TP Complexité M1 - SII

TP6: NP-complete Problems

Teachers:

- Belkadi Widad Hassina (G1-G2): belkadi.wh.usthb@gmail.com

Types of problems

Decision Problems

- Input: problem
- Output: yes/no

Optimisation Problems

- Input: problem
- Output: best solution

Counting Problems

- Input: problem
- Output: number of solutions

Enumeration Problems

- Input: problem
- Output: all solutions

Types of problems - Examples

Problem P1
Instance: A set S of N numbers, and an integer k.
Question : How many values of S are greater than k.

Problem P2
Instance: A graph G = (S, A), and an integer k.
Question: Is there a cycle of length equal to k?

Problem P3
Instance: A graph G = (S, A).
Question : What is the chromatic number of the graph G?

Problem P4
Instance: A graph G = (S, A).
Question: What is the size of the longest cycle of G?

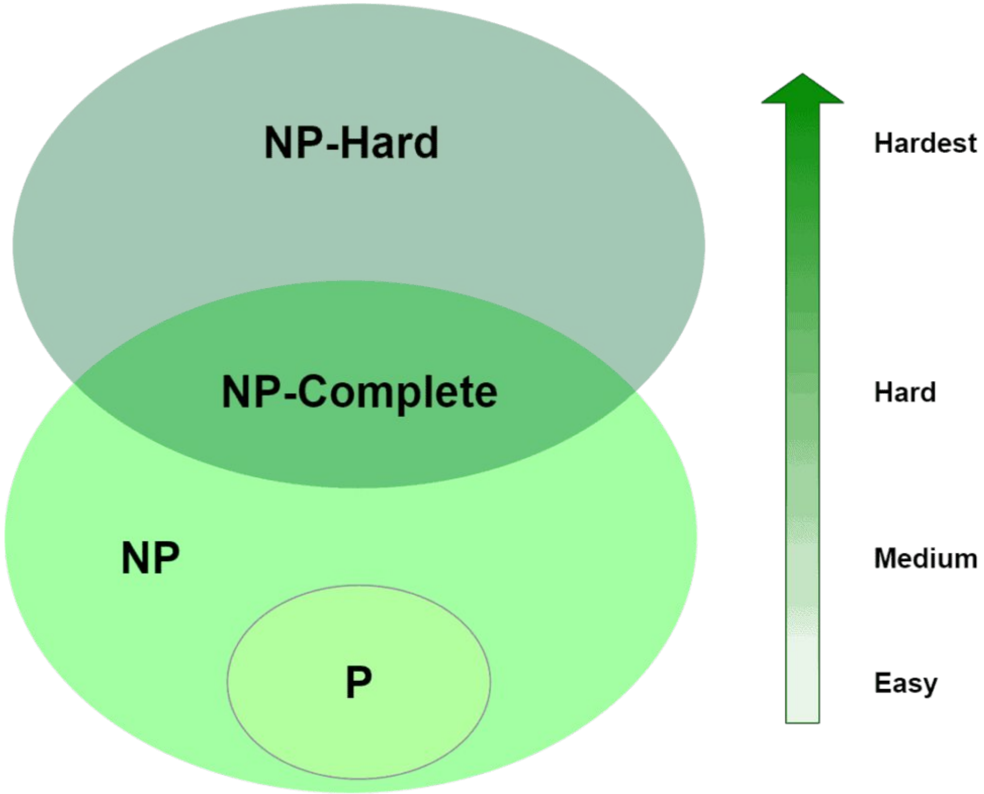
Types of problems - Examples

Problem P1
Instance: A set S of N numbers, and an integer k.
Question : How many values of S are greater than k.
=> Counting problem (dénombrement)
Problem P2
Instance: A graph G = (S, A), and an integer k.
Question: Is there a cycle of length equal to k?
=> Decision problem

