Discrete Optimization
Lecture-15

Ngày 6 tháng 12 năm 2011

A decision problem is a problem for which the answer is **YES** or **NO** (true or false, 1 or 0 etc.).

Example

Computational Complexity

A decision problem is a problem for which the answer is **YES** or **NO** (true or false, 1 or 0 etc.).

Example

- *Is the graph G 3−colorable?*

Computational Complexity

A decision problem is a problem for which the answer is **YES** or **NO** (true or false, 1 or 0 etc.).

Example

- *Is the graph G* $3-$*colorable?*
- *Is n a compound integer?*

Computational Complexity

A decision problem is a problem for which the answer is **YES** or **NO** (true or false, 1 or 0 etc.).

Example

- *Is the graph G* $3-$*colorable?*
- *Is n a compound integer?*
- *Is p a prime number?*

Computational Complexity

A decision problem is a problem for which the answer is **YES** or **NO** (true or false, 1 or 0 etc.).

Example

- *Is the graph G* $3-$*colorable?*
- *Is n a compound integer?*
- *Is p a prime number?*
- *Does the given assignment instance have an assignment whose cost is* $\leq 10,000,000,000$ *VND?*

Computational Complexity

A decision problem is a problem for which the answer is **YES** or **NO** (true or false, 1 or 0 etc.).

Example

- *Is the graph G* $3-$*colorable?*
- *Is n a compound integer?*
- *Is p a prime number?*
- *Does the given assignment instance have an assignment whose cost is* $\leq 10,000,000,000$ *VND?*
- *Does the given network have a cut of size* $\leq$ *m?*

A decision problem is a problem for which the answer is **YES** or **NO** (true or false, 1 or 0 etc.).

Example

- *Is the graph G $3-$colorable?*
- *Is n a compound integer?*
- *Is p a prime number?*
- *Does the given assignment instance have an assignment whose cost is $\leq 10,000,000,000$ VND?*
- *Does the given network have a cut of size $\leq m$?*
- *Does the graph G have a vertex cover of size k?*

Computational Complexity

A decision problem is a problem for which the answer is **YES** or **NO** (true or false, 1 or 0 etc.).

Example

- *Is the graph G* $3-$*colorable?*
- *Is n a compound integer?*
- *Is p a prime number?*
- *Does the given assignment instance have an assignment whose cost is* $\leq 10,000,000,000$ *VND?*
- *Does the given network have a cut of size* $\leq m$*?*
- *Does the graph G have a vertex cover of size k?*
- *Does the Digraph D have a Hamiltonian cycle?*

Computing models

Our common computing, whether by a computer, calculator, cell-phone or by hand is an execution of sequential operations. Thus to multiply $456301 * 432$ we will execute single digit multiplications and additions of single digits (including carry over).

There are alternative computing models. One of them is the **Non-Deterministic** computing model. In a deterministic computing model an algorithm will execute on a given instance the same steps in different runs. In a non-deterministic model it may execute different steps on the same instance.

Computing models

Our common computing, whether by a computer, calculator, cell-phone or by hand is an execution of sequential operations. Thus to multiply $456301 * 432$ we will execute single digit multiplications and additions of single digits (including carry over).

There are alternative computing models. One of them is the **Non-Deterministic** computing model. In a deterministic computing model an algorithm will execute on a given instance the same steps in different runs. In a non-deterministic model it may execute different steps on the same instance.

There are a few equivalent definitions of the **NP** (non-deterministic) computing model. Before discussing the definition we shall examine an example.

A non-deterministic algorithm

Example

Suppose we wish to decide whether an integer p is prime. Here is an "algorithm:"

A non-deterministic algorithm

Example

Suppose we wish to decide whether an integer p is prime. Here is an "algorithm:"

- *Repeat* 100 *times:*

A non-deterministic algorithm

Example

Suppose we wish to decide whether an integer p is prime. Here is an "algorithm:"

- *Repeat* 100 *times:*
- *Randomly select an integer $a < p$.*

A non-deterministic algorithm

Example

Suppose we wish to decide whether an integer p is prime. Here is an "algorithm:"

- *Repeat* 100 *times:*
- *Randomly select an integer $a < p$.*
- *Calculate: $b = a^{p-1} \bmod p$.*

A non-deterministic algorithm

Example

Suppose we wish to decide whether an integer p is prime. Here is an "algorithm:"

- *Repeat* 100 *times:*
- *Randomly select an integer* $a < p$.
- *Calculate:* $b = a^{p-1} \mod p$.
- *If* $b \neq 1$ **STOP!** *p is not prime (composite).*

A non-deterministic algorithm

Example

Suppose we wish to decide whether an integer p is prime. Here is an "algorithm:"

- *Repeat* 100 *times:*
- *Randomly select an integer* $a < p$.
- *Calculate:* $b = a^{p-1} \bmod p$.
- *If* $b \neq 1$ **STOP!** *p is not prime (composite).*
- *"p is probably prime."*

A non-deterministic algorithm

Example

Suppose we wish to decide whether an integer p is prime. Here is an "algorithm:"

- *Repeat* 100 *times:*
- *Randomly select an integer $a < p$.*
- *Calculate: $b = a^{p-1} \mod p$.*
- *If $b \neq 1$* **STOP!** *p is not prime (composite).*
- *"p is probably prime."*

A non-deterministic algorithm

Example

Suppose we wish to decide whether an integer p is prime. Here is an "algorithm:"

- *Repeat* 100 *times:*
- *Randomly select an integer* $a < p$.
- *Calculate:* $b = a^{p-1} \bmod p$.
- *If* $b \neq 1$ **STOP!** *p is not prime (composite).*
- *"p is probably prime."*

1. Let $p = 998667686017$.

A non-deterministic algorithm

Example

Suppose we wish to decide whether an integer p is prime. Here is an "algorithm:"

- *Repeat* 100 *times:*
- *Randomly select an integer* $a < p$.
- *Calculate:* $b = a^{p-1} \mod p$.
- *If* $b \neq 1$ **STOP!** *p is not prime (composite).*
- *"p is probably prime."*

1. Let $p = 998667686017$.
2. $73^{998667686016} \mod 998667686017 = 1$.

A non-deterministic algorithm

Example

Suppose we wish to decide whether an integer p is prime. Here is an "algorithm:"

- *Repeat* 100 *times:*
- *Randomly select an integer* $a < p$.
- *Calculate:* $b = a^{p-1} \bmod p$.
- *If* $b \neq 1$ **STOP!** *p is not prime (composite).*
- *"p is probably prime."*

1. Let $p = 998667686017$.
2. $73^{998667686016} \bmod 998667686017 = 1$.
3. $739^{998667686016} \bmod 998667686017 = 1$.

