Reductions

- designing algorithms
- establishing lower bounds
- establishing intractability
- classifying problems

Bird's-eye view

Desiderata. Classify problems according to computational requirements.

- Linear: min/max, median, Burrows-Wheeler transform, ...
- Linearithmic: sort, convex hull, closest pair, ...
- Quadratic:
- Cubic:
- ...
- Exponential:

Frustrating news.
Huge number of fundamental problems have defied classification.

Bird's-eye view

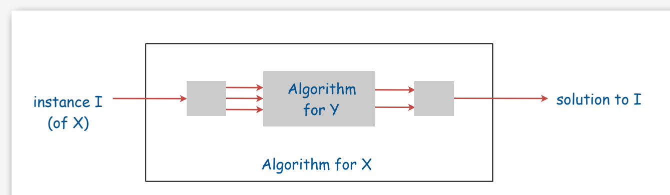
Desiderata. Classify problems according to computational requirements.

Desiderata'.
Suppose we could (couldn't) solve problem X efficiently.
What else could (couldn't) we solve efficiently?

" Give me a lever long enough and a fulcrum on which to place it, and I shall move the world. " — *Archimedes*

Reduction

Def. Problem X reduces to problem Y if you can use an algorithm that solves Y to help solve X.

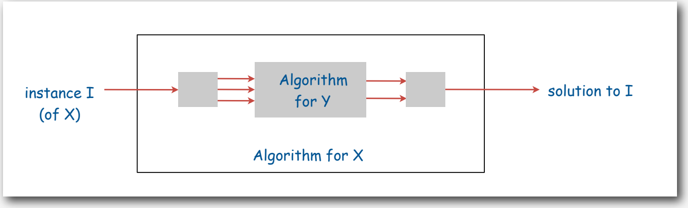


Cost of solving X = total cost of solving Y + cost of reduction.

perhaps many calls to Y
on problems of different sizes

Reduction

Def. Problem X reduces to problem Y if you can use an algorithm that solves Y to help solve X.



Ex 1. [element distinctness reduces to sorting]

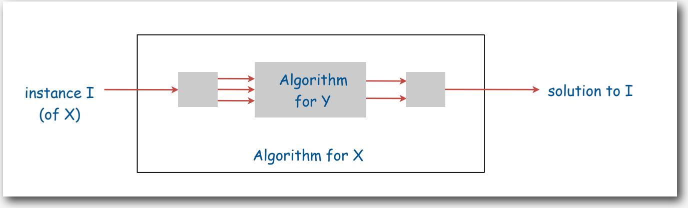
To solve element distinctness on N integers:

- Sort N integers.
- Scan through consecutive pairs and check if any are equal.

Cost of solving element distinctness. N log N + N.

Reduction

Def. Problem X reduces to problem Y if you can use an algorithm that solves Y to help solve X.



Ex 2. [3-collinear reduces to sorting]

To solve 3-collinear instance on N points in the plane:

- For each point, sort other points by polar angle.
  - scan through consecutive triples and check if they are collinear

Cost of solving 3-collinear. $N^2 \log N + N^2$.

- **designing algorithms**
- establishing lower bounds
- establishing intractability
- classifying problems

Reduction: design algorithms

Def. Problem X reduces to problem Y if you can use an algorithm that solves Y to help solve X.

Design algorithm. Given algorithm for Y, can also solve X.

Ex.

- Element distinctness reduces to sorting.
- 3-collinear reduces to sorting.
- PERT reduces to topological sort. [see digraph lecture]
- h-v line intersection reduces to 1d range searching. [see geometry lecture]

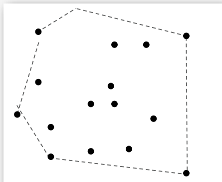
Mentality. Since I know how to solve Y, can I use that algorithm to solve X?

programmer's version: I have code for Y. Can I use it for X?

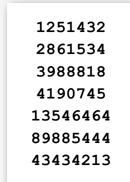
Convex hull reduces to sorting

Sorting. Given N distinct integers, rearrange them in ascending order.

Convex hull. Given N points in the plane, identify the extreme points of the convex hull (in counter-clockwise order).



convex hull

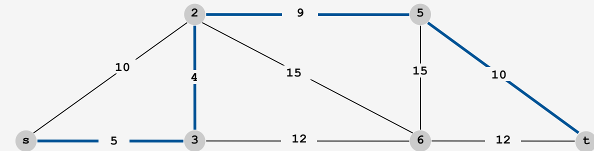


sorting

Proposition. Convex hull reduces to sorting.
Pf. Graham scan algorithm.

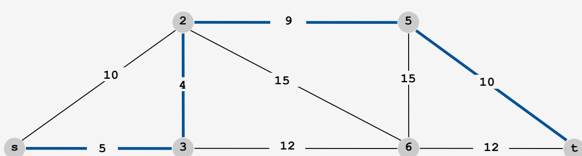
Shortest path on graphs and digraphs

Proposition. Undirected shortest path (with nonnegative weights) reduces to directed shortest path.

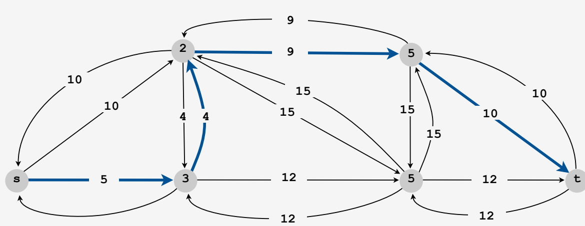


Shortest path on graphs and digraphs

Proposition. Undirected shortest path (with nonnegative weights) reduces to directed shortest path.

