Reductions

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

Bird's-eye view

Desiderata. Classify problems according to computational requirements.

- Linear: min/max, median, BWT, smallest enclosing circle, ...
- Linearithmic: sorting, convex hull, closest pair, furthest pair, ...
- Quadratic: ???
- Cubic: ???
- ...
- Exponential: ???

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

Bird's-eye view

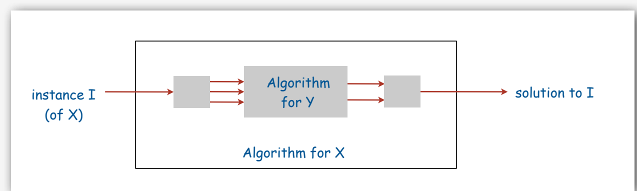
Desiderata. Classify problems according to computational requirements.

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.



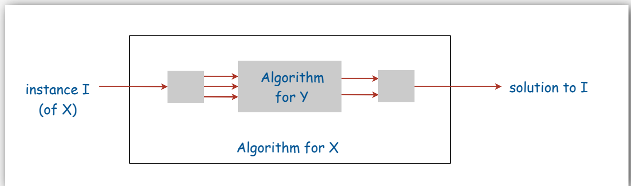
Cost of solving X = total cost of solving Y + cost of reduction.

perhaps many calls to Y on problems of different sizes

preprocessing and postprocessing

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]

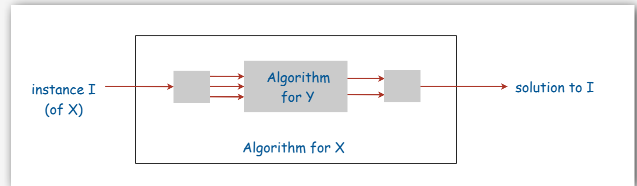
To solve element distinctness on N integers:

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

Cost of solving element distinctness. $\underbrace{N \log N}_{\text{cost of sorting}} + \underbrace{N}_{\text{cost of reduction}}$

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 adjacent triples and check if they are collinear

Cost of solving 3-collinear. $\underbrace{N^2 \log N}_{\text{cost of sorting}} + \underbrace{N^2}_{\text{cost of reduction}}$.

- 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]
- Euclidean MST reduces to Delaunay triangulation. [see geometry lecture]

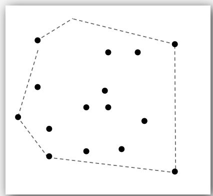
Mentality. Since I know how to solve Y, can I use that algorithm to solve X?

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

```
1251432
2861534
3988818
4190745
13546464
89885444
43434213
34435312
```

sorting

Proposition. Convex hull reduces to sorting.

Pf. Graham scan algorithm.

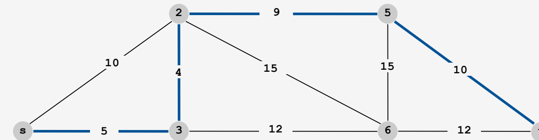
Cost of convex hull. N log N + N.

cost of sorting

cost of reduction

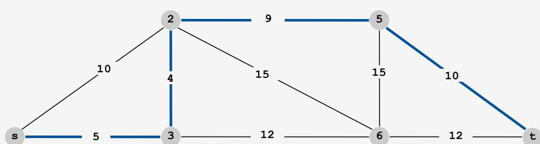
Shortest path on graphs and digraphs

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

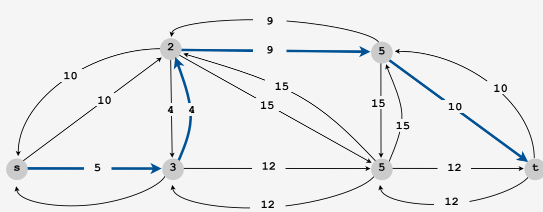


Shortest path on graphs and digraphs

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

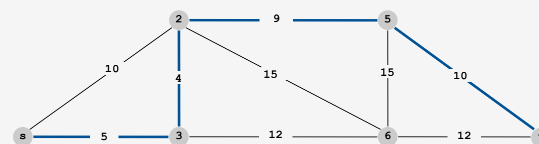


Pf. Replace each undirected edge by two directed edges.



