Introduction to Algorithms

Meng-Tsung Tsai

12/17/2019

NP Complexity Class

# algorithms v.s. # problems

In this class,
we learn how to solve
many problems
in polynomial time.

# algorithms v.s. # problems

In this class,
we learn how to solve
many problems
in polynomial time.

# polynomial-time algorithms: countably infinite many.
# problems: uncountably infinite many.

# algorithms v.s. # problems

In this class,
we learn how to solve
many problems
in polynomial time.

In the real life,
there are more problems
that require
superpolynomial time.

# polynomial-time algorithms: countably infinite many.
# problems: uncountably infinite many.

P Class

It is a class of decision problems, whose output is either "Yes" or "No", and whose running time is a polynomial of the input size.

<u>Example</u>.

Primality Test

Input: a positive integer k (log k bits)

Output: "Yes" if k is a prime, or "No" otherwise.

To prove Primality is in P, we need to give an $O(\log^c k)$-time algorithm for any constant $c \geq 0$. [Proved in 2002]

NP Class

It is a class of decision problems, whose output is either "Yes" or "No".

A problem $L \in NP$ if and only if L satisfies the property that:

$\exists$ a polynomial-time algorithm A and a constant $c \geq 0$

$\forall$ input x of L

$L(x) =$ "Yes" if and only if there exists a certificate y of length $|x|^c$ so that $A(x, y) =$ "Yes".

An example problem in NP

Maximum Independent Set (decision version)

Input: an undirected graph G (n^2 bits) and an integer k (log n bits)

Output: "Yes" if G has an independent set of size k, or "No" otherwise.

certificate: some node subset Y of G (n log n bits)

polynomial-time algorithm:

```
A(G∘k, Y){
    check whether Y is an independent set; // $O(n^2)$ time
    return ($|Y| \geq k$) ? "Yes" : "No"; // $O(1)$ time
}
```

Another example problem in NP

Path (decision version)

Input: an undirected graph G (n^2 bits), a pair of nodes u and v (log n bits), and an integer k (log n bits)

Output: "Yes" if G has a simple path from u to v whose length is no more than k, or "No" otherwise.

Another example problem in NP

Path (decision version)

Input: an undirected graph G (n^2 bits), a pair of nodes u and v (log n bits), and an integer k (log n bits)

Output: "Yes" if G has a simple path from u to v whose length is no more than k, or "No" otherwise.

certificate: some sequence of some nodes in G (n log n bits)

polynomial-time algorithm:

```
A(G○u○v○k, P){
    check whether P is a path from u to v; // O(n) time
    return (|P| ≥ k) ? "Yes" : "No"; // O(1) time
}
```

Yet another example problem in NP

Connectivity (decision version)

Input: an undirected graph G (n^2 bits)

Output: "Yes" if G is conncected, or "No" otherwise.

Yet another example problem in NP

Connectivity (decision version)

Input: an undirected graph G (n^2 bits)

Output: "Yes" if G is conncected, or "No" otherwise.

certificate: $\varnothing$ (0 bits)

polynomial-time algorithm:

```
A(G, ∅){
    runs BFS on G; // O(n^2) time
    return (BFS traverses all nodes in G) ? "Yes" : "No";
    // O(1) time
}
```

NP-hard problems

We say a problem L is NP-hard if L satisfies that:

for every problem L' in NP, there exists a polynomial-time reduction from L' to L.

NP-complete problems

We say a problem L is NP-complete if L satisfies that:

$L \in NP$ and L is NP-hard.

People believe that if a problem is NP-hard, then it requires superpolynomial time to compute. Up to date, people don't know whether this belief is correct or not.

This is the famous problem whether $NP \neq P$.

Why do we define NP in this way?

1. It has some natural (not artificial) complete problems.

2. It asks whether parallelism gives us more power to solve a problem. Though it sounds like an obvious fact, we do not know whether the answer is "Yes" or "No."

3. Indeed, there are many other complexity classes, and NP is just one of them. If interested, google "complexity zoo."

Some known NP-complete problems

1. SAT
2. 3SAT
3. Clique
4. Vertex Cover
5. Maximum Independent Set
6. Hamiltonian Cycle
7. Hamiltonian Path
8. Traveling Salesman Problem

...

Since they are NP-complete problems, each of these problems requires superpolynomial time unless NP = P.

NP-hardness proof

polynomial-time reduction:

Let L_1 be an NP-hard problem. (no P-time solution unless NP=P)
Let L_2 be a problem that we don't know how hard it is.

If there exists an algorithm T that can convert

$L_1(x_1)$ into $L_2(x_2)$ in time polynomial in $|x_1|$ so that $L_1(x_1)=L_2(x_2)$.

By definition, L_2 is an NP-hard problem.

Maximum Independent Set $\in$ NPC

Given Clique $\in$ NPC, then we are able to show that MIS $\in$ NPC.

Clique (decision version)
Input: an undirected graph G and an integer k
Output: whether G has a clique of size $\geq$ k

MIS (decision version)
Input: an undirected graph G and an integer k
Output: whether G has an independent set of size $\geq$ k

A clique is a set of nodes so that every two nodes in it are connected by an edge.

