Reductions

Desiderata

Desiderata. Classify problems according to their computational requirements.

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

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

Reduction

Def. Problem X reduces to problem Y if given a subroutine for Y, can solve X.

- Cost of solving X = cost of solving Y + cost of reduction.
- Ex: X = closest pair, Y = Voronoi.

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

Linear Time Reductions
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.
- Notation: $X \leq_L Y$.

**Some familiar examples.**
- Dedup $\leq_L$ sorting.
- Median $\leq_L$ sorting.
- Convex hull $\leq_L$ Voronoi.
- Closest pair $\leq_L$ Voronoi.
- Arbitrage $\leq_L$ negative cycle detection.
- Brewer’s problem $\leq_L$ linear programming.

---

**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.

---

**Shortest Paths**

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

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

**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.
Convex Hull and Sorting

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

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

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

**Pf.** Graham scan algorithm.

Sorting instance.
Convex hull instance.

**Observation.** Region \( \{x : x^2 \geq x\} \) 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.

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.
3-SUM Reduces to 3-COLLINEAR

Claim. Given N distinct integers, are there 3 that sum to 0?

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

Claim. 3-SUM \( \leq L \) 3-COLLINEAR.

\textbf{Pf.} 3-SUM instance: \( x_1, x_2, \ldots, x_N \)

3-COLLINEAR instance: \((x_1, x_1^3), (x_2, x_2^3), \ldots, (x_N, x_N^3)\)

we just proved this

3-SUM and 3-COLLINEAR

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

Claim. 3-SUM \( \leq L \) 3-COLLINEAR.

\textbf{Corollary.} If no sub-quadratic algorithm for 3-SUM, then no sub-quadratic algorithm for 3-COLLINEAR.
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 decimal), is x prime?

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

Claim. \( \text{COMPOSITE} \leq_L \text{PRIME} \).

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

Reduction Gone Wrong

**Caveat.**
- System designer specs the interfaces for project.
- One programmer might implement \( \text{isComposite} \) using \( \text{isPrime} \).
- Another programmer might implement \( \text{isPrime} \) using \( \text{isComposite} \).
- Be careful to avoid infinite reduction loops in practice.

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

```
public static boolean isComposite(int x) {
    if (isPrime(x)) return false;
    else                return true;
}
```
Poly-Time Reductions

**Def.** Problem $X$ polynomial reduces to problem $Y$ if arbitrary instances of problem $X$ can be solved using:
- Polynomial number of standard computational steps, plus
- One call to subroutine for $Y$.

**Notation.** $X \leq_p Y$.

**Ex.** Assignment problem $\leq_p$ LP.
**Ex.** 3-SAT $\leq_p$ 3-COLOR.

---

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

**Establish tractability.** If $X \leq_p Y$ and $Y$ can be solved in poly-time, then $X$ can be solved in poly-time.

**Establish intractability.** If $Y \leq_p X$ and $Y$ cannot be solved in poly-time, then $X$ cannot be solved in poly-time.

**Useful property.** If $X \leq_p Y$ and $Y \leq_p Z$ then $X \leq_p Z$.

---

**Assignment Problem**

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

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8</td>
<td>9</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>7</td>
<td>16</td>
<td>14</td>
</tr>
<tr>
<td>9</td>
<td>13</td>
<td>11</td>
<td>19</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>13</td>
<td>12</td>
<td>20</td>
<td>13</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

$\text{cost} = 3 \times 10 + 11 \times 9 + 10 \times 11 = 53$

---

**Applications.** Match jobs to machines, match personnel to tasks, match PU students to writing seminars.
Assignment Problem Reduces to Linear Programming

LP formulation. \( x_{ij} = 1 \) if job \( j \) assigned to machine \( i \).

\[
\min \sum_{i \in \mathcal{M}, j \in \mathcal{J}} c_{ij} x_{ij} \\
\text{s.t.} \\
\sum_{i \in \mathcal{M}} x_{ij} = 1 \quad 1 \leq i \leq n \\
\sum_{j \in \mathcal{J}} x_{ij} = 1 \quad 1 \leq j \leq n \\
x_{ij} \geq 0 \quad 1 \leq i, j \leq n
\]

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

Corollary. Assignment problem reduces to LP; can solve in poly-time.

we assume LP returns an extreme point solution

3-Satisfiability

Literal: A Boolean variable or its negation. \( x_i \) or \( \overline{x}_i \)

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

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

SAT. Given CNF formula \( \Phi \), does it have a satisfying truth assignment?

Ex. \( (\overline{x}_1 \lor x_2 \lor x_3) \land (x_1 \lor \overline{x}_2 \lor \overline{x}_3) \land (\overline{x}_1 \lor \overline{x}_2 \lor \overline{x}_3) \)

Yes. \( x_1 = \text{true}, x_2 = \text{true}, x_3 = \text{false} \)

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

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.
1. Create one vertex for each literal.
2. Create 3 new vertices T, F, and B; connect them in a triangle, and connect each literal to B.
3. Connect each literal to its negation.
4. For each clause, attach a gadget of 6 vertices and 13 edges.

Graph 3-Colorability

Claim. Graph is 3-colorable iff \(\Phi\) is satisfiable.

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

Graph 3-Colorability

Claim. Graph is 3-colorable iff \(\Phi\) is satisfiable.

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

Claim. Graph is 3-colorable iff $\Phi$ is satisfiable.

Pf. Suppose 3-SAT formula $\Phi$ is satisfiable.
- Color all true literals $T$.
- Color node below green node $F$, and node below that $B$.
- Color remaining middle row nodes $B$.
- Color remaining bottom nodes $T$ or $F$ as forced.

More Poly-Time Reductions

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

Reductions are important in theory to:
  - Classify problems according to their computational requirements.
  - Establish intractability.
  - Establish tractability.

Reductions are important in practice to:
  - Design algorithms.
  - Design reusable software modules.
    - stack, queue, sorting, priority queue, symbol table
    - 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 NP-hard problems
      e.g., bin packing