2. Complexity Theory

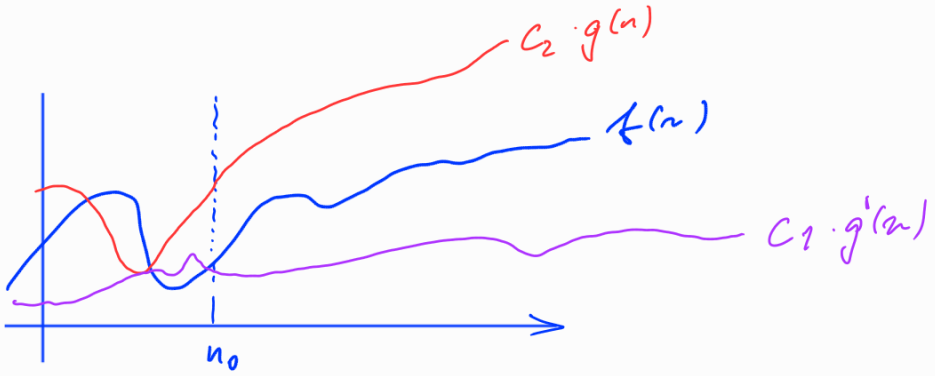
Short Recap runtime (O-, Θ-, Ω- - notation)

given an arbitrary function $g: \mathbb{N} \to \mathbb{R}$

- $\Theta(g(n)) = \{ f(n) : \exists$ pos. constants c_1, c_2, n_0 s.t. $0 \le c_1 \cdot g(n) \le f(n) \le c_2 (g(n))$ $\forall n \ge n_0 \}$

  asymptotic sharp boundary

"visual!"



- $O(g(n)) = \{ f(n) :$ $\exists$ pos. const. c, n_0 s.t $0 \le f(n) \le c \cdot g(n)$ $\forall n \ge n_0 \}$

  asympt. upper bound.

- $\Omega(g(n)) = \{ f(n) :$ $\exists$ pos. const. c, n_0 s.t $0 \le c \cdot g(n) \le f(n)$ $\forall n \ge n_0 \}$

  asympt. lower bound.

Notation: often $f(n) = O(g(n))$ instead of $f(n) \in O(g(n))$ [analog for Ω, Θ]

If the runtime $f(n)$ of an algorithm is in $O(n^k)$ the the algorithm runs in polynomial time

Theorem: $\forall$ Fnctions $f(n), g(n)$:

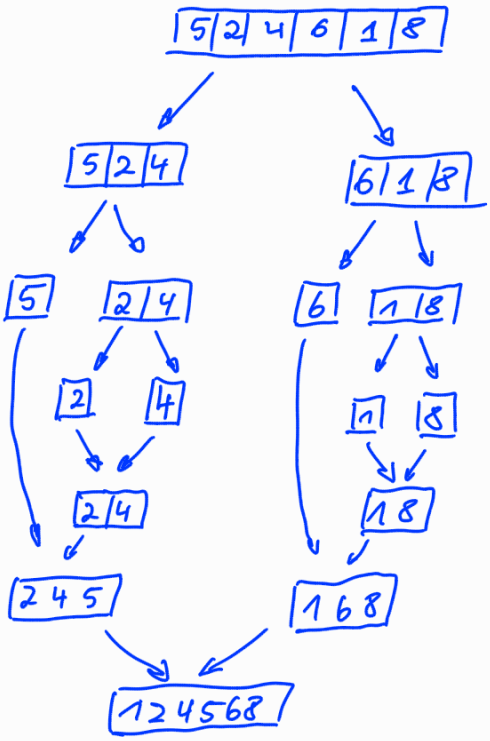
$f(n) \in \Theta(g(n)) \iff f(n) \in O(g(n)) \;\&\; f(n) \in \Omega(g(n))$

In particular, $\Omega(g(n)) \subseteq \Theta(g(n)) \subseteq O(g(n))$

$\&\; \Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$

[proof: Exercise]

Example merge sort vs insertion sort.



5 2 4 6 1 8

5 2

2 5 4

2 4 5 6

2 4 5 6 1

1 2 4 5 6 8

1 2 4 5 6 8

sorted

chosen Element to be inserted into correct pos into already sorted list.

Runtime $c_1 \cdot (n \log_2 n)$ // $c_2 \cdot n^2$ (c_1, c_2 sufficiently large)

Both run in polynomial time

-> should we care about factor $\log_2 n$ vs n?

Assume list has 10^7 entries & can perform 10^9 operations per sec.

WLOG $c_1 = c_2 = 100$.

insertion sort: $\frac{100 \cdot (10^7)^2}{10^9 \text{ ops/sec.}} = 10^7 \text{ sec.} \approx 2780\text{h} \approx 115 \text{ days}$

merge sort: $\frac{100 \cdot 10^7 \cdot \log(10^7)}{10^9 \text{ ops/sec}} = \log(10^7) \text{ sec} \approx 24 \text{ seconds}$

=> RUNTIME MATTERS!

Q: Does there exist an algorithm for all problems to solve it (in polynomial time)?

A: 1) ∃ problem that cannot be solved by algorithms.

2) ∃ problem that can be solved by algorithms but not in polynomial time (under reasonable assumptions)

1) UNSOLVABLE PROBLEMS

HALTING problem.

Is there algorithm to solve the problem: Does a given algorithm terminate?

NO, sketch: Assume there is such an algorithm HALT.

```
HALT ( Alg. A)
  | IF ( A terminates)   return TRUE
  | ELSE                 return FALSE
```

Let Alg. A be:

```
A( )
| WHILE ( HALT (A( )))
```

⇒ 2 cases: A terminates ⇒ HALT returns TRUE
⇒ while-loop in A runs forever
⇒ A does not terminate ↯

A does not terminate ⇒ HALT returns FALSE
⇒ while loop in A stop
⇒ A terminates ↯

⇒ HALT does not exist.

even more, almost all problems cannot be solved by an algorithms.

Sketch: concrete subset of problems: decision problem (output: yes/No)
eg. "is string in language L?"

Algorithm $\hat{=}$ written as finite text
$\hat{=}$ written as finite binary string
$\hat{=}$ is a natural number $n \in \mathbb{N} = \{0, 1, 2 \dots\}$

$\Rightarrow$ space of all Algorithms $= \mathbb{N}$

Decision problem $\hat{=}$ function that maps inputs to yes or no

eg. Is $s \in L = \{\dots, 0^N 1^N, \dots\}$

$f(0011) = \text{yes} = 1$
$f(1001) = \text{No} = 0$

input $\hat{=}$ binary string
$\hat{=}$ integer in $\mathbb{N}$

$\Big\}\ f : \mathbb{N} \to \{0,1\} = \{\text{yes/No}\}$

$\Rightarrow$ $\mathbb{N} \hat{=}$ set of algorithms, $D = \{ f : \mathbb{N} \to \{0,1\} \} \hat{=}$ set of decision-problems.

we show now that $|\mathbb{N}| < |D|$

To this end, let g be any map from $\mathbb{N}$ to D: $g : \mathbb{N} \to D$

so $g(n) = f$ for some f in D $\forall n \in \mathbb{N}$
& thus $(g(n))(n) = f(n)$ for some $n \in \mathbb{N}$.

