Coping With NP-Completeness

Q. Suppose I need to solve an NP-complete problem. What should I do?

A. Theory says you’re unlikely to find poly-time algorithm.

Must sacrifice one of three desired features:

- Solve problem to optimality.
- Solve problem in polynomial time.
- Solve arbitrary instances of the problem.

This lecture. Solve some special cases of NP-complete problems that arise in practice.
**Finding Small Vertex Covers**

**Claim.** Let $u-v$ be an edge of $G$. $G$ has a vertex cover of size $\leq k$ iff at least one of $G - \{u\}$ and $G - \{v\}$ has a vertex cover of size $\leq k-1$.

**Pf.**
- Suppose $G$ has a vertex cover $S$ of size $\leq k$.
  - $S$ contains either $u$ or $v$ (or both). Assume it contains $u$.
  - $S - \{u\}$ is a vertex cover of $G - \{u\}$.

**Claim.** If $G$ has a vertex cover of size $k$, it has $\leq k(n-1)$ edges.

**Pf.** Each vertex covers at most $n-1$ edges.

**Finding Small Vertex Covers: Algorithm**

**Claim.** The following algorithm determines if $G$ has a vertex cover of size $\leq k$ in $O(2^k \cdot k^n)$ time.

```
boolean Vertex-Cover(G, k) {
    if (G contains no edges)   return true
    if (G contains $\geq kn$ edges) return false
    let (u, v) be any edge of G
    a = Vertex-Cover(G - \{u\}, k-1)
    b = Vertex-Cover(G - \{v\}, k-1)
    return a or b
}
```

**Finding Small Vertex Covers: Recursion Tree**

**Inductive Step:**

$T(n,k) \leq 2 \times T(n,k-1) + ckn$

$= 2^k ckn - 2^k cn + ckn$

$\leq 2^k ckn$
**Vertex Cover**

Vertex cover. Given an undirected graph $G = (V, E)$, a vertex cover is a subset of vertices $S \subseteq V$ such that for each edge $(u, v) \in E$, either $u \in S$ or $v \in S$ or both.

$S = \{3, 4, 5, 1', 2'\}$

$|S| = 5$

**Weak duality.** Let $M$ be a matching, and let $S$ be a vertex cover.

Then, $|M| \leq |S|$.

**Pf.** Each vertex can cover at most one edge in any matching.

$M = 1-2', 3-1', 4-5'$

$|M| = 3$

**König-Egerváry Theorem.** In a bipartite graph, the max cardinality of a matching is equal to the min cardinality of a vertex cover.

$S^* = \{3, 1', 2', 5\}$

$|S^*| = 4$

$M^* = 1-1', 2-2', 3-3', 5-5'$

$|M^*| = 4$

**Proof of König-Egerváry Theorem.**

- Suffices to find matching $M$ and cover $S$ such that $|M| = |S|$.
- Formulate max flow problem as for bipartite matching.
- Let $M$ be max cardinality matching and let $(A, B)$ be min cut.

**Proof of König-Egerváry Theorem.**

- Suffices to find matching $M$ and cover $S$ such that $|M| = |S|$.
- Formulate max flow problem as for bipartite matching.
- Let $M$ be max cardinality matching and let $(A, B)$ be min cut.
Vertex Cover: Proof of König-Egerváry Theorem

König-Egerváry Theorem. In a bipartite graph, the max cardinality of a matching is equal to the min cardinality of a vertex cover.

- Suffices to find matching \( M \) and cover \( S \) such that \( |M| = |S| \).
- Formulate max flow problem as for bipartite matching.
- Let \( M \) be max cardinality matching and let \((A, B)\) be min cut.

Define \( L_A = L \cap A, L_B = L \cap B, R_A = R \cap A, R_B = R \cap B \).

- **Claim 1.** \( S = L_B \cup R_A \) is a vertex cover.
  - Consider \((u, v) \in E\)
  - \( u \in L_A \subset A \) then \((s, u)\) contributes 0 to \( \text{cap}(A, B) \)
  - If \( v \in R_B \subset B \) then \((v, t)\) contributes 0 to \( \text{cap}(A, B) \)
  - \( |M| = \text{cap}(A, B) = |L_B| + |R_A| = |S| \).

- **Claim 2.** \( |S| = |M| \).
  - max-flow min-cut theorem \( \Rightarrow |M| = \text{cap}(A, B) \)
  - only edges of form \((s, u)\) or \((v, t)\) contribute to \( \text{cap}(A, B) \)
  - If \( u \in L_A \subset A \) then \((s, u)\) contributes 0 to \( \text{cap}(A, B) \)
  - If \( v \in R_B \subset B \) then \((v, t)\) contributes 0 to \( \text{cap}(A, B) \)
  - \( |M| = \text{cap}(A, B) = |L_B| + |R_A| = |S| \).