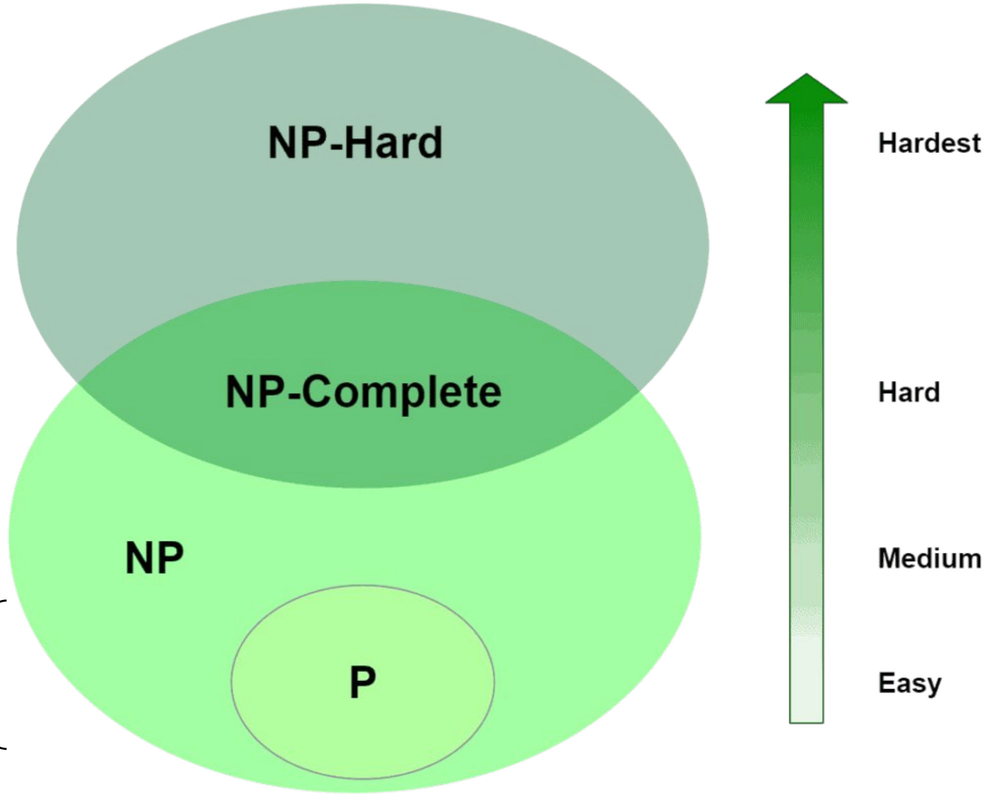
Problem P3
Instance: A graph G = (S, A).
Question : What is the chromatic number of the graph G?
=> Optimisation problem

Problem P4
Instance: A graph G = (S, A).
Question: What is the size of the longest cycle of G?
=> Optimisation problem

