COMPSCI 632: Approximation Algorithms 8/28/2024

Lecture 2: Threshold Rounding and Introduction to Primal-Dual

Lecturer: Debmalya Panigrahi *Scribe: Qilin Ye*

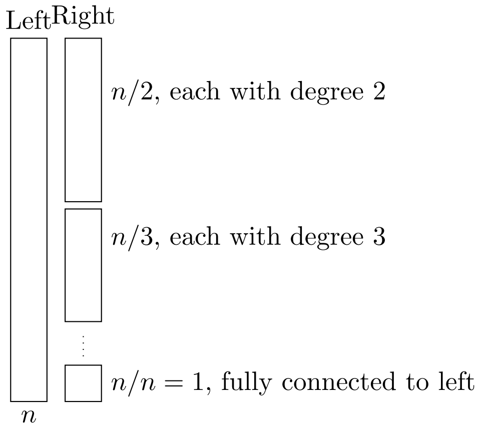
1 Greedy Approaches (continuing from last lecture)

The most intuitive approach would be to greedily select the vertex with highest degree, and iteratively repeat this process — after all, we want to more edges with fewer vertices, so high-degree vertices seem to be a natural choice. Unfortunately, curated examples show that this algorithm is of approximation factor $\Omega(\log n)$ (in fact it is $\Theta(\log n)$).

A "bad" graph for the naïve greedy algorithm.

Consider a bipartite graph. The left component consists of n nodes/vertices. The right component consists of $n-1$ blocks, each with n/j nodes, $j \in [n]$[1]. This way, the right component has a total of approximately $n \log n$ nodes.

For each right side block of size n/j, we assume that each node has degree j, and that they all are connected to different nodes on the left side. (For example, the first node in the $n/2$ component can connect to the first two in the left component, and the second node in the $n/2$ component can connect to the 3rd and 4th on the left side, and so on.)


Left
Right
$n/2$, each with degree 2
$n/3$, each with degree 3
$n/n = 1$, fully connected to left
n

Now, what will our naïve greedy algorithm do? First, it will pick the one single node from the bottom right component since it has degree n and no other node does (the ones on left side each are connected to $n-1$ nodes, one from each block on the right). After picking this node and removing all the connected edges, in the remaining graph, the nodes in the $n/(n-1)$ block have degrees $n-1$, whereas the left nodes now have degrees $n-2$. So we pick the nodes in the $n/(n-1)$ block. Iteratively, we see that we never get to pick the nodes from the left, and in the end we must have picked every single node on the right. That is $\mathcal{O}(n \log n)$ nodes, whereas we clearly see that an more optimal solution is to just pick every node on the left, a total of $\mathcal{O}(n)$ nodes.

From this instance alone we see that our algorithm must have an approximation factor $\geqslant \log n$. (And we'll see later that this algorithm is isomorphic to a $\log n$-approximation algorithm of Set Cover, so here the optimality factor $\Omega(\log n)$ becomes $\Theta(\log n)$.)

The fix, though seemingly less reasonable than the naïve greedy algorithm, is to add both vertices of the same edge in each iteration, and discard all edges that are not completely disjoint from our set:

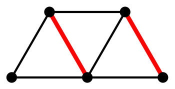
[1]$i \in [n]$ is an abbreviation of $1 \leqslant i \leqslant n$.

Algorithm 1: Improved Greedy Algorithm

```
Inputs: graph $G = (V, E)$

start with $S = \emptyset$
while $S$ is not a vertex cover do
    choose any edge $e = (u, v)$ with $|\{u, v\} \cap S| = 0$
    $S \leftarrow S \cup \{u, v\}$
end
return $S$
```

Indeed, even for simple cases, we see that sometimes this algorithm fails to choose optimally. Consider the following graph: it is clear that there exists a set of 3 vertices that covers the graph, but our algorithm terminates after 2 iterations, resulting in a total of 4 vertices being picked. Nevertheless, the following proof shows that this algorithm is a 2-approximation.



Proof. Another way to view this algorithm is by paying attention to the edges it has selected among the iterations (i.e., the e instead of the $\{u, v\}$). The key observation of this proof lies in the following fact: for the edges chosen by this algorithm (call it ALG), they all are incident on two distinct sets of vertices. In other words, these chosen edges cannot share *any* common endpoint. A matching!

Suppose our graph G has a matching of size k. Consider any vertex cover and any edge $e = (u, v)$. It follows that either u and/or v is in the cover. That we have a matching of size k means we have k mutually disjoint edges, so the vertex cover must contain one endpoint from each edge, resulting in a cardinality of $\geqslant k$. In particular, if G admits a matching of size n, then the *optimal* vertex cover must also have at least k vertices.

