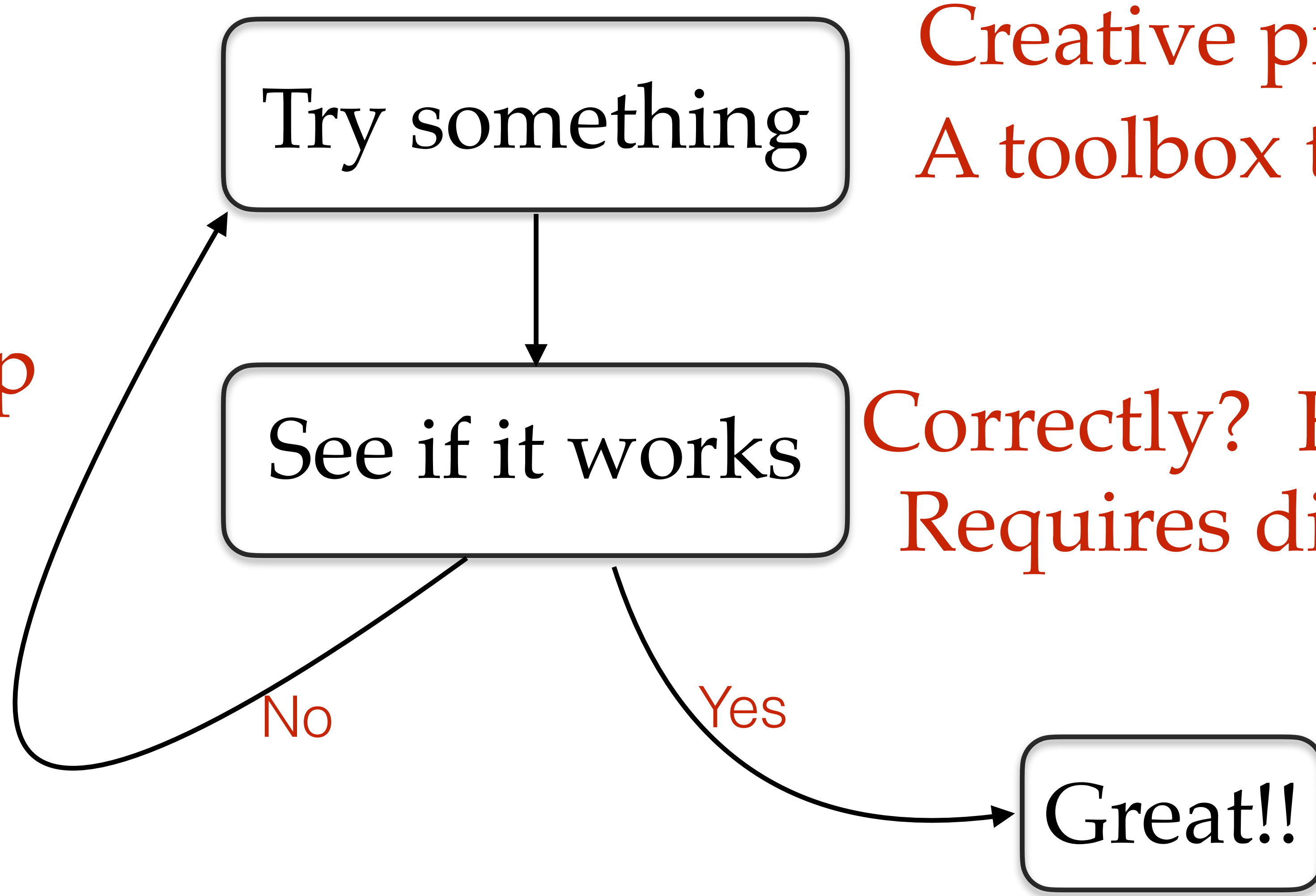
NP, NP-completeness and a Million Dollar Question

March 7, 2024

Objectives

- How to design algorithms.



