7.5 Reductions

- designing algorithms
- establishing lower bounds
- intractability

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

Frustrating news. Huge number of problems have defied classification.

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.
- Check adjacent pairs for equality.

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

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.
- Check adjacent triples for collinearity.

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

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]
- Burrows-Wheeler transform reduces to suffix sort. [see assignment 8]

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).

**Proposition.** Convex hull reduces to sorting.

**Pf.** Graham scan algorithm.

**Cost of convex hull.** $N \log N + N$.

**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.

**Cost of undirected shortest path.** $E \log E + E$. 
**Shortest path with negative weights**

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

![Graph with negative edges](image)

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

**Some reductions involving familiar problems**

- Directed shortest paths (nonnegative)
- Bipartite matching (nonnegative)
- Convex hull
- Sorting
- Median
- Distinctness
- Element
- Farthest pair 2d
- Euclidean MST 2d
- Delaunay triangulation
- Linear programming
- Arbitrage
- Undirected shortest paths (nonnegative)
- Maximum flow
- Shortest paths (no negative cycles)
- Arbitrage
- Directed shortest paths (nonnegative)
- Convex hull
- Sorting
- Median
- Distinctness
- Element
- Farthest pair 2d
- Euclidean MST 2d
- Delaunay triangulation
- Linear programming
- Arbitrage
- Undirected shortest paths (nonnegative)
- Maximum flow
- Shortest paths (no negative cycles)
- Arbitrage
- Directed shortest paths (nonnegative)
- Convex hull
- Sorting
- Median
- Distinctness
- Element
- Farthest pair 2d
- Euclidean MST 2d
- Delaunay triangulation
- Linear programming
- Arbitrage
- Undirected shortest paths (nonnegative)
- Maximum flow
- Shortest paths (no negative cycles)

**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.

- **Present context.** Many important problems reduce to LP.

---

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

"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
- Maximum flow.
- Bipartite matching.
- [a very long list]
Single-source shortest-paths problem reduces to LP

**LP formulation.**
- One variable per vertex, one inequality per edge.
- Interpretation: \( x_i \) = length of shortest path from \( s \) to \( i \).

![Graph showing shortest paths](image1)

Maxflow problem

**Given:** Weighted digraph, source \( s \), destination \( t \).

- Interpret edge weights as capacities
- \( \text{Ex: oil flowing through pipes} \)
- \( \text{Ex: goods in trucks on roads} \)
- [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 \( t \).

![Graph showing max flow](image2)

Maximum flow reduces to LP

**One variable per edge, one inequality per edge.**
- Interpretation: \( x_{ij} \) = flow in edge \( i-j \)

![Graph showing max flow](image3)
Maxflow problem reduces to LP

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

LP formulation.
- One variable per edge, one equality per vertex.
- Interpretation: an edge is in matching iff \( x_{ij} = 1 \).

\[
\begin{align*}
\text{maximize} & \quad x_{A1} + x_{A2} + x_{A3} + x_{B0} + x_{B1} + x_{B5} + x_{C2} + x_{C3} + x_{C4} + x_{C5} + x_{D0} + x_{D1} + x_{E3} + x_{E4} + x_{E5} + x_{F2} + x_{F4} + x_{F5} \\
\text{subject to the constraints} & \quad x_{A2} + x_{A3} + x_{A4} + x_{B0} + x_{B1} + x_{B5} + x_{C2} + x_{C3} + x_{C4} + x_{C5} + x_{D0} + x_{D1} + x_{E3} + x_{E4} + x_{E5} + x_{F2} + x_{F4} + x_{F5} + x_{A1} + x_{A2} + x_{A3} + x_{A4} + x_{A5} + x_{B0} + x_{B1} + x_{B2} + x_{B3} + x_{B4} + x_{B5} + x_{C2} + x_{C3} + x_{C4} + x_{C5} + x_{D0} + x_{D1} + x_{E3} + x_{E4} + x_{E5} + x_{F2} + x_{F4} + x_{F5} = 1 \\
& \quad x_{A2} + x_{A3} + x_{A4} + x_{B0} + x_{B1} + x_{B5} + x_{C2} + x_{C3} + x_{C4} + x_{C5} + x_{D0} + x_{D1} + x_{E3} + x_{E4} + x_{E5} + x_{F2} + x_{F4} + x_{F5} + x_{A1} + x_{A2} + x_{A3} + x_{A4} + x_{A5} + x_{B0} + x_{B1} + x_{B2} + x_{B3} + x_{B4} + x_{B5} + x_{C2} + x_{C3} + x_{C4} + x_{C5} + x_{D0} + x_{D1} + x_{E3} + x_{E4} + x_{E5} + x_{F2} + x_{F4} + x_{F5} = 1 \\
& \quad \vdots \\
& \quad x_{A2} + x_{A3} + x_{A4} + x_{B0} + x_{B1} + x_{B5} + x_{C2} + x_{C3} + x_{C4} + x_{C5} + x_{D0} + x_{D1} + x_{E3} + x_{E4} + x_{E5} + x_{F2} + x_{F4} + x_{F5} + x_{A1} + x_{A2} + x_{A3} + x_{A4} + x_{A5} + x_{B0} + x_{B1} + x_{B2} + x_{B3} + x_{B4} + x_{B5} + x_{C2} + x_{C3} + x_{C4} + x_{C5} + x_{D0} + x_{D1} + x_{E3} + x_{E4} + x_{E5} + x_{F2} + x_{F4} + x_{F5} = 1 \\
& \quad x_{A2} + x_{A3} + x_{A4} + x_{B0} + x_{B1} + x_{B5} + x_{C2} + x_{C3} + x_{C4} + x_{C5} + x_{D0} + x_{D1} + x_{E3} + x_{E4} + x_{E5} + x_{F2} + x_{F4} + x_{F5} = 1 \\
& \quad \vdots \\
& \quad \text{all } x_{ij} \geq 0
\end{align*}
\]

