

Algorithm Design and Analysis

CSE
565

LECTURES 38, 39 Extending the Limits of Tractability

- Finding Small Vertex Covers
- Solving NP-hard Graph Problems on Trees

Sofya Raskhodnikova

11/28/2007

S. Raskhodnikova; based on slides by K. Wayne

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.

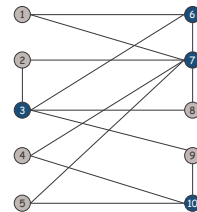
This lecture. Solve some special cases of NP-complete problems that arise in practice.

Finding Small Vertex Covers

Vertex Cover

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

Problem: Given a graph $G = (V, E)$ and an integer k , find a vertex cover of size $\leq k$ if it exists.



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

Finding Small Vertex Covers

Q. What if k is small?

Brute force. $O(k n^{k-1})$.

- 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 , e.g., to $O(2^k n)$.

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

Brute force. $k n^{k-1} = 10^{34} \Rightarrow$ infeasible.

Better. $2^k n = 10^6 \Rightarrow$ feasible.

Remark. If k is a constant, algorithm is poly-time; if k is a small constant, then it's also practical.

Important. The algorithm is still exponential, and hence scales badly (e.g., consider $k=40$). However, it is better than brute force.

Finding Small Vertex Covers

Claim 1. 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 $\leq k-1$.

delete v and all incident edges

Pf. \Rightarrow

- Suppose G has a vertex cover S of size $\leq k$.
- S contains either u or v (or both). WLOG 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 .

Finding Small Vertex Covers: Algorithm

Claim 2. The following algorithm find a vertex cover of size $\leq k$ in G if it exists and runs in $O(2^k n)$ time.

```

SmallVC(G, k) {
  if k=0
    if (G contains no edges) return {}
    else return false

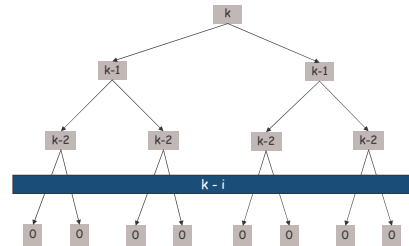
  let (u, v) be any edge of G
  S_u = SmallVC(G - {u}, k-1)
  if S_u ≠ false return S_u ∪ {u}

  S_v = SmallVC(G - {v}, k-1)
  if S_v ≠ false return S_v ∪ {v}
  else return false
}
    
```

- Pf.
- Correctness follows from Claim 1.
 - There are $\leq 2^{k+1}$ nodes in the recursion tree: each invocation takes $O(n)$ time. •

Finding Small Vertex Covers: Recursion Tree

$$T(n, k) \leq \begin{cases} cn & \text{if } k=0 \\ 2T(n, k-1) + cn & \text{if } k > 0 \end{cases} \Rightarrow T(n, k) \leq 2^{k+1} cn$$



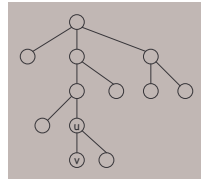
Solving NP-Hard Graph Problems on Trees

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.

↙ degree = 1



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 $u \in S$ and $v \notin S$, then $S \cup \{v\}$ is independent $\Rightarrow S$ not maximum.
 - If $u \in S$ and $v \in S$, 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 ← ∅
  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 ∪ {remaining nodes in F}
}
    
```

- Pf. Correctness follows from the previous key observation. •
 Remark. Can implement in $O(n)$ time by considering nodes in postorder.
 ensures a node is visited after all its children

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 $\sum_{v \in S} w_v$.

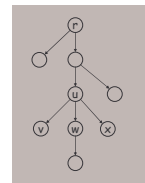
Observation. If (u, v) is an edge such that v is a leaf node, then either OPT includes u , or it includes all leaf nodes incident to u .

Dynamic programming solution. Mark some node r as the root.

- $OPT_{in}(u) = \max$ 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 .

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

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



Independent Set on Trees: Algorithm

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

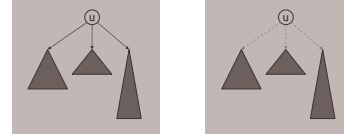
```
Weighted-Independent-Set-In-A-Tree-Value(T)
Root the tree at a node r
foreach (node u of T in postorder)
  if (u is a leaf) {
    then  $M_{in}[u] \leftarrow w_u$ 
        $M_{out}[u] \leftarrow 0$ 
    }
  else
     $M_{in}[u] \leftarrow \sum_{v \in \text{children}(u)} M_{out}[v] + w_u$ 
     $M_{out}[u]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

Exercise. Recover the independent set of maximum weight by tracing back.

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

Context

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.



see Chapter 10.4

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.