Algorithmics and Complexity
Cours 6/7 : Theory of complexity

CentraleSupélec – Gif

ST2 – Gif

Problems in algorithmics

We saw:

- decision problems (existence of a path,. . .);
- optimization problems (minimum spanning tree, maximum flow, sequence alignment, . . .);
- algorithms running in polynomial time $\mathcal{O}(n^c)$ with n the size of the instance and c a constant number.

Problems in algorithmics

We saw:

- decision problems (existence of a path,. . .);
- optimization problems (minimum spanning tree, maximum flow, sequence alignment, . . .);
- algorithms running in polynomial time $\mathcal{O}(n^c)$ with n the size of the instance and c a constant number.

Remaining questions

- What can be computed?

Problems in algorithmics

We saw:

- decision problems (existence of a path,...);
- optimization problems (minimum spanning tree, maximum flow, sequence alignment, ...);
- algorithms running in polynomial time $\mathcal{O}(n^c)$ with n the size of the instance and c a constant number.

Remaining questions

- What can be computed?
- What can be computed effectively?

  $\longrightarrow$ Is there always a polynomial time algorithm for any problem?

Problems in algorithmics

We saw:

- decision problems (existence of a path,...);
- optimization problems (minimum spanning tree, maximum flow, sequence alignment, ...);
- algorithms running in polynomial time $\mathcal{O}(n^c)$ with n the size of the instance and c a constant number.

Remaining questions

- What can be computed?
- What can be computed effectively?
  $\longrightarrow$ Is there always a polynomial time algorithm for any problem?
- How to formalize the notion of complexity?
- How to classify problems by complexity?

Plan

1. The Turing Machine
   - Definition
   - Class *P*
2. Class *NP*
3. Polynomial-time reduction
4. Complements

Model of computer: the Turing Machine, 1936

Resources

1. a data file as input
2. the program instructions
3. a memory
4. registers and execution stack
5. a data file as output

Model of computer: the Turing Machine, 1936

Resources

1. a data file as input
2. the program instructions
3. a memory
4. registers and execution stack
5. a data file as output

The Turing Machine

- The *Turing Machine* is a formalization of this structure.

Model of computer: the Turing Machine, 1936

Resources $\longrightarrow$ Formalization

1. a data file as input — an input tape
2. the program instructions
3. a memory
4. registers and execution stack
5. a data file as output

The Turing Machine

- The *Turing Machine* is a formalization of this structure.

Model of computer: the Turing Machine, 1936

Resources →	Formalization
1. a data file as input	an input tape
2. the program instructions	a table of rules
3. a memory	
4. registers and execution stack	
5. a data file as output	

The Turing Machine

- The *Turing Machine* is a formalization of this structure.

Model of computer: the Turing Machine, 1936

Resources	$\longrightarrow$	Formalization
1. a data file as input		an input tape
2. the program instructions		a table of rules
3. a memory		a work tape
4. registers and execution stack		
5. a data file as output		

The Turing Machine

- The *Turing Machine* is a formalization of this structure.

Model of computer: the Turing Machine, 1936

Resources	$\longrightarrow$	Formalization
1. a data file as input		an input tape
2. the program instructions		a table of rules
3. a memory		a work tape
4. registers and execution stack		a register
5. a data file as output		

The Turing Machine

- The *Turing Machine* is a formalization of this structure.

Model of computer: the Turing Machine, 1936

Resources	⟶	Formalization
1. a data file as input		an input tape
2. the program instructions		a table of rules
3. a memory		a work tape
4. registers and execution stack		a register
5. a data file as output		an output tape

The Turing Machine

- The *Turing Machine* is a formalization of this structure.

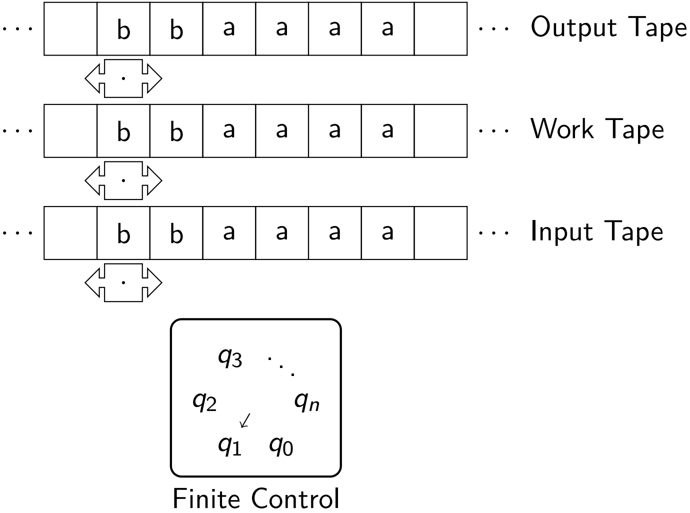
Model of computer: the Turing Machine, 1936

