Graph algorithms

Master 2 ICS

Background on algorithms and complexity classes

Florent Foucaud

Organisation

- 3 lectures this week and next week, 2 more lectures in about 6 weeks
- 4 more lectures by Laurent Beaudou
- contact : `florent.foucaud@uca.fr`

Contents :

- Background on algorithms and complexity
- Algorithms for specific graphs
- Parameterized complexity

History

- First algorithms :
  - Babylone, -2500 / ancient Egypt, -1500 / India, -800 : first algorithms (ex : division)
  - Ancient Greece, -250 : prime numbers (Euclid, Ératosthenes)
  - India, 450 : solving equations (Kuttaka)
  - arab-persian world, 850 : cryptography, arithmetics
    (Muhammad ibn Musa al Khwarizmi, most read mathematician in the middle ages)
  - 1230 : → Alchoarismi → Algorismo : notion of an algorithm

Muhammad ibn Musa al Khwarizmi
(780-850)

al-Kitab al-mukhtasar fi hisab
al-jabr wal-muqabala (820)

History

- First algorithms :
  - Babylone, -2500 / ancient Egypt, -1500 / India, -800 : first algorithms (ex : division)
  - Ancient Greece, -250 : prime numbers (Euclid, Ératosthenes)
  - India, 450 : solving equations (Kuttaka)
  - arab-persian world, 850 : cryptography, arithmetics
    (Muhammad ibn Musa al Khwarizmi, most read mathematician in the middle ages)
  - 1230 : → Alchoarismi → Algorismo : notion of an algorithm

- David Hilbert, 1928 : is there an algorithm to solve any mathematical question ? (*Entscheidungsproblem*, translated : “decision problem”)

David Hilbert (1862-1943)

History

- First algorithms :
  - Babylone, -2500 / ancient Egypt, -1500 / India, -800 : first algorithms (ex : division)
  - Ancient Greece, -250 : prime numbers (Euclid, Ératosthenes)
  - India, 450 : solving equations (Kuttaka)
  - arab-persian world, 850 : cryptography, arithmetics
    (Muhammad ibn Musa al Khwarizmi, most read mathematician in the middle ages)
  - 1230 : → Alchoarismi → Algorismo : notion of an algorithm

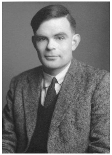
- David Hilbert, 1928 : is there an algorithm to solve any mathematical question ? (*Entscheidungsproblem*, translated : "decision problem")
- Kurt Gödel, 1931 : incompleteness theorem (mathematical logic)

Kurt Gödel (1906-1978)

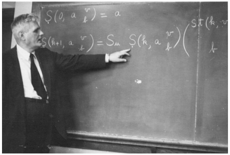
History

- First algorithms :
  - Babylone, -2500 / ancient Egypt, -1500 / India, -800 : first algorithms (ex : division)
  - Ancient Greece, -250 : prime numbers (Euclid, Ératosthenes)
  - India, 450 : solving equations (Kuttaka)
  - arab-persian world, 850 : cryptography, arithmetics
    (Muhammad ibn Musa al Khwarizmi, most read mathematician in the middle ages)
  - 1230 : → Alchoarismi → Algorismo : notion of an algorithm

- David Hilbert, 1928 : is there an algorithm to solve any mathematical question ? (*Entscheidungsproblem*, translated : "decision problem")
- Kurt Gödel, 1931 : incompleteness theorem (mathematical logic)
- Alonzo Church and Alan Turing, 1936 : solution of the Entscheidungsproblem : NO !
  (→ Halting problem, Turing machines)



Alan Turing (1912-1954)



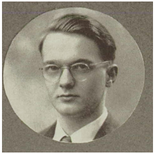
Alonzo Church (1903-1995)

History

- First algorithms :
  - Babylone, -2500 / ancient Egypt, -1500 / India, -800 : first algorithms (ex : division)
  - Ancient Greece, -250 : prime numbers (Euclid, Ératosthenes)
  - India, 450 : solving equations (Kuttaka)
  - arab-persian world, 850 : cryptography, arithmetics
    (Muhammad ibn Musa al Khwarizmi, most read mathematician in the middle ages)
  - 1230 : → Alchoarismi → Algorismo : notion of an algorithm

- David Hilbert, 1928 : is there an algorithm to solve any mathematical question ? (*Entscheidungsproblem*, translated : "decision problem")
- Kurt Gödel, 1931 : incompleteness theorem (mathematical logic)
- Alonzo Church and Alan Turing, 1936 : solution of the Entscheidungsproblem : NO !
  (→ Halting problem, Turing machines)
- Alan Cobham and Jack Edmonds, 1965 :
  an algorithm is efficient if it is polynomial-time

Jack Edmonds (1934-)



Alan B. Cobham (1927-2011)

History

- First algorithms :
  - Babylone, -2500 / ancient Egypt, -1500 / India, -800 : first algorithms (ex : division)
  - Ancient Greece, -250 : prime numbers (Euclid, Ératosthenes)
  - India, 450 : solving equations (Kuttaka)
  - arab-persian world, 850 : cryptography, arithmetics
    (Muhammad ibn Musa al Khwarizmi, most read mathematician in the middle ages)
  - 1230 : → Alchoarismi → Algorismo : notion of an algorithm

- David Hilbert, 1928 : is there an algorithm to solve any mathematical question ? (*Entscheidungsproblem*, translated : “decision problem”)
- Kurt Gödel, 1931 : incompleteness theorem (mathematical logic)
- Alonzo Church and Alan Turing, 1936 : solution of the Entscheidungsproblem : NO !
  (→ Halting problem, Turing machines)
- Alan Cobham and Jack Edmonds, 1965 :
  an algorithm is efficient if it is polynomial-time
- 1970s : complexity theory

Complexity of an algorithmic problem

Algorithmic problem : input, output

Exemples :

- Multiply two numbers n_1 et n_2 encoded in binary
- Sort a table of n integers
- Find a shortest path from A to B in a graph on n vertices
- Cover a network with n vertices with k radio antennas

Complexity of an algorithm : quantity of ressources needed by the algorithms, as a function of the size n of the input

Complexity of an algorithmic problem

Algorithmic problem : input, output

Exemples :

- Multiply two numbers n_1 et n_2 encoded in binary
- Sort a table of n integers
- Find a shortest path from A to B in a graph on n vertices
- Cover a network with n vertices with k radio antennas

Complexity of an algorithm : quantity of ressources needed by the algorithms, as a function of the size n of the input

Complexity of algorithmic problem P : lowest complexity of any algorithm solving P

Complexity of an algorithmic problem

Algorithmic problem : input, output

Exemples :

- Multiply two numbers n_1 et n_2 encoded in binary
- Sort a table of n integers
- Find a shortest path from A to B in a graph on n vertices
- Cover a network with n vertices with k radio antennas

Complexity of an algorithm : quantity of ressources needed by the algorithms, as a function of the size n of the input

Complexity of algorithmic problem P : lowest complexity of any algorithm solving P

What ressources ?

