CS38
Introduction to Algorithms

Lecture 17

May 27, 2014

Outline

- coping with intractibility
  - NP-completeness
  - special cases
  - fixed parameter complexity
  - approximation algorithms

Hardness and completeness

- Reasonable that can efficiently transform one problem into another.
- Surprising:
  - can often find a special language L so that **every** language in a given complexity class reduces to L!
  - powerful tool

Hardness and completeness

- Recall:
  - a language L is a set of strings
  - a complexity class C is a set of languages

Definition: a language L is C-hard if for every language A $\in$ C, A poly-time reduces to L; i.e., $A \leq_P L$.

meaning: L is at least as "hard" as anything in C

Hardness and completeness

- Recall:
  - a language L is a set of strings
  - a complexity class C is a set of languages

Definition: a language L is C-complete if L is C-hard and L $\in$ C

meaning: L is a "hardest" problem in C

Lots of NP-complete problems

- logic problems
  - 3-SAT = {φ : φ is a satisfiable 3-CNF formula}
  - NAE3SAT, (3,3)-SAT
  - Max-2-SAT
- finding objects in graphs
  - independent set
  - vertex cover
  - clique
- sequencing
  - Hamilton Path
  - Hamilton Cycle and TSP
- problems on numbers
  - subset sum
  - knapsack
  - partition
- splitting things up
  - max cut
  - min/max bisection

Example: Integer programming

Definition: Integer Linear Program (ILP) = {LPs with integer variables that have a feasible solution}

Theorem: ILP is NP-complete.

- Proof:
  - Part 1: ILP $\in$ NP. Proof? (try just for 0/1)
  - Part 2: ILP is NP-hard.
    - reduce from?

Integer programming

- We are reducing from the language:

3-SAT = {φ : φ is a satisfiable 3-CNF formula}

to the language:

ILP = {LPs with integer variables that have a feasible solution}

Integer programming

$\varphi = (x \vee y \vee \neg z) \wedge (\neg x \vee w \vee z) \wedge \ldots \wedge (\ldots)$

- ILP variable x for each Boolean variable x
- $0 \leq x \leq 1$
- represent $\neg$ x by $(1 - x)$
- each clause has a natural linear expression:
  - e.g. $(x \vee y \vee \neg z) \rightarrow (x + y + (1 - z))$
- constrain each such expression to be ≥ 1

is this reduction polynomial time?

Integer programming

$\varphi = (x \vee y \vee \neg z) \wedge (\neg x \vee w \vee z) \wedge \ldots \wedge (\ldots)$

- ILP variable x for each Boolean variable x
- $0 \leq x \leq 1$
- represent $\neg$ x by $(1 - x)$
- each clause has a natural linear expression:
  - e.g. $(x \vee y \vee \neg z) \rightarrow (x + y + (1 - z))$
- constrain each such expression to be ≥ 1

YES maps to YES?

Integer programming

$\varphi = (x \vee y \vee \neg z) \wedge (\neg x \vee w \vee z) \wedge \ldots \wedge (\ldots)$

- ILP variable x for each Boolean variable x
- $0 \leq x \leq 1$
- represent $\neg$ x by $(1 - x)$
- each clause has a natural linear expression:
  - e.g. $(x \vee y \vee \neg z) \rightarrow (x + y + (1 - z))$
- constrain each such expression to be ≥ 1

NO maps to NO?

Coping with intractability

- NP-complete problem cannot have a polynomial-time algorithm, unless P = NP
  - considered unlikely

NP-complete problems are everywhere!

we need strategies to deal with them

Coping with intractability

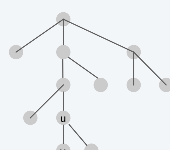
- Strategies for coping with intractability
  - consider special case or more restrictive version of the problem
  - parameterized complexity
    - problem size n, parameter k
    - find O(exp(k)· poly(n)) instead of O(n^k) algorithm
  - approximation algorithms: for optimization problems, find an approximate solution
  - heuristics…

Special case example

Independent set on trees

Independent set on trees. Given a tree, find a maximum cardinality subset of nodes such that no two are adjacent.

