ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

M.Sc. Program in Computer Science
Department of Informatics

Design and Analysis of Algorithms

The classes P and NP

Vangelis Markakis

markakis@gmail.com

Complexity Class P

P =

All problems Π, such that for every instance I∈Π, there exists an algorithm with worst case complexity O(p(|I|)), for some polynomial p (i.e., O(poly(|I|)))

Known Problems in P: finding min/max: O(n)
sorting: O(n log n)
Integer multiplication: $O(n^{1.59})$
GCD(a,b), a>b: O(log a)
Primality testing: $O(\log^{12} n)$ [August 2002]
… many others …

Problems without a known polynomial time algorithm:
SUBSET SUM : O(nB)
SAT: $O(2^n)$
… many others too…

Decision and Optimization Problems

Decision problems: problems where the answer is YES or NO

e.g. Primality testing, SUBSET SUM, SAT,...

Optimization problems: maximize/minimize some objective function

TSP (Traveling Salesman Problem)

I: A complete weighted digraph $G = (V, E)$

Q: Find a minimum weight tour of G

(tour: a cycle visiting each node exactly once)

CLIQUE

I: A graph $G = (V,E)$

Q: Find the maximum subset $C \subseteq V$ s. t. $\forall\ u,v \in C: (u,v) \in E$

(the maximum complete subgraph of G)

Decision and Optimization Problems

An optimization problem has three versions/questions:

Function version: find an optimal feasible solution

Evaluation version: find the cost of an optimal feasible solution

Decision version: Given a bound B, is there a feasible solution of cost $\leq$ B (for minimization problems) or of cost $\geq$ B (for maximization problems)

TSP

Q1: Find a tour of minimum cost

Q2: Find the actual cost of a minimum tour

Q3: Is there a tour of cost $\leq$ B ?

CLIQUE

Q1: Find the vertices of a vertex cover C

Q2: Find the size of C

Q3: Is there $C \subseteq V$ such that $\forall\, u,v \in C$: $(u,v) \in E$ and $|C| \geq B$?

For any optimization problem we can state its corresponding decision version

Decision and Optimization Problems

In the sequel, we will mostly consider decision problems or the decision version of optimization problems

- Complexity theory is mostly built around decision problems
- they are used to define complexity classes
- the decision version of an optimization problem is equivalent to its function and evaluation versions!

Decision and Optimization Problems

For all the problems that we have seen and will see:
Given an algorithm for the decision version of an optimization problem, there exists a polynomial time algorithm to answer both its evaluation and function versions

Example: TSP

(1) decision → evaluation

Apply the question "is there a tour of cost ≤ B" for several values of B

For what values of B ? (not for all)

Optimal value bounded by the sum of the first n weights

Apply binary search in this range

How many calls needed to the decision version?

O(log(sum of n largest weights)) = O(poly(|I|))

Hence a polynomial "reduction" from decision to evaluation

Decision and Optimization Problems

Example: TSP (cont.)

(2) decision → function

Use (1) to find the cost of an optimal solution, say C*

```
T:= { }   //T will store an optimal tour
For each edge e ∈ E do
    {  x :=w(e);
       w(e) := w(e) + M; //M is some positive number
       "is there a tour of cost ≤ C* ? "
       if NO  then
            {  T=T U {e} ; //e  is contained in an optimal tour
               w(e):=x    } //restore the weight of e
       // if YES then e is not in any optimal tour
       // keep its weight to w(e) + M
```

Complexity $O(|E|) \sim O(n^2)$

The Complexity Class NP

For a decision problem Π, an instance $I \in \Pi$ is a

- yes-instance, if there exists a solution to the question posed by I
- no-instance, otherwise

The class NP - high level definition:

- A problem Π is in NP if we can verify efficiently the validity of a candidate solution
- i.e., we can convince somebody for a yes-instance in poly-time

In TSP: a candidate solution (a certificate) = one of the possible tours

Definition: A problem $\Pi \in$ NP iff for every yes-instance $I \in \Pi$,

there is a polynomial time verification algorithm A,

and a certificate x with $|x| \leq \text{poly}(|I|)$, such that A(I, x) = yes

In complexity theory terms: NP = all problems for which there is a non-deterministic polynomial time algorithm

The Complexity Class NP

Verification of yes-instances is not the same as distinguishing between yes/no instances

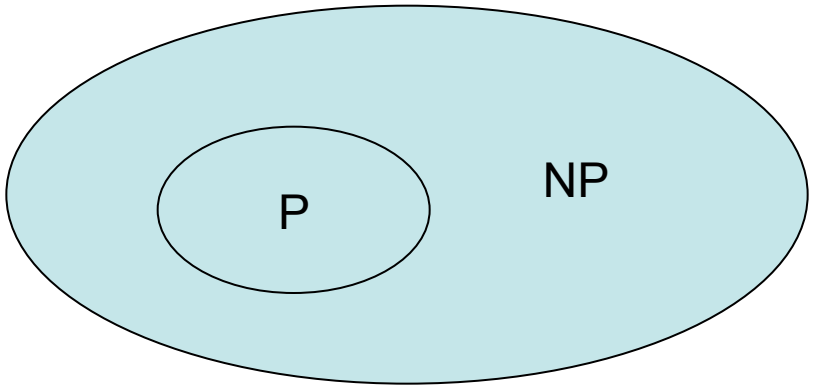
Example: TSP

- yes-instance: certificate x = a collection of edges
  (expected to be a tour)
  Verification algorithm:
  check if it is a tour;
  evaluate its cost;
  check if its cost ≤ B ; O(n) time in total
- no-instance: the only way to convince someone for a no-instance of TSP would be to check all tours !