Resources	$\longrightarrow$	Formalization
1. a data file as input		an input tape
2. the program instructions		a table of rules
3. a memory		a work tape
4. registers and execution stack		a register
5. a data file as output		an output tape

The Turing Machine

- The *Turing Machine* is a formalization of this structure.
- There are many definitions/variants of the Turing Machine (number of tapes, alphabet. . .).

Turing Machine Illustration



Principle of the Turing Machine, 1936

Each machine has:

- a register that stores the current state (the number of possible states is finite);
- three tapes (input, work and output tape) divided into cells storing symbols (finite alphabet);
- three heads that can read or write on the tapes and move left or right to the next cell on a tape;

Principle of the Turing Machine, 1936

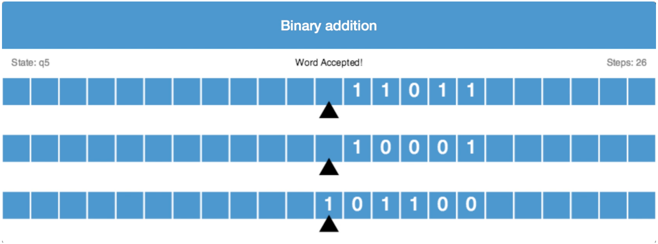
Each machine has:

- a register that stores the current state (the number of possible states is finite);
- three tapes (input, work and output tape) divided into cells storing symbols (finite alphabet);
- three heads that can read or write on the tapes and move left or right to the next cell on a tape;
- a table of rules/actions which, depending on:
  - the current state
  - the values read on tapes

  indicates :
  - what symbol to write on each tape
  - how to move the heads (left/right)
  - what is the next state.

Demo

- Addition of 27 and 17 in binary : 11011#10001
- Simulator of Turing Machine available online : `https://turingmachinesimulator.com/`



The Church Turing thesis

- Other models of computation exist. So far all were simulated by a Turing Machine.

The Church Turing thesis

- Other models of computation exist. So far all were simulated by a Turing Machine.

The Church Turing thesis

Any physical computing system (based on silicon, DNA, neurons or any other alien technology) can be simulated by a Turing Machine.

- This is not a theorem, only a widely accepted theory
- It implies that what can be computed doesn't depend on the model of computation we use

Limits of this model

We cannot compute everything using a Turing Machine.

Undecidable problems

There are some functions that are not computable by any Turing Machine.

Example, halting problem (Turing, Church, 1936)

There is no Turing Machine M which takes as input a Turing Machine M' and determines whether M' halts or not.

- verify whether a piece of code terminates is undecidable

Runtime of a Turing Machine

How to measure the execution time of a TM?

- Each action of the machine is a step (reading, writing on a tape, moving a head)

Runtime of a Turing Machine

How to measure the execution time of a TM?

- Each action of the machine is a step (reading, writing on a tape, moving a head)
- Computation requires reading the whole entry, we then measure the execution time according to the size of the entry.
  - For an entry x, we note $|x|$ the size of x.

Runtime of a Turing Machine

How to measure the execution time of a TM?

- Each action of the machine is a step (reading, writing on a tape, moving a head)
- Computation requires reading the whole entry, we then measure the execution time according to the size of the entry.
  - For an entry x, we note $|x|$ the size of x.
- A TM computes a function f in time $T(n)$ if the computation of $f(x)$ of any entry x such that $n = |x|$ requires at most $T(|x|)$ steps.

The class P

Definition of P

The class P is the set of problems that can be solved by Turing Machines in polynomial time $poly(n)$.

The class P

Definition of P

The class P is the set of problems that can be solved by Turing Machines in polynomial time $poly(n)$.

Simple Turing Machines vs complex ones

- Any function f computable by a Turing Machine M with k tapes and an alphabet Γ in time $T(n)$ can be computed by a Turing Machine $\tilde{M}$ with a single tape and a binary alphabet in time $poly(T(n))$.
- The class P does not depend on the Turing Machine model we consider.

