Overview: introduction to advanced topics

Main topics. [next 2 lectures]
- Reduction: design algorithms, establish lower bounds, classify problems.
- Intractability: problems beyond our reach.
- Combinatorial search: coping with intractability.

Shifting gears.
- From individual problems to problem-solving models.
- From linear/quadratic to polynomial/exponential scale.
- From details of implementation to conceptual framework.

Goals.
- Place algorithms we’ve studied in a larger context.
- Introduce you to important and essential ideas.
- Inspire you to learn more about algorithms!

6.5 Reductions

Overview: introduction to advanced topics

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, element distinctness, convex hull, closest pair, ...</td>
</tr>
<tr>
<td>quadratic</td>
<td>( N^2 )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( c^n )</td>
<td></td>
</tr>
</tbody>
</table>

Frustrating news. Huge number of problems have defied classification.
Bird's-eye view

Desiderata. Classify problems according to computational requirements.

Suppose we could (could not) solve problem $X$ efficiently. What else could (could not) 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 $X$.

Ex 1. [finding the median reduces to sorting]
To find the median of $N$ items:
- Sort $N$ items.
- Return item in the middle.

Cost of solving finding the median. $N \log N + 1$.

Ex 2. [element distinctness reduces to sorting]
To solve element distinctness on $N$ items:
- Sort $N$ items.
- Check adjacent pairs for equality.

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

- Introduction
- Designing algorithms
- Establishing lower bounds
- Classifying problems
- Intractability

Reduction: design algorithms

Definition. 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 $X$.

More familiar reduction examples.
- 3-collinear reduces to sorting.
- CPM reduces to topological sort.
- Arbitrage reduces to shortest paths.
- Baseball elimination reduces to maxflow.
- Burrows-Wheeler transform reduces to suffix sort.
- ...

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

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 counterclockwise order).

Convex hull reduces to sorting

Proposition. Convex hull reduces to sorting.

Proof. Graham scan algorithm.

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

Shortest paths on edge-weighted graphs and digraphs

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

Proof. Replace each undirected edge by two directed edges.
Shortest paths on edge-weighted graphs and digraphs

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

Cost of undirected shortest paths. \( E \log V + E. \)

Shortest paths with negative weights

**Caveat.** Reduction is invalid for edge-weighted graphs with negative weights (even if no negative cycles).

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

Some reductions involving familiar problems

- **Computational geometry**
  - 2d farthest pair
  - convex hull
  - median
  - element distinctness
  - sorting
  - 2d closest pair
  - 2d Euclidean MST
  - Delaunay triangulation

- **Combinatorial optimization**
  - undirected shortest paths (nonnegative)
  - directed shortest paths (nonnegative)
  - arbitrage
  - maximum flow
  - directed shortest paths (no neg cycles)
  - baseball elimination
  - linear programming

6.5 **Reductions**

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

http://algs4.cs.princeton.edu
Bird’s-eye view

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

**Ex.** In decision tree model, any compare-based sorting algorithm requires $\Omega(N \log N)$ compares in the worst case.

![Decision tree diagram]

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

**Good news.** 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 ones weren’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 $Y$.

Lower bound for convex hull

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

![Decision tree diagram]

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

**Pf.** [see next slide]

![Sorting vs. convex hull]

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

Sorting linear-time reduces to convex hull

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

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

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

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:

$$0 = \begin{vmatrix} a & a^3 & 1 \\ b & b^3 & 1 \\ c & c^3 & 1 \end{vmatrix}$$

$$= a(b^3 - c^3) - b(a^3 - c^3) + c(a^3 - b^3)$$

$$= (a - b)(b - c)(c - a)(a + b + c)$$
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.

Classifying problems: summary

Desiderata. Problem with algorithm that matches lower bound.
Ex. Sorting and convex hull have complexity $N \log N$.

Desiderata’. Prove that two problems $X$ and $Y$ have the same complexity.
• First, show that problem $X$ linear-time reduces to $Y$.
• Second, show that $Y$ linear-time reduces to $X$.
• Conclude that $X$ and $Y$ have the same complexity.