Classes of Problems - Difficulty of the problems

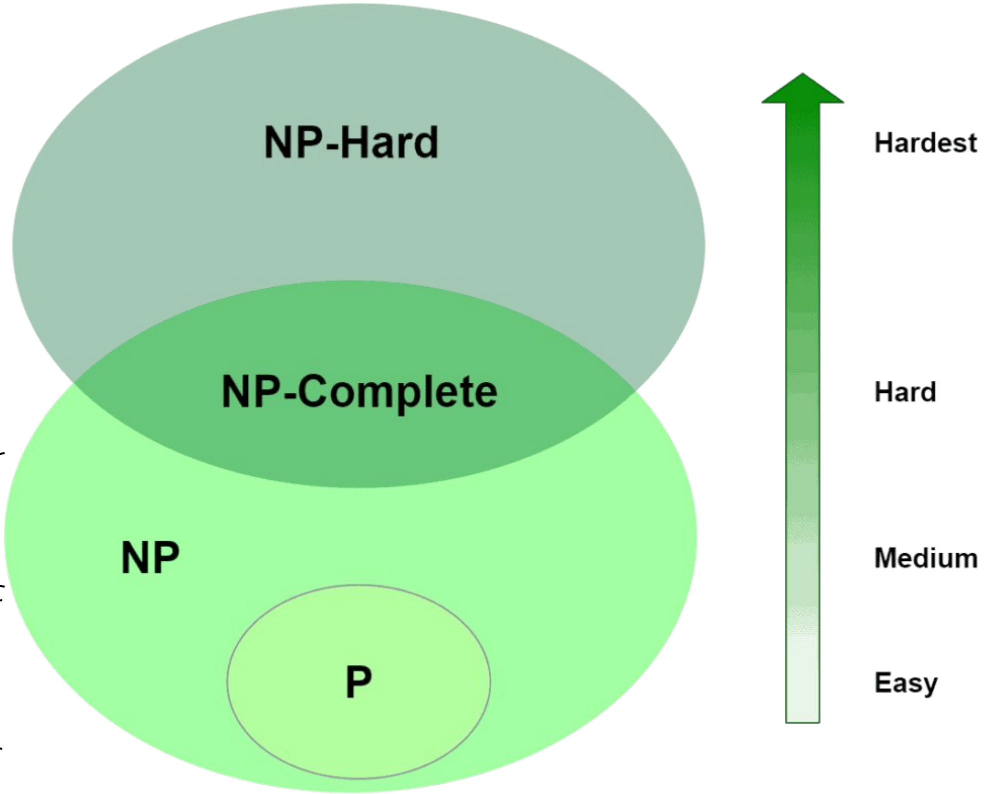


Classes of Problems - Difficulty of the problems



Decision problems
Verification + Resolution in
Polynomiale time.

Classes of Problems - Difficulty of the problems



Decision problems
Verification in Polynomiale time.

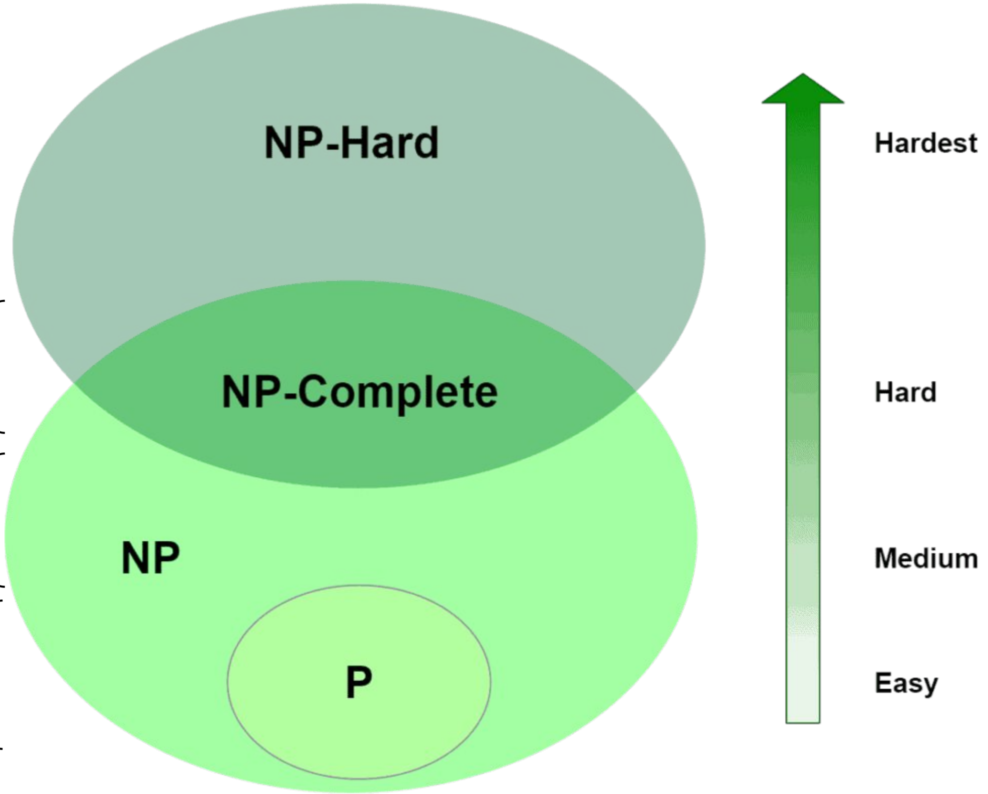
Decision problems
Verification + Resolution in Polynomiale time.