Assuming the previous notations, our outputted vertex cover by ALG contains both endpoints of each edge of the matching, so its cardinality is $2k$. The previous paragraph showed that OPT has at least k vertices. Hence ALG achieves an approximation factor of 2, completing the proof. $\square$

Remark 1. *We will later show that optimal solution exist on bipartite graphs.*

Remark 2. *When the optimal solution isn't accessible, a common trick is that we compare it against a surrogate that serves as a stand-in for the optimal value, providing a lower bound (e.g., in the previous problem, the optimal vertex cover has at least size k). This approach allows us to establish a worst-case approximation factor.*

2 Threshold Rounding

Another way to construct a 2-approximation algorithm for Vertex Cover is by relaxing the problem and drawing some connection to **linear programming** (LP). The main idea is simple: we want to assign binary values x_v to each vertex v while preserving the algebraic structures of the original problem:

$$\min_{v\in V} x_v \qquad \text{subject to} \qquad \begin{cases} x_u + x_v \geqslant 1 \text{ for each edge } (u,v) \in E \\ x_v \in \{0,1\} \text{ for each } v \in V. \end{cases}$$

However, a problem immediately arises: the feasible set is not convex! Consider the simplest graph of one edge and two vertices u, v. We can set either one to be 1 and the other to be 0, but taking the average, $x_u = x_v = 0.5$, no longer a feasible solution. The fix? We deviate from the original problem and allow *fractional* values for x_v:

$$\min_{v\in V} x_v \qquad \text{subject to} \qquad \begin{cases} x_u + x_v \geqslant 1 \text{ for each edge } (u,v) \in E \\ x_v \in [0,1] \text{ for each } v \in V. \end{cases} \tag{1}$$

Notice that we can still assume $x_v \in [0,1]$ since if $x_v > 1$, we can simply set x_v to be 1 to further decrease the objective function. We will assume the fact that there are polynomial-time algorithms that solve these type of LP.

A few problems nevertheless persist. *Firstly, how do we define an approximation factor?* Now we have three quantities: the output of our algorithm, the output of the optimal solution to Vertex Cover, and the output of the optimal solution to LP. Let us call these ALG, OPT(VC), and OPT(LP), respectively. Immediately we have the following inequality:

$$\text{ALG} \geqslant \text{OPT(VC)} \geqslant \text{OPT(LP)} \tag{2}$$

where the first $\geqslant$ follows from the optimality of OPT(VC) on Vertex Cover, and the second $\geqslant$ follows from the fact that the LP has less constraints than Vertex Cover. Therefore we can consider the ratio between ALG and OPT(LP) and define an approximation factor out of it.

Second problem: how do we recover an ALG *from* OPT(LP) *that makes sense?* The answer here is via **rounding**: given our optimal fractional solution, we want to recover an integer solution from which we can extract a valid vertex cover. And most naturally we round our fractional values to 0 and 1 with a threshold of 0.5: if $x_v \geqslant 0.5$ we round it to 1, and 0 otherwise.

***Proof that rounding is a* 2*-approximation*.** We first observe that rounding yields a valid vertex cover: for any edge $e = (u,v)$, if $x_u + x_v = 0$ post-rounding, then it means x_u, x_v are both < 0.5 pre-rounding, contradicting the constraints in (1). Therefore each edge has at least one value of 1 after rounding, i.e., this is a valid vertex cover.

To see this algorithm has an approximation factor of 2, we note that the values assigned to each edge is at most doubled (0.5 to 1). Therefore the objective is at most doubled. □

Third problem: OPT(*LP*) *is just as elusive as* OPT(VC)*, not to mention we obtained it using a LP-solving black box. What is going on here?* To answer this question, we will introduce a common technique in LP:

3 The Primal-Dual Method

The **primal-dual method** is used in LP as a tool to provide an alternative perspective that sometimes reveals additional relationships between constraints and objective function values. With the help of *strong* or *weak duality*, this method helps derive bounds or even optimal values for our objective functions, as we will see below.

Overall, the transformation of primal LP into dual LP assumes the following form:

$$\text{Primal} : \min\{c^T x \mid Ax \geqslant b, x \geqslant 0\} \iff \text{Dual} : \max\{b^T y \mid A^T y \leqslant c, y \geqslant 0\} \tag{3}$$

In slightly more details:

- Dual variables correspond to primal constraints: for each constraint in the primal, define a dual variable y. *In the* Vertex Cover *example, each constraint is of form $x_u + x_v \geqslant 1$, one for each edge (u, v), so a dual variable would be defined w.r.t. edges: $y_e, e \in E$.*
- Dual constraints correspond to primal variables: for each variable in the primal, there will be a constraint in the dual. *The primal coefficient for each vertex is* 1*, so the dual constraint requires*

$$\sum_{\text{edge } e \text{ incident to } v} y_e \leqslant 1 \qquad \text{for all } v \in V.$$

- If the primal is a minimization problem, the dual will be a maximization. Vice versa. *In our example, the dual would be a maximization.*
- The signs of dual variables also correspond to the primal constraints. If the primal constraint is $\leqslant$, the corresponding dual variable is non-negative. *This means $y_e \geqslant 0$.*