A independent set is a set of nodes so that every two nodes in it do not have a connecting edge.

Maximum Independent Set $\in$ NPC

First, we show that MIS is NP-hard.

```
Clique(G, k){ // O(n^2) bits
    G' ← the complement graph of G; // O(n^2) time
    return MIS(G', k); // O(1) time
}
```

This reduction works because of the fact:
G has clique of size k iff G' has an IS of size k.

Maximum Independent Set $\in$ NPC

It is known that MIS is in NP, as shown on the 8th slide.
Combining MIS is NP-hard and is in NP, we have MIS is NPC.

Hence, MIS is an NP-complete problem. Consequently,
MIS has no P-time solution unless P=NP.

Optimization Problems

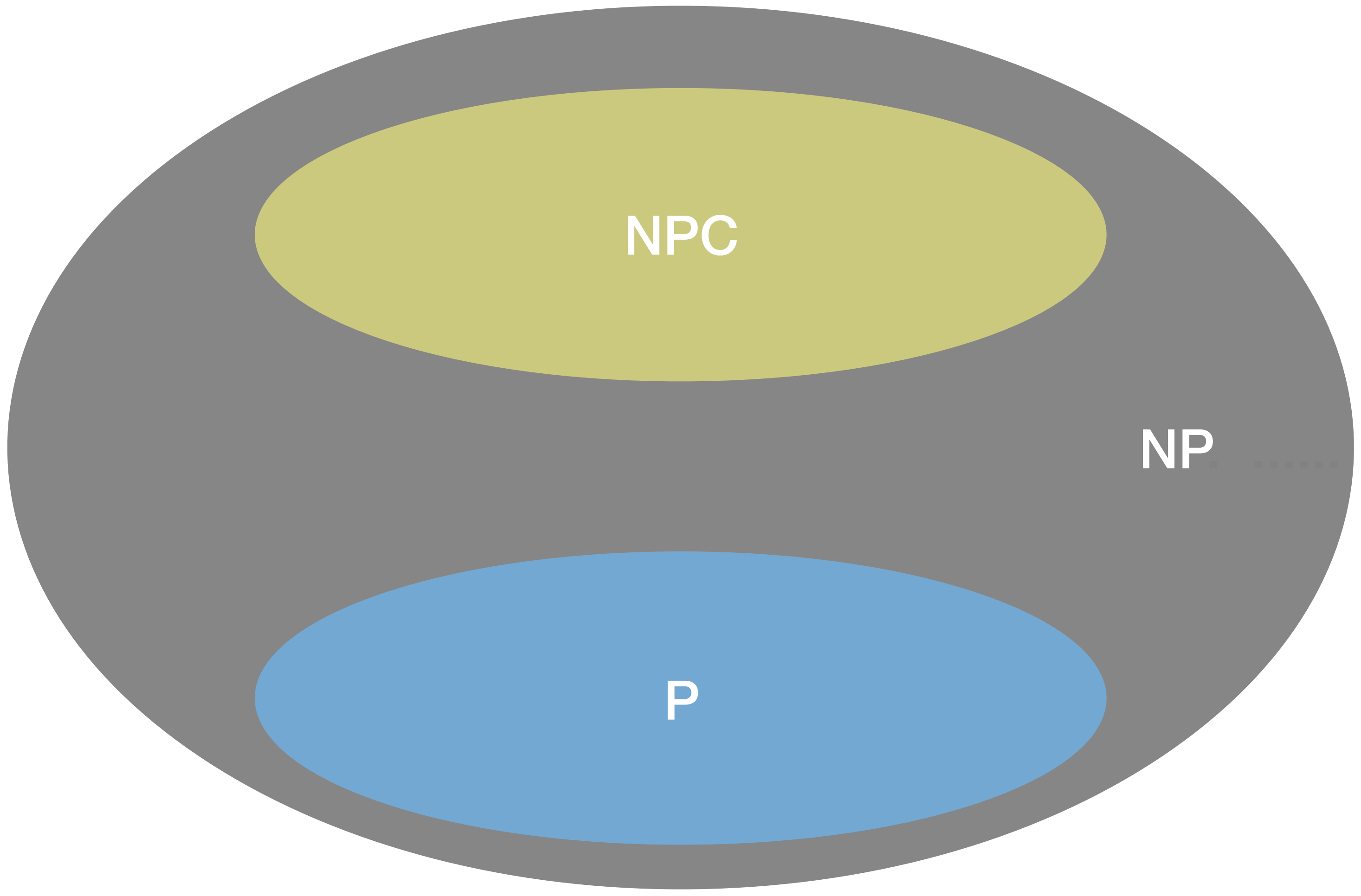
NP class only contain decision problems. For an optimization problem,

MIS_O (optimization version)
Input: an undirected graph G
Output: an integer k so that the maximum independent set of G has size k.