Classes of Problems - Difficulty of the problems

Decision problems
NP + At least as difficult as Any NP problem

Decision problems
Verification in Polynomiale time.

Decision problems
Verification + Resolution in Polynomiale time.



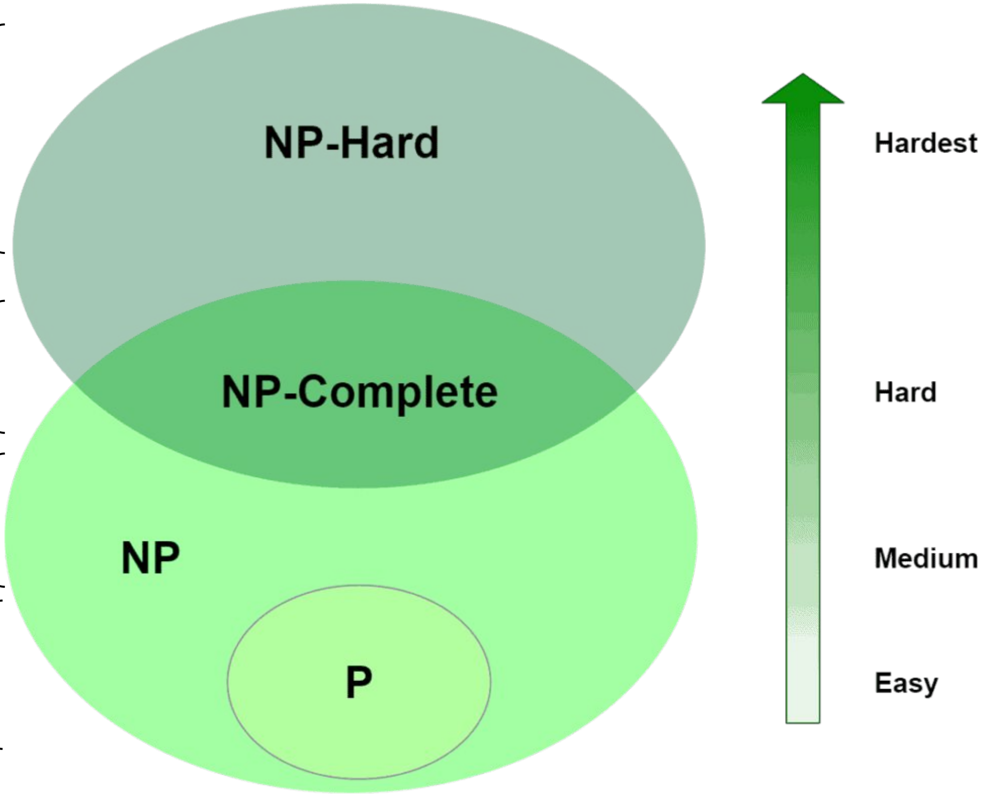
Classes of Problems - Difficulty of the problems

Optimisation / Counting/ Enumeration problems
NP + At least as difficult as Any NP problem.

Decision problems
NP + At least as difficult as Any NP problem.

Decision problems
Verification in Polynomiale time.

Easy problems
Verification + Resolution in Polynomiale time.



Classes of problems - Examples

Problem P1
Instance: A set S of N numbers, and an integer k.
Question : How many values of S are greater than k.
=> Counting problem (dénombrement)
Problem P2
Instance: A graph G = (S, A), and an integer k.
Question: Is there a cycle of length equal to k?
=> Decision problem

Problem P3
Instance: A graph G = (S, A).
Question : What is the chromatic number of the graph G?
=> Optimisation problem

Problem P4
Instance: A graph G = (S, A).
Question: What is the size of the longest cycle of G?
=> Optimisation problem

Classes of problems - Examples

Problem P1
Instance: A set S of N numbers, and an integer k.
Question : How many values of S are greater than k.
=> Counting problem (dénombrement) + **in P class**

Problem P2
Instance: A graph G = (S, A), and an integer k.
Question: Is there a cycle of length equal to k?
=> Decision problem + **in NP complete class**

Problem P3
Instance: A graph G = (S, A).
Question : What is the chromatic number of the graph G?
=> Optimisation problem + **in NP hard class**

Problem P4
Instance: A graph G = (S, A).
Question: What is the size of the longest cycle of G?
=> Optimisation problem + **in NP hard class**

How to proof that a problem is in NP class?

