Reductions

- designing algorithms
- proving limits
- classifying problems
- polynomial-time reductions
- NP-completeness

Desiderata

Desiderata. Classify problems according to their 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

Desiderata

Desiderata. Classify problems according to their computational requirements.

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

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 = cost of solving Y + cost of reduction.

Ex. Euclidean MST reduces to Voronoi.
To solve Euclidean MST on N points
- solve Voronoi
- construct graph with linear number of edges
- use Prim/Kruskal to find MST in time proportional to N log N
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 = cost of solving Y + cost of reduction.

**Consequences.**
- Algorithm design: given algorithm for Y, can also solve X.
- Establish intractability: if X is hard, then so is Y.
- Classify problems: establish relative difficulty between two problems.

Linear-time reductions

**Def.** Problem X linear reduces to problem Y if X can be solved with:
- Linear number of standard computational steps for reduction
- One call to subroutine for Y.
- Notation: \( X \preceq_L Y \).

**Some familiar examples.**
- Median \( \preceq_L \) sorting.
- Element distinctness \( \preceq_L \) sorting.
- Closest pair \( \preceq_L \) Voronoi.
- Euclidean MST \( \preceq_L \) Voronoi.
- Arbitrage \( \preceq_L \) Negative cycle detection.
- Linear programming \( \preceq_L \) Linear programming in std form.

Linear-time reductions for algorithm design

**Def.** Problem X linear reduces to problem Y if X can be solved with:
- Linear number of standard computational steps for reduction
- One call to subroutine for Y.

**Applications.**
- Designing algorithms: given algorithm for Y, can also solve X.
- Proving limits: if X is hard, then so is Y.
- Classifying problems: establish relative difficulty of problems.

Mentality: Since I know how to solve Y, can I use that algorithm to solve X?
Convex Hull

**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).

**Claim.** Convex hull linear reduces to sorting.

**Pf.** Graham scan algorithm.

Shortest Paths on Graphs and Digraphs

**Claim.** Undirected shortest path (with nonnegative weights) linearly reduces to directed shortest path.

**Pf.** Replace each undirected edge by two directed edges.

Shortest Paths with negative weights

**Caveat.** Reduction 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.

designing algorithms
proving limits
classifying problems
poly-time reductions
NP-completeness
Linear-time reductions to prove limits

Def. Problem X linear reduces to problem Y if X can be solved with:
- linear number of standard computational steps for reduction
- one call to subroutine for Y.

Applications.
- designing algorithms: given algorithm for Y, can also solve X.
- proving limits: if X is hard, then so is Y.
- classifying problems: establish relative difficulty of problems.

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.

Proving limits on convex-hull algorithms

Lower bound on sorting: Sorting N integers requires $\Omega(N \log N)$ steps.

Claim. Sorting linear-reduces to convex hull [see next slide].

Theorem. Any ccw-based convex hull algorithm requires $\Omega(N \log N)$ steps.

Sorting linear-reduces to convex hull

Sorting instance. $x_1, x_2, ..., x_N$
Convex hull instance. $(x_1, x_1^2), (x_2, x_2^2), ..., (x_N, x_N^2)$

Observation. Region $\{x : x^2 > x\}$ is convex $\Rightarrow$ all points are on hull.

Consequence. Starting at point with most negative $x$, counter-clockwise order of hull points yields items in ascending order.

3-SUM Reduces to 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?

Claim. 3-SUM $\leq_L$ 3-COLLINEAR.

Conjecture. Any algorithm for 3-SUM requires $\Omega(N^2)$ time.

Corollary. Sub-quadratic algorithm for 3-COLLINEAR unlikely.

Recall Assignment 2
Your $N^2 \log N$ algorithm from Assignment 2 was pretty good.
3-SUM Reduces to 3-COLLINEAR

**Claim.** 3-SUM \(\leq_L\) 3-COLLINEAR.
- 3-SUM instance: \(x_1, x_2, ..., x_N\)
- 3-COLLINEAR instance: \((x_1, x_1^3), (x_2, x_2^3), ..., (x_N, x_N^3)\)

\[f(x) = x^3\]

3-SUM Reduces to 3-COLLINEAR

**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:

\[
\begin{align*}
(a^3 - b^3) / (a - b) &= (b^3 - c^3) / (b - c) \\
(a - b)(a^2 + ab + b^2) / (a - b) &= (b - c)(b^2 + bc + c^2) / (b - c) \\
(a^2 + ab + b^2) &= (b^2 + bc + c^2) \\
a^2 + ab - bc - c^2 &= 0 \\
(a - c)(a + b + c) &= 0 \\
a + b + c &= 0
\end{align*}
\]

slopes are equal
factor numerators
a-b and b-c are nonzero
collect terms
factor
a-c is nonzero

Linear Time Reductions

**Def.** Problem X **linear reduces** to problem Y if X can be solved with:
- Linear number of standard computational steps.
- One call to subroutine for Y.