Fact. A tree has at least one node that is a leaf (degree = 1).



Key observation. If node v is a leaf, there exists a max cardinality independent set containing v.

Pf. [exchange argument]

- Consider a max cardinality independent set S.
- If $v \in S$, we're done.
- Let (u, v) be some edge.
  - 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 max cardinality independent set in forests (and hence trees).

Pf. Correctness follows from the previous key observation. ▪

INDEPENDENT-SET-IN-A-FOREST (F)

$S \leftarrow \emptyset$.
WHILE (F has at least 1 edge)
  $e \leftarrow (u, v)$ such that v is a leaf.
  $S \leftarrow S \cup \{ v \}$.
  $F \leftarrow F - \{ u, v \}$. ← delete u and v and all incident edges
RETURN S.

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

Weighted independent set on trees

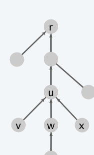
Weighted independent set on trees. Given a tree and node weights $w_v > 0$, find an independent set S that maximizes $\Sigma_{v \in S} w_v$.

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.
- $OPT = \max \{ OPT_{in}(r),\ OPT_{out}(r) \}$.

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

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



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

Weighted independent set on trees: dynamic programming algorithm

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

can also find independent set itself (not just value)

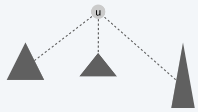
WEIGHTED-INDEPENDENT-SET-IN-A-TREE (T)

Root the tree T at a node r.
$S \leftarrow \emptyset$.
FOREACH (node u of T in postorder)
  IF (u is a leaf)
    $M_{in}[u] = w_u$.
    $M_{out}[u] = 0$.
  ELSE
    $M_{in}[u] = w_u + \Sigma_{v \in \text{children}(u)} M_{out}[v]$.
    $M_{out}[u] = \Sigma_{v \in \text{children}(u)} \max \{ M_{in}[v],\ M_{out}[v] \}$.
RETURN $\max \{ M_{in}[r],\ M_{out}[r] \}$.

ensures a node is visited after all its children

NP-hard problems on trees: context

Independent set on trees. Tractable because we can find a node that breaks the communication among the subproblems in different subtrees.

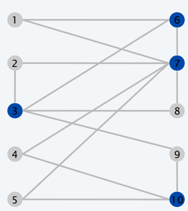


Linear-time on trees. VERTEX-COVER, DOMINATING-SET, GRAPH-ISOMORPHISM, ...

Parameterized complexity example

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. VERTEXCOVER is NP-complete. But 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 $O(k\,n)$ time to check whether a subset is a vertex cover.

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

Ex. $n = 1{,}000$, $k = 10$.
Brute. $k\,n^{k+1} = 10^{34} \Rightarrow$ infeasible.
Better. $2^k\,k\,n = 10^7 \Rightarrow$ 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 $\leq k - 1$.

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). 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. ▪

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 $O(2^k\,kn)$ time.