Shortest path on graphs and digraphs

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



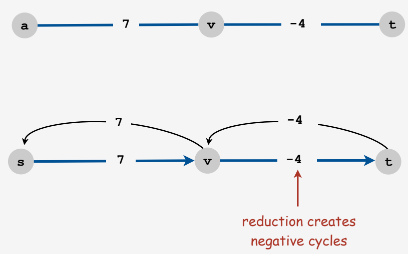
Cost of undirected shortest path. E log V + E.

cost of shortest path in digraph

cost of reduction

Shortest path with negative weights

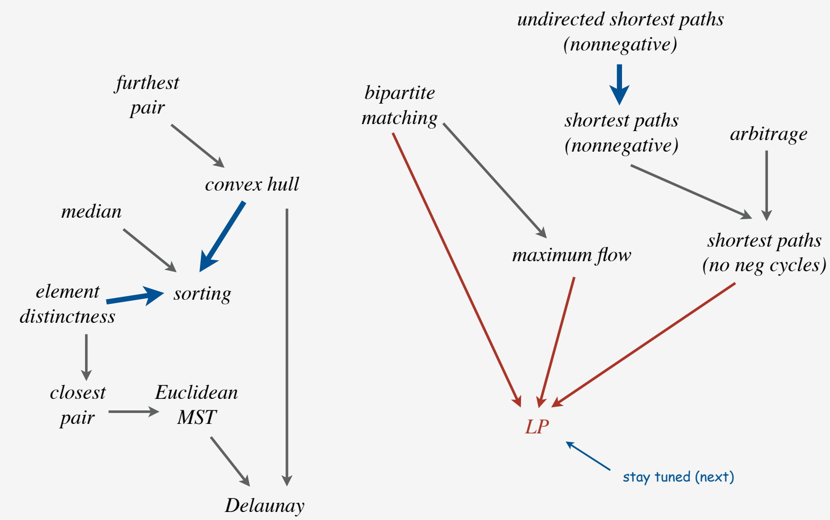
Caveat. Reduction is invalid in networks with negative weights (even if no negative cycles).



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 (!)

Some reductions involving familiar problems



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

Linear Programming

What is it? see ORF 307

- Quintessential tool for optimal allocation of scarce resources
- Powerful and general problem-solving method

Why is it significant?

- Widely applicable.
- Dominates world of industry.
- Fast commercial solvers available: CPLEX, OSL.
- Powerful modeling languages available: AMPL, GAMS.
- Ranked among most important scientific advances of 20th century.

Ex: Delta claims that LP saves $100 million per year.

Present context: Many important problems reduce to LP

Applications

Agriculture. Diet problem.
Computer science. Compiler register allocation, data mining.
Electrical engineering. VLSI design, optimal clocking.
Energy. Blending petroleum products.
Economics. Equilibrium theory, two-person zero-sum games.
Environment. Water quality management.
Finance. Portfolio optimization.
Logistics. Supply-chain management.
Management. Hotel yield management.
Marketing. Direct mail advertising.
Manufacturing. Production line balancing, cutting stock.
Medicine. Radioactive seed placement in cancer treatment.
Operations research. Airline crew assignment, vehicle routing.
Physics. Ground states of 3-D Ising spin glasses.
Plasma physics. Optimal stellarator design.
Telecommunication. Network design, Internet routing.
Sports. Scheduling ACC basketball, handicapping horse races.

Linear programming

Model problem as maximizing an objective function subject to constraints

Input: real numbers a_{ij} c_j, b_i.

Output: real numbers x_j.

n variables

maximize $c_1 x_1 + c_2 x_2 + \ldots + c_n x_n$

subject to the constraints (m equations)

$$a_{11} x_1 + a_{12} x_2 + \ldots + a_{1n} x_n \le b_1$$
$$a_{21} x_1 + a_{22} x_2 + \ldots + a_{2n} x_n \le b_2$$
$$\ldots$$
$$a_{m1} x_1 + a_{m2} x_2 + \ldots + a_{mn} x_n \le b_m$$
$$x_1, x_2, \ldots, x_n \ge 0$$

matrix version

maximize $c^T x$

subject to the constraints $A x \le b$, $x \ge 0$

Solutions (see ORF 307)
- Simplex algorithm has been used for decades to solve practical LP instances
- Newer algorithms guarantee fast solution

