ITCS 532:
7. More on p-Time Reduction, and an Introduction to **NP**

Rob Egrot

The Directed Hamiltonian Circuit Problem

Definition 1 (DHCP)

The directed Hamiltonian circuit problem (*DHCP*) asks whether a given finite *directed* simple graph has a Hamiltonian circuit. A Hamiltonian circuit in a directed graph is like a Hamiltonian circuit in an undirected graph except we take the edge directions into account.

$HCP \leq_p DHCP$

Theorem 2
$HCP \leq_p DHCP$, (p-time in $|V| + |E|$).

Proof

- We start with a finite undirected simple graph $G = (V, E)$.
- Must construct in p-time a finite directed simple graph G' that has a Hamiltonian circuit if and only if G does.
- The vertices of G' are copies of the vertices of G.
- For every undirected edge in G we add two directed edges to G' between the same vertices (one in each direction).
- Construction of G' is linear time ($O(m)$ if $m = |V| + |E|$).

$HCP \leq_p DHCP$ - Checking the Reduction

- G is a yes instance if and only if it has a Hamiltonian circuit $v_1, v_2, \ldots, v_n$.
- If $v_1, v_2, \ldots, v_n$ is a Hamiltonian circuit in G then there are directed edges in G' from v_i to v_{i+1} for all $i = 1, 2, \ldots, n-1$, and also from v_n to v_1.
- So $v_1, v_2, \ldots, v_n$ is also a Hamiltonian circuit in G'.
- Conversely, if $v_1, v_2, \ldots, v_n$ is a Hamiltonian circuit in G' then there must be edges in G connecting v_i and v_{i+1} for all $i = 1, 2, \ldots, n-1$, and also an edge connecting v_n and v_1.
- So $v_1, v_2, \ldots, v_n$ is also a Hamiltonian circuit in G as required.

$DHCP \leq_p HCP$

Theorem 3

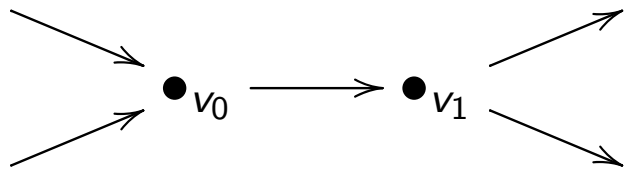
$DHCP \leq_p HCP$ (p-time in $|V| + |E|$).

Proof

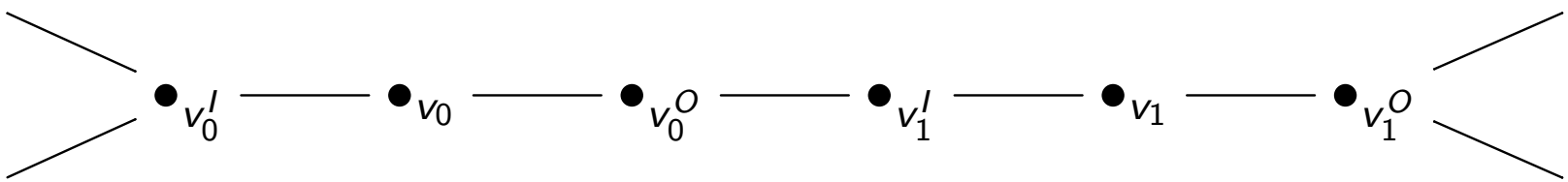
- This direction is not easy.
- We start with a directed simple graph G and we must construct in p-time an undirected simple graph G' that preserves 'yes' and 'no' instances.
- This is hard because it's not at all obvious how to encode the direction of edges of G in G'.
- This is a problem because a Hamiltonian circuit in G must travel along edges in the right direction.
- If we lose information about the direction of edges then Hamiltonian circuits in G' may not correspond to Hamiltonian circuits in G.

