Topic 7. Complexity

Data Structures and Algorithms

FIB

Antoni Lozano
(translated by Albert Oliveras)

Q2 2017–2018
Version of December 19, 2018

Classes

Algorithm analysis studies the amount of resources that an algorithm needs to solve a problem.

Complexity theory considers all possible algorithms that solve the same problem.

- Algorithm analysis focuses on algorithms, whereas complexity theory focuses on problems
- We will study some basic tools to classify problems according to their complexity

Classes

Algorithm analysis studies the amount of resources that an algorithm needs to solve a problem.

Complexity theory considers all possible algorithms that solve the same problem.

- Algorithm analysis focuses on algorithms, whereas complexity theory focuses on problems
- We will study some basic tools to classify problems according to their complexity

Classes

Algorithm analysis studies the amount of resources that an algorithm needs to solve a problem.

Complexity theory considers all possible algorithms that solve the same problem.

- Algorithm analysis focuses on algorithms, whereas complexity theory focuses on problems
- We will study some basic tools to classify problems according to their complexity

Decision problems

In order to better classify problems, we will consider their decision versions.

Definition

A decision problem is a problem where one has to determine whether an instance satisfies a certain property.

Decision problems

Lots of problems seen so far are or can be made decisional.

Some decision problems on graphs:

- **connectivity**: given a graph, determine whether it is connected
- **reachability**: given a graph $G = (V, E)$ and two vertices $i, j \in V$, determine whether there is a path from i to j in G
- **shortest path**: given a graph $G = (V, E)$, two vertices $i, j \in V$ and a natural number k, determine whether there is a path between i and j in G of length at most k
- **longest path**: given a graph $G = (V, E)$, two vertices $i, j \in V$ and a natural number k, determine whether there is a path between i and j in G of length at least k
- **3-colorability**: given a graph, determine whether it is 3-colorable

Decision problems

Some problems do not make sense in their decision version.

Decision n-queens problem (1st version)

Given a natural number n, determine whether we can place n queens on an $n \times n$ board so that no two queens threaten each other.

It is known that there are solutions for all $n \neq 2, 3$. Hence, the following algorithm decides the problem in time $\Theta(1)$.

```
QUEENS(n)
    if n = 2 o n = 3 then
        return FALSE
    else
        return TRUE
```

Decision problems

What is interesting is not whether there is a solution, but finding one.

Decision n-queens problem (2nd version)

Given a natural number n and k values $r_1, \ldots r_k$, with $k \leq n$, determine whether we can place n queens on an $n \times n$ board so that no two queens threaten each other and for all i such that $1 \leq i \leq k$, the queen in row i is in column r_i.

This version, despite being decisional, allows one to find a solution with

$$(n-1) + (n-2) \cdots + 2 = \sum_{i=2}^{n-1} i = \frac{n(n-1)}{2} - 1 \in \Theta(n^2)$$

executions of the algorithm that solves it.

Decision problems

Some other decision problems:

1. **primality**: given a natural number, determine whether it is a prime
2. **traveling salesperson problem (TSP)**: given n cities, the distances among them and a number of kilometers k, determine whether there is a route of at most k kilometers that visits each city exactly once and goes back to the origin

A decision problem is a set consisting of an infinite number of instances.

If a problem consists of a finite number of instances, it can be solved by a constant-time algorithm (e.g. 8-queens).

Decision problems

A decision problem is formally represented as a set.

If T is a property that can be checked on the elements of an instance set E, we can formulate the following decision problem:

Problem A
Given $x \in E$, determine whether $T(x)$ holds.

Formally, A can be described as the set:

$$A = \{ x \in E \mid T(x) \}.$$

Decision problems

The problem instances will belong to some concrete domains such as:

- natural numbers
- tuples of natural numbers
- graphs
- weighted dags
- Boolean formulas

In each case, we will consider a size or length function.

Size function

Given $x \in E$, where E is a domain, the size of x, represented as $|x|$, is the number of symbols of a standard encoding of x.

Decision problems

Given a problem A defined over an input set E, we will distinguish between

- positive instances: the ones belonging to A
- negative instances: the ones belonging to $E - A$

Primality

The primality problem can be described informally

> **Primality** (PRIMES)
> Given a natural number x, determine whether x is prime.

Or formally as the set of positive inputs:

$$\text{PRIMES} = \{x \in \mathbb{N} \mid x \text{ is prime } \}.$$

A size function for the natural numbers counts the number of digits of its binary representation:

$$|x| = \text{ number of digits of } x \text{ in binary} = \lfloor \log_2 x \rfloor + 1.$$

Decision problems

Given a problem A defined over an input set E, we will distinguish between

- positive instances: the ones belonging to A
- negative instances: the ones belonging to $E - A$

Primality

The primality problem can be described informally

Primality (PRIMES)
Given a natural number x, determine whether x is prime.

Or formally as the set of positive inputs:

$$\text{PRIMES} = \{x \in \mathbb{N} \mid x \text{ is prime } \}.$$

A size function for the natural numbers counts the number of digits of its binary representation:

$$|x| = \text{ number of digits of } x \text{ in binary} = \lfloor \log_2 x \rfloor + 1.$$

Decision problems

Once we can describe problems as mathematical objects (decision problems as sets), we can group them into classes according to their complexity.

- We will consider classes of problems that can be solved using a certain amount of resources
- A class groups problems in the same way as a problem groups instances
- We have to distinguish between three levels of abstraction:
    - Instances ——> strings of characters
    - Problems —-> sets of instances
    - Classes ——-> sets of problems

Polynomial and exponential time

Let us assume that $t : \mathbb{N} \to \mathbb{R}^+$ is a function.

Algorithms of cost t

We say that an algorithm $\mathcal{A}$ has cost t if its worst-case cost belongs to $O(t)$.

Problems decidable in time t

If an algorithm $\mathcal{A}$ takes inputs from a set E and has a binary output, we write

$$\mathcal{A} : E \to \{0, 1\}.$$

We say that a decision problem A is decidable in time t if there exists an algorithm $\mathcal{A} : E \to \{0, 1\}$ of cost t such that, for all $x \in E$:

$$x \in A \Rightarrow \mathcal{A}(x) = 1$$

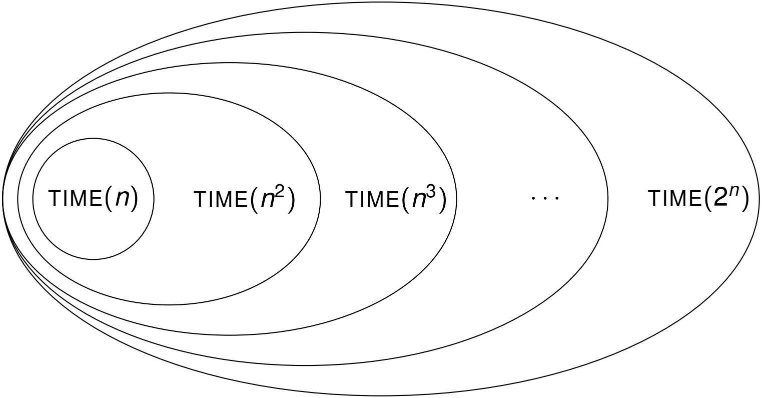
$$x \notin A \Rightarrow \mathcal{A}(x) = 0$$

Polynomial and exponential time

Class TIME(t)

Given a function $t : \mathbb{N} \rightarrow \mathbb{R}^+$, we group the problems decidable in time t:

$$\text{TIME}(t) = \{A \mid A \text{ is decidable in time } t\}.$$



Polynomial and exponential time

We remind that there is a huge difference between having a polynomial or an exponential algorithm for a problem. In Topic 1 we saw two tables showing:

- quantitative differences (table 1)
- qualitative differences (table 2)

between polynomials and exponentials.

Polynomial and exponential time

Table 1 (Garey/Johnson, *Computers and Intractability*)

Comparison between polynomial and exponential functions.

cost	10	20	30	40	50
n	0.00001 s	0.00002 s	0.00003 s	0.00004 s	0.00005 s
n^2	0.0001 s	0.0004 s	0.0009 s	0.0016 s	0.0025 s
n^3	0.001 s	0.008 s	0.027 s	0.064 s	0.125 s
n^5	0.1 s	3.2 s	24.3 s	1.7 min	5.2 min
2^n	0.001 s	1.0 s	17.9 min	12.7 days	35.7 years
3^n	0.059 s	58 min	6.5 years	3855 cents.	2×10^8 cents.

Polynomial and exponential time

Table 2 (Garey/Johnson, *Computers and Intractability*)

Effect of technological improvements on polynomial and exponential algorithms.

cost	current technology	technology $\times 100$	technology $\times 1000$
n	N_1	$100N_1$	$1000N_1$
n^2	N_2	$10N_2$	$31.6N_2$
n^3	N_3	$4.64N_3$	$10N_3$
n^5	N_4	$2.5N_4$	$3.98N_4$
2^n	N_4	$N_4 + 6.64$	$N_4 + 9.97$
3^n	N_5	$N_5 + 4.19$	$N_5 + 6.29$

Polynomial and exponential time

Class P

We define the class P as the union of all polynomial-time classes:

$$\mathrm{P} = \bigcup_{k>0} \mathsf{TIME}(n^k).$$

That is, a problem belongs to P if it is decidable in time n^k for some k.

Class EXP

We define the class EXP as the union as the union of all exponential classes:

$$\mathrm{EXP} = \bigcup_{k>0} \mathsf{TIME}(2^{n^k}).$$

That is, a problem is in EXP if it is decidable in time 2^{n^k} for some k.

Examples

Problems in P:

- **connectivity**
- **reachability**
- **primality**
- **shortest path**
- **2-colorability**

Problems in EXP (not known to be in P):

- **longest path**
- **3-colorability**
- **travelling salesperson problem**

Other problems in EXP:

- **generalized chess**, **checkers** and **go**

Theorem

$\mathrm{P} \subsetneq \mathrm{EXP}$.

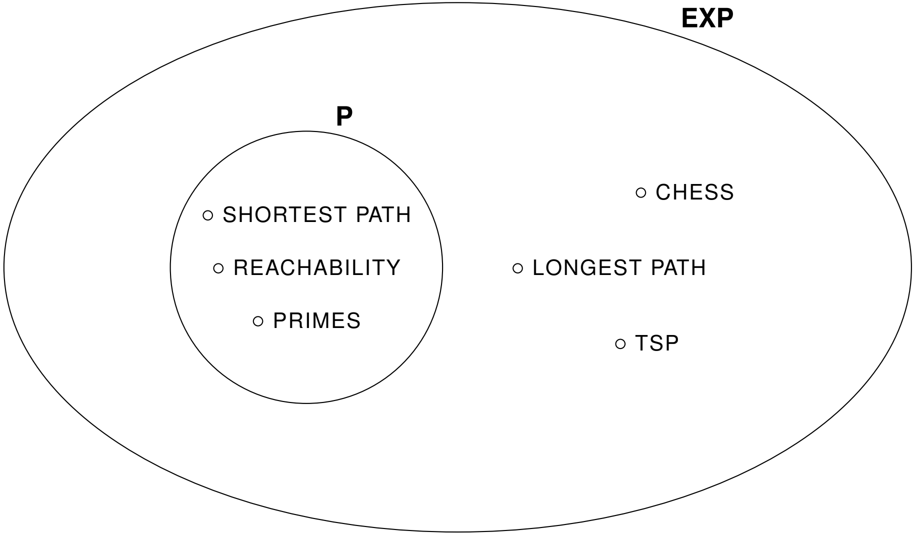
Strict inclusion in the theorem can be divided into two parts:

1. $\mathrm{P} \subseteq \mathrm{EXP}$. Obvious from the definitions:

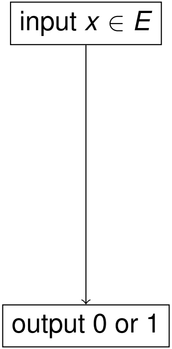
$$\mathrm{P} = \bigcup_{k>0} \mathsf{TIME}(n^k) \subseteq \bigcup_{k>0} \mathsf{TIME}(2^{n^k}) = \mathrm{EXP}$$

2. $\mathrm{P} \neq \mathrm{EXP}$. Proved using the diagonalization technique

Polynomial and exponential time



- Algorithms seen so far are deterministic: they follow a unique computation path from the input to the output
- The execution of an algorithm $\mathcal{A} : E \rightarrow \{0, 1\}$ on a domain E can be seen as a path:



Nondeterminism

A nondeterministic algorithm can reach a result via different paths. Its behavior is more similar to a tree.