A non-deterministic algorithm

Example

Suppose we wish to decide whether an integer p is prime. Here is an "algorithm:"

- *Repeat* 100 *times:*
- *Randomly select an integer* $a < p$.
- *Calculate:* $b = a^{p-1} \bmod p$.
- *If* $b \neq 1$ **STOP!** *p is not prime (composite).*
- *"p is probably prime."*

1. Let $p = 998667686017$.
2. $73^{998667686016} \bmod 998667686017 = 1$.
3. $739^{998667686016} \bmod 998667686017 = 1$.
4. After 100 random choices a, the algorithm found $a^{998667686016} \bmod 998667686017 = 1$.

A non-deterministic algorithm

Example

Suppose we wish to decide whether an integer p is prime. Here is an "algorithm:"

- *Repeat* 100 *times:*
- *Randomly select an integer* $a < p$.
- *Calculate:* $b = a^{p-1} \bmod p$.
- *If* $b \neq 1$ **STOP!** *p is not prime (composite).*
- *"p is probably prime."*

1. Let $p = 998667686017$.
2. $73^{998667686016} \bmod 998667686017 = 1$.
3. $739^{998667686016} \bmod 998667686017 = 1$.
4. After 100 random choices a, the algorithm found $a^{998667686016} \bmod 998667686017 = 1$.
5. " 998667686017 is probably prime."

The class NP

In the 55^{th} attempt of the fourth run, the algorithm calculated:
$737^{998667686016} \bmod 998667686017 = 402448171978$.

998667686017 is not prime.

In the 55^{th} attempt of the fourth run, the algorithm calculated: $737^{998667686016} \bmod 998667686017 = 402448171978$.

998667686017 is not prime.

The same algorithm executed different steps on the same input and with different results.

Definition

A decision problem $\mathbb{A}$ *is* **efficiently certifiable** *if for every instance of size n one can attach a certificate and an algorithm* $\mathbb{B}$ *such that:*

The class NP

In the 55^{th} attempt of the fourth run, the algorithm calculated:
$737^{998667686016} \bmod 998667686017 = 402448171978$.

998667686017 is not prime.

The same algorithm executed different steps on the same input and with different results.

Definition

A decision problem $\mathbb{A}$ *is* **efficiently certifiable** *if for every instance of size n one can attach a certificate and an algorithm* $\mathbb{B}$ *such that:*

- *The size of the certificate C is polynomial in n.*

In the 55^{th} attempt of the fourth run, the algorithm calculated: $737^{998667686016} \bmod 998667686017 = 402448171978$.

998667686017 is not prime.

The same algorithm executed different steps on the same input and with different results.

Definition

A decision problem $\mathbb{A}$ is **efficiently certifiable** *if for every instance of size n one can attach a certificate and an algorithm $\mathbb{B}$ such that:*

- *The size of the certificate C is polynomial in n.*
- *The algorithm $\mathbb{B}$ runs in polynomial time on the combined instance (n, C).*

The class NP

In the 55^{th} attempt of the fourth run, the algorithm calculated: $737^{998667686016} \bmod 998667686017 = 402448171978$.

998667686017 is not prime.

The same algorithm executed different steps on the same input and with different results.

Definition

A decision problem $\mathbb{A}$ is **efficiently certifiable** *if for every instance of size n one can attach a certificate and an algorithm $\mathbb{B}$ such that:*

- *The size of the certificate C is polynomial in n.*
- *The algorithm $\mathbb{B}$ runs in polynomial time on the combined instance (n, C).*
- *$\mathbb{B}$ returns true if and only if n is a true instance for $\mathbb{A}$.*

Examples

To understand this definition we shall study some examples:

- Decision problem: Input: n. Output: YES if n is composite?

Examples

To understand this definition we shall study some examples:

- Decision problem: Input: n. Output: YES if n is composite?
  - Certficate: an integer k. Algorithm : $gcd(n, k) > 1$?.

Examples

To understand this definition we shall study some examples:

- Decision problem: Input: n. Output: YES if n is composite?
  - Certficate: an integer k. Algorithm : $gcd(n, k) > 1$?.
  - Certficate: an integer m. Algorithm: $m^{n-1} \bmod n > 1$?.

Examples

To understand this definition we shall study some examples:

- Decision problem: Input: n. Output: YES if n is composite?
  - Certficate: an integer k. Algorithm : $gcd(n, k) > 1$?.
  - Certficate: an integer m. Algorithm: $m^{n-1} \bmod n > 1$?.
- Decision problem: Input: (G, k).

  Output: YES, if G is $3 - colorable$.

Examples

To understand this definition we shall study some examples:

- Decision problem: Input: n. Output: YES if n is composite?
  - Certficate: an integer k. Algorithm : $gcd(n,k) > 1$?.
  - Certficate: an integer m. Algorithm: $m^{n-1} \bmod n > 1$?.
- Decision problem: Input: (G,k).

  Output: YES, if G is $3 - colorable$.
- Certificate: a list $\chi(v_i) = j$.

  Verify: $j = 1, 2 \text{ or } 3, \quad (v_i, v_j) \in E(G) \Rightarrow \chi(v_i) \neq \chi(v_j)$.