1- Say that the problem is an intractable decision problem (Yes / No answer) (solve with DFS or BFS in **exponential** time).

2- Find a non-deterministic algorithm to generate a potential solution. (Find an algo to generate a random solution in polynomial time).

3- Write an algorithm to verify a solution with polynomial complexity.

How to proof that a problem is in P class?

1- Prove that X is **NP**.

2- We can design a deterministic algorithm to solve the problem in **polynomial** time.

Example:

- Problem: Determine whether the given input is a prime number.
- Explanation: The problem of checking if a number n is prime can be solved using efficient algorithms, which operates in polynomial time. Thus, it belongs to the class P, as it can be resolved deterministically within a time complexity that is a polynomial function of the size of the input.

How to proof that a problem is in NP complete ?

1- Prove that X is **NP**.
2- Prove that any NP problem can be transformed via a **polynomial reduction** to X.

Example:

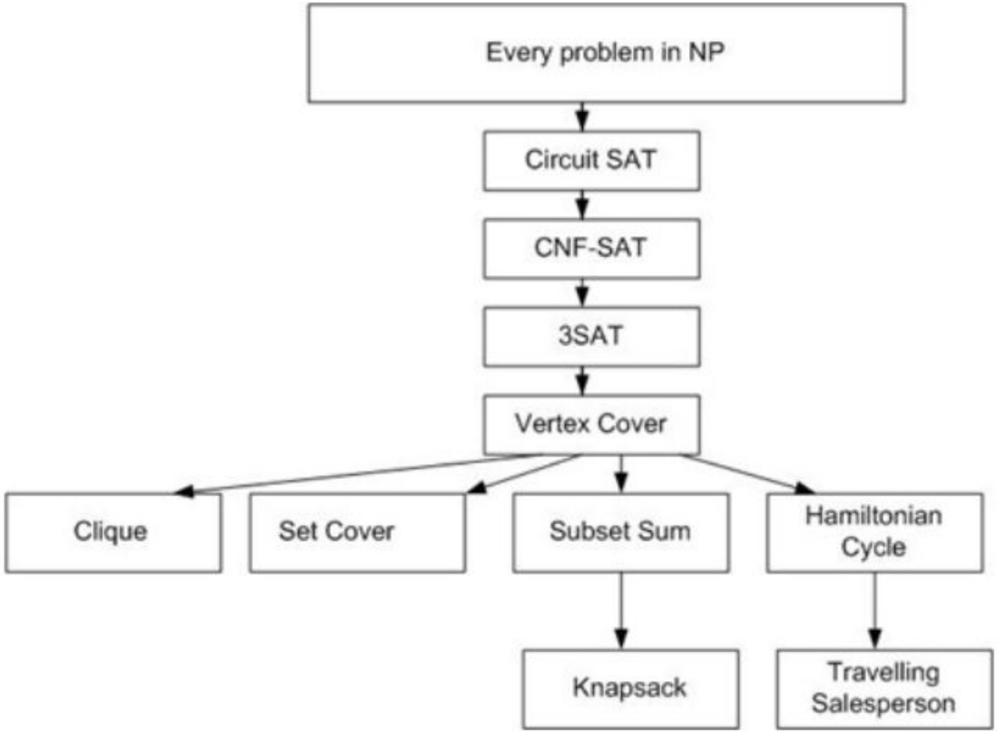
- Problem: The Boolean Satisfiability Problem (SAT).
- Explanation: The SAT problem asks whether there exists an assignment of truth values (true/false) to variables in a given Boolean formula such that the formula evaluates to true. It was **the first problem proven** to be NP-complete (Cook-Levin theorem).
    - SAT is in NP because verifying a given solution is feasible in polynomial time,
    - and it is NP-complete because any problem in NP can be reduced to SAT in polynomial time.