Nondeterministic algorithms (informal idea)

An algorithm $\mathcal{A} : E \to \{0, 1\}$ is *nondeterministic* if it can use a new function

$$\text{CHOOSE}(y)$$

that, for an input x and $y \leq x$, splits the computation into y branches, and returns a distinct value between 0 and y on each branch.

- **Computation tree**: The computation starts in a deterministic way until the first CHOOSE instruction; for every value returned by CHOOSE, an independent computation branch is generated with the corresponding value
- **Returned value**: We say that $\mathcal{A}$ returns 1 if some branch returns 1; otherwise, $\mathcal{A}$ returns 0
- **Cost**: The cost of $\mathcal{A}$ is that of the branch with highest cost

Nondeterminism

A nondeterministic algorithm can reach a result via different paths. Its behavior is more similar to a tree.

Nondeterministic algorithms (informal idea)

An algorithm $\mathcal{A} : E \to \{0, 1\}$ is *nondeterministic* if it can use a new function

$$\text{CHOOSE}(y)$$

that, for an input x and $y \leq x$, splits the computation into y branches, and returns a distinct value between 0 and y on each branch.

- **Computation tree**: The computation starts in a deterministic way until the first CHOOSE instruction; for every value returned by CHOOSE, an independent computation branch is generated with the corresponding value
- **Returned value**: We say that $\mathcal{A}$ returns 1 if some branch returns 1; otherwise, $\mathcal{A}$ returns 0
- **Cost**: The cost of $\mathcal{A}$ is that of the branch with highest cost

Nondeterminism

Example: Composites

The problem

$$\text{COMPOSITES} = \{x \mid \exists y \;\; 1 < y < x \text{ and } y \text{ divides } x\}$$

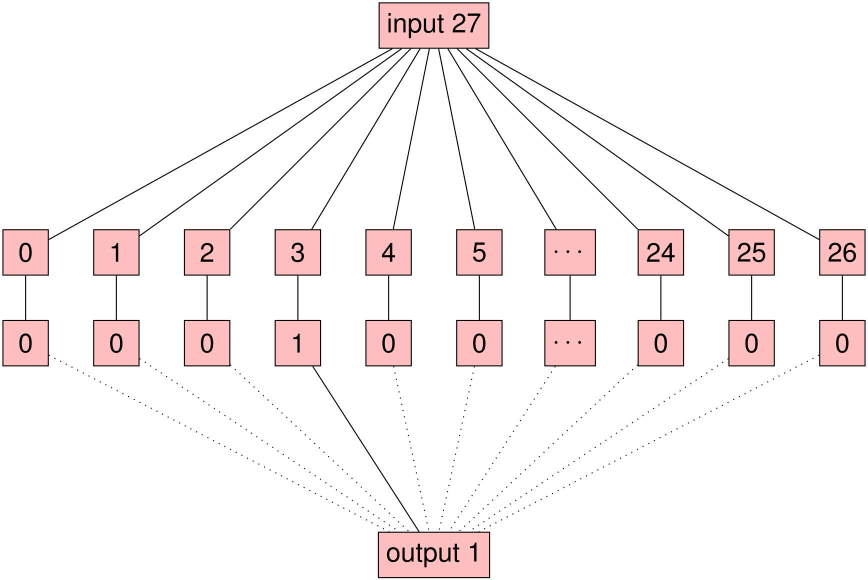
has a trivial exponential deterministic algorithm

> input x
> **for** $y = 2$ **until** $x - 1$
>   **if** y divides x **then**
>   **return** 1
> **return** 0

and a polynomial nondeterministic algorithm

> input x
> $y \leftarrow \text{CHOOSE}(x - 1)$
> **if** $y > 1$ **and** y divides x **then**
>   **return** 1
> **return** 0

Nondeterminism



Nondeterminism

- In the previous example, we say that 3 is a witness of the fact that 27 is not a prime
- That is, in the problem COMPOSITES there exist:
  - Possible witnesses ($y < x$) of the fact that x is composite
  - A polynomial-time verifier algorithm that, given x and y, checks whether y divides x

Nondeterminism

Unlike COMPOSITES, the problem GENERALIZED CHESS has no short witnesses that allow one to check that a player has a winning strategy.

But there are a lot of problems for which it is easy to find short witnesses. For all of them, there are polynomial nondeterministic algorithms.

Nondeterminism

Example: 3-colorability

The 3-colorability problem, represented by the set

$$3\text{-COLOR} = \{\ G \mid G \text{ is 3-colorable }\}$$

has an exponential-time brute-force algorithm

input $G = (V, E)$
$n \leftarrow |V|$
for each tuple $(c_1, \ldots, c_n)$ where $\forall i \leq n \;\; c_i \in \{0, 1, 2\}$
    if $(c_1, \ldots, c_n)$ is a 3-coloring of G **then**
        return 1
return 0

Example: 3-colorability

and a polynomial nondeterministic algorithm

input $G = (V, E)$
$n \leftarrow |V|$
for $i = 1$ **until** n
    $c_i \leftarrow \text{CHOOSE}(2)$
if $(c_1, \ldots, c_n)$ is a 3-coloring of G **then**
    return 1
else
    return 0

Nondeterminism

The formal definition of nondeterministic polynomial algorithms distinguishes:

- the witness computation
- the deterministic computations

Nondeterminism

Decidability in nondeterministic polynomial time

Let Σ be an alphabet and A a decision problem defined over inputs of a set E. We say that A is decidable in nondeterministic polynomial time if there exist

- a polynomial algorithm $\mathcal{V} : E \times \Sigma^* \to \{0, 1\}$ (called verifier) and
- a polynomial $p(n)$

such that for all $x \in E$, we have

$$x \in A \Rightarrow \mathcal{V}(x, y) = 1 \text{ for some } y \in \Sigma^* \text{ such that } |y| = p(|x|)$$

$$x \notin A \Rightarrow \mathcal{V}(x, y) = 0 \text{ for all } y \in \Sigma^* \text{ such that } |y| = p(|x|)$$

If $x \in A$, the y such that $\mathcal{V}(x, y) = 1$ are called witnesses or certificates.

Nondeterminism

In order to know that a problem A is decidable in nondeterministic polynomial time we will have to check that:

1. positive inputs have polynomial-sized witnesses
   (witnesses have to be defined)
2. witnesses can be verified in polynomial time
   (a verifier has to be designed)

Nondeterminism

Composites

Let us consider the problem

$$\text{COMPOSITES} = \{x \mid \exists y \quad 1 < y < x \ \text{ and } y \text{ divides } x\}$$

1. The witnesses for x are all $y \neq 1, x$ that divide x
2. The polynomial is $p(n) = n$
3. The verifier is

$\mathcal{V}(x, y)$
    if $(1 < y < x)$ **and** $(y$ divides $x)$ **then**
        return 1
    else
        return 0

COMPOSITES is decidable in nondeterministic polynomial time because

$$x \in \text{COMPOSITES} \Leftrightarrow \mathcal{V}(x, y) = 1 \text{ for some } y \text{ s.t. } |y| = p(|x|)$$

3-colorability

Let us consider the problem

$$\text{3-COLOR} = \{\ G \mid G \text{ is 3-colorable}\ \}$$

1. The witnesses for $G = (V, E)$ are all 3-colorings C of G of the form $C = (c_1, c_2, \ldots, c_n)$, where $n = |V|$ and $c_i \in \{0, 1, 2\}$ for all $i \leq n$
2. The polynomial (with reasonable encodings of G and C) can be $p(n) = n$
3. The verifier is

$\mathcal{V}(G, C)$
  if C is a 3-coloring of G **then**
    return 1
  else
    return 0

All problems decidable in nondeterministic polynomial time are grouped in one class.

Class NP

We define the class NP (from *nondeterministic polynomial time*) as:

$$\mathrm{NP} = \{A \mid A \text{ is decidable in nondeterministic polynomial time}\}.$$

How does NP compare to P and EXP?

Nondeterminism

All problems decidable in nondeterministic polynomial time are grouped in one class.

Class NP

We define the class NP (from *nondeterministic polynomial time*) as:

$$\mathrm{NP} = \{A \mid A \text{ is decidable in nondeterministic polynomial time}\}.$$

How does NP compare to P and EXP?

Main difference between P and NP:

- solutions to problems in P can be found in polynomial time
- solutions to problems in NP can be verified in polynomial time

Example: Perfect squares and composites

1. SQUARES $= \{x \in \mathbb{N} \mid \exists y \quad 1 \leq y < x \quad \text{and} \quad x = y^2\}$
2. COMPOSITES $= \{x \in \mathbb{N} \mid \exists y \quad 1 < y < x \quad \text{and} \quad y \text{ divides } x\}$

Example: 2 and 3-colorability

1. 2-COLOR $= \{ G \mid G \text{ is 2-colorable} \}$
2. 3-COLOR $= \{ G \mid G \text{ is 3-colorable} \}$

Nondeterminism

Main difference between P and NP:

- solutions to problems in P can be found in polynomial time
- solutions to problems in NP can be verified in polynomial time

Example: Perfect squares and composites

1. SQUARES $= \{x \in \mathbb{N} \mid \exists y \quad 1 \leq y < x \quad \text{and} \quad x = y^2 \}$
2. COMPOSITES $= \{x \in \mathbb{N} \mid \exists y \quad 1 < y < x \quad \text{and} \quad y \text{ divides } x \}$

Example: 2 and 3-colorability

1. 2-COLOR $= \{ G \mid G \text{ is 2-colorable} \}$
2. 3-COLOR $= \{ G \mid G \text{ is 3-colorable} \}$

Nondeterminism

Main difference between P and NP:

- solutions to problems in P can be found in polynomial time
- solutions to problems in NP can be verified in polynomial time

Example: Perfect squares and composites

1. SQUARES $= \{x \in \mathbb{N} \mid \exists y \;\; 1 \leq y < x \;\; \text{and} \;\; x = y^2 \}$
2. COMPOSITES $= \{x \in \mathbb{N} \mid \exists y \;\; 1 < y < x \;\; \text{and} \;\; y \text{ divides } x \}$

Example: 2 and 3-colorability

1. 2-COLOR $= \{ G \mid G \text{ is 2-colorable} \}$
2. 3-COLOR $= \{ G \mid G \text{ is 3-colorable} \}$

Nondeterminism

Theorem

$\mathrm{P} \subseteq \mathrm{NP}$.