P versus NP

If for a problem Π, we have a polynomial time algorithm that solves it, then, we can obviously validate a yes-instance in polynomial time

Hence: **P** $\subseteq$ **NP**



What about the reverse direction? Million dollar question!!
http://www.claymath.org/millennium-problems

Polynomial time (Karp) reductions

- Let A, B: decision problems

- We say that A polynomially reduces to B ($A \leq_p B$) if

  there is a polynomial time transformation R such that:

  - for every instance $I \in A$, it produces an instance $R(I) \in B$
  - I is a yes-instance for A iff R(I) is a yes-instance of B

- i.e. the answer to B with input R(I) is YES iff the answer to A with input x is YES

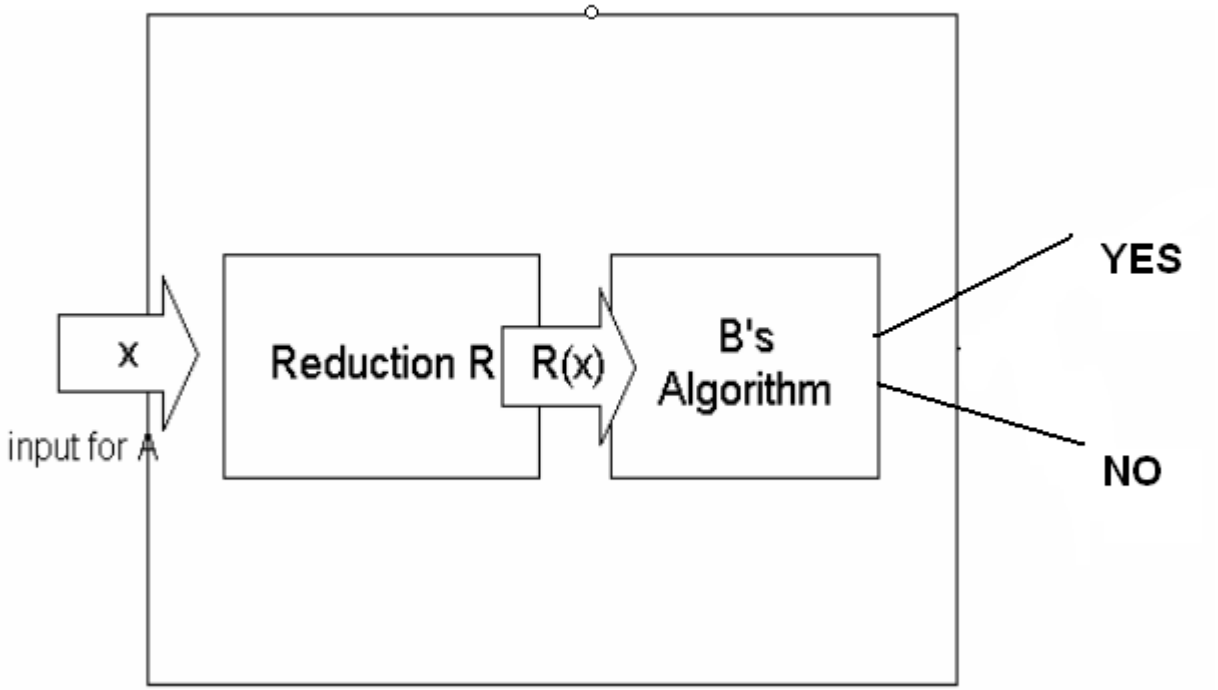
In other words: to solve A on input I we just have to:

$\Rightarrow$ compute R(I)

$\Rightarrow$ solve B on R(I)

O(poly) for R(x) + O(poly) algorithm for B ➔ O(poly) algorithm for A

Polynomial time (Karp) reductions



Possible algorithm for A

A polynomial time reduction from A to B shows that A is not more difficult than B

