NP, NP-completeness and a Million Dollar Question

March 25, 2023

History of P

- Notion of Efficient Algorithms there since ancient times
- Addition, Multiplication, GCD, Repeated squaring (Pingala), Astronomical calculations.
- [1950s] Dynamic Programming, Shortest Path, Simplex algorithm, Minimum spanning tree
- [1960s] FFT, Scheduling, Network flow, bipartite matching, and related combinatorial problems
- Doing better than brute force search

P (Polynomial time solvable)

- Edmonds [1965] proposed polynomial time as a characterization of efficient computation
  “It is by no means obvious whether or not there exists an algorithm whose difficulty increases only algebraically with the size of the graph”
  ”For practical purposes the difference between algebraic and exponential is more crucial than between finite and non-finite.”

- Why polynomial time?
  - if a procedure is considered efficient, running it n times might also be considered efficient.
  - Polynomial time remains independent of computation model.
  - Another perspective: if you double the input size, the running time gets multiplied by a constant.

Not in P

- [1960s] For many problems, people could not find better than exponential time algorithms.
- There was no clear explanation why some problems are in P, while others are not.
- Try to guess, whether a polynomial time algorithm is known or not.

In P or not in P ?

- Roommate Allocation:
  - n students, some like each other, some don't.
  - Allocate rooms s.t. roommates like each other.
  - Polynomial time algorithm known or not?
  - Yes [Edmonds 1965]

In P or not in P ?

- Triple Roommate Allocation:
  - n students, some like each other, some don't.
  - Allocate rooms s.t. all 3 roommates like each other.
  - Polynomial time algorithm known or not?
  - No

In P or not in P ?

- Given a graph and a number k, is there a path of length k?
  - Not known to be in P
- Given a graph with s and t vertices,
  are there two edge disjoint paths from s to t ?
  - In P
- Given a graph and four vertices s_1 , s_2, t_1, t_2,
  are there disjoint paths $s_1 \rightsquigarrow t_1$ and $s_2 \rightsquigarrow t_2$?
  - In P
- Same problem in directed graphs?
  - Not known to be in P

In P or not in P ?

- Given a graph with edges colored red or blue, is there an s-t path with alternating red and blue edges?
  - In P
- Same problem in directed graphs?
  - Not known to be in P

In P or not in P ?

- Given a graph, does it have a cycle?
  - In P
- Given a graph, does it have two vertex-disjoint cycles?
  - Not known to be in P. *
- Given a graph, partition the vertices into vertex-disjoint cycles?
  - In P

In P or not in P ?

- Given a number (in binary), is it factorizable?
  - In P (only in 2002)
- Given a number (in binary), find its factors?
  - Not known to be in P

In P or not in P ?

- Given a set of intervals, largest subset of disjoint intervals
  - In P
- Given a graph, find the largest independent set (vertices sharing no edges).
  - Not known to be in P
- Given a graph, find the largest set of edges not sharing any vertex
  - In P
- Given a graph, find the largest set of triangles not sharing any vertex
  - Not known to be in P

In P or not in P ?

- Set of trains arriving/departing at a station, can we schedule using k platforms ?
  - In P
- Given a list of courses, and pairs which should avoid a clash, can we schedule using k time slots?
  - Not known to be in P
  - Also known as graph coloring (easy for 2 colors)
- n Jobs, m processors, not every processor can handle every job. Processors can work in parallel. Can we finish in k units of time?
  - In P (via Network Flow)

In P or not in P ?

- Given a set of integers, is there a subset with sum equal to zero?
  - Not known to be in P
- Given a set of integers (loads), distribute them among m machines, so that maximum total load (makespan) is minimized.
  - Not known to be in P
  - known as load balancing
- Given a set of integers, partition them into two groups with equal sum
  - Not known to be in P
  - Known as Partitioning

In P or not in P ?

- Minimum weight spanning tree
  - In P
- Steiner Tree: Given a subset of vertices (terminals), find the minimum weight tree that connects the terminals
  - Not known to be in P
- Traveling Salesperson problem: given a list of cities, you have to visit every city and come back with minimum cost
  - Not known to be in P

In P or not in P ?