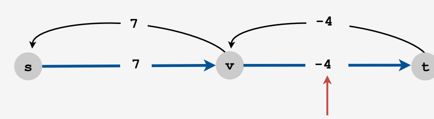


Pf. Replace each undirected edge by two directed edges.



Shortest path with negative weights

Caveat. Reduction is invalid in networks with negative weights (even if no negative cycles).



reduction creates
negative cycles

Remark. Can still solve shortest path problem in undirected graphs (if no negative cycles), but need more sophisticated techniques.

reduces to weighted
non-bipartite matching (!)

Primality testing

PRIME. Given an integer x (represented in binary), is x prime?
COMPOSITE. Given an integer x, does x have a nontrivial factor?

Proposition. PRIME reduces to COMPOSITE.

```
public static boolean isPrime(BigInteger x)
{
   if (isComposite(x)) return false;
   else                return true;
}
```

PRIME

↓

COMPOSITE

147573952589676412931

prime

147573952589676412927

composite

Primality testing

PRIME. Given an integer x (represented in binary), is x prime?
COMPOSITE. Given an integer x, does x have a nontrivial factor?

Proposition. COMPOSITE reduces to PRIME.

```
public static boolean isComposite(BigInteger x)
{
   if (isPrime(x)) return false;
   else            return true;
}
```

PRIME

↓↑

COMPOSITE

147573952589676412931

prime

147573952589676412927

composite

Caveat

PRIME. Given an integer x (represented in binary), is x prime?
COMPOSITE. Given an integer x, does x have a nontrivial factor?

Proposition. COMPOSITE reduces to PRIME.
Proposition. PRIME reduces to COMPOSITE.

A possible real-world scenario.

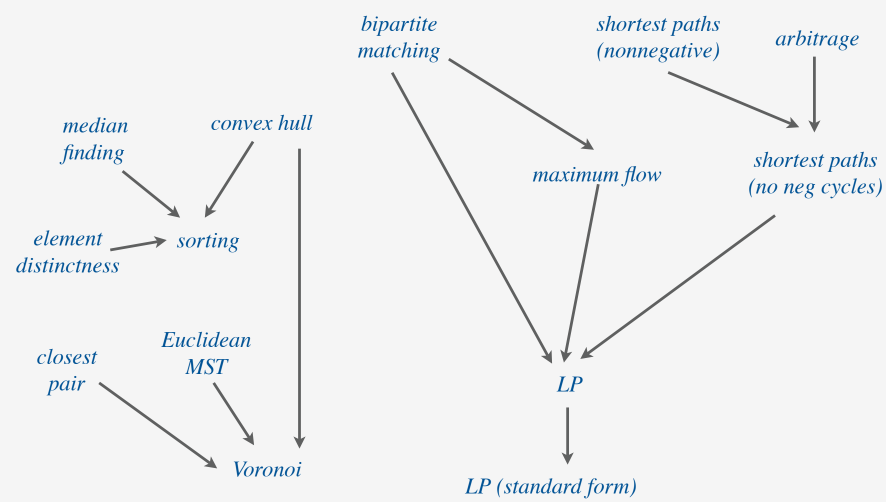
- System designer specs the APIs for project.
- Programmer A implements `isComposite()` using `isPrime()`.
- Programmer B implements `isPrime()` using `isComposite()`.
- Infinite reduction loop! ← whose fault?

PRIME

↓↑

COMPOSITE

Some reductions



- designing algorithms
- **establishing lower bounds**
- establishing intractability
- classifying problems

Bird's-eye view

Goal. Prove that a problem requires a certain number of steps.

Ex. $\Omega(N \log N)$ lower bound for sorting.

1251432
2861534
3988818
4190745
13546464
89885444
43434213

argument must apply to all conceivable algorithms

Bad news. Very difficult to establish lower bounds from scratch.

Good news. Can establish $\Omega(N \log N)$ lower bound for Y by reducing sorting to Y.

assuming cost of reduction is not too large

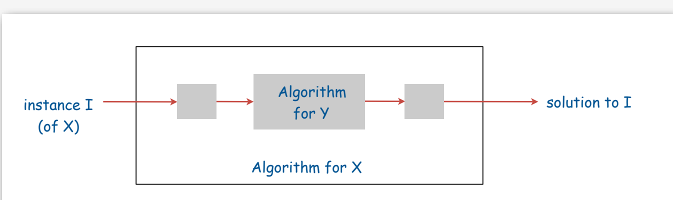
Linear-time reductions

Def. Problem X linear-time reduces to problem Y if X can be solved with:

- linear number of standard computational steps
- one call to Y

Ex. Almost all of the reductions we've seen so far.

Q. Which one was not a linear-time reduction?



Linear-time reductions

Def. Problem X linear-time reduces to problem Y if X can be solved with:

- linear number of standard computational steps
- one call to Y

Establish lower bound:

- If X takes $\Omega(N \log N)$ steps, then so does Y.
- If X takes $\Omega(N^2)$ steps, then so does Y.

Mentality.

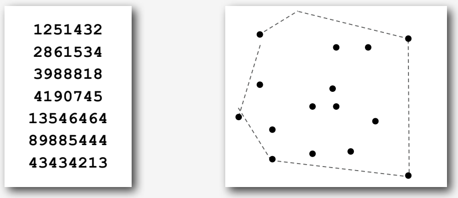
- If I could easily solve Y, then I could easily solve X.
- I can't easily solve X.
- Therefore, I can't easily solve Y.