Linear programming

"Linear programming"
- process of formulating an LP model for a problem
- solution to LP for a specific problem gives solution to the problem
- equivalent to "reducing the problem to LP"

1. Identify variables
2. Define constraints (inequalities and equations)
3. Define objective function

Examples:
- shortest paths
- maxflow
- bipartite matching

  . . .
- [a very long list]

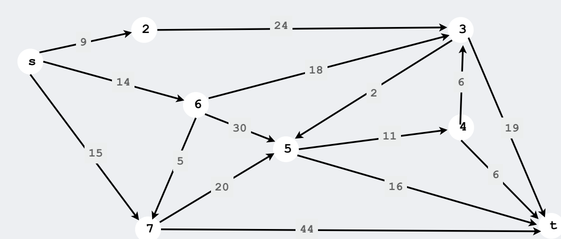
stay tuned (next)

Single-source shortest-paths problem (revisited)

Given. Weighted digraph, single source s.

Distance from s to v: length of the shortest path from s to v .

Goal. Find distance (and shortest path) from s to every other vertex.

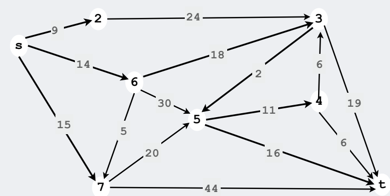


Single-source shortest-paths problem reduces to LP

One variable per vertex, one inequality per edge.

maximize	x_t
subject to the constraints	$x_s + 9 \geq x_2$
	$x_s + 14 \geq x_6$
	$x_s + 15 \geq x_7$
	$x_2 + 24 \geq x_3$
	$x_3 + 2 \geq x_5$
	$x_3 + 19 \geq x_t$
	$x_4 + 6 \geq x_3$
	$x_4 + 6 \geq x_t$
	$x_5 + 11 \geq x_4$
	$x_5 + 16 \geq x_t$
	$x_6 + 18 \geq x_3$
	$x_6 + 30 \geq x_5$
	$x_6 + 5 \geq x_7$
	$x_7 + 20 \geq x_5$
	$x_7 + 44 \geq x_t$
	$x_s = 0$

interpretation:
x_i = length of shortest path from source to i

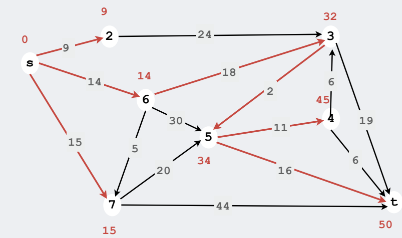


Single-source shortest-paths problem reduces to LP

One variable per vertex, one inequality per edge.

maximize	x_t
subject to the constraints	$x_s + 9 \geq x_2$
	$x_s + 14 \geq x_6$
	$x_s + 15 \geq x_7$
	$x_2 + 24 \geq x_3$
	$x_3 + 2 \geq x_5$
	$x_3 + 19 \geq x_t$
	$x_4 + 6 \geq x_3$
	$x_4 + 6 \geq x_t$
	$x_5 + 11 \geq x_4$
	$x_5 + 16 \geq x_t$
	$x_6 + 18 \geq x_3$
	$x_6 + 30 \geq x_5$
	$x_6 + 5 \geq x_7$
	$x_7 + 20 \geq x_5$
	$x_7 + 44 \geq x_t$
	$x_s = 0$

interpretation:
x_i = length of shortest path from source to i



solution

$x_s = 0$
$x_2 = 9$
$x_3 = 32$
$x_4 = 45$
$x_5 = 34$
$x_6 = 14$
$x_7 = 15$
$x_t = 50$

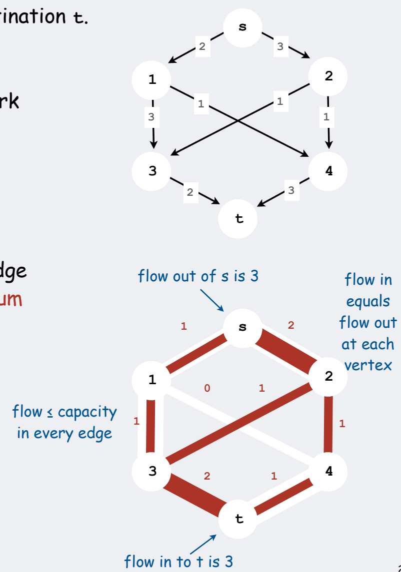
Maxflow problem

Given: Weighted digraph, source s, destination t.