Integer arithmetic reductions

Integer multiplication. Given two $N$-bit integers, compute their product.
Brute force. $N^2$ bit operations.
Integer arithmetic reductions

**Integer multiplication.** Given two $N$-bit integers, compute their product.

**Brute force.** $N^2$ bit operations.

<table>
<thead>
<tr>
<th>problem</th>
<th>arithmetic</th>
<th>order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer multiplication</td>
<td>$a \times b$</td>
<td>$M(N)$</td>
</tr>
<tr>
<td>integer division</td>
<td>$a \div b, a \mod b$</td>
<td>$M(N)$</td>
</tr>
<tr>
<td>integer square</td>
<td>$a^2$</td>
<td>$M(N)$</td>
</tr>
<tr>
<td>integer square root</td>
<td>$\lfloor \sqrt{a} \rfloor$</td>
<td>$M(N)$</td>
</tr>
</tbody>
</table>

integer arithmetic problems with the same complexity as integer multiplication

**Q.** Is brute-force algorithm optimal?

History of complexity of integer multiplication

<table>
<thead>
<tr>
<th>year</th>
<th>algorithm</th>
<th>order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>brute force</td>
<td>$N^2$</td>
</tr>
<tr>
<td>1962</td>
<td>Karatsuba</td>
<td>$N^{1.585}$</td>
</tr>
<tr>
<td>1963</td>
<td>Toom-3, Toom-4</td>
<td>$N^{1.465}, N^{1.404}$</td>
</tr>
<tr>
<td>1966</td>
<td>Toom-Cook</td>
<td>$N^{1+\epsilon}$</td>
</tr>
<tr>
<td>1971</td>
<td>Schönhage-Strassen</td>
<td>$N \log N \log \log N$</td>
</tr>
<tr>
<td>2007</td>
<td>Furer</td>
<td>$N \log N 2^{\log^{\log N}}$</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>$N$</td>
</tr>
</tbody>
</table>

number of bit operations to multiply two $N$-bit integers

**Remark.** GNU Multiple Precision Library uses one of five different algorithm depending on size of operands.

Linear algebra reductions

**Matrix multiplication.** Given two $N$-by-$N$ matrices, compute their product.

**Brute force.** $N^3$ flops.

<table>
<thead>
<tr>
<th>row $i$</th>
<th>0.1</th>
<th>0.2</th>
<th>0.8</th>
<th>0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.3</td>
<td>0.9</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>0.0</td>
<td>0.7</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>0.3</td>
<td>0.3</td>
<td>0.1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>column $j$</th>
<th>0.4</th>
<th>0.3</th>
<th>0.1</th>
<th>0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>0.2</td>
<td>0.0</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.4</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>0.4</td>
<td>0.1</td>
<td>0.9</td>
<td></td>
</tr>
</tbody>
</table>

$0.5 \cdot 0.1 + 0.3 \cdot 0.0 + 0.9 \cdot 0.4 + 0.6 \cdot 0.1 = 0.47$

**Q.** Is brute-force algorithm optimal?
History of complexity of matrix multiplication

<table>
<thead>
<tr>
<th>year</th>
<th>algorithm</th>
<th>order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>brute force</td>
<td>(N^3)</td>
</tr>
<tr>
<td>1969</td>
<td>Strassen</td>
<td>(N^{2.808})</td>
</tr>
<tr>
<td>1978</td>
<td>Pan</td>
<td>(N^{2.796})</td>
</tr>
<tr>
<td>1979</td>
<td>Bini</td>
<td>(N^{2.780})</td>
</tr>
<tr>
<td>1981</td>
<td>Schönhage</td>
<td>(N^{2.522})</td>
</tr>
<tr>
<td>1982</td>
<td>Romani</td>
<td>(N^{2.517})</td>
</tr>
<tr>
<td>1982</td>
<td>Coppersmith-Winograd</td>
<td>(N^{2.496})</td>
</tr>
<tr>
<td>1986</td>
<td>Strassen</td>
<td>(N^{2.479})</td>
</tr>
<tr>
<td>1989</td>
<td>Coppersmith-Winograd</td>
<td>(N^{2.376})</td>
</tr>
<tr>
<td>2010</td>
<td>Strother</td>
<td>(N^{2.3737})</td>
</tr>
<tr>
<td>2011</td>
<td>Williams</td>
<td>(N^{2.3727})</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>(N^{2 + \epsilon})</td>
</tr>
</tbody>
</table>

The number of floating-point operations to multiply two \(N\)-by-\(N\) matrices.

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 provably 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.** Very few successes.

6.5 Reductions

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

A key problem: satisfiability

**SAT.** Given a system of boolean equations, find a solution.

**Ex.**

\[
\begin{align*}
\neg x_1 \lor x_2 \lor x_3 &= \text{true} \\
x_1 \lor \neg x_2 \lor x_4 &= \text{true} \\
\neg x_1 \lor \neg x_2 \lor \neg x_3 &= \text{true} \\
\neg x_1 \lor \neg x_2 \lor x_4 &= \text{true} \\
x_2 \lor x_3 \lor x_4 &= \text{true}
\end{align*}
\]

3-SAT. All equations of this form (with three variables per equation).

**Key applications.**

- Automatic verification systems for software.
- Mean field diluted spin glass model in physics.
- Electronic design automation (EDA) for hardware.
- ...
Satisfiability is conjectured to be 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).

Independent set

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

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

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

Polynomial-time reductions

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 \( Y \).