- **Time complexity** $T(n)$: lowest number of steps
- **Space complexity** $S(n)$: lowest memory size
- ...

Complexity of an algorithmic problem

Algorithmic problem : input, output

Exemples :

- Multiply two numbers n_1 et n_2 encoded in binary
- Sort a table of n integers
- Find a shortest path from A to B in a graph on n vertices
- Cover a network with n vertices with k radio antennas

Complexity of an algorithm : quantity of ressources needed by the algorithms, as a function of the size n of the input

Complexity of algorithmic problem P : lowest complexity of any algorithm solving P

What ressources ?

- **Time complexity** $T(n)$: lowest number of steps
- **Space complexity** $S(n)$: lowest memory size
- ...

Remark : $S(n) \leq T(n)$

Complexity of an algorithmic problem

Algorithmic problem : input, output

Exemples :

- Multiply two numbers n_1 et n_2 encoded in binary
- Sort a table of n integers
- Find a shortest path from A to B in a graph on n vertices
- Cover a network with n vertices with k radio antennas

Complexity of an algorithm : quantity of ressources needed by the algorithms, as a function of the size n of the input

Complexity of algorithmic problem P : lowest complexity of any algorithm solving P

What ressources ?

- **Time complexity** $T(n)$: lowest number of steps
- **Space complexity** $S(n)$: lowest memory size
- ...

Remark : $S(n) \leq T(n)$

How to measure them ?

- **Worst-case complexity**
- Average complexity

  → according to some probability distribution of the input
- ...

Input size

Beware of the encoding !

Input size

Beware of the encoding !

- Integer $n \to \lceil \log_2(n) \rceil$ bits

Input size

Beware of the encoding !

- Integer $n \longrightarrow \lceil \log_2(n) \rceil$ bits
- Integer in *C* language $\longrightarrow$ 8 bytes (constant)

Input size

Beware of the encoding !

- Integer $n \rightarrow \lceil \log_2(n) \rceil$ bits
- Integer in C language $\rightarrow$ 8 bytes (constant)
- Graph with n vertices, m edges $\rightarrow$ $(n+m)\times$ (size of integer) $\rightarrow$ **adjacency list**

Combinatorial explosion

Time complexity : $T(n)$

Best of the best problems : constant complexity $T(n) \rightarrow 1, 10$ or logarithmic complexity $T(n) \rightarrow \log_2(n), 3\log(n)$...

Very good problems : linear complexity $T(n) \rightarrow 10n, 2n, 1000n, n$ ...

Reasonable problems : polynomial complexity $T(n) \rightarrow 4n^2, 10n^3, n^{1000}$ ...
(in practice n^3 or more, not so good)

Difficult problems : exponential complexity $T(n) \rightarrow 2^n, n!, n^n, 2^{2^n}$ ...
→ Intuition : *check all possible solutions*

Combinatorial explosion

Time complexity : $T(n)$

Best of the best problems : constant complexity $T(n) \to 1, 10$ or logarithmic complexity $T(n) \to \log_2(n), 3\log(n)$...

Very good problems : linear complexity $T(n) \to 10n, 2n, 1000n, n$ ...

Reasonable problems : polynomial complexity $T(n) \to 4n^2, 10n^3, n^{1000}$ ...
(in practice n^3 or more, not so good)

Difficult problems : exponential complexity $T(n) \to 2^n, n!, n^n, 2^{2^n}$ ...

$\to$ Intuition : *check all possible solutions*

$T(n)$	$n=10$	$n=50$	$n=100$	$n=200$	$n=300$
n	10	50	100	200	300
$100n$	1000	5000	10000	20000	30000
n^2	100	2500	10000	40000	90000
2^n	1024	(16 digits)	(31 digits)	(60 digits)	(91 digits)
$n!$	3628800	(64 digits)	(157 digits)	(374 digits)	(614 digits)

Combinatorial explosion

Time complexity : $T(n)$

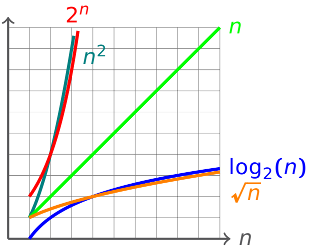
Best of the best problems : constant complexity $T(n) \rightarrow 1, 10$ or logarithmic complexity $T(n) \rightarrow \log_2(n), 3\log(n)$...

Very good problems : linear complexity $T(n) \rightarrow 10n, 2n, 1000n, n$ ...

Reasonable problems : polynomial complexity $T(n) \rightarrow 4n^2, 10n^3, n^{1000}$ ...
(in practice n^3 or more, not so good)

Difficult problems : exponential complexity $T(n) \rightarrow 2^n, n!, n^n, 2^{2^n}$ ...

→ Intuition : *check all possible solutions*



n between 0 and 10

Combinatorial explosion

Time complexity : $T(n)$

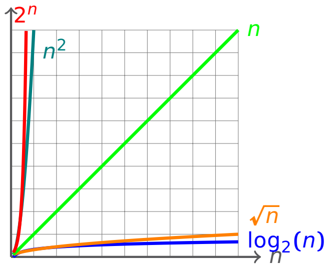
Best of the best problems : constant complexity $T(n) \rightarrow 1, 10$ or logarithmic complexity $T(n) \rightarrow \log_2(n), 3\log(n)$...

Very good problems : linear complexity $T(n) \rightarrow 10n, 2n, 1000n, n$ ...

Reasonable problems : polynomial complexity $T(n) \rightarrow 4n^2, 10n^3, n^{1000}$ ...
(in practice n^3 or more, not so good)

Difficult problems : exponential complexity $T(n) \rightarrow 2^n, n!, n^n, 2^{2^n}$ ...

$\rightarrow$ Intuition : *check all possible solutions*



n between 0 and 100

Combinatorial explosion

Time complexity : $T(n)$

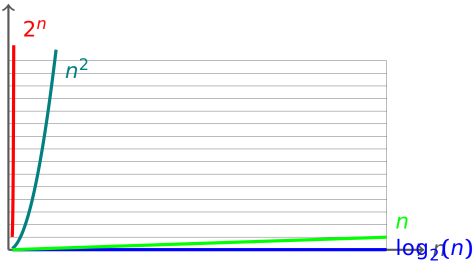
Best of the best problems : constant complexity $T(n) \rightarrow 1, 10$ or logarithmic complexity $T(n) \rightarrow \log_2(n), 3\log(n)$...

Very good problems : linear complexity $T(n) \rightarrow 10n, 2n, 1000n, n$ ...

Reasonable problems : polynomial complexity $T(n) \rightarrow 4n^2, 10n^3, n^{1000}$ ...
(in practice n^3 or more, not so good)

Difficult problems : exponential complexity $T(n) \rightarrow 2^n, n!, n^n, 2^{2^n}$ ...

→ Intuition : *check all possible solutions*



n between 0 and 1000

Asymptotic notations

Most of the time, only need to distinguish types of complexity
→ logarithmic, linear, quadratic, exponential...