Interpret edge weights as capacities
- Models material flowing through network
- Ex: oil flowing through pipes
- Ex: goods in trucks on roads
- [many other examples]

Flow: A different set of edge weights
- flow does not exceed capacity in any edge
- flow at every vertex satisfies equilibrium
  [flow in equals flow out]

Goal: Find maximum flow from s to t

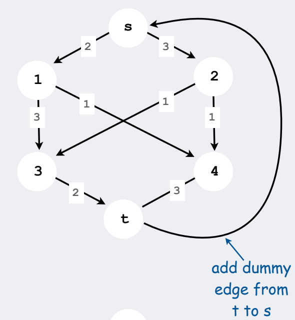


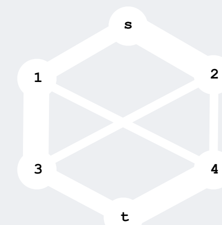
Maxflow problem reduces to LP

One variable per edge.
One inequality per edge, one equality per vertex.

maximize	$x_{3t} + x_{4t}$	
subject to the constraints	$x_{s1} \leq 2$	capacity constraints
	$x_{s2} \leq 3$	
	$x_{13} \leq 3$	
	$x_{14} \leq 1$	
	$x_{23} \leq 1$	
	$x_{24} \leq 1$	
	$x_{3t} \leq 2$	
	$x_{4t} \leq 3$	
equilibrium constraints	$x_{s1} = x_{13} + x_{14}$	
	$x_{s2} = x_{23} + x_{24}$	
	$x_{13} + x_{23} = x_{3t}$	
	$x_{14} + x_{24} = x_{4t}$	
	all $x_{ij} \geq 0$	

interpretation:
x_{ij} = flow in edge i-j





Maxflow problem reduces to LP

One variable per edge.
One inequality per edge, one equality per vertex.

maximize $x_{3t} + x_{4t}$

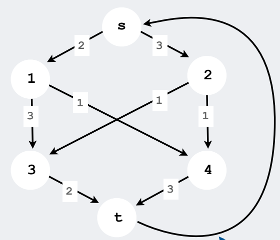
subject to the constraints

capacity constraints:
$x_{s1} \le 2$
$x_{s2} \le 3$
$x_{13} \le 3$
$x_{14} \le 1$
$x_{23} \le 1$
$x_{24} \le 1$
$x_{3t} \le 2$
$x_{4t} \le 3$

equilibrium constraints:
$x_{s1} = x_{13} + x_{14}$
$x_{s2} = x_{23} + x_{24}$
$x_{13} + x_{23} = x_{3t}$
$x_{14} + x_{24} = x_{4t}$

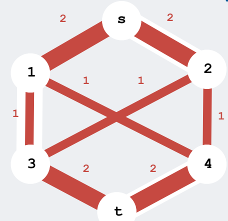
all $x_{ij} \ge 0$

interpretation: x_{ij} = flow in edge i-j



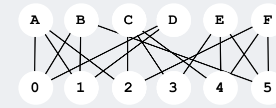
add dummy edge from t to s

solution
$x_{s1} = 2$
$x_{s2} = 2$
$x_{13} = 1$
$x_{14} = 1$
$x_{23} = 1$
$x_{24} = 1$
$x_{3t} = 2$
$x_{4t} = 2$



Maximum cardinality bipartite matching problem

Given: Two sets of vertices, set of edges
(each connecting one vertex in each set)



Matching: set of edges
with no vertex appearing twice

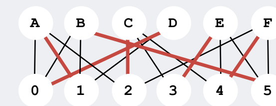
Interpretation: mutual preference constraints
- Ex: people to jobs
- Ex: medical students to residence positions
- Ex: students to writing seminars
- [many other examples]