The class of NP-complete problems

A problem $\Pi \in NP$ is NP-complete iff

for every other problem $\Pi' \in NP$: $\Pi' \leq_p \Pi$

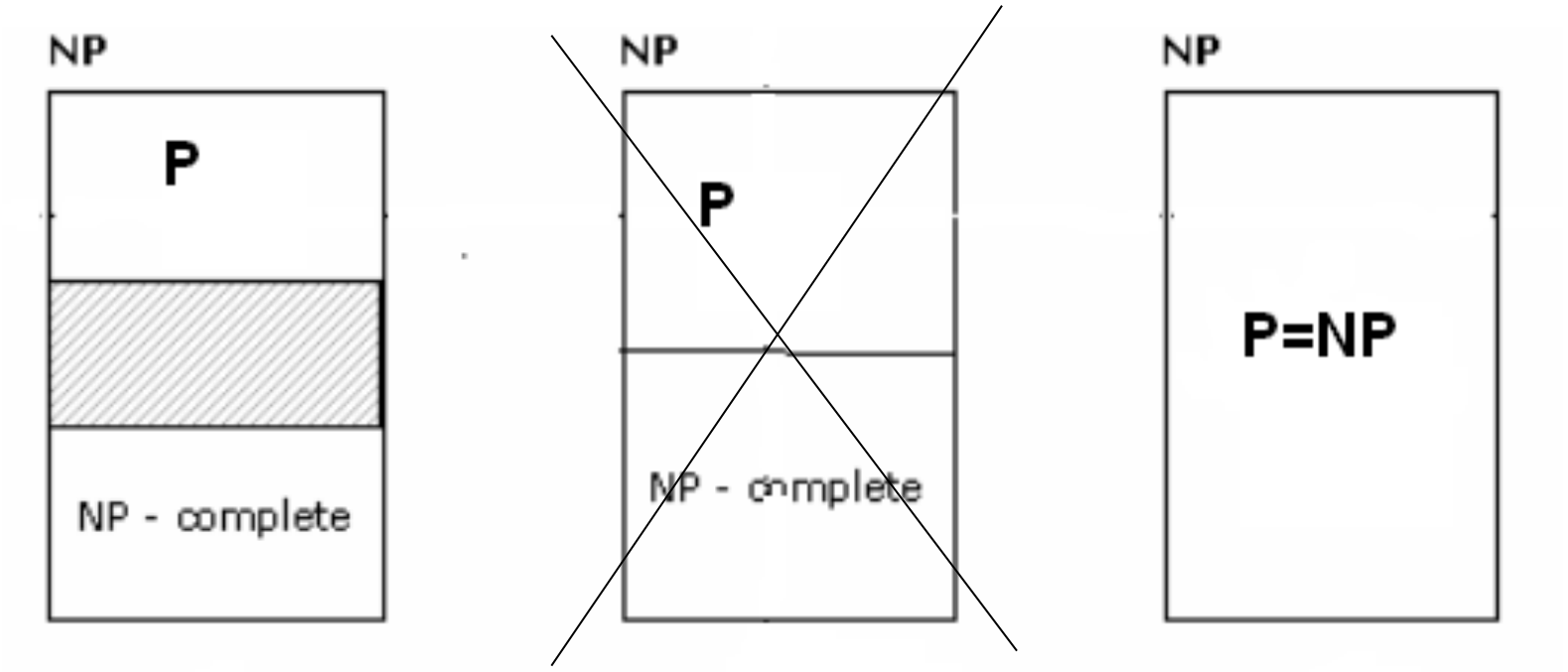
NP-completeness:

- captures the essence and the difficulty of NP
- identifies the most difficult problems in NP

To prove that a problem Π is NP-complete:

1. prove that $\Pi \in NP$
2. prove that $\forall\, \Pi' \in NP$: $\Pi' \leq_p \Pi$

The class of NP-complete problems



the common belief

Not possible

Ladner's theorem

Nobody believes this

No poly-time algorithm is known for any NP-complete problem.
If you ever derive one you will have proven that P = NP !

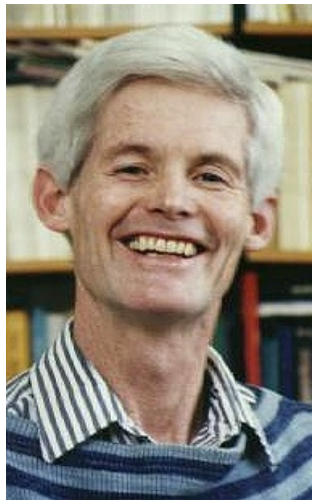
The class of NP-complete problems

But how do we start proving NP-completeness results?

Fortunately we have:

Cook's Theorem [Cook 1970, Levin 1972]:

SAT is NP –Complete (for every $\Pi' \in NP : \Pi' \leq_p SAT$)



S. Cook

Poly-time reductions can be composed

poly-time reduction from Π_1 to Π_2

+ poly-time reduction from Π_2 to Π_3

➔ poly-time reduction from Π_1 to Π_3

➔ proving that Π is NP-complete requires:

1. proving that $\Pi \in NP$
2. proving that there exists some NP-complete problem Π' s.t. $\Pi' \leq_p \Pi$

L. Levin

Hence with Cook's theorem, we have a starting point!

Boolean Formulas and SAT

Boolean variable x: T(RUE) / F(ALSE) or 1 / 0

Boolean operators: AND ($x \wedge y$), OR ($x \vee y$), NOT ($\neg x$)

Literal: a Boolean variable (x) or its negation ($\neg x$ / x)

Boolean formula: $\phi(x,y) = (\neg x \vee y) \wedge (x \vee \neg y)$

SAT

I: a boolean formula ϕ

Q: Is ϕ *satisfiable* ?