Asymptotic notations

Most of the time, only need to distinguish types of complexity
→ logarithmic, linear, quadratic, exponential...

Exact complexity depends on the machine, the programming language, the compiler...

Asymptotic notations

Most of the time, only need to distinguish types of complexity
$\longrightarrow$ logarithmic, linear, quadratic, exponential...

Exact complexity depends on the machine, the programming language, the compiler...

Use asymptotic notations that omit constant factors and “pathological base cases”.

Big Oh

Invented from 1894 to 1960 by Bachmann, Hardy, **Knuth**, Landau, Littlewood...

Definition (Big Oh)

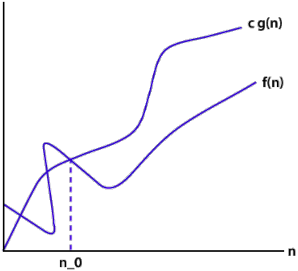
Two functions $f, g : \mathbb{N} \to \mathbb{N}$. $f(n) \in O(g(n))$ if there exists a constant $c > 0 \in \mathbb{R}$ and a rank $n_0 \in \mathbb{N}$ s.t. for any integer $i \geq n_0$, we have $f(i) \leq c \cdot g(i)$.

Big Oh

Invented from 1894 to 1960 by Bachmann, Hardy, **Knuth**, Landau, Littlewood...

Definition (Big Oh)

Two functions $f, g : \mathbb{N} \to \mathbb{N}$. $f(n) \in O(g(n))$ if there exists a constant $c > 0 \in \mathbb{R}$ and a rank $n_0 \in \mathbb{N}$ s.t. for any integer $i \geq n_0$, we have $f(i) \leq c \cdot g(i)$.



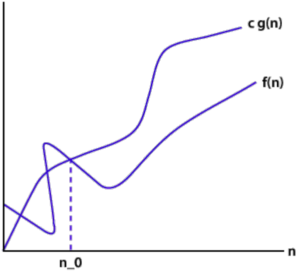
f(n)=O(g(n))

Big Oh

Invented from 1894 to 1960 by Bachmann, Hardy, **Knuth**, Landau, Littlewood...

Definition (Big Oh)

Two functions $f, g : \mathbb{N} \rightarrow \mathbb{N}$. $f(n) \in O(g(n))$ if there exists a constant $c > 0 \in \mathbb{R}$ and a rank $n_0 \in \mathbb{N}$ s.t. for any integer $i \geq n_0$, we have $f(i) \leq c \cdot g(i)$.



f(n)=O(g(n))

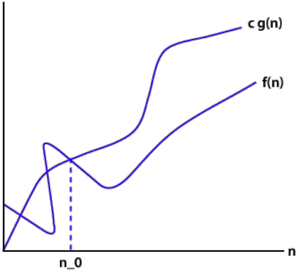
Intuitively : *f does not grow faster than g (up to constant factors) when n is large*

Big Oh

Invented from 1894 to 1960 by Bachmann, Hardy, **Knuth**, Landau, Littlewood...

Definition (Big Oh)

Two functions $f, g : \mathbb{N} \to \mathbb{N}$. $f(n) \in O(g(n))$ if there exists a constant $c > 0 \in \mathbb{R}$ and a rank $n_0 \in \mathbb{N}$ s.t. for any integer $i \geq n_0$, we have $f(i) \leq c \cdot g(i)$.



f(n)=O(g(n))

Examples :

- $2^{1000} \in O(1)$
- $100n \in O(n/1000)$
- $1000n \in O(n^2/10)$
- $n + n^2 + \log(n) \in O(n^3)$

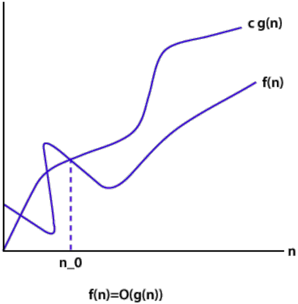
Intuitively : *f does not grow faster than g (up to constant factors) when n is large*

Big Oh

Invented from 1894 to 1960 by Bachmann, Hardy, **Knuth**, Landau, Littlewood...

Definition (Big Oh)

Two functions $f, g : \mathbb{N} \to \mathbb{N}$. $f(n) \in O(g(n))$ if there exists a constant $c > 0 \in \mathbb{R}$ and a rank $n_0 \in \mathbb{N}$ s.t. for any integer $i \geq n_0$, we have $f(i) \leq c \cdot g(i)$.



Examples :

- $2^{1000} \in O(1)$
- $100n \in O(n/1000)$
- $1000n \in O(n^2/10)$
- $n + n^2 + \log(n) \in O(n^3)$

Intuitively : *f does not grow faster than g (up to constant factors) when n is large*

Abuse of notation : $10n = O(n^2)$

Big Omega

"The reverse of Big Oh"

Definition (Big Omega)

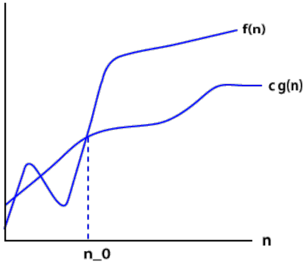
Two functions $f, g : \mathbb{N} \to \mathbb{N}$. $f(n) \in \Omega(g(n))$ if there exists a constant $c > 0 \in \mathbb{R}$ and a rank $n_0 \in \mathbb{N}$ s.t. for any integer $i \geq n_0$, we have $f(i) \geq c \cdot g(i)$.

Big Omega

"The reverse of Big Oh"

Definition (Big Omega)

Two functions $f, g : \mathbb{N} \rightarrow \mathbb{N}$. $f(n) \in \Omega(g(n))$ if there exists a constant $c > 0 \in \mathbb{R}$ and a rank $n_0 \in \mathbb{N}$ s.t. for any integer $i \geq n_0$, we have $f(i) \geq c \cdot g(i)$.



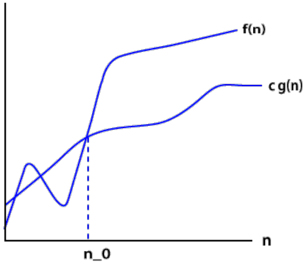
f(n)= Omega(g(n))

Big Omega

"The reverse of Big Oh"

Definition (Big Omega)

Two functions $f, g : \mathbb{N} \rightarrow \mathbb{N}$. $f(n) \in \Omega(g(n))$ if there exists a constant $c > 0 \in \mathbb{R}$ and a rank $n_0 \in \mathbb{N}$ s.t. for any integer $i \geq n_0$, we have $f(i) \geq c \cdot g(i)$.



f(n)= Omega(g(n))

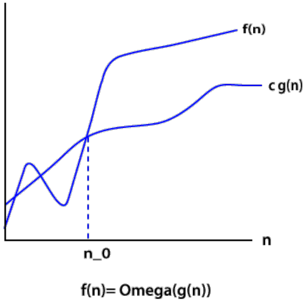
Intuitively : *f does not grow slower than g (up to constant factors) when n is large*

Big Omega

"The reverse of Big Oh"

Definition (Big Omega)

Two functions $f, g : \mathbb{N} \to \mathbb{N}$. $f(n) \in \Omega(g(n))$ if there exists a constant $c > 0 \in \mathbb{R}$ and a rank $n_0 \in \mathbb{N}$ s.t. for any integer $i \geq n_0$, we have $f(i) \geq c \cdot g(i)$.