Alice
Adobe, Apple, Google
Bob
Adobe, Apple, Yahoo
Carol
Google, IBM, Sun
Dave
Adobe, Apple
Eliza
IBM, Sun, Yahoo
Frank
Google, Sun, Yahoo

Adobe
Alice, Bob, Dave
Apple
Alice, Bob, Dave
Google
Alice, Carol, Frank
IBM
Carol, Eliza
Sun
Carol, Eliza, Frank
Yahoo
Bob, Eliza, Frank

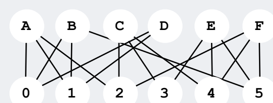
Example: Job offers

Goal: find a maximum cardinality matching



Maximum cardinality bipartite matching problem reduces to LP

One variable per edge, one equality per vertex.



maximize $x_{A0} + x_{A1} + x_{A2} + x_{B0} + x_{B1} + x_{B5} + x_{C2} + x_{C3} + x_{C4} + x_{D0} + x_{D1} + x_{E3} + x_{E4} + x_{E5} + x_{F2} + x_{F4} + x_{F5}$

subject to the constraints

constraints on top vertices:
$x_{A0} + x_{A1} + x_{A2} = 1$
$x_{B0} + x_{B1} + x_{B5} = 1$
$x_{C2} + x_{C3} + x_{C4} = 1$
$x_{D0} + x_{D1} = 1$
$x_{E3} + x_{E4} + x_{E5} = 1$
$x_{F2} + x_{F4} + x_{F5} = 1$

constraints on bottom vertices:
$x_{A0} + x_{B0} + x_{D0} = 1$
$x_{A1} + x_{B1} + x_{D1} = 1$
$x_{A2} + x_{C2} + x_{F2} = 1$
$x_{C3} + x_{E3} = 1$
$x_{C4} + x_{E4} + x_{F4} = 1$
$x_{B5} + x_{E5} + x_{F5} = 1$

all $x_{ij} \ge 0$

interpretation: An edge is in the matching iff $x_{ij} = 1$

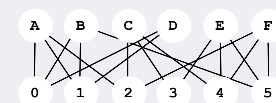
Crucial point: not always so lucky!

Theorem. [Birkhoff 1946, von Neumann 1953]
All extreme points of the above polyhedron have integer (0 or 1) coordinates

Corollary. Can solve bipartite matching problem by solving LP

Maximum cardinality bipartite matching problem reduces to LP

One variable per edge, one equality per vertex.



maximize $x_{A0} + x_{A1} + x_{A2} + x_{B0} + x_{B1} + x_{B5} + x_{C2} + x_{C3} + x_{C4} + x_{D0} + x_{D1} + x_{E3} + x_{E4} + x_{E5} + x_{F2} + x_{F4} + x_{F5}$

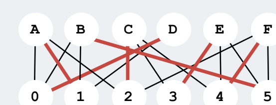
subject to the constraints

$x_{A0} + x_{A1} + x_{A2} = 1$
$x_{B0} + x_{B1} + x_{B5} = 1$
$x_{C2} + x_{C3} + x_{C4} = 1$
$x_{D0} + x_{D1} = 1$
$x_{E3} + x_{E4} + x_{E5} = 1$
$x_{F2} + x_{F4} + x_{F5} = 1$
$x_{A0} + x_{B0} + x_{D0} = 1$
$x_{A1} + x_{B1} + x_{D1} = 1$
$x_{A2} + x_{C2} + x_{F2} = 1$
$x_{C3} + x_{E3} = 1$
$x_{C4} + x_{E4} + x_{F4} = 1$
$x_{B5} + x_{E5} + x_{F5} = 1$

all $x_{ij} \ge 0$

interpretation: An edge is in the matching iff $x_{ij} = 1$

solution
$x_{A1} = 1$
$x_{B5} = 1$
$x_{C2} = 1$
$x_{D0} = 1$
$x_{E3} = 1$
$x_{F4} = 1$
all other $x_{ij} = 0$



Linear programming perspective

Got an optimization problem?
ex: shortest paths, maxflow, matching, . . . [many, many, more]