Lower bound for convex hull

Fact. In quadratic decision tree model, any algorithm for sorting N integers requires $\Omega(N \log N)$ steps.

allows quadratic tests of the form:
$x_i < x_j$ or $(x_j - x_i)(x_k - x_i) - (x_j)(x_j - x_i) < 0$

Proposition. Sorting linear-time reduces to convex hull.
Pf. [see next slide]



sorting *convex hull*

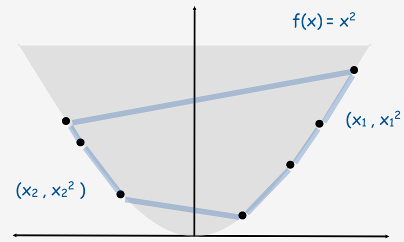
a quadratic test

Implication. Any ccw-based convex hull algorithm requires $\Omega(N \log N)$ ccw's.

Sorting linear-time reduces to convex hull

Proposition. Sorting linear-time reduces to convex hull.

- Sorting instance. $X = \{ x_1, x_2, \ldots, x_N \}$
- Convex hull instance. $P = \{ (x_1, x_1^2), (x_2, x_2^2), \ldots, (x_N, x_N^2) \}$



Pf.

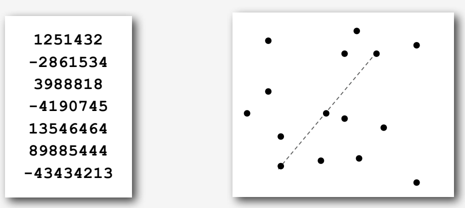
- Region $\{x : x^2 \geq x\}$ is convex $\Rightarrow$ all points are on hull.
- Starting at point with most negative x, counter-clockwise order of hull points yields integers in ascending order.

Lower bound for 3-COLLINEAR

3-SUM. Given N distinct integers, are there three that sum to 0?

3-COLLINEAR. Given N distinct points in the plane, are there 3 that all lie on the same line?

recall Assignment 3



3-sum *3-collinear*

Lower bound for 3-COLLINEAR

3-SUM. Given N distinct integers, are there three that sum to 0?

3-COLLINEAR. Given N distinct points in the plane, are there 3 that all lie on the same line?

Proposition. 3-SUM linear-time reduces to 3-COLLINEAR.
Pf. [see next 2 slide]

in certain restricted model of computation

Fact. Any algorithm for 3-SUM requires $\Omega(N^2)$ time.
Implication. No sub-quadratic algorithm for 3-COLLINEAR.

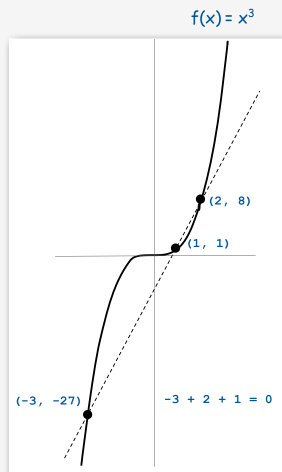
your $N^2 \log N$ algorithm was pretty good

3-SUM linear-time reduces to 3-COLLINEAR

Proposition. 3-SUM linear-time reduces to 3-COLLINEAR.

- 3-SUM instance: $X = \{ x_1, x_2, \ldots, x_N \}$
- 3-COLLINEAR instance: $P = \{ (x_1, x_1^3), (x_2, x_2^3), \ldots, (x_N, x_N^3) \}$

Lemma. If a, b, and c are distinct, then $a + b + c = 0$ if and only if (a, a^3), (b, b^3), (c, c^3) are collinear.



3-SUM linear-time reduces to 3-COLLINEAR

Proposition. 3-SUM linear-time reduces to 3-COLLINEAR.

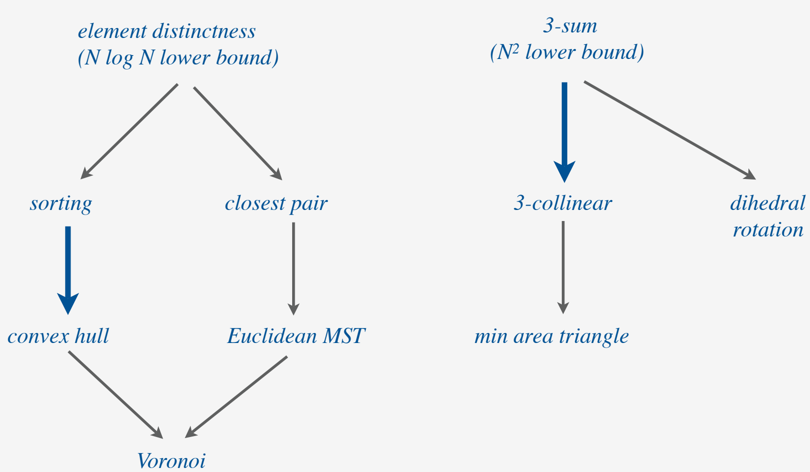
- 3-SUM instance: $X = \{ x_1, x_2, \ldots, x_N \}$
- 3-COLLINEAR instance: $P = \{ (x_1, x_1^3), (x_2, x_2^3), \ldots, (x_N, x_N^3) \}$

Lemma. If a, b, and c are distinct, then $a + b + c = 0$ if and only if (a, a^3), (b, b^3), (c, c^3) are collinear.

Pf. Three points (a, a^3), (b, b^3), (c, c^3) are collinear iff:

$$\begin{aligned}
(a^3 - b^3) / (a - b) &= (b^3 - c^3) / (b - c) && \textit{slopes are equal} \\
(a - b)(a^2 + ab + b^2) / (a - b) &= (b - c)(b^2 + bc + c^2) / (b - c) && \textit{factor numerators} \\
(a^2 + ab + b^2) &= (b^2 + bc + c^2) && \textit{a-b and b-c are nonzero} \\
a^2 + ab - bc - c^2 &= 0 && \textit{collect terms} \\
(a - c)(a + b + c) &= 0 && \textit{factor} \\
a + b + c &= 0 && \textit{a-c is nonzero}
\end{aligned}$$

More lower bounds



Establishing lower bounds: summary

Establishing lower bounds through reduction is an important tool in guiding algorithm design efforts.