Examples :

- $2^{1000} \in \Omega(1)$
- $n^2/1000 \in \Omega(100000n^2)$
- $n \log(n)/10 \in \Omega(100n)$
- $n^3 - n^2 \in \Omega(n^2)$

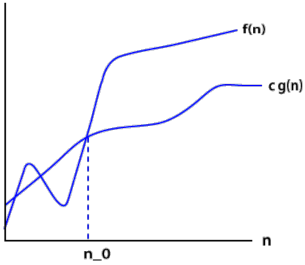
Intuitively : *f does not grow slower than g (up to constant factors) when n is large*

Big Omega

"The reverse of Big Oh"

Definition (Big Omega)

Two functions $f, g : \mathbb{N} \to \mathbb{N}$. $f(n) \in \Omega(g(n))$ if there exists a constant $c > 0 \in \mathbb{R}$ and a rank $n_0 \in \mathbb{N}$ s.t. for any integer $i \geq n_0$, we have $f(i) \geq c \cdot g(i)$.



f(n)= Omega(g(n))

Examples :

- $2^{1000} \in \Omega(1)$
- $n^2/1000 \in \Omega(100000n^2)$
- $n \log(n)/10 \in \Omega(100n)$
- $n^3 - n^2 \in \Omega(n^2)$

Intuitively : *f does not grow slower than g (up to constant factors) when n is large*

Remark : if $f(n) \in O(g(n))$, then $g(n) \in \Omega(f(n))$ and conversely

Big Theta

"Combination of Big Oh and Big Omega"

Definition (Big Theta)

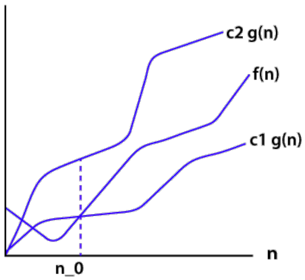
Two functions $f, g : \mathbb{N} \to \mathbb{N}$. $f(n) \in \Theta(g(n))$ if $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$.

Big Theta

"Combination of Big Oh and Big Omega"

Definition (Big Theta)

Two functions $f, g : \mathbb{N} \to \mathbb{N}$. $f(n) \in \Theta(g(n))$ if $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$.



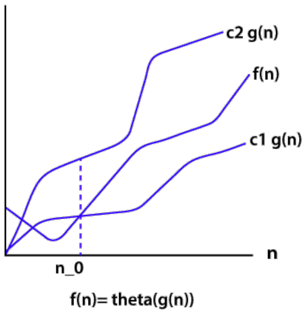
f(n)= theta(g(n))

Big Theta

"Combination of Big Oh and Big Omega"

Definition (Big Theta)

Two functions $f, g : \mathbb{N} \to \mathbb{N}$. $f(n) \in \Theta(g(n))$ if $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$.



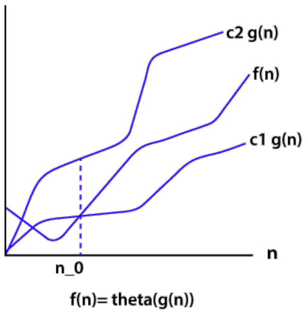
Intuitively : *f and g grow equally fast (up to constant factors) when n is large*

Big Theta

"Combination of Big Oh and Big Omega"

Definition (Big Theta)

Two functions $f, g : \mathbb{N} \to \mathbb{N}$. $f(n) \in \Theta(g(n))$ if $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$.



Examples :

- $2^{1000} \in \Theta(1)$
- $1000n \in \Theta(n/1000)$
- $10n^2 + 36n \in \Theta(n^2)$
- $n^3 - n^2 \in \Theta(n^3)$

Intuitively : *f and g grow equally fast (up to constant factors) when n is large*

Complexity classes

- logarithmic : $\Theta(\log(n))$
- linear : $\Theta(n)$
- quadratic : $\Theta(n^2)$
- cubic : $\Theta(n^3)$
- polynomial : $\Theta(n^c)$ pour $c > 1$
- single-exponential : $\Theta(c^n)$ for $c > 1$
- double-exponential : $\Theta(c_1^{c_2^n})$ for $c_1, c_2 > 1$
- ...

Typical complexities : concrete examples

- Binary search in an ordered set of size n : $\log_2(n)$

(logarithmic)

Typical complexities : concrete examples

- Binary search in an ordered set of size n : $\log_2(n)$

  (logarithmic)

- Go through an unordered set of size n : n

  (linear)

Typical complexities : concrete examples

- Binary search in an ordered set of size n : $\log_2(n)$
  (logarithmic)
- Go through an unordered set of size n : n
  (linear)
- Sort an integer table, explore a graph, find a shortest path, etc. : n^c
  (polynomial)

Typical complexities : concrete examples

- Binary search in an ordered set of size n : $\log_2(n)$
  (logarithmic)
- Go through an unordered set of size n : n
  (linear)
- Sort an integer table, explore a graph, find a shortest path, etc. : n^c
  (polynomial)
- k nested loops each of length n : n^k

Typical complexities : concrete examples

- Binary search in an ordered set of size n : $\log_2(n)$
  (logarithmic)
- Go through an unordered set of size n : n
  (linear)
- Sort an integer table, explore a graph, find a shortest path, etc. : n^c
  (polynomial)
- k nested loops each of length n : n^k
- Go through all subsets of a set of size n : 2^n
  (single-exponential)

Typical complexities : concrete examples

- Binary search in an ordered set of size n : $\log_2(n)$
  (logarithmic)
- Go through an unordered set of size n : n
  (linear)
- Sort an integer table, explore a graph, find a shortest path, etc. : n^c
  (polynomial)
- k nested loops each of length n : n^k
- Go through all subsets of a set of size n : 2^n
  (single-exponential)
- Go through all partitions of a set of size n : n^n
  ($= 2^{n \log_2(n)}$, super-exponential)

Typical complexities : concrete examples

- Binary search in an ordered set of size n : $\log_2(n)$
  (logarithmic)
- Go through an unordered set of size n : n
  (linear)
- Sort an integer table, explore a graph, find a shortest path, etc. : n^c
  (polynomial)
- k nested loops each of length n : n^k
- Go through all subsets of a set of size n : 2^n
  (single-exponential)
- Go through all partitions of a set of size n : n^n
  ($= 2^{n \log_2(n)}$, super-exponential)
- Go through all permutations of a set of size n : $n!$
  $\approx n^n$ by *Stirling's formula* $n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$