Now define $h : \mathbb{N} \to \{0,1\}$

$$h(n) := \begin{cases} 0, \text{ if } (g(n))(n) = 1 \\ 1, \text{ if } (g(n))(n) = 0. \end{cases}$$

$(g(n))(n)) = f(n) \neq h(n) \quad \forall n \in \mathbb{N}$,

d.h. $g(n) \neq h \Rightarrow h$ is not in the image of g

$\Rightarrow g : \mathbb{N} \to D$ is not surjective

$\Rightarrow |\mathbb{N}| < |D|$

(in fact one can show that $|\mathbb{N}| << |\mathbb{D}| = |\mathbb{R}|$.)

Q: Does there exists for all problems that solve it (in polynomial time)?

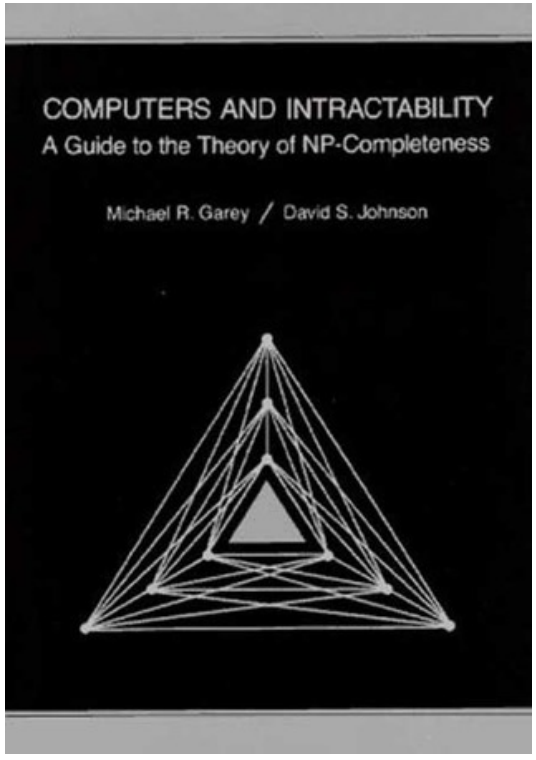
A: 1) $\exists$ problem that cannot be solved by algorithms.

2) $\exists$ problem that can be solved by algorithms but not in polynomial time (under reasonable assumptions

↓

Theory of NP-completeness

Literature



Computers and Intractability: A Guide to the Theory of NP-Completeness
by Michael R. Garey, David S. Johnson
Published January 15th 1979 by W. H. Freeman

In a nutshell: About this lecture

Aim: How difficult or easy is a problem that we want to solve?

Recap:

- There are problems that can be solved in polynomial time
  (EASY, tractable problems)

- There are problems where no algorithms exist to solve them
  in finite time (UNSOLVABLE)
  e.g. Halting-problem [determine whether a given algorithm terminates]

Questions:

- What about problems that can be solved in finite time, say there is an exponential-time algorithm with runtime $\mathcal{O}(1.5^n)$, but no polynomial-time algorithm has been found so-far?

  $\Rightarrow$ Should we search further for polynomial-time algorithms?

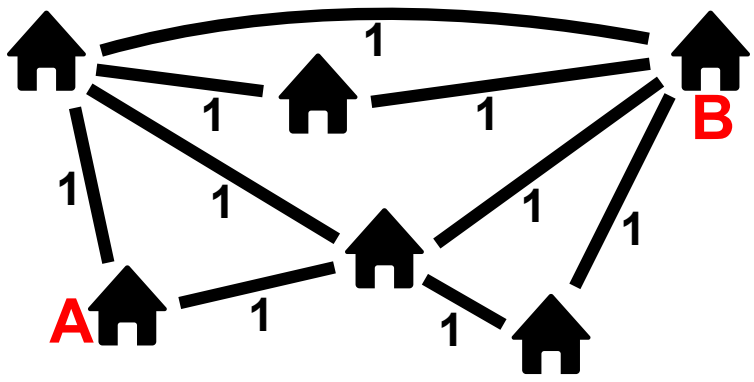
  Or could we stop searching, since there is some evidence that no polynomial-time algorithm may exist?

Easy vs. Difficult is not obvious!

Very small modifications of the problem formulation can make an “easy” problem to a “difficult” one.

Easy vs. Difficult is not obvious!

Very small modifications of the problem formulation can make an “easy” problem to a “difficult” one.



Find a shortest path between A and B under the assumption that all edges e have weight

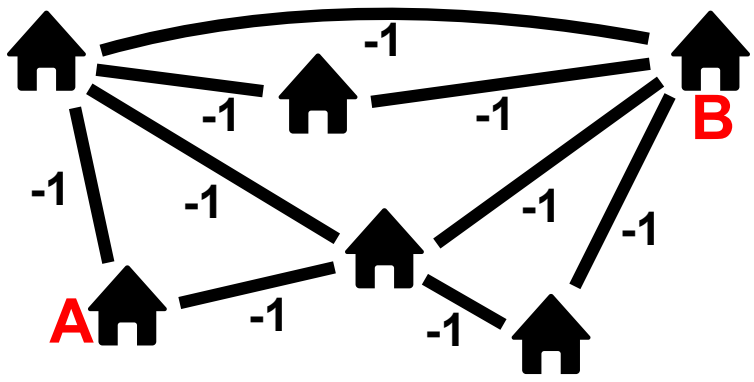
$$w(e) = 1$$

(solvable in polynomial time)

EASY!

Easy vs. Difficult is not obvious!

Very small modifications of the problem formulation can make an “easy” problem to a “difficult” one.



Find a shortest path between A and B under the assumption that all edges e have weight

$w(e) = \mathbf{1}$

(solvable in polynomial time)

EASY!

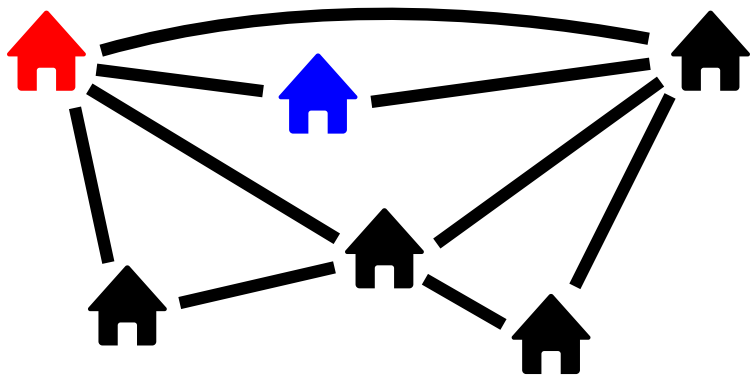
$w(e) = \mathbf{-1}$

Up to now, only exponential-time algorithms are known!

DIFFICULT? 🤔

Easy vs. Difficult is not obvious!

Very small modifications of the problem formulation can make an “easy” problem to a “difficult” one.



Find a coloring of the houses such that neighbors have different colors

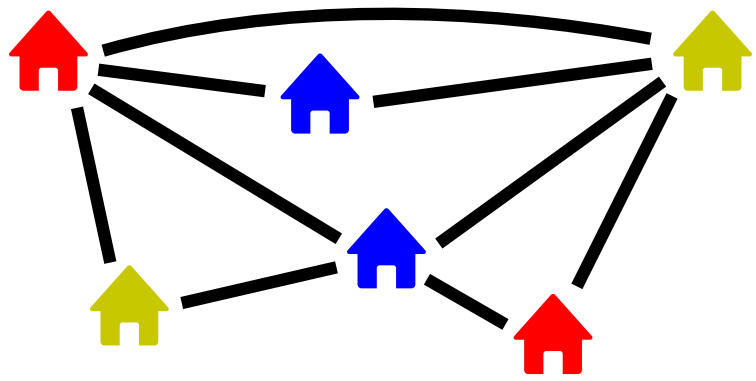
with **2 colors**

solvable in polynomial time

EASY!