Proof

Any deterministic algorithm is nondeterministic (but does not use CHOOSE).

Equivalently, for all $A \in \mathrm{P}$, we can create verifiers $\mathcal{V}$ such that for any x:

$$x \in A \Rightarrow \mathcal{V}(x, y) = 1 \text{ for all } y \in \Sigma^* \text{ such that } |y| = |x|$$

$$x \notin A \Rightarrow \mathcal{V}(x, y) = 0 \text{ for all } y \in \Sigma^* \text{ such that } |y| = |x|$$

To find $\mathcal{V}(x, y)$, it is only needed to simulate $\mathcal{A}(x)$ and return the same value 0 or 1 (independently of y). Hence, $A \in \mathrm{NP}$.

Nondeterminism

Differences between NP and EXP:

- problems in NP have solutions verifiable in polynomial time
- problems in EXP can have exponentially large solutions
- in order to solve problems in NP there is a standard algorithm that searches for a witness, but this is not the case for EXP problems

Nondeterminism

Theorem

$\mathrm{NP} \subseteq \mathrm{EXP}$.

Proof

Let $A \in \mathrm{NP}$. Hence, there is a polynomial $p(n)$ and a verifier $\mathcal{V}$ such that

$$x \in A \Rightarrow \mathcal{V}(x, y) = 1 \text{ for some } y \in \Sigma^* \text{ such that } |y| = p(|x|)$$

$$x \notin A \Rightarrow \mathcal{V}(x, y) = 0 \text{ for all } y \in \Sigma^* \text{ such that } |y| = p(|x|)$$

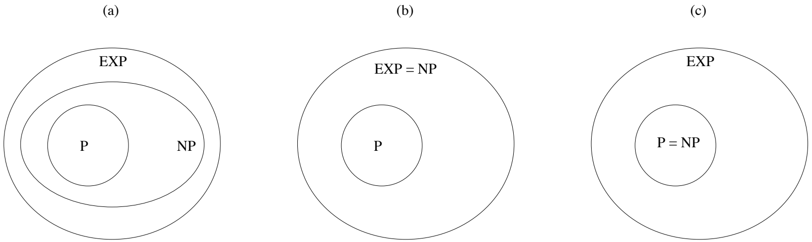
We can consider an exponential algorithm for A that looks for a witness:

```
input x
for all y such that |y| = p(|x|)
    if V(x, y) = 1 then
        return 1
return 0
```

It is easy to see that the previous algorithm is exponential and decides A. Hence, $A \in \mathrm{EXP}$.

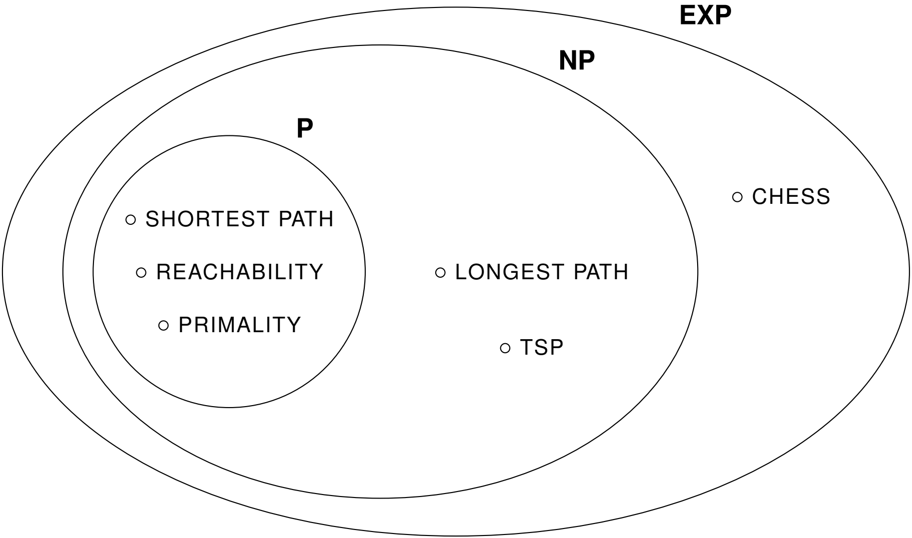
Nondeterminism

- We know that $\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{EXP}$
- We also know that $\mathrm{P} \neq \mathrm{EXP}$
- Thus, we can assure that either $\mathrm{P} \neq \mathrm{NP}$ or $\mathrm{NP} \neq \mathrm{EXP}$ (or both), and we are left with three possibilities:



We will take (a) as our working hypothesis.

Nondeterminism



The cup of tea story

Concept of reduction

Reductions

Let A and B be two decision problems with input sets E and E', respectively. We say *A reduces to B in polynomial time* if there exists a polynomial-time algorithm $\mathcal{F}$ such that

$$x \in A \Rightarrow \mathcal{F}(x) \in B$$

$$x \notin A \Rightarrow \mathcal{F}(x) \notin B$$

In this case, we write $A \leq^p B$ (or $A \leq^p B$ *via* $\mathcal{F}$) and we say that $\mathcal{F}$ is a polynomial-time reduction from A to B.

Examples of reductions

Parity

Let us consider the language of even numbers

$$\text{EVEN} = \{x \in \mathbb{N} \mid \exists y \in \mathbb{N} \quad x = 2y\}$$

and that of odd numbers

$$\text{ODD} = \{x \in \mathbb{N} \mid \exists y \in \mathbb{N} \quad x = 2y + 1\}$$

As one can see, EVEN reduces to ODD via an algorithm $\mathcal{F}$ that adds 1 to the input: $\mathcal{F}(x) = x + 1$. It is obvious that for all x:

$$x \in \text{EVEN} \Leftrightarrow \mathcal{F}(x) \in \text{ODD}.$$

In this case, one can also reduce ODD to EVEN using the same algorithm $\mathcal{F}$. That is, ODD $\leq^p$ EVEN via $\mathcal{F}$.

Examples of reductions

Parity

Let us consider the language of even numbers

$$\text{EVEN} = \{x \in \mathbb{N} \mid \exists y \in \mathbb{N} \quad x = 2y\ \}$$

and that of odd numbers

$$\text{ODD} = \{x \in \mathbb{N} \mid \exists y \in \mathbb{N} \quad x = 2y + 1\ \}$$

As one can see, EVEN reduces to ODD via an algorithm $\mathcal{F}$ that adds 1 to the input: $\mathcal{F}(x) = x + 1$. It is obvious that for all x:

$$x \in \text{EVEN} \Leftrightarrow \mathcal{F}(x) \in \text{ODD}.$$

In this case, one can also reduce ODD to EVEN using the same algorithm $\mathcal{F}$. That is, ODD $\leq^p$ EVEN via $\mathcal{F}$.

Examples of reductions

Partitions

Consider the following two problems:

Partition
Given natural numbers $x_1, x_2, \ldots, x_n$, determine whether they can be divided into two groups having the same sum.

Knapsack
Given natural numbers $x_1, x_2, \ldots, x_n$ and a capacity $C \in \mathbb{N}$, determine whether there is a selection of the x_i's that sums exactly C.

Formally:

$$\text{PARTITION} = \{(x_1, \ldots, x_n) \mid \exists I \subseteq \{1, \ldots, n\} \quad \sum_{i \in I} x_i = \sum_{i \notin I} x_i\}$$

$$\text{KNAPSACK} = \{(x_1, \ldots, x_n, C) \mid \exists I \subseteq \{1, \ldots, n\} \quad \sum_{i \in I} x_i = C\}$$

Examples of reductions

Partitions

Consider the following two problems:

Partition
Given natural numbers $x_1, x_2, \ldots, x_n$, determine whether they can be divided into two groups having the same sum.

Knapsack
Given natural numbers $x_1, x_2, \ldots, x_n$ and a capacity $C \in \mathbb{N}$, determine whether there is a selection of the x_i's that sums exactly C.

Formally:

$$\text{PARTITION} = \{(x_1, \ldots, x_n) \mid \exists I \subseteq \{1, \ldots, n\} \quad \sum_{i \in I} x_i = \sum_{i \notin I} x_i\}$$

$$\text{KNAPSACK} = \{(x_1, \ldots, x_n, C) \mid \exists I \subseteq \{1, \ldots, n\} \quad \sum_{i \in I} x_i = C\}$$

Examples of reductions

Partitions

The algorithm

$\mathcal{F}(x_1, \ldots, x_n)$
    $S \leftarrow \sum_{i=1}^{n} x_i$
    if S is odd **then**
        return $(x_1, \ldots, x_n, S+1)$
    else
        return $(x_1, \ldots, x_n, S/2)$

is a polynomial-time reduction from PARTITION to KNAPSACK:

$$(x_1, \ldots, x_n) \in \text{PARTITION} \Leftrightarrow \mathcal{F}(x_1, \ldots, x_n) \in \text{KNAPSACK}.$$

Exercise

We define the following collection of coloring problems:

k-**Colorability** (k-COLOR)
Given an undirected graph G, determine whether the vertices in G can be colored with at most k colors, so that each pair of adjacent vertices of get different colors.

Prove that, for all $k \geq 1$, it holds that:

$$k\text{-COLOR} \leq^p (k+1)\text{-COLOR}.$$

Examples of reductions

Definition

A Hamiltonian path in a graph G is a path in G containing all of its vertices without repetitions.

Exercise

We define the Hamiltonian path problem (HP) i and the Hamiltonian path problem between two points (HP_2) as:

- $\mathrm{HP} = \{G \mid G \text{ has a Hamiltonian path}\}$
- $\mathrm{HP}_2 = \{ (G, u, v) \mid G \text{ has a Hamiltonian path with endpoints } u, v\}$

Propose:

1. a reduction proving $\mathrm{HP} \leq^p \mathrm{HP}_2$
2. a reduction proving $\mathrm{HP}_2 \leq^p \mathrm{HP}$

Properties

Properties: Reflexivity

For all A, $A \leq^p A$.

We can consider the algorithm that computes the identity function:

$\mathcal{F}(x)$
return x

It is obvious that, for all x

$$x \in A \Leftrightarrow \mathcal{F}(x) = x \in A.$$

Properties

Properties: Transitivity

For all A, B, C, if $A \leq^p B$ and $B \leq^p C$, then $A \leq^p C$.

If

- $A \leq^p B$ via $\mathcal{F}$ and
- $B \leq^p C$ via $\mathcal{G}$,

then the composition $\mathcal{G} \circ \mathcal{F}$ ($\mathcal{F}|\mathcal{G}$ in UNIX *pipe* notation) proves that $A \leq^p C$.

We will consider that $\mathcal{G} \circ \mathcal{F}(x) = \mathcal{G}(\mathcal{F}(x))$.

Properties

Exercise

Prove that

$$3\text{-}\textsc{color} \leq^p k\text{-}\textsc{color}$$

for all $k \geq 4$ by two different methods:

1. using transitivity of reductions
2. providing an explicit reduction

Properties

Corollary

Reductions form a preorder.

Question

Observe that, although reductions form a preorder, they do not form a partial order due to the fact that they do not satisfy antisymmetry:

- $\forall A, B \quad A \leq^p B \wedge B \leq^p A \Rightarrow A = B$

Properties

Corollary

Reductions form a preorder.

