Lecture 17: Approximation Algorithms

- Definitions
- Vertex Cover
- Set Cover
- Partition

Approximation Algorithms and Schemes

Let C_{opt} be the cost of the optimal algorithm for a problem of size n. An approximation algorithm for this problem has an approximation ratio $\varrho(n)$ if, for any input, the algorithm produces a solution of cost C such that:

$$max(\frac{C}{C_{opt}}, \frac{C_{opt}}{C}) \leq \varrho(n)$$

Such an algorithm is called a $\varrho(n)$-approximation algorithm.

An approximation scheme that takes as input $\epsilon > 0$ and produces a solution such that $C = (1 + \epsilon)C_{opt}$ for any fixed ϵ, is a $(1 + \epsilon)$-approximation algorithm.

A Polynomial Time Approximation Scheme (PTAS) is an approximation algorithm that runs in time polynomial in the size of the input, n. A Fully Polynomial Time Approximation Scheme (FPTAS) is an approximation algorithm that runs in time polynomial in both n and ϵ. For example, a $O(n^{2/\epsilon})$ approximation algorithm is a PTAS but not a FPTAS. A $O(n/\epsilon^2)$ approximation algorithm is a FPTAS.

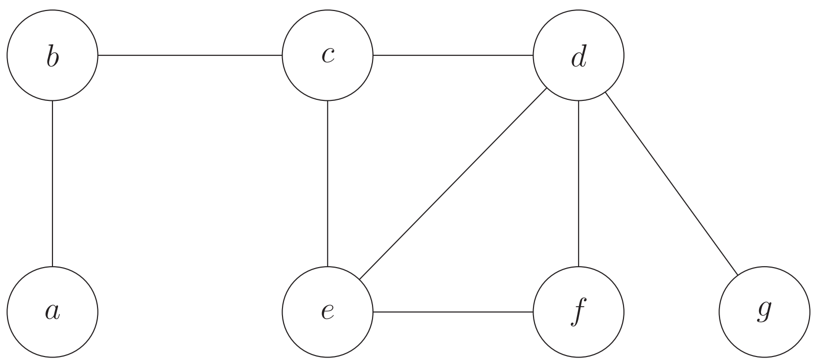
Vertex Cover

Given an undirected graph $G(V, E)$, find a subset $V' \subseteq V$ such that, for every edge $(u, v) \in E$, either $u \in V'$ or $v \in V'$ (or both). Furthermore, find a V' such that $|V'|$ is minimum. This is an NP-Complete problem.

Approximation Algorithm For Vertex Cover

Here we define algorithm *Approx_Vertex_Cover*, an approximation algorithm for Vertex Cover. Start with an empty set V'. While there are still edges in E, pick an edge (u, v) arbitrarily. Add both u and v into V'. Remove all edges incident on u or v. Repeat until there are no more edges left in E. *Approx_Vertex_Cover* runs in polynomial time.

Take for example the following graph G:



Approx_Vertex_Cover could pick edges (b, c), (e, f) and (d, g), such that $V' = \{b, c, e, f, d, g\}$ and $|V'| = 6$. Hence, the cost is $C = |V'| = 6$. The optimal solution for this example is $\{b, d, e\}$, hence $C_{opt} = 3$.

Claim: *Approx_Vertex_Cover* is a 2-approximation algorithm.

Proof: Let $U \subseteq V$ be the set of all the edges that are picked by *Approx_Vertex_Cover*. The optimal vertex cover must include at least one endpoint of each edge in U (and other edges). Furthermore, no two edges in U share an endpoint. Therefore, $|U|$ is a lower bound for C_{opt}. i.e. $C_{opt} \geq |U|$. The number of vertices in V' returned by *Approx_Vertex_Cover* is $2 \cdot |U|$. Therefore, $C = |V'| = 2 \cdot |U| \leq 2C_{opt}$. Hence $C \leq 2 \cdot C_{opt}$. $\square$