Approach 1: Use a specialized algorithm to solve it
- Algs in Java
- vast literature on complexity
- performance on real problems not always well-understood

Approach 2: Reduce to a linear programming model, use a commercial solver
- a direct mathematical representation of the problem often works
- immediate solution to the problem at hand is often available
- might miss faster specialized solution, but might not care

Got an LP solver? Learn to use it!

```
% ampl
AMPL Version 20010215 (SunOS 5.7)
ampl: model maxflow.mod;
ampl: data maxflow.dat;
ampl: solve;
CPLEX 7.1.0: optimal solution;
objective 4;
```


- designing algorithms
- linear programming
- **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 spread $\Omega(N \log N)$ lower bound to Y by reducing sorting to Y.

assuming cost of reduction is not too high

Linear-time reductions

Def. Problem X linear-time reduces to problem Y if X can be solved with:
- Linear number of standard computational steps.
- Constant number of calls to Y.

Ex. Almost all of the reductions we've seen so far.
Q. Which one was not a linear-time reduction?

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

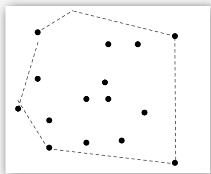
Proposition. In quadratic decision tree model, any algorithm for sorting N integers requires $\Omega(N \log N)$ steps.

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]

```
1251432
2861534
3988818
4190745
13546464
89885444
43434213
```

sorting



convex hull

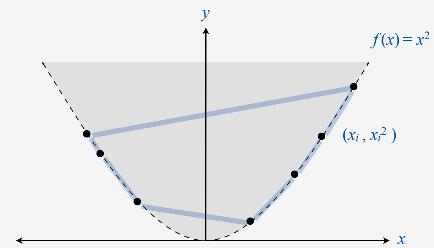
a quadratic test

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

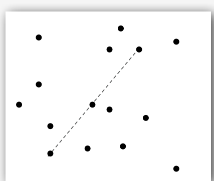
3-SUM. Given N distinct integers, are there three that sum to 0?

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

recall Assignment 3

```
1251432
-2861534
3988818
-4190745
13546464
89885444
-43434213
```

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]

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

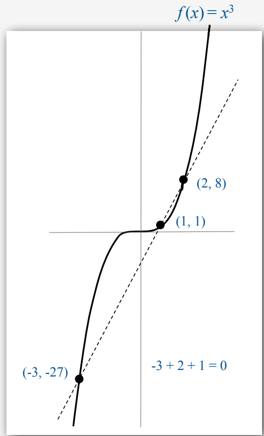
your $N^2 \log N$ algorithm was pretty good

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.

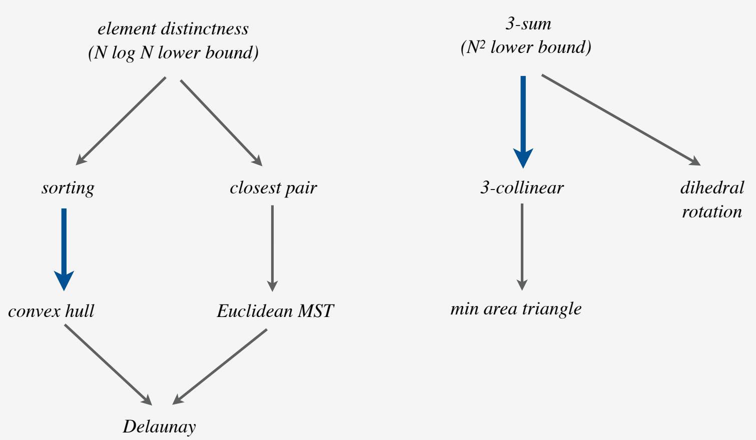
- 3-SUM instance: $X = \{ x_1, x_2, \ldots, x_N \}$
- 3-COLLINEAR instance: $P = \{ (x_1, x_1^3), (x_2, x_2^3), \ldots, (x_N, x_N^3) \}$

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 reductions and 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?
A. [hard way] Long futile search for a linear-time algorithm.
A. [easy way] Reduction from sorting.

Q. How to convince yourself no sub-quadratic 3-COLLINEAR algorithm exists.
A. [hard way] Long futile search for a sub-quadratic algorithm.
A. [easy way] Reduction from 3-SUM.