How to proof that a problem is in NP complete ?

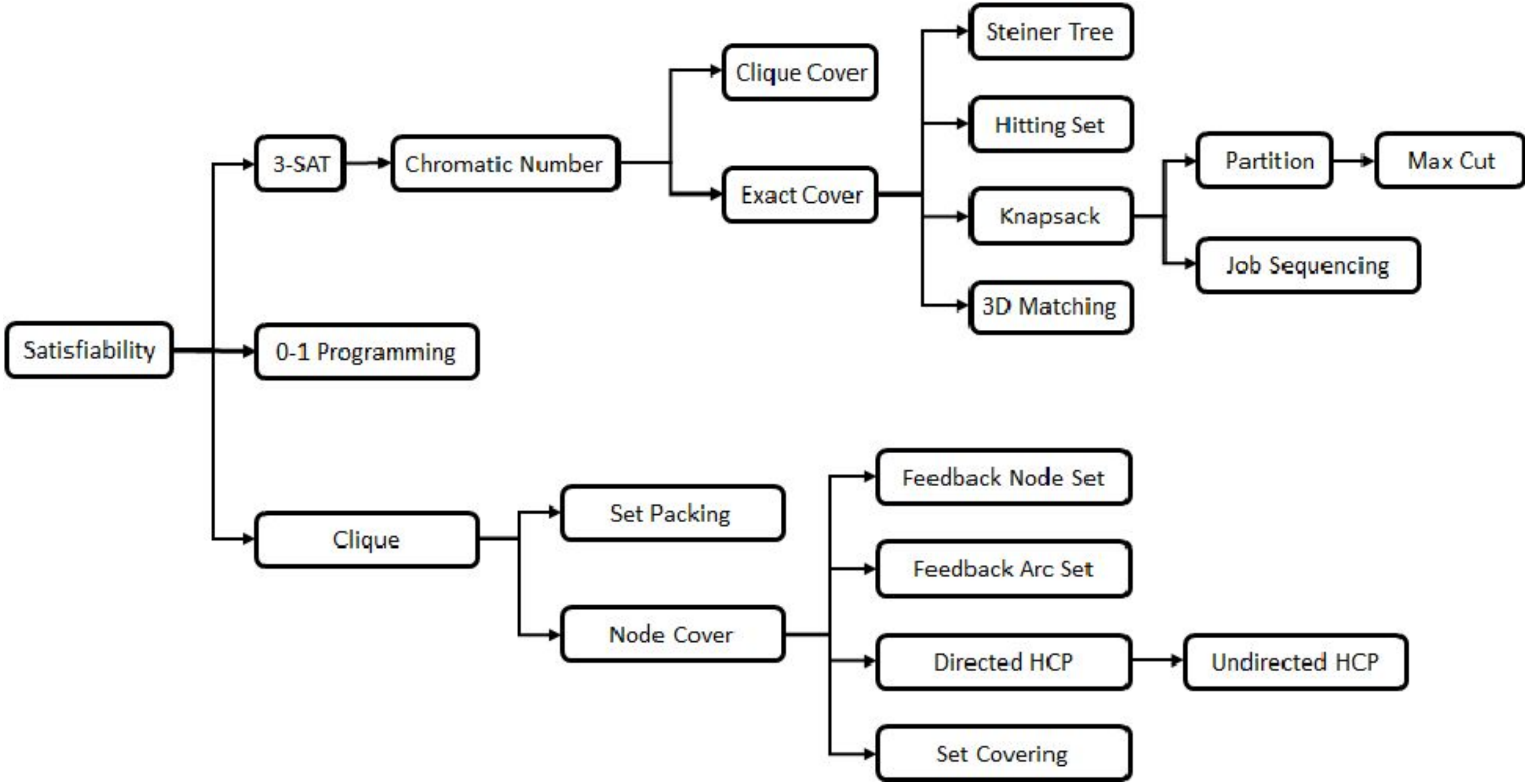
We know that SAT is NP-Complete
⇒ it's the 1st proven NP-Complete problem.