Set Cover

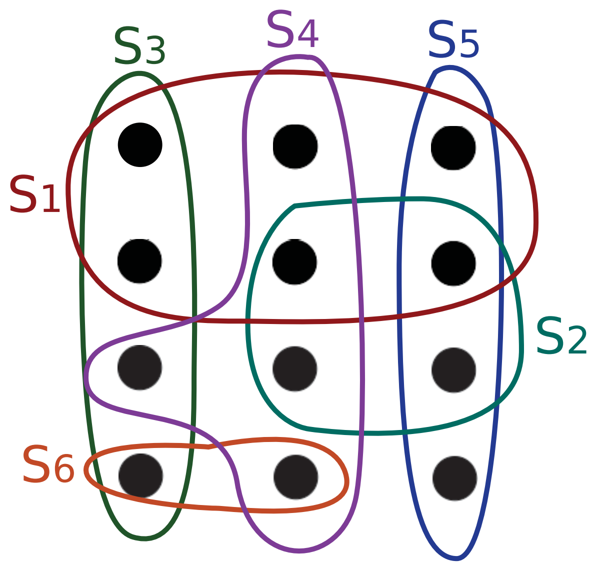
Given a set X and a family of (possibly overlapping) subsets $S_1, S_2, \cdots, S_m \subseteq X$ such that $\cup_{i=1}^{m} S_i = X$, find a set $P \subseteq \{1, 2, 3, \cdots, m\}$ such that $\cup_{i \in P} S_i = X$. Furthermore find a P such that $|P|$ is minimum.

Set Cover is an NP-Complete problem.

Approximation Algorithm for Set Cover

Here we define algorithm *Approx_Set_Cover*, an approximation algorithm for Set Cover. Start by initializing the set P to the empty set. While there are still elements in X, pick the largest set S_i and add i to P. Then remove all elements in S_i from X and all other subsets S_j. Repeat until there are no more elements in X. *Approx_Set_Cover* runs in polynomial time.

In the following example, each dot is an element in X and each S_i are subsets of X.



Approx_Set_Cover selects sets S_1, S_4, S_5, S_3 in that order. Therefore it returns $P = \{1, 4, 5, 3\}$ and its cost $C = |P| = 4$. The optimal solution is $P_{opt} = \{S_3, S_4, S_5\}$ and $C_{opt} = |P_{opt}| = 3$.

Claim: *Approx_Set_Cover* is a $(\ln(n)+1)$-approximation algorithm (where $n = |X|$).

Proof: Let the optimal cover be P_{opt} such that $C_{opt} = |P_{opt}| = t$. Let X_k be the set of elements remaining in iteration k of *Approx_Set_Cover*. Hence, $X_0 = X$. Then:

- for all k, X_k can be covered by t sets (from the optimal solution)
- one of them covers at least $\frac{|X_k|}{t}$ elements
- *Approx_Set_Cover* picks a set of (current) size $\geq \frac{|X_k|}{t}$

- for all k, $|X_{k+1}| \leq (1-\frac{1}{t})|X_k|$ (More careful analysis (see CLRS, Ch. 35) relates $\varrho(n)$ to harmonic numbers. t should shrink.)
- for all k, $|X_{k+1}| \leq (1-\frac{1}{t})^k \cdot n \leq e^{-k/t} \cdot n$ $(n = |X_0|)$

Algorithm terminates when $|X_k| < 1$, i.e., $|X_k| = 0$ and will have cost $C = k$.

$$e^{-k/t} \cdot n < 1$$

$$e^{k/t} > n$$

Hence algorithm terminates when $\frac{k}{t} > \ln(n)$. Therefore $\frac{k}{t} = \frac{C}{C_{opt}} \leq \ln(n) + 1$. Hence *Approx_Set_Cover* is a $(\ln(n) + 1)$-approximation algorithm for Set Cover. □

Notice that the approximation ratio gets worse for larger problems as it changes with n.