Theorem. [Birkhoff 1946, von Neumann 1953]
All extreme points of the above polyhedron have integer \((0, 1)\) coordinates.
Corollary. Can solve bipartite matching problem by solving LP.

Maximum cardinality bipartite matching reduces to LP

Bipartite graph. Two sets of vertices; edges connect vertices in one set to the other.

Matching. Set of edges with no vertex appearing twice.

Goal. Find a maximum cardinality matching.

Interpretation. Mutual preference constraints.
- Ex: people to jobs.
- Ex: medical students to residency positions.
- Ex: students to writing seminars.
- [many other examples]
Linear programming perspective

Got an optimization problem?

Ex. Shortest paths, maximum flow, matching, ....

Approach 1. Use a specialized algorithm to solve it.
- Algorithms in Java.
- Vast literature on complexity.
- Performance on real problems not always well-understood.

Approach 2. Reduce to a LP 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!

Bird’s-eye view

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

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

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 \( Y \).

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. [Which one wasn’t?]

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

allows quadratic tests of the form:
\[
x_i \times x_j \text{ or } (x_i - x_k)(x_j - x_k) < 0
\]

**Proposition.** Sorting linear-time reduces to convex hull.

\textbf{Pf.} [see next slide]

\begin{align*}
&1251432 \\
&2861534 \\
&3988818 \\
&4190745 \\
&13546464 \\
&89885444 \\
&43434213
\end{align*}

\begin{align*}
\text{sorting} & \quad \text{convex hull}
\end{align*}

\[ a \text{ quadratic test} \]

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

\[
\begin{aligned}
\text{Proposition.} & \quad \text{Sorting linear-time reduces to convex hull.} \\
\text{• Sorting instance: } & x_1, x_2, ..., x_N. \\
\text{• Convex hull instance: } & (x_1, x_1^2), (x_2, x_2^2), ..., (x_N, x_N^2).
\end{aligned}
\]

\begin{align*}
&y \\
&x \\
&f(x) = x^2
\end{align*}

\textbf{Pf.}
\begin{itemize}
  \item Region \((x : x^2 \geq x)\) is convex \(\Rightarrow\) all points are on hull.
  \item Starting at point with most negative \( x \), counter-clockwise order of hull points yields integers in ascending order.
\end{itemize}

\textbf{Proposition.} 3-SUM linear-time reduces to 3-COLLINEAR.

\textbf{Pf.} [see next 2 slide]

\begin{align*}
&1251432 \\
&-2861534 \\
&3988818 \\
&-4190745 \\
&13546464 \\
&89885444 \\
&-43434213
\end{align*}

\begin{align*}
\text{3-sum} & \quad \text{3-collinear}
\end{align*}

\textbf{Proposition.} 3-SUM requires \( \Omega(N^2) \) steps.

\textbf{Implication.} No sub-quadratic algorithm for 3-COLLINEAR likely.