- Satisfiability: given a Boolean formula, is it satisfiable?
  - Not known to be in P
- Minimum circuit size: Given a Boolean function, is there a circuit for it with at most k Boolean operations?
  - Not known to be in P

Towards NP

- [1960s] For many problems, people could not find better than exponential time algorithms.
  - Subset sum, Load balancing, Traveling Salesperson, Graphs Isomorphism, Primality, Linear programming, Minimum Circuit Size, Satisfiability, 3-colorability
- People observed some of these can be reduced to others.
- For example, 3-colorability <= SAT

NP or Easily Verifiable Proofs

- Many problem which seemed hard have easily verifiable proofs for `yes' inputs.
- Load Balancing: is there a load allocation with makespan at most k?
  - Proof: an allocation with makespan k'<= k
  - Verifier: check if the proposed allocation is valid and its makespan
- Factorize Numbers: is a given number factorizable?
  - Proof: two factors
  - Verifier: multiply the proposed two numbers and check if you get the input number.
- Not clear if 'no' inputs have easily verifiable proofs.

Easily Verifiable Proofs

- SAT: given a Boolean formula (CNF - AND of ORs), is there an assignment of variables, which makes it true?
  Example. $(\neg x \vee y) \wedge (\neg y \vee x)$
  - Proof: an satisfying assignment to the variables (example: True, False)
  - Verifier: check if the proposed assignment makes the formula true.
- Graph Isomorphism: given two graphs, are they isomorphic?
  - Proof: a mapping between two sets of vertices
  - Verifier: check if the given mapping preserves edges and non-edges

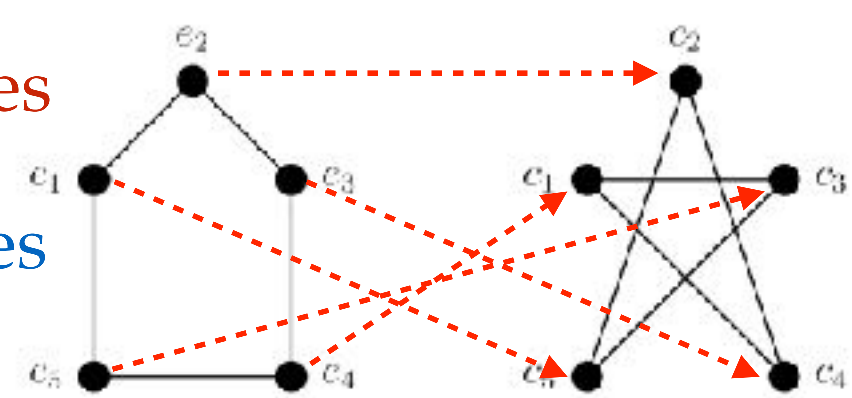
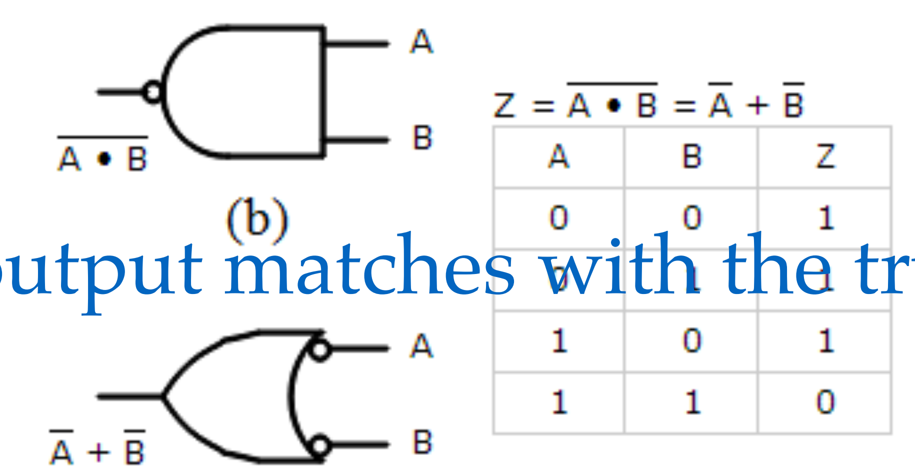


image source: [1]

Easily Verifiable Proofs

- Subset Sum: given numbers $a_1, a_2, a_3, \ldots, a_n$ and a number b, is there a subset of a_i's that sum up to b ?
    - Proof: a subset of numbers
    - Verifier: check if the proposed subset has sum equal to b
- Circuit Size: given a Boolean function f truth table, is there a circuit with at most s gates that computes f ?
    - Proof: a circuit
    - Verifier: verify if the circuit output matches with the truth table for every input



The Class NP

- Definition: a 'yes or no' decision problem is in NP if there is an easily verifiable proof for each `yes' input.