Wide questions

Importance of P

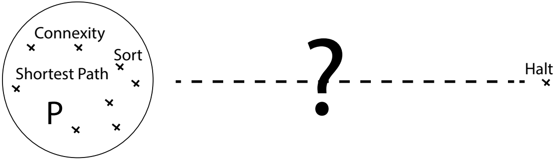
- P is the class/set of easy problems.

Criticism about P

- Although one can question the efficiency of an algorithm with a complexity of $\mathcal{O}(n^{100})$, in practice complexity does not exceed $\mathcal{O}(n^5)$ for problems in P.
- Worst case analysis can be too restrictive
- Other computing models should be investigated (quantum computing, randomized computing).

Issue

- P is the class of problems for which we have efficient algorithms.
- There is a class of problems for which there is no algorithm.
- Is there something in between ?

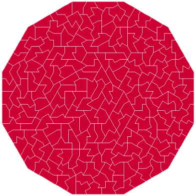


Plan

Eternity

Eternity puzzle

- Puzzle of 209 pieces, released in june 1999, with a reward of £1'000'000.
- Two mathematicians of Cambridge won in October 2000.



Verifying vs Solving

- In Eternity, it is simple to verify that a solution is correct but it is extremely difficult to solve it.
- The *NP*-complete problem have a similar property.

Decision problems

Definition

A decision problem divides the set D of instances into two sub-sets:

- D^+ of positive instances (for which the answer is true);
- D^- of negative instances (for which the answer is false).

Solving a decision problem is to determine, given $I \in D$, if $I \in D^+$ (or if $I \in D^-$).

Class NP : Intuition

Intuition

The set of decision problems for which:
Each positive instance $I \in D^+$ has a solution S that can be checked/verified by a polynomial-time algorithm

Class NP : Intuition

Intuition

The set of decision problems for which:
Each positive instance $I \in D^+$ has a solution S that can be checked/verified by a polynomial-time algorithm

- The checking algorithm receives as input a solution and answers yes to the decision problem

Class *NP* : Intuition

Intuition

The set of decision problems for which:
Each positive instance $I \in D^+$ has a solution S that can be checked/verified by a polynomial-time algorithm

- The checking algorithm receives as input a solution and answers yes to the decision problem
- The checking algorithm is polynomial
- No other constraint on the solving algorithm: it can be exponential!

  (which answers yes/no to the problem without being given a solution)

Class NP : Intuition

Intuition

The set of decision problems for which:
Each positive instance $I \in D^+$ has a solution S that can be checked/verified by a polynomial-time algorithm

- The checking algorithm receives as input a solution and answers yes to the decision problem
- The checking algorithm is polynomial
- No other constraint on the solving algorithm: it can be exponential!

(which answers yes/no to the problem without being given a solution)

To illustrate!

we can understand and verify the proof of a theorem, even if it would be difficult to find the proof by ourselves.

skip formal definition

Class *NP* : Formal definition

Definition of *NP*

A decision problem belongs to *NP* if it exists a polynomial time binary relation R and a polynomial p such that :

$$I \in D^+ \Leftrightarrow \exists x.\ R(I, x) \text{ and } |x| \leq p(|I|)$$

Remarks

- x is a *certificate* proving that the instance I is positive.
- R is computable in polynomial time by a Turing Machine.
- We only consider the positive instances.
- *NP* means "Nondeterministic Polynomial Time Turing Machine", it comes from the original definition of the class.

Relations between P and NP

$$P \overset{?}{\subseteq} NP$$

$$P \overset{?}{=} NP$$

$$P \overset{?}{\supseteq} NP$$

Relations between *P* and *NP*

obvious!

$$P \subseteq NP$$

- The solving algorithm is a checking algorithm:
    - It produces a certificate (any solution) and answers yes to the decision problem

Relations between P and NP

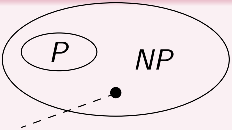
obvious!

$$P \subseteq NP$$

- The solving algorithm is a checking algorithm:
  - It produces a certificate (any solution) and answers yes to the decision problem

conjecture!

$$P \subsetneq NP$$



- Find an NP problem and prove it not in P
- $P \neq NP$ is the most important conjecture of the computer science

Example of problems in NP 1/2

Stable

Instance :

- $G = (V, E)$ a graph ;
- $k \in \mathbb{N}$

Question : is there a stable S (independent set) such that:

- $S \subseteq V$ (sub-graph) with $|S| \geq k$
- $\forall u, v \in S.\ \{u, v\} \notin E$ (the sub-graph induced does not contain any edge)

Stable is in NP

We can check whether a solution $S \subseteq V$ (certificate) is a stable of size greater than or equal to k in polynomial time w.r.t the size of G.