Creative process
A toolbox to help

Correctly? Efficiently?
Requires diligence

When to stop this loop?

check if it's NP-complete

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."

P (Polynomial time solvable)

- 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.
  - If the running time is n^{100}, is it still efficient?
    - In reality, we never get such running times.

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 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

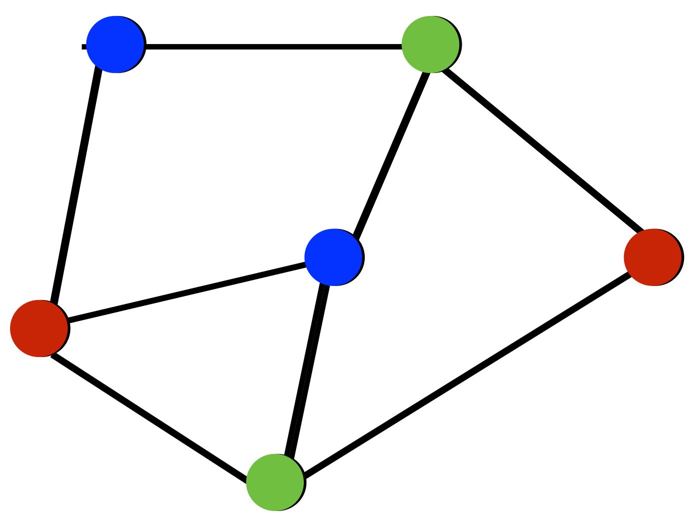
In P or not in P ?

- 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

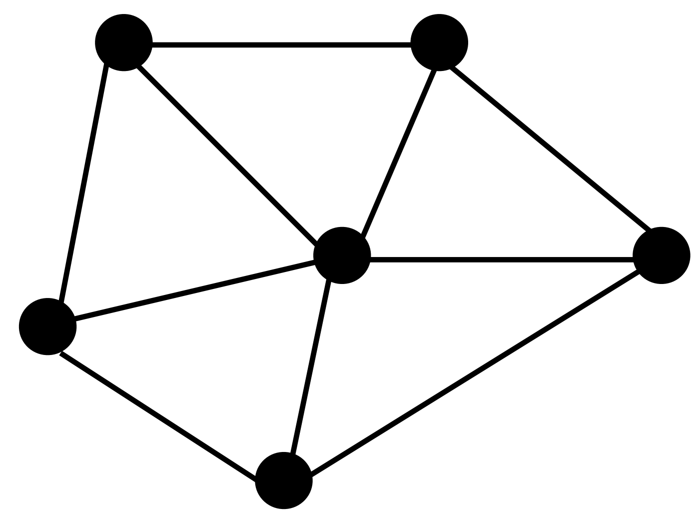
Towards NP

- [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 $\leq$ SAT
- In fact, all of these problems reduce to SAT.
- The key common property of these problems was having "easily verifiable proofs" for the 'yes' instances.

3-colorability



A 3-colorable graph



A graph
not 3-colorable

Satisfiability

- $(x \vee y \vee z)$ AND $(\neg x \vee y)$ AND $(x \vee y)$ AND $(x \vee \neg y)$
- Satisfiable
- $(\neg x \vee y)$ AND $(\neg y \vee z)$ AND $(\neg z \vee \neg x)$ AND (x)
- Unsatisfiable

3-colorability reduces to SAT

- Given a graph, can we color vertices with 3 colors?
  - create Boolean variables to represent the coloring
  - 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
    - for every edge (i,j) : $\neg(x_i \wedge x_j)$, $\neg(y_i \wedge y_j)$, $\neg(z_i \wedge z_j)$
    - 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
  - An algorithm for SAT will give an algorithm for 3-colorability
- [Cook, Levin 1971] All of the problems mentioned reduce to SAT

