NP and computational intractability

Kleinberg and Tardos, chapter 8
So far we have studied certain algorithmic patterns: Greedy, Divide and conquer, Dynamic programming to develop efficient algorithms.

Now we want to classify and quantify problems that cannot be solved efficiently.

Our tool for doing this is another algorithmic pattern: reduction.

Reduction transforms the input and output of an algorithm so that it can be used to solve a different problem.
An engineer is handed an **empty** kettle and is asked to make tea. The engineer fills up the kettle, boils the water and makes tea.

A mathematician is handed a **full** kettle and is asked to make tea.

What do you think the mathematician does?

The mathematician empties the kettle and hands it to the engineer 😊
Reductions

Reuse, Reduce, Recycle...

If we have a solution to one problem and we can use this solution to solve another problem, we do not need to write a new program, we can reuse the existing code, and reduce the new problem (change its input (and output)), so it can use the existing code to solve it.

eg 1: We have a max heap, but we need a min heap. How can we use the max heap to perform min heap operations, without changing one bit of the max heap code?

Insert(x): InsertMaxHeap(-x);

Extract(): x=ExtractMaxHeap(); return –x;
Arbitrage

We have a large number of currencies and conversion rates from each currency to each other currency.

Is there a cyclic sequence of currency exchanges that provides a profit? eg:

loss: $ \rightarrow \£ \rightarrow \€ \rightarrow ¥ \rightarrow $: \( 0.63 \times 1.14 \times 106 \times 0.013 = 0.981 \)

profit: $ \rightarrow \£ \rightarrow \€ \rightarrow ¥ \rightarrow $: \( 0.64 \times 1.14 \times 106 \times 0.013 = 1.005 \)
Arbitrage

We have a large number of currencies and conversion rates from each currency to each other currency.

Is there a cyclic sequence of currency exchanges that provides a profit?

Is there an algorithm that we studied, that we can use to solve his problem?

Bellman Ford: finding negative cycles in a graph

How would we reduce to that algorithm?

rate $\rightarrow$ -(log rate)

Try it for: $2 \frac{1}{4}$  $4$  $\frac{1}{4}$ and for: $2 \frac{1}{2}$  $4$  $\frac{1}{4}$ and $2$  $1$  $4$  $\frac{1}{4}$
Algorithm Design Patterns and Anti-Patterns

Algorithm design patterns

- **Greedy.**
  
  Example: \(O(n \log n)\) interval scheduling

- **Divide-and-conquer.**
  
  Example: \(O(n \log n)\) closest point pair

- **Dynamic programming.**
  
  Example: \(O(n^2)\) sequence alignment

- **Reductions.**

- **Approximation algorithms.**

- **Randomized algorithms.**

Algorithm design anti-patterns

- **NP-completeness.**
  
  Example: \(O(n^k)\) algorithm unlikely.

- **Undecidability.**
  
  Example: No algorithm possible.
Q. Which problems will we be able to solve in practice?

A working definition: $P$
Those with polynomial-time algorithms.

Some problems provably require exponential-time.
- Towers of Hanoi
- Generate all permutations of 1 ... $n$

Grey area:
A huge number of fundamental problems have defied classification for decades.
We don’t know of a polynomial algorithm for them, and we cannot prove that no polynomial algorithm exists.
Suppose we could solve X in polynomial-time. What else could we solve in polynomial time?

**Reduction.** Problem Y polynomialsly reduces to problem X if arbitrary instances of problem Y can be solved using:
- Polynomial number of standard computational steps, plus
- Polynomial number of calls to black box that solves problem X.
- Because polynomial (plus or times) polynomial is polynomial

**Notation.** \( Y \leq_p X \).

**Remarks.**
- We pay for the time to write down instances sent to black box
  \( \Rightarrow \) inputs of X must be of polynomial size.
Purpose. Classify problems according to relative difficulty.

Design algorithms. If \( Y \leq_p X \) and \( X \) can be solved in polynomial-time, then \( Y \) can also be solved in polynomial time.

Establish intractability. If \( Y \leq_p X \) and \( Y \) cannot be solved in polynomial-time, then \( X \) cannot be solved in polynomial time.