(is there a value assignment to its variables making ϕ TRUE ?
= Truth Assignment)

Example: $\phi(x,y) = (\neg x \vee y) \wedge (x \vee \neg y)$ is satisfiable
by the assignments x=y=T
and x=y=F

CNF-SAT

- Clause = A set of OR-ed literals, e.g. $(x \vee \neg y \vee z)$
- A formula is in Conjunctive Normal Form (CNF) if:
  - it is the AND of a set of clauses

E.g. $\phi = (x \vee \neg y \vee z) \wedge (x \vee y) \wedge (\neg x \vee z) \wedge (\neg y \vee \neg z)$

Any formula ϕ can be written in CNF

(CNF)-SAT

I: a boolean formula ϕ in CNF

Q: Is ϕ *satisfiable* ?

From now on, when we refer to SAT, we mean CNF-SAT

3-SAT

(CNF) 3-SAT

I: a 3-CNF boolean formula ϕ (all clauses of ϕ have 3 literals)

Q: Is ϕ *satisfiable* ?

e.g.

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

More NP-complete problems

CLIQUE:

I: A graph G = (V,E), an integer B ≤ |V|

Q: Is there $C \subseteq V$ s.t. $\forall$ u,v $\in$ C: (u, v) $\in$ E and |C| ≥ B?

VERTEX COVER (VC):

I: A graph G = (V,E), an integer B ≤ |V|

Q: Is there $S \subseteq V$ s.t. $\forall$ (u, v) $\in$ E either u$\in$S or v$\in$S (or both) and |S| ≤ B?

INDEPENDENT SET (IS):

I: A graph G = (V,E), an integer B ≤ |V|

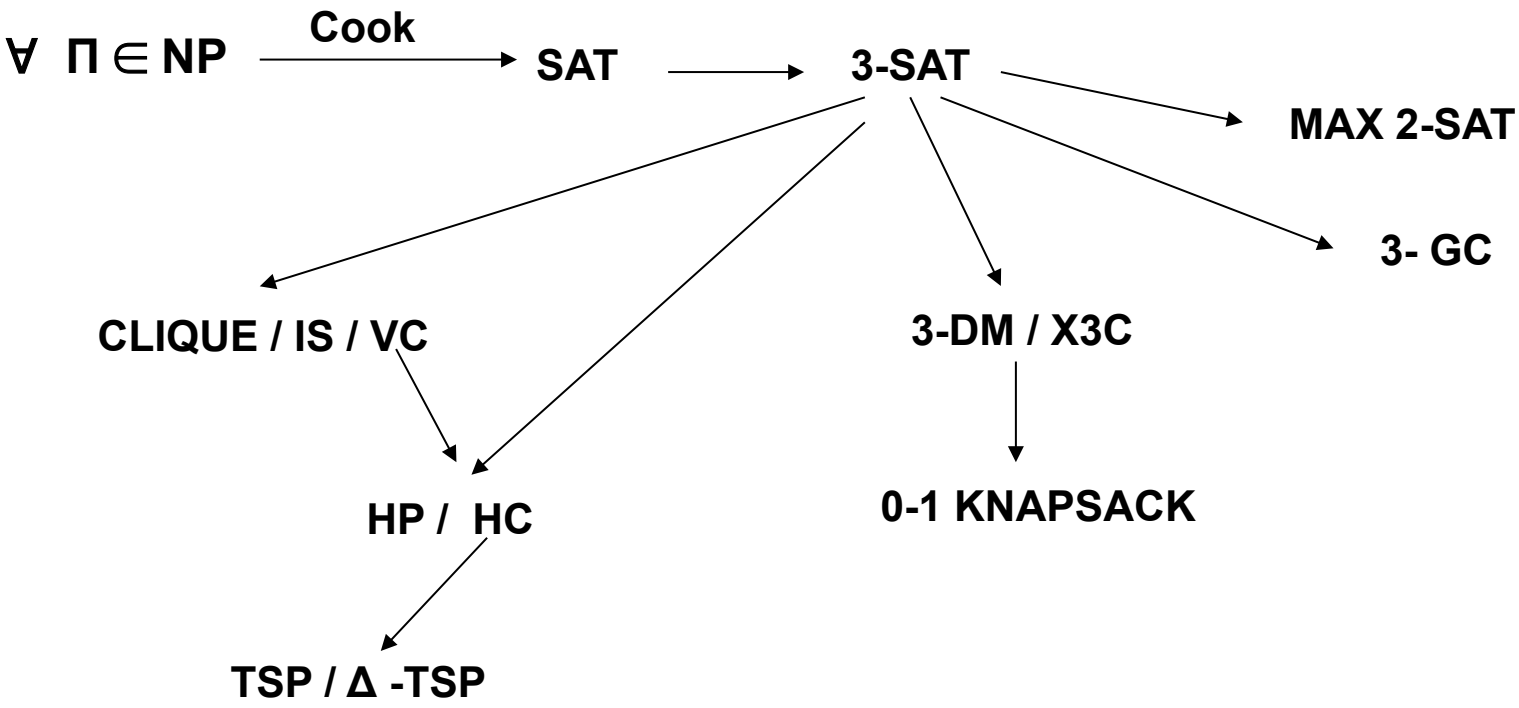
Q: Is there $I \subseteq V$ s.t. $\forall$ u, v $\in$ I: (u, v) $\notin$ E and |I| ≥ B?

