ADVANCED ALGORITHMS · SESSION 2015–16

Lecture 9

Approximation Algorithms

Faculty: K.R. Chowdhary

Professor of CS

LP rounding and the primal-dual method for weighted vertex cover

Why Approximate?

For many important optimization problems, no polynomial-time algorithm is known that computes the exact optimum — many NP-complete problems are hard in the sense that solving any one in polynomial time would solve them all. Approximation algorithms circumvent this by relaxing the goal: compute a solution whose value is as close as possible to optimal, in polynomial time.

Even when exact algorithms exist…

An approximation algorithm can still be worthwhile for its faster running time, lower space requirements, or easier parallel/distributed implementation.

This matters even more for "big data" — where even a quadratic-time algorithm may be too slow for astronomically large inputs on present-day hardware.

α-Approximation Algorithms

Maximization problem

$$\mathbf{ALG(x) \le OPT(x) \le \alpha \cdot ALG(x)}$$

for every input instance x. (0 < α < 1 here)

Minimization problem

$$\mathbf{OPT(x) \le ALG(x) \le \alpha \cdot OPT(x)}$$

for every input instance x. (α > 1 here)

- OPT(x) is the optimal objective value; ALG(x) is the algorithm's output value — the definition only needs the value, not the solution itself (though usually both are produced).
- α is called the performance guarantee, approximation ratio, or approximation factor of the algorithm.
- Convention here: α > 1 for minimization, α < 1 for maximization — either way, α measures how close ALG gets to OPT.

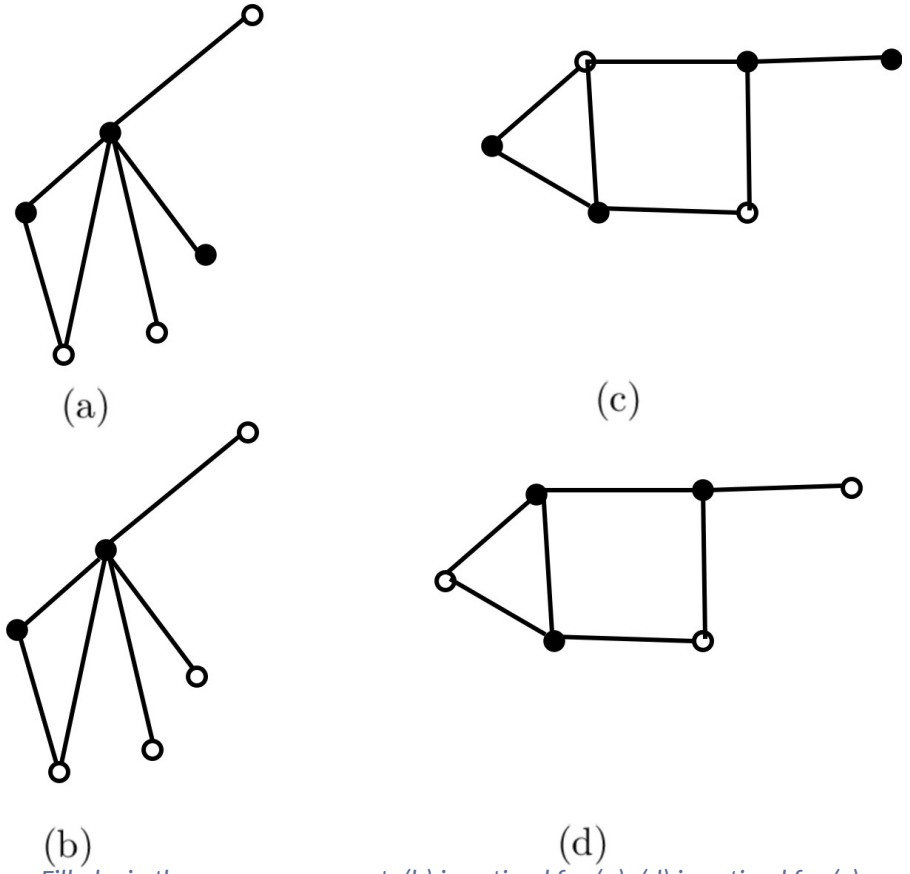
The Vertex Cover Problem

Definition

Given G = (V, E), a set S ⊆ V covers edge e if at least one endpoint of e is in S. S is a vertex cover if it covers every edge.

Two forms

- Optimization: find the smallest k such that G has a vertex cover of size k.
- Decision: given G and k, does G have a vertex cover of size ≤ k?
- Weighted version: each vertex v has weight $w_v \geq 0$; minimize total covering weight.



Filled = in the cover, open = not. (b) is optimal for (a); (d) is optimal for (c).

From Integer Program to Linear Program

INTEGER PROGRAM (EXACT)

$$\min \sum_{v \in V} w_v x_v$$

$$\text{s.t. } x_u + x_v \geq 1 \quad \forall\, e=(u,v) \in E$$

$$x_v \in \{0,1\} \quad \forall\, v \in V$$

$x_v = 1$ means v is chosen in the cover. Constraint forces every edge to have ≥ 1 endpoint chosen. NP-hard to solve exactly.

LP RELAXATION (POLY-TIME)

$$\min \sum_{v \in V} w_v x_v$$

$$\text{s.t. } x_u + x_v \geq 1 \quad \forall\, e=(u,v) \in E$$

$$x_v \geq 0 \quad \forall\, v \in V$$

Writing $x_v \geq 0$ rather than $x_v \in [0,1]$ loses nothing: an optimum never assigns $x_v > 1$, since reducing it to 1 keeps feasibility and only helps the objective.