- a 'yes or no' decision problem is in NP if there is a polynomial time Algorithm V (verifier), such that for any input x,

  - if x is a 'yes' input then there exists y s.t. $|y| <= poly(|x|)$ and $V(x,y) = True$.

  - if x is a 'no' input then for any y, $V(x,y) = False$

The Class NP

- P - can find the solution in polynomial time
  NP - can verify a proposed solution in polynomial time
  $P \subseteq NP$
- if a problem is in P, it is also in NP.
- A problem falling into NP is a positive thing.
- NP does not mean Non-Polynomial time.

Problems in NP?

- Given an integer n, are there integers x, y, z such that

  $$x^3 + y^3 + z^3 = n\,?$$

  e.g. for n = 39: $134476^3 - 159380^3 + 117367^3 = 39$.

  - Not clear, because x, y, z can much larger than n. Efficient verification might not be possible.

- Given a graph, can we remove at most k edges to make it 3-colorable?

  - Proof: k edges to remove and a coloring scheme with 3 colors.

- Given a graph, is there a matching of size at least k?

  - Proof: a matching of size k, which verifier can check.

  - Proof: verifiers ignores the proof and just finds the maximum matching. If at least k, then say yes.

- Can you color the edges with red/blue, s.t. every subset of k vertices has edges with both colors

  - Not clear. If someone gives a coloring scheme, not clear how to verify it for every subset.

Towards NP

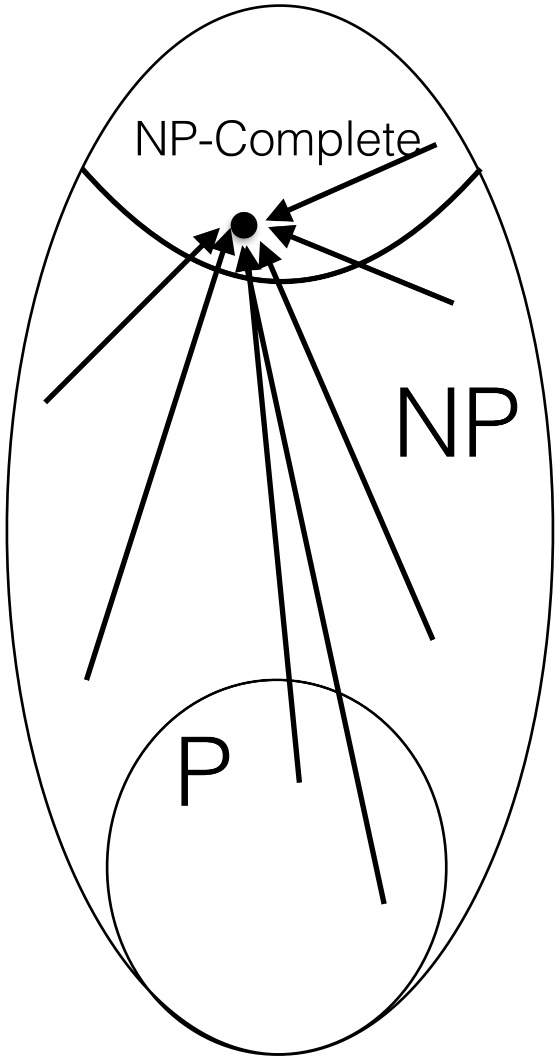
- [1960s] For many problems, people could not find better than exponential time algorithms.
  - Subset sum, Load balancing, Traveling Salesperson, Graphs Isomorphism, Primality,
    Linear programming, Minimum Circuit Size, Satisfiability, 3-colorability
- People observed some of these can be reduced to others.
- For example, 3-colorability <= SAT
- [Cook, Levin 1971] All of these problems reduce to SAT

3-colorability reduces to SAT

- Example: 3-colorability reduces to SAT
- Given a graph, can we color vertices with 3 colors?
  - create Boolean variables to represent the `proof'
  - 3 Boolean variables for each vertex - $x_{i,}\, y_{i,}\, z_i$
  - encode the verification procedure as Boolean constraints
    - each vertex has a color — $(x_i \vee y_i \vee z_i)$ for each i
    - adjacent vertices have different colors — $\neg(x_i \wedge x_j)$, $\neg(y_i \wedge y_j)$, $\neg(z_i \wedge z_j)$ for every edge (i,j)
    - Boolean formula = AND of all the constraints.
  - Graph is 3-colorable if and only if there is an satisfying assignment for the above Boolean formula