Easy vs. Difficult is not obvious!

Very small modifications of the problem formulation can make an “easy” problem to a “difficult” one.



Find a coloring of the houses such that neighbors have different colors

with **2 colors**

solvable in polynomial time

EASY!

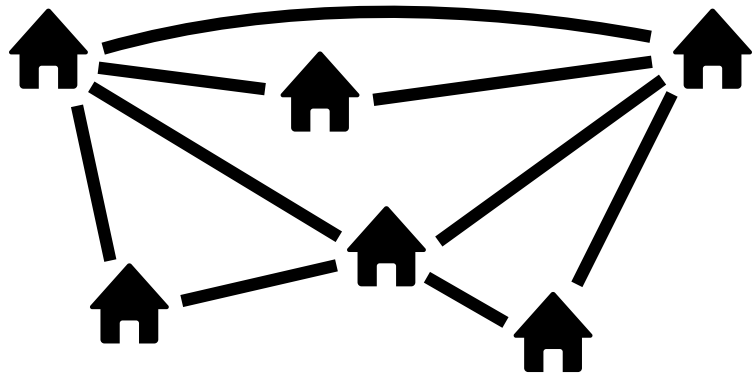
with **3 colors**

Up to now, only exponential-time algorithms are known!

DIFFICULT? 🤔

Easy vs. Difficult is not obvious!

Very small modifications of the problem formulation can make an “easy” problem to a “difficult” one.



It doesn’t seem to be a simple task to distinguish between **difficult** and **easy**!

(Q1) What does **difficult** formally mean?

(Q2) Are there problems for which no polynomial-time algorithm exists *under reasonable assumptions*? **(= intractable problems)**

What are these assumptions?

Easy vs. Difficult is not obvious!

- Main Ingredients
  - Optimization problems vs. decision problems
  - Classes P and NP
  - Reduction and NP-hardness
  - NP-completeness

We start with a brief overview of the main ingredients to answer these questions.

Optimization Problems vs. Decision Problems

- An **optimization problem** has as solution one that satisfies pre-described optimality constraints

  Shortest Path: Find between two houses A and B a path of min-length.

- A **decision problem** has as solution only an answer YES or NO

  Shortest Path: Does there exist a path between two houses A and B of length $\leq K$ for some $K \in \mathbb{N}$?

Memory Hook: Opt. Problem "easy" $\Rightarrow$ Dec. Problem "easy"
Dec. Problem "difficult" $\Rightarrow$ Opt. Problem "difficult"

To classify "difficulty", decisions problems are used:

- To prove that an optimization problem is difficult, it suffices to show that the corresponding decision problem is difficult.
- Decision problems have a very natural, formal counterpart called "language", as we have seen in the section about "TM and languages that are accepted by TM"

Outline

- Main Ingredients
  - Optimization problems vs. decision problems ✓
  - Classes P and NP
  - Reduction and NP-hardness
  - NP-completeness

P vs. NP

$\mathrm{P} = \{$dec. probl. that can be solved by **deterministic** alg. in polynomial time$\}$
$= \{$dec. probl. **solvable** in polynomial time$\}$

$\mathrm{NP} = \{$dec. probl. that can be solved by **non-deterministic** alg. in polynomial time$\}$
$= \{$dec. probl. **verifiable** in polynomial time$\}$

What is a non-deterministic algorithm ?

A non-deterministic algorithm consists of two separate stages:

guessing stage [*Guess a solution once*]

non-deterministic part: If there is a YES-solution, then one of these solutions is returned!

checking stage [*Verify if the solution is a YES-answer*]

computed in a normal deterministic manner.

NP means "**N**on-deterministic **P**olynomial" (**NOT:** *non-polynomial*!)

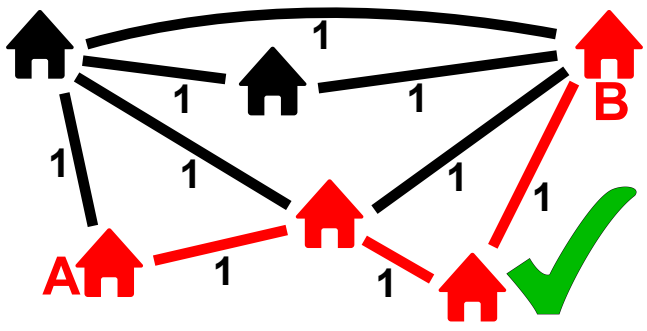
Keep in mind that algorithms are essentially defined in terms of TM.

P vs. NP

$\mathrm{P} = \{$dec. probl. that can be solved by **deterministic** alg. in polynomial time$\}$
$= \{$dec. probl. **solvable** in polynomial time$\}$

$\mathrm{NP} = \{$dec. probl. that can be solved by **non-deterministic** alg. in polynomial time$\}$
$= \{$dec. probl. **verifiable** in polynomial time$\}$

Example: Shortest Path $\in$ NP



Question: Does there exist a path of length ≤ 4 from A to B?

guessing stage [*Guess a solution once*]

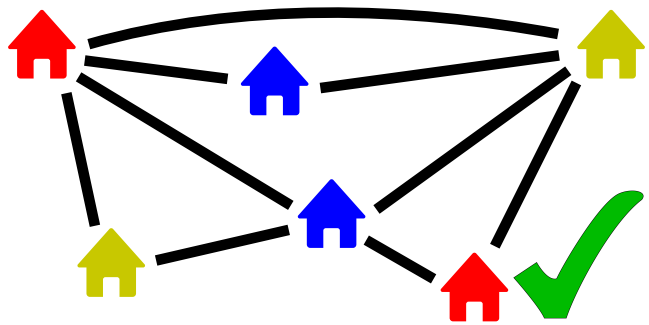
checking stage [*Verify if the solution is a YES-answer*]

P vs. NP

$$\begin{aligned} \mathrm{P} &= \{\text{dec. probl. that can be solved by } \textbf{deterministic} \text{ alg. in polynomial time}\} \\ &= \{\text{dec. probl. } \textbf{solvable} \text{ in polynomial time}\} \end{aligned}$$

$$\begin{aligned} \mathrm{NP} &= \{\text{dec. probl. that can be solved by } \textbf{non-deterministic} \text{ alg. in polynomial time}\} \\ &= \{\text{dec. probl. } \textbf{verifiable} \text{ in polynomial time}\} \end{aligned}$$

Example: 3-coloring $\in$ NP



Question: Does there exist a 3-coloring?

guessing stage [*Guess a solution once*]

checking stage [*Verify if the solution is a YES-answer*]

P vs. NP

$\mathrm{P} = \{$dec. probl. that can be solved by **deterministic** alg. in polynomial time$\}$
$= \{$dec. probl. **solvable** in polynomial time$\}$

$\mathrm{NP} = \{$dec. probl. that can be solved by **non-deterministic** alg. in polynomial time$\}$
$= \{$dec. probl. **verifiable** in polynomial time$\}$

Observation: $P \subseteq NP$

P-NP-Problem: $P \subset NP$ or $P = NP$?

If $P = NP$, then verifying a solution is as easy as finding a solution
(quite unlikely, but still unsolved!)

(Q2) Are there problems for which no polynomial-time algorithm exists *under reasonable assumptions*? **(= intractable problems)**

What are these assumptions? **Answer: $\mathbf{P \subset NP}$**

$\Rightarrow$ There are problems in $NP \setminus P \neq \emptyset$
These problems can be verified but not solved in polynomial time!