Examples

To understand this definition we shall study some examples:

- Decision problem: Input: n. Output: YES if n is composite?
  - Certficate: an integer k. Algorithm : $gcd(n, k) > 1?$.
  - Certficate: an integer m. Algorithm: $m^{n-1} \bmod n > 1?$.
- Decision problem: Input: (G, k).

  Output: YES, if G is $3 - colorable$.
- Certificate: a list $\chi(v_i) = j$.

  Verify: $j = 1, 2$ *or* $3, \quad (v_i, v_j) \in E(G) \Rightarrow \chi(v_i) \neq \chi(v_j)$.
- Decision problem: Input: a graph G.

  Output: YES if G is Hamiltonian.

Examples

To understand this definition we shall study some examples:

- Decision problem: Input: n. Output: YES if n is composite?
  - Certficate: an integer k. Algorithm : $gcd(n,k) > 1$?.
  - Certficate: an integer m. Algorithm: $m^{n-1} \bmod n > 1$?.
- Decision problem: Input: (G,k).

  Output: YES, if G is $3 - colorable$.
- Certificate: a list $\chi(v_i) = j$.

  Verify: $j = 1, 2 \ or\ 3, \quad (v_i, v_j) \in E(G) \Rightarrow \chi(v_i) \neq \chi(v_j)$.
- Decision problem: Input: a graph G.

  Output: YES if G is Hamiltonian.
- Certifcate: $v_1, v_2, \ldots, v_n$.

  Verify: $(v_{i-1}, v_i) \vee (v_n, v_1) \in E(G),\ i = 2, \ldots n$.

Examples

To understand this definition we shall study some examples:

- Decision problem: Input: n. Output: YES if n is composite?
  - Certficate: an integer k. Algorithm : $gcd(n, k) > 1$?.
  - Certficate: an integer m. Algorithm: m^{n-1} mod $n > 1$?.
- Decision problem: Input: (G, k).

  Output: YES, if G is $3 - colorable$.
- Certificate: a list $\chi(v_i) = j$.

  Verify: $j = 1, 2$ *or* $3, \quad (v_i, v_j) \in E(G) \Rightarrow \chi(v_i) \neq \chi(v_j)$.
- Decision problem: Input: a graph G.

  Output: YES if G is Hamiltonian.
- Certifcate: $v_1, v_2, \ldots, v_n$.

  Verify: $(v_{i-1}, v_i) \vee (v_n, v_1) \in E(G),\ i = 2, \ldots n$.
- Decision problem: Input a graph G.

  Output: YES if G is Eulerian.

Examples

To understand this definition we shall study some examples:

- Decision problem: Input: n. Output: YES if n is composite?
  - Certficate: an integer k. Algorithm : $gcd(n, k) > 1$?.
  - Certficate: an integer m. Algorithm: $m^{n-1} \bmod n > 1$?.
- Decision problem: Input: (G, k).

  Output: YES, if G is $3 - colorable$.
- Certificate: a list $\chi(v_i) = j$.

  Verify: $j = 1, 2$ *or* 3, $\quad (v_i, v_j) \in E(G) \Rightarrow \chi(v_i) \neq \chi(v_j)$.
- Decision problem: Input: a graph G.

  Output: YES if G is Hamiltonian.
- Certifcate: $v_1, v_2, \ldots, v_n$.

  Verify: $(v_{i-1}, v_i) \vee (v_n, v_1) \in E(G),\ i = 2, \ldots n$.
- Decision problem: Input a graph G.

  Output: YES if G is Eulerian.
- Verify that G is connecetd and $d_G(v_i) \equiv 0 \bmod 2$.

Definition

The class of efficiently certifiable decision problems is called **NP** *(non-deterministic polynomial).*

Definition

The class of efficiently certifiable decision problems is called **NP** *(non-deterministic polynomial).*

Definition

The class of decision problems for which there is an efficient algorithm is denoted by **P**.

Definition

The class of efficiently certifiable decision problems is called **NP** *(non-deterministic polynomial).*

Definition

The class of decision problems for which there is an efficient algorithm is denoted by **P***.*

Comment

Definition

The class of efficiently certifiable decision problems is called **NP** *(non-deterministic polynomial).*

Definition

The class of decision problems for which there is an efficient algorithm is denoted by **P**.

Comment

- *As clearly demosntrated by the last example,* $\mathbf{P} \subseteq \mathbf{NP}$.

Definition

The class of efficiently certifiable decision problems is called **NP** *(non-deterministic polynomial).*

Definition

The class of decision problems for which there is an efficient algorithm is denoted by **P**.

Comment

- *As clearly demosntrated by the last example,* $\mathbf{P} \subseteq \mathbf{NP}$.
- *Notice the difference between the two certificates in the first example:*

Definition

The class of efficiently certifiable decision problems is called **NP** *(non-deterministic polynomial).*

Definition

The class of decision problems for which there is an efficient algorithm is denoted by **P***.*

Comment

- *As clearly demosntrated by the last example,* **P** $\subseteq$ **NP***.*
- *Notice the difference between the two certificates in the first example:*
  - *In the first example the first certificate actually identifies a divisor of the input n.*

Definition

The class of efficiently certifiable decision problems is called **NP** *(non-deterministic polynomial).*

Definition

The class of decision problems for which there is an efficient algorithm is denoted by **P**.

Comment

- *As clearly demosntrated by the last example,* **P** $\subseteq$ **NP**.
- *Notice the difference between the two certificates in the first example:*
  - *In the first example the first certificate actually identifies a divisor of the input n.*
  - *The second certificate does not help us find such a divisor.*

Definition

The class of efficiently certifiable decision problems is called **NP** *(non-deterministic polynomial).*

Definition

The class of decision problems for which there is an efficient algorithm is denoted by **P**.

Comment

- *As clearly demosntrated by the last example,* **P** $\subseteq$ **NP**.
- *Notice the difference between the two certificates in the first example:*
  - *In the first example the first certificate actually identifies a divisor of the input n.*
  - *The second certificate does not help us find such a divisor.*
  - *But finding a certficate for the second verification is usually quite easy allowing us to implement the function is_prime(n) in SAGE and other mathematical packages.*

The Satifiability Problem

Another computing model (heuristic) of the **NP** computing is the following example. Suppose you wish to decide whether a given boolean function $f(x_1, x_2, \ldots, x_n)$ in CNF is satisfiable.