NP-completeness

- Various problems like TSP, SAT were conjectured to be not in P, but not enough evidence.
- Cook-Levin [1971]: If we have a subroutine for SAT problem, we can design a polynomial time algorithm for every problem in NP
  - for any problem A in NP, A <= SAT
  - SAT is 'NP-complete'.
- Karp [1972]: 21 other problems are NP-complete.
  - TSP, Subset Sum, Integer Programming, Graph Coloring, Job Sequencing, Independent Set, 3D-matching etc.
  - They are all equivalent and are hardest problems in NP



The tree of Reductions

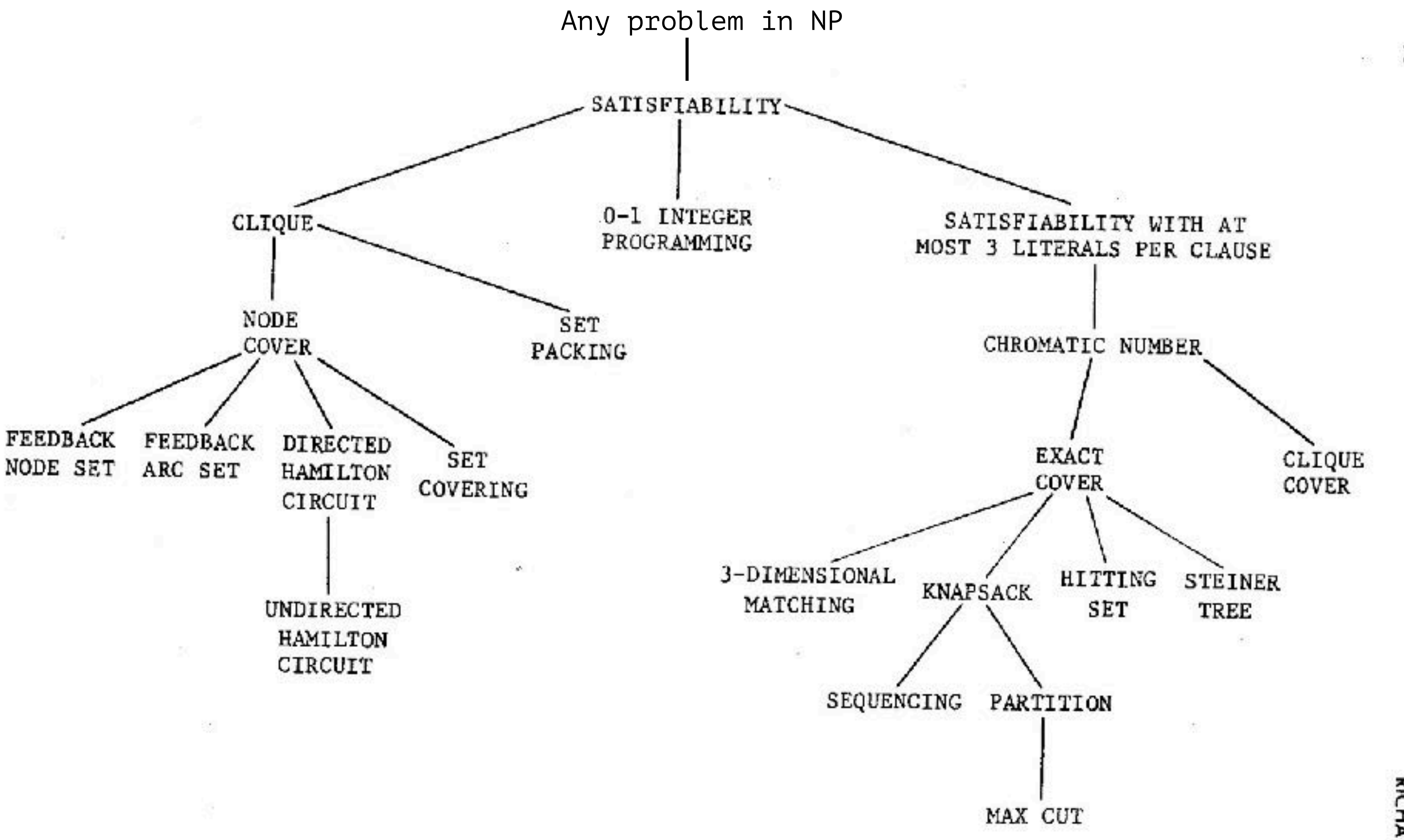


FIGURE 1 - Complete Problems

P vs NP

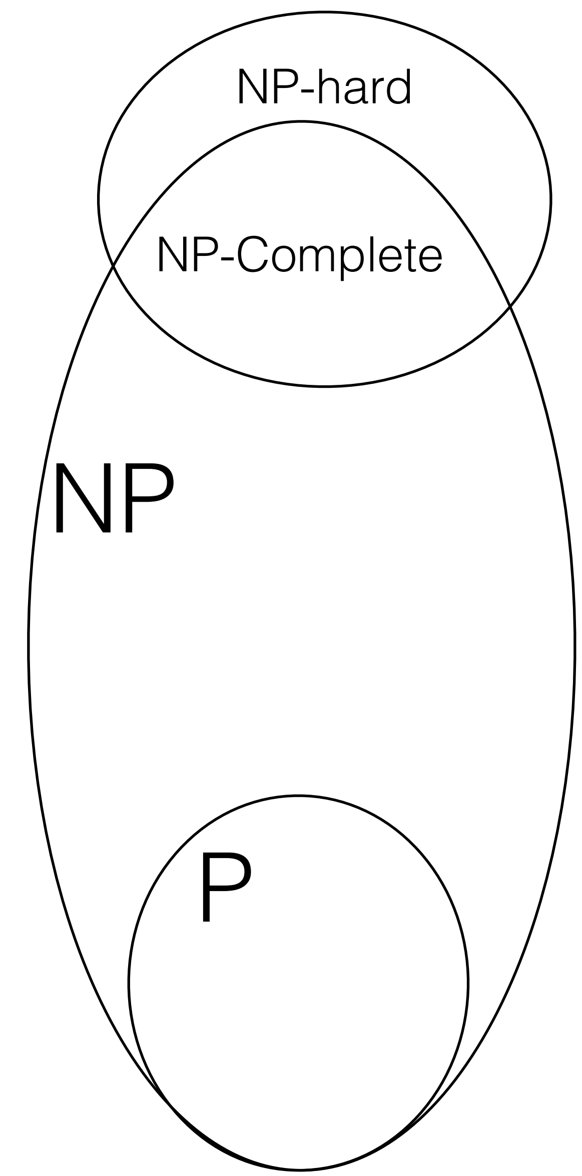
- Problems intuitively/philosophically in class NP
  - is a given mathematical statement (provably) true? (a proposed proof can be verified)
  - is there a cure for a mentioned disease? (a proposed cure can be verified)
  - given the public key, can you find the private key? (a private-public key pair can be verified)
- P vs NP = Mechanical vs Creativity

Reductions

- Problem A reduces to problem B ($A \leq B$)
  - if A can be solved in polynomial time using a given subroutine that solves B.
  - task of solving A reduces to task of solving B
- Example: Taxi scheduling reduces to bipartite matching
- Example: Multiplication reduces to squaring
  - Multiplication is as easy as squaring
  - Squaring is as hard as Multiplication