Typical complexities : concrete examples

- Binary search in an ordered set of size n : $\log_2(n)$
  (logarithmic)
- Go through an unordered set of size n : n
  (linear)
- Sort an integer table, explore a graph, find a shortest path, etc. : n^c
  (polynomial)
- k nested loops each of length n : n^k
- Go through all subsets of a set of size n : 2^n
  (single-exponential)
- Go through all partitions of a set of size n : n^n
  ($= 2^{n \log_2(n)}$, super-exponential)
- Go through all permutations of a set of size n : $n!$
  $\approx n^n$ by *Stirling's formula* $n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$
- Go through all subsets of subsets : 2^{2^n}
  (double-exponential)

Typical complexities : concrete examples

- Binary search in an ordered set of size n : $\log_2(n)$
  (logarithmic)
- Go through an unordered set of size n : n
  (linear)
- Sort an integer table, explore a graph, find a shortest path, etc. : n^c
  (polynomial)
- k nested loops each of length n : n^k
- Go through all subsets of a set of size n : 2^n
  (single-exponential)
- Go through all partitions of a set of size n : n^n
  ($= 2^{n \log_2(n)}$, super-exponential)
- Go through all permutations of a set of size n : $n!$
  $\approx n^n$ by *Stirling's formula* $n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$
- Go through all subsets of subsets : 2^{2^n}
  (double-exponential)
- ...

Decision problems

Goal : classify algorithmic problems according to their complexities.

Decision problems

Goal : classify algorithmic problems according to their complexities.

Simplest problems :
→ Decision problem : input, question with YES/NO answer

- Is this list sorted ?
- Is this graph 3-colorable ?
- Does this program always stop ?
- ...

Decision problems

Goal : classify algorithmic problems according to their complexities.

Simplest problems :
→ Decision problem : input, question with YES/NO answer

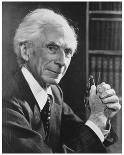
- Is this list sorted ?
- Is this graph 3-colorable ?
- Does this program always stop ?
- ...

3-Coloring
Input: A graph G
Question: Is G 3-colorable ?

Barber paradox

In a village, a barber shaves exactly all men that do not shave themselves..

Question : Who shaves the barber ?



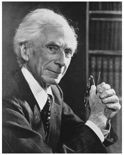
Bertrand Russell (1872-1970)

Barber paradox

In a village, a barber shaves exactly all men that do not shave themselves..

Question : Who shaves the barber ?

PARADOX !



Bertrand Russell (1872-1970)

To stop or not to stop ? That is the question

Halting problem

Goven a computer code and an input parameter for it, decide in finite time :

1. if it will stop one day *or* 2. if it will loop forever ?

To stop or not to stop ? That is the question

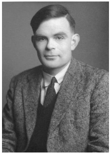
Halting problem

Goven a computer code and an input parameter for it, decide in finite time :

1. if it will stop one day *or* 2. if it will loop forever ?

Theorem (Alan Turing, 1936)

There is no algorithm that solves the Halting problem.



Alan Turing (1912-1954)

To stop or not to stop ? That is the question

Halting problem

Goven a computer code and an input parameter for it, decide in finite time :
1. if it will stop one day *or* 2. if it will loop forever ?

Theorem (Alan Turing, 1936)

There is no algorithm that solves the Halting problem.

Proof : Suppose by contradiction there is such a finite-time algorithm :

```
halt(code, parameter)
```

that returns
- YES if the given code and parameter stop one day, and
- NO if it loops forever.

To stop or not to stop ? That is the question

Halting problem

Goven a computer code and an input parameter for it, decide in finite time :

1. if it will stop one day *or* 2. if it will loop forever ?

Theorem (Alan Turing, 1936)

There is no algorithm that solves the Halting problem.

Proof : Suppose by contradiction there is such a finite-time algorithm :

`halt(code, parameter)`

that returns
- YES if the given code and parameter stop one day, and
- NO if it loops forever.

Define the following algorithm :

```
def diag(x):
  • if halt(x,x) returns YES then:
      ▶ loop forever
  • else:
      ▶ return YES
```

To stop or not to stop ? That is the question

Halting problem

Goven a computer code and an input parameter for it, decide in finite time :

1. if it will stop one day *or* 2. if it will loop forever ?

Theorem (Alan Turing, 1936)

There is no algorithm that solves the Halting problem.

Proof : Suppose by contradiction there is such a finite-time algorithm :

```
halt(code, parameter)
```

that returns
- YES if the given code and parameter stop one day, and
- NO if it loops forever.

Define the following algorithm :

```
def diag(x):
  • if halt(x,x) returns YES then:
      ▶ loop forever
  • else:
      ▶ return YES
```

What is returned by diag(diag) ?

To stop or not to stop ? That is the question

Halting problem

Goven a computer code and an input parameter for it, decide in finite time :
1. if it will stop one day *or* 2. if it will loop forever ?

Theorem (Alan Turing, 1936)

There is no algorithm that solves the Halting problem.

Proof : Suppose by contradiction there is such a finite-time algorithm :

`halt(code, parameter)`

that returns
- YES if the given code and parameter stop one day, and
- NO if it loops forever.

Define the following algorithm :

```
def diag(x):
  • if halt(x,x) returns YES then:
      ▸ loop forever
  • else:
      ▸ return YES
```

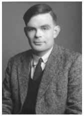
What is returned by diag(diag) ?

PARADOX !

Undecidable problems

Undecidable problems :

- Halting problem (Alan Turing, 1936)
- Word correspondence : two sets of words $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$
  → Can we re-arrange them to create two identical words ? (Emil Post, 1946)
- Integer solutions of Diophantine equations
  of the form $2x^2 + 3y^3 - 2z = 0$ (Hilbert's 10th problem, 1900 - Youri Matyasevitch, 1970)
- Determine the winner of the card game "Magic : The gathering"
  (Churchill-Biderman-Herrick, 2019)



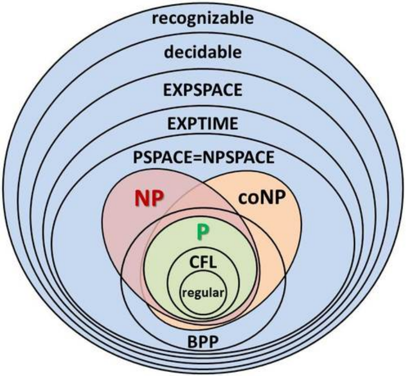
Alan Turing (1912-1954)

Emil L. Post (1897-1954)

Youri Matyasevitch (1947-)

David Hilbert (1862-1943)

Some complexity classes

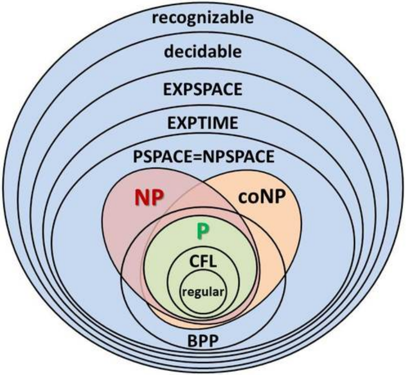


Class P ("polynomial") : "reasonable problems" (Cobham-Edmonds, 1965)

Jack Edmonds (1934-)

Alan B. Cobham (1927-2011)

Some complexity classes



Class P ("polynomial") : "reasonable problems" (Cobham-Edmonds, 1965)

Jack Edmonds (1934-)

Alan B. Cobham (1927-2011)

Higher up : probably hard problems

P and NP complexity classes

Definition (Class P)

Decision problems that can be solved in polynomial time.