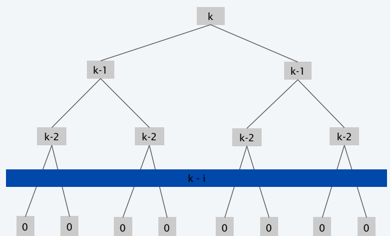
```
Vertex-Cover(G, k) {
   if (G contains no edges)    return true
   if (G contains ≥ kn edges) return false

   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

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



Approximation algorithms

Optimization Problems

- many hard problems (especially **NP**-hard) are optimization problems
  - e.g. find *shortest* TSP tour
  - e.g. find *smallest* vertex cover
  - e.g. find *largest* clique

  - may be minimization or maximization problem
  - "OPT" = value of optimal solution

Approximation Algorithms

- often happy with approximately optimal solution
  - warning: lots of heuristics
  - we want approximation algorithm with guaranteed approximation ratio of r
  - meaning: on every input x, output is guaranteed to have value

    at most r*opt for minimization

    at least opt/r for maximization

Approximation Algorithms

- Example approximation algorithm:

  Vertex Cover (VC): given a graph G, what is the *smallest* subset of vertices that touch every edge?

Theorem: decision version of VC is NP-complete

Proof: in NP (why?)

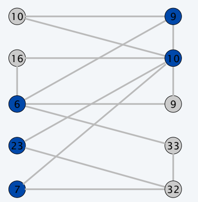
- reduce from?

Approximation Algorithms

- Approximation algorithm for VC:
  - pick an edge (x, y), add vertices x and y to VC
  - discard edges incident to x or y; repeat.
- Claim: approximation ratio is 2.
- Proof:
  - an optimal VC must include at least one endpoint of each edge considered
  - therefore 2·OPT≥ actual

Weighted vertex cover

Given a graph $G = (V, E)$ with vertex weights $w_i \geq 0$, find a min weight subset of vertices $S \subseteq V$ such that every edge is incident to at least one vertex in S.



total weight = 6 + 23 + 7 + 9 + 10 = 55

31

Weighted vertex cover: IP formulation

Given a graph $G = (V, E)$ with vertex weights $w_i \geq 0$, find a min weight subset of vertices $S \subseteq V$ such that every edge is incident to at least one vertex in S.

Integer programming formulation.

- Model inclusion of each vertex i using a 0/1 variable x_i.

$$x_i = \begin{cases} 0 & \text{if vertex } i \text{ is not in vertex cover} \\ 1 & \text{if vertex } i \text{ is in vertex cover} \end{cases}$$

  Vertex covers in 1-1 correspondence with 0/1 assignments: $S = \{ i \in V : x_i = 1\}$.

- Objective function: maximize $\Sigma_i w_i x_i$.

- Must take either vertex i or j (or both): $x_i + x_j \geq 1$.

32

Weighted vertex cover: IP formulation

Weighted vertex cover. Integer programming formulation.

$$\begin{array}{llcll} (ILP) \ \min & \sum_{i \in V} w_i x_i & & & \\ \text{s.t.} & x_i + x_j & \geq & 1 & (i,j) \in E \\ & x_i & \in & \{0,1\} & i \in V \end{array}$$

Observation. If x^* is optimal solution to (ILP), then $S = \{ i \in V : x_i^* = 1\}$ is a min weight vertex cover.

33

Integer programming

Given integers a_{ij}, b_i, and c_j, find integers x_j that satisfy:

$$\begin{array}{lllll} \max & c^t x & & \\ \text{s.t.} & Ax & \geq & b \\ & x & & \text{integral} \end{array} \qquad \begin{array}{rcll} \sum_{j=1}^{n} a_{ij} x_j & \geq & b_i & 1 \leq i \leq m \\ x_j & \geq & 0 & 1 \leq j \leq n \\ x_j & & \text{integral} & 1 \leq j \leq n \end{array}$$

34

Linear programming

Given integers a_{ij}, b_i, and c_j, find real numbers x_j that satisfy:

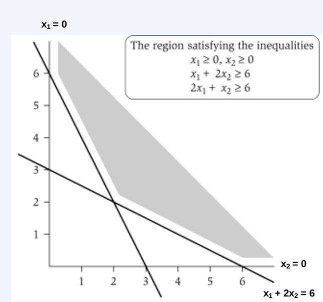
$$\begin{array}{llcl} (P) \ \max & c^t x & & \\ \text{s.t.} & Ax & \geq & b \\ & x & \geq & 0 \end{array} \qquad \begin{array}{llcll} (P) \ \max & \sum_{j=1}^{n} c_j x_j & & & \\ \text{s.t.} & \sum_{j=1}^{n} a_{ij} x_j & \geq & b_i & 1 \leq i \leq m \\ & x_j & \geq & 0 & 1 \leq j \leq n \end{array}$$