Independent Set on Trees

Independent set on trees. Given a tree, find a maximum cardinality subset of nodes such that no two share an edge.

**Fact.** A tree on at least two nodes has at least two leaf nodes.

Computational complexity of independent set can be solved in linear time on trees.

**Key Observation.** If \( v \) is a leaf, there exists a maximum size independent set containing \( v \).

**Pf.** (exchange argument)
- Consider a max cardinality independent set \( S \).
- If \( v = S \), we’re done.
- If \( u \in S \) and \( v \notin S \), then \( S \cup \{v\} \) is independent \( \Rightarrow S \) not maximum.
- If \( u \in S \) and \( v \notin S \), then \( S \cup \{v\} \) is independent.
Independent Set on Trees: Greedy Algorithm

**Theorem.** The following greedy algorithm finds a maximum cardinality independent set in forests (and hence trees).

**Algorithm:** $\text{Independent-Set-In-A-Forest}(F) \{ $

$\text{while} (F \text{ has at least one edge} ) $ \{

$\text{Let } e = (u,v) \text{ be an edge such that } v \text{ is a leaf} $ \{

$\text{Add } v \text{ to } S $ \{

$\text{Delete from } F \text{ nodes } u \text{ and } v \text{, and all edges}$

$\text{incident to them.} $ \{

$\text{return } S \cup V(F) $ //S and all remaining nodes in F

$\}$

$\}$

**Pf.** Correctness follows from the previous key observation. □

**Remark.** Can implement in $O(n)$ time by considering nodes in postorder.

**Weighted Independent Set on Trees**

**Weighted independent set on trees.** Given a tree and node weights $w_v > 0$, find an independent set $S$ that maximizes $\sum_{v \in S} w_v$.

**Observation.** If $(u, v)$ is an edge such that $v$ is a leaf node, then either $\text{OPT}$ includes $u$, or it includes all leaf nodes incident to $u$.

**Dynamic programming solution.** Root tree at some node, say $r$.

- $\text{OPT}_\text{in}(u) = \max \text{ weight independent set of subtree rooted at } u$, containing $u$.
- $\text{OPT}_\text{out}(u) = \max \text{ weight independent set of subtree rooted at } u$, not containing $u$.

- $\text{OPT}_\text{in}(u) = w_u + \sum_{v \in \text{children}(u)} \text{OPT}_\text{out}(v)$
- $\text{OPT}_\text{out}(u) = \max \{ \text{OPT}_\text{in}(v), \text{OPT}_\text{out}(v) \}$

**Register Allocation**

**Register.** One of $k$ of high-speed memory locations in computer’s CPU.

**Register allocator.** Part of an optimizing compiler that controls which variables are saved in the registers as compiled program executes.

**Interference graph.** Nodes are “live ranges.” Edge $u-v$ if there exists an operation where both $u$ and $v$ are “live” at the same time.

**Observation.** [Chaitin, 1982] Can solve register allocation problem if interference graph is $k$-colorable.

**Spilling.** If graph is not $k$-colorable (or we can’t find a $k$-coloring), we “spill” certain variables to main memory and swap back as needed.
A Useful Property

Remark. Register allocation problem is NP-hard.

Key fact. If a node $v$ in graph $G$ has fewer than $k$ neighbors, $G$ is $k$-colorable iff $G - \{v\}$ is $k$-colorable.

Pf. Delete node $v$ from $G$ and color $G - \{v\}$.

- If $G - \{v\}$ is not $k$-colorable, then neither is $G$.
- If $G - \{v\}$ is $k$-colorable, then there is at least one remaining color left for $v$.

\[
\begin{bmatrix}
 & & k = 3 & & \\
 & & & & \\
G & is & 3-colorable & even & though
\]
all nodes have degree 2.

\[
\begin{bmatrix}
 & & k = 2 & & \\
 & & & & \\
\end{bmatrix}
\]

Chaitin’s Algorithm

Vertex-Color($G$, $k$) {

while ($G$ is not empty) {

Pick a node $v$ with fewer than $k$ neighbors
Push $v$ on stack
Delete $v$ and all its incident edges
}

while (stack is not empty) {

Pop next node $v$ from the stack
Assign $v$ a color different from its neighboring nodes which have already been colored
}

}\}

Theorem. [Kempe 1879, Chaitin 1982] Chaitin’s algorithm produces a $k$-coloring of any graph with max degree $k-1$.