The formal theory of P and NP is defined in term of languages and Turing machines

Outline

- Main Ingredients
  - Optimization problems vs. decision problems ✓
  - Classes P and NP ($P \subsetneq NP$) ✓
  - Reduction and NP-hardness
  - NP-completeness

Reduction . . .

. . . is about transforming one problem into another problem

ARRR, we leave you at this lonely island if you cannot answer the following question correctly within 2 minutes:

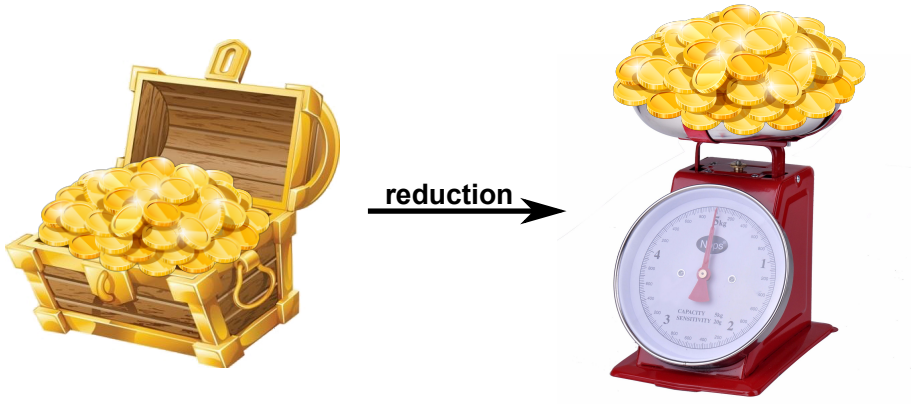
Are there less then 12222 coins in the chest?

YOU: 1 coin = 8g
all coins = 97760 g
= **12220** * 8g

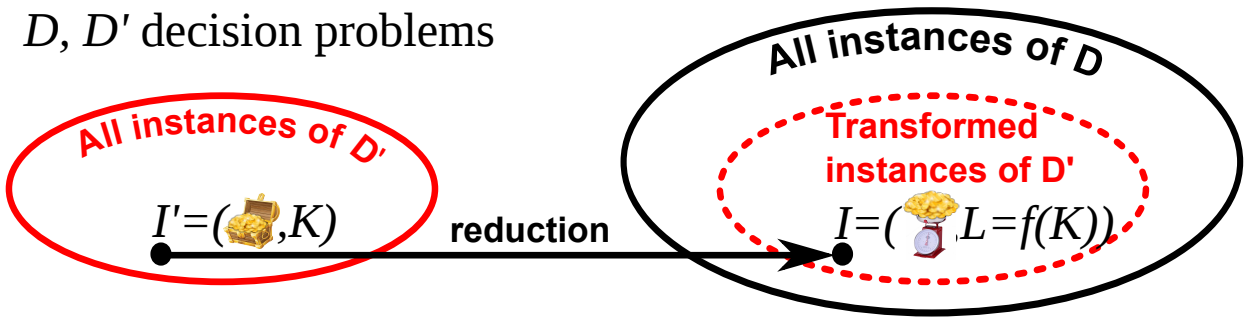
Answer: YESSSS!!

Reduction . . .

. . . is about transforming one problem into another problem



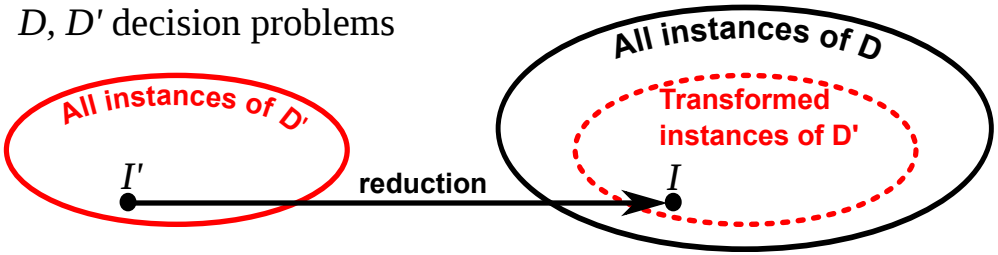
We reduced a counting problem to a weighting problem!



D' (number coins)
IN: Set of coins & integer K
Q: Number of coins $\leq K$?

D (weight of items)
IN: Item i & integer L
Q: Weight(i)$\leq L$?

Reduction and NP-hard



$\mathrm{P} = \{$dec. probl. *solvable* in polynomial time$\}$

$\mathrm{NP} = \{$dec. probl. *verifiable* in polynomial time$\}$

instance = *specified input*.

A **reduction** from D' to D is a procedure that transforms every instance I' of D' to an instance I of D such that

- the transformation can be done in *polynomial time* (= "easy") and
- I has YES-answer if and only if I' has YES-answer.

$\Rightarrow$ Every algorithm that solves D can be used to solve D'.

A dec. problem D is **NP-hard** if *every* problem $D' \in \mathrm{NP}$ can be reduced to D.

If D is NP-hard, then

- every problem $D' \in \mathrm{NP}$ can be considered as a "special case" of D.
- D is at least as difficult to solve as any other problem in NP.
- D is at least as difficult to solve as those problems in NP for which no polynomial-time algorithm exists ($P \subset NP$)
- it is reasonable to assume that for D there are no polynomial-time algorithm.

(Q1) What does **difficult** formally mean? **Answer:** NP-hard

Outline

- Main Ingredients
  - Optimization problems vs. decision problems ✓
  - Classes P and NP ($P \subsetneq NP$)✓
  - Reduction and NP-hardness ✓
  - NP-completeness

NP-complete

(Q1) What does **difficult** formally mean? **Answer:** NP-hard

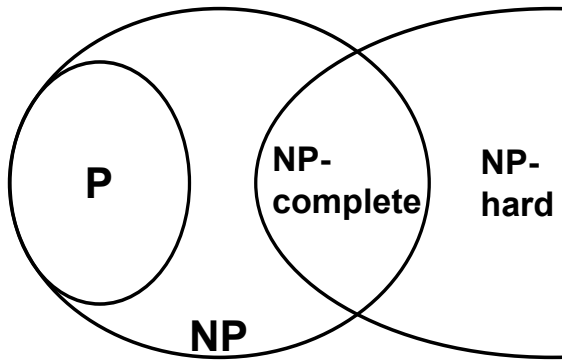
(Q2) Are there problems for which no polynomial-time algorithm exists *under reasonable assumptions*? **(= intractable problems)**

Answer: Yes, NP-complete problems

What are these assumptions? **Answer:** $\mathbf{P \subset NP}$

A decision problem D is **NP-complete** if

- $D \in \mathrm{NP}$
- D is NP-hard: *every* problem in NP can be reduced to D.



NP-complete problems are the most difficult problems in NP.

NP-complete

Note, Languages and Decision problem can be seen as being “equivalent”, i.e., in the following definition, we can use these terms interchangeably.

A decision problem D is **NP-complete** if

- $D \in \mathrm{NP}$
- D is NP-hard: *every* problem in NP can be reduced to D.
  In symbols,
  $$\forall D' \in \mathrm{NP} : D' \preceq_p D$$

Q: How to show that EVERY problem in NP can be reduced to D?

