The Design and Analysis of Algorithms

Lecture 19 Extending the Limits of Tractability

Zhenbo Wang

Department of Mathematical Sciences, Tsinghua University

Content

Coping With NP-Completeness

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

A. Theory says it is unlikely to find a poly-time algorithm.

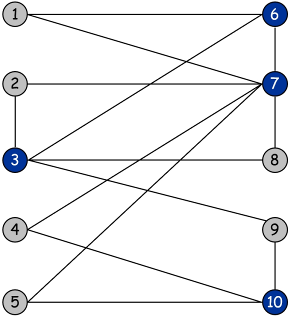
Practice. Must sacrifice one of three desired features.

1. Solve *arbitrary instances* of the problem.
2. Solve problem to optimality.
3. Solve problem in polynomial time.

- This lecture focuses on solving some special cases of NP-complete problems that arise in practice.

Vertex Cover

Def. 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.



k = 4
S = { 3, 6, 7, 10 }

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 $k - 1$.

Pf. "$\Rightarrow$" Suppose G has a vertex cover of size $\leq k$.

S contains either u or v (or both). Assume it contains u.

$G - u$ is a vertex cover of $G - u$.

"$\Leftarrow$" Suppose S is a a vertex cover of size $\leq k - 1$.

Then $S \cup u$ is a vertex cover of G. $\square$

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. $\square$

Finding Small Vertex Covers

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

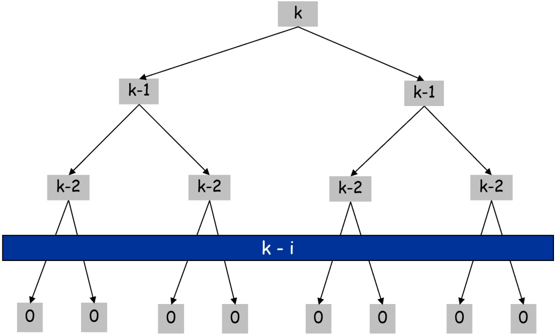
Vertex-Cover (G, k)

if G contains no edges **then**
    return true
else if G contains $\geq kn$ edges **then**
    return false
else
    let (u, v) be any edge of G
    $a = Vertex - Cover(G - u, k - 1)$
    $b = Vertex - Cover(G - v, k - 1)$
    return $a \vee b$
end if

Pf. Correctness follows previous two claims.

Recursion Tree

$$T(n,k) \le \begin{cases} cn & \text{if } k = 1 \\ 2T(n,k-1) + cnk & \text{if } k > 1 \end{cases} \Rightarrow T(n,k) \le 2^k ckn$$



There are $\le 2^{k+1}$ nodes in the recursion tree; each invocation takes $O(kn)$ time. □

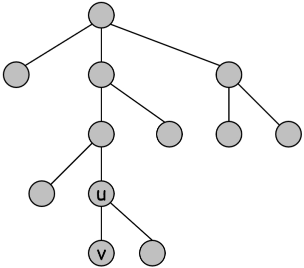
Independent Set on Trees

Independent Set on Trees

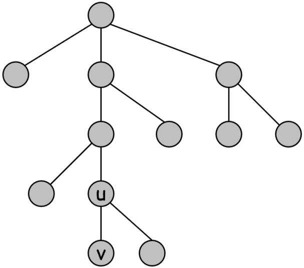
Given a tree, find a maximum cardinality subset of nodes such that no two share an edge.

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.



Independent Set on Trees



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. $\square$

Greedy Algorithm

Theorem 1

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

Independent-Set-In-A-Forest (F)

$S \leftarrow \emptyset$
while F has at least one edge **do**
    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.
end while
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 $\sum_{v \in S} w_v$.

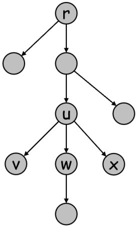
Observation 1

If (u, v) is an edge such that v is a leaf node, then either OPT includes u, or it includes all leaf nodes incident to u.

- *Dynamic programming solution.* Root tree at some node r.
- $OPT_{in}(u) =$ max weight independent set rooted at u, containing u.
- $OPT_{out}(u) =$ max weight independent set rooted at u, not containing u.