**Consequences.**
- Design algorithms: given algorithm for Y, can also solve X.
- Establish intractability: if X is hard, then so is Y.
- Classify problems: establish relative difficulty between two problems.
Primality and Compositeness

PRIME. Given an integer x (represented in binary), is x prime?

COMPOSITE. Given an integer x, does x have a nontrivial factor?

Claim. PRIME \not\leq_{L} COMPOSITE.

\begin{verbatim}
public static boolean isPrime(BigInteger x)
{
    if (isComposite(x)) return false;
    else                return true;
}
\end{verbatim}

Reduction Gone Wrong

Caveat.

- System designer specs the interfaces for project.
- One programmer might implement isComposite() using isPrime().
- Other programmer might implement isPrime() using isComposite().
- Be careful to avoid infinite reduction loops in practice.

\begin{verbatim}
public static boolean isComposite(BigInteger x)
{
    if (isPrime(x)) return false;
    else            return true;
}
\end{verbatim}

Poly-Time Reduction

Def. Problem X polynomial reduces to problem Y if arbitrary instances of problem X can be solved using:

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

Notation. X \leq_{p} Y.

Ex. Assignment problem \leq_{p} LP
Ex. 3-SAT \leq_{p} 3-COLOR.
Ex. Any linear reduction.

Conclusion. COMPOSITE and PRIME have same complexity.
**Poly-time reductions**

**Goal.** Classify and separate problems according to relative difficulty.
- Those that can be solved in polynomial time.
- Those that seem to require exponential time.

**Establish tractability.** If \( X \preceq_p Y \) and \( Y \) can be solved in poly-time, then \( X \) can be solved in poly-time.

**Establish intractability.** If \( Y \preceq_p X \) and \( Y \) cannot be solved in poly-time, then \( X \) cannot be solved in poly-time.

**Transitivity.** If \( X \preceq_p Y \) and \( Y \preceq_p Z \) then \( X \preceq_p Z \).

---

**Assignment problem reduces to LP**

**Assignment problem.** Assign \( n \) jobs to \( n \) machines to minimize total cost, where \( c_{ij} \) = cost of assigning job \( j \) to machine \( i \).

Maximize

\[
\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}
\]

subject to the constraints

\[
\begin{align*}
\sum_{j=1}^{n} x_{ij} &= 1, \quad i = 1, \ldots, n \\
\sum_{i=1}^{n} x_{ij} &= 1, \quad j = 1, \ldots, n
\end{align*}
\]

Interpretation: if \( x_{ij} = 1 \), then assign job \( j \) to machine \( i \)

**Theorem.** [Birkhoff 1946, von Neumann 1953] All extreme points of the above polyhedron are \((0,1)-valued\).

**Corollary.** Can solve assignment problem by solving LP since LP algorithms return an optimal solution that is an extreme point.

---

**Assignment Problem**

**Applications.** Match jobs to machines, match personnel to tasks, match Princeton students to writing seminars.

---

**Assignment problem reduces to LP**

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 \\
1 & 3 & 8 & 9 & 15 & 10 \\
2 & 4 & 10 & 7 & 16 & 14 \\
3 & 9 & 13 & 11 & 19 & 10 \\
4 & 8 & 13 & 12 & 20 & 13 \\
5 & 1 & 7 & 5 & 11 & 9 \\
\end{array}
\]

\[
\text{cost} = 3 \times 10 + 11 + 20 + 9 = 53
\]

\[
\begin{array}{cccccc}
1' & 2' & 3' & 4' & 5' \\
1' & 3 & 8 & 9 & 15 & 10 \\
2' & 4 & 10 & 7 & 16 & 14 \\
3' & 9 & 13 & 11 & 19 & 10 \\
4' & 8 & 13 & 12 & 20 & 13 \\
5' & 1 & 7 & 5 & 11 & 9 \\
\end{array}
\]

\[
\text{cost} = 8 \times 7 + 20 + 8 + 11 = 44
\]
3-Satisfiability

Literal: A Boolean variable or its negation. \( x_i \) or \( \neg x_i \)

Clause. A disjunction of 3 distinct literals. \( C_j = (x_1 \lor \neg x_2 \lor x_3) \)

Conjunctive normal form. A propositional formula \( \Phi \) that is the conjunction of clauses. \( \text{CNF} = (C_1 \land C_2 \land C_3 \land C_4) \)

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

Ex:

\[
(\neg x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2 \lor x_3) \land (\neg x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_2 \lor x_4) \land (\neg x_2 \lor x_3 \lor x_4)
\]

solution

\[
\begin{array}{cccc}
 x_1 & x_2 & x_3 & x_4 \\
 T & T & F & T \\
\end{array}
\]

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?

