.

- There are $(T-1) \times (T-1)$ possible window locations,
- i.e. a poly number.

Each window can contain $(S+4)^6$ possible values

- i.e. a poly number.

ϕ_{move}

However, some of the windows are “illegal”

The illegal ones correspond to transitions which cannot take place.

Example

The head cannot move two positions in one step,

So if s is a head state the following window is illegal

s	0	0
0	0	s

ϕ_{move}

The constraint that a window w is legal can be written as

$$\phi_w = \bigvee_{\text{legal windows } v} [w_{ij} = v_{ij} \text{ for } i \in \{1,2\}, j \in \{1,2,3\}].$$

Note that this constraint is a boolean formula with a fixed, finite number of terms, which can be written in CNF.

Note as we are talking about a NDTM there are multiple paths, and hence multiple legal tableau for any specific point we have reached in the computation path.

- We only need one path to exist
- i.e. one assignment to the variables in our SAT problem
- But we encode *all* paths in the legal moves.

ϕ_{move}

We set

$$\phi_{\text{move}} = \bigwedge_{\text{windows } w} \phi_w.$$

Need to show that if all windows in a tableau are legal, then each row is a valid configuration which follows from the previous one.

Do this by induction:

- For the base case, ϕ_{start} ensures that the first row is valid.
- If row r is a valid configuration, then row $r+1$ is also a valid configuration.
  - Need to show each row only contains one head configuration.
  - We dont know where it is, but we do know there is only one.
  - See notes for how this is done (its easy).

Obtaining a CNF

That the resulting CNF is poly-size is trivial to see as all components are poly-size

So what have we done?

Given a problem $L \in \text{NP}$ we have

- Used the fact *there exists* a NDTM accepting L to encode a tableau.
- The tableau defines a SAT formula. The variables being the cells in the tableau.
- If there is an accepting state then for L then there is a solution to this SAT formulae.
- Calling a sub-procedure to solve SAT will decide if there is an accepting assignment.
- Thus if we can solve SAT then we can decide any language in NP.
- Thus SAT is NP-complete.

All is not as it seems

There is a rather cool logical issue with the previous slide

We started with a problem which we knew was in NP

Then we showed that we could decide whether it was in NP.

The issue is that NP is defined by the fact that there exists a NDTM (or verifier)

It does not say we can find it!

All is not as it seems

There is a rather cool logical issue with the previous slide

We started with a problem which we knew was in NP

Then we showed that we could decide whether it was in NP.

The issue is that NP is defined by the fact that there exists a NDTM (or verifier)

It does not say we can find it!

Constructive

The point is the proof is non-constructive.

- Given an NP language you cannot write down the SAT formula via the proof.
- Many parts of math are non-constructive, e.g. mean value theorem.

However, in practice for most problems in NP we know an explicit verifier.

Given this explicit verifier writing down the SAT formulae is relatively easy.

So in practice if we have an efficient procedure to solve SAT, then we can solve any problem in NP efficiently.

More NP-complete Languages

3-SAT

- Literal: x or $\overline{x}$
- Clause: Disjunction of literals, e.g. $(x_1 \vee \overline{x}_2 \vee x_3)$
- ϕ is in conjunctive normal form if ϕ is a conjunction of clauses
- 3-CNF formula: A CNF formula with all clauses having 3 literals, e.g. $(x_1 \vee \overline{x}_2 \vee \overline{x}_3) \wedge (x_2 \vee \overline{x}_5 \vee x_6) \wedge (x_3 \vee \overline{x}_6 \vee x_4)$.

$$\textit{3-SAT} := \{\langle\phi\rangle \mid \phi \text{ is a satisfiable 3-CNF formula}\}$$

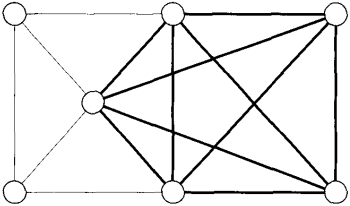
Theorem. *3-SAT* is NP-complete.

Theorem. *2-SAT* $\in$ P.

