7.5 Reductions

- designing algorithms
- establishing lower bounds
- intractability
Bird's-eye view

**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.
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 $X$.

\[
\text{Cost of solving } X = \text{total cost of solving } Y + \text{cost of reduction.}
\]

- perhaps many calls to $Y$
- on problems of different sizes
- preprocessing and postprocessing
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. $N \log N + N$
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.
  - check adjacent triples for collinearity

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

establishing lower bounds

intractability
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.** $N \log N + N$. 
Proposition. Undirected shortest path (with nonnegative weights) reduces to directed shortest path.
**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. \( E \log E + E. \)
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.
Some reductions involving familiar problems

1. Linear programming
2. Element distinctness
3. Sorting
4. Convex hull
5. Median
6. Element distinctness
7. Closest pair 2d
8. Euclidean MST 2d
9. Delaunay triangulation
10. Directed shortest paths (nonnegative)
11. Bipartite matching
12. Convex hull
13. Median
14. Arbitrage
15. Shortest paths (no neg cycles)
16. Maximum flow
17. Undirected shortest paths (nonnegative)
18. Linear programming
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.

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 \( a_{ij}, c_j, \) and \( b_i. \)

Output: real numbers \( x_j. \)

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
- Maximum flow.
- Bipartite matching.
  ... 
- [ a very long list ]
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

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

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

maximize $x_t$
subject to the constraints

\[
egin{align*}
x_s + 9 & \geq x_2 \\
x_s + 14 & \geq x_6 \\
x_s + 15 & \geq x_7 \\
x_2 + 24 & \geq x_3 \\
x_3 + 2 & \geq x_5 \\
x_3 + 19 & \geq x_t \\
x_4 + 6 & \geq x_3 \\
x_4 + 6 & \geq x_t \\
x_5 + 11 & \geq x_4 \\
x_5 + 16 & \geq x_t \\
x_6 + 18 & \geq x_3 \\
x_6 + 30 & \geq x_5 \\
x_6 + 5 & \geq x_7 \\
x_7 + 20 & \geq x_5 \\
x_7 + 44 & \geq x_t \\
x_s & = 0 
\end{align*}
\]

solution
\[
\begin{align*}
x_s & = 0  \\
x_2 & = 9  \\
x_3 & = 32  \\
x_4 & = 45  \\
x_5 & = 34  \\
x_6 & = 14  \\
x_7 & = 15  \\
x_t & = 50
\end{align*}
\]
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 \( t \).
Maximum flow reduces to LP

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

\[
\begin{align*}
\text{maximize} & \quad x_{3t} + x_{4t} \\
\text{subject to the constraints} & \quad \begin{aligned}
  x_{s1} & \leq 2 \\
  x_{s2} & \leq 3 \\
  x_{13} & \leq 3 \\
  x_{14} & \leq 1 \\
  x_{23} & \leq 1 \\
  x_{24} & \leq 1 \\
  x_{3t} & \leq 1 \\
  x_{4t} & \leq 3 \\
\end{aligned} \\
\text{equilibrium constraints} & \quad \begin{aligned}
  x_{s1} = x_{13} + x_{14} \\
  x_{s2} = x_{23} + x_{24} \\
  x_{13} + x_{23} = x_{3t} \\
  x_{14} + x_{24} = x_{4t} \\
  \text{all } x_{ij} \geq 0
\end{aligned}
\]

interpretation:
\( x_{ij} = \text{flow in edge } i-j \)

interpretation:
\( x_{ij} = \text{flow in edge } i-j \)

capacity constraints

add dummy edge from \( t \) to \( s \)
Maxflow problem reduces to LP

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

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

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

equilibrium constraints
\[
\begin{align*}
x_{s1} &= x_{13} + x_{14} \\
x_{s2} &= x_{23} + x_{24} \\
x_{13} + x_{23} &= x_{3t} \\
x_{14} + x_{24} &= x_{4t}
\end{align*}
\]

subject to the constraints

maximize
\[
x_{3t} + x_{4t}
\]

subject to the constraints

capacity constraints
\[
\begin{align*}
x_{s1} &\leq 2 \\
x_{s2} &\leq 3 \\
x_{13} &\leq 3 \\
x_{14} &\leq 1 \\
x_{23} &\leq 1 \\
x_{24} &\leq 1 \\
x_{3t} &\leq 2 \\
x_{4t} &\leq 3
\end{align*}
\]

solution
\[
\begin{align*}
x_{s1} &= 2 \\
x_{s2} &= 2 \\
x_{13} &= 1 \\
x_{14} &= 1 \\
x_{23} &= 1 \\
x_{24} &= 1 \\
x_{3t} &= 2 \\
x_{4t} &= 2
\end{align*}
\]

add dummy edge from t to s
Maximum cardinality bipartite matching problem

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 residence positions.
- Ex: students to writing seminars.
- [many other examples]
Maximum cardinality bipartite matching reduces to LP

LP formulation.
• One variable per edge, one equality per vertex.
• Interpretation: an edge is in matching iff $x_i = 1$.