$DHCP \leq_p HCP$ - The Construction of G'

- If V and E are the sets of vertices and edges of G respectively, we define $G' = (V', E')$ as follows:
  - $V' = \bigcup\{\{v, v^I, v^O\} : v \in V\}$
  - $E' = \{\{v_0^O, v_1^I\} : (v_0, v_1) \in E\} \cup \{\{v, v^I\}, \{v, v^O\} : v \in V\}$
- So every vertex v of G gets two partner vertices, v^I and v^O, that encode the direction of edges.
- There are edges in G' connecting v to v^O and v^I, and for a directed edge (v, v') in G there's an edge between v^O and v'^I in G'. E.g.



becomes



$DHCP \leq_p HCP$ - Is the Reduction Correct?

- We need to show G has an HC if and only if G' does.
- If $|V| = n$ and $v_1, v_2, \ldots, v_n$ is a Hamiltonian circuit in G, then

$$v_1^I, v_1, v_1^O, v_2^I, v_2, v_2^O, \ldots, v_n^I, v_n, v_n^O$$

is a Hamiltonian circuit in G'.

$DHCP \leq_p HCP$ - Is the Reduction Correct?

- Conversely, if $|V| = n$ then $|V'| = 3n$.
- Suppose $c = u_1, u_2, \ldots, u_{3n}$ is a Hamiltonian circuit in G'.
- Then every $v \in V$ appears somewhere in c.
- For each $v \in V$, the edges to v are $\{v, v^I\}$ and $\{v, v^O\}$.
- Since $|V| = n$ and c has exactly $3n$ elements, a simple counting argument tells us that, after choosing a suitable starting point, c must have form either

$$c = v_1^I, v_1, v_1^O, v_2^I, v_2, v_2^O, \ldots, v_n^I, v_n, v_n^O,$$

or

$$c = v_1^O, v_1, v_1^I, v_2^O, v_2, v_2^I, \ldots, v_n^O, v_n, v_n^I.$$

- So by the construction of G', we get a Hamiltonian circuit in G by either following c forwards or backwards.

$DHCP \leq_p HCP$ - Is the Reduction p-time?

- We still have to check the reduction is p-time in $|V| + |E|$.
- First we constructed the vertices of G'.
- There are three of these for every vertex of G, so this step runs in $O(|V|)$ time.
- Then we added edges of form (v^I, v) and (v, v^O) for every vertex of G. Again this is $O(|V|)$.
- Finally we added edges of form (u^O, v^I) for every edge of G. This is obviously $O(|E|)$, to so the combined running time is also $O(|V| + |E|)$.

Verification vs Construction

- Every natural number can be written as a product of prime numbers in an essentially unique way.
- This is the *Fundamental Theorem of Arithmetic*.
- Given $n \in \mathbb{N}$, an obvious question is, 'what is the prime decomposition of n?'.
- This is not, in general, an easy question to answer.
- In fact, no p-time algorithm is known.
- It *is* easy to check whether a proposed factorization is correct.
- This illustrates an important principle:
  - It can be easy to *check* a solution, but hard to *find* a solution.
- In the case of the prime factorization problem, this asymmetry lies at the heart of RSA encryption.
- We can formalize this in terms of decision problems and Turing machines with something called a *verifier*.

Verifiers

Definition 4 (Verifier)

Given a language L over a finite alphabet Σ, a verifier for L is a modified Turing machine V, where V takes two inputs instead of one (consider the inputs to be separated by a special symbol), and such that

$$x \in L \iff \text{there is } y \in \Sigma^* \text{ such that } V(x, y) \text{ accepts}$$

- Here y is called a *certificate* for x.
- A verifier must halt for all inputs.
- We say a verifier for L runs in p-time if it runs in p-time with respect to the length of x for all inputs (x, y).
- The certificate is like a proposed solution.
- The verifier accepts x, which encodes an instance of a decision problem, if and only if there's some proposed solution y that it can check is actually a solution.

Example

Example 5 (Composite numbers)

- We could build a verifier to check if numbers are composite (i.e. not prime).
- Here the certificates would be potential non-trivial divisors.
- V would run by checking whether y non-trivially divides x (so we exclude $y = 1$ or $y = x$).
- If $y \mid x$ and $y \notin \{1, x\}$ then $V(\mathbf{code}(x), \mathbf{code}(y))$ accepts.
- If not it rejects.