Q. How to convince yourself no linear-time convex hull algorithm exists?
Hard way. Long futile search for a linear-time algorithm.
Easy way. Reduction from sorting.

sorting → *convex hull*

Q. How to convince yourself no subquadratic 3-COLLINEAR algorithm exists.
Hard way. Long futile search for a subquadratic algorithm.
Easy way. Reduction from 3-SUM.

3-SUM → *3-COLLINEAR*

- designing algorithms
- establishing lower bounds
- **establishing intractability**
- classifying problems

Bird's-eye view

Desiderata. Prove that a problem can't be solved in poly-time.

EXPTIME-complete.

input size = lg k

- Given a fixed-size program and input, does it halt in at most k steps?
- Given N-by-N checkers board position, can the first player force a win (using forced capture rule)?

Frustrating news. Extremely difficult and few successes.

3-satisfiability

Literal. A boolean variable or its negation. x_i or $\neg x_i$

Clause. An or of 3 distinct literals. $C_j = (x_1 \vee \neg x_2 \vee x_3)$

Conjunctive normal form. An and of clauses. $\Phi = (C_1 \wedge C_2 \wedge C_3 \wedge C_4)$

3-SAT. Given a CNF formula Φ consisting of k clauses over n literals, does it have a satisfying truth assignment?

yes instance

$$(\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee x_4)$$

x_1	x_2	x_3	x_4
T	T	F	T

$$(\neg T \vee T \vee F) \wedge (T \vee \neg T \vee F) \wedge (\neg T \vee \neg T \vee \neg F) \wedge (\neg T \vee \neg T \vee T) \wedge (\neg T \vee F \vee T)$$

no instance

$$(\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4) \wedge (\neg x_2 \vee x_3 \vee x_4)$$

Applications. Circuit design, program correctness, ...

3-satisfiability is intractable

Q. How to solve an instance of 3-SAT with n variables?
A. Exhaustive search: try all 2^n truth assignments.

Q. Can we do anything substantially more clever?

"intractable"

Conjecture (P ≠ NP). No poly-time algorithm for 3-SAT.

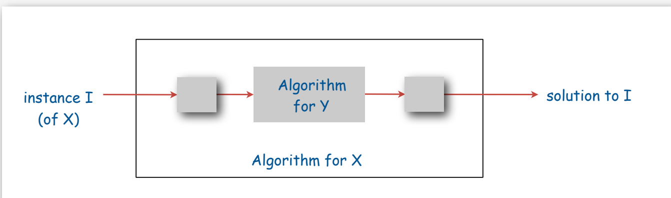
Good news. Can prove problems "intractable" via reduction from 3-SAT.

Polynomial-time reductions

Def. Problem X poly-time reduces to problem Y if X can be solved with:

- Polynomial number of standard computational steps.
- One call to Y.

Ex. All reductions we've seen.



Polynomial-time reductions

Def. Problem X poly-time reduces to problem Y if X can be solved with:

- Polynomial number of standard computational steps.
- One call to Y.

Establish intractability. If 3-SAT poly-time reduces to Y, then Y is intractable.

Mentality.

- If I could solve Y in poly-time, then I could also solve 3-SAT.
- I can't solve 3-SAT.
- Therefore, I can't solve Y.

Integer linear programming

ILP. Minimize a linear objective function, subject to linear inequalities, and integer variables.

Proposition. 3-SAT poly-time reduces to ILP.

Pf. [by example]

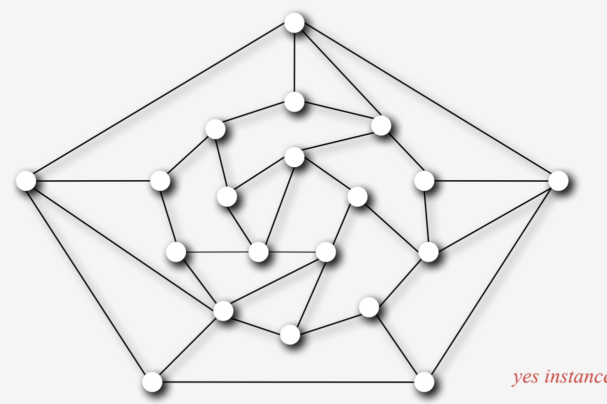
$$(\neg x1 \vee x2 \vee x3) \wedge (x1 \vee \neg x2 \vee x3) \wedge (\neg x1 \vee \neg x2 \vee \neg x3) \wedge (\neg x1 \vee \neg x2 \vee x4) \wedge (\neg x2 \vee x3 \vee x4)$$

minimize	$C1 + C2 + C3 + C4 + C5$	CNF formula satisfiable iff min = 5
subject to the constraints	$(1 - x1) \le C1$	
	$x2 \le C1$	C1 = 1 iff clause 1 is satisfied
	$x3 \le C1$	
	...	add 3 inequalities for each clause
	all xi and Cj = { 0, 1 }	

Interpretation. Boolean variable xi is true iff integer variable xi = 1.