Simplex algorithm. [Dantzig 1947] Can solve LP in practice.
Ellipsoid algorithm. [Khachian 1979] Can solve LP in poly-time.

35

LP feasible region

LP geometry in 2D.



36

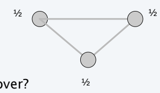
Weighted vertex cover: LP relaxation

Linear programming relaxation.

$$(LP)\ \min \sum_{i \in V} w_i x_i$$
$$\text{s.t.}\quad x_i + x_j \geq 1 \quad (i,j) \in E$$
$$x_i \geq 0 \quad i \in V$$

Observation. Optimal value of (LP) is δ optimal value of (ILP).
Pf. LP has fewer constraints.

Note. LP is not equivalent to vertex cover.



Q. How can solving LP help us find a small vertex cover?
A. Solve LP and round fractional values.

Weighted vertex cover: LP rounding algorithm

Lemma. If x^* is optimal solution to (LP), then $S = \{ i \in V : x_i^* \geq ½\}$ is a vertex cover whose weight is at most twice the min possible weight.

Pf. [S is a vertex cover]
- Consider an edge $(i, j) \in E$.
- Since $x_i^* + x_j^* \geq 1$, either $x_i^* \geq ½$ or $x_j^* \geq ½$ ⇒ (i, j) covered.

Pf. [S has desired cost]
- Let S^* be optimal vertex cover. Then

$$\sum_{i \in S^*} w_i \underset{\text{LP is a relaxation}}{\geq} \sum_{i \in S} w_i x_i^* \underset{x_i^* \geq ½}{\geq} \frac{1}{2} \sum_{i \in S} w_i$$

Theorem. The rounding algorithm is a 2-approximation algorithm.
Pf. Lemma + fact that LP can be solved in poly-time.

Approximation Algorithms

- diverse array of ratios achievable
- some examples:
  - (min) Vertex Cover: 2
  - MAX-3-SAT (satisfy max # clauses): 8/7
  - (min) Set Cover: ln n
  - (max) Clique: n/log^{2}n
  - (max) Knapsack: (1 + ε) for any ε > 0
- many known to be "correct" unless P=NP

Approximation Algorithms

(max) Knapsack: (1 + ε) for any ε > 0

- called Polynomial Time Approximation Scheme (PTAS)
  - algorithm runs in poly time for every fixed ε>0
  - poor dependence on ε allowed
- If all **NP** optimization problems had a PTAS, almost like **P** = **NP** (!)

Knapsack problem

Knapsack problem.
- Given n objects and a knapsack.
- Item i has value $v_i > 0$ and weighs $w_i > 0$. ← we assume $w_i \leq W$ for each i
- Knapsack has weight limit W.
- Goal: fill knapsack so as to maximize total value.

Ex: { 3, 4 } has value 40.

item	value	weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

original instance (W = 11)

Knapsack is NP-complete

KNAPSACK. Given a set X, weights $w_i \geq 0$, values $v_i \geq 0$, a weight limit W, and a target value V, is there a subset $S \subseteq X$ such that:

$$\sum_{i \in S} w_i \leq W$$
$$\sum_{i \in S} v_i \geq V$$

SUBSET-SUM. Given a set X, values $u_i \geq 0$, and an integer U, is there a subset $S \subseteq X$ whose elements sum to exactly U?

Theorem. SUBSET-SUM $\leq_P$ KNAPSACK.
Pf. Given instance $(u_1, \ldots, u_n, U)$ of SUBSET-SUM, create KNAPSACK instance:

$$v_i = w_i = u_i \qquad \sum_{i \in S} u_i \leq U$$
$$V = W = U \qquad \sum_{i \in S} u_i \geq U$$

Knapsack problem: dynamic programming I

Def. $OPT(i, w)$ = max value subset of items 1,..., i with weight limit w.

Case 1. OPT does not select item i.
- OPT selects best of 1, …, $i-1$ using up to weight limit w.

Case 2. OPT selects item i.
- New weight limit = $w - w_i$.
- OPT selects best of 1, …, $i-1$ using up to weight limit $w - w_i$.

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1, w) & \text{if } w_i > w \\ \max\{ OPT(i-1, w),\ \ v_i + OPT(i-1, w-w_i)\} & \text{otherwise} \end{cases}$$

Theorem. Computes the optimal value in $O(n\ W)$ time.
- Not polynomial in input size.
- Polynomial in input size if weights are small integers.

Knapsack problem: dynamic programming II

Def. $OPT(i, v)$ = min weight of a knapsack for which we can obtain a solution of value $\geq v$ using a subset of items 1,..., i.