- You start by choosing a variable x_1 and spawn two threads: in one thread make $x_1 = T$ and mark all clauses that are satisfied by this selection. In the second thread make $x_1 = F$ and do the same.

The Satifiability Problem

Another computing model (heuristic) of the **NP** computing is the following example. Suppose you wish to decide whether a given boolean function $f(x_1, x_2, \dots, x_n)$ in CNF is satisfiable.

- You start by choosing a variable x_1 and spawn two threads: in one thread make $x_1 = T$ and mark all clauses that are satisfied by this selection. In the second thread make $x_1 = F$ and do the same.
- Each thread spawns two new threads: in one $x_2 = T$ and the other $x_2 = F$. Each thread marks the additional clauses that are satisfied.

The Satifiability Problem

Another computing model (heuristic) of the **NP** computing is the following example. Suppose you wish to decide whether a given boolean function $f(x_1, x_2, \ldots, x_n)$ in CNF is satisfiable.

- You start by choosing a variable x_1 and spawn two threads: in one thread make $x_1 = T$ and mark all clauses that are satisfied by this selection. In the second thread make $x_1 = F$ and do the same.
- Each thread spawns two new threads: in one $x_2 = T$ and the other $x_2 = F$. Each thread marks the additional clauses that are satisfied.
- All current threads work in parallel.

The Satifiability Problem

Another computing model (heuristic) of the **NP** computing is the following example. Suppose you wish to decide whether a given boolean function $f(x_1, x_2, \ldots, x_n)$ in CNF is satisfiable.

- You start by choosing a variable x_1 and spawn two threads: in one thread make $x_1 = T$ and mark all clauses that are satisfied by this selection. In the second thread make $x_1 = F$ and do the same.
- Each thread spawns two new threads: in one $x_2 = T$ and the other $x_2 = F$. Each thread marks the additional clauses that are satisfied.
- All current threads work in parallel.
- After 100 assignments there will be 2^{100} threads.

The Satifiability Problem

Another computing model (heuristic) of the **NP** computing is the following example. Suppose you wish to decide whether a given boolean function $f(x_1, x_2, \ldots, x_n)$ in CNF is satisfiable.

- You start by choosing a variable x_1 and spawn two threads: in one thread make $x_1 = T$ and mark all clauses that are satisfied by this selection. In the second thread make $x_1 = F$ and do the same.
- Each thread spawns two new threads: in one $x_2 = T$ and the other $x_2 = F$. Each thread marks the additional clauses that are satisfied.
- All current threads work in parallel.
- After 100 assignments there will be 2^{100} threads.
- Computation ends when a thread discovers a T assignment to all clauses.

Non-Deterministic Computing

If $f(x_1, x_2, \ldots, x_n)$ is satisfiable then a path of length $\leq n$ along the "correct" threads will indentify the assignment for which $f(x_1, x_2, \ldots, x_n) = T$.

Non-Deterministic Computing

If $f(x_1, x_2, \ldots, x_n)$ is satisfiable then a path of length $\leq n$ along the "correct" threads will indentify the assignment for which $f(x_1, x_2, \ldots, x_n) = T$.

The non-deterministic nature of this process should be clear now. Assume that at every execution we flip a coin and if it falls on HEAD we assign $x_i = T$ and if it is TAIL we assign $x_i = F$.

If we were "lucky" at every step we will identify the answer in no more than n steps. Also, every execution may execute different steps.

The generalization to any other decision problem is straight forward.

Comment

Comment

- *A decision problem* $\mathbb{P} \in$ **P** *can be easily adjusted to an efficient algorithm for solving a similar general problem.*

Comment

- *A decision problem $\mathbb{P} \in$ **P** can be easily adjusted to an efficient algorithm for solving a similar general problem.*
- *Example. Suppose we wish to find the chromatic number of a graph G. We can use the decision problem "Is G k-colorable" no more than $\log n$ times to determine $\chi(G)$.*

Comment

- *A decision problem $\mathbb{P} \in$ **P** can be easily adjusted to an efficient algorithm for solving a similar general problem.*
- *Example. Suppose we wish to find the chromatic number of a graph G. We can use the decision problem "Is G k-colorable" no more than $\log n$ times to determine $\chi(G)$.*
- *Another class of problems is called* **P-Space** *problems (again, P stands for polynomial).*

Comment

- *A decision problem* $\mathbb{P} \in$ **P** *can be easily adjusted to an efficient algorithm for solving a similar general problem.*
- *Example. Suppose we wish to find the chromatic number of a graph G. We can use the decision problem "Is G k-colorable" no more than* $\log n$ *times to determine* $\chi(G)$.
- *Another class of problems is called* **P-Space** *problems (again, P stands for polynomial).*
- *In this model we are only concerned about the amount of memory needed to solve a problem.*

Comment

- *A decision problem $\mathbb{P} \in$* **P** *can be easily adjusted to an efficient algorithm for solving a similar general problem.*
- *Example. Suppose we wish to find the chromatic number of a graph G. We can use the decision problem "Is G k-colorable" no more than $\log n$ times to determine $\chi(G)$.*
- *Another class of problems is called* **P-Space** *problems (again, P stands for polynomial).*
- *In this model we are only concerned about the amount of memory needed to solve a problem.*
- *For example the Satisfiability problem belongs to* **P-Space***.*

Comment

- *A decision problem $\mathbb{P} \in$* **P** *can be easily adjusted to an efficient algorithm for solving a similar general problem.*
- *Example. Suppose we wish to find the chromatic number of a graph G. We can use the decision problem "Is G k-colorable" no more than $\log n$ times to determine $\chi(G)$.*
- *Another class of problems is called* **P-Space** *problems (again, P stands for polynomial).*
- *In this model we are only concerned about the amount of memory needed to solve a problem.*
- *For example the Satisfiability problem belongs to* **P-Space**.
- *The space we need is for a single integer whose binary representation has length n (the number of variables in $f(x_1, \ldots, x_n)$.*