We still can show that MIS_O is NP-hard, but $MIS_O \notin NP$ because it is not a decision problem.

```
MIS(G, k){ // $O(n^2)$ bits
    $\ell \leftarrow MIS_O(G)$; // O(1) time
    return $(\ell \geq k)$ ? "Yes" : "No"; // O(1) time
}
```

The scope of NP, P, and NPC if NP $\neq$ P



Is NP = NPC ∪ P?

Given a new problem L in NP, it either runs in polynomial time or superpolynomial time.

In the former case, we can show that L is in P.

In the later case, can we show that L is in NPC?

Is NP = NPC $\cup$ P?

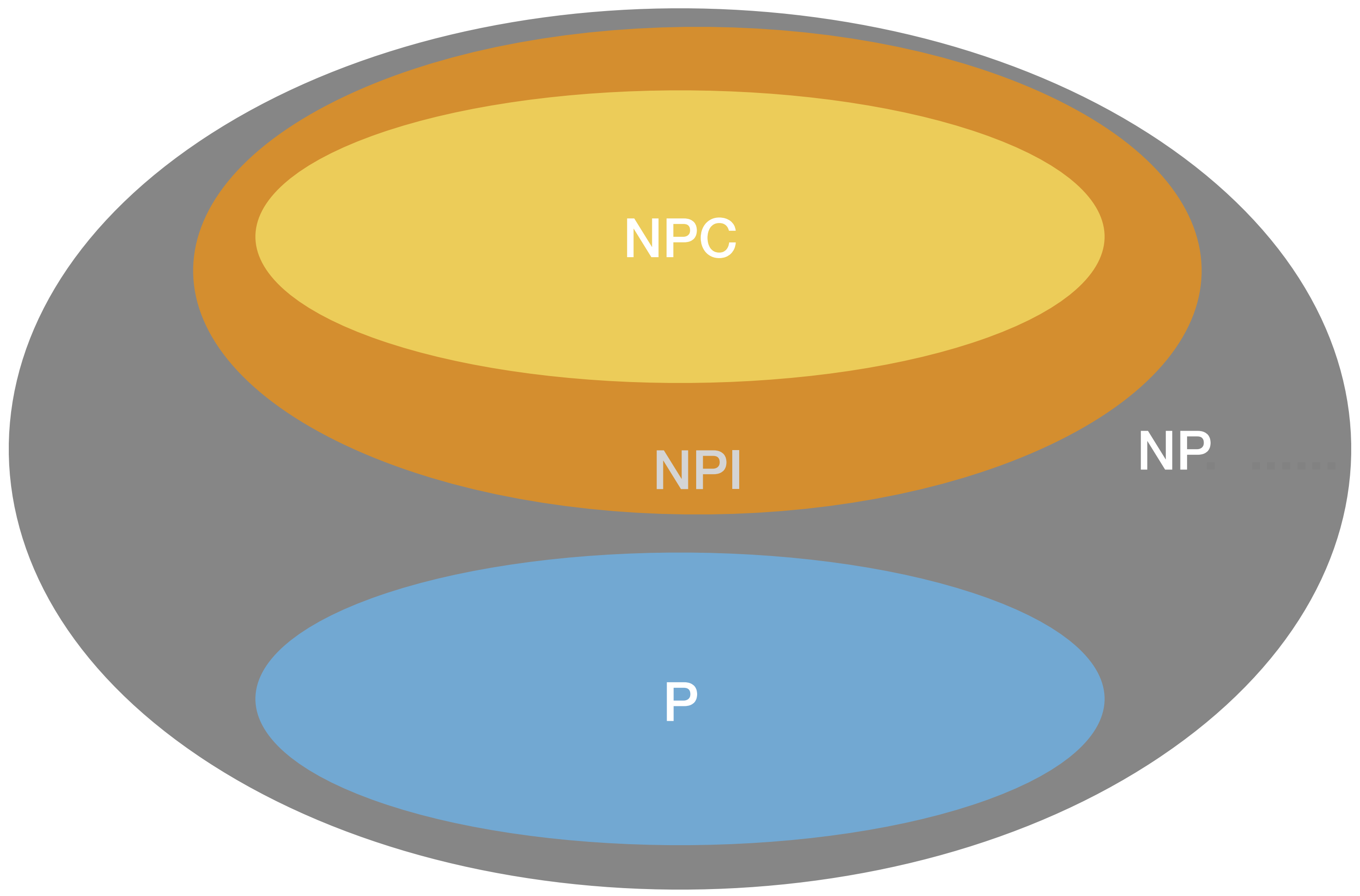
Given a new problem L in NP, it either runs in polynomial time or superpolynomial time.

In the former case, we can show that L is in P.

In the later case, can we show that L is in NPC?

No, NPC is not the class of problems that contains all problems that requires superpolynomial running time.

If NP ≠ P



There is an non-empty class of problems called NP-intermediate, which has an empty intersection with NPC and P.

NPI

People suspected that the following problems might be an NPI problem, but some of them have been shown not in NPI.

(1) Primality Test [Proved in P]

(2) Linear Programming [Proved in P]

(3) Graph Isomorphism [Remains open]