Example of problems in *NP* 2/2

Non-prime number

Instance :

- $x \in \mathbb{N}$

Question : is x a non-prime number?

The non-prime problem is in *NP*

- There is a polynomial size certificate (n, m) with $n \leq m < x$.
- We verify in polynomial time (in the size of x) that $x = n \times m$.

List of problems in *NP*

Problems in $P \subseteq NP$

shortest path, minimum spanning tree, max flow

Problems in *NP*

Stable, knapsack, sudoku

List of problems in NP

Problems in $P \subseteq NP$

shortest path, minimum spanning tree, max flow

Problems in NP

Stable, knapsack, sudoku

- In practice, for those NP problems of the second list, we failed to find a P solving algorithm

- Most of researchers believe that $P \neq NP$

How to deal with NP problems which seem not to be in P ?

Can we classify these problems ?

Plan

Clique problem

Clique

Instance :

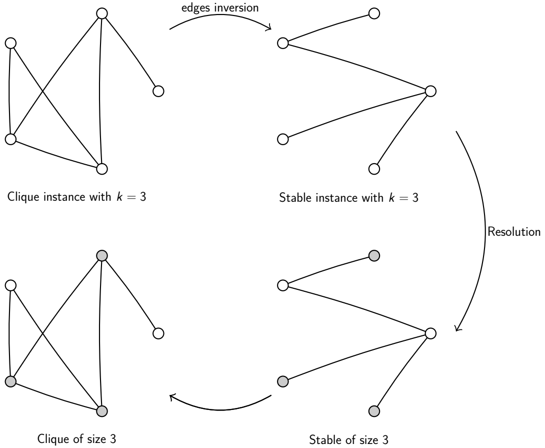
- $G = (V, E)$ a graph ;
- $k \in \mathbb{N}$

Question : is there a clique S such that:

- $S \subseteq V$ (sub-graph) with $|S| \geq k$
- $\forall u, v \in S.\ \{u, v\} \in E$ (the sub-graph induced is complete)

Clearly this problem belongs to NP.

Solving Clique using Stable 1/2



Solving Clique using Stable 2/2

Consequence of the previous algorithm

- The previous transformation can be done in polynomial time.
  If there is a polynomial time algorithm to solve stable then we can solve clique in polynomial time!

Solving Clique using Stable 2/2

Consequence of the previous algorithm

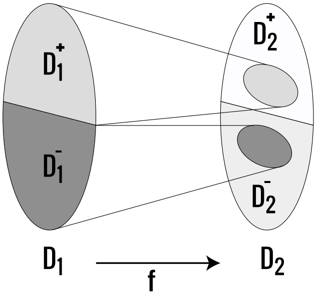
- The previous transformation can be done in polynomial time.
  If there is a polynomial time algorithm to solve stable then we can solve clique in polynomial time!
- This notion is polynomial time reduction

Polynomial time reduction $D_1 \leq D_2$

Given $D_1 = (D_1^+, D_1^-)$ and $D_2 = (D_2^+, D_2^-)$ two decision problems.

We say that D_1 is reduced to D_2 using a Karp reduction ($D_1 \leq_K D_2$) if it exists a function $f : D_1 \rightarrow D_2$ such that :

- $I \in D_1^+ \iff f(I) \in D_2^+$;
- f is computable in polynomial time.



NP-completeness

NP-hard

A decision problem D is NP-hard if $D' \leq D, \forall D' \in NP$.

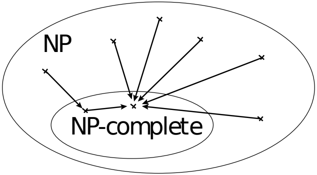
NP-complete

A decision problem D is NP-complete if D is NP-hard and $D \in NP$.

Theorem

- If $D \leq D'$ and $D' \leq D''$ then $D \leq D''$.
- If D is NP-hard and $D \in P$ then $P = NP$.
- If D is NP-complete then: $D \in P$ iff $P = NP$.

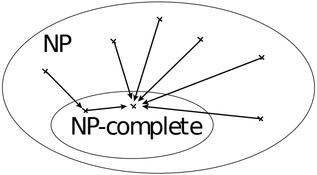
NP-completeness



Corollary

We can reduce any decision problem of *NP* into an *NP*-complete problem.

NP-completeness



Corollary

We can reduce any decision problem of *NP* into an *NP*-complete problem.

is there an *NP*-complete problem?