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

Reductions

- Problem A reduces to Problem B:
  - (1) convert input φ_A for A to input φ_B for B (or set of inputs φ_{B1} , φ_{B2} , φ_{B3})
  - (2) Solution(φ_B) should be converted to solution(φ_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 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 $k' \leq k$
  - 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. $(\neg x \lor y) \land (\neg y \lor x)$
  - 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

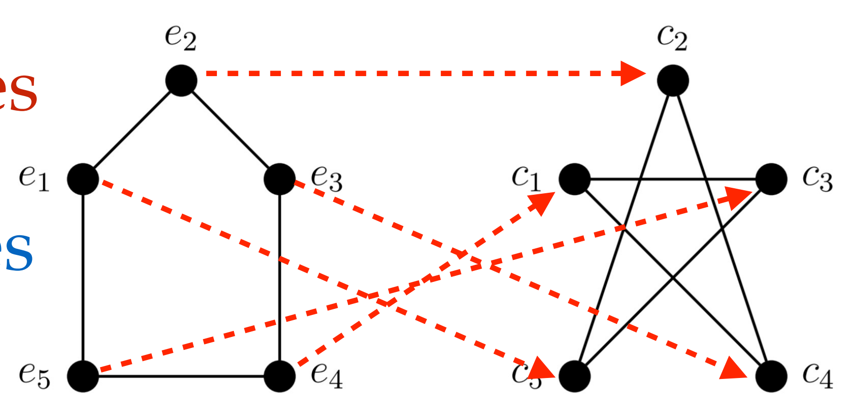
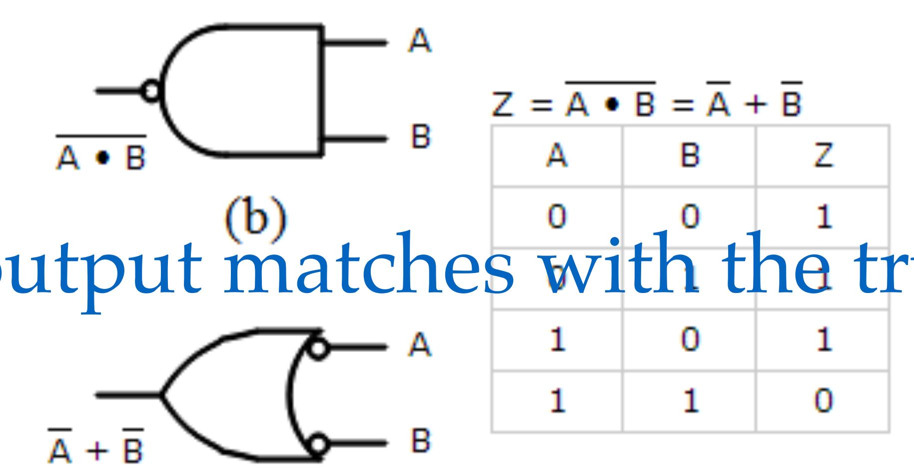


image source: [1]

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 c s.t. $\text{size}(c) \leq \text{poly}(\text{size}(x))$ and $V(x, c) = \text{True}$.
  - if x is a 'no' input then for any c, $V(x, c) = \text{False}$

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.
- NP stands for Non-deterministic polynomial time.

Problems in NP?