Claim. 3-SAT \( \leq_p \) 3-COLOR.

Pf. Given 3-SAT instance \( \Phi \), we construct an instance of 3-COLOR that is 3-colorable iff \( \Phi \) is satisfiable.

Construction.

(i) Create one vertex for each literal.
(ii) Create 3 new vertices \( T, F, \) and \( B \); connect them in a triangle, and connect each literal to \( B \).
(iii) Connect each literal to its negation.
(iv) For each clause, attach a gadget of 6 vertices and 13 edges.

Key application. Electronic design automation (EDA).
Claim. Graph is 3-colorable iff % is satisfiable.

Pf. Suppose graph is 3-colorable.
* Consider assignment that sets all T literals to true.
* (ii) [triangle] ensures each literal is T or F.
* (iii) ensures a literal and its negation are opposites.
* (iv) [gadget] ensures at least one literal in each clause is T.

Therefore, % is satisfiable.
**Claim.** Graph is 3-colorable iff \( \Phi \) is satisfiable.

**Pf.** Suppose 3-SAT formula \( \Phi \) is satisfiable.  
- Color all true literals \( T \) and false literals \( F \).  
- Color vertex below one green vertex \( F \), and vertex below that \( B \).  
- Color remaining middle row vertices \( B \).  
- Color remaining bottom vertices \( T \) or \( F \) as forced.

Therefore, graph is 3-colorable.

**Claim.** 3-SAT \( \leq_p \) 3-COLOR.

**Pf.** Given 3-SAT instance \( \Phi \), we construct an instance of 3-COLOR that is 3-colorable iff \( \Phi \) is satisfiable.

**Construction.**  
(i) Create one vertex for each literal.  
(ii) Create 3 new vertices \( T \), \( F \), and \( B \); connect them in a triangle, and connect each literal to \( B \).  
(iii) Connect each literal to its negation.  
(iv) For each clause, attach a gadget of 6 vertices and 13 edges

**Conjecture:** No polynomial-time algorithm for 3-SAT

**Implication:** No polynomial-time algorithm for 3-COLOR.

**Note:** Construction is not intended for use, just for proof.

---

**More Poly-Time Reductions**

3-SAT \( \leq_p \) 3-COLOR  
3DM \( \leq_p \) VERTEX COVER  
HAM-CYCLE \( \leq_p \) CLIQUE  
INDEPENDENT SET \( \leq_p \) 3-COLOR  
PLANAR-3-COLOR \( \leq_p \) EXACT COVER  
HAM-PATH \( \leq_p \) SUBSET-SUM  
PARTITION \( \leq_p \) INTEGER PROGRAMMING  
KNAPSACK \( \leq_p \) BIN-PACKING

Conjecture: no poly-time algorithm for 3-SAT. (and hence none of these problems)
**Cook's Theorem**

NP: set of problems solvable in polynomial time by a nondeterministic Turing machine

**THM.** Any problem in NP $\leq_p$ 3-SAT.

**Pf sketch.**

Each problem $P$ in NP corresponds to a TM $M$ that accepts or rejects any input in time polynomial in its size.

Given $M$ and a problem instance $I$, construct an instance of 3-SAT that is satisfiable iff the machine accepts $I$.

**Construction.**

- Variables for every tape cell, head position, and state at every step.
- Clauses corresponding to each transition.
- [many details omitted]

**Implications of Cook's theorem**

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

**Implications of Karp + Cook**

All of these problems poly-reduce to one another!

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

3-COLOR reduces to 3-SAT

All of these problems poly-reduce to one another!

**Poly-Time Reductions: Implications**

“I can’t find an efficient algorithm, I guess I’m just too dumb.”
Poly-Time Reductions: Implications

“‘I can’t find an efficient algorithm, because no such algorithm is possible!’”

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 expressions, linear programming
- Determine difficulty of your problem and choose the right tool.
  - use exact algorithm for tractable problems
  - use heuristics for intractable problems

“‘I can’t find an efficient algorithm, but neither can all these famous people.’”