- designing algorithms
- linear programming
- 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 constant-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. $C_1 = (\neg x_1 \vee x_2 \vee x_3)$

Conjunctive normal form. An *and* of clauses. $\Phi = (C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5)$

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

yes instance

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

$$\begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \\ T & T & F & T \end{array} \quad (\neg T \vee T \vee F) \wedge (T \vee \neg T \vee F) \wedge (\neg T \vee \neg T \vee \neg F) \wedge (\neg T \vee \neg T \vee T) \wedge (\neg T \vee F \vee T)$$

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?

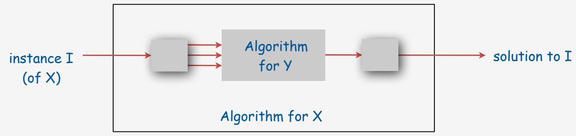
Conjecture (P ≠ NP). No poly-time algorithm for 3-SAT. "intractable"

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

Polynomial-time reductions

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

- Polynomial number of standard computational steps.
- Polynomial number of calls to Y.



Establish tractability. If Y can be solved in poly-time, and X poly-time reduces to Y, then X can be solved in poly-time.

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.

Example: 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]

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

maximize	$C_1 + C_2 + C_3 + C_4 + C_5$
subject to the constraints	$C_1 \le (1 - x_1) + x_2 + x_3$
	$C_2 \le x_1 + (1 - x_2) + x_3$
	$C_3 \le (1 - x_1) + (1 - x_2) + (1 - x_1)$
	$C_4 \le (1 - x_1) + (1 - x_2) + x_4$
	$C_5 \le (1 - x_2) + x_3 + x_4$
	all x_i and $C_j = \{0, 1\}$

CNF formula is satisfiable iff max = 5

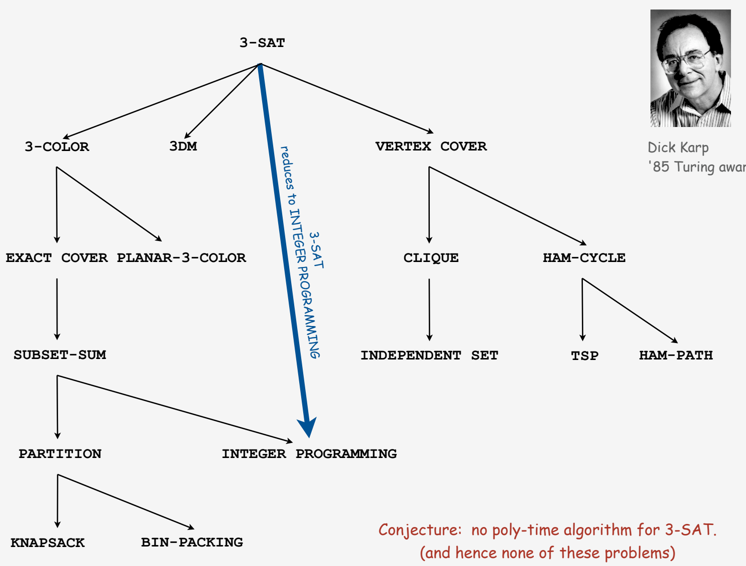
C_1 = 1 iff clause 1 is satisfied (either x_1 = 0, x_2 = 1, or x_3 = 1)

same argument for each clause

boolean variable x_i is true iff integer variable x_i = 1

Therefore, ILP 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.

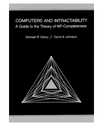
Q. How to convince yourself that a new problem is intractable?

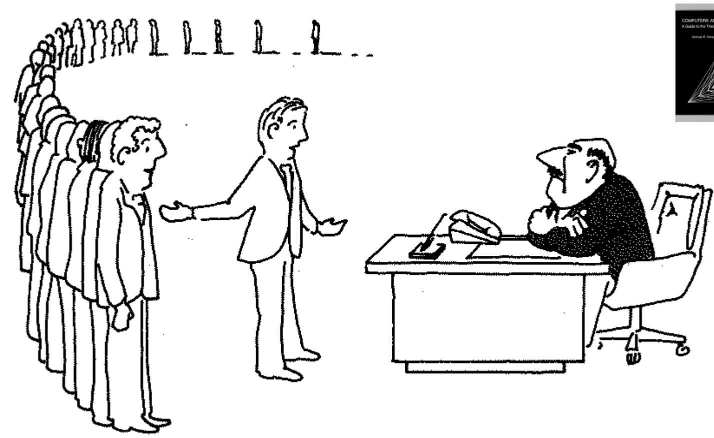
A. [hard way] Long futile search for an efficient algorithm (as for 3-SAT).

