COMPSCI 632: Approximation Algorithms 09/04/2024

Lecture 3: Weighted Vertex Cover and Set Cover

Lecturer: Debmalya Panigrahi *Scribe: Qilin Ye*

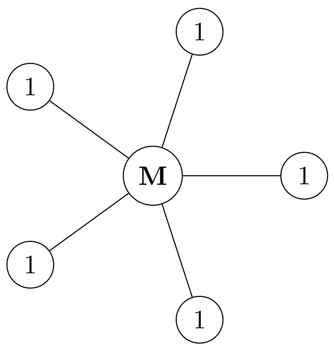
1 Approximations of Vertex Cover, Continued

1.1 Weighted Vertex Cover

The weighted Vertex Cover problem is identical to the standard Vertex Cover, except now each vertex $v \in V$ has a weight $w_v \geqslant 0$, and we want to find a cover with the least weight. Put in fancy notations,

Given $G = (V, E)$, where each vertex $v \in V$ is assigned a weight $w_v \geqslant 0$, find a $S \subset V$ with the smallest $\sum_{v \in S} w_v$ subject to the constraint that, for all $e = (u, v) \in E$, $|\{u, v\} \cap S| \geqslant 1$.

As promised in the previous remark (cf. previous lecture), we will first show how even our improved greedy algorithm fails miserably. The simplest example to consider here is a graph $G = (\{u, v\}, (u, v))$, with $w_u = 1$ and $w_v = M$ for some large M. The optimal solution yields $\{u\}$ with total weight u, but the greedy algorithm would pick both endpoints, giving us a total weight of $(M + 1)$, which makes the approximation factor arbitrarily large!



Perhaps a less trivial example (which shares the same idea) is the star graph shown here, where each of the n leaf nodes has weight 1 and the center node has weight M. An optimal solution, assuming $M \gg n$ is large, would pick all leaf nodes, resulting in a total weight of n, whereas our greedy algorithm would have chosen the center node plus any leaf node, resulting in a vertex cover of weight $M + 1$. Again, this resulting approximation factor can be arbitrarily bad depending on how large M is, compared to n.

The fix for this problem directly is not to complicated: our intuition suggests that we should somehow prefer nodes with less weight when adding to our vertex cover. It's not hard to rigorously formulate this algorithm directly, but what's charming is that in fact, *the dual algorithm works with almost no modification.* The only change lies in the dual constraint: now the primal objective is $\min \sum_v w_v x_v$, so each dual constraint is bonded by w_v, namely, $\sum_{e \text{ incident to v}} y_e \leqslant w_v$ for each v:

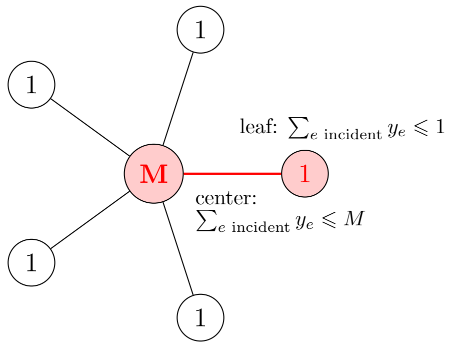
$$\min \sum_{v \in V} w_v x_v \text{ subject to } \begin{cases} x_u + x_v \geqslant 1, (u, v) \in E \\ x_v \geqslant 0 \end{cases} \implies \max \sum_{e \in E} y_e \text{ subject to } \begin{cases} \sum_{e \in S_v} y_e \leqslant w_v \text{ for all } v \in V \\ y_e \geqslant 0. \end{cases} \tag{1}$$

The algorithm itself works verbatim.

Algorithm 1: Dual LP for Weighted Vertex Cover

```
Inputs: graph $G = (V, E)$ with weights $w : V \to \mathbb{R}^+$, primal $x$, dual $y$
Initialization: $x(\cdot) = 0$ and $y(\cdot) = 0$
while Primal LP is not satisfied do
    find $y_e$ that can be increased without violating dual feasibility
    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 $\{v \in V : x_v = 1\}$
```

Before analyzing it, let's first see the algorithm in action. Same star graph example as above. Suppose our algorithm chose the red edge (call it e) initially. Clearly, y_e can be increased because none of the dual constraint related to y_e is currently tight. We note that in this graph, the dual constraint associated with the center is $\sum_{e \text{ incident}} y_e \leqslant M$, whereas for each leaf it is $\sum_{e \text{ incident}} y_e \leqslant 1$. Therefore we are free to increase y_e by 1, after which the constraint corresponding to the red leaf grows tight. Therefore we select the red leaf node but not the center node. If M is sufficiently large such that it always carries more weight in primal (and leaves more room in dual), then the center node will never be chosen. This solves our problem encountered before.