Question

Observe that, although reductions form a preorder, they do not form a partial order due to the fact that they do not satisfy antisymmetry:

- $\forall A, B \quad A \leq^p B \wedge B \leq^p A \Rightarrow A = B$

Properties

Closure of P under reductions

For all A, B, if $A \leq^p B$ and $B \in \mathrm{P}$, then $A \in \mathrm{P}$.

If

- $\mathcal{B}$ is a polynomial algorithm for B and
- $\mathcal{F}$ is a polynomial algorithm that proves $A \leq^p B$,

then the composition $\mathcal{F} \circ \mathcal{B}$ is a polynomial algorithm for A:

1. $\mathcal{B} \circ \mathcal{F}$ is polynomial since it is a composition of polynomial-time algorithms
2. $\mathcal{B} \circ \mathcal{F}(x)$ accepts $\Leftrightarrow$ $\mathcal{B}$ accepts $\mathcal{F}(x) \Leftrightarrow \mathcal{F}(x) \in B \Leftrightarrow x \in A$

Properties

Closure of P under reductions

For all A, B, if $A \leq^p B$ and $B \in \mathrm{P}$, then $A \in \mathrm{P}$.

If

- $\mathcal{B}$ is a polynomial algorithm for B and
- $\mathcal{F}$ is a polynomial algorithm that proves $A \leq^p B$,

then the composition $\mathcal{F} \circ \mathcal{B}$ is a polynomial algorithm for A:

1. $\mathcal{B} \circ \mathcal{F}$ is polynomial since it is a composition of polynomial-time algorithms
2. $\mathcal{B} \circ \mathcal{F}(x)$ accepts $\Leftrightarrow$ $\mathcal{B}$ accepts $\mathcal{F}(x) \Leftrightarrow \mathcal{F}(x) \in B \Leftrightarrow x \in A$

Properties

Closure of P under reductions

For all A, B, if $A \leq^p B$ and $B \in \mathrm{P}$, then $A \in \mathrm{P}$.

If

- $\mathcal{B}$ is a polynomial algorithm for B and
- $\mathcal{F}$ is a polynomial algorithm that proves $A \leq^p B$,

then the composition $\mathcal{F} \circ \mathcal{B}$ is a polynomial algorithm for A:

1. $\mathcal{B} \circ \mathcal{F}$ is polynomial since it is a composition of polynomial-time algorithms
2. $\mathcal{B} \circ \mathcal{F}(x)$ accepts $\Leftrightarrow \mathcal{B}$ accepts $\mathcal{F}(x) \Leftrightarrow \mathcal{F}(x) \in B \Leftrightarrow x \in A$

Properties

Notation: Polynomial equivalence

Given two decision problems A, B, we write $A \equiv^p B$ if $A \leq^p B$ and $B \leq^p A$.

Problem: Equivanlence classes of P

1. Prove that $\equiv^p$ is an equivalence relation (reflexive, symmetric, and transitive)
2. Prove that for all A, B, if $A \in \mathrm{P}$ and $B \neq \emptyset, \Sigma^*$, then $A \leq^p B$
3. Obtain the partition of P into equivalence classes induced by relation $\equiv^p$

Properties

Notation: Polynomial equivalence

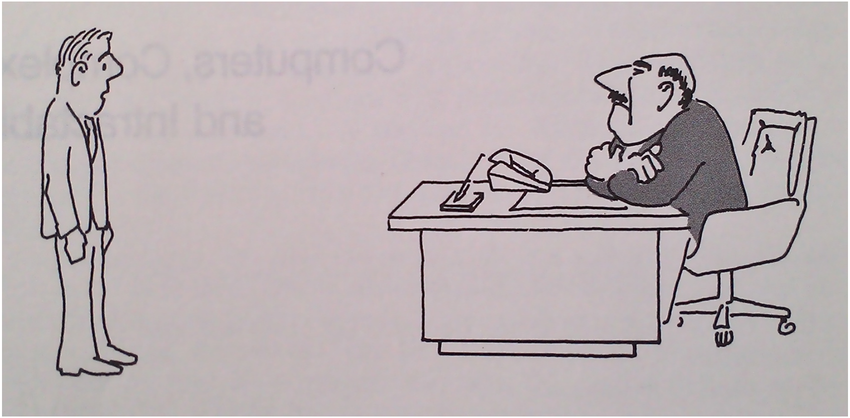
Given two decision problems A, B, we write $A \equiv^p B$ if $A \leq^p B$ and $B \leq^p A$.

Problem: Equivanlence classes of P

1. Prove that $\equiv^p$ is an equivalence relation (reflexive, symmetric, and transitive)
2. Prove that for all A, B, if $A \in \mathrm{P}$ and $B \neq \emptyset, \Sigma^*$, then $A \leq^p B$
3. Obtain the partition of P into equivalence classes induced by relation $\equiv^p$

NP-completeness theory

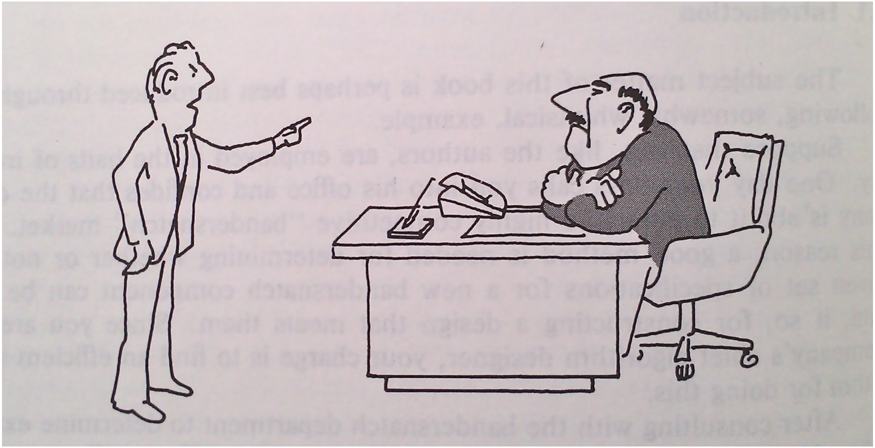
– I can't find an efficient algorithm, I guess I'm just too dumb.



Garey & Johnson, *Computers and Intractability*

NP-completeness theory

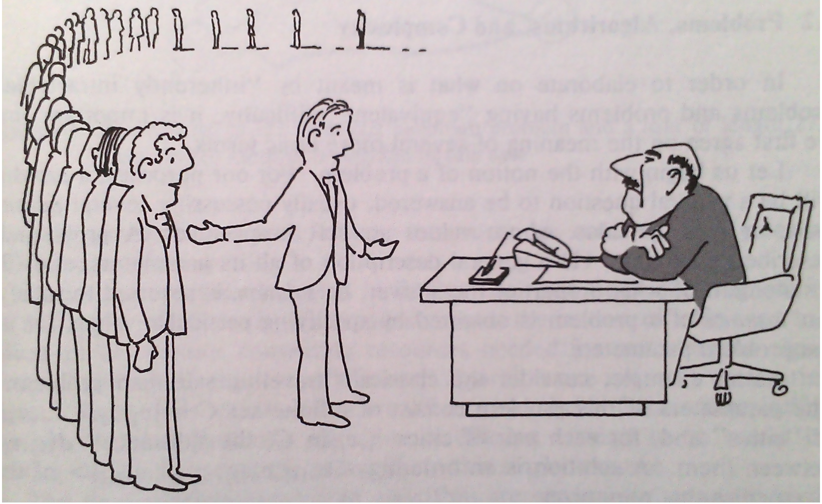
– I can't find an efficient algorithm because no such algorithm is possible!



Garey & Johnson, *Computers and Intractability*

NP-completeness theory

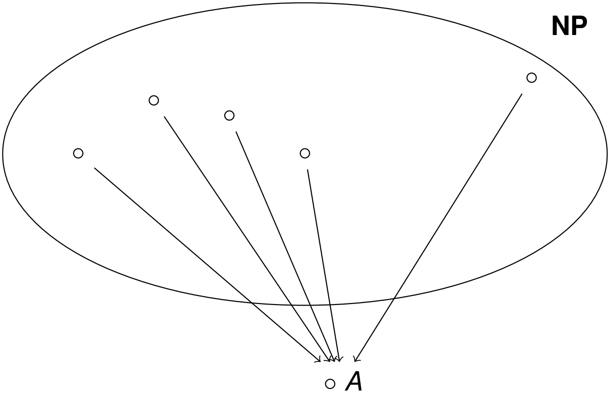
– I can't find an efficient algorithm, but neither can all these famous people.



Garey & Johnson, *Computers and Intractability*

Definition

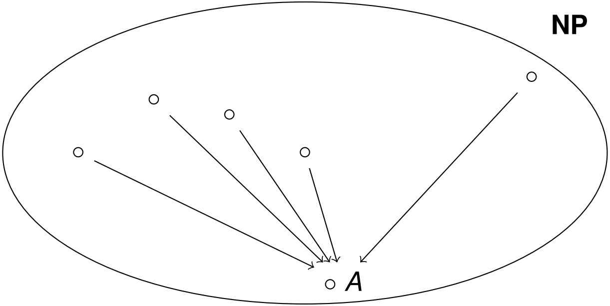
A problem A is NP-hard if for any problem $B \in \mathrm{NP}$ it holds that $B \leq^p A$.



NP-completeness theory

Definition

A problem A is NP-complete if it is NP-hard and $A \in \mathrm{NP}$.



NP-completeness theory

Any NP-complete problem "represents" the whole NP class in relation to P.

More formally...

Proposition

Let A be an NP-complete problem. Then, $\mathrm{P} = \mathrm{NP}$ if and only if $A \in \mathrm{P}$.

$\Rightarrow$ Since A is NP-complete, $A \in \mathrm{NP}$ and hence $A \in \mathrm{P}$.

$\Leftarrow$ Let $A \in \mathrm{P}$.

1. Due to the closure of P under reductions, we know that for all B such that $B \leq^p A$ we have $B \in \mathrm{P}$.
2. Since A is NP-complete, we know that for all $B \in \mathrm{NP}$, $B \leq^p A$.

Using 1 and 2, $\mathrm{NP} \subseteq \mathrm{P}$ and hence $\mathrm{P} = \mathrm{NP}$.

Any NP-complete problem "represents" the whole NP class in relation to P.

More formally...

Proposition

Let A be an NP-complete problem. Then, $\mathrm{P} = \mathrm{NP}$ if and only if $A \in \mathrm{P}$.

$\Rightarrow$ Since A is NP-complete, $A \in \mathrm{NP}$ and hence $A \in \mathrm{P}$.

$\Leftarrow$ Let $A \in \mathrm{P}$.

1. Due to the closure of P under reductions, we know that for all B such that $B \leq^p A$ we have $B \in \mathrm{P}$.
2. Since A is NP-complete, we know that for all $B \in \mathrm{NP}$, $B \leq^p A$.

Using 1 and 2, $\mathrm{NP} \subseteq \mathrm{P}$ and hence $\mathrm{P} = \mathrm{NP}$.

NP-completeness theory

Any two NP-complete problems are equivalent.

More formally...

Definition

We write $A \equiv^p B$ when $A \leq^p B$ and $B \leq^p A$.