3-satisfiability reduces to independent set

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.

\[\Phi = (x_1 \text{ or } x_2 \text{ or } x_3) \text{ and } (\neg x_1 \text{ or } \neg x_2 \text{ or } x_4) \text{ and } (\neg x_1 \text{ or } x_3 \text{ or } \neg x_4) \text{ and } (x_1 \text{ or } x_3 \text{ or } x_4)\]
3-satisfiability reduces to independent set

**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.

\[
\Phi = \left( x_1 \text{ or } x_2 \text{ or } x_3 \right) \text{ and } \left( \neg x_1 \text{ or } \neg x_2 \text{ or } x_4 \right) \text{ and } \left( \neg x_1 \text{ or } x_3 \text{ or } \neg x_4 \right) \text{ and } \left( x_1 \text{ or } x_3 \text{ or } x_4 \right)
\]

- \( \Phi \) satisfiable \( \Rightarrow \) \( G \) has independent set of size \( k \).
- For each of \( k \) clauses, include in independent set one vertex corresponding to a true literal

3-satisfiability reduces to independent set

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

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

\[
\Phi = \left( x_1 \text{ or } x_2 \text{ or } x_3 \right) \text{ and } \left( \neg x_1 \text{ or } \neg x_2 \text{ or } x_4 \right) \text{ and } \left( \neg x_1 \text{ or } x_3 \text{ or } \neg x_4 \right) \text{ and } \left( x_1 \text{ or } x_3 \text{ or } x_4 \right)
\]

Integer linear programming

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

\[
\begin{align*}
3x_1 + 5x_2 + 2x_3 + x_4 + 4x_5 & \geq 10 \\
5x_1 + 2x_2 + 4x_4 + 1x_3 & \leq 7 \\
x_1 + x_3 + 2x_4 & \leq 2 \\
3x_1 + 4x_3 + 7x_4 & \leq 7 \\
x_1 + x_3 & \leq 1 \\
0 & \leq x_i \leq 1
\end{align*}
\]

yes instance:
\[
\begin{bmatrix}
x_1 & x_2 & x_3 & x_4 & x_5 \\
0 & 1 & 0 & 1 & 1
\end{bmatrix}
\]

**Context.** Cornerstone problem in operations research.

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

**Pf.** Given instance \( \{ G, k \} \) of \( IND-SET \), create an instance of \( ILP \) as follows:

\[
\begin{align*}
    &x_1 + x_2 + x_3 + x_4 = 3 & \text{number of vertices selected} \\
    &x_1 + x_2 \leq 1 & \text{at most one vertex selected from each edge} \\
    &x_2 + x_3 \leq 1 \\
    &x_1 + x_3 \leq 1 \\
    &x_1 + x_4 \leq 1 \\
    &x_3 + x_5 \leq 1 \\
    &\text{all } x_i = \{ 0, 1 \} & \text{binary variables}
\end{align*}
\]

**Intuition.** \( x_i = 1 \) if and only if vertex \( v_i \) is in independent set.

**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.
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.

\[ k = 3 \]
\[ \{v_2, \: v_4, \: v_5\} \]

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-Levin theorem

An NP problem is NP-COMPLETE if all problems in NP poly-time to reduce to it.

Cook-Levin theorem. 3-SAT is NP-COMPLETE.
Corollary. 3-SAT is tractable if and only if P = NP.

Two worlds.

Implications of Cook-Levin theorem

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

Stephen Cook ‘82 Turing award
Leonid Levin

\[ P = \text{NP?} \]

P = NP ?
Implications of Karp + Cook-Levin

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,</td>
</tr>
<tr>
<td>linearithmic</td>
<td>( N \log N )</td>
<td>sorting, convex hull,</td>
</tr>
<tr>
<td>quadratic</td>
<td>( N^2 )</td>
<td>?</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>NP-complete</td>
<td>probably not ( N^b )</td>
<td>3-SAT, IND-SET, ILP, ...</td>
</tr>
</tbody>
</table>

Frustrating news. Huge number of problems have defied classification.

Birds-eye view: revised

Desiderata. Classify problems according to computational requirements.

Good news. Can put many problems into equivalence classes.

Complexity zoo

Complexity class. Set of problems sharing some computational property.

http://qwiki.stanford.edu/index.php/Complexity_Zoo

Bad news. Lots of complexity classes.
Summary

Reductions are important in theory to:

- Design algorithms.
- Establish lower bounds.
- Classify problems according to their computational requirements.

Reductions are important in practice to:

- Design algorithms.
- Design reusable software modules.
  - stacks, queues, priority queues, symbol tables, sets, graphs
  - sorting, regular expressions, suffix arrays
  - MST, shortest path, maxflow, linear programming
- Determine difficulty of your problem and choose the right tool.