3-GRAPH COLORABILITY (3-GC):

I: A graph G = (V,E)

Q: Is there a function f: V→ {1,2,3} s.t. $\forall$ (u, v) $\in$ E: f(u) ≠ f(v)?

Tree of reductions (partial)

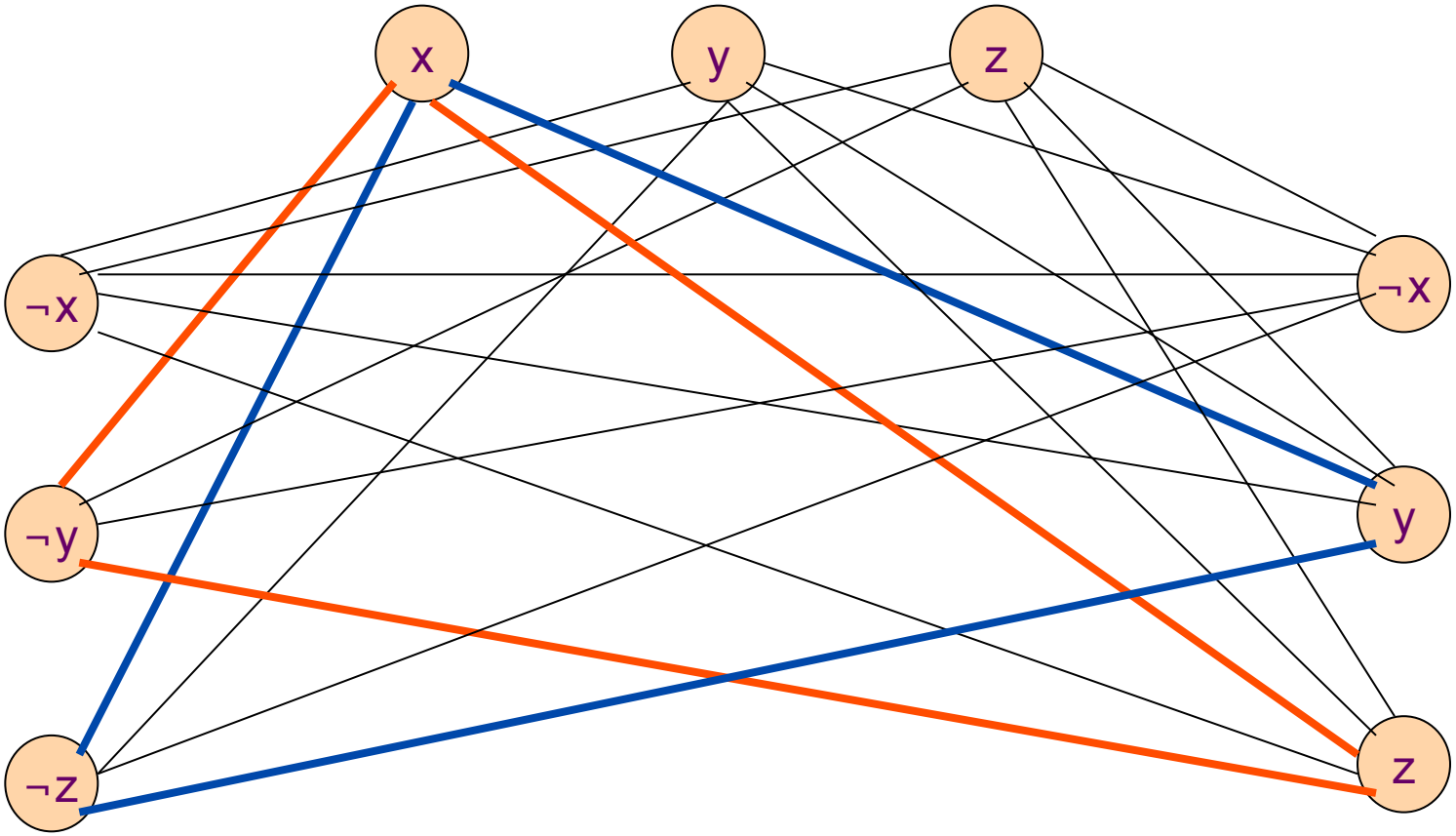


CLIQUE is NP-Complete

1. CLIQUE $\in$ NP – (why?)
2. 3-SAT $\leq_p$ CLIQUE
   - Let ϕ be a 3-CNF formula, and let k = # clauses in ϕ
   - We construct an instance of the form $\langle G,k \rangle$ of CLIQUE such that ϕ is satisfiable iff G has a k-clique
   - Given a formula ϕ, the graph we construct has k triplets of nodes (3k nodes in total)
     - Each triplet corresponds to a clause of ϕ
     - Each node within a triplet corresponds to a literal of the clause
   - Edges of G: all pairs of nodes except
     - Pairs of nodes belonging to the same triplet (i.e., literals belonging to the same clause of ϕ)
     - Pairs of nodes that correspond to opposite literals (x and $\neg$x)