- A reduces to B: 1) convert phi_A to phi_B. 2) Solution(phi_B) should be converted to solution(phi_A).
  Conclusion: A is as easy as B. B is as hard as A.
- $A \leq B$ and $B \leq C$ implies $A \leq C$

NP-complete and NP-hard

- Problem X is said to be NP-complete if
  1. X is in NP
  2. Every problem in NP reduces to X
- Problem Y is said to be NP-hard if
  - Every problem in NP reduces to Y



Summary

- Thousands of problems have been shown to be NP-complete.
- If you solve any of them, all of them get solved.
- One can say, there is just one NP-complete problem.
- People have not been able to give an efficient algorithm in last 50 years.
- P=NP would mean all these problems have efficient algorithms.
  - all diseases can be cured,
  - all mathematical conjectures can be resolved,
  - crypto systems can broken
  - All film critics can make great films

Summary

- Widely believed P ≠ NP , but no proof for it.
  Million Dollars for a proof either way.
- If you can't find a Polynomial time algorithm for a problem *X*, try to prove that it is NP-hard.
  - Choose a suitable NP-complete/NP-hard problem *H* and reduce *H* to your problem *X*.
  - I.e., *H* can be solved using a subroutine for *X*.
  - "I am not able to design an algorithm for it, but nobody could in last 50 years 😊"
- Most problems turn out to be either in P or NP-complete.
  - Exceptions: Graph Isomorphism, Minimum circuit Size