Proposition

If A and B are NP-complete, then $A \equiv^p B$.

Since A and B are NP-complete, we have

1. $A \in \mathrm{NP}$ and
2. B is NP-hard

and then, $A \leq^p B$.

Symmetrically, we can argue that $B \leq^p A$. Therefore, $A \equiv^p B$.

NP-completeness theory

Any two NP-complete problems are equivalent.

More formally...

Definition

We write $A \equiv^p B$ when $A \leq^p B$ and $B \leq^p A$.

Proposition

If A and B are NP-complete, then $A \equiv^p B$.

Since A and B are NP-complete, we have

1. $A \in \mathrm{NP}$ and
2. B is NP-hard

and then, $A \leq^p B$.

Symmetrically, we can argue that $B \leq^p A$. Therefore, $A \equiv^p B$.

NP-completeness theory

Any two NP-complete problems are equivalent.

More formally...

Definition

We write $A \equiv^p B$ when $A \leq^p B$ and $B \leq^p A$.

Proposition

If A and B are NP-complete, then $A \equiv^p B$.

Since A and B are NP-complete, we have

1. $A \in \mathrm{NP}$ and
2. B is NP-hard

and then, $A \leq^p B$.

Symmetrically, we can argue that $B \leq^p A$. Therefore, $A \equiv^p B$.

NP-completeness theory

But...do NP-complete problems exist?

Boolean formulas

- A Boolean formula (BF) is a formula over Boolean variables with no quantifiers
- We will use the connectives:
  $\vee$ (disjunction), $\wedge$ (conjunction) and $\neg$ (negation)

For example,

$$F(x, y, z) = (x \vee y \vee \neg z) \wedge \neg(x \wedge y \wedge z)$$

is a Boolean formula.

Conjunctive Normal Form (CNF)

- A literal is a positive or negative variable (x, $\neg x$)
- A clause is a disjunction of literals ($x \vee \neg y \vee z$)
- A Boolean formula is in CNF if it is a conjunction of clauses

$$F(x, y, z) = (x \vee \neg y \vee z) \wedge (\neg x \vee \neg z)$$

Boolean formulas

- A Boolean formula (BF) is a formula over Boolean variables with no quantifiers
- We will use the connectives:
  $\vee$ (disjunction), $\wedge$ (conjunction) and $\neg$ (negation)

For example,

$$F(x, y, z) = (x \vee y \vee \neg z) \wedge \neg(x \wedge y \wedge z)$$

is a Boolean formula.

Conjunctive Normal Form (CNF)

- A literal is a positive or negative variable (x, $\neg x$)
- A clause is a disjunction of literals ($x \vee \neg y \vee z$)
- A Boolean formula is in CNF if it is a conjunction of clauses

$$F(x, y, z) = (x \vee \neg y \vee z) \wedge (\neg x \vee \neg z)$$

Satisfiability

A Boolean formula is satisfiable if there exists an assignment from variables to $\{0,1\}$ under which the formula evaluates to true. For example,

$$F(x,y,z) = (x \vee \neg y \vee z) \wedge (\neg x \vee \neg z)$$

is satisfiable with $x = 1$, $y = 0$, $z = 0$. We write $F(100) = 1$.

We define

$$\text{SAT} = \{\ F \mid F \text{ is a satisfiable Boolean formula}\ \}$$

$$\text{CNF-SAT} = \{\ F \mid F \text{ is a satisfiable BF in CNF}\ \}$$

Cook-Levin Theorem (1971)

CNF-SAT is NP-complete.

Satisfiability

A Boolean formula is satisfiable if there exists an assignment from variables to $\{0, 1\}$ under which the formula evaluates to true. For example,

$$F(x, y, z) = (x \vee \neg y \vee z) \wedge (\neg x \vee \neg z)$$

is satisfiable with $x = 1$, $y = 0$, $z = 0$. We write $F(100) = 1$.

We define

$$\text{SAT} = \{\ F \mid F \text{ is a satisfiable Boolean formula}\ \}$$

$$\text{CNF-SAT} = \{\ F \mid F \text{ is a satisfiable BF in CNF}\ \}$$

Cook-Levin Theorem (1971)

CNF-SAT is NP-complete.

Cook-Levin Theorem (1971)

CNF-SAT is NP-complete.

In order to prove Cook-Levin theorem, we need to show:

1. CNF-SAT $\in \mathrm{NP}$
2. CNF-SAT is NP-hard

(1) CNF-SAT $\in \mathrm{NP}$

- The witnesses are functions from Boolean variables to $\{0, 1\}$.
- In any reasonable encoding of a formula F with n variables, $n \leq |F|$. Since a witness α has n bits, $|\alpha| = n \leq |F|$.
- Hence, choosing $p(n) = n$, we have that $|\alpha| \leq p(|F|)$.
- We can verify whether an assignment α satisfies F in polynomial time:
  - replace variables by their values given by α
  - evaluate the connectives bottom up

Example

If we consider the following BF in CNF

$$F(x, y, z) = (x \vee \neg y \vee z) \wedge (x \vee \neg z)$$

and the assignment $\alpha = 100$ (that is, $x = 1$, $y = 0$, $z = 0$), the verifier would evaluate:

- $F(\alpha) = (1 \vee \neg 0 \vee 0) \wedge (1 \vee \neg 0)$ (replace values)
- $F(\alpha) = (1 \vee 1 \vee 0) \wedge (1 \vee 1)$ (negations)
- $F(\alpha) = 1 \wedge 1$ (disjunctions)
- $F(\alpha) = 1$ (conjunctions)

Lemma

Given an algorithm $\mathcal{A} : E \to \{0, 1\}$ with worst-case polynomial-space cost, we can find a BF in CNF $F_{\mathcal{A}}$ in polynomial time such that for all $y \in E$:

$$F_{\mathcal{A}}(y) = 1 \Leftrightarrow \mathcal{A}(y) = 1$$

(2) CNF-SAT is NP-hard.

Let $A \in \mathrm{NP}$. Then, there is a polynomial q and a verifier $\mathcal{V}$ s.t. for all x:

$$x \in A \Leftrightarrow \exists y \quad |y| = q(|x|) \wedge \mathcal{V}(x, y) = 1.$$

Let $\mathcal{V}_x(y)$ be a new verifier, for a fixed x, such that

$$\mathcal{V}_x(y) = 1 \Leftrightarrow |y| = q(|x|) \wedge \mathcal{V}(x, y) = 1.$$

Then,

$$x \in A \Leftrightarrow \exists y \;\; F_{\mathcal{V}_x}(y) \;\; \Leftrightarrow F_{\mathcal{V}_x}(y) \in \text{CNF-SAT}.$$

Hence, $A \leq^p$ CNF-SAT.

NP-completeness theory

Finding a first NP-complete problem (CNF-SAT) makes it possible to find others via reductions.

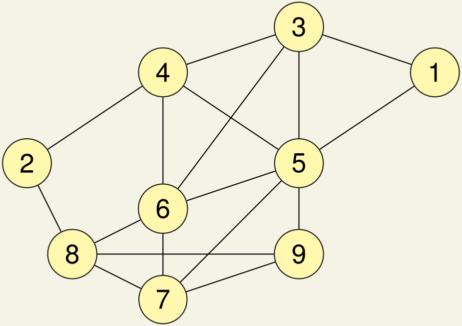
NP-complete problems

Clique problem

We say that H is a complete subgraph of G if it contains all possible edges among its vertices, i.e., if H is isomorphic to K_i for some i. Now define

$$\text{CLIQUE} = \{ (G, k) \mid G \text{ has a complete subgraph with } k \text{ vertices} \}.$$

Given graph G



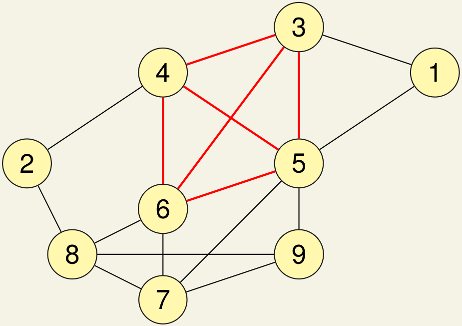
NP-complete problems

Clique problem

We say that H is a complete subgraph of G if it contains all possible edges among its vertices, i.e., if H is isomorphic to K_i for some i. Now define

$$\text{CLIQUE} = \{ (G, k) \mid G \text{ has a complete subgraph with } k \text{ vertices} \}.$$

Given graph G



observe that $(G, 4) \in \text{CLIQUE}$ but $(G, 5) \notin \text{CLIQUE}$.

NP-complete problems

Theorem

CLIQUE is NP-complete

In order to prove that CLIQUE is NP-complete we have to see that:

1. CLIQUE $\in \mathrm{NP}$
2. CLIQUE is NP-hard

(1) CLIQUE $\in \mathrm{NP}$

Let (G, k) be an instance of CLIQUE.

- Witnesses are the vertices of a k-sized complete subgraph of G (in the previous example, the set $C = \{3, 4, 5, 6\}$)
- The polynomial $p(n) = n$ is enough because a witness C satisfies $|C| \leq |(G, k)| = p(|(G, k)|)$
- We can verify in polynomial time whether a set C is a witness: any pair of vertices in C should have an edge in G ($\binom{n}{2} \leq n^2$ checks)

NP-complete problems

Theorem

CLIQUE is NP-complete

In order to prove that CLIQUE is NP-complete we have to see that:

1. CLIQUE $\in \mathrm{NP}$
2. CLIQUE is NP-hard

(2) CLIQUE is NP-hard

We will prove that CNF-SAT $\leq^p$ CLIQUE. Then,

- Since CNF-SAT is NP-hard, any $S \in \mathrm{NP}$ satisfies $S \leq^p$ CNF-SAT
- By transitivity, any $S \in \mathrm{NP}$ satisfies $S \leq^p$ CLIQUE
- Hence, CLIQUE is NP-hard

NP-complete problems

We can express the previous property in general.

Proposition

Let A be an NP-complete problem and B a problem such that $B \in \mathrm{NP}$ and $A \leq^p B$. Then, B is also NP-complete.

- Since A is NP-hard, any $S \in \mathrm{NP}$ satisfies $S \leq^p A$
- By transitivity, any $S \in \mathrm{NP}$ satisfies $S \leq^p B$
- Hence, B is NP-hard

NP-complete problems

CNF-SAT $\leq^p$ CLIQUE

Let F be a Boolean formula in CNF with:

- clauses $C_1, \ldots, C_m$
- literals $l_1, \ldots, l_r$

We define the reduction algorithm $\mathcal{R}(F) = (G, m)$, where $G = (V, E)$ is:

- $V = \{(i, j) \mid \; l_i \text{ appears in } C_j \}$
  (Vertices represent occurrences of literals in clauses)
- $E = \{ \{(i, j), (k, l)\} \mid j \neq l \wedge \neg l_i \neq l_k \}$
  (Edges represent pairs of literals that can be simultaneously true)

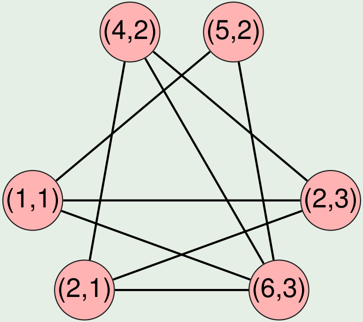
NP-complete problems

Example

$F(x_1, x_2, x_3) = C_1 \wedge C_2 \wedge C_3$, where

- $C_1 = (x_1 \vee x_2)$, $C_2 = (\neg x_1 \vee \neg x_2)$, $C_3 = (x_2 \vee \neg x_3)$
- $l_1 = x_1$, $l_2 = x_2$, $l_3 = x_3$, $l_4 = \neg x_1$, $l_5 = \neg x_2$, $l_6 = \neg x_3$

The reduction $\mathcal{R}(F) = (G, 3)$, where G is the graph



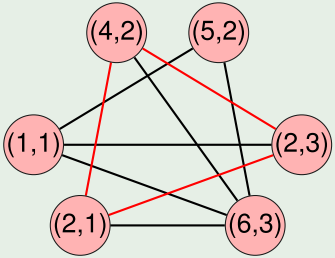
NP-complete problems

In general, we have that $F \in$ CNF-SAT $\Leftrightarrow (G, m) \in$ CLIQUE:

$\Rightarrow$ Let α be an assignment satisfying F. Hence, there are m literals that α simultaneously satisfies and hence they form a complete subgraph in G.