So if we can go from SAT to 3-SAT in polynomial time ⇒ we'll have proved that 3-SAT is NP-Complete
...

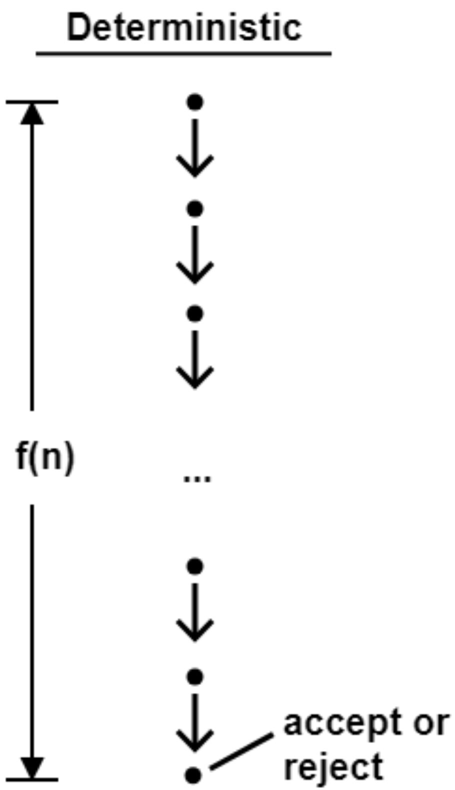
and so on ... we can prove that the clique problem is NP-Complete through 3-SAT ...



How to proof that a problem is in NP complete ?



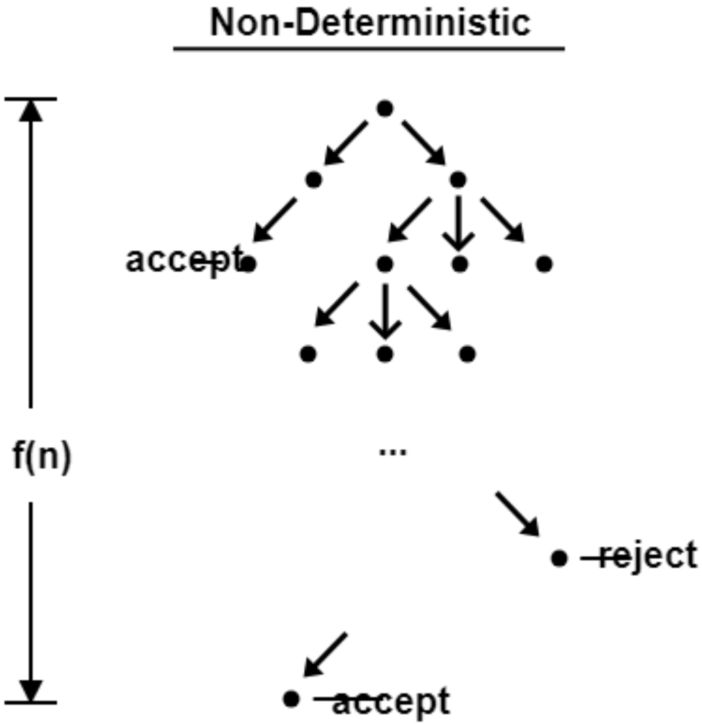
Resolution Techniques



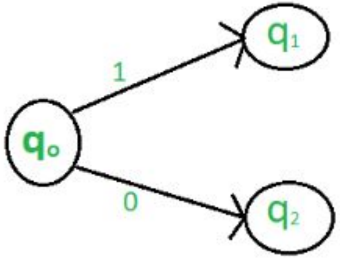
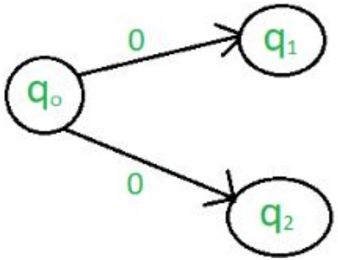
- **Exacte** Approach
- Given a particular input the algorithm will always give same output.
- Polynomial Deterministic Resolution ⇒ for problems in P class.
- Such as DFS, BFS...