Comment

- *A decision problem* $\mathbb{P} \in$ **P** *can be easily adjusted to an efficient algorithm for solving a similar general problem.*
- *Example. Suppose we wish to find the chromatic number of a graph G. We can use the decision problem "Is G k-colorable" no more than* $\log n$ *times to determine* $\chi(G)$.
- *Another class of problems is called* **P-Space** *problems (again, P stands for polynomial).*
- *In this model we are only concerned about the amount of memory needed to solve a problem.*
- *For example the Satisfiability problem belongs to* **P-Space**.
- *The space we need is for a single integer whose binary representation has length n (the number of variables in* $f(x_1, \ldots, x_n)$.
- *As we shall see, this implies that* **NP**$\subseteq$ **P-Space**

The class NP

We can summarise:

The class **NP** contains the problems for which a given answer can be verified quickly.

A central question in Mathematics and CS is: **P = NP** ?

The class NP

We can summarise:

The class **NP** contains the problems for which a given answer can be verified quickly.

A central question in Mathematics and CS is: **P = NP** ?

This means that if an answer can be verified quickly, does it necessarily mean that an answer can also be found quickly?

The class NP

We can summarise:

The class **NP** contains the problems for which a given answer can be verified quickly.

A central question in Mathematics and CS is: **P = NP** ?

This means that if an answer can be verified quickly, does it necessarily mean that an answer can also be found quickly?

Examples

- Input: (n, m) Verify $n = k \cdot m$ can be verified very quickly.

  But given n can the divisor m be found quickly ?

The class NP

We can summarise:

The class **NP** contains the problems for which a given answer can be verified quickly.

A central question in Mathematics and CS is: **P = NP** ?

This means that if an answer can be verified quickly, does it necessarily mean that an answer can also be found quickly?

Examples

- Input: (n, m) Verify $n = k \cdot m$ can be verified very quickly.

  But given n can the divisor m be found quickly ?
- The subset sum problem: given a list of integers $A = (a_1, a_2, \dots, a_n)$ and an integer k.

  Is there a subset $B \subset A$ such that $\sum_{a_i \in B} a_i = k$?

The class NP

We can summarise:

The class **NP** contains the problems for which a given answer can be verified quickly.

A central question in Mathematics and CS is: **P = NP** ?

This means that if an answer can be verified quickly, does it necessarily mean that an answer can also be found quickly?

Examples

- Input: (n, m) Verify $n = k \cdot m$ can be verified very quickly.

  But given n can the divisor m be found quickly ?

- The subset sum problem: given a list of integers $A = (a_1, a_2, \ldots, a_n)$ and an integer k.

  Is there a subset $B \subset A$ such that $\sum_{a_i \in B} a_i = k$?

- This is easily verifiable using less than n additions. But we do not know how to find B.

The class NP-Complete

Question

Among all **NP** *complete problems, are there problems that are more difficult than all other* **NP** *problems?*

The class **NP-Complete**

Question

Among all **NP** *complete problems, are there problems that are more difficult than all other* **NP** *problems?*

Answer

In 1971 Steven Cook of the University of Toronto in his paper The complexity of theorem proving procedures introduced the class of **NP** *problems (even though it is not mentioned explicitly in this paper).*

He proved that every **NP** *problem is polynomially reducible to the* **SAT** *problem.*

The class **NP-Complete**

Question

Among all **NP** *complete problems, are there problems that are more difficult than all other* **NP** *problems?*

Answer

In 1971 Steven Cook of the University of Toronto in his paper The complexity of theorem proving procedures introduced the class of **NP** *problems (even though it is not mentioned explicitly in this paper).*

He proved that every **NP** *problem is polynomially reducible to the* **SAT** *problem.*

The class NP-Complete

Question

Among all **NP** *complete problems, are there problems that are more difficult than all other* **NP** *problems?*

Answer

In 1971 Steven Cook of the University of Toronto in his paper The complexity of theorem proving procedures introduced the class of **NP** *problems (even though it is not mentioned explicitly in this paper).*

He proved that every **NP** *problem is polynomially reducible to the* **SAT** *problem.*

This means that if you had a "black box" that could efficiently solve any instance of a **SAT** *problem then you can efficiently solve any problem in* **NP**.

NPC problems

Cook's seminal paper was almost immediately followed by papers identifying other such "black boxes." We studied two such problems:

- 3-SAT.

NPC problems

Cook's seminal paper was almost immediately followed by papers identifying other such "black boxes." We studied two such problems:

- 3-SAT.
- 3-colorabilty of graphs.

NPC problems

Cook's seminal paper was almost immediately followed by papers identifying other such “black boxes.” We studied two such problems:

- 3-SAT.
- 3-colorabilty of graphs.

NPC problems

Cook's seminal paper was almost immediately followed by papers identifying other such "black boxes." We studied two such problems:

- 3-SAT.
- 3-colorabilty of graphs.

This means that our class, K53, knows 3 "black boxes" that are **NPC**.

NPC problems

Cook's seminal paper was almost immediately followed by papers identifying other such "black boxes." We studied two such problems:

- 3-SAT.
- 3-colorabilty of graphs.

This means that our class, K53, knows 3 "black boxes" that are **NPC**.

To prove that a decison problem $\mathbb{A} \in$ **NPC** we need first to prove that it is in **NP** and then prove that one of the "black boxes" is polynomially reducible to it.

NPC problems

Cook's seminal paper was almost immediately followed by papers identifying other such "black boxes." We studied two such problems:

- 3-SAT.
- 3-colorabilty of graphs.

This means that our class, K53, knows 3 "black boxes" that are **NPC**.

To prove that a decison problem $\mathbb{A} \in$ **NPC** we need first to prove that it is in **NP** and then prove that one of the "black boxes" is polynomially reducible to it.