\textbf{Conjecture.} Any algorithm for 3-SUM requires \( \Omega(N^2) \) steps.
Proposition. 3-SUM linear-time reduces to 3-COLLINEAR.
• 3-SUM instance: \( x_1, x_2, \ldots, x_N \).
• 3-COLLINEAR instance: \((x_1, x_1^3), (x_2, x_2^3), \ldots, (x_N, x_N^3)\).

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

Pf. Three distinct points \((a, a^3), (b, b^3), \) and \((c, c^3)\) are collinear iff:
\[
0 = \begin{vmatrix} a & a^3 & 1 \\ b & b^3 & 1 \\ c & c^3 & 1 \end{vmatrix} = a(b^3 - c^3) - b(a^3 - c^3) + c(a^3 - b^3) = (a - b)(b - c)(c - a)(a + b + c)
\]

More linear-time 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?
A2. [easy way] Linear-time reduction from sorting.

Q. How to convince yourself no sub-quadratic 3-COLLINEAR algorithm exists.
A2. [easy way] Linear-time reduction from 3-SUM.
Bird’s-eye view

Def. A problem is intractable if it can’t be solved in polynomial time.
Desiderata. Prove that a problem is intractable.

Two problems that require exponential time.
• Given a constant-size program, does it halt in at most $K$ steps?
• Given $N$-by-$N$ checkers board position, can the first player force a win?

Frustrating news. Few successes.

3-satisfiability

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

Clause. An or of 3 distinct literals. $C_1 = (\neg x_1 \lor x_2 \lor x_3)$

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

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

$\Phi = (\neg x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2 \lor x_3) \land (\neg x_1 \lor \neg x_2 \lor x_4) \land (\neg x_2 \lor x_3 \lor x_4)$

yes instance $x_1 x_2 x_3 x_4$ $T$ $T$ $F$ $T$

$(\neg T \lor T \lor F) \land (T \lor \neg T \lor F) \land (\neg T \lor \neg T \lor F) \land (\neg T \lor \neg T \lor T) \land (\neg T \lor F \lor T)$

Applications. Circuit design, program correctness, ...

3-satisfiability is believed 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 ($P \neq NP$). 3-SAT is intractable (no poly-time algorithm).
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 intractability. If 3-SAT poly-time reduces to Y, then Y is intractable.
(assuming 3-SAT is intractable)

Mentality.
- If I could solve Y in poly-time, then I could also solve 3-SAT in poly-time.
- 3-SAT is believed to be intractable.
- Therefore, so is Y.

Independent set

**Def.** An independent set is a set of vertices, no two of which are adjacent.

**IND-SET.** Given a graph G and an integer k, find an independent set of size k.

Applications. Scheduling, computer vision, clustering, ...

3-satisfiability reduces to independent set

**Proposition.** 3-SAT poly-time reduces to IND-SET.

**Pf.** Given an instance φ of 3-SAT, create an instance G of IND-SET:
- For each clause in φ, create 3 vertices in a triangle.
- Add an edge between each literal and its negation.

3-satisfiability reduces to independent set

**Proposition.** 3-SAT poly-time reduces to IND-SET.

**Pf.** Given an instance φ of 3-SAT, create an instance G of IND-SET:
- For each clause in φ, create 3 vertices in a triangle.
- Add an edge between each literal and its negation.

- G has independent set of size k ⇒ φ satisfiable.
Proposition. 3-SAT poly-time reduces to IND-SET.

Pf. Given an instance $\Phi$ of 3-SAT, create an instance $G$ of IND-SET:

- For each clause in $\Phi$, create 3 vertices in a triangle.
- Add an edge between each literal and its negation.

$G$ has independent set of size $k \iff \Phi$ satisfiable.

Assuming 3-SAT is intractable, so is IND-SET.

ILP. Given a system of linear inequalities, find an integral solution.

<table>
<thead>
<tr>
<th>Linear Inequalities</th>
<th>Integer Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>$3x_1 + 5x_2 + 2x_3 + x_4 + 4x_5 \geq 10$</td>
<td>$x_i \in {0, 1}$</td>
</tr>
<tr>
<td>$5x_1 + 2x_2 + 4x_4 + 1x_3 \leq 7$</td>
<td></td>
</tr>
<tr>
<td>$x_1 + x_3 + 2x_2 \leq 2$</td>
<td></td>
</tr>
<tr>
<td>$3x_1 + 4x_3 + 7x_4 \leq 7$</td>
<td></td>
</tr>
<tr>
<td>$x_1 + x_3 \leq 1$</td>
<td></td>
</tr>
<tr>
<td>$x_1 + x_5 \leq 1$</td>
<td></td>
</tr>
<tr>
<td>all $x_i \in {0, 1}$</td>
<td></td>
</tr>
</tbody>
</table>

Remark. Finding a real-valued solution is tractable (linear programming).
Proposition. 3-SAT poly-time reduces to IND-SET.
Proposition. IND-SET poly-time reduces to ILP.