Fractional Solutions Can Beat Integral Ones

Worked example: a 3-cycle

Let G be a triangle on vertices u, v, w, each with weight 1. The vector x = (½, ½, ½) satisfies every LP constraint (each edge sums to ½+½=1), and gives objective value ½+½+½ = 3/2.

But the minimum-weight integer vertex cover of a triangle has weight 2 — any 2 of the 3 vertices. The fractional LP optimum (3/2) is strictly smaller than the integer optimum (2).

THE CONSEQUENCE

We can solve the LP relaxation in polynomial time, but the solution may be fractional — so we need to post-process it into an actual (integral) vertex cover. This is where rounding comes in.

The Rounding Algorithm

THE ROUNDING RULE

Let x be an optimal LP solution. Define $\bar{x}_v = 1$ if $x_v \geq ½$, else $\bar{x}_v = 0$. Let $S = \{ v \mid \bar{x}_v = 1 \}$.

S is a valid vertex cover

For every edge e=(u,v), the LP constraint $x_u + x_v \geq 1$ forces at least one of x_u, x_v to be $\geq ½$ — so that endpoint is rounded to 1 and lands in S.

The key rounding inequality

$\bar{x}_v \leq 2x_v$ for every vertex v — since rounding at most doubles a value that was already $\geq ½$.

This one inequality is exactly what drives the approximation-ratio proof on the next slide.

Proving the Factor-2 Approximation

Let S be the vertex cover chosen by rounding, and OPT the optimum vertex cover.

$$\sum_{v \in S} w_v \;=\; \sum_{v \in V} w_v \bar{x}_v \;\le\; 2 \sum_{v \in V} w_v x_v \;\le\; 2 \sum_{v \in OPT} w_v$$

First inequality

Follows directly from the rounding rule: $\bar{x}_v \le 2x_v$ for every vertex.

Second inequality

The LP's feasible region is at least as large as the integer program's, so the fractional optimum can never exceed the integer optimum.

Approximation factor: 2

A Faster Route to the Same Guarantee

LP rounding needs to solve a linear program — polynomial time, but not the fastest option. The primal-dual method is guided by the LP and its dual without solving either to optimality.

PRIMAL (relaxed)

$$\min \ \sum_v w_v x_v$$

$$\text{s.t. } x_u + x_v \geq 1 \quad \forall e=(u,v) \in E$$

$$x_v \geq 0 \qquad \forall v \in V$$

DUAL

$$\max \ \sum_e y_e$$

$$\text{s.t. } \sum_{e \in \delta(v)} y_e \leq w_v \quad \forall v \in V$$

$$y_e \geq 0 \qquad \forall e \in E$$

$\delta(v)$ = edges incident to v. Read y^e as a price on edge e, and w_v as vertex v's wealth: the constraint says v has enough wealth to pay for its incident edges.

The Algorithm: Greedily Raising Edge Prices

Rather than exactly maximizing total edge price, use a greedy heuristic: go through edges in any order, raising each price as much as possible without breaking a dual constraint.

ALGORITHM — PRIMAL-DUAL VERTEX COVER

```
S = ∅ , y_e = 0 ∀e∈ E, s_v = 0 ∀v∈ V
foreach e=(u,v)∈ E:
  δ = min(w_u − s_u, w_v − s_v)
  y_e = y_e + δ
  s_u = s_u + δ ; s_v = s_v + δ
  if s_u = w_u: S = S ∪ {u}
  if s_v = w_v: S = S ∪ {v}
return S
```

Reading the Algorithm, and Its Invariants

s_v tracks $\Sigma_{e \in \delta(v)} y_e$ — the left side of vertex v's dual constraint — as it grows. A vertex joins S exactly when its constraint goes tight ($s_v = w_v$), following the complementary-slackness principle: primal choices should track which dual constraints are tight.

LOOP INVARIANTS (hold at the start/end of every iteration)

1. y is a feasible vector for the dual LP.
2. $s_v = \Sigma_{e \in \delta(v)} y_e$ for every vertex v.
3. $S = \{ v \mid s_v = w_v \}$.
4. $\Sigma_{v \in V} s_v = 2 \Sigma_{e \in E} y_e$ (each edge contributes to exactly two vertices' sums).

Correctness and the Factor-2 Guarantee

S is a valid cover

Each loop iteration runs in constant time — the algorithm is linear-time. After processing edge (u,v), at least one endpoint must have joined S (its slack must be exhausted), so S covers every edge.

Using weak duality ($\Sigma_{e \in E} y_e \le \Sigma_{v \in OPT} w_v$) and the invariants:

$$\Sigma_{v \in S} w_v = \Sigma_{v \in S} s_v \le \Sigma_{v \in V} s_v = 2\Sigma_{e \in E} y_e \le 2\Sigma_{v \in OPT} w_v$$

Same 2-approximation guarantee — without ever solving an LP.

References

[1] **Thomas H. Cormen, et al.,**
"Introduction to Algorithms, 3rd ed." — PHI, 2009

[2] **David P. Williamson & David B. Shmoys,**
"The Design of Approximation Algorithms" — e-book (downloadable)

[3] **Mikhail J. Atallah & Marina Blanton,**
"Algorithms and Theory of Computation Handbook, Special Topics and Techniques, 2nd ed." — CRC Press, 2010

[4] **Allen B. Tucker, Jr.,**
"The Computer Science and Engineering Handbook" — CRC Press, 1997

Disclaimer: these notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the faculty.