Boolean satisfaction

SAT

Entry: a CNF formula (Conjunctive Normal Form)

- A set U of variables
- A collection C of disjunctive clauses of literals, where each literal is a variable or the negation of a variable.

$$(U_1 \vee U_2 \vee \neg U_3) \wedge (\neg U_1 \vee \neg U_4) \wedge (U_1 \vee U_2 \vee \neg U_5 \vee U_4)$$

Boolean satisfaction

SAT

Entry: a CNF formula (Conjunctive Normal Form)

- A set U of variables
- A collection C of disjunctive clauses of literals, where each literal is a variable or the negation of a variable.

Question : Is there an assignment of values to variables such that all clauses are true?

$$(U_1 \vee U_2 \vee \neg U_3) \wedge (\neg U_1 \vee \neg U_4) \wedge (U_1 \vee U_2 \vee \neg U_5 \vee U_4)$$

Boolean satisfaction

SAT

Entry: a CNF formula (Conjunctive Normal Form)

- A set U of variables
- A collection C of disjunctive clauses of literals, where each literal is a variable or the negation of a variable.

Question : Is there an assignment of values to variables such that all clauses are true?

$$(U_1 \vee U_2 \vee \neg U_3) \wedge (\neg U_1 \vee \neg U_4) \wedge (U_1 \vee U_2 \vee \neg U_5 \vee U_4)$$

a first set of solutions. . .

Boolean satisfaction

SAT

Entry: a CNF formula (Conjunctive Normal Form)

- A set U of variables
- A collection C of disjunctive clauses of literals, where each literal is a variable or the negation of a variable.

Question : Is there an assignment of values to variables such that all clauses are true?

$$(U_1 \vee U_2 \vee \neg U_3) \wedge (\neg U_1 \vee \neg U_4) \wedge (U_1 \vee U_2 \vee \neg U_5 \vee U_4)$$

a second set of solutions. . .

Cook-Levin theorem, 1971

Theorem

- SAT is *NP*-complete

Cook-Levin theorem, 1971

Theorem

- SAT is NP-complete

Sketch of the proof

- Proving that SAT belongs to NP is obvious
- We admit that SAT is NP-hard :
  Given a problem $D \in NP$ and a Turing Machine M solving D. For any instance I of D, it is possible to build in polynomial time a SAT formula $\varphi(I)$ which evaluates to true if and only if M verifies I.

is there other NP-Complete problems?

Proving stable is NP-complete

Stable — recall

Instance :

- $G = (V, E)$ a graph ;
- $k \in \mathbb{N}$

Question : is there a stable S of size greater than or equal to k ?

How to prove that Stable is NP-complete?

Proving stable is NP-complete

Stable recall

Instance :

- $G = (V, E)$ a graph ;
- $k \in \mathbb{N}$

Question : is there a stable S of size greater than or equal to k ?

How to prove that Stable is NP-complete?

- Prove it in NP (done)

Proving stable is NP-complete

Stable — recall

Instance :

- $G = (V, E)$ a graph ;
- $k \in \mathbb{N}$

Question : is there a stable S of size greater than or equal to k ?

How to prove that Stable is NP-complete?

- Prove it in NP (done)
- Prove that it is NP-hard?

Proving stable is NP-complete

Stable (recall)

Instance :

- $G = (V, E)$ a graph ;
- $k \in \mathbb{N}$

Question : is there a stable S of size greater than or equal to k ?

How to prove that Stable is NP-complete?

- Prove it in NP (done)
- Prove that it is NP-hard?
  We may reduce one NP-complete problem to Stable as polynomial reduction is transitive.

SAT $\leq$ Stable

We consider a CNF of k clauses:

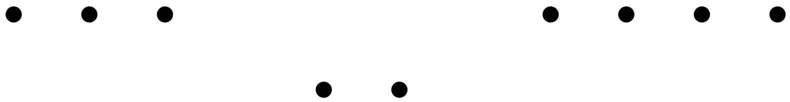
$$(U_1 \lor U_2 \lor \neg U_3) \quad \bigwedge \quad (\neg U_1 \lor \neg U_4) \quad \bigwedge \quad (U_1 \lor U_2 \lor \neg U_5 \lor U_4)$$

Method

SAT $\leq$ Stable

We consider a CNF of k clauses:

$$(U_1 \vee U_2 \vee \neg U_3) \quad \bigwedge \quad (\neg U_1 \vee \neg U_4) \quad \bigwedge \quad (U_1 \vee U_2 \vee \neg U_5 \vee U_4)$$