Therefore, we have successfully identified the dual LP of (1):

$$\underbrace{\min \sum_{v \in V} x_v \text{ subject to } \begin{cases} x_u + x_v \geqslant 1, (u, v) \in E \\ x_v \in [0, 1] \end{cases}}_{\text{Primal LP}} \implies \underbrace{\max \sum_{e \in E} y_e \text{ subject to } \begin{cases} \sum_{e \in S_v} y_e \leqslant 1 \text{ for all } v \in V \\ y_e \geqslant 0 \end{cases}}_{\text{Dual LP}}$$

where S_v denotes the "star graph" at v, or the collection of edges incident on v. Combining (2) and duality, we see that

$$\text{ALG} \geqslant \text{OPT(VC)} \geqslant \text{OPT(LP)} \qquad \text{and} \qquad \text{OPT(dual)} \geqslant \text{SOL(dual)}$$

since the dual LP is a maximization problem and OPT(dual) is assumed to attain the maximum value, and it must be no less than any valid dual solution, SOL(dual).

To relate the primal constraints to dual, we note that for each edge $e = (u, v)$,

$$y_e(x_u + x_v) = y_{(u,v)}(x_u + x_v) \geqslant y_{u,v}$$

since $x_u + x_v \geqslant 1$. Therefore,

$$\begin{aligned}
\sum_{e\in E} y_e \leqslant \sum_{(u,v)} y_{(u,v)}(x_u + x_v) &= \sum_{(u,v)} y_{(u,v)} x_u + \sum_{(u,v)} y_{(u,v)} x_v \\
&= \sum_v \left(\sum_{(u,v)\in S_v} y_{(u,v)} x_v \right) && \text{(each } y_e \text{ is multiplied by both of its endpoint values)} \\
&\leqslant \sum_v x_v && \text{(dual constraint } \sum_{(u,v)\in S_v} y_{(u,v)} \leqslant 1\text{)}
\end{aligned}$$

which shows that the objective value of the dual is always less than the objective value of the matching primal. By weak duality we therefore have

$$\text{ALG} \geqslant \text{OPT(VC)} \geqslant \text{OPT(LP)} \geqslant \text{OPT(dual)} \geqslant \text{SOL(dual)} \tag{4}$$

and so anything valid for the dual LP is a lower bound for OPT(VC). This is more useful because we have *complete* control over what SOL(dual) looks like.

Remark 3. *The algorithm was designed without any explicit mention of LP, but as we have seen, the analysis heavily involved it. This technique is called* ***dual fitting****.*

Designing & Analyzing the Primal-Dual Algorithm

The high level idea is that we want to build up a primal solution, build up a dual solution simultaneously, and draw a connection between the primal and the dual via e.g. weak duality.

More generally, below is the general roadmap:

(1) Start with P = D = 0.

(2) (The following applies to Vertex Cover; adjustment may be needed for general problems.) Recall that the primal is a minimization problem and dual maximization.

(3) Eventual goals:

- (Primal feasibility) P is feasible for primal LP.
- (Dual feasibility) D is feasible for dual LP.

(4) Observation:

- D is already feasible (set each $y_e = 0$). Therefore, *we want to keep dual feasibility invariant* under each iteration. Consequently,
- *The algorithm terminates when primal feasibility is satisfied.*

(5) To maintain an approximation factor, always ensure $\Delta\text{P} \leqslant \alpha \cdot \Delta\text{D}$ so that the primal will remain within a factor of α of D, resulting in an α-approximation. (In this case, $\alpha = 2$.)

Let us now apply the primal-dual algorithm to Vertex Cover.

Algorithm 2: Dual LP for Vertex Cover

```
Inputs: graph $G = (V, E)$, primal $x : V \to \mathbb{R}$, dual $y : E \to \mathbb{R}$
Initialization: $x(\cdot) = 0$ and $y(\cdot) = 0$
while Primal LP is not satisfied do
    find a dual variable $y_e$ that can be increased without violating dual
      feasibility (guaranteed to exist when primal not satisfied)
    increase $y_e$ until some dual constraint is tight
    for each primal variable $v$ corresponding to the tight dual constraint
      do
        set $x_v \leftarrow 1$
    end
end
return $\{v \in V : x_v = 1\}$
```

The correctness proof is omitted, but to see that this provides a 2-approximation algorithm, simply notice that ΔD comes from line 4 and ΔP comes from line 7. Each time a dual constraint becomes tight, its increment is precisely 1 (y_e must be going directly from 0 to 1, no in-betweens, since the y_e's we perturb are always disjoint), and line 7 repeats at most 2 times, one on each endpoint of y_e. Therefore, in each iteration, $\Delta P \leqslant 2\Delta D$, proving an approximation factor of 2.

Remark 4. *The primal-dual method is a very roundabout way of giving the exact result we pursued earlier. Why bother? The answer is because this alternative perspective is important: once we start look at even slightly more complicated problems (e.g. weighted vertices instead of uniform weight), our naïve approach quickly fails, but the primal dual algorithm, providing a more rigorous and systematic approach, continues to shine.*