Weighted Independent Set on Trees

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



children(u) = { v, w, x }

Theorem 2

The dynamic programming algorithm find a maximum weighted independent set in trees in $O(n)$ time.

Greedy Algorithm

Weighted-Independent-Set-In-A-Forest (F)

Root the tree at a node r
for node u of T in postorder **do**
    if u is a leaf **then**
        $M_{in}[u] = w_u$
        $M_{out}[u] = 0$
    else
        $M_{in}[u] = w_u + \sum_{v \in \text{children}(u)} OPT_{out}(v)$
        $M_{out}[u] = \sum_{v \in \text{children}(u)} \max\{OPT_{in}(v), OPT_{out}(v)\}$
    end if
end for
return $\max\{M_{in}[r], M_{out}[r]\}$

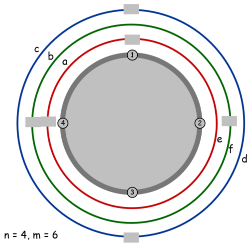
Pf. Takes $O(n)$ time since we visit nodes in postorder and examine each edge exactly once. □

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.

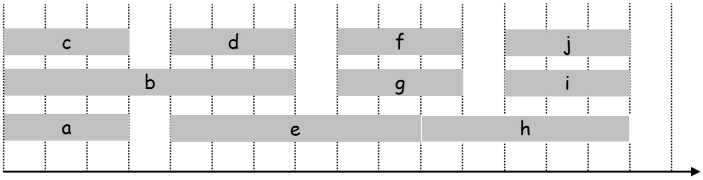
- *Ring topology.* Special case is when network is a cycle on n nodes.
- *Bad news.* NP-complete, even on rings.

- *Brute force.* Can determine if k colors suffice in $O(k^m)$ time by trying all k-colorings.
- *Goal.* $O(f(k) \cdot poly(m, n))$ on rings.



Interval Coloring

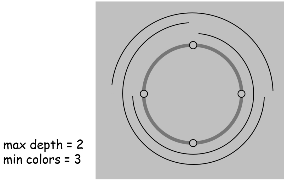
Greedy algorithm finds coloring such that number of colors equals depth of schedule (maximum number of streams at one location).



- Circular arc coloring.

  Weak duality: number of colors $\geq$ depth.

  Strong duality does not hold.

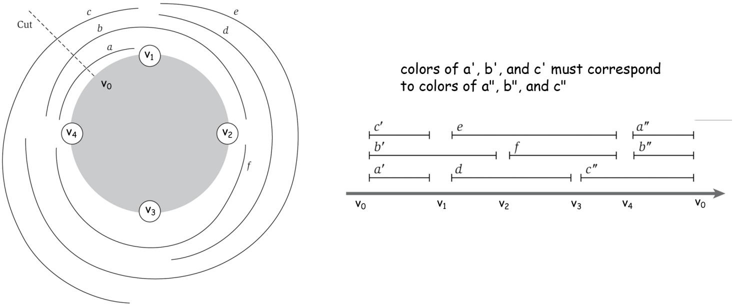


Transforming Circular Arc Coloring to Interval Coloring

Circular arc coloring

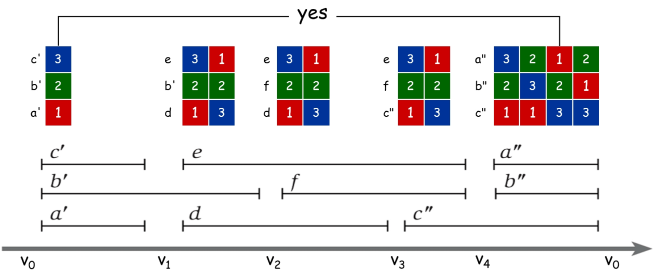
Given a set of n arcs with depth $d \leq k$, can the arcs be colored with k colors?

- *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.



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: $O(k! \cdot n)$.

  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 $k!$ 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.

Homework

- Read Chapter 10 of the textbook.
- Exercises 5 & 7 in Chapter 10.