Establish equivalence. If \( X \leq_p Y \) and \( Y \leq_p X \) they are as hard, notation: \( X \equiv_p Y \).
Optimization vs Decision Problems

**Decision problem**
Does there exist an independent set of size $\geq k$?

**Optimization problem**
Find independent set of maximum cardinality.

Easier to focus on **decision problems** (yes / no answers)
This is what we will do.
Reduction Strategies

- Reduction by equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.
INDEPENDENT SET: Given a graph $G = (V, E)$ and an integer $k$, is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$, and for each edge at most one of its endpoints is in $S$? (i.e. no edges join nodes in $S$)

- Is there an independent set of size $\geq 6$? Yes.
- Is there an independent set of size $\geq 7$? No.
VERTEX COVER: Given a graph $G = (V, E)$ and an integer $k$, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$, and for each edge, at least one of its endpoints is in $S$? (i.e., the nodes in $S$ cover all edges in $E$)

Is there a vertex cover of size $\leq 4$? Yes.
Is there a vertex cover of size $\leq 3$? No.
**Claim.** Let $G=(V, E)$ be a graph. Then $S$ is an independent set iff $V-S$ is a vertex cover.
Claim. Let $G=(V, E)$ be a graph. Then $S$ is an independent set iff $V - S$ is a vertex cover.

$\Rightarrow$

- Let $S$ be an independent set.
- Consider an arbitrary edge $(u, v)$.
- $S$ independent $\Rightarrow u \notin S$ or $v \notin S$ $\Rightarrow u \in V - S$ or $v \in V - S$.
- Thus, $V - S$ covers $(u, v)$.

$\Leftarrow$

- Let $V - S$ be a vertex cover.
- Consider two nodes $u \in S$ and $v \in S$.
- Observe that $(u, v) \notin E$ since $V - S$ is a vertex cover, and therefore all edges have at least one node in $V - S$.
- Thus, no two nodes in $S$ are joined by an edge $\Rightarrow S$ independent set.
Vertex Cover and Independent Set

$S$ is an independent set iff $V - S$ is a vertex cover.

This shows:

\[
\text{VERTEX-COVER} \leq_P \text{INDEPENDENT-SET} \quad \text{and} \quad \text{INDEPENDENT-SET} \leq_P \text{VERTEX-COVER}.
\]

from which we conclude:

\[
\text{VERTEX-COVER} \equiv_P \text{INDEPENDENT-SET}.
\]
Reduction from Special Case to General Case

In this case we do not prove $X \equiv_p Y$, we prove $X \leq_p Y$

Eg: Vertex cover versus set cover

Vertex cover is phrased in terms of graphs

Set cover is phrased, more generally, in terms of sets
Set Cover

**SET COVER:** Given a set $U$ of elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of $U$, and an integer $k$, is there a collection of at most $k$ of these sets whose union is equal to $U$?

Example:

<table>
<thead>
<tr>
<th>$U$</th>
<th>${ 1, 2, 3, 4, 5, 6, 7 }$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>2</td>
</tr>
<tr>
<td>$S_1$</td>
<td>${3, 7}$</td>
</tr>
<tr>
<td>$S_2$</td>
<td>${3, 4, 5, 6}$</td>
</tr>
<tr>
<td>$S_3$</td>
<td>${1}$</td>
</tr>
<tr>
<td>$S_4$</td>
<td>${2, 4}$</td>
</tr>
<tr>
<td>$S_5$</td>
<td>${5}$</td>
</tr>
<tr>
<td>$S_6$</td>
<td>${1, 2, 6, 7}$</td>
</tr>
</tbody>
</table>
Vertex Cover Reduces to Set Cover

Claim. VERTEX-COVER \leq_p SET-COVER.

Proof. Given a VERTEX-COVER instance \( G = (V, E), k \), we reduce it to a set cover instance with \( U=E \) and a subset \( S_v \) for each \( v \) in \( V \):

- Create SET-COVER instance:
  - \( U = E \), \( S_v = \{ e \in E : e \text{ incident to } v \} \)
- Set-cover of size \( \leq k \) iff vertex cover of size \( \leq k \).
- Hence we have shown that VERTEX-COVER \leq_p SET-COVER.
Satisfiability

**Literal:** A Boolean variable or its negation. \[ x_i \text{ or } \overline{x_i} \quad (\overline{x_i} = \text{not } x_i) \]

**Clause:** A disjunction of literals. \[ C_j = x_1 \lor x_2 \lor x_3 \]

**Conjunctive normal form (CNF):**
A propositional formula \( \Phi \) that is a conjunction of clauses. \[ \Phi = C_1 \land C_2 \land C_3 \land C_4 \]

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

**3-SAT:** SAT where each clause contains exactly 3 literals.

Example: \[ \left( \overline{x_1} \lor x_2 \lor x_3 \right) \land \left( x_1 \lor \overline{x_2} \lor x_3 \right) \land \left( x_2 \lor x_3 \right) \land \left( \overline{x_1} \lor \overline{x_2} \lor \overline{x_3} \right) \]

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

What is a necessary and sufficient condition for a CNF to be true?