A: There was a first problem “SAT” that was shown to be NP-complete

$\Rightarrow$ Every problem $D' \in NP$ can be reduced to SAT.
$\Rightarrow$ If we can show for some problem D that SAT can be reduced to D then every problem $D' \in NP$ can be reduced to D.
($\preceq_p$ is transitive)

$$\forall D' \in NP : D' \preceq_p \mathrm{SAT} \text{ and } \mathrm{SAT} \preceq_p D \implies \forall D' \in NP : D' \preceq_p D$$

Outline

- Main Ingredients
  - Optimization problems vs. decision problems ✓
  - Classes P and NP ($P \subsetneq NP$)✓
  - Reduction and NP-hardness ✓
  - NP-completeness ✓

The SAT-problem:

given: Boolean expression φ consisting of:

- AND = $\wedge$, OR = $\vee$, NOT = $\neg$
- literal = negation of Boolean variable or Bool. var. itself.
- bracket "(", ")"

and φ is conjunctive normal form (CNF)

= conjunction (AND) of clauses,
clauses are disjunction of literals. (OR)

eg. $\varphi = (x_1 \vee \neg x_2) \wedge (x_1 \vee \neg x_1 \vee x_4 \vee \neg x_5)$

Q: Is there an assignment of value 0/1 to Boolean variables s.t. the expression φ becomes 1 (true)?
["φ is satisfiable"]

Cook-Levin-Thm (1971)

The decisionproblem SAT is NP-complete

[proof: not difficult, but rather lengthy
↗ Book of Garey & Johnson]

We want to use this Thm, to show that other decision problems D are NP-complete:

(1) show $D \in NP$: "show a guessed solution can be verified as yes/no in poly-time"

(2) show D is NP hard: "take known NP-complete problem D' & provide poly-time reduction $D' \leq_p D$

- transform any instance $I' \in D'$ to instance $I \in D$
- show transform. goes in poly-time
- show I' yes $\Leftrightarrow$ I yes

3SAT: is SAT instance s.t. clause consists of exactly 3 literals

$\varphi = C_1 \wedge C_2 \wedge C_3 \wedge \dots \wedge C_n$, $|C_i| = 3$ $1 \le i \le n$.

Q: $\exists$ truth-assignment to variables s.t. φ = true?
[φ is satisfiable]

Thm: 3SAT is NP-complete

proof: 3SAT $\in$ NP: guess solution = assignment of 0/1 to Boolean variables

check if each clause is "true".

works in poly-time ✓

3SAT is NP hard: (so-far we only know that SAT is NP-C so let's try this)

show SAT $\le_p$ 3SAT

Let φ be an arbitrary instance of SAT.

Let nr of clauses of φ be $\underline{m}$: $\varphi = C_1 \wedge C_2 \dots \wedge \dots \wedge C_m$

IF all clauses C_i have size $|C_i| = 3$, then there is nothing to transform.

but there are instances of SAT where $|C_i| \neq 3$

We consider the following cases for each C_i: $|C_i| = 2$
$|C_i| = 1$
$|C_i| > 3$

To show that φ is true
it suffices to show that each clause is true!

$|C_i| = 2$: introduce new variable u, remove $C_i = (x \vee y)$

eg. $C_i = (x \vee y)$

& 2 new clauses $C_i' = (x \vee y \vee u)$

$C_i'' = (x \vee y \vee \bar{u})$ ↑ not(u)

to show: C_i true $\Leftrightarrow$ $C_i' \wedge C_i''$ true.

IF C_i true $\Rightarrow$ at least one of x & y must be true

$\Rightarrow$ C_i' & C_i'' true

IF $C_i' \wedge C_i''$ true $\Rightarrow$ C_i' true & C_i'' true

IF $u = 1 \Rightarrow \bar{u} = 0 \Rightarrow$ in C_i'': $x = 1$ or $y = 1$

$\Rightarrow$ C_i true

IF $u = 0 \Rightarrow$ in C_i': $x = 1$ or $y = 1 \Rightarrow$ C_i true. ✓

$|C_i| = 1$: introduce new variable u,

remove $C_i = (x)$

add $C_i' = (x \vee u)$, $C_i'' = (x \vee \bar{u})$

eg. $C_i = (x)$

[Ex.: C_i satisf. $\Leftrightarrow$ $C_i' \wedge C_i''$ satisfiable]

add new variable v

& replace C_i' & C_i'' by

$C_i^{1'} = (x \vee u \vee v)$

$C_i^{2'} = (x \vee u \vee \bar{v})$

$C_i^{1''} = (x \vee \bar{u} \vee v)$

$C_i^{2''} = (x \vee \bar{u} \vee \bar{v})$

} in this way the clauses are linked and one of uv, $u\bar{v}$, $\bar{u}v$, $\bar{u}\bar{v}$ must false $\Rightarrow$ x must be true!

[Ex. C_i satisf $\Leftrightarrow$ $C_i^{1'} \wedge C_i^{2'} \wedge C_i^{1''} \wedge C_i^{2''}$ satisf.]

$|C_i| = 3$ nothing to do.

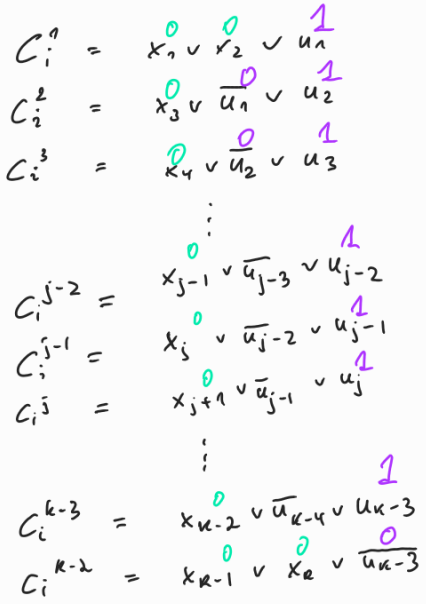
$|C_i| > 3$, let $C_i = (x_1 \vee \dots \vee x_k)$, where x_i are literals

add new variables $u_1, u_2 \dots u_{k-3}$

remove C_i & add new clauses:

$$C_i^1 = x_1 \vee x_2 \vee u_1$$
$$C_i^2 = x_3 \vee \bar{u}_1 \vee u_2$$
$$C_i^3 = x_4 \vee \bar{u}_2 \vee u_3$$
$$\vdots$$
$$C_i^{j-2} = x_{j-1} \vee \bar{u}_{j-3} \vee u_{j-2}$$
$$C_i^{j-1} = x_j \vee \bar{u}_{j-2} \vee u_{j-1}$$
$$C_i^{j} = x_{j+1} \vee \bar{u}_{j-1} \vee u_j$$
$$\vdots$$
$$C_i^{k-3} = x_{k-2} \vee \bar{u}_{k-4} \vee u_{k-3}$$
$$C_i^{k-2} = x_{k-1} \vee x_k \vee \bar{u}_{k-3}$$

✓ main idea: link the literals $x_1 \dots x_k$ in clauses s.t. C_i true $\Leftrightarrow C_i^1 \dots C_i^{k-2}$ true



to show: C_i satisfiable $\Leftrightarrow$ $C_i^1 \dots C_i^{k-2}$ satisf.

„$\Rightarrow$" C_i satisf. $\Rightarrow$ at least one $x_j = 1$

Set $u_r = 1$, $r \le j-2$
$u_r = 0$, else

$\Rightarrow$ all $C_i^1 \dots C_i^{k-2}$ are true.

„$\Leftarrow$" assume $C_i^1 \dots C_i^{k-2}$ true

Proof by contradiction,
ie, assume C_i is not satisf.