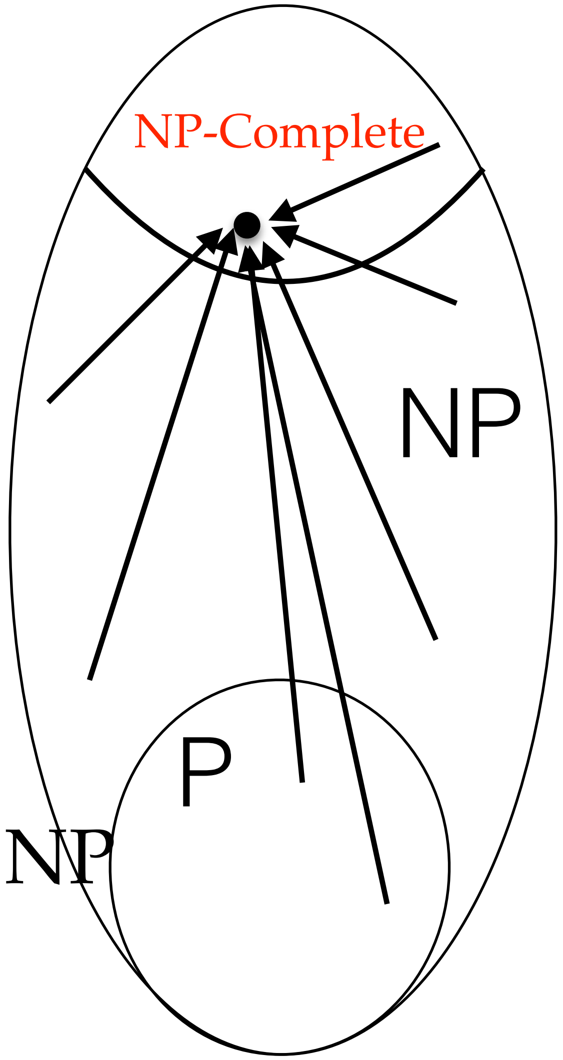
- Given an integer n, are there integers x, y, z such that
$$x^3 + y^3 + z^3 = n\ ?$$
e.g. for $n = 39$: $134476^3 - 159380^3 + 117367^3 = 39$.
    - Not clear, because x, y, z can be much larger than n. Efficient verification may not be possible.
- Given a flow network, is there a flow with $f^{out}(s) = k$?
    - Yes.
    - Proof: for every edge a flow value.
        - Verifier checks flow conservation, capacity constraints, and computes $f^{out}(s)$
    - Proof: 0
        - Verifier runs the max flow algorithm and checks whether $max\ flow \geq k$

Problems in NP?

- Given two Polynomial expressions, are they equal?
  - e.g., $(a^2 + n\, b^2)\,(c^2 + n\, d^2) = (ac - n\, bd)^2 + n(ad + bc)^2$
- Not clear if it is in NP
- Given two Polynomial expressions, are they different?
  - It is in NP
  - Proof: a substitution of variables with numbers
  - Verifier: evaluates both the expressions on this substitution and verifies if evaluations are different.

NP-completeness

- 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, $A \leq$ 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