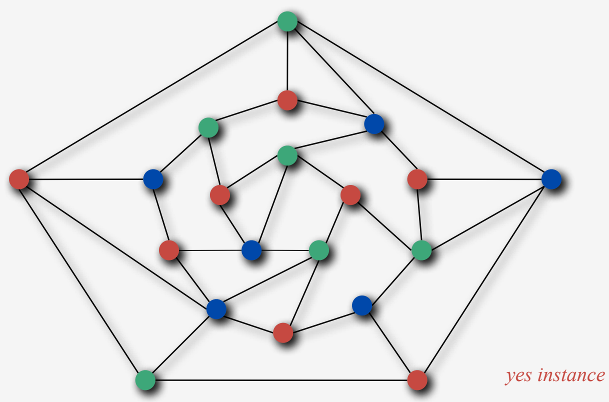
Graph 3-colorability

3-COLOR. Given a graph, is there a way to color the vertices red, green, and blue so that no adjacent vertices have the same color?



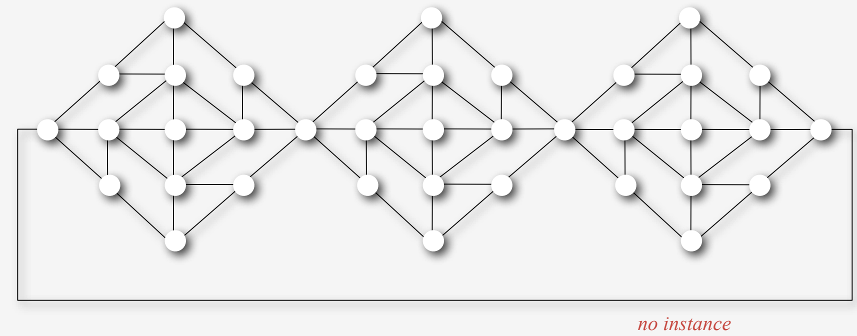
Graph 3-colorability

3-COLOR. Given a graph, is there a way to color the vertices red, green, and blue so that no adjacent vertices have the same color?



Graph 3-colorability

3-COLOR. Given a graph, is there a way to color the vertices red, green, and blue so that no adjacent vertices have the same color?



Applications. Register allocation, Potts model in physics, ...

3-satisfiability reduces to graph 3-colorability

Proposition. 3-SAT poly-time reduces to 3-COLOR.

Pf. Given 3-SAT instance Φ, we construct an instance G of 3-COLOR that is 3-colorable if and only if Φ is satisfiable.

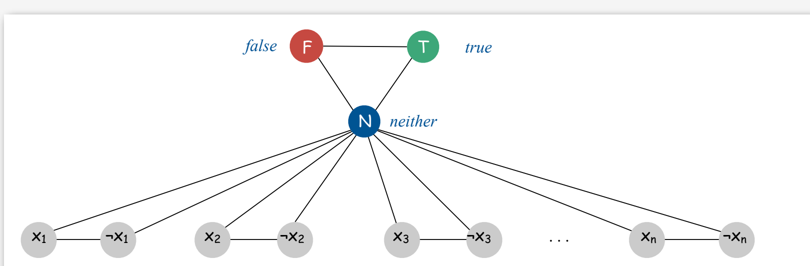
Construction.

(i) Create one vertex for each literal and 3 vertices F, T, and N.

(ii) Connect F, T, and N in a triangle and connect each literal to N.

(iii) Connect each literal to its negation.

(iv) For each clause, attach a 6-vertex gadget [details to follow].

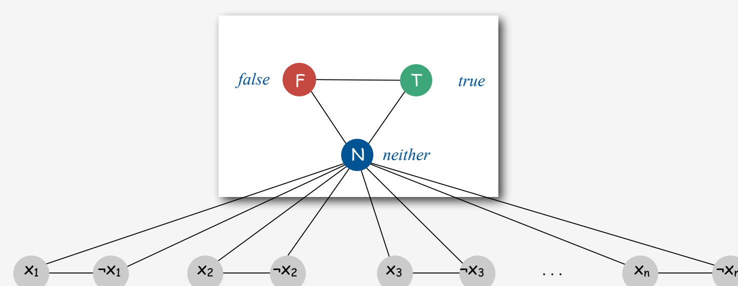


3-satisfiability reduces to graph 3-colorability

Claim. If graph G is 3-colorable then Φ is satisfiable.

Pf.

- Consider assignment where F corresponds to false and T to true.
- (ii) [triangle] ensures each literal is true or false.

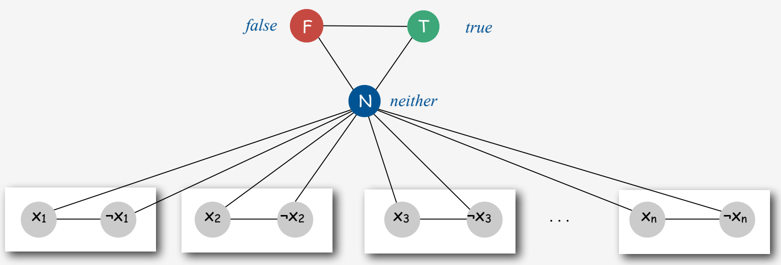


3-satisfiability reduces to graph 3-colorability

Claim. If graph G is 3-colorable then Φ is satisfiable.

Pf.

- Consider assignment where F corresponds to false and T to true.
- (ii) [triangle] ensures each literal is true or false.
- (iii) ensures a literal and its negation are opposites.