Babai 2015,
quasi-poly

Any problem in NP reduces to SAT [Section 8.4 in Kleinberg Tardos]

- There is a verifier algorithm V such that for any input x,
  - if x is a 'yes' input then there exists y s.t. $V(x,y)$ = True.
  - if x is a 'no' input then for all y, $V(x,y)$ = False
- Reduction: Given x, output a boolean formula $f(x)$ such that
  - if x is a 'yes' input then $f(x)$ has a satisfying assignment
  - if x in a 'no' input then $f(x)$ does not have a satisfying assignment
- Proof y encoded as Boolean variables.
- Each step of algorithm V will be converted to a Boolean constraint.

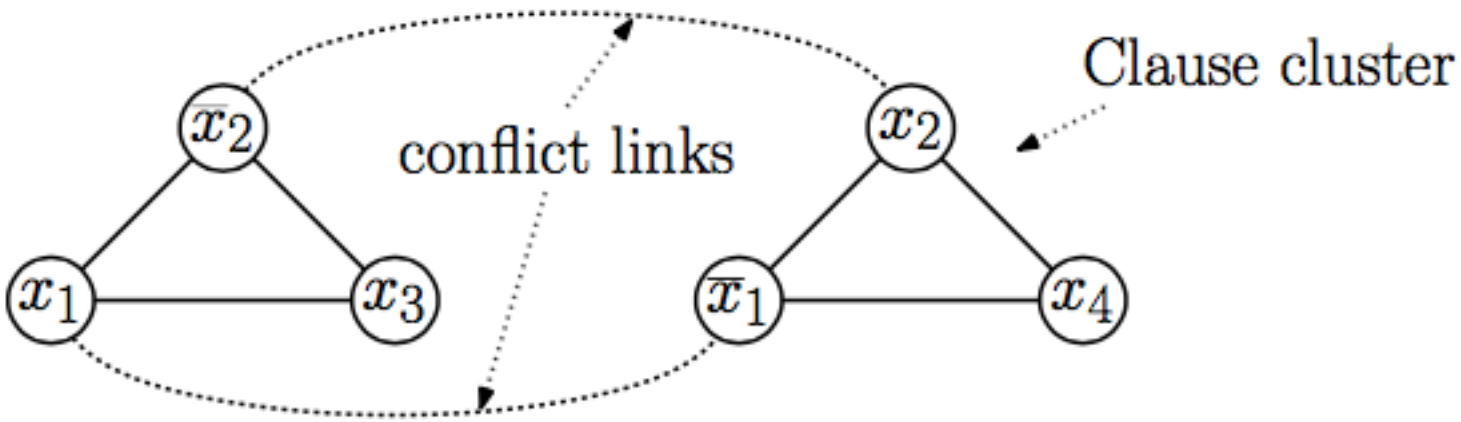
Any problem Q in NP reduces to SAT

- Algorithm *V*: Input $(1,0,1,0,0,.., y_1, y_2, \ldots, y_m)$
- Say it uses *p* bits memory and time *T*.
- Create another *pT* Boolean variables.
- At time t, an instruction will apply AND/OR/NOT on some memory locations and store it in another location
- $$z_{t+1,5} = z_{t,3} \vee z_{t,9}$$
- $f(x) =$ AND of all such Boolean constraints.
- $f(x)$ has a satisfying assignment (y,z)
  if and only if algorithm *V* outputs True on input $(x, y_1, y_2, \ldots, y_m)$
  if and only if x is a yes input.

SAT to IND-SET Reduction

- IND-SET: given a graph *G* and a number *k*, is there an independent set of size *k*?
- Reduction: Given a CNF formula ϕ, output a graph G(ϕ) and a number k(ϕ) such that
  - if ϕ has a satisfying assignment, then G(ϕ) has an independent set of size k(ϕ)
  - if ϕ does not have a satisfying assignment, then G(ϕ) does not have any independent set of size k(ϕ)
  - $\phi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$.
  - you have to choose one literal form every clause to be true and you can choose only one between x_2 and $\neg x_2$.

G(ϕ) =



Thank you

References

- [1] https://math.stackexchange.com/questions/3141500/are-these-two-graphs-isomorphic-why-why-not
- [2] http://electronics-course.com/logic-gates