CLIQUE is NP-Complete

Example: $\phi = (x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee z)$



CLIQUE is NP-Complete

ϕ is satisfiable $\Rightarrow$ G has a k-clique

- Satisfiability of $\phi \Rightarrow$ in every clause there is at least one TRUE literal
- Take one such literal (i.e., a node) from every clause
- Let C be the corresponding set of nodes, $|C|=k$
- Then for every $u, v \in C \Rightarrow (u, v) \in E$, because u and v
  - belong to different triplets
  - they cannot correspond to opposite literals (since they are both true)
- Hence, C is a k-clique

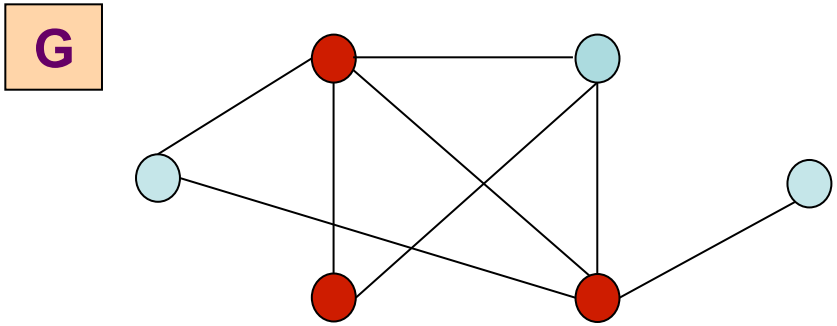
G has a k-clique $\Rightarrow$ ϕ is satisfiable

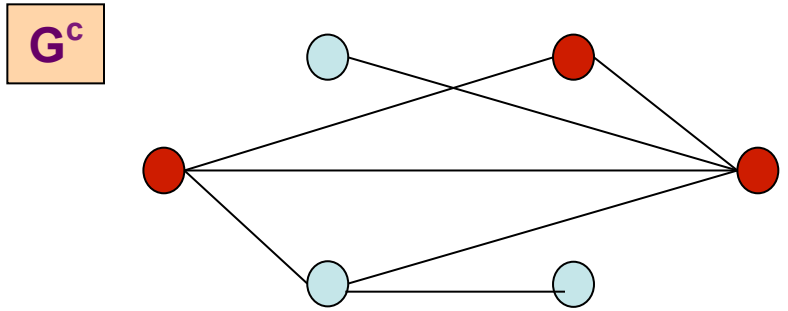
- Let C be a k-clique of G
- Each node (literal) belongs to a different triplet (clause of ϕ)
- No pair of nodes of C can form an opposite pair of literals
- Set all literals corresponding to C as TRUE
- All k clauses of ϕ are satisfied

Note: this is a polynomial time construction (#nodes = 3 • clauses)

Complement of a graph

- Consider a graph G = (V, E)
- Complement of G: $G^c = (V, E^c)$, $E^c = \{(u, v): (u, v) \notin E\}$
- Same set of vertices, edges = all the absent edges from G





VERTEX COVER (VC) is NP-complete

1. VC $\in$ NP – (why?)
2. CLIQUE $\leq_p$ VC
   - Claim: G has a clique of size k iff G^c has a vertex cover of size n-k
   - It suffices to show that:
     C is a clique in G iff V-C is a vertex cover in G^c

   - C is a clique in G $\Rightarrow$ V-C is a vertex cover in G^c
     - suppose $\exists$ edge e = (u, v) $\in E^c$ not covered by V-C
     - then u, v must belong to C,
     - a contradiction, since (u, v) $\notin$ E, and C is a clique in G

   - V-C is a vertex cover in G^c $\Rightarrow$ C is a clique in G
     - suppose $\exists$ u, v $\in$ C and (u, v) $\notin$ E
     - then (u, v) $\in E^c$
     - But V-C is a vertex cover in G^c, hence (u, v) cannot belong to E^c

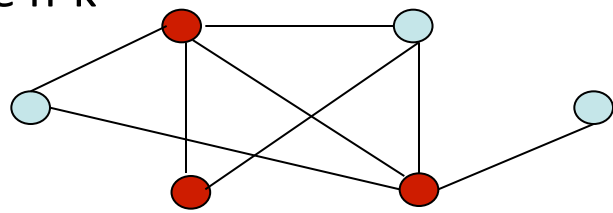
VERTEX COVER (VC) is NP-complete

Reduction:

- Consider an instance I of CLIQUE, given by I = (G, k)
- $\Rightarrow$ construct instance I' = (G^c, n-k) for VC
- Obviously a poly-time reduction!
- By the claim, I has a solution iff I' has a solution

INDEPENDENT SET (IS) is NP-complete

1. IS $\in$ NP – why?
2. VC $\leq_p$ IS
   - G has an IS of size k iff G has a vertex cover of size n-k
   - Let S be an IS in G $\Rightarrow$ V-S is a vertex cover in G
     - suppose $\exists$ edge (u, v) $\in$ E not covered by V-S
     - then u, v $\in$ S, a contradiction since S is an independent set
   - If V-S is a vertex cover in G $\Rightarrow$ S is an IS in G
     - suppose $\exists$ u, v $\in$ S such that (u, v) $\in$ E
     - a contradiction, since V-S is a vertex cover



- Reduction: from an instance I = (G, k) of VC $\Rightarrow$ instance I' = (G, n-k) of IS

CLIQUE, IS and VC

To summarize:

Given a graph G=(V,E), and a subset $C \subseteq V$, the following are equivalent

- C is a vertex cover of G
- V-C is an independent set of G
- V-C is a clique in G^C

3-SAT is NP-complete

1. 3-SAT is in NP
   - Given an assignment, we just check that each clause is TRUE
   - Complexity: O(# clauses), that is O(m)

2. 3-SAT is NP-complete
   - By a reduction from SAT to 3-SAT
   - We will transform each clause of a SAT formula into a set of clauses with 3 literals
   - Independently for each clause based on its length k
   - different cases: k =1, k=2, k=3 (not needed), k >3

3-SAT is NP-complete

- Case 1: consider a clause with k=1, that is C = {z} for some z
  - We use 2 new variables v_1 and v_2
  - We replace C with 4 new clauses

$$C' = C'_1 \wedge C'_2 \wedge C'_3 \wedge C'_4 = (v_1 \vee v_2 \vee z) \wedge (v_1 \vee \neg v_2 \vee z) \wedge (\neg v_1 \vee v_2 \vee z) \wedge (\neg v_1 \vee \neg v_2 \vee z)$$

  - If C is TRUE then C' is TRUE
    - C = T => z = T => C' = T
    - since z is in all four clauses of C'
  - If C' is TRUE then C is TRUE
    - C' = T => C_i' = T, i=1,2,3,4
    - Consider all possible assignments for v_1 and v_2
    - v_1=F, v_2=F: C'_1 = T => z = T
    - v_1=F, v_2=T: C'_2 = T => z = T
    - v_1=T, v_2=F: C'_3 = T => z = T
    - v_1=T, v_2=T: C'_4 = T => z = T
    - that is C = T in each case

3-SAT is NP-complete

- Case 2: consider a clause with k=2, that is C = {z_1 v z_2}
  - We use 1 new variable v_1
  - We replace C with 2 new clauses

$$C' = C'_1 \wedge C'_2 = \{v_1 \vee z_1 \vee z_2\} \wedge \{\neg v_1 \vee z_1 \vee z_2\}$$

  - If C is TRUE then C' is TRUE
    - C = T => z_1 = T or z_2 = T => C' = T
    - since the clauses of C' include z_1 and z_2
  - If C' is TRUE then C is TRUE
    - C' = T => C_i' = T, i=1,2
    - Consider the 2 possible assignments for v_1
    - v_1=F: C'_1 = T => z_1=T or z_2=T
    - v_1=T: C'_2 = T => z_1=T or z_2=T
    - => C = T in each case

3-SAT is NP-complete

- Case 3: consider a clause with k>3, that is C = (z_1 ∨ z_2 ∨ z_3 ∨ … ∨ z_k)
  - We use k-3 new variables $v_1, v_2, v_3, \ldots, v_{k-3}$
  - We replace C with k-2 new clauses

$$C' = C'_1 \wedge C'_2 \wedge \ldots \wedge C'_{k-2}$$

$$= (z_1 \vee z_2 \vee v_1)$$

$$\wedge (\neg v_1 \vee z_3 \vee v_2)$$

$$\wedge \ldots$$

$$\wedge (\neg v_{i-2} \vee z_i \vee v_{i-1})$$

$$\wedge \ldots$$

$$\wedge (\neg v_{k-3} \vee z_{k-1} \vee z_k)$$

3-SAT is NP-complete

- Case 3: consider a clause with k>3, that is C = (z_1 ∨ z_2 ∨ z_3 ∨ … ∨ z_k)
  - If C is TRUE then C' is TRUE
    - C = T => there is at least one z_i=T, 1≤ i ≤ k
    - Let p = min index such that z_p=T
    - Truth assignment for C':

$$v_i = \begin{cases} T, & \text{if } i \le p-2 \\ F, & \text{otherwise} \end{cases}$$