Transitivity. If X poly-time reduces to Y and Y poly-time reduces to Z, then X-poly-time reduces to Z.

Implication. Assuming 3-SAT is intractable, so is ILP.

Implications of poly-time reductions from 3-satisfiability

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 (probably) intractable?
A1. [hard way] Long futile search for an efficient algorithm (as for 3-SAT).
A2. [easy way] Reduction from 3-SAT.

Caveat. Intricate reductions are common.

More poly-time reductions from 3-satisfiability

Conjecture. 3-SAT is intractable.
Implication. All of these problems are intractable.

Search problems

Search problem. Problem where you can check a solution in poly-time.

Ex 1. 3-SAT.
\[
\Phi = (x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_2 \lor x_4) \land (\neg x_1 \lor x_3 \lor \neg x_4) \land (x_1 \lor x_3 \lor x_4)
\]
\[x_1 = \text{true}, \ x_2 = \text{true}, \ x_3 = \text{false}, \ x_4 = \text{true}\]

Ex 2. IND-SET.

\[
\{v_0, v_1, v_2\}
\]
\[d = 3\]
**P vs. NP**

**P.** Set of search problems solvable in poly-time.

**Importance.** What scientists and engineers can compute feasibly.

**NP.** Set of search problems.

**Importance.** What scientists and engineers aspire to compute feasibly.

**Fundamental question.**

**Consensus opinion.** No.

---

**Cook’s theorem**

**Def.** An NP is **NP-complete** if all problems in NP poly-time to reduce to it.

**Cook’s theorem.** 3-SAT is NP-complete.

**Corollary.** 3-SAT is tractable if and only if P = NP.

**Two worlds.**

---

**Implications of Cook’s theorem**

- 3-SAT reduces to 3-SAT
- 3-COLOR reduces to 3-SAT
- EXACT COVER reduces to 3-SAT
- SUBSET-SUM reduces to 3-SAT
- PARTITION reduces to 3-SAT
- KNAPSACK reduces to 3-SAT

All of these problems (and many, many more) poly-time reduce to 3-SAT

---

**Implications of Karp + Cook**

- 3-SAT reduces to 3-COLOR
- 3-COLOR reduces to 3-SAT
- 3-SAT reduces to 3-COLOR

All of these problems are NP-complete; they are manifestations of the same really hard problem.
Implications of NP-completeness

Birds-eye view: review

Desiderata. Classify problems according to computational requirements.

<table>
<thead>
<tr>
<th>complexity</th>
<th>order of growth</th>
<th>examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear</td>
<td>$N$</td>
<td>min, max, median, Burrows-Wheeler transform, …</td>
</tr>
<tr>
<td>linearithmic</td>
<td>$N \log N$</td>
<td>sorting, convex hull, closest pair, farthest pair, …</td>
</tr>
<tr>
<td>quadratic</td>
<td>$N^2$</td>
<td>???</td>
</tr>
<tr>
<td>exponential</td>
<td>$c^N$</td>
<td>???</td>
</tr>
</tbody>
</table>

Frustrating news. Huge number of problems have defied classification.

Birds-eye view: revised

Desiderata. Classify problems according to computational requirements.

<table>
<thead>
<tr>
<th>complexity</th>
<th>order of growth</th>
<th>examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear</td>
<td>$N$</td>
<td>min, max, median, Burrows-Wheeler transform, …</td>
</tr>
<tr>
<td>linearithmic</td>
<td>$N \log N$</td>
<td>sorting, convex hull, closest pair, farthest pair, …</td>
</tr>
<tr>
<td>3-SUM complete</td>
<td>probably $N^2$</td>
<td>3-SUM, 3-COLLINEAR, 3-CONCURRENT, …</td>
</tr>
<tr>
<td>NP-complete</td>
<td>probably $c^N$</td>
<td>3-SAT, IND-SET, ILP, …</td>
</tr>
</tbody>
</table>

Good news. Can put problems in equivalence classes.

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, priority queue, symbol table, set, graph
  - sorting, regular expression, Delaunay triangulation
  - minimum spanning tree, shortest path, maximum flow, linear programming
- Determine difficulty of your problem and choose the right tool.
  - use exact algorithm for tractable problems
  - use heuristics for intractable problems