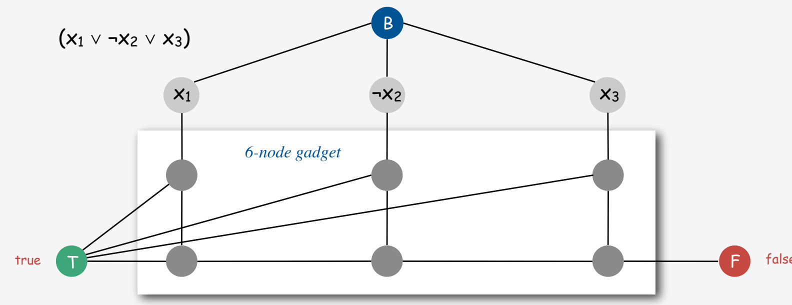
3-satisfiability reduces to graph 3-colorability

Claim. If graph G is 3-colorable then Φ is satisfiable.

Pf.

- Consider assignment where F corresponds to false and T to true.
- (ii) [triangle] ensures each literal is true or false.
- (iii) ensures a literal and its negation are opposites.
- (iv) [gadget] ensures at least one literal in each clause is true.

next slide



3-satisfiability reduces to graph 3-colorability

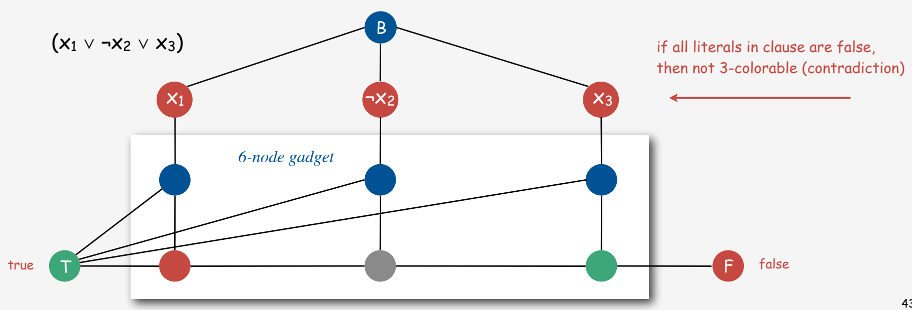
Claim. If graph G is 3-colorable then Φ is satisfiable.

Pf.

- Consider assignment where F corresponds to false and T to true.
- (ii) [triangle] ensures each literal is true or false.
- (iii) ensures a literal and its negation are opposites.
- (iv) [gadget] ensures at least one literal in each clause is true.

next slide

Therefore, Φ is satisfiable. ▪



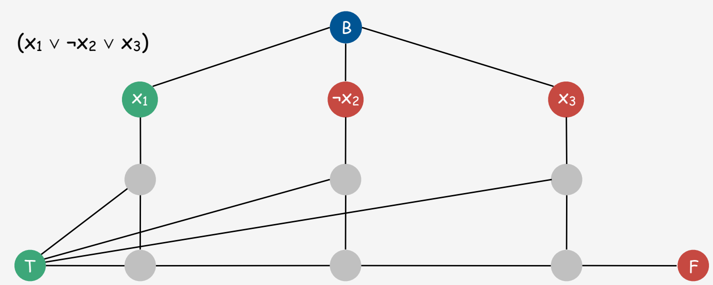
3-satisfiability reduces to graph 3-colorability

Claim. If Φ is satisfiable then graph G is 3-colorable.

at least one in each clause

Pf.

- Color nodes corresponding to false literals and to true literals .



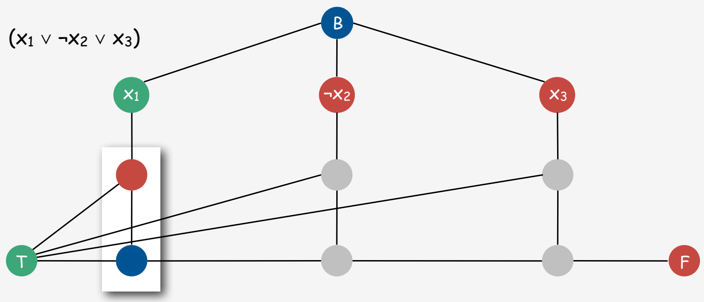
3-satisfiability reduces to graph 3-colorability

Claim. If Φ is satisfiable then graph G is 3-colorable.

Pf.

- Color nodes corresponding to false literals and to true literals.
- Color vertex below one vertex, and vertex below that.

$(x_1 \vee \neg x_2 \vee x_3)$



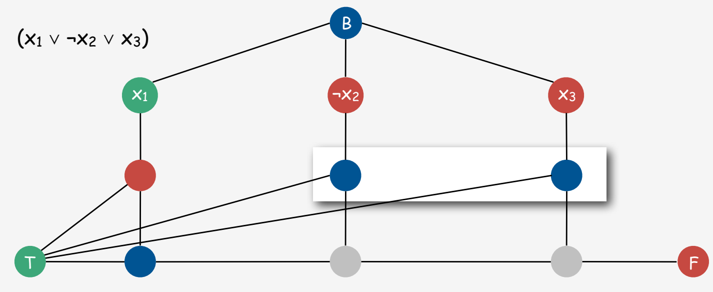
3-satisfiability reduces to graph 3-colorability

Claim. If Φ is satisfiable then graph G is 3-colorable.

Pf.

- Color nodes corresponding to false literals and to true literals.
- Color vertex below one vertex, and vertex below that.
- Color remaining middle row vertices.

$(x_1 \vee \neg x_2 \vee x_3)$



3-satisfiability reduces to graph 3-colorability

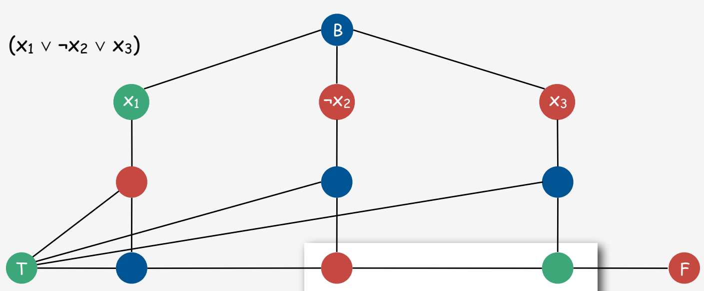
Claim. If Φ is satisfiable then graph G is 3-colorable.