P and NP complexity classes

Definition (Class P)

Decision problems that can be solved in polynomial time.

Definition (Class NP = "non-deterministic polynomial")

Decision problems for which there exists a certificate (function of input) s.t., for an input I and its certificate $C(I)$, one can check in time polynomial in I, whether I is a YES-input or not.

P and NP complexity classes

Definition (Class P)

Decision problems that can be solved in polynomial time.

Definition (Class NP = "non-deterministic polynomial")

Decision problems for which there exists a certificate (function of input) s.t., for an input I and its certificate $C(I)$, one can check in time polynomial in I, whether I is a YES-input or not.

Examples :

- All problems in P
- Graph coloring
- ...

P versus NP

Do checking a solution and finding a solution inherently have the same complexity ?

P versus NP

Do checking a solution and finding a solution inherently have the same complexity ?

→ Intuitively, no...

P versus NP

Do checking a solution and finding a solution inherently have the same complexity ?

→ Intuitively, no...

Question (P versus NP - a question worth a million US$)

Is it true that $P = NP$?

P versus NP

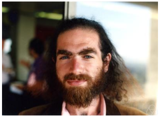
Do checking a solution and finding a solution inherently have the same complexity ?

→ Intuitively, no...

Question (P versus NP - a question worth a million US$)

Is it true that $P = NP$?

7 millenium problems woth 1 million US$



Grigori Perelman (1966-)

Some problems are easier than others

Minimum « easier » than Sorting

- know how to sort a list ⇝ know how to find the minimum
- can build an algorithm for **Minimum** using an algorithm for **Sorting**

(Polynomial) reduction

$\mathcal{P}_1$

- `Input` : E_1
- `Question` : Does E_1 have property P_1 ?

$\mathcal{P}_2$

- `Input` : E_2
- `Question` : Does E_2 have property P_2 ?

Transform E_1 into $f(E_1) = E_2$ (in polynomial time)
such that
E_1 has property $P_1 \iff E_2$ has property P_2

$\mathcal{P}_1$ reduces to $\mathcal{P}_2$ (in polynomial time)

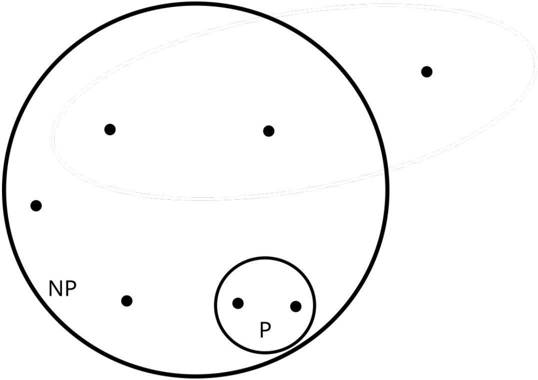
- Algorithm for $\mathcal{P}_2$: A_2 (polynomial)
- Algo for $\mathcal{P}_1$: $E_1 \rightsquigarrow E_2 \overset{A_2}{\rightsquigarrow}$ YES or NO (in polynomial time)

$\mathcal{P}_1$ is « easier » than $\mathcal{P}_2$

NP-complete problems

Definition (NP-hard and NP-complete problems)

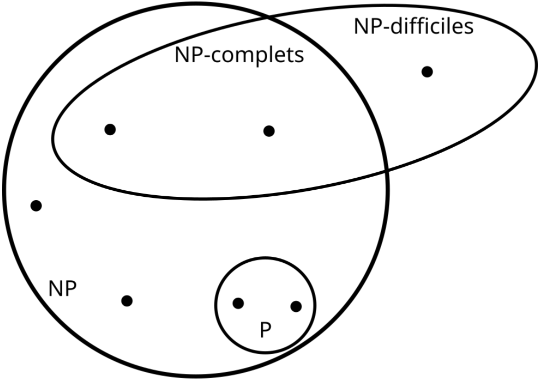
Decision problem P_1 is NP-hard if all problems in NP admit a reduction to P_1.
Decision problem P_1 is NP-complete if it belongs to NP and is NP-hard.



NP-complete problems

Definition (NP-hard and NP-complete problems)

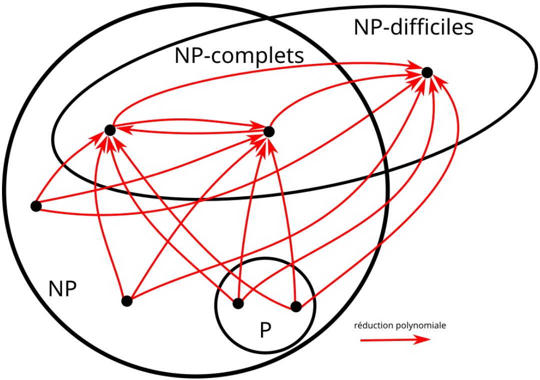
Decision problem P_1 is NP-hard if all problems in NP admit a reduction to P_1.
Decision problem P_1 is NP-complete if it belongs to NP and is NP-hard.



NP-complete problems

Definition (NP-hard and NP-complete problems)

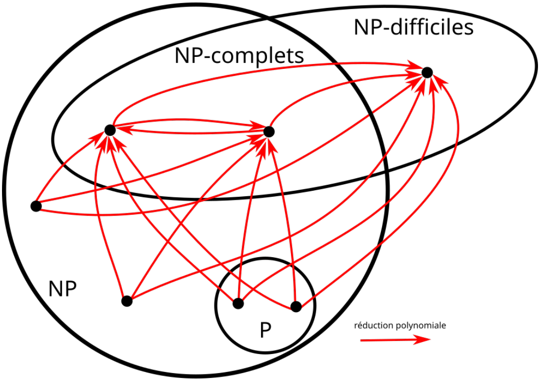
Decision problem P_1 is NP-hard if all problems in NP admit a reduction to P_1.
Decision problem P_1 is NP-complete if it belongs to NP and is NP-hard.



NP-complete problems

Definition (NP-hard and NP-complete problems)

Decision problem P_1 is NP-hard if all problems in NP admit a reduction to P_1.
Decision problem P_1 is NP-complete if it belongs to NP and is NP-hard.



Intuitively : An NP-hard problem is “at least as hard” as all problems of NP (up to polynomial factors).

The first NP-complete problem

SAT
Input: A boolean formula F in CNF
Question: is F satisfiable ?

The first NP-complete problem

SAT
Input: A boolean formula F in CNF
Question: is F satisfiable ?

Theorem (Cook-Levin, 1971)

SAT *is NP-complete.*

Stephen Cook (1939-)

Leonid Levin (1948-)

Richard C. Karp (1935-)

The first NP-complete problem

SAT
Input: A boolean formula F in CNF
Question: is F satisfiable ?