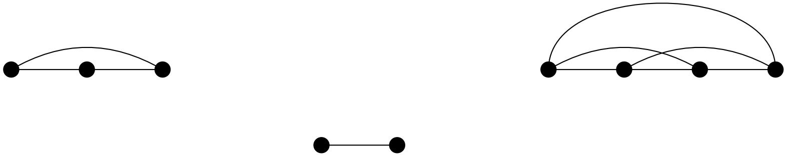
Method

1. one vertex for each variable occurrence within a clause

SAT $\leq$ Stable

We consider a CNF of k clauses:

$$(U_1 \vee U_2 \vee \neg U_3) \quad \bigwedge \quad (\neg U_1 \vee \neg U_4) \quad \bigwedge \quad (U_1 \vee U_2 \vee \neg U_5 \vee U_4)$$



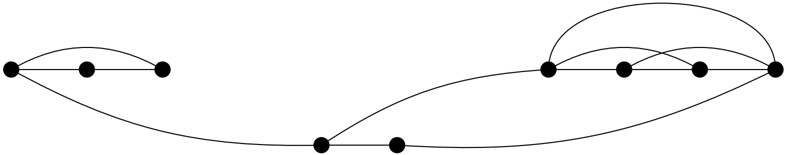
Method

1. one vertex for each variable occurrence within a clause
2. link the vertices of the same clause

SAT $\leq$ Stable

We consider a CNF of k clauses:

$$(U_1 \vee U_2 \vee \neg U_3) \quad \bigwedge \quad (\neg U_1 \vee \neg U_4) \quad \bigwedge \quad (U_1 \vee U_2 \vee \neg U_5 \vee U_4)$$



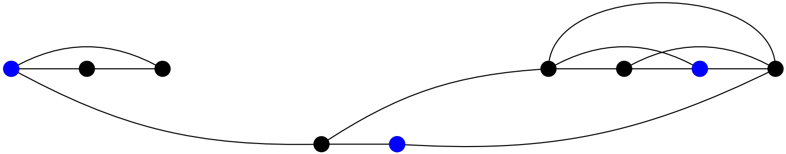
Method

1. one vertex for each variable occurrence within a clause
2. link the vertices of the same clause
3. link positive and negative occurrences

SAT $\leq$ Stable

We consider a CNF of k clauses:

$$(U_1 \vee U_2 \vee \neg U_3) \quad \bigwedge \quad (\neg U_1 \vee \neg U_4) \quad \bigwedge \quad (U_1 \vee U_2 \vee \neg U_5 \vee U_4)$$



- satisfiable $\Rightarrow$ stable of size greater than or equal to k
  - ➜ each clause is satisfied.
  - ➜ build a stable of size k by selecting the true literal in each clause.
- stable of size k $\Rightarrow$ satisfiable
  - ➜ each vertex of the stable corresponds to a literal satisfying a different clause.
  - ➜ by definition, the stable does not contains a pair of vertices corresponding to a variable and its negation.

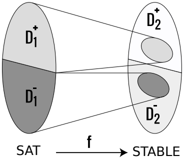
SAT $\leq$ Stable

Conclusion

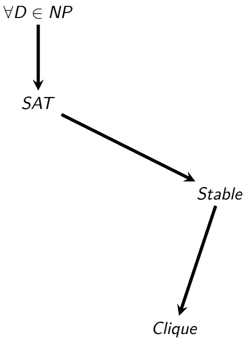
We defined a reduction f which for each instance I_{SAT} of SAT:

- creates an instance of Stable $I_{Stable} = f(I_{SAT})$
- I_{SAT} is positive $\Rightarrow$ I_{Stable} is positive
- I_{SAT} is negative $\Rightarrow$ I_{Stable} is negative
  - by I_{Stable} is positive $\Rightarrow$ I_{SAT} is positive
- f is polynomial time

➜ SAT $\leq$ Stable



Network of reductions of NP-complete problems



skip next reduction

Directed Hamiltonian problem

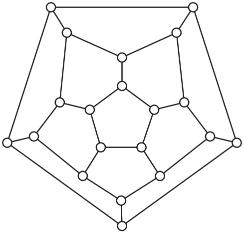
D-HAM

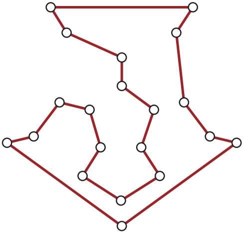
Instance :

- $G = (V, A)$ a directed graph;

Question : is there a Hamiltonian cycle, i.e., cycles passing through each vertex exactly one time?

