10. **Extending Tractability**

- finding small vertex covers
- solving NP-hard problems on trees
- circular arc coverings
- vertex cover in bipartite graphs
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.
10. **Extending Tractability**

- finding small vertex covers
- solving NP-hard problems on trees
- circular arc coverings
- vertex cover in bipartite graphs
**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 $(u, v)$ either $u \in S$ or $v \in S$ or both?

$S = \{3, 6, 7, 10\}$ is a vertex cover of size $k = 4$
Finding small vertex covers

Q. \textsc{VERTEXCOVER} is NP-complete. But what if $k$ is small?

Brute force. $O(kn^{k+1})$.
• Try all $C(n, k) = O(n^k)$ subsets of size $k$.
• Takes $O(kn)$ time to check whether a subset is a vertex cover.

Goal. Limit exponential dependency on $k$, say to $O(2^k kn)$.

Ex. $n = 1,000, k = 10$.

Brute. $kn^{k+1} = 10^{34} \implies$ infeasible.
Better. $2^k kn = 10^7 \implies$ feasible.

Remark. If $k$ is a constant, then the algorithm is poly-time; if $k$ is a small constant, then it's also practical.
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. \(\Rightarrow\)
- 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 \(G - \{u\}\).

Pf. \(\Leftarrow\)
- Suppose \(S\) is a vertex cover of \(G - \{u\}\) of size \(\leq k - 1\).
- Then \(S \cup \{u\}\) is a vertex cover of \(G\). \(\blacksquare\)

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. \(\blacksquare\)
Finding small vertex covers: algorithm

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

```java
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
}
```

Pf.

- Correctness follows from previous two claims.
- There are $\leq 2^{k+1}$ nodes in the recursion tree; each invocation takes $O(kn)$ time. □
Finding small vertex covers: recursion tree

\[ T(n, k) \leq \begin{cases} 
  c & \text{if } k = 0 \\
  cn & \text{if } k = 1 \\
  2T(n, k-1) + ckn & \text{if } k > 1 
\end{cases} \Rightarrow T(n, k) \leq 2^k c k n \]
10. Extending Tractability

- finding small vertex covers
- solving NP-hard problems on trees
- circular arc coverings
- vertex cover in bipartite graphs

Section 10.2
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.

**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 $v \in S$, we're done.
- If $u \notin S$ and $v \notin S$, then $S \cup \{v\}$ is independent $\Rightarrow S$ not maximum.
- If $u \in S$ and $v \notin S$, then $S \cup \{v\} - \{u\}$ is independent. □
Independent set on trees: greedy algorithm

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

```
Independent-Set-In-A-Forest(F) {
    S ← φ
    while (F has at least one edge) {
        Let e = (u, v) be an edge such that v is a leaf
        Add v to S
        Delete from F nodes u and v, and all edges incident to them.
    }
    return S
}
```

**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 $OPT$ includes $u$ or $OPT$ includes all leaf nodes incident to $u$.

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

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

\[
OPT_{in}(u) = w_u + \sum_{v \in \text{children}(u)} OPT_{out}(v)
\]

\[
OPT_{out}(u) = \sum_{v \in \text{children}(u)} \max \{OPT_{in}(v), OPT_{out}(v)\}
\]

$\text{children}(u) = \{v, w, x\}$
Weighted independent set on trees: dynamic programming algorithm

**Theorem.** The dynamic programming algorithm finds a maximum weighted independent set in a tree in $O(n)$ time.

```c
Weighted-Independent-Set-In-A-Tree(T) {
    Root the tree at a node r
    foreach (node u of T in postorder) {
        if (u is a leaf) {
            $M_{in}[u] = w_u$
            $M_{out}[u] = 0$
        }
        else {
            $M_{in}[u] = w_u + \sum_{v \in \text{children}(u)} M_{out}[v]$
            $M_{out}[u] = \sum_{v \in \text{children}(u)} \max(M_{in}[v], M_{out}[v])$
        }
    }
    return $\max(M_{in}[r], M_{out}[r])$
}
```

- ensures a node is visited after all its children
- can also find independent set itself (not just value)
**Independent set on trees.** This structured special case is tractable because we can find a node that breaks the communication among the subproblems in different subtrees.

**Graphs of bounded tree width.** Elegant generalization of trees that:
- Captures a rich class of graphs that arise in practice.
- Enables decomposition into independent pieces.

*see Chapter 10.4 (but proceed with caution)*
10. **Extending Tractability**

- finding small vertex covers
- solving NP-hard problems on trees
- circular arc coverings
- vertex cover in bipartite graphs

Section 10.3
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 $O(k^n)$ time by trying all $k$-colorings.

Goal. $O(f(k)) \cdot poly(m, n)$ 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 iff 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 $v_0$.
- 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 $v_{i-1}$.
- The arcs are $k$-colorable iff some coloring of intervals ending at cut node $v_0$ is consistent with original coloring of the same intervals.
Circular arc coloring: running time

Running time. $O(k! \cdot n)$.

- The algorithm has $n$ phases.
- Bottleneck in each phase is enumerating all consistent colorings.
- There are at most $k$ intervals through $v_i$, so there are at most $k!$ 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.
10. Extending Tractability

- finding small vertex covers
- solving NP-hard problems on trees
- circular arc coverings
- vertex cover in bipartite graphs
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 $(u, v)$ either $u \in S$ or $v \in S$ or both?

vertex cover $S = \{3, 4, 5, 1', 2'\}$
Vertex cover and matching

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.
Vertex cover in bipartite graphs: König-Egerváry Theorem

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

Matching $M$: $1-1'$, $2-2'$, $3-4'$, $4-5'$

Vertex cover $S = \{3, 4, 5, 1', 2'\}$
Proof of König-Egerváry theorem

**Theorem.** [König-Egerváry] 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.
Proof of König-Egerváry theorem

**Theorem.** [König-Egerváry] 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 $(u, v) \in E$
  - $u \in L_A, v \in R_B$ impossible since infinite capacity
  - thus, either $u \in L_B$ or $v \in R_A$ or both

- **Claim 2.** $|M| = |S|$.
  - max-flow min-cut theorem $\Rightarrow |M| = cap(A, B)$
  - only edges of form $(s, u)$ or $(v, t)$ contribute to $cap(A, B)$
  - $|M| = cap(A, B) = |L_B| + |R_A| = |S|$. $\blacksquare$