Pf. Follows from key fact since each node has fewer than $k$ neighbors.

Remark. If algorithm never encounters a graph where all nodes have degree $\geq k$, then it produces a $k$-coloring.

Practice. Chaitin’s algorithm (and variants) are extremely effective and widely used in real compilers for register allocation.

3-Colorability

**3-COLOR**. Given an undirected graph $G$ does there exists a way to color the nodes red, green, and blue so that no adjacent nodes have the same color?

Register Allocation

Register allocation. Assign program variables to machine register so that no more than $k$ registers are used and no two program variables that are needed at the same time are assigned to the same register.

Interference graph. Nodes are program variables names, edge between $u$ and $v$ if there exists an operation where both $u$ and $v$ are “live” at the same time.

Observation. [Chaitin 1982] Can solve register allocation problem iff interference graph is $k$-colorable.

Fact. 3-COLOR $\leq_k$ $k$-REGISTER-ALLOCATION for any constant $k \geq 3$. 

8.7 Graph Coloring

Basic genres.

- Packing problems: SET-PACKING, INDEPENDENT SET.
- Covering problems: SET-COVER, VERTEX-COVER.
- Constraint satisfaction problems: SAT, 3-SAT.
- Sequencing problems: HAMILTONIAN CYCLE, TSP.
- Partitioning problems: 3D-MATCHING, 3-COLOR.
- Numerical problems: SUBSET-SUM, KNAPSACK.
### 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. For each literal, create a node.
2. Create 3 new nodes T, F, B; connect them in a triangle, and connect each literal to B.
3. Connect each literal to its negation.
4. For each clause, add gadget of 6 nodes and 13 edges.

To be described next.

---

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

**Pf.**

\( \Rightarrow \) 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.

---

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

**Pf.**

\( \Rightarrow \) 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.

---

**10.3 Circular Arc Coloring**
Wavelength-Division Multiplexing

Wavelength-division multiplexing (WDM). Allows m communication streams (arcs) to share a portion of a fiber optic cable, provided they are transmitted using different wavelengths.

Ring topology. Special case is when network is a cycle on n nodes.

Bad news. NP-complete, even on rings.

Brute force. Can determine if k colors suffice in $O(km)$ time by trying all k-colorings.

Goal. $O(f(k) \cdot \text{poly}(m, n))$ on rings.

Review: Interval Coloring

Interval coloring. Greedy algorithm finds coloring such that number of colors equals depth of schedule.

Circular arc coloring.
- Weak duality: number of colors $\geq$ depth.
- Strong duality does not hold.

(Almost) Transforming Circular Arc Coloring to Interval Coloring

Circular arc coloring. Given a set of n arcs with depth $d \leq k$, can the arcs be colored with k colors?

Equivalent problem. Cut the network between nodes $v_1$ and $v_n$. The arcs can be colored with k colors if the intervals can be colored with k colors in such a way that “sliced” arcs have the same color.

Circular Arc Coloring: Dynamic Programming Algorithm

Dynamic programming algorithm:
- Assign distinct color to each interval which begins at cut node $v_j$.
- At each node $v_i$, some intervals may finish, and others may begin.
- Enumerate all k-colorings of the intervals through $v_i$ that are consistent with the colorings of the intervals through $v_j$.
- The arcs are k-colorable if some coloring of intervals ending at cut node $v_j$ is consistent with original coloring of the same intervals.

Circular Arc Coloring: Running Time

Running time. $O(k! \cdot n)$.
- n phases of the algorithm.
- Bottleneck in each phase is enumerating all consistent colorings.
- There are at most k intervals through $v_i$, so there are at most $k^i$ colorings to consider.

Remark. This algorithm is practical for small values of k (say k = 10) even if the number of nodes n (or paths) is large.
8.10 A Partial Taxonomy of Hard Problems

Subset Sum (proof from book)

Construction. Let \( X \cup Y \cup Z \) be a instance of 3D-MATCHING with triplet set \( T \). Let \( n = |X| = |Y| = |Z| \) and \( m = |T| \).

- Let \( X = \{ x_1, x_2, x_3, x_4 \}, Y = \{ y_1, y_2, y_3, y_4 \}, Z = \{ z_1, z_2, z_3, z_4 \} \)
- For each triplet \( (x_i, y_j, z_k) \in T \), create an integer \( w_{ij} \) with \( 3n \) digits that has a 1 in positions \( i, n+j, 2n+k \).