Currently, hundreds of such "black boxes" have been identified.

The central theme of Discrete Optimization is finding practical, efficient, solutions to problems. Thus understanding what is feasible and what is not is a key issue in Discrete Optimization. If we know that a problem is “hard” we may opt to seek alternative solutions.

The central theme of Discrete Optimization is finding practical, efficient, solutions to problems. Thus understanding what is feasible and what is not is a key issue in Discrete Optimization. If we know that a problem is “hard” we may opt to seek alternative solutions.

For instance, we shall soon see that the TSP is **NPC**. The alternative solution was to develop an approximation algorithm, preferably with a performance gurantee.

The central theme of Discrete Optimization is finding practical, efficient, solutions to problems. Thus understanding what is feasible and what is not is a key issue in Discrete Optimization. If we know that a problem is “hard” we may opt to seek alternative solutions.

For instance, we shall soon see that the TSP is **NPC**. The alternative solution was to develop an approximation algorithm, preferably with a performance gurantee.

Another striking example was the chromatic index of a graph. The problem is $\chi_1(G) = \Delta(G)$ is known to be in **NPC**, Yet we identified an efficient algorithm that guaranteed a coloring by no more than $\Delta(G) + 1$ colors.

The central theme of Discrete Optimization is finding practical, efficient, solutions to problems. Thus understanding what is feasible and what is not is a key issue in Discrete Optimization. If we know that a problem is “hard” we may opt to seek alternative solutions.

For instance, we shall soon see that the TSP is **NPC**. The alternative solution was to develop an approximation algorithm, preferably with a performance gurantee.

Another striking example was the chromatic index of a graph. The problem is $\chi_1(G) = \Delta(G)$ is known to be in **NPC**, Yet we identified an efficient algorithm that guaranteed a coloring by no more than $\Delta(G) + 1$ colors.

Finally, the problem is G_1 isomorphic to G_2 is clearly in **NP**. It is not known whether it is in **NPC** or in **P**. It is suspected that it might be “in between” the two classes if indeed **NP** $\neq$ **P**.

NPC Black Boxes

Theorem

3-SAT is reducible to the problem:

Input: G, k Output: YES if $\alpha(G) \geq k$.

NPC Black Boxes

Theorem

3-SAT is reducible to the problem:

Input: G, k *Output: YES if* $\alpha(G) \geq k$.

Chứng minh.

Assume that the 3-SAT instance has n boolean variables $x_1, x_2, \ldots, x_n$ and m clauses $(v_{i,1} \vee v_{i,2}, \vee v_{i,3})$ $i = 1, 2, \ldots m$.

Each $v_{i,j} = x_k$ *or* $v_{i,j} = \neg x_k$.

NPC Black Boxes

Theorem

3-SAT is reducible to the problem:

Input: G, k Output: YES if $\alpha(G) \geq k$.

Chứng minh.

Assume that the 3-SAT instance has n boolean variables $x_1, x_2, \ldots, x_n$ and m clauses $(v_{i,1} \vee v_{i,2}, \vee v_{i,3})$ $i = 1, 2, \ldots m$.

Each $v_{i,j} = x_k$ *or* $v_{i,j} = \neg x_k$.

NPC Black Boxes

Theorem

3-SAT is reducible to the problem:

Input: G, k *Output: YES if* $\alpha(G) \geq k$.

Chứng minh.

Assume that the 3-SAT instance has n boolean variables $x_1, x_2, \ldots, x_n$ and m clauses $(v_{i,1} \vee v_{i,2}, \vee v_{i,3})$ $i = 1, 2, \ldots m$.

Each $v_{i,j} = x_k$ *or* $v_{i,j} = \neg x_k$.

- Construct a graph G with $3k$ vertices labeled by $\{v_{i,j} \mid 1 \leq k \leq m,\ 1 \leq j \leq 3\}$.

□

NPC Black Boxes

Theorem

3-SAT is reducible to the problem:

Input: G, k Output: YES if $\alpha(G) \geq k$.

Chứng minh.

Assume that the 3-SAT instance has n boolean variables $x_1, x_2, \ldots, x_n$ and m clauses $(v_{i,1} \vee v_{i,2}, \vee v_{i,3})$ $i = 1, 2, \ldots m$.

Each $v_{i,j} = x_k$ *or* $v_{i,j} = \neg x_k$.

- Construct a graph G with $3k$ vertices labeled by $\{v_{i,j} \mid 1 \leq k \leq m,\ 1 \leq j \leq 3\}$.
- $(v_{i,j}, v_{i,j'}) \in E(G)$ (this will create a disjoint set of k triangles in G).

□

NPC Black Boxes

Theorem

3-SAT is reducible to the problem:

Input: G, k Output: YES if $\alpha(G) \geq k$.

Chứng minh.

Assume that the 3-SAT instance has n boolean variables $x_1, x_2, \dots, x_n$ and m clauses $(v_{i,1} \vee v_{i,2}, \vee v_{i,3})$ $i = 1, 2, \dots m$.

Each $v_{i,j} = x_k$ *or* $v_{i,j} = \neg x_k$.

- Construct a graph G with $3k$ vertices labeled by $\{v_{i,j} \mid 1 \leq k \leq m, \ 1 \leq j \leq 3\}$.
- $(v_{i,j}, v_{i,j'}) \in E(G)$ (this will create a disjoint set of k triangles in G).
- $(v_{i,j}, v_{i',j'}) \in E(G)$ *if* $v_{i,j} = x_t$ *and* $v_{i',j'} = \neg x_t$.

□

- Clearly $\alpha(G) \leq k$.

- Clearly $\alpha(G) \le k$.
- $\alpha(G) = k$ if and only if the 3-SAT instance is satisfiable.

- Clearly $\alpha(G) \leq k$.
- $\alpha(G) = k$ if and only if the 3-SAT instance is satisfiable.

Corollary

The problem:

Input: (G, m) *Output: G has a vertex cover of size* $\leq k$ *is* **NPC**.