Theorem (Cook-Levin, 1971)

SAT *is NP-complete.*

Stephen Cook (1939-)

Leonid Levin (1948-)

Richard C. Karp (1935-)

3-SAT
Input: A boolean formula F in 3-CNF
Question: Is F satisfiable ?

The first NP-complete problem

SAT
Input: A boolean formula F in CNF
Question: is F satisfiable ?

Theorem (Cook-Levin, 1971)

__SAT__ is NP-complete.

Stephen Cook (1939-)

Leonid Levin (1948-)

Richard C. Karp (1935-)

3-SAT
Input: A boolean formula F in 3-CNF
Question: Is F satisfiable ?

Theorem (Karp, 1972)

__3-SAT__ is NP-complete.

The first NP-complete problem

SAT
Input: A boolean formula F in CNF
Question: is F satisfiable ?

Theorem (Cook-Levin, 1971)

__SAT__ is NP-complete.

Stephen Cook (1939-)

Leonid Levin (1948-)

Richard C. Karp (1935-)

3-SAT
Input: A boolean formula F in 3-CNF
Question: Is F satisfiable ?

Theorem (Karp, 1972)

__3-SAT__ is NP-complete.

Consequence

If one finds a polynomial-time algorithm for __SAT__ or __3-SAT__, there is one for each problem of NP ! (and we win 1 million US$)

How to show a problem is NP-hard ?

Recall : P_1 is NP-hard = all problems in NP admit a polynomial reduction to P_1

Proposition

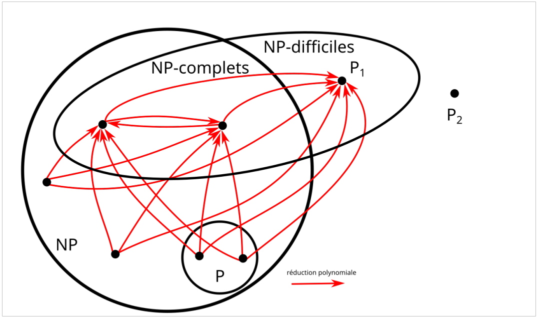
If P_1 is NP-hard and P_1 has a polynomial reduction to P_2, then P_2 is also NP-hard

How to show a problem is NP-hard ?

Recall : P_1 is NP-hard = all problems in NP admit a polynomial reduction to P_1

Proposition

If P_1 is NP-hard and P_1 has a polynomial reduction to P_2, then P_2 is also NP-hard

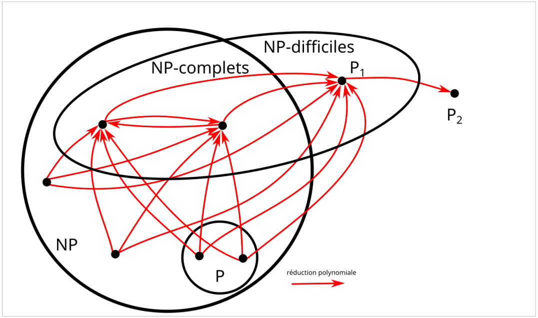


How to show a problem is NP-hard ?

Recall : P_1 is NP-hard = all problems in NP admit a polynomial reduction to P_1

Proposition

If P_1 is NP-hard and P_1 has a polynomial reduction to P_2, then P_2 is also NP-hard

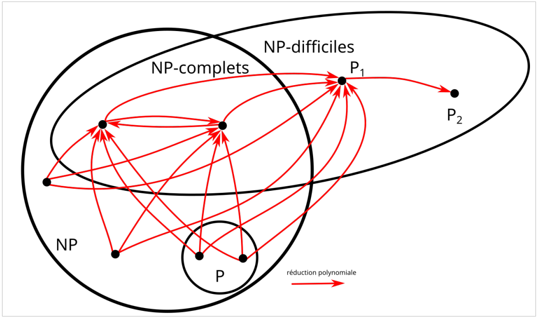


How to show a problem is NP-hard ?

Recall : P_1 is NP-hard = all problems in NP admit a polynomial reduction to P_1

Proposition

If P_1 is NP-hard and P_1 has a polynomial reduction to P_2, then P_2 is also NP-hard



A reduction from 3-SAT to 3-Coloring

3-SAT
Input: A boolean formula F in 3-CNF
Question: Is F satisfiable ?

Example : $(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_3)$

3-Coloring
Input: A graph G
Question: Is G 3-colorable ?

Theorem (Garey, Johnson, Stockmeyer, 1976)

***3-Coloring** is NP-complete.*

A reduction from 3-SAT to 3-Coloring

3-SAT
Input: A boolean formula F in 3-CNF
Question: Is F satisfiable ?

Example : $(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_3)$

3-Coloring
Input: A graph G
Question: Is G 3-colorable ?

Theorem (Garey, Johnson, Stockmeyer, 1976)

3-Coloring *is NP-complete.*

1) **3-Coloring** Is in NP : for a 3-coloring of G (= the certificate), one can check in polynomial time if it is valid (for each edge, check that the colors are distinct).

A reduction from 3-SAT to 3-Coloring

3-SAT
Input: A boolean formula F in 3-CNF
Question: Is F satisfiable ?

Example : $(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_3)$

3-Coloring
Input: A graph G
Question: Is G 3-colorable ?

Theorem (Garey, Johnson, Stockmeyer, 1976)

***3-Coloring** is NP-complete.*

2) **3-Coloring** is NP-hard : build a polynomial reduction from **3-SAT** to **3-Coloring** : for every 3-CNF formula F, create a graph $G(F)$ such that F is satisfiaable $\Leftrightarrow$ $G(F)$ is 3-colorable.

A reduction from 3-SAT to 3-Coloring

3-SAT
Input: A boolean formula F in 3-CNF
Question: Is F satisfiable ?

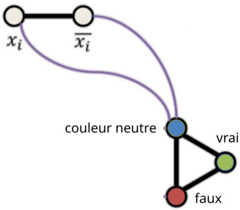
Example : $(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_3)$

3-Coloring
Input: A graph G
Question: Is G 3-colorable ?

Theorem (Garey, Johnson, Stockmeyer, 1976)

__3-Coloring__ is NP-complete.

2.1) model a variable x_i :



A reduction from 3-SAT to 3-Coloring

3-SAT
Input: A boolean formula F in 3-CNF
Question: Is F satisfiable ?

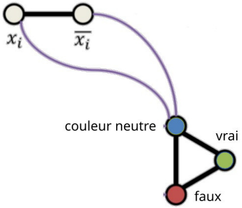
Example : $(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_3)$

3-Coloring
Input: A graph G
Question: Is G 3-colorable ?