Claim. 3D-matching iff some subset sums to \( W = 111, \ldots, 111 \).

Partition

\( \text{SUBSET-SUM} \). Given natural numbers \( w_1, \ldots, w_n \) and an integer \( W \), is there a subset that adds up to exactly \( W \)?

\( \text{PARTITION} \). Given natural numbers \( v_1, \ldots, v_m \), can they be partitioned into two subsets that add up to the same value?

Claim. \( \text{SUBSET-SUM} \leq P \text{ PARTITION} \).

Proof. Let \( W, w_1, \ldots, w_n \) be an instance of \( \text{SUBSET-SUM} \).

- Create instance of \( \text{PARTITION} \) with \( m = n+2 \) elements
  - \( v_1 = w_1, v_2 = w_2, \ldots, v_n = w_n, v_{n+1} = 2W - W = W, v_{n+2} = 2W - W = W \)

- There exists a subset that sums to \( W \) if and only if there exists a partition since two new elements cannot be in the same partition.
4 Color Theorem

Planar 3-Colorability

PLANAR-3-COLOR. Given a planar map, can it be colored using 3 colors so that no adjacent regions have the same color?

YES instance.

Planar 3-Colorability

NO instance.

Def. A graph is planar if it can be embedded in the plane in such a way that no two edges cross.

Applications: VLSI circuit design, computer graphics.

Kuratowski’s Theorem. An undirected graph $G$ is non-planar iff it contains a subgraph homeomorphic to $K_5$ or $K_{3,3}$.

Planarity

Planarity testing. [Hopcroft-Tarjan 1974] $O(n)$.

simple planar graph can have at most $3n$ edges

Remark. Many intractable graph problems can be solved in poly-time if the graph is planar; many tractable graph problems can be solved faster if the graph is planar.

Planarity Testing

Planar Graph 3-Colorability

Q. Is this planar graph 3-colorable?
### Planar 3-Colorability and Graph 3-Colorability

**Claim.** PLANAR-3-COLOR \( \leq_P \) PLANAR-GRAPH-3-COLOR.

**Pf sketch.** Create a vertex for each region, and an edge between regions that share a nontrivial border.

### Planar Graph 3-Colorability

**Claim.** W is a planar graph such that:
- In any 3-coloring of W, opposite corners have the same color.
- Any assignment of colors to the corners in which opposite corners have the same color extends to a 3-coloring of W.

**Pf.** Only 3-colorings of W are shown below (or by permuting colors).

### Planar Graph 3-Colorability

**Claim.** W is a planar graph such that:
- In any 3-coloring of W, opposite corners have the same color.
- Any assignment of colors to the corners in which opposite corners have the same color extends to a 3-coloring of W.

**Pf.** Given instance of 3-COLOR, draw graph in plane, letting edges cross.
- Replace each edge crossing with planar gadget W.
- In any 3-coloring of W, \( a \neq a' \) and \( b \neq b' \).
- If \( a = a' \) and \( b = b' \) then can extend to a 3-coloring of W.

### Planar k-Colorability

**PLANAR-2-COLOR.** Solvable in linear time.

**PLANAR-3-COLOR.** NP-complete.

**PLANAR-4-COLOR.** Solvable in \( O(1) \) time.

**Theorem.** [Appel-Haken, 1976] Every planar map is 4-colorable.
- Resolved century-old open problem.
- Used 50 days of computer time to deal with many special cases.
- First major theorem to be proved using computer.

**False intuition.** If PLANAR-3-COLOR is hard, then so is PLANAR-4-COLOR and PLANAR-5-COLOR.

### Polynomial-Time Detour

**Graph minor theorem.** [Robertson-Seymour 1980s]

**Corollary.** There exist an \( O(n^3) \) algorithm to determine if a graph can be embedded in the torus in such a way that no two edges cross.

**Pf of theorem.** Tour de force.
**Graph minor theorem.** [Robertson-Seymour 1980s]

**Corollary.** There exist an $O(n^3)$ algorithm to determine if a graph can be embedded in the torus in such a way that no two edges cross.

**Mind boggling fact 1.** The proof is highly non-constructive!

**Mind boggling fact 2.** The constant of proportionality is enormous!

Unfortunately, for any instance $G = (V, E)$ that one could fit into the known universe, one would easily prefer $n^{70}$ to even constant time, if that constant had to be one of Robertson and Seymour’s. - David Johnson

**Theorem.** There exists an explicit $O(n)$ algorithm.

**Practice.** LEDA implementation guarantees $O(n^3)$. 

---

**Polynomial-Time Detour**