In this example, presented by Lord Hamilton, the graph is non directed.





D-HAM is NP-Complete

First : D-HAM $\in NP$

Given an instance of D-HAM (a graph $G = (V, A)$) and a cycle C, it is possible to verify in polynomial time whether C is a Hamiltonian cycle. D-HAM is in NP.

D-HAM is NP-Complete

First : D-HAM $\in NP$

Given an instance of D-HAM (a graph $G = (V, A)$) and a cycle C, it is possible to verify in polynomial time whether C is a Hamiltonian cycle. D-HAM is in NP.

Second: polynomial reduction, SAT $\leq$ D-HAM

Let's reduce SAT to D-HAM

SAT $\leq$ D-HAM

Given I an instance of SAT with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_k$.

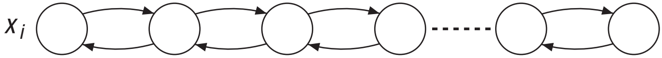
$$(x_1 \vee x_2) \wedge (\neg x_3 \vee x_4 \vee \neg x_5) \wedge \ldots \wedge (\neg x_1)$$

Sketch of the reduction

1. Build graph structures to represent the variables and the clauses.
2. Organize the structures together to encode the formula.
3. Prove that the final structure has a Hamiltonian cycle iff the formula is satisfiable.

SAT ≤ D-HAM

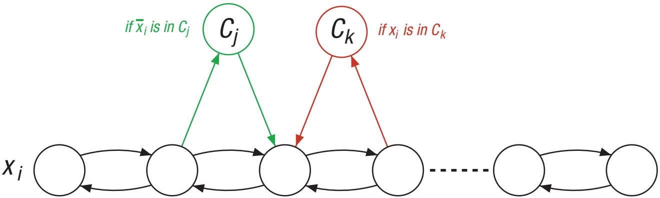
For each variable x_i, we build the following structure



The cycle have to go through these structures. As a convention, we set the corresponding variable to false if the cycle passes through the structure from left to right, else we set it to true.

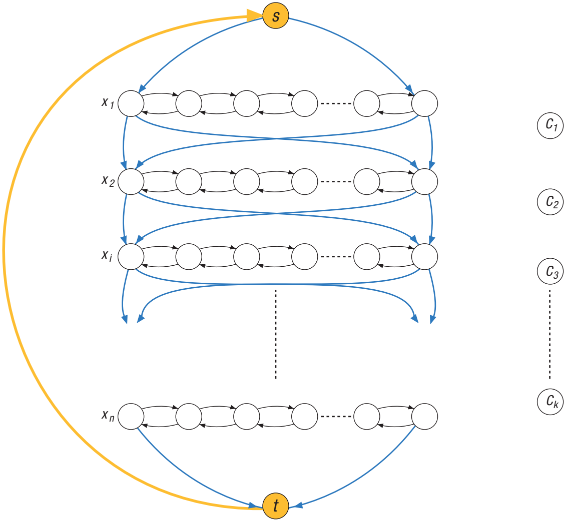
SAT ≤ D-HAM

We add a vertex for each clause and we link it to the existing variable structures.

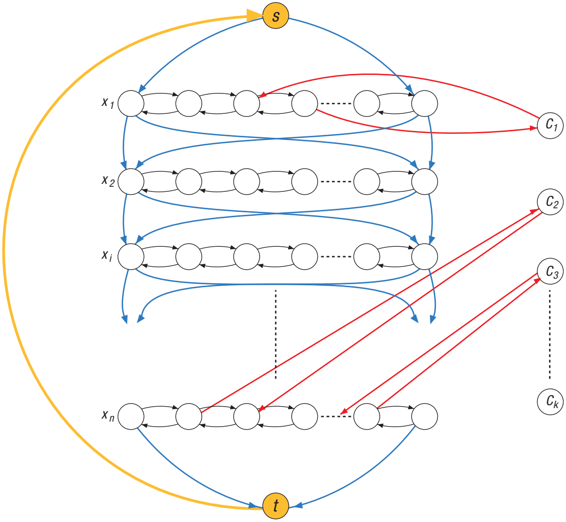


Remark : For each variable, the structures must be long enough to place the clauses (at least 3 + k nodes). The total number of nodes (3n + kn) remains polynomial depending on the size of the SAT formula.