$\Rightarrow x_1 = x_2 = \dots = x_k = 0$

$\Rightarrow$ in C_i^1: $u_1 = 1$

$\Rightarrow C_i^2 : x_3 = \bar{u}_1 = 0 \Rightarrow u_2 = 1$

$\Rightarrow u_i = 0 \quad \forall i$

$\Rightarrow C_i^{k-2} : x_{k-1} = x_k = 0$
& $\bar{u}_{k-3} = 0$

$\Rightarrow C_i^{k-2}$ not true ↯

$\Rightarrow$ C_i satisf $\Leftrightarrow$ $C_i^1 \dots C_i^{k-2}$ satisf.

[Ex: transformation can be done in poly-time]

$\Rightarrow$ $SAT \le_p 3SAT$ / □

CLIQUE problem

Input: undirected graph $G=(V,E)$, integer $L \in \{1,\dots,|V|\}$

Question: exists $H \subseteq G$ s.t. $H \cong K_r$, $r \geq L$?

Theorem: CLIQUE is NP-complete

proof: CLIQUE $\in$ NP [Exercise]

NP hardness show by reduction from 3-SAT.

let φ be an instance of 3SAT,

$\varphi = C_1 \wedge C_2 \wedge \dots \wedge C_k$, $C_i = \{ l_1^i \vee l_2^i \vee l_3^i \}$

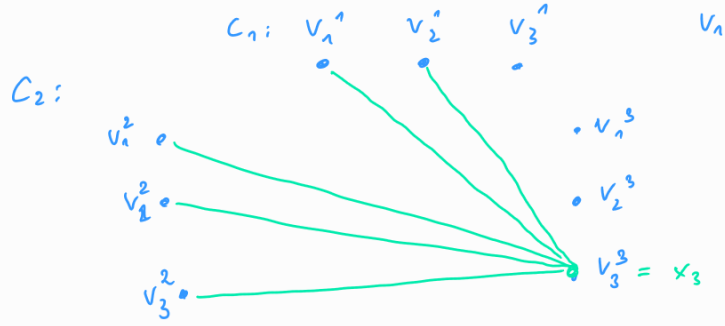
Need an instance of CLIQUE, i.e., a graph G.

Construct $G_\varphi = (V, E)$ as follows:

1) f.a. C_i add 3 vertices v_1^i, v_2^i, v_3^i
and "associate" v_r^i with literal l_r^i

2) add edge $v_r^i v_s^j$ if (i) $i \neq j$
(ii) corresp. literals l_r^i & l_s^j are not negations of from each other, i.e.
$l_r^i = x$, $l_s^j = \bar{x}$
resp $l_r^i = \bar{x}$, $l_s^j = x$ not allowed.

Exmpl $\varphi = (x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$
(C_1, C_2, C_3)



$v_1^1 \hat{=} x_1$, $v_2^1 = \bar{x}_2$, $v_3^1 \hat{=} \bar{x}_3$
(analog for all other)
only edges from V_3^3 are shown.

to show: φ satisfiable $\Leftrightarrow$ G_φ has clique of size k

„$\Rightarrow$" if φ is satisfiable $\Rightarrow$ each C_i has at least one true literal l_r^i

$l_r^i \hat{=} v_r^i$ in G_φ

take from each C_i one true literal and save corresponding vertices in V'

$\Rightarrow V' \subseteq V, \quad |V'| = k$

let $v_r^i, v_s^j \in V' \xrightarrow[\text{by construction}]{(i)} i \neq j$

& corresp. literal l_r^i, l_s^j are not negations of each other, since they are both true.

$\xrightarrow[\text{by construction}]{(ii)}$ in G_φ exists edge (v_i^r, v_j^s)

$\Rightarrow$ $G_\varphi[V']$ is complete subgraph with k vertices ✓

„$\Leftarrow$" Assume $\exists\, V' \subseteq V, |V'| = k$ s.t. $\forall u, w \in V' : (u,w) \in E$.

Since G_φ does not contain edge between vertices corresponding to the same clause

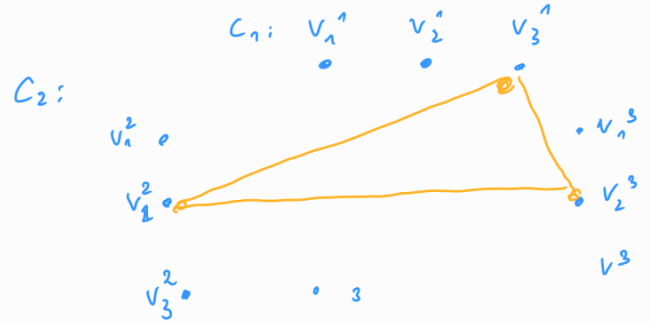
$\Rightarrow$ V' contains exactly 1 vertex of each clause

$\Rightarrow$ for each $v_r^i \in V'$ set literal $l_r^i = 1$

$\Rightarrow$ each C_i true $\Rightarrow$ φ satisfiable.

Exmpl $\varphi = (x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$

(C_1 ↘ 1, C_2 ↘ 1, C_3 ↘ 1)



$v_1^1 \hat{=} x_1$, $v_2^1 = \bar{x}_2$, $v_3^1 \hat{=} \bar{x}_3$

(analog for all other)

eg. $k = 3$

Reduction can be done in polyn. time [Exerc]

$\Rightarrow$ 3SAT $\leq_p$ CLIQUE

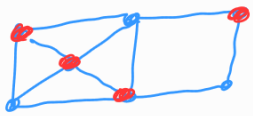
□

Vertex-Cover Problem (VCP)

input: undirected graph $G = (V, E)$, integer L

question: Does G have a vertex cover of size $k \leq L$, i.e.,
is there a subset $C \subseteq V$
s.t.: (i) $|C| = k \leq L$
(ii) $C \cap e \neq \emptyset \quad \forall e \in E$

Exmpl



VC of size 4

Theorem: VCP is NP-complete

proof: VCP $\in$ NP [Exercise]

VCP NP-hard, by reduction from CLIQUE

we show: $G=(V,E)$ has CLIQUE of size k
$\Leftrightarrow$ $\bar{G}$ has VC of size $|V|-k$

Let $V' \subseteq V$ s.t. $G[V']$ forms a clique of size k
& put $C = V \setminus V'$
Assume, for contradiction, C is not VC of $\bar{G}$.
$\Rightarrow$ $\exists$ edge $(u,v) \in E(\bar{G})$ s.t. $u, v \notin C = V \setminus V'$
$\Rightarrow$ $u, v \in V'$
but since $(u,v) \in E(\bar{G}) \Rightarrow (u,v) \notin E(G)$ $\lightning$

Let $C = V \setminus V'$ be a VC of $\bar{G}$
$\Rightarrow$ $\forall (u,v) \in E(\bar{G})$: $u \in V \setminus V'$ or $v \in V \setminus V'$
$\Rightarrow$ at least one of the vertices u & v is not in V'
&
if $x, y \in V' \Rightarrow x, y \notin V \setminus V'$ $\xrightarrow[\text{VC of } \bar{G}]{C = V \setminus V'}$ $xy \notin E(\bar{G})$
$\Rightarrow$ $\forall x, y \in V'$: $xy \in E(G) \Rightarrow G[V']$ is complete graph on $|V|-k$ vertices

reduction works in polytime [Exns]

$\Rightarrow$ CLIQUE $\leq_p$ VCP

$\square$

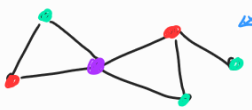
Vertex Coloring

A vertex-coloring of a (di)graph $G=(V,E)$ is a map $\chi: V \to C$ s.t. for all $(uv) \in E$ we have $\chi(u) \neq \chi(v)$.