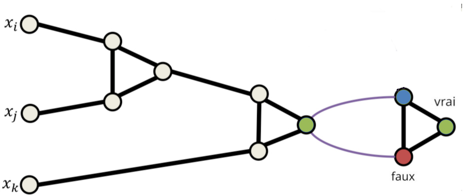
Theorem (Garey, Johnson, Stockmeyer, 1976)

***3-Coloring** is NP-complete.*

2.1) model a variable x_i :



2.2) model a clause $(x_i \vee x_j \vee x_k)$:



A reduction from 3-SAT to 3-Coloring

3-SAT
Input: A boolean formula F in 3-CNF
Question: Is F satisfiable ?

Example : $(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_3)$

3-Coloring
Input: A graph G
Question: Is G 3-colorable ?

Theorem (Garey, Johnson, Stockmeyer, 1976)

3-Coloring *is NP-complete.*

2.3) Put everything together !

A reduction from 3-SAT to 3-Coloring

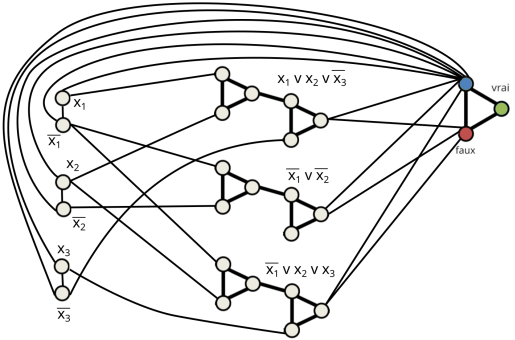
3-SAT
Input: A boolean formula F in 3-CNF
Question: Is F satisfiable ?

Example : $(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_3)$

3-Coloring
Input: A graph G
Question: Is G 3-colorable ?

Theorem (Garey, Johnson, Stockmeyer, 1976)

***3-Coloring** is NP-complete.*



A reduction from 3-SAT to 3-Coloring

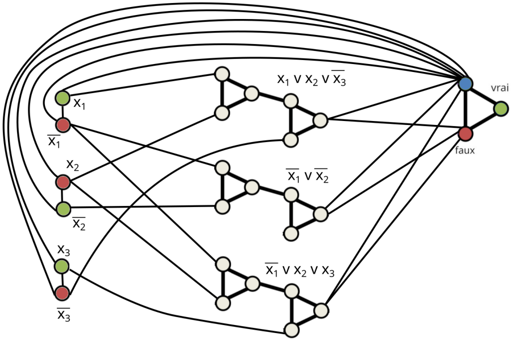
3-SAT
Input: A boolean formula F in 3-CNF
Question: Is F satisfiable ?

Example : $(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_3)$

3-Coloring
Input: A graph G
Question: Is G 3-colorable ?

Theorem (Garey, Johnson, Stockmeyer, 1976)

3-Coloring *is NP-complete.*



If F is satisfiable,
then $G(F)$ is 3-colorable

A reduction from 3-SAT to 3-Coloring

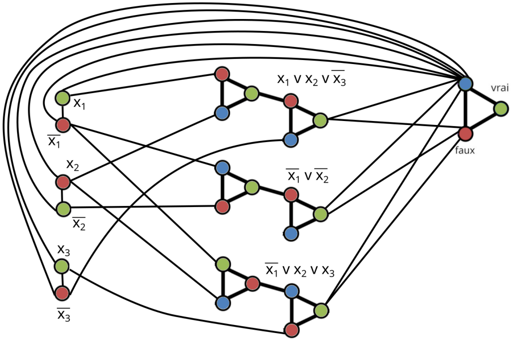
3-SAT
Input: A boolean formula F in 3-CNF
Question: Is F satisfiable ?

Example : $(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_3)$

3-Coloring
Input: A graph G
Question: Is G 3-colorable ?

Theorem (Garey, Johnson, Stockmeyer, 1976)

3-Coloring *is NP-complete.*



If F is satisfiable, then $G(F)$ is 3-colorable

A reduction from 3-SAT to 3-Coloring

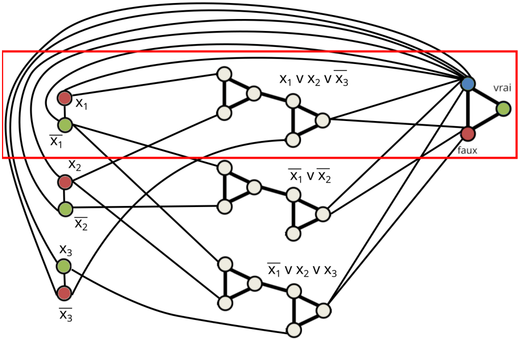
3-SAT
Input: A boolean formula F in 3-CNF
Question: Is F satisfiable ?

Example : $(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_3)$

3-Coloring
Input: A graph G
Question: Is G 3-colorable ?

Theorem (Garey, Johnson, Stockmeyer, 1976)

3-Coloring *is NP-complete.*



If $G(F)$ is 3-colorable, then F is satisfiable

A reduction from 3-SAT to 3-Coloring

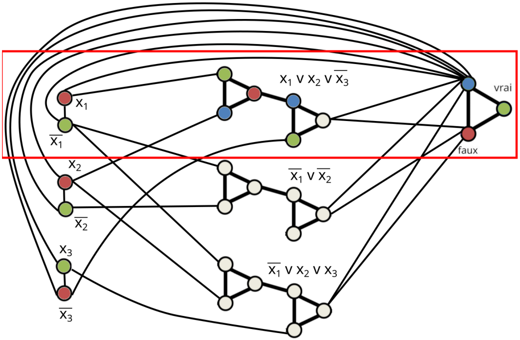
3-SAT
Input: A boolean formula F in 3-CNF
Question: Is F satisfiable ?

Example : $(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_3)$

3-Coloring
Input: A graph G
Question: Is G 3-colorable ?

Theorem (Garey, Johnson, Stockmeyer, 1976)

3-Coloring *is NP-complete.*



If $G(F)$ is 3-colorable, then F is satisfiable

A reduction from 3-SAT to 3-Coloring

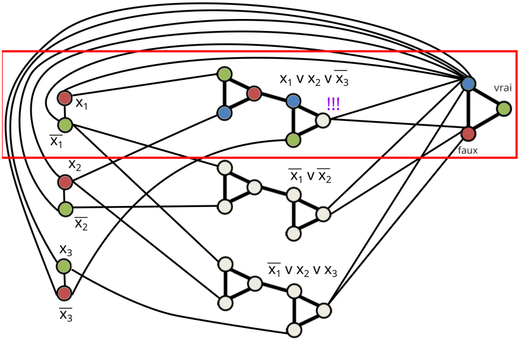
3-SAT
Input: A boolean formula F in 3-CNF
Question: Is F satisfiable ?

Example : $(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_3)$

3-Coloring
Input: A graph G
Question: Is G 3-colorable ?

Theorem (Garey, Johnson, Stockmeyer, 1976)

3-Coloring is NP-complete.



If $G(F)$ is 3-colorable, then F is satisfiable

Super Mario

Super Mario
Input: A Super Mario level.
Question: Can Mario go from start to finish ?

Theorem (Aloupis, Demaine, Guo, 2012)

Super Mario *is NP-hard.*

Reduction from **3-SAT** :