- Clearly $\alpha(G) \leq k$.
- $\alpha(G) = k$ if and only if the 3-SAT instance is satisfiable.

Corollary

The problem:

Input: (G, m) *Output: G has a vertex cover of size* $\leq k$ *is* **NPC**.

Chứng minh.

G has a vertex cover of size $\leq m$ if it has an independent set of size $\geq |V(G)| - m$.
So we can use the vertex cover black box to solve the independent set problem.

We can also use the independent set black box to solve the vertex cover problem. □

We conclude with two more problems, the set cover problem and the TSP. Let S be the set of all students in HUS. Given $S_1, S_2, \ldots, S_n$ subsets of students and an integer k.

We conclude with two more problems, the set cover problem and the TSP. Let S be the set of all students in HUS. Given $S_1, S_2, \ldots, S_n$ subsets of students and an integer k.

Decision problem: are there subsets $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$ such that: $\bigcup_{n=1}^{k} S_{i_n} = S$?.

Theorem

The set cover "black box" can efficiently solve the vertex cover problem.

We conclude with two more problems, the set cover problem and the TSP. Let S be the set of all students in HUS. Given $S_1, S_2, \ldots, S_n$ subsets of students and an integer k.

Decision problem: are there subsets $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$ such that: $\bigcup_{n=1}^{k} S_{i_n} = S$?.

Theorem

The set cover "black box" can efficiently solve the vertex cover problem.

Chứng minh.

□

We conclude with two more problems, the set cover problem and the TSP. Let S be the set of all students in HUS. Given $S_1, S_2, \ldots, S_n$ subsets of students and an integer k.

Decision problem: are there subsets $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$ such that: $\bigcup_{n=1}^{k} S_{i_n} = S$?.

Theorem

The set cover "black box" can efficiently solve the vertex cover problem.

Chứng minh.

Question

Is this problem similar to any of the problems we discussed so far?

□

We conclude with two more problems, the set cover problem and the TSP. Let S be the set of all students in HUS. Given $S_1, S_2, \ldots, S_n$ subsets of students and an integer k.

Decision problem: are there subsets $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$ such that: $\bigcup_{n=1}^{k} S_{i_n} = S$?.

Theorem

The set cover "black box" can efficiently solve the vertex cover problem.

Chứng minh.

Question

Is this problem similar to any of the problems we discussed so far?

Answer

Indeed it is! and it is easy to reduce the vertex cover to it.

□

Hamiltonian Cycles

Except for the $3-$ colorability, all reductions we studied were quite simple. We will end our discussion by studying a more complicated example: the **TSP**.

Hamiltonian Cycles

Except for the $3-$ colorability, all reductions we studied were quite simple. We will end our discussion by studying a more complicated example: the **TSP**. We first show that the problem:

Input: digraph G.

Output: YES if G has a Hamitonian cycle.

Is in **NPC**.

Hamiltonian cycle is NPC

We shall show how a black-box that can effciently find Hamiltonian cycles in A Digraph can decide whether a given instance of 3-SAT is satisfiable.

Hamiltonian cycle is NPC

We shall show how a black-box that can effciently find Hamiltonian cycles in A Digraph can decide whether a given instance of 3-SAT is satisfiable.

Let $F(x_0, x_1, \ldots, x_{n-1})$ be a given 3-SAT instance with k clauses. The graph that will help us decide whether F is satisfiable will be constructed in two steps.

- We start with n bi-diretional paths, each of length $3k + 3$.

Hamiltonian cycle is NPC

We shall show how a black-box that can effciently find Hamiltonian cycles in A Digraph can decide whether a given instance of 3-SAT is satisfiable.

Let $F(x_0, x_1, \ldots, x_{n-1})$ be a given 3-SAT instance with k clauses. The graph that will help us decide whether F is satisfiable will be constructed in two steps.

- We start with n bi-diretional paths, each of length $3k + 3$.
- $P_i = v_{i,1} \overrightarrow{\longleftarrow} v_{i_2} \overrightarrow{\longleftarrow} \ldots \overrightarrow{\longleftarrow} v_{i,3k+3}$

Hamiltonian cycle is NPC

We shall show how a black-box that can effciently find Hamiltonian cycles in A Digraph can decide whether a given instance of 3-SAT is satisfiable.

Let $F(x_0, x_1, \ldots, x_{n-1})$ be a given 3-SAT instance with k clauses. The graph that will help us decide whether F is satisfiable will be constructed in two steps.

- We start with n bi-diretional paths, each of length $3k + 3$.
- $P_i = v_{i,1} \overset{\rightarrow}{\leftarrow} v_{i_2} \overset{\rightarrow}{\leftarrow} \ldots \overset{\rightarrow}{\leftarrow} v_{i,3k+3}$
- For $i = 0, 1, \ldots, n-1$ we add edges $v_{i,1} \rightarrow \{v_{i+1,1}, v_{i+1,3k+3}\}$ *and* $v_{i,1} \leftarrow \{v_{i+1,1}, v_{i+1,3k+3}\}$ (index arithmetic is done mod n).

Hamiltonian cycle is NPC

We shall show how a black-box that can effciently find Hamiltonian cycles in A Digraph can decide whether a given instance of 3-SAT is satisfiable.

Let $F(x_0, x_1, \ldots, x_{n-1})$ be a given 3-SAT instance with k clauses. The graph that will help us decide whether F is satisfiable will be constructed in two steps.

- We start with n bi-diretional paths, each of length $3k+3$.
- $P_i = v_{i,1} \overset{\rightarrow}{\leftarrow} v_{i_2} \overset{\rightarrow}{\leftarrow} \ldots \overset{\rightarrow}{\leftarrow} v_{i,3k+3}$
- For $i = 0, 1, \ldots, n-1$ we add edges $v_{i,1} \rightarrow \{v_{i+1,1}, v_{i+1,3k+3}\}$ *and* $v_{i,1} \leftarrow \{v_{i+1,1}, v_{i+1,3k+3}\}$ (index arithmetic is done mod n).
- We add two additional vertices: $t, s: \quad t \rightarrow s,\ s \rightarrow \{v_{0,1}, v_{0,3k+3}\}$ *and* $\{v_{n-1,1}, v_{n-1,3k=3}\} \rightarrow t$.