Example

Example 6 (Hamiltonian circuits)

- An instance of the Hamiltonian circuit problem is a graph G.
- A certificate for G is just a sequence s of vertices (in coded form).
- A verifier V for the Hamiltonian circuit problem would accept on input $(\mathbf{code}(G), \mathbf{code}(s))$ if and only if s defines a Hamiltonian circuit in G.

What's the Point of Verifiers?

- If we don't care about run time, then verifiers don't really add anything to our toolkit for describing languages.
- The next theorem demonstrates this.
- However, when we do care about run times this changes, as we will see.

Verifiers and R.E. languages

Theorem 7

A language has a verifier if and only if it is recursively enumerable.

Proof

- Let L be r.e. and let T be a TM that semidecides L.
- We construct a verifier V_T so that when V_T runs on $(x, \mathbf{code}(n))$ it computes $T(x)$ for n steps ($n \in \mathbb{N}$).
- It accepts if $T(x)$ accepts within n steps, and rejects otherwise.
- It also rejects if y is not $\mathbf{code}(n)$ for some n.
- V_T is a verifier for L because:
  - If $x \in L$ there is n such that $T(x)$ halts in n steps.
  - So $V_T(x, \mathbf{code}(n))$ accepts.
  - I.e. strings in L have a certificate.
  - $V_T(x, y)$ rejects if y is not $\mathbf{code}(n)$ for any n.
  - $V_T(x, \mathbf{code}(n))$ rejects if $T(x)$ does not halt within n steps.
  - I.e. strings not in L have no certificate.

Verifiers and R.E. languages

Proof Continued

- For the converse, let L have a verifier V.
- We construct a Turing machine T_V such that $T_V(x)$ operates by generating strings y in lexicographic order and computing $V(x, y)$, and halting if $V(x, y)$ accepts.
- Then T_V semidecides L because $x \in L$ if and only if there is y such that $V(x, y)$ accepts, and, as T_V computes $V(x, y)$ for every y eventually, it will halt if and only if $x \in L$.

Complexity Classes

- We say that languages/decision problems are in the complexity class **P** if there is a Turing machine that decides them in p-time.
- We say languages/decision problems are in the complexity class **NP** if they have a verifier that runs in p-time.
- The distinction between **P** and **NP** formalizes the distinction between problems that can be efficiently solved, and problems whose solutions can be efficiently checked.
- As is often the case in complexity theory, these classes are rather coarse.
- There's a big difference between a problem with a linear time solution, and a problem with an $O(n^{10,000,000})$ solution, but we gloss over it.

Turing Machine Variants

- We have defined **P** and **NP** in terms of kinds of Turing machines, and, as we have seen, there are many variant definitions we could use (e.g. multi-tape etc.).
- These are equivalent to the 'standard' definition in terms of what problems they can decide.
- But could using a different model of computation affect which problems are in **P** and **NP**.
- For example, we know a problem that can be solved by a 'standard' Turing machine could likely be solved in fewer steps by a Turing machine with more tapes.
- Could this speed increase be such that there is a problem solvable in p-time by a multi-tape machine that is not solvable in p-time by a 'standard' machine?

Turing Machine Variants

- The answer turns out to be no.
- If we carefully analyze the modeling of multi-tape Turing machines by single-tape Turing machines from a previous class, we see that the number of steps taken is polynomially bounded by the number of steps the multi-tape machine takes.
- Therefore, a problem that is p-time for a multi-tape machine will also be p-time for a single tape machine.
- The same is true for the other Turing machine variants too.
- This means our definitions for **P** and **NP** are *robust*.
- They do not depend unreasonably on the definition of 'Turing machine'.

The Class **NP**

- Note that **NP** does *not* stand for *non-polynomial*.
- This is a common but understandable misconception.
- **NP** actually stands for *non-deterministic polynomial*.
- The name comes from an alternative definition of the class **NP**, which we will see in the next class.