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 reduces to LP

LP formulation.
- One variable per edge, one equality per vertex.
- Interpretation: an edge is in matching iff $x_i = 1$. 

\[
\begin{align*}
\text{maximize} & \quad x_{A0} + x_{A1} + x_{A2} + x_{B0} + x_{B1} + x_{B5} + x_{C2} + x_{C3} + x_{C4} + x_{D0} + x_{D1} + x_{E3} + x_{E4} + x_{E5} + x_{F2} + x_{F4} + x_{F5} \\
\text{subject to the constraints} & \quad x_{A0} + x_{A1} + x_{A2} = 1  &  x_{A0} + x_{B0} + x_{D0} = 1 \\
& \quad x_{B0} + x_{B1} + x_{B5} = 1  &  x_{A1} + x_{B1} + x_{D1} = 1 \\
& \quad x_{C2} + x_{C3} + x_{C4} = 1  &  x_{A2} + x_{C2} + x_{F2} = 1 \\
& \quad x_{D0} + x_{D1} = 1  &  x_{C3} + x_{E3} = 1 \\
& \quad x_{E3} + x_{E4} + x_{E5} = 1  &  x_{C4} + x_{E4} + x_{F4} = 1 \\
& \quad x_{F2} + x_{F4} + x_{F5} = 1  &  x_{B5} + x_{E5} + x_{F5} = 1 \\
\text{all } x_{ij} \geq 0
\end{align*}
\]
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!

% 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
• establishing lower bounds
• intractability
**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 $Y$.

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

argument must apply to all conceivable algorithms

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. [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.
Lower bound for convex hull

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

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

**Pf.** [see next slide]

**Implication.** Any ccw-based convex hull algorithm requires $\Omega(N \log N)$ ccw's.
Proposition. Sorting linear-time reduces to convex hull.

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

**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?

| 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:** \(x_1, x_2, \ldots, x_N\).
- **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 \((a, a^3), (b, b^3), \text{ and } (c, c^3)\) are collinear.
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

- **Element distinctness**
  - (N log N lower bound)
  - Sorting
  - Convex hull 2d
  - Delaunay

- **3-sum**
  - (Conjectured N^2 lower bound)
  - Closest pair 2d
  - Euclidean MST 2d
  - 3-collinear
  - 3-concurrent
  - Min area triangle
  - Dihedral rotation
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.
› designing algorithms
› establishing lower bounds
› intractability
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?

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

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

\[
(\neg T \lor T \lor F) \land (T \lor \neg T \lor F) \land (\neg T \lor \neg T \lor \neg F) \land (\neg T \lor \neg T \lor T) \land (\neg T \lor F \lor T)
\]

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.
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 $\Phi$ 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.

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

$k = 4$

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

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

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

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

for each clause, take vertex corresponding to one true literal
Proposition. 3-SAT poly-time reduces to IND-SET.

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

$\Phi = (x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_2 \lor x_4) \land (\neg x_1 \lor x_3 \lor \neg x_4) \land (x_1 \lor x_3 \lor x_4)$
**Integer linear programming**

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

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

**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 an instance \( G, k \) of IND-SET, create an instance of ILP as follows:

**Intuition.** \( x_i = 1 \) if and only if vertex \( v_i \) is in independent set.
3-satisfiability reduces to integer 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.
More poly-time reductions from 3-satisfiability

- 3-SAT
  - 3-COLOR
    - EXACT COVER
      - SUBSET-SUM
  - IND-SET
    - 3-SAT reduces to ILP
      - ILP
        - PARTITION
          - KNAPSACK
  - VERTEX COVER
    - CLIQUE
      - HAM-CYCLE
        - TSP
        - HAM-PATH

Dick Karp
'85 Turing award

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

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

\( x_1 = true, \ x_2 = true, \ x_3 = true, \ x_4 = true \)

Ex 2. IND-SE T.

\[ \left\{ v_2, x_4, v_5 \right\} \]

\( k = 3 \)
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.

\[ \begin{array}{c}
\text{NP} \\
\text{P} \\
\text{NPC}
\end{array} \quad \begin{array}{c}
P \neq NP \\
P = NP
\end{array} \]

\[ \begin{array}{c}
P \neq NP \\
P = NP
\end{array} \]
Implications of Cook’s theorem

3-COLOR reduces to 3-SAT

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

Stephen Cook
'82 Turing award
All of these problems are NP-complete; they are manifestations of the same really hard problem.
Implications of NP-completeness

"I can’t find an efficient algorithm, but neither can all these famous people."
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.
**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