(C: set of colors)

χ is called $|C|$-coloring

Trivially, $\chi: V \to V$, $x \mapsto x$ $\forall x \in V$ is a vertex-coloring.



But here instead of $|V|=6$, 3 colors are sufficient.

The smallest k, s.t. there is a vertex-coloring for a given graph $G=(V,E)$ is called chromatic number $\chi(G)$.

Min_Vertex_Coloring (MVC)

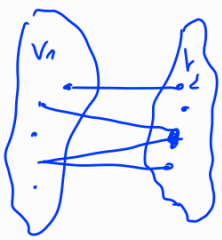
Input: A graph $G=(V,E)$ & integer $k \geq 3$

Question: Is $\chi(G) \leq k$?

$k=1$: $E=\emptyset$

$k=2$: $G=(V,E)$ is bipartite, i.e., $\exists$ partition V_1, V_2 of V ($V_1 \cup V_2 = V$, $V_1 \cap V_2 = \emptyset$)

s.t. $\forall (xy) \in E$: $x \in V_i$, $y \in V_j$, $i \neq j$



Lemma: G bipartite $\Leftrightarrow$ G does not contain simple cycles of odd length

proof: „$\Rightarrow$" Assume, for contradiction, that G contains odd-length simple cycle

$k = 2l+1$ $\Rightarrow$ if we use 2 colors $\Rightarrow$ $V_1 \neq V_2$

$V_3 \neq V_4$

$\vdots$

$V_1 = V_k$ ↯

$\Leftarrow$: Assume all simple cycles in G are of even length

Let C be some connected component

& v_0 be a vertex in C & color it red

Let $d(v_0, x)$ denote the number of edges on a shortest simple path connecting v_0 & x

Exmpl: 

$d(v_0, x) = 2$.

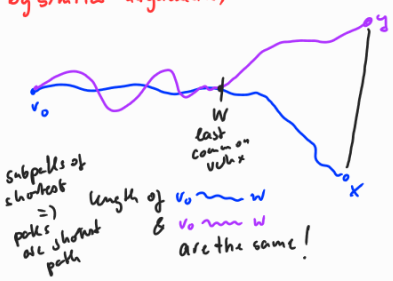
$\forall x \in V$: $d(v_0, x)$ is either even or odd.

Put $f(x) = \begin{cases} \text{red}, & \text{if } d(v_0,x) = \text{even} \\ \text{purple}, & \text{if } d(v_0,x) = \text{odd}. \end{cases}$

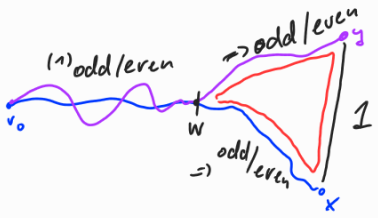
Assume, for contradiction, that there are vertices $x, y \in V$ s.t. $\{x,y\} \in E$ with $f(x) = f(y)$

$\Rightarrow$ $d(v_0, x)$, $d(v_0, y)$ are both odd or even

even (odd by similar argument)



$\Rightarrow$



C is simple cycle of odd-length $\lightning$

$\square$

The latter proof directly leads to a polynomial-time alg. to test & find a 2-coloring of a given graph G.

[Exercise]

Question: what if $k \geq 3$?

$=$

Theorem: MVC with input G & $k = 3$ is NP-complete.

Proof: MVC $\in$ NP (Exc.)

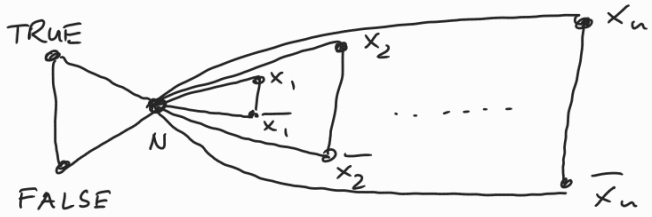
NP-hardness: 3SAT $\leq_p$ MVC

Let $\varphi = C_1 \wedge C_2 \wedge \dots \wedge C_m$ a 3SAT instance

$|C_i| = 3$

with bool. variables $x_1 \dots x_n$

1. construct gadget for variables:

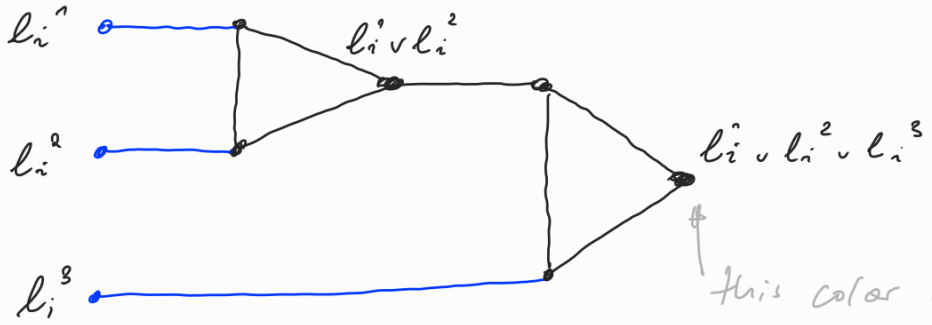


← "easy to see" its 3-colorable

Now we need to connect the variables to their clauses

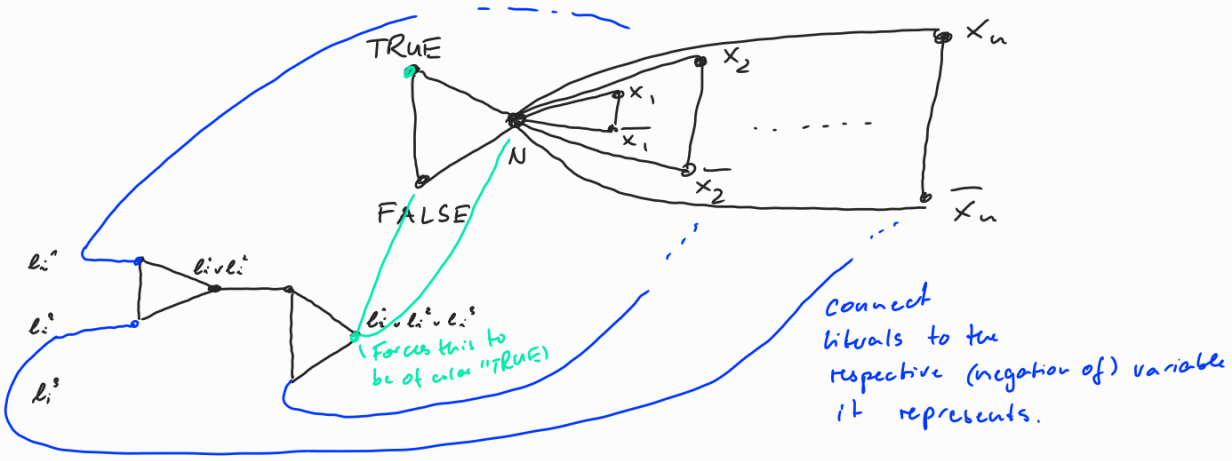
2. gadgets for clauses:

For all $C_i = (l_i^1 \vee l_i^2 \vee l_i^3)$ where l_i^j is a bool. variable or its negation.