$\Leftarrow$ If G has a complete subgraph with m vertices, each vertex belongs to a different clause. Hence, we can simultaneously satisfy one literal in each clause, thus satisfying F.

Previous example with $l_2 = 1, l_4 = 1$



NP-complete problems

Definitions

- H is an independent subset of G if it consists of isolated vertices
- H is a vertex cover of G if it has an endpoint of any edge in G

Exercise

Given the following problems:

- CLIQUE $= \{ (G, k) \mid$ G has a complete subgraph with k vertices $\}$
- IS $= \{ (G, k) \mid$ G has an independent subset of k vertices $\}$
- VC $= \{ (G, k) \mid$ G has a vertex cover of k vertices $\}$

prove that

1. CLIQUE $\leq^p$ IS
2. IS $\leq^p$ VC
3. VC $\leq^p$ CLIQUE

Lots of NP-complete problems have “particular cases” that are in P.

For example, in CNF-SAT we can fix the number of literals per clause in order to obtain an infinite family of problems.

***k*-Bounded Satisfiability** (*k*-SAT)
Given a Boolean formula in CNF over *n* variables and at most *k* literals per clause, determine whether it is satisfiable.

We will see how to classify *k*-SAT for the different values of *k*.

NP-complete problems

1-Bounded Satisfiability (1-SAT)
Given a Boolean formula F in CNF with n variables and 1 literal per clause, determine whether it is satisfiable.

For example,

$$F(x, y, z, t) = (x) \wedge (\neg y) \wedge (z) \wedge (\neg t).$$

1-SAT is decidable in polynomial time with the following algorithm:

input F
if F has two contradictory literals **then**
  return 0
else
  return 1

1-Bounded Satisfiability (1-SAT)
Given a Boolean formula F in CNF with n variables and 1 literal per clause, determine whether it is satisfiable.

For example,

$$F(x, y, z, t) = (x) \wedge (\neg y) \wedge (z) \wedge (\neg t).$$

1-SAT is decidable in polynomial time with the following algorithm:

input F
if F has two contradictory literals **then**
  return 0
else
  return 1

NP-complete problems

2-Bounded Satisfiability (2-SAT)
Given a Boolean formula F in CNF with n variables and ≤ 2 literals per clause, determine whether it is satisfiable.

For example,

$$F(x, y, z) = (x \vee y) \wedge (x \vee \neg z) \wedge (\neg x \vee y) \wedge (\neg y \vee \neg z).$$

2-SAT is decidable in polynomial time

- transforming the formula into a directed graph
- applying a paths algorithm to the graph

2-Bounded Satisfiability (2-SAT)
Given a Boolean formula F in CNF with n variables and ≤ 2 literals per clause, determine whether it is satisfiable.

For example,

$$F(x, y, z) = (x \vee y) \wedge (x \vee \neg z) \wedge (\neg x \vee y) \wedge (\neg y \vee \neg z).$$

2-SAT is decidable in polynomial time

- transforming the formula into a directed graph
- applying a paths algorithm to the graph

NP-complete problems

Sketch of the algorithm

Given a 2-CNF Boolean formula

$$F(x,y,z) = (x \vee y) \wedge (x \vee \neg z) \wedge (\neg x \vee y) \wedge (\neg y \vee \neg z)$$

it can be rewritten using implications

$$F(x,y,z) = (\neg x \Rightarrow y) \wedge (z \Rightarrow x) \wedge (x \Rightarrow y) \wedge (y \Rightarrow \neg z)$$

that are based on the equivalences

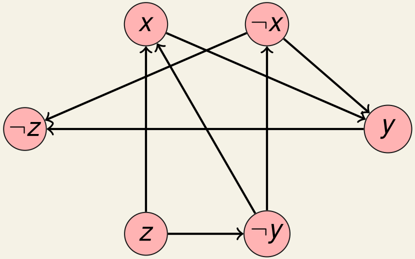
- $(a \vee b) \equiv (\neg a \Rightarrow b) \equiv (\neg b \Rightarrow a)$
- $(a) \equiv (a \vee a) \equiv (\neg a \Rightarrow a) \equiv (a \Rightarrow \neg a)$

NP-complete problems

The Boolean formula with implications

$$F(x, y, z) = (\neg x \Rightarrow y) \wedge (z \Rightarrow x) \wedge (x \Rightarrow y) \wedge (y \Rightarrow \neg z)$$

is transformed into a digraph D_F and we apply the following lemma.



Lemma

F is unsatisfiable if and only if $\exists x$ for which D_F has paths from x to $\neg x$ and from $\neg x$ to x.

NP-complete problems

3**-Bounded Satisfiability** (3-SAT)
Given a Boolean formula F in CNF with n variables and $\leq$ 3 literals per clause, determine whether it is satisfiable.

Theorem
3-SAT is NP-complete.

To prove it, we need two facts:

1. 3-SAT $\in$ NP
(similar to CNF-SAT)
2. 3-SAT is NP-hard
(we show CNF-SAT $\leq^P$ 3-SAT)

NP-complete problems

3-Bounded Satisfiability (3-SAT)
Given a Boolean formula F in CNF with n variables and ≤ 3 literals per clause, determine whether it is satisfiable.

Theorem

3-SAT is NP-complete.

To prove it, we need two facts:

1. 3-SAT $\in \mathrm{NP}$
   (similar to CNF-SAT)
2. 3-SAT is NP-hard
   (we show CNF-SAT $\leq^p$ 3-SAT)

CNF-SAT $\leq^p$ 3-SAT

The following method transforms a Boolean formula in CNF into an equisatisfiable one in 3-CNF.

Given a BF F in CNF,

1. Let F' be an empty BF
2. For each clause $C = (a_1 \vee \cdots \vee a_k)$ in F:
   - if $k \leq 3$, add C to F'
   - if $k > 3$, add the clause

     $$(a_1 \vee a_2 \vee z_1) \wedge (\neg z_1 \vee a_3 \vee z_2) \wedge (\neg z_2 \vee a_4 \vee z_3) \ldots (\neg z_{k-3} \vee a_{k-1} \vee a_k)$$

     to F', where $z_1, \ldots, z_{k-3}$ are new variables.
3. Return F'

NP-complete problems

Example

Given a clause with five literals $C = (a_1 \vee a_2 \vee a_3 \vee a_4 \vee a_5)$, the reduction returns

$$C' = (a_1 \vee a_2 \vee z_1) \wedge (\neg z_1 \vee a_3 \vee z_2) \wedge (\neg z_2 \vee a_4 \vee a_5).$$

- It is obvious that if C is true with assignment α, C' can be satisfied with α and appropriate values for z_1 and z_2
- If C' is true with assignment β, some a_i will be true and C will be true with β

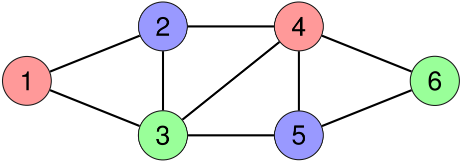
NP-complete problems

Definition

A graph $G = (V, E)$ with n vertices is *k*-colorable if there exists a total function

$$\chi : V \to \{1, \dots, k\}$$

such that $\chi(u) \neq \chi(v)$ for any edge $\{u, v\} \in E$. Function χ is a *k*-coloring.



3-coloring

NP-complete problems

With the number of colors k as an external parameter, we can formulate the coloring problem as a function of k.

k-**Colorability** (k-COLOR)
Given a graph G, determine whether it is k-colorable.

Polynomial algorithms are known for the following cases:

- 1-COLOR
- 2-COLOR

For 3-COLOR, we prove NP-completeness:

- We already showed that 3-COLOR $\in$ NP
- Next, we show that it is NP-complete via a reduction from 3-CNF-SAT

NP-complete problems

With the number of colors k as an external parameter, we can formulate the coloring problem as a function of k.

> k-**Colorability** (k-COLOR)
> Given a graph G, determine whether it is k-colorable.

Polynomial algorithms are known for the following cases:

- 1-COLOR
- 2-COLOR

For 3-COLOR, we prove NP-completeness:

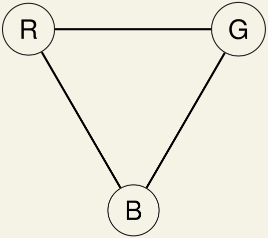
- We already showed that 3-COLOR $\in \mathrm{NP}$
- Next, we show that it is NP-complete via a reduction from 3-CNF-SAT

NP-complete problems

CNF-SAT $\leq^p$ 3-COLOR

Let F be a Boolean formula in CNF. We will construct a graph G that is 3-colorable if and only if F is satisfiable.

- There will be 3 special vertices called R, G, B forming a triangle:

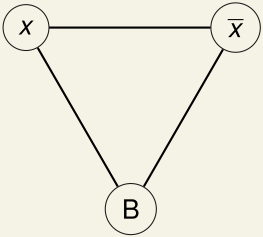


We can assume that in any coloring, vertices R, G, B have the colors:

R $\rightarrow$ red, G $\rightarrow$ green, B $\rightarrow$ blue

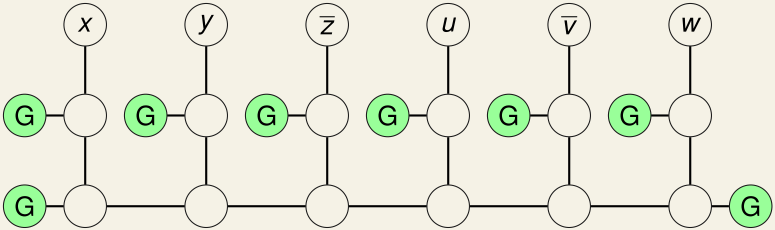
NP-complete problems

- We add a vertex for each literal. Then, we connect each literal and its negation to vertex B.



- For each clause, we add a subgraph as follows. In the case

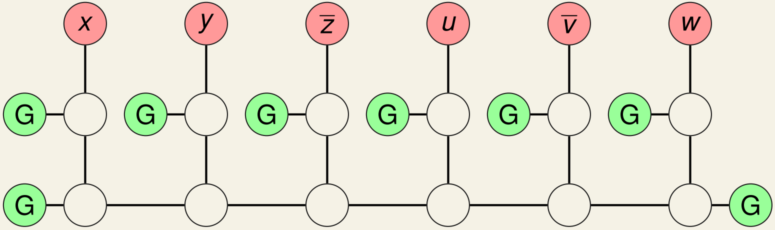
$$(x \vee y \vee \overline{z} \vee u \vee \overline{v} \vee w).$$



Property: A coloring of the upper vertices with red or green can be extended to a global 3-coloring if and only if at least one has green color.