    - C_i' is now T for all i

$$C' = C'_1 \wedge C'_2 \wedge \ldots \wedge C'_{k-2}$$

$$= (z_1 \vee z_2 \vee v_1)$$

$$\wedge (\neg v_1 \vee z_3 \vee v_2)$$

$$\wedge \ldots$$

$$\wedge (\neg v_{p-2} \vee z_p \vee v_{p-1})$$

$$\wedge \ldots$$

$$\wedge (\neg v_{k-3} \vee z_{k-1} \vee z_k)$$

3-SAT is NP-complete

- Case 3: consider a clause with k>3, that is C = (z_1 ∨ z_2 ∨ z_3 ∨ … ∨ z_k)
  - If C' is TRUE then C is TRUE
    - C' = T => C_i' = T for 1≤ i ≤ k-2
    - Assume that C = F
    - => z_i = F, 1 ≤ i ≤ k
    - C_1' = T => v_1 = T
    - C_2' = T => v_2 = T
    - C_3' = T => v_3 = T
    - ….
    - C_{k-3}' = T => v_{k-3} = T
    - => C_{k-2}' = F, a contradiction

$$C' = C'_1 \wedge C'_2 \wedge \ldots \wedge C'_{k-2}$$

$$= (z_1 \vee z_2 \vee v_1)$$

$$\wedge (\neg v_1 \vee z_3 \vee v_2)$$

$$\wedge \ldots$$

$$\wedge (\neg v_{p-2} \vee z_p \vee v_{p-1})$$

$$\wedge \ldots$$

$$\wedge (\neg v_{k-3} \vee z_{k-1} \vee z_k)$$

3-SAT is NP-complete

Complexity of the reduction

SAT formula: n variables, m clauses

In the worst case, all the clauses contain n literals

3-SAT formula: there will be m(n-2) clauses

That is, O(mn) clauses

Coping with NP-complete problems

1. Small instances
2. Special cases
3. Exponential algorithms
4. Approximation algorithms
5. Randomized algorithms
6. Heuristic algorithms

Coping with NP-complete problems

1. Small instances

If we want to run an algorithm in small instances only, then an exponential algorithm may be satisfactory

2. Special cases

Identify families of instances where we can have an efficient algorithm, e.g., 2-SAT

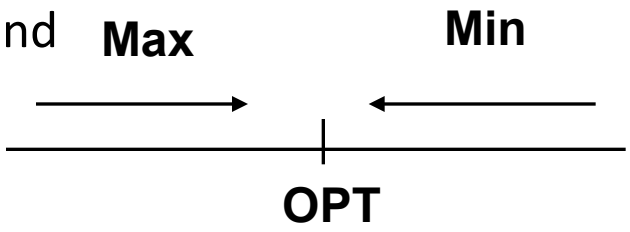
3. Exponential algorithms

Some worst-case exponential algorithms may still be better than brute-force or have a good average-case behavior: Pseudo-polynomial algorithms, Dynamic Programming, Backtracking, Branch-and-Bound

Approximation algorithms

4. Approximation algorithms

algorithms for which we can have a provable bound on the quality of the solution returned



Given an instance I of an optimization problem Π:

- OPT(I) = optimal solution
- C(I) = cost of solution returned by the algorithm under consideration

Definition: An algorithm A, for a minimization problem Π, achieves an approximation factor of ρ (ρ ≥ 1), if for **every** instance I of the problem, A returns a solution with:

$$C(I) \leq \rho\, OPT(I)$$

(analogous definition for maximization problems)

Approximations: Good, better, best and more …

Non - constant approximation: $C(I)/OPT(I) \leq f(n)$ for some function that depends on n

Constant (ρ-)approximation: $C(I)/OPT(I) \leq \rho$, where ρ is a constant, e.g. 3/2

Polynomial Time Approximation Schemes (PTAS):

- $C(I)/OPT(I) \leq 1 + \varepsilon$, for any $\varepsilon > 0$ (any constant factor is achievable)
- Complexity should be O(poly (|I|)) and O(exp (1/ε)), e.g. $O(n^{3/\varepsilon})$

Fully Polynomial Time Approximation Schemes (FPTAS):

- $C(I)/OPT(I) \leq 1 + \varepsilon$, for any $\varepsilon > 0$
- Complexity should be O(poly (|I|)), O(poly (1/ ε)) !!!
- e.g. $O((1/\varepsilon)^2 n^3)$, dependence on 1/ε should be on the exponent

Additive approximation:

- $C(I) \leq OPT(I) + f(n)$ or $C(I) \leq OPT(I) + k$ (a constant), e.g. $C \leq OPT+1$

Coping with NP-complete problems

5. Randomized algorithms

algorithms that use randomization (e.g. flipping coins) and make randomized decisions

Performace: Such algorithms may

- produce a good solution with high probability
- Produce a good expected cost
- Run in expected polynomial time

Power of randomization: for some problems, the only decent algorithms known are randomized!

Coping with NP-complete problems

6. Heuristic algorithms

Algorithms that seem to work well in practice
without a formal guarantee though of their performance

- No guarantee on the approximation achieved by the solution returned
- Some times no guarantee that they terminate in polynomial time