Pf.

- Color nodes corresponding to false literals and to true literals.
- Color vertex below one vertex, and vertex below that.
- Color remaining middle row vertices.
- Color remaining bottom vertices or as forced.

Works for all gadgets, so graph is 3-colorable. ▪

$(x_1 \vee \neg x_2 \vee x_3)$



3-satisfiability reduces to graph 3-colorability

Proposition. 3-SAT poly-time reduces to 3-COLOR.

Pf. Given 3-SAT instance Φ, we construct an instance G of 3-COLOR that is 3-colorable if and only if Φ is satisfiable.

Construction.

(i) Create one vertex for each literal and 3 vertices F, T, and N.

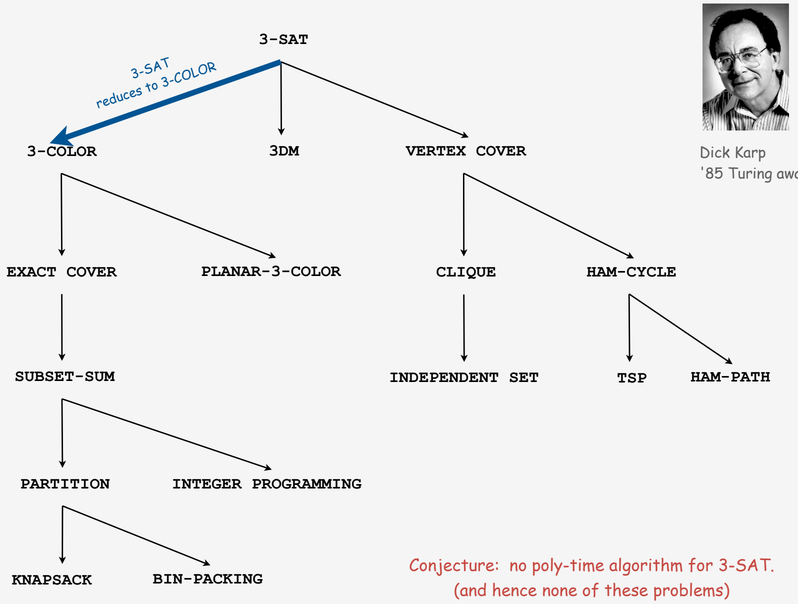
(ii) Connect F, T, and N in a triangle and connect each literal to N.

(iii) Connect each literal to its negation.

(iv) For each clause, attach a 6-vertex gadget.

Consequence. 3-COLOR is intractable.

More poly-time reductions from 3-satisfiability



Dick Karp
'85 Turing award

Conjecture: no poly-time algorithm for 3-SAT.
(and hence none of these problems)

Establishing intractability: summary

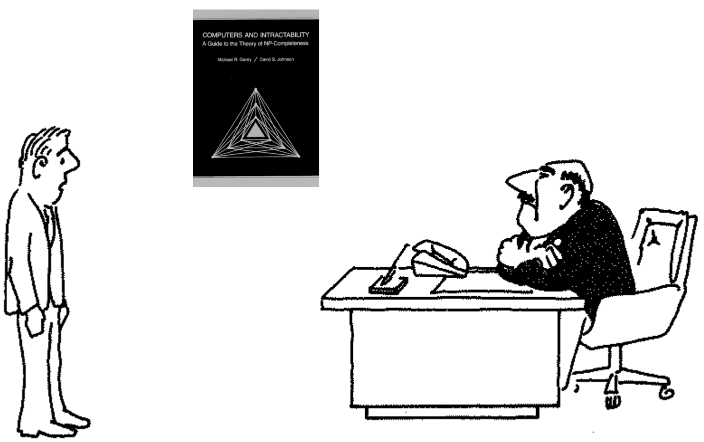
Establishing intractability through poly-time reduction is an important tool in guiding algorithm design efforts.

Q. How to convince yourself that a new problem is intractable?
Hard way. Long futile search for an efficient algorithm (as for 3-SAT).
Easy way. Reduction from a know intractable problem (such as 3-SAT).

hence, intricate reductions are common

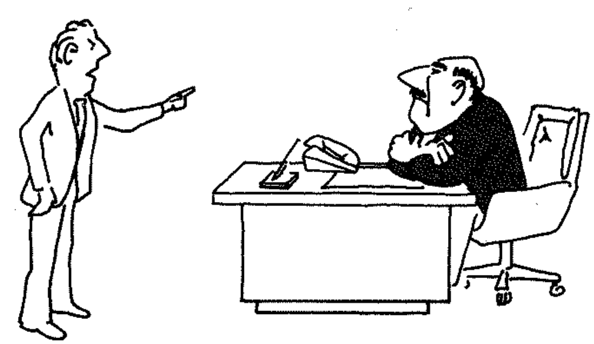
3-SAT → *3-COLOR*

Implications of poly-time reductions



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

Implications of poly-time reductions



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

Implications of poly-time reductions

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

- designing algorithms
- establishing lower bounds
- establishing intractability
- **classifying problems**

Classify problems

Desiderata. Classify problems according to difficulty.
- Linear: can be solved in linear time.
- Linearithmic: can be solved in linearithmic time.
- Quadratic: can be solved in quadratic time.

…

- Tractable: can be solved in poly-time.
- Intractable: seem to require exponential time.