1
1
leaf: $\sum_{e \text{ incident}} y_e \leqslant 1$
M
1
center:
$\sum_{e \text{ incident}} y_e \leqslant M$
1
1

Of course, whenever a dual constraint becomes tight, nothing will make it loose again, so the algorithm is guaranteed to terminate in polynomial runtime. Let us now prove its correctness, and that it is 2-approximate like before.

Proof that the dual weighted* Vertex Cover *algorithm is* 2*-approximate.

- Claim 1. The dual is feasible. This follows since we explicitly demanded so when tweaking y_e.
- Claim 2. The primal is feasible when the algorithm finishes. Suppose not, that two adjacent vertices $(u, v) = e$ are both unselected. This means $y_e = 0$ for any e incident on u and v. Hence

$$\sum_{e \text{ incident on } u} y_e = \sum_{e \text{ incident on } v} y_e = 0.$$

  But then we can increase $y_{(u,v)}$ by 1, contradiction.
- Claim 3. The algorithm is 2-approximate. Different from before, instead of showing $\Delta P \leqslant 2\Delta D$, here we directly relate the final output P to D. We recall out objectives:

$$P = \sum_{v \in V} w_v \mathbf{1}[x_v = 1]^1 \qquad \text{and} \qquad D = \sum_{e \in E} y_e.$$

The rest is just pushing notations. When the algorithm terminates, the nodes labeled $x_v = 1$ are precisely those whose dual constraints are saturated. Therefore, for each such v, $\sum_{e \text{ incident on } v} y_e = w_v$. Hence

$$
\begin{aligned}
P = \sum_{v \in V} w_v \mathbf{1}[x_v = 1] &= \sum_{x_v = 1} \left[\sum_{e \text{ incident on } v} y_e \right] \\
&= \sum_{(u,v) \in E} y_e \cdot |\{u, v\} \cap \{v : x_v = 1\}| \quad \text{(count of } y_e = \text{num. of tight endpoints)} \\
&\leqslant \sum_{(u,v) \in E} y_e \cdot 2 = 2D. \qquad \square
\end{aligned}
$$

Remark 1. *Previously we showed a 2-approximate algorithm for* VERTEX COVER *based on rounding. The very same algorithm extends trivially to the weighted case, giving us another 2-approximation (round x_v to 1 if the fractional optimal solution $x_v^* \geqslant 1/2$ and 0 otherwise.)*

1.2 A Taste of Randomized Algorithms

Once again, we look back at "bad" examples we derived earlier, this time the carefully constructed graph (cf. last lecture) which yielded an approximation factor of $\log n$ for our original greedy algorithm. Recall that our algorithm failed miserably, because we were forced to always choose nodes from the right side. A natural attempt is to allow some randomness, so that we may find ourselves an opportunity to break the curse of only choosing nodes from the right side:

Algorithm 2: Randomized VERTEX COVER 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 $|\{u, v\} \cap S| = 0$ (i.e. uncovered)
    pick one of $\{u, v\}$ u.a.r. (uniformly at random) and add to $S$
end
return $S$
```

Not surprisingly, this gives us yet another 2-approximation algorithm for VERTEX COVER. The proof is quite different from the ones we've seen in primal-dual. Here, we will exploit the known / provable properties of the optimal solution, OPT, despite not having the slightest clue what it may look like. We will directly compare OPT with our randomized algorithm, ALG and attain useful bounds.

Proof. Let V_o be the set of vertices chosen by (an) OPT. For each $v \in V_o$, let S_v be the star graph with respect to the original $G = (V, E)$, i.e., $S_v = \{(v, u) \in E\}$, the collection of all edges incident on v. Since OPT is feasible, every edge is covered and in particular belongs to one of the star graphs, i.e., $\bigcup_{v \in V_o} S_v = E$. Therefore, our algorithm RLG is essentially *choosing a bunch of edges (or equivalently a bunch of vertices)*

[1] I use $\mathbf{1}[S]$ to denote the indicator function of S, i.e., $\mathbf{1}[S](\omega) = 1$ if $\omega \in S$ and 0 otherwise.

from a bunch of star graphs. (This is criminally oversimplifying things, of course, but the italicized claim is true nonetheless.)

So a natural question to ask is, how many vertices (on expectation, since our algorithm is probabilistic) will we choose from a *specific* graph? Let's just consider a certain star graph S_u with center u, and edges e_i leading to leaf nodes v_i. Let the random variable X_u denote the number of vertices chosen from S_u.

- The first time ALG considers an edge $e_i = (u, v_i)$ in S_u, there is a probability of 1/2 such that it chooses u. If this is the case, ALG will never consider *any* edge in S_u again because u has been chosen. So $\mathbb{P}(X_u = 1) = 1/2$.

- The event $X_u = 2$ means (i) when ALG consider an edge from S_u for the first time, it chose the leaf node over u, but (ii) the second time ALG considers an edge in S_u, it did choose u. So $\mathbb{P}(X_u = 2) = 1/2 \cdot 1/2 = 1/4$.

- We see that X_u is essentially a (truncated, since S_u is finite) geometric random variable with parameter 1/2, i.e., $\mathbb{P}(X_u = k) = 2^{-k}$. Therefore $\mathbb{E}X_u <= 2$.

Since on expectation ALG chooses $\leqslant 2$ vertices from each star graph, and there are $|V_o|$ star graphs in total, we see that the expected number of vertices chosen by ALG is $\leqslant 2|V_o|$, i.e., it is 2-approximate. $\square$

Now what? Update adding this randomized algorithm to approximate weighted Vertex Cover, of course. Revisiting the simplest weighted example of $G = (\{u, v\}, \{(u, v)\})$, if the vertex weights are 1 and M respectively, then the most natural way to balance our randomized algorithm is to choose the vertex with small weight (1) with high probability, and choose the big weight (M) with low probability. We make the proportion linear, and prove that this indeed gives a 2-approximation.