NP-complete problems

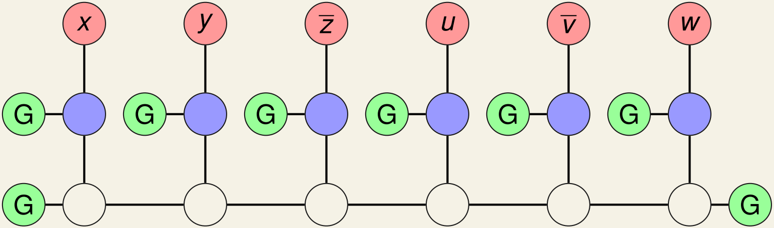
If all of the above are red....



...we cannot complete the 3-coloring.

NP-complete problems

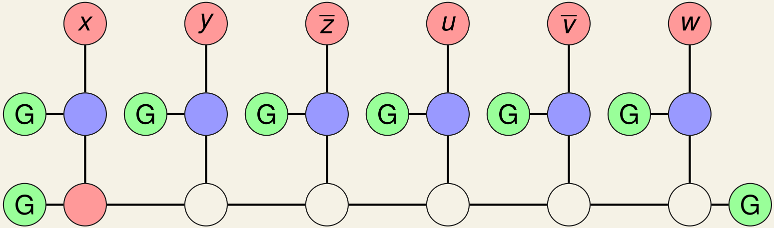
If all of the above are red....



...we cannot complete the 3-coloring.

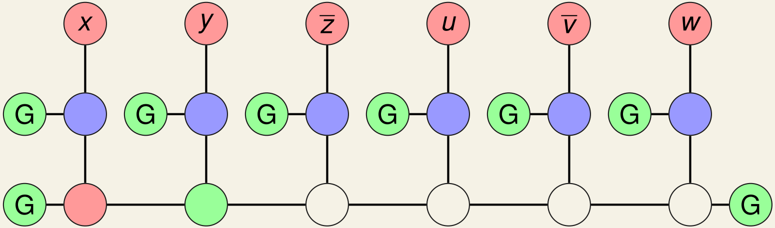
NP-complete problems

If all of the above are red....



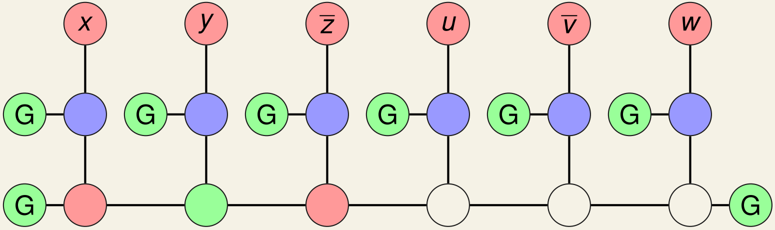
...we cannot complete the 3-coloring.

If all of the above are red....



...we cannot complete the 3-coloring.

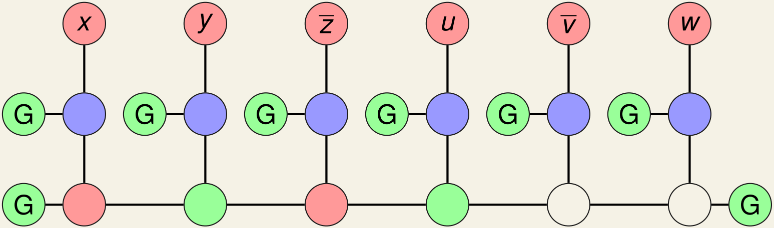
If all of the above are red....



...we cannot complete the 3-coloring.

NP-complete problems

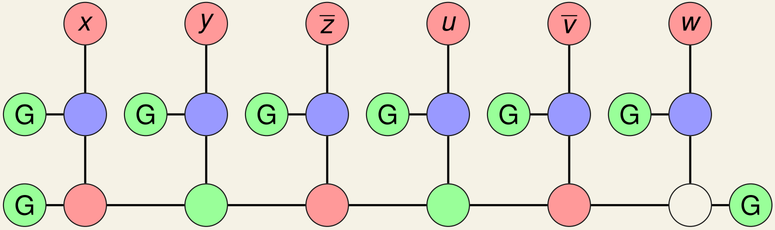
If all of the above are red....



...we cannot complete the 3-coloring.

NP-complete problems

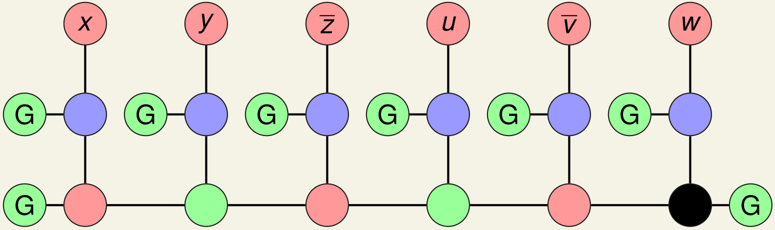
If all of the above are red....



...we cannot complete the 3-coloring.

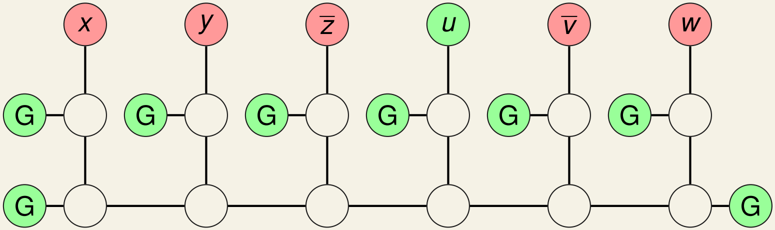
NP-complete problems

If all of the above are red....



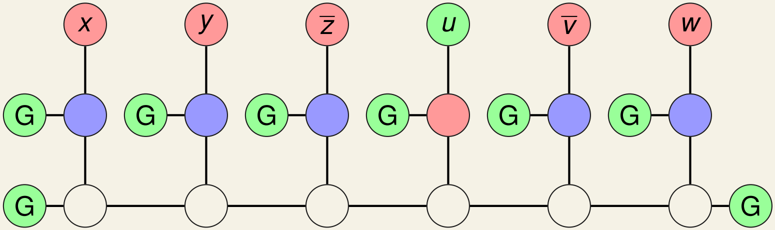
...we cannot complete the 3-coloring.

If at least one is green...



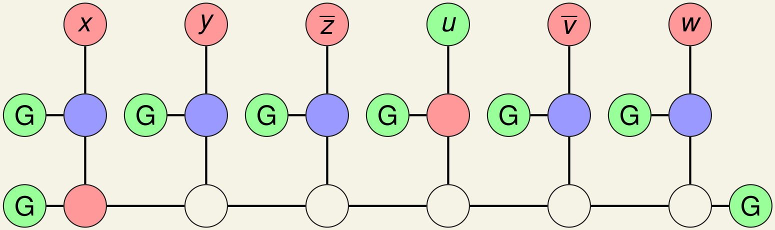
...we can obtain a global 3-coloring.

If at least one is green...



...we can obtain a global 3-coloring.

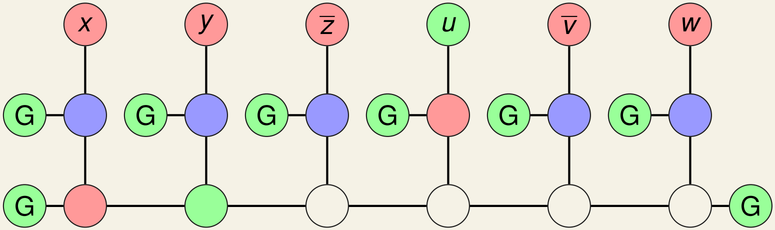
If at least one is green...



...we can obtain a global 3-coloring.

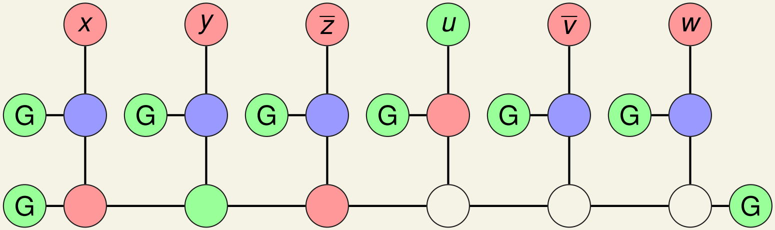
NP-complete problems

If at least one is green...



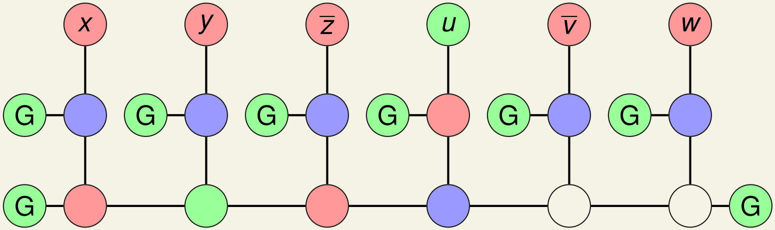
...we can obtain a global 3-coloring.

If at least one is green...



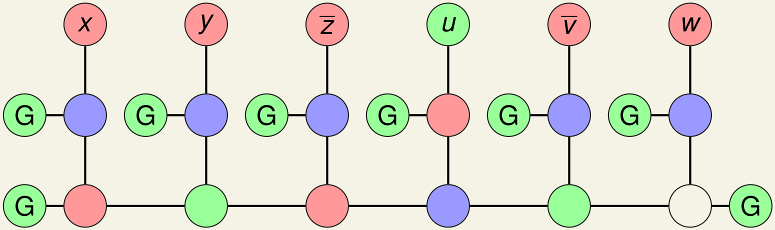
...we can obtain a global 3-coloring.

If at least one is green...



...we can obtain a global 3-coloring.

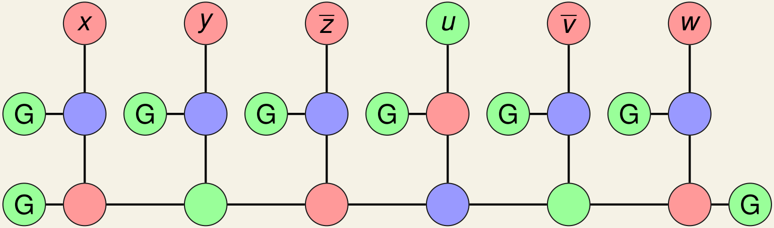
If at least one is green...



...we can obtain a global 3-coloring.

NP-complete problems

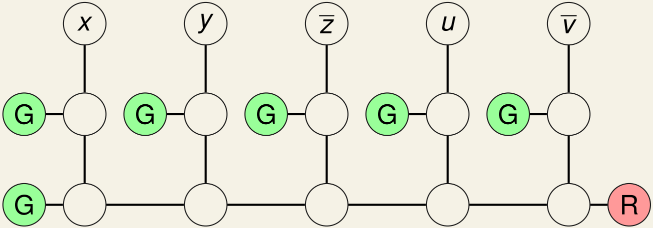
If at least one is green...