A. [easy way] Reduction from a know intractable problem (such as 3-SAT).

Caveat. Intricate reductions are common.

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.

  …
- 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.
- Moreover, we have N log N upper and lower bound.

Cook's theorem

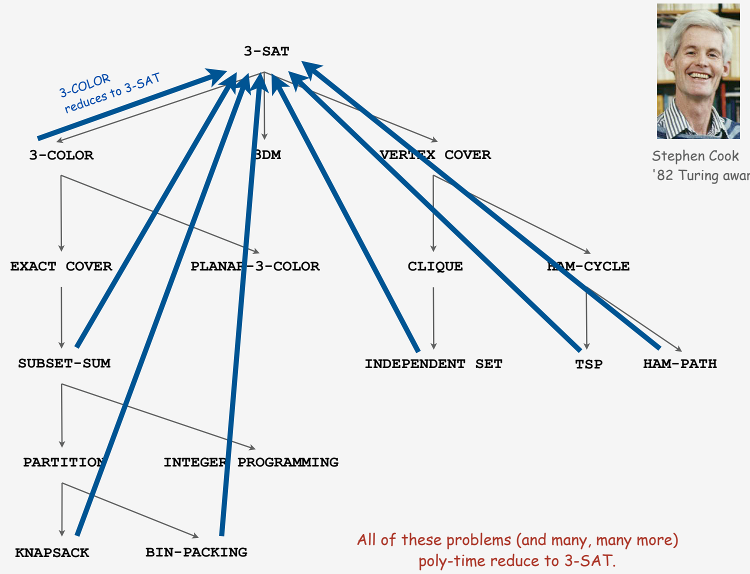
P. Set of problems solvable in poly-time.
Importance. What scientists and engineers can compute feasibly.

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

"NP-complete"

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

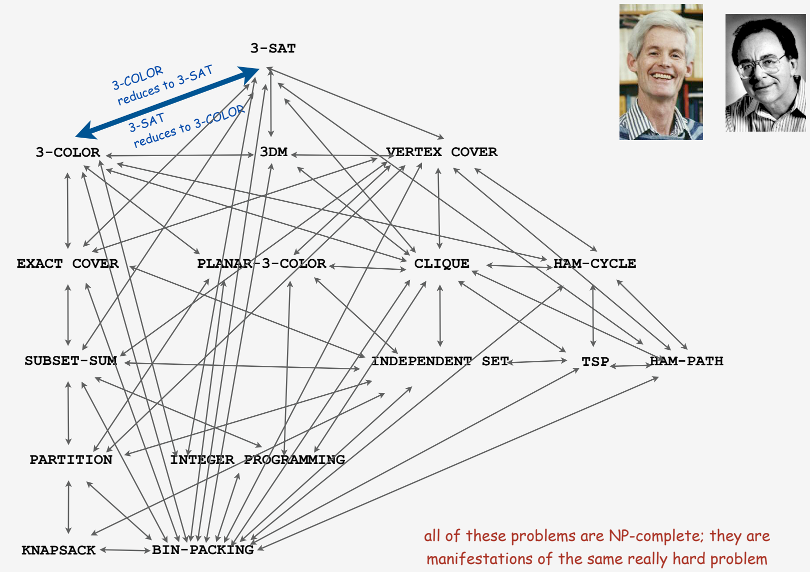
Implications of Cook's theorem



Stephen Cook
'82 Turing award

All of these problems (and many, 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, Delaunay triangulation
  - Voronoi, max flow, LP
- Determine difficulty of your problem and choose the right tool.
  - use exact algorithm for tractable problems
  - use heuristics for intractable problems