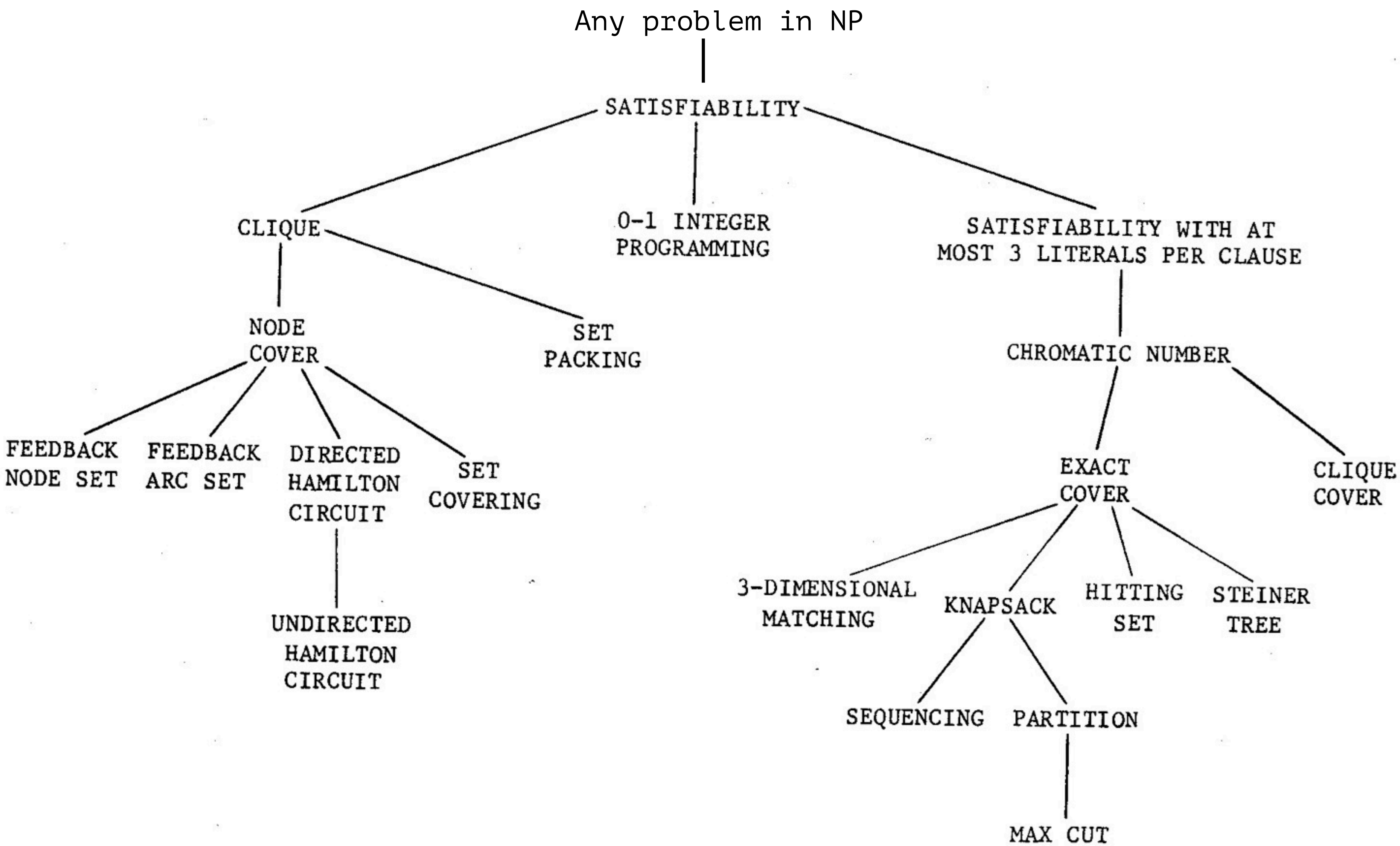


FIGURE 1 - Complete Problems

P vs NP

- Thousands of problems have been shown to be NP-complete.
- If you solve any of them, all of them get solved.
- People have not been able to give an efficient algorithm in last 50 years, for any of these.
- One can say, there is just one NP-complete problem.
- P= NP $\iff$
  SAT (and every other problem in NP) has a polynomial time algorithm

Philosophically…

- 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)
  - is there a great film? (given a film, you can critic it)

Philosophically…

- P vs NP = Mechanical vs Creativity
- P = NP would mean
  - all diseases can be cured,
  - all mathematical conjectures can be resolved,
  - crypto systems can broken
  - All film critics can make great films