Note. Optimal value is the largest value v such that $OPT(i, v) \leq W$.

Case 1. OPT does not select item i.
- OPT selects best of 1, …, $i-1$ that achieves value v.

Case 2. OPT selects item i.
- Consumes weight w_i, need to achieve value $v - v_i$.
- OPT selects best of 1, …, $i-1$ that achieves value $v - v_i$.

$$OPT(i,v) = \begin{cases} 0 & \text{if } v \leq 0 \\ \infty & \text{if } i = 0 \text{ and } v > 0 \\ \min\{OPT(i-1, v),\ w_i + OPT(i-1, v - v_i)\} & \text{otherwise} \end{cases}$$

Knapsack problem: dynamic programming II

Theorem. Dynamic programming algorithm II computes the optimal value in $O(n^2\ v_{max})$ time, where v_{max} is the maximum of any value.

Pf.
- The optimal value $V^* \leq n\ v_{max}$.
- There is one subproblem for each item and for each value $v \leq V^*$.
- It takes $O(1)$ time per subproblem. ▪

Remark 1. Not polynomial in input size!

Remark 2. Polynomial time if values are small integers.

Knapsack problem: polynomial-time approximation scheme

Intuition for approximation algorithm.
- Round all values up to lie in smaller range.
- Run dynamic programming algorithm II on rounded instance.
- Return optimal items in rounded instance.

item	value	weight
1	934221	1
2	5956342	2
3	17810013	5
4	21217800	6
5	27343199	7

original instance (W = 11)

item	value	weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

rounded instance (W = 11)

Knapsack problem: polynomial-time approximation scheme

Round up all values:
- v_{max} = largest value in original instance.
- ε = precision parameter.
- θ = scaling factor = $\varepsilon\ v_{max} / n$.

$$\bar{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil \theta, \quad \hat{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil$$

Observation. Optimal solutions to problem with $\bar{v}$ are equivalent to optimal solutions to problem with $\hat{v}$.

Intuition. $\bar{v}$ close to v so optimal solution using $\bar{v}$ is nearly optimal; $\hat{v}$ small and integral so dynamic programming algorithm II is fast.

Knapsack problem: polynomial-time approximation scheme

Round up all values: $\bar{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil \theta$

Theorem. If S is solution found by rounding algorithm and S^* is any other feasible solution, then $(1+\varepsilon) \sum_{i \in S} v_i \geq \sum_{i \in S^*} v_i$

Pf. Let S^* be any feasible solution satisfying weight constraint.

$$\begin{aligned} \sum_{i \in S^*} v_i &\leq \sum_{i \in S^*} \bar{v}_i && \text{always round up} \\ &\leq \sum_{i \in S} \bar{v}_i && \text{solve rounded instance optimally} \\ &\leq \sum_{i \in S} (v_i + \theta) && \text{never round up by more than } \theta \\ &\leq \sum_{i \in S} v_i + n\theta && |S| \leq n \\ &\leq (1+\varepsilon) \sum_{i \in S} v_i && n\theta = \varepsilon v_{max},\ \ \underbrace{v_{max}}_{\text{DP alg can take } v_{max}} \leq \Sigma_{i \in S} v_i \end{aligned}$$

Knapsack problem: polynomial-time approximation scheme

Theorem. For any $\varepsilon > 0$, the rounding algorithm computes a feasible solution whose value is within a $(1 + \varepsilon)$ factor of the optimum in $O(n^3 / \varepsilon)$ time.

Pf.
- We have already proved the accuracy bound.
- Dynamic program II running time is $O(n^2 \hat{v}_{\max})$, where

$$\hat{v}_{\max} = \left\lceil \frac{v_{\max}}{\theta} \right\rceil = \left\lceil \frac{n}{\varepsilon} \right\rceil$$

PTAS. $(1 + \varepsilon)$-approximation algorithm for any constant $\varepsilon > 0$.
- Produces arbitrarily high quality solution.
- Trades off accuracy for time.
- But such algorithms are unlikely to exist for certain problems...

49