Each clause must have one literal evaluating to true.
3-SAT Reduces to Independent Set

Claim. 3-SAT ≤<sub>p</sub> INDEPENDENT-SET.

Proof. Given an instance Φ of 3-SAT, we construct an instance (G, k) of INDEPENDENT-SET that has an independent set of size k iff Φ is satisfiable.

Construction.
- G contains 3 vertices for each clause, one for each literal.
- Connect 3 literals in a clause in a triangle.
- Connect literal to each of its negations.
- We call these connected triangles gadgets, creating a useful instance of independent set.

\[
\Phi = (\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_2 \lor x_4)
\]
3-SAT Reduces to Independent Set

Claim. \( G \) contains independent set of size \( k = |\Phi| \) iff \( \Phi \) is satisfiable.

Proof. \( \Rightarrow \) Let \( S \) be independent set of size \( k \).

1) \( S \) must contain exactly one vertex in each triangle, and cannot contain both \( x_i \) and \( \bar{x}_i \) (they are connected)
2) Set these literals to true, there are no conflicts, because (1)
3) All clauses are satisfied.

\( \Leftarrow \) Given satisfying assignment, select one true literal from each triangle. This is an independent set of size \( k \). 

\[
\Phi = (\bar{x}_1 \lor x_2 \lor x_3) \land (x_1 \lor \bar{x}_2 \lor x_3) \land (\bar{x}_1 \lor x_2 \lor x_4)
\]
Circuit Satisfiability

**CIRCUIT-SAT.** A combinational circuit is a directed acyclic graph built out of \textsc{and}, \textsc{or}, and \textsc{not} nodes. Given such circuit, is there a way to set the circuit inputs so that the output is 1?

\[
\text{output} \\
\land \\
\neg \\
\land \\
\land \\
\lor \\
\lor \\
\lor
\]

\[1 \ 0 \ 1\]

**yes: 1 0 1**
CIRCUIT-SAT ≤ₚ 3-SAT

- Let K be any circuit.
- Create a 3-SAT variable $x_i$ for each circuit element $i$.
- Make 3-SAT clauses compute values for each circuit-SAT node, eg:
  - $x_2 = \neg x_3 \Rightarrow$ add 2 clauses: $x_2 \lor \underline{x_3}, \underline{x_2} \lor \underline{x_3}$
  - $x_1 = x_4 \lor x_5 \Rightarrow$ add 3 clauses: $x_1 \lor x_4, \underline{x_1} \lor \underline{x_5}, x_1 \lor x_4 \lor \underline{x_5}$
  - $x_0 = x_1 \land x_2 \Rightarrow$ add 3 clauses: $x_0 \lor x_1, x_0 \lor x_2, x_0 \lor x_1 \lor \underline{x_2}$

- Hard-coded input values and output value.
  - $x_5 = 0 \Rightarrow$ add 1 clause: $\underline{x_5}$
  - $x_0 = 1 \Rightarrow$ add 1 clause: $x_0$

- Final step: turn clauses of length < 3 into clauses of length exactly 3. **HOW?**
Independent Set $\leq_p$ CircSAT

To show that any Independent Set$_k$ problem $\leq_p$ CircSAT we construct a circuit $C$ with inputs and one output:

inputs:
1: **graph description**, in terms of all its edges 
   (1 bit per node pair)
2: **set description**, in terms of the nodes in the set 
   (1 bit per node)

output: one output bit: yes/no

The components of the circuit:
1: define the graph in terms of the edges it contains
2: check whether the set has a pair of nodes that is an edge in the graph.
   **If here is such a pair, the set is not independent**
3: count whether there are at least $k$ nodes in the set
Independent Set $\leq_p$ CircSAT

**Example.** Construction below creates a circuit $C$ whose inputs can be set so that $C$ outputs true iff graph $G$ has an independent set of size $\geq 2$.

$G = (V, E), n = 3$

Try it for $\{u, v\}$

Try it for $\{u, w\}$

hard-coded inputs (graph description)  n inputs (nodes in independent set)
Basic reduction strategies.

- Equivalence: \textsc{Independent-Set} $\equiv_p \textsc{Vertex-Cover}$. 
- Special case to general case: \textsc{Vertex-Cover} $\leq_p \textsc{Set-Cover}$. 
- Encoding with gadgets: 3-SAT $\leq_p \textsc{Independent-Set}$. 

Transitivity. If $X \leq_p Y$ and $Y \leq_p Z$, then $X \leq_p Z$.

Proof idea. Compose the two algorithms.

Example:

3-SAT $\leq_p \textsc{Independent-Set} \leq_p \textsc{Vertex-Cover} \leq_p \textsc{Set-Cover}$. 