So far our graph has $n(3k+3)+2$ vertices. It has many different Hamiltonian cycles, 2^n of them.
There are 2^n possible truth assignment to the variables $x_1, x_2, \ldots, x_n$.
The idea is to associate every truth assignment with one of the cycles and use the clauses to "force" a Hamiltonian cycle if and only if $F(x_0, x_1, \ldots, x_{n-1})$ is satisfiable.

So far our graph has $n(3k+3)+2$ vertices. It has many different Hamiltonian cycles, 2^n of them.
There are 2^n possible truth assignment to the variables $x_1, x_2, \ldots, x_n$.
The idea is to associate every truth assignment with one of the cycles and use the clauses to “force” a Hamiltonian cycle if and only if $F(x_0, x_1, \ldots, x_{n-1})$ is satisfiable.

We also note that once a Hamiltonian cycle enters a path P_i it has to traverse the path in the same direction with no interruption.

So far our graph has $n(3k+3)+2$ vertices. It has many different Hamiltonian cycles, 2^n of them.
There are 2^n possible truth assignment to the variables $x_1, x_2, \ldots, x_n$.
The idea is to associate every truth assignment with one of the cycles and use the clauses to "force" a Hamiltonian cycle if and only if $F(x_0, x_1, \ldots, x_{n-1})$ is satisfiable.

We also note that once a Hamiltonian cycle enters a path P_i it has to traverse the path in the same direction with no interruption.

Also since $t \to s$ is the only edge entering s it must be contained in every Hamiltonian cycle.

We now add k vertices corresponding to the k clauses $C_1, C_2, \ldots, C_k$.
We use the following example to describe the edges connectiing these vertices to the current vertices.

Assume that $C_i = (x_3 \vee \neg x_9 \vee \neg x_{67})$. We add the edges:
$v_{i,9} \to c_i$, $c_i \to v_{i,10}$, $c_i \to v_{i,27}$, $c_i \leftarrow v_{i,28}$, $c_i \to v_{i,201}$, $c_i \leftarrow v_{i,202}$.

Assume that $F(x_1, x_2, \ldots, x_n)$ is satisfiable. For instance assume that $x_0 = T, x_1 = T, x_2 = F$ etc. and assume that $C_i = (x_0 \vee \neg x_i \vee x_j)$ is a clause which is satisfied by the choice $x_0 = T$.

Assume that $F(x_1, x_2, \ldots, x_n)$ is satisfiable. For instance assume that $x_0 = T, x_1 = T, x_2 = F$ etc. and assume that $C_i = (x_0 \vee \neg x_i \vee x_j)$ is a clause which is satisfied by the choice $x_0 = T$.

Our graph will have the following Hamiltonian cycle:

$$t \to s \to v_{i,3} \to c_i \to v_{i,4} \to \ldots \to v_{i,3k+3} \to \ldots \to t$$

Assume that $F(x_1, x_2, \ldots, x_n)$ is satisfiable. For instance assume that $x_0 = T, x_1 = T, x_2 = F$ etc. and assume that $C_i = (x_0 \vee \neg x_i \vee x_j)$ is a clause which is satisfied by the choice $x_0 = T$.

Our graph will have the following Hamiltonian cycle:

$$t \to s \to v_{i,3} \to c_i \to v_{i,4} \to \ldots \to v_{i,3k+3} \to \ldots \to t$$

Conversely, if the graph is Hamiltonian, for every path P_i traversed by the cycle either the vertex c_i is traversed from right to left (we will assign it the value F) or from left to right (in which case it will be assigned F). It is easy to see that each clause C_i will be satisfied by this assignment.

The TSP is NPC

This is a real easy reduction to the Hamiltonian cycle problem.

The TSP is NPC

This is a real easy reduction to the Hamiltonian cycle problem.

Given a diGraph $D(V, E)$. Construct a TSP instance with "cities" v_i and $\omega(v_i, v_j) = 1$ *if* $(v_i, v_j) \in E(D)$ *and* $\omega(v_i, v_j) = 10$ otherwise. Now ask the TSP blackbox whether this instance has a tour of weight $\leq n$.

The TSP is NPC

This is a real easy reduction to the Hamiltonian cycle problem.

Given a diGraph $D(V, E)$. Construct a TSP instance with “cities” v_i and $\omega(v_i, v_j) = 1$ *if* $(v_i, v_j) \in E(D)$ *and* $\omega(v_i, v_j) = 10$ otherwise. Now ask the TSP blackbox whether this instance has a tour of weight $\leq n$.

Clearly, this TSP instance has a tour of weight n if and only if D is Hamiltonian.

The TSP is NPC

This is a real easy reduction to the Hamiltonian cycle problem.

Given a diGraph $D(V, E)$. Construct a TSP instance with "cities" v_i and $\omega(v_i, v_j) = 1$ *if* $(v_i, v_j) \in E(D)$ *and* $\omega(v_i, v_j) = 10$ otherwise. Now ask the TSP blackbox whether this instance has a tour of weight $\leq n$.

Clearly, this TSP instance has a tour of weight n if and only if D is Hamiltonian.

This brings us back to the first lecture. We started with two problems that looked almost identical.

The TSP is NPC

This is a real easy reduction to the Hamiltonian cycle problem.

Given a diGraph $D(V, E)$. Construct a TSP instance with "cities" v_i and $\omega(v_i, v_j) = 1$ *if* $(v_i, v_j) \in E(D)$ *and* $\omega(v_i, v_j) = 10$ otherwise. Now ask the TSP blackbox whether this instance has a tour of weight $\leq n$.

Clearly, this TSP instance has a tour of weight n if and only if D is Hamiltonian.

This brings us back to the first lecture. We started with two problems that looked almost identical.

And now we learned that one, the assignment problem can be solved effciently while the TSP problem is very difficult.