```
Algorithm 3: Randomized Weighted Vertex Cover Algorithm
  Inputs: graph $G = (V, E)$ with weights $w : V \to \mathbb{R}^+$
  start with $S = \emptyset$
  while $S$ is not a vertex cover do
      choose any edge $e = (u, v)$ with $|\{u, v\} \cap S| = 0$
      $S \leftarrow \begin{cases} S \cup \{u\} & \text{with probability } w_v/(w_u + w_v) \\ S \cup \{v\} & \text{with probability } w_u/(w_u + w_v) \end{cases}$
  end
  return $S$
```

Proof that randomized weighted* Vertex Cover *is a 2-approximation. The idea is similar to that of the unweighted case. If V_o is the set of vertices chosen by OPT, then the total weight is $\sum_{v \in V_o} w_v$. Therefore, similar to the unweighted case, our goal is to show that for each star graph S_v, the expected total weight of the vertices chosen by our algorithm ALG is bounded by $2w_v$, so that linearity of expectation says

$$\mathbb{E}[\text{ALG}] = \mathbb{E}\left[\sum_{v \in \text{ALG}} w_v\right] = \mathbb{E}\left[\sum_{v \in V_o} \text{weights on } S_v\right] \leqslant \mathbb{E}\left[\sum_{v \in V_o} 2w_v\right] = 2\mathbb{E}\left[\sum_{v \in V_o} w_v\right] = 2[\text{OPT}].$$

So let us consider a star graph S. Let the center node be v with weight w, and let the leaves v_i each have weights w_i, $i \in [k]$ (so a total of k leaves). To compute the expected total weight, for each node, we can multiply the weight by the probability that this vertex is chosen, then sum over all vertices in S. We WLOG[2]assume the edges are considered in the order of $(v, v_1), (v, v_2), \ldots$. [We'll later show this is valid.] Like the unweighted counterpart, we observe that as soon as w is chosen, ALG will never consider any remaining edge from S again. Therefore,

$$\mathbb{P}(v_i \text{ is chosen}) = \mathbb{P}(v_i \text{ is chosen in } (v, v_i)) \cdot \mathbb{P}(v_j \text{ is chosen in } (v, v_j) \text{ for all } j < i). \tag{*}$$

And now, time for algebra.

$$\begin{aligned}
\mathbb{E}[\text{total weight}] &= w \cdot \mathbb{P}(v \text{ is chosen}) + \sum_{i=1}^{k} w_i \cdot \mathbb{P}(v_i \text{ is chosen}) \\
&\leqslant w \cdot 1 + \sum_{i=1}^{k} w_i \cdot \mathbb{P}(v_i \text{ is chosen in } (v, v_i)) \cdot \prod_{j<i} (v_j \text{ is chosen in } (v, v_j)) \\
&= w + \sum_{i=1}^{k} w_i \cdot \frac{w}{w + w_i} \cdot \prod_{j<i} \frac{w}{w + w_j} = w + w \cdot \sum_{i=1}^{k} \frac{w_i}{w + w_i} \cdot \prod_{j<i} \frac{w}{w + w_j} \\
&= w + w \cdot \sum_{i=1}^{k} \frac{w_i}{w + w_i} \cdot \prod_{j<i} \left(1 - \frac{w_j}{w + w_j}\right) \\
&= w + w \cdot \sum_{i=1}^{k} p_i \cdot \prod_{j<i} (1 - p_j) \qquad \text{(define } p_i := w_i/(w + w_i)\text{)}.
\end{aligned}$$

An easy probabilistic interpretation of the highlighted term is as follows. Suppose we have n coins, where the j^{th} coin has a probability p_i of landing on heads. Then $p_i \prod_{j<i}(1 - p_i)$ is the probability that the first $i - 1$ coins land on tails but the i^{th} coin lands on heads. Summing over i we see that the highlighted term is the probability of the union of some disjoint events which is obviously bounded by 1. So $\mathbb{E}[\text{total weight}] \leqslant 2w$. Now apply (*) and the proof is complete! $\square$

Enough Vertex Cover. Let's now look at something more general: the Set Cover problem.

2 Set Cover

> Given a universal set U of n elements and m subsets $S_1, \ldots, S_m \subset U$ with $\bigcup_i S_i = U$, find a minimum number of subsets that cover U.

As usual, we approach this problem using a naïve greedy algorithm: always pick the set that covers as many remaining elements as possible. Note that an instance of Vertex Cover can be transformed into that of Set Cover via the following:

[2]Without loss of generality.

- The universal set $U = E$, the set of all edges in the bipartite graph $G = (V, E)$.
- For each vertex v, the star graph $S_v = \{(v, u) \in E\}$ can be viewed as a collection of edges, hence a subset of U. Notice that $\deg(v) = |S_v|$, so naturally the notion of choosing the highest degree vertex in the naïve greedy Vertex Cover algorithm corresponds to that of picking the set that covers most elements in Set Cover.

Since the naïve greedy Vertex Cover problem has an approximation factor of (at least) $\log n$ we know our greedy Set Cover algorithm can do no better. Fortunately, the following proof shows it is doing no worse either.

***Greedy* Set Cover *algorithm is* $\log n$*-approximate*.** Let $S_1, S_2, \ldots$ be the sequence of sets picked by our greedy algorithm, ALG. Say S_1 covers x_1 elements, and for each $i > 1$, S_i covers i *additional* (uncovered) elements. Consider the optimal solution OPT which consists of, with the abuse of notations, OPT subsets of U. The key observation is that:

- $x_1 \geqslant n/\text{OPT}$. To see this, note that $|U| = n$, so by pigeonhole the largest set has $\geqslant n/\text{OPT}$ elements.
- By the same token, $x_2 \geqslant (n - x_1)/\text{OPT}$ since all sets in OPT must still collectively cover the remaining $n - x_1$ elements. Thus, the best remaining set performs at least as good as the average.
- Iteratively, $x_i \geqslant (n - \sum_{j<i} x_j)/\text{OPT}$.

Rearranging the last inequality, we obtain (abusing the notation that ALG also means the cardinality of the outputted set)

$$1 \leqslant \text{OPT} \cdot \frac{x_i}{\sum_{j \geqslant i} x_j} \qquad \text{for each } i \in [\text{ALG}]$$

Summing over i we see

$$\begin{aligned}
\text{ALG} &\leqslant \text{OPT} \cdot \sum_{i=1}^{\text{ALG}} \frac{x_i}{\sum_{j \geqslant i} x_j} \\
&= \text{OPT} \cdot \left[\frac{x_1}{n} + \frac{x_2}{n - x_1} + \frac{x_3}{n - (x_1 + x_2)} + \ldots \right] \\
&\leqslant \text{OPT} \cdot \left[\underbrace{\frac{1}{n} + \frac{1}{n-1} + \ldots + \frac{1}{n - x_1 + 1}}_{n \text{ total}} \right. \\
&\qquad + \underbrace{\frac{1}{n - x_1} + \frac{1}{n - x_1 - 1} + \ldots + \frac{1}{n - (x_1 + x_2) + 1}}_{x_2 \text{ total}} \\
&\qquad \left. + (x_3) \text{ total decomposed terms for } \frac{x_3}{n - (x_1 + x_2)} + \ldots \right] \\
&\leqslant \text{OPT} \cdot (1 + 1/2 + \ldots + 1/n) = \text{OPT} \cdot \Theta(\log n). \qquad \square
\end{aligned}$$