...we can obtain a global 3-coloring.

If the number of literals is odd, the rightmost vertex will be R.
For example,

$$(x \vee y \vee \overline{z} \vee u \vee \overline{v})$$



NP-complete problems

If G is the graph with all vertices and edges defined as before, then

$$F \text{ is satisfiable} \Leftrightarrow G \text{ is 3-colorable.}$$

Since G can be constructed in polynomial time, we have that

$$\text{CNF-SAT} \leq^p \text{3-COLOR}.$$

Theorem

3-COLOR is NP-complete.

NP-complete problems

For the other k-COLOR problems, we have the following.

Proposition

For all $k > 3$, 3-COLOR $\leq^p$ k-COLOR.

The reduction consists of, given a graph G, adding to it a complete subgraph with $k - 3$ vertices connected to all vertices of G.

Corollary

For all $k > 3$, k-COLOR is NP-complete.

Hence, we have:

- k-COLOR $\in$ P for all $k \leq 2$
- k-COLOR is NP-complete for all $k \geq 3$

NP-complete problems

For the other k-COLOR problems, we have the following.

Proposition

For all $k > 3$, 3-COLOR $\leq^p$ k-COLOR.

The reduction consists of, given a graph G, adding to it a complete subgraph with $k - 3$ vertices connected to all vertices of G.

Corollary

For all $k > 3$, k-COLOR is NP-complete.

Hence, we have:

- k-COLOR $\in \mathrm{P}$ for all $k \leq 2$
- k-COLOR is NP-complete for all $k \geq 3$

NP-complete problems

What can we say about colorability of planar graphs? Let us consider the following family of problems.

k-**Planar Colorability** (k-COLOR-PL)
Given a planar graph G, determine whether it is k-colorable.

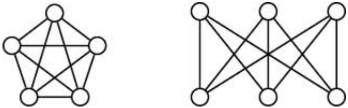
Planarity can be checked in polynomial time.

NP-complete problems

Definition

A graph is planar if it can be drawn on the plane without any edge intersection.

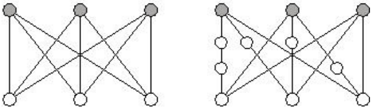
Planar graphs have applications in circuit design and graphics.



K_5 $K_{3,3}$

Kuratowski Theorem

A graph is planar if and only if it does not contain a subgraph homeomorphic to K_5 or $K_{3,3}$.



$K_{3,3}$ and homeomorphic graph

NP-complete problems

Kuratowski Theorem

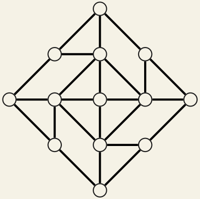
A graph is planar if and only if it does not contain a subgraph homeomorphic to K_5 or $K_{3,3}$.

Planarity test

- **Brute force**: $O(n^6)$
  - Contract edges of degree 2
  - Check whether some set of 5 vertices is K_5
  - Check whether some set of 6 vertices is $K_{3,3}$
- **Efficient**: $O(n)$
  - Apply DFS

3-COLOR $\leq^p$ 3-COLOR-PL

Given a graph G, we will considered a representation of G, possibly with edge intersections. Each intersection will be replaced by the gadget W:



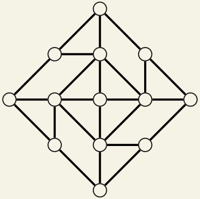
W has interesting properties:

1. in any 3-coloring of W, opposite extreme points have the same color
2. any color assignment where opposite extreme points have the same color can be extended to a 3-coloring of W

NP-complete problems

3-COLOR $\leq^p$ 3-COLOR-PL

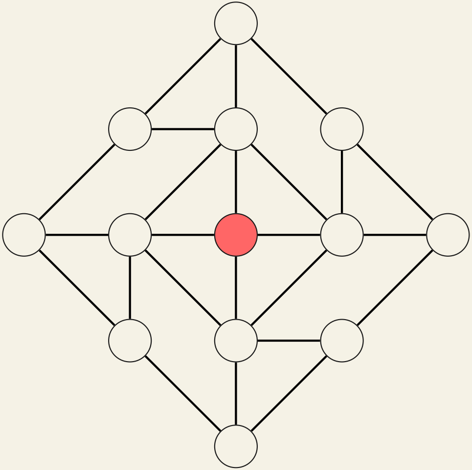
Given a graph G, we will considered a representation of G, possibly with edge intersections. Each intersection will be replaced by the gadget W:



W has interesting properties:

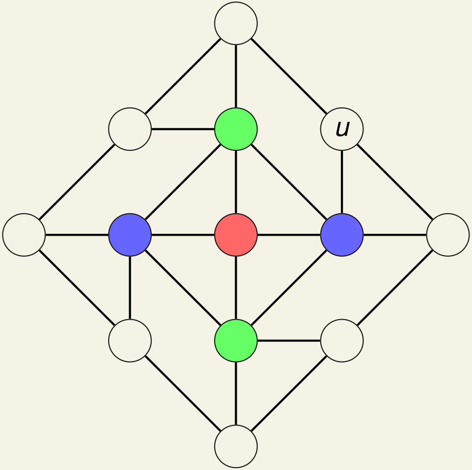
1. in any 3-coloring of W, opposite extreme points have the same color
2. any color assignment where opposite extreme points have the same color can be extended to a 3-coloring of W

NP-complete problems



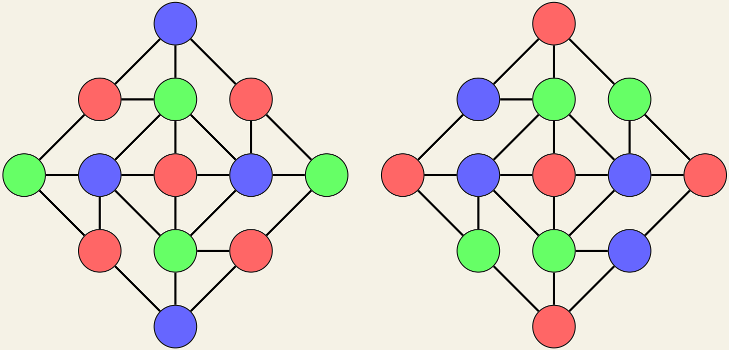
There are two colors available for vertex u.

NP-complete problems



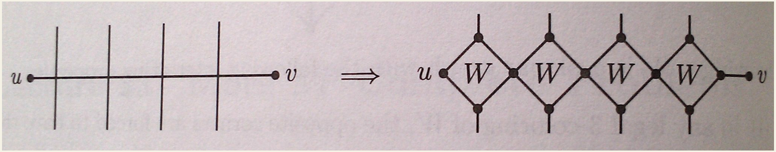
There are two colors available for vertex *u*.

This allows two colorings (up to isomorphism).



It is easy to check that they fullfill properties (1) i (2).

The graph we obtain after the replacements

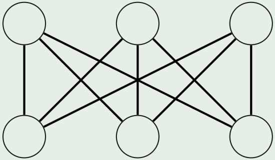


in the representation of *G*

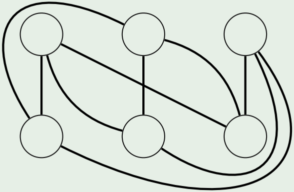
- is planar and
- is 3-colorable if and only if *G* is 3-colorable

Example

Let us assume that we have $K_{3,3}$ as input to 3-COLOR:

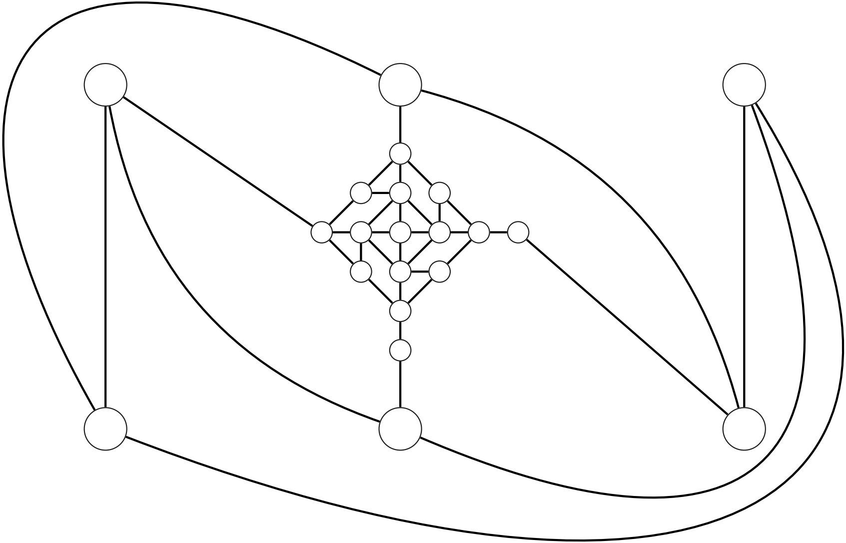


But we consider the following representation with just one intersection:



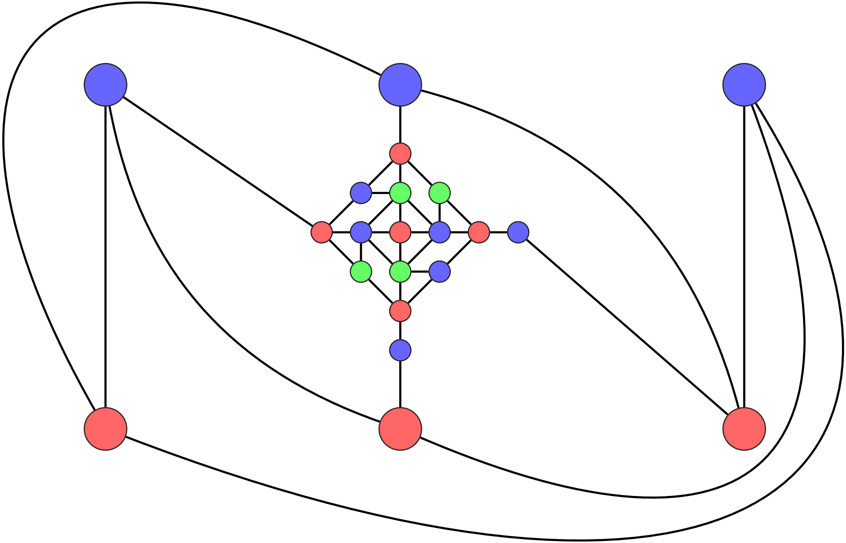
NP-complete problems

A 3-coloring for $K_{3,3}$ induces a 3-coloring for the this graph (and viceversa):



NP-complete problems

A 3-coloring for $K_{3,3}$ induces a 3-coloring for the this graph (and viceversa):



NP-complete problems

Corollary

3-COLOR-PL is NP-complete.

Hence, we have:

- k-COLOR-PL $\in$ P for all $k \leq 2$
- 3-COLOR-PL is NP-complete
- k-COLOR-PL $\in$ P for all $k \geq 4$

Corollary

3-COLOR-PL is NP-complete.

Hence, we have:

- k-COLOR-PL $\in \mathrm{P}$ for all $k \leq 2$
- 3-COLOR-PL is NP-complete
- k-COLOR-PL $\in \mathrm{P}$ for all $k \geq 4$
  (due to the 4-color theorem)

NP-complete problems

So far, we have seen the following tree of reductions.