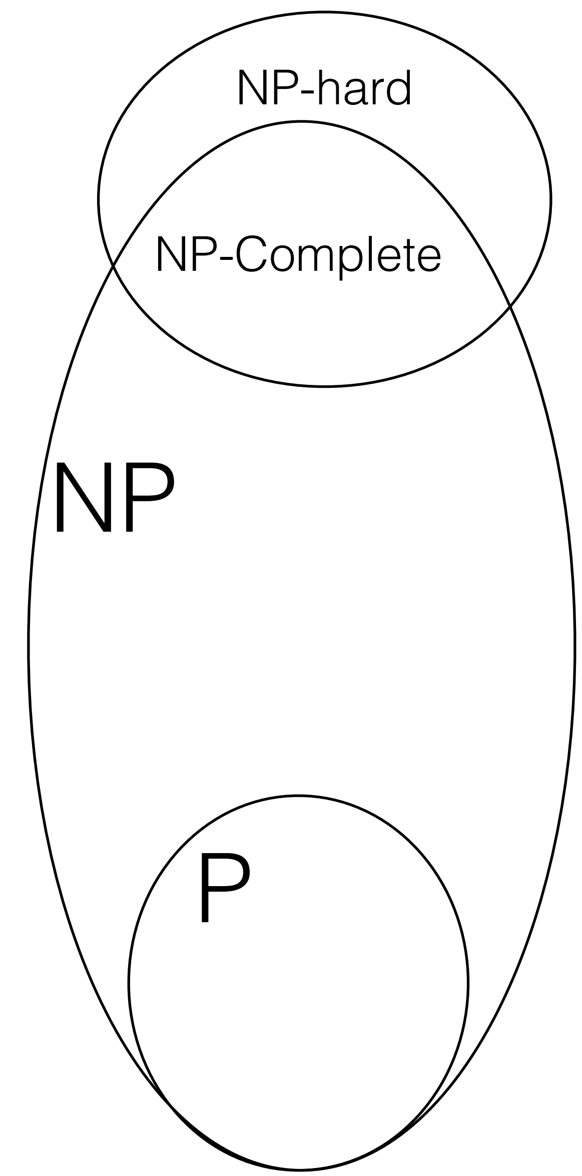
How is it useful?

- Widely believed P ≠ NP , but no proof for it.
  Million Dollars for a proof either way.
- You encounter a new problem X
  and can't find a Polynomial time algorithm for it
  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 😊"
- Amazingly, most problems turn out to be either in P or NP-complete.
  - Exceptions: Graph Isomorphism, Minimum circuit Size, Factoring

Babai 2015,
quasi-poly

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



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, …, 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, …, y_m)*
  if and only if *x* is a yes input.

IND-SET decision

- IND-SET (OPT): given a graph G, find the largest independent set of vertices.
- IND-SET (decision): given a graph G, and a number k, is there an independent set of size k?
- Reduction from IND-SET (OPT) to IND-SET (decision)
  - first find the largest size of an independent set
    - start with $k \leftarrow n$
    - keep decreasing k till G has no independent set of size $k\text{-}1$

IND-SET optimization and decision

- Reduction from IND-SET (OPT) to IND-SET (decision)
  - first find the largest size of an independent set, say it is k
  - check whether $G - v_1$ has an independent set of size k
    - if yes, recursively find an independent set of size k in $G - v_1$
    - if no, then recursively find an independent set of size $k-1$ in (G – neighbors(v_1))
      and include v_1 with it.

IND-SET is NP-complete

- (1) IND-SET is in NP
  - the proof will be an independent set of size *k*
  - the verifier will check if it is indeed an independent set and has size *k*
- (2) IND-SET is in NP-hard
  - That is, any problem in NP reduces to IND-SET
  - We already know that any problem in NP reduces to SAT
  - We will show that SAT reduces to IND-SET
  - From transitivity of reductions, it will follow that IND-SET is NP-hard
  - Step 1 (homework): SAT reduces to 3-SAT (given formula is 3-CNF)