Resolution Techniques

- Approximate Approach
- Given a particular input, the algorithm can give different outputs.
- Non-Polynomial Deterministic Resolution ⇒ for problems in NP class.
- In semester 2, you are going to learn Meta-heuristics, one good example of non deterministic algorithms..



Resolution Techniques

Deterministic	Non Deterministic
Given a particular input the algorithm will always give same output.	In this case, the algorithm can give different outputs.
	
can solve the problem in polynomial time (P).	can not solve the problem in polynomial time.
Can determine what is the next step.	Can not determine.
DFS, BFS, Dynamic programming	Heuristic (Simulated Annealing…) and metaheuristic based (Genetic Algorithms…)

TP instructions (A simple template for the project)

Problem: The travelling salesman's problem

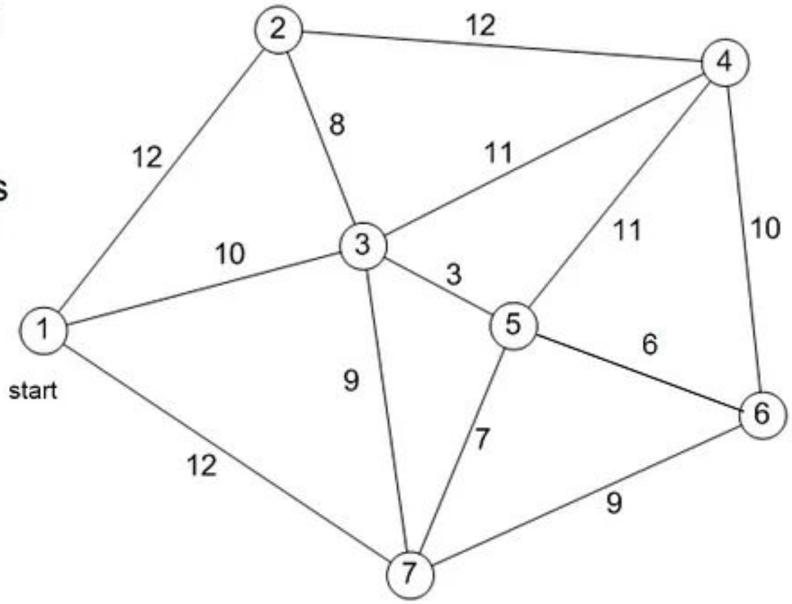
Instance: S is a set of n cities {S1, S2, ... Sn} and each pair of cities is separated by a certain distance Dij = distance(Si, Sj). A distance constraint **Dmax**.

Question: Is there a cycle that passes through each city once and only once, such that the sum of the distances covered is less than **Dmax**?

TP instructions (A simple template for the project)

Problem: The travelling salesman's problem

- Starting from city 1, the salesman must travel to all cities once before returning home
- The distance between each city is given, and is assumed to be the same in both directions
- Only the links shown are to be used
- Objective - Minimize the total distance to be travelled



TP instructions (A simple template for the project)

Typical questions / steps of resolution:

1. Illustrate an example of **a correct and an incorrect solution**.
2. Give the appropriate **data structures** to represent a solution.
3. What criteria must a given solution S satisfy to be valid?
4. Propose an algorithm that **generates a random solution** to the problem.
5. Propose an algorithm for validating a given solution S' and calculate its complexity.
6. Implement the resolution algorithm (deterministic), such as DFS.
7. Roughly estimate **the size of the solution tree** and deduce the order of **complexity** of the resolution algorithm.
8. Deduce the classification associated with the problem studied. Justify your answer.

TP instructions (A simple template for the project)

Typical questions / steps of resolution:

<u>Notes:</u>

- In the project you **have to** do experiments by varying the size of the problem.