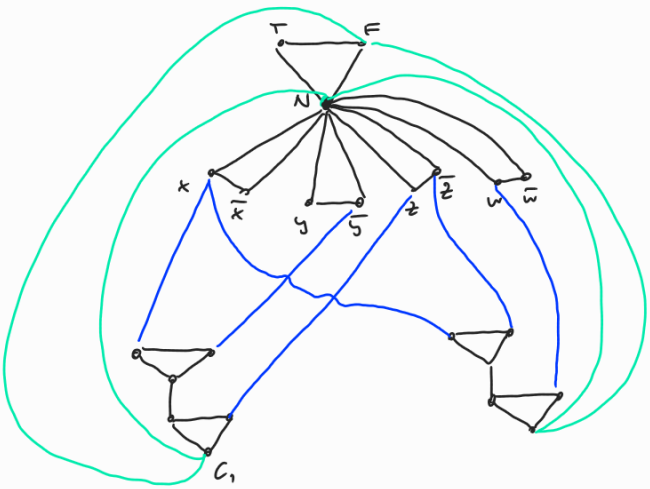
this color should be true, if at least one of the l_i^j is true.

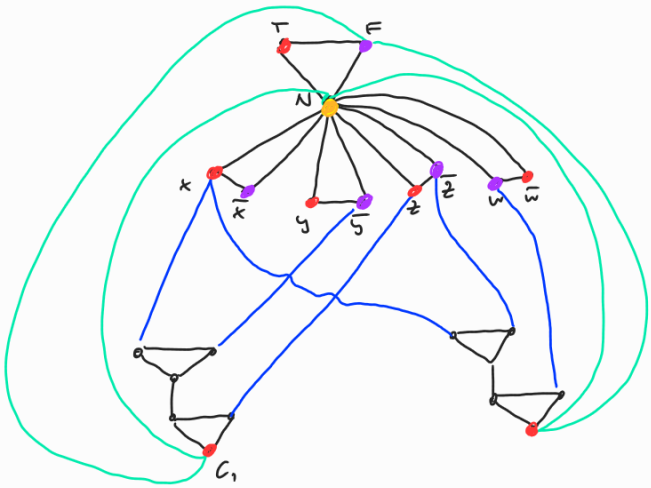
3. connect all gadgets



Resulting graph is called G_φ. [can be constructed in poly-time]

Exampl: $(x \vee \bar{y} \vee z) \wedge (x \vee w \vee \bar{z})$ — with $x = 1$, $\bar{y} = 0$, $z = 1$, $w = 0$, $\bar{z} = 0$





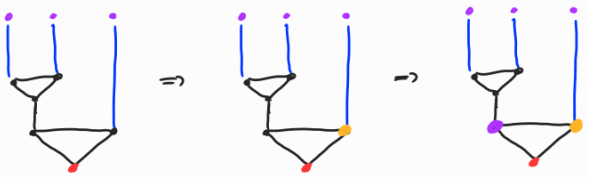
to show: 6q 3colorable $\Leftrightarrow$ φ satisfiable

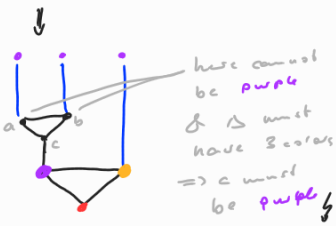
Observation:

if C_i = true, i.e.  if at least one lit. red (= true)

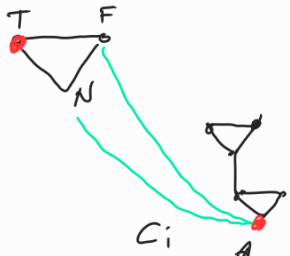
this vertex red (1) $\hat{=}$ true

possible?





If 6q 3colorable $\Rightarrow$



forced to be True = red.

by A) at least one literal is true

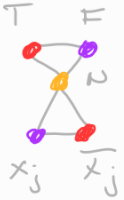
$\Rightarrow$ C_i true $\forall i$.

$\Rightarrow$ φ satisfiable.

If φ satisfiable: $x_j = \begin{cases} 1 & \text{if } \bar{x}_j = 0 \\ 0 & \text{if } \bar{x}_j = 1 \end{cases}$ $\longrightarrow$ color

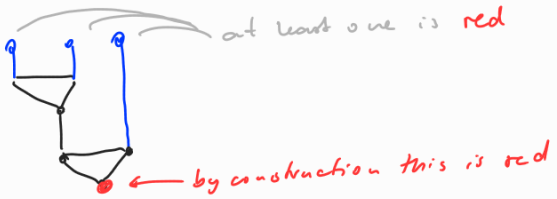
respect. & use color "true" for T

eg:



Since φ satisfiable $\Rightarrow$ at least one literal is true

$\Rightarrow$ C_i



$\Rightarrow$

to show this C_i is 3 colorable

[Exercise]

Theorem: MVC with input G & $K \geq 3$ is NP-complete.

proof Exercise

SIDENOTE: There are problem that are NP-hard but not in NP, e.g. HALTING problem.

We state now some further NP-complete problems, but without proof.

HAMILTONIAN PATH/CYCLE

Input: (di)graph G

Q: $\exists$ a path/cycle in G that visits every vertex precisely ones [called Hamiltonian path] ?

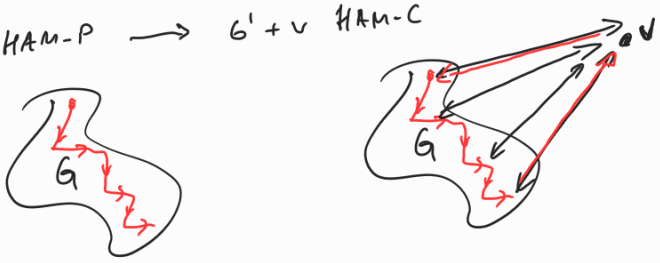
theorem: HAMILTONIAN PATH for di-graphs is NP complete

proof: use reduction from 3-SAT.

theorem: HAMILTONIAN CYCLE for di-graph is NP-complete

proof: use reduction from HAMILTONIAN PATH

HAM-P $\longrightarrow$ $G' + v$ HAM-C



theorem: HAMILTONIAN PATH for graphs is NP complete

proof: use reduction from HAM. PATH for di-graphs

IDEA: replace all v in digraph by

$\forall$  $\cong$ 

theorem: HAMILTONIAN CYCLE for graph is NP-complete

proof: use reduction from HAMILTONIAN PATH for graphs analog as for directed case

Traveling-Salesperson Problem (TSP)

Input: complete graph $G=(V,E)$ with edge weight $w: E \to \mathbb{N}_0$ & integer k

Q: Is there a cycle C in G that visits each vertex precisely once & $\sum_{e \in C} w(e) \leq k$?

Theorem: TSP is NP-complete

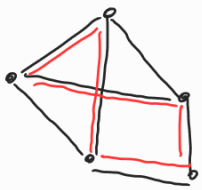
proof idea: reduction from HAM Cycle for graphs.

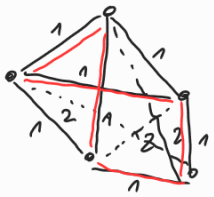
instance G of HAM-C

Def. G' = complete graph with

$w: E \to \{1,2\}$

s.t. $w(e) = \begin{cases} 1, & e \in E(G) \\ 2, & \text{else} \end{cases}$

$G =$ 

$G' =$ 

G has HAM-C $\iff$ G' has cycle of weight $k = |V(G')|$