Ex. Sorting and convex hull are in same complexity class.
- Sorting linear-time reduces to convex hull.
- Convex hull linear-time reduces to sorting.

linearithmic

Classify problems

Desiderata. Classify problems according to difficulty.
- Linear: can be solved in linear time.
- Linearithmic: can be solved in linearithmic time.
- Quadratic: can be solved in quadratic time.

…

- Tractable: can be solved in poly-time.
- Intractable: seem to require exponential time.

Ex. PRIME and COMPOSITE are in same complexity class.
- PRIME linear-time reduces to COMPOSITE.
- COMPOSITE linear-time reduces to PRIME.

tractable, but
nobody knows which class

Classify problems

Desiderata. Classify problems according to difficulty.

- Linear: can be solved in linear time.
- Linearithmic: can be solved in linearithmic time.
- Quadratic: can be solved in quadratic time.

…

- Tractable: can be solved in poly-time.
- Intractable: seem to require exponential time.

Ex. 3-SAT and 3-COLOR are in the same complexity class.

- 3-SAT poly-time reduces to 3-COLOR.
- 3-COLOR poly-time reduces to 3-SAT.

probably intractable

Cook's theorem (stay tuned)

Cook's theorem

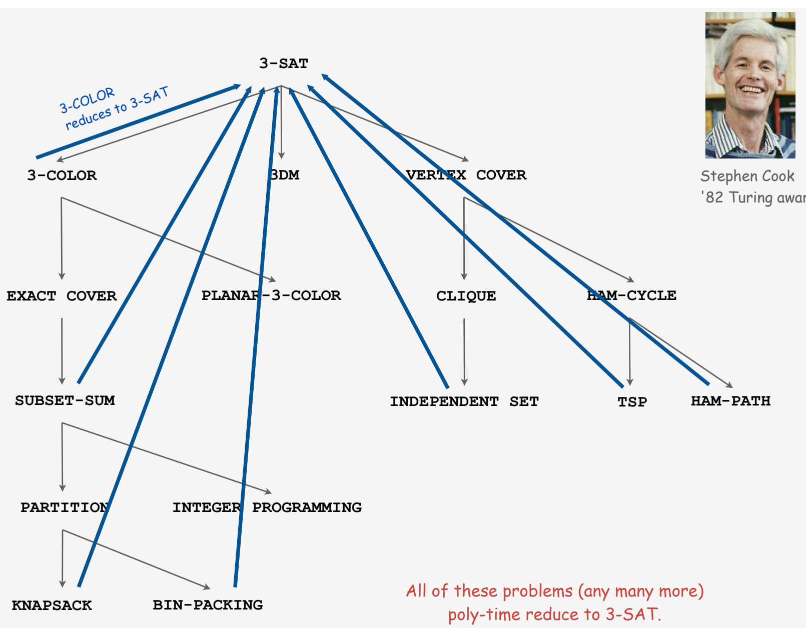
P. Set of problems solvable in poly-time.
Importance. What scientists and engineers can compute feasibly.

NP. Set of problems checkable in poly-time.
Importance. What scientists and engineers aspire to compute feasibly.

Cook's theorem. All problems in NP poly-time reduces to 3-SAT.

"NP-complete"

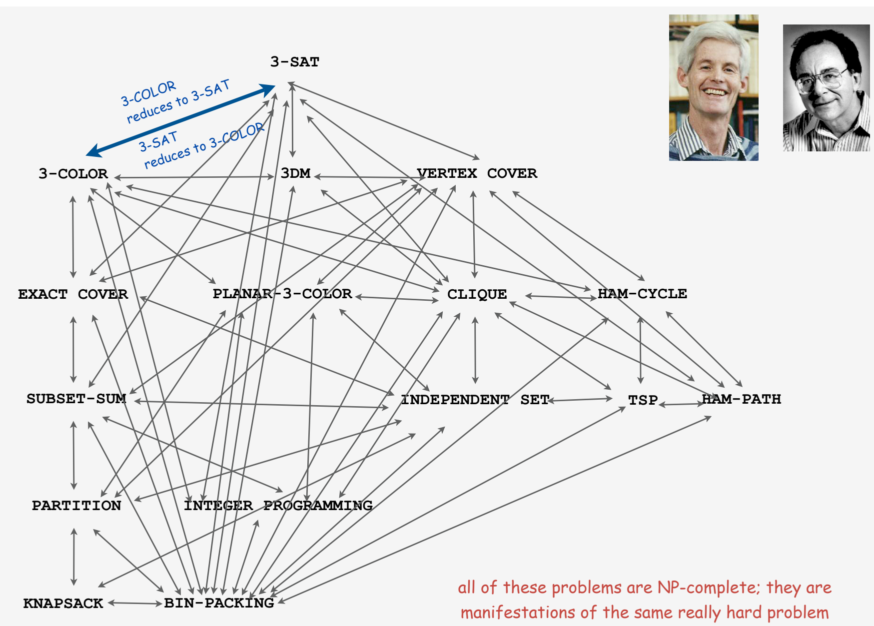
Implications of Cook's theorem



Stephen Cook
'82 Turing award

All of these problems (any many more) poly-time reduce to 3-SAT.

Implications of Karp + Cook



all of these problems are NP-complete; they are manifestations of the same really hard problem

Summary

Reductions are important in theory to:

- Establish tractability.
- Establish intractability.
- Classify problems according to their computational requirements.

Reductions are important in practice to:

- Design algorithms.
- Design reusable software modules.
  - stack, queue, sorting, priority queue, symbol table, set,
  - graph, shortest path, regular expression
- Determine difficulty of your problem and choose the right tool.
  - use exact algorithm for tractable problems
  - use heuristics for intractable problems