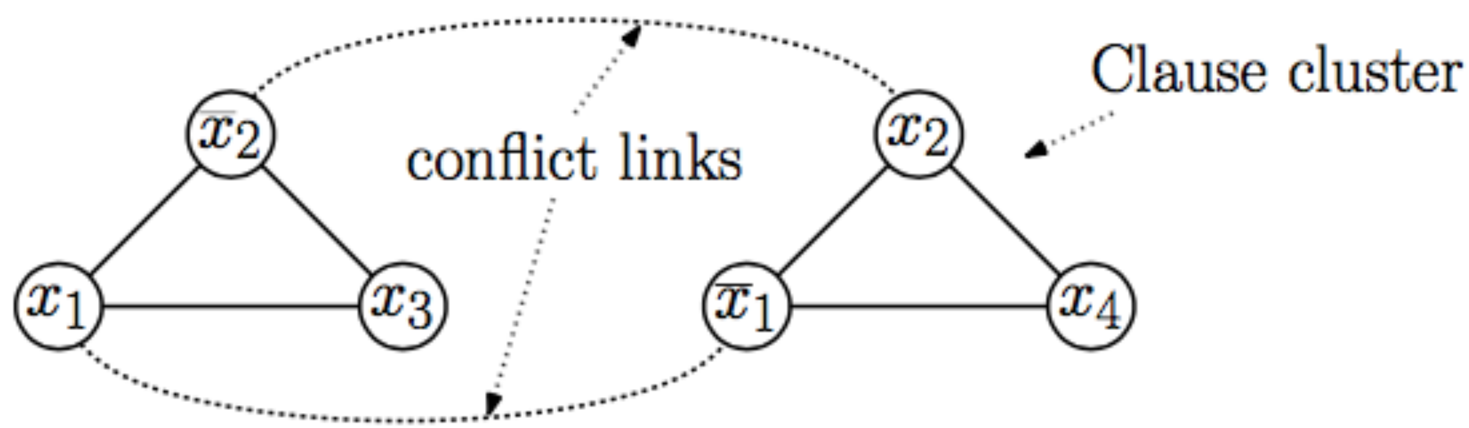
3-SAT to IND-SET Reduction

- IND-SET: given a graph G and a number k, is there an independent set of size k?
- Reduction: Given a 3-CNF formula ϕ, we want to construct a graph $G(\phi)$ and a number $k(\phi)$ such that
  - if ϕ has a satisfying assignment, then $G(\phi)$ has an independent set of size $k(\phi)$
  - if ϕ does not have a satisfying assignment, then $G(\phi)$ does not have any independent set of size $k(\phi)$
  - Construction of $G(\phi)$:
    - for every clause, introduce a triangle with one vertex corresponding to each literal in the clause
    - add an edge between two vertices across triangles if one corresponds to x_i and the other corresponds to $\neg x_i$
  - $k(\phi)$ = number of clauses = number of triangles

SAT to IND-SET Reduction

- Example:
- $\phi = (x_1 \lor \neg x_2 \lor x_3) \;\land\; (\neg x_1 \lor x_2 \lor x_4)$

$G(\phi) =$



- $k(\phi) = 2$

Proof of correctness ($\Longrightarrow$)

- Suppose ϕ is satisfiable, choose one satisfying assignment.
- From each clause, pick an arbitrary literal which is set to true.
- Pick vertices corresponding to the picked literals.
- The number of vertices picked is equal to the number of clauses.
- The picked vertices form an independent set because
  - 1) we pick one vertex from each triangle
  - 2) if vertex corresponding to literal x_i is picked then its neighbors correspond to $\neg x_i$, and hence will not be picked.

Proof of correctness ($\Longleftarrow$)

- Suppose $G(\phi)$ has an independent set of size $k(\phi)$.
- Since $k(\phi)$ = number of triangles, the independent set must have one vertex from each triangle.
- For each vertex in the independent set, set corresponding literal true.
- This is possible because the independent set cannot have vertices corresponding to both x_i and $\neg x_i$
- Set the remaining variables (if any) arbitrarily.
- This is a satisfying assignment ϕ for because for every clause at least one literal is set true.

Homework

- Easy reductions:
  - IND-SET reduces to VERTEX-COVER
  - IND-SET reduces to CLIQUE
  - VERTEX-COVER reduces to SET-COVER
  - SAT reduces to INTEGER LINEAR PROG
- Not so easy:
  - k-CLIQUE reduces to k-COLORING
  - SAT reduces to DIRECTED HAMILTONIAN CYCLE

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