SAT $\leq$ D-HAM



SAT ≤ D-HAM



SAT $\leq$ D-HAM

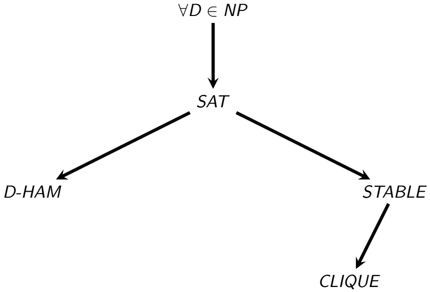
Suppose it exists a Hamiltonian cycle

- The cycle encode the assignments of the variables (depending on the traversal direction).
- The Hamiltonian cycle has to visit each clause structure.
- When visited, a clause is satisfied by setting one of its literals to true.
- So if there is a Hamiltonian cycle, it exists an assignment satisfying the formula.

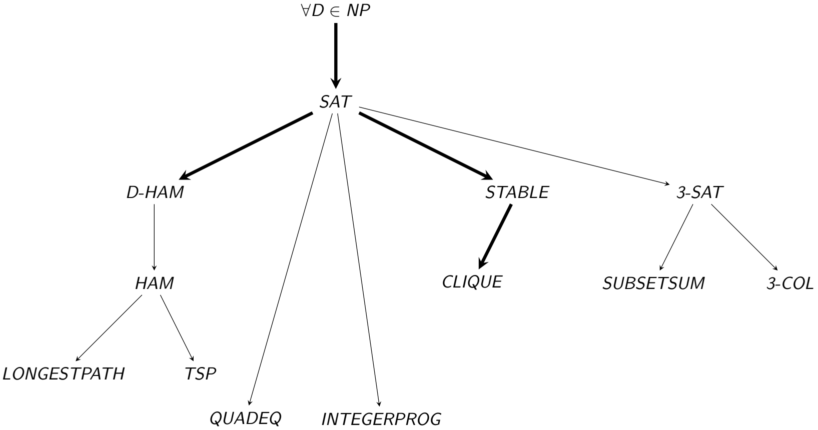
Suppose the formula is satisfiable

The assignment describes a traversal (be careful not to re-visit clauses already satisfied)

Network of reductions of *NP*-complete problems



Network of reductions of NP-complete problems



Plan

do we have classified all problems ?

NP **contains only the decision problems** $D = D^+ \cup D^-$ **for which it exists a certificate for positive instances (**$I \in D^+$**...).**

Other complexity classes: *coNP*

coNP

- *NP* is the group of decision problems for which verifying a solution is polynomial.
- *coNP* is the group of decision problems for which verifying a counter-example is polynomial.
- We suppose $coNP \neq NP$
- In a similar way, it exists *coNP*-complete problems.
- $P \subseteq NP \cap coNP$

Examples of *coNP* problems

Tautology

- Consider a boolean formula, is it true for any assignment?
- It is easier to exhibit a counter example than to prove that it is true for all assignments.

Examples of *coNP* problems

Tautology

- Consider a boolean formula, is it true for any assignment?
- It is easier to exhibit a counter example than to prove that it is true for all assignments.

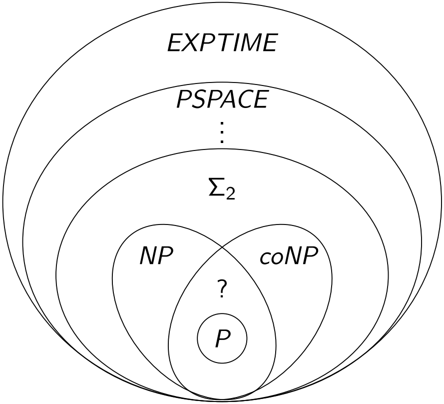
Primality

- Testing the primality of a number is in *coNP*, we can exhibit a factor as a certificate.
- In contrary, finding a certificate proving that it is prime and that no factors exists, seems to be more difficult. However, since 2002, Primality is in *P*!

And now? do we have classified all problems ?

What if verifying the certificate is not in P?

Hierarchy of complexity classes



Examples

- Σ_2 : given a boolean formula $\phi(x, y)$, satisfy $\exists x \forall y \phi(x, y)$
- *PSPACE* : Othello (Reversi), QBF
- *EXPTIME* : Chess, Go

What you should remember

- Definitions of the P and NP classes
- Definition of polynomial reduction
- Application of polynomial reduction on simple problems (see tutorial #5 and above)
- Classical problems (SAT, Stable, HAM)
  - ✗ You don't have to remember the reductions...
  - ✓ ...but you should understand them and be able to explain them!