More NP-complete Languages

A k-**clique** in a graph is a set of k nodes in which every two nodes are connected by an edge.

$$\mathit{CLIQUE} := \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-clique}\}$$

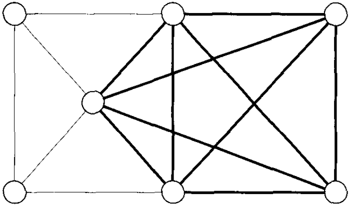


More NP-complete Languages

A k-**clique** in a graph is a set of k nodes in which every two nodes are connected by an edge.

$$CLIQUE := \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-clique}\}$$

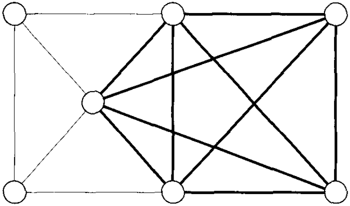
Theorem. *CLIQUE* is NP-complete.



More NP-complete Languages

A k-**clique** in a graph is a set of k nodes in which every two nodes are connected by an edge.

$$CLIQUE := \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-clique}\}$$



Theorem. *CLIQUE* is NP-complete.

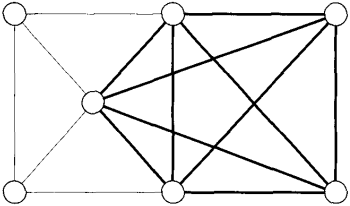
More NP-complete languages:

- *HAMPATH*

More NP-complete Languages

A k-**clique** in a graph is a set of k nodes in which every two nodes are connected by an edge.

$$CLIQUE := \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-clique}\}$$



Theorem. *CLIQUE* is NP-complete.

More NP-complete languages:

- *HAMPATH*
- $SUBSET\text{-}SUM = \{\{x_1, \ldots, x_k\} \mid$ for some $\{y_1, \ldots y_\ell\} \subseteq \{x_1, \ldots, x_k\}$ we have: $\sum y_i = 0\}$

CLIQUE is NP-complete: Proof Example

Clearly *CLIQUE* $\in$ NP

To show *CLIQUE* is NP-complete, we reduce 3-SAT to CLIQUE.

Given a 3-SAT formulae ϕ with m clauses create a graph G:

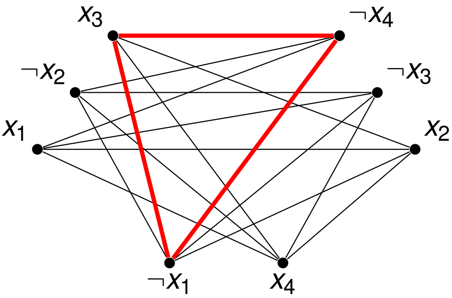
- Have at most $3 \cdot m$ vertices, one for each term in a clause.
- Associate at most three vertices with each clause in ϕ.
- Connect every pair of vertices with an edge
  - Except those in the same clause
  - Except vertices associated to a variable and its negation

CLIQUE is NP-complete: Proof Example

The graph for the formulae

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee \neg x_4)$$

is...



One 3 clique is highlighted corresponding to the witness

$$\{x_3 = 1, \neg x_1 = 1, \neg x_4 = 1\}.$$

CLIQUE is NP-complete: Proof Example

If ϕ is satisfiable then G has an m-clique:

- Let w be the witness to satisfiability
- Form set S of variables by picking x_i in each clause C such that
  - $x_i = w_i$
  - x_i satisfies C.
- There is an edge from every element in S to every other element
- Thus S is a clique

CLIQUE is NP-complete: Proof Example

If G has an m-clique then ϕ is satisfiable:

- Any m-clique must contain exactly one vertex from each clause
- The vertices are consistent (as x and $\neg x$ not connected)
- Thus taking the labels for the vertices as being true we get a satisfying assignment

Summary: A procedure to determine decidability of CLIQUE implies a procedure to determine decidability of 3-SAT.

As 3-SAT is NP-complete (proved in notes), so is CLIQUE.