Partition

The input is a set $S = \{1, 2, \cdots, n\}$ of n items with weights $s_1, s_2, \cdots, s_n$. Assume, without loss of generality, that the items are ordered such that $s_1 \geq s_2 \geq \cdots \geq s_n$. Partition S into sets A and B to minimize $\max(w(A), w(B))$, where $w(A) = \sum_{i \in A} S_i$ and $w(B) = \sum_{j \in B} S_j$.

Define $2L = \sum_{i=1}^{n} s_i = w(S)$. Then optimal solution will have cost $C_{opt} \geq L$ by definition.

Partition is an NP-Complete problem. Want to find a PTAS $(1 + \epsilon)$-approximation. (Note that 2-approximation in this case is trivial). Also, an FPTAS also exists for this problem.

Approximation Algorithm for Partition

Here we define *Approx_Partition*. Define $m = \lceil \frac{1}{\epsilon} \rceil - 1$. $(\epsilon \approx \frac{1}{m+1})$ The algorithm proceeds in two phases.

First Phase: Find an optimal partition A', B' of $s_1, \cdots, s_m$. This takes $O(2^m)$ time.

Second Phase: Initialize sets A and B to A' and B' respectively. Hence they already contain a partition of elements $s_1, \cdots, s_m$. Then, for each i, where i goes from $m+1$ to n, if $w(A) \leq w(B)$, add i to A, otherwise add i to B.

Claim: *Approx_Partition* is a PTAS for Partition.

Proof: Without loss of generality, assume $w(A) \geq w(B)$. Then the approximation ratio is $\frac{C}{C_{opt}} = \frac{w(A)}{L}$. Let k be the last item added to A. There are two cases, either k was added in the first phase, or in the second phase.

Case 1: k is added to A in the first phase. This means that $A = A'$. We have an optimal partition since we can't do better than $w(A')$ when we have $n \geq m$ items, and we know that $w(A')$ is optimal for the m items.

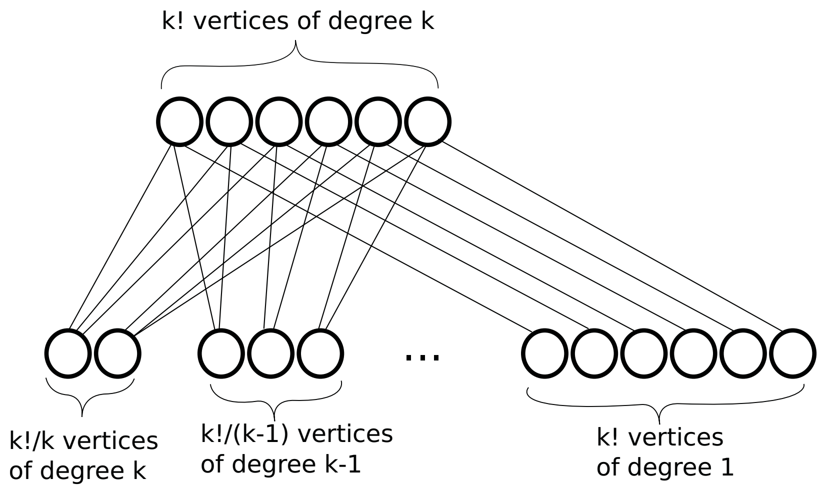
Case 2: k is added to A in the second phase. Here we know $w(A) - s_k \leq w(B)$ since this is why k was added to A and not to B. (Note that $w(B)$ may have increased after this last addition to A). Now, because $w(A) + w(B) = 2L$, $w(A) - s_k \leq w(B) = 2L - w(A)$. Therefore $w(A) \leq L + \frac{s_k}{2}$. Since $s_1 \geq s_2 \geq \cdots \geq s_n$, we can say that $s_1, s_2, \cdots, s_m \geq s_k$. Now since $k > m$, $2L \geq (m+1)s_k$.

Now, $\frac{w(A)}{L} \leq \dfrac{L + \frac{s_k}{2}}{L} = 1 + \frac{s_k}{2L} \leq 1 + \frac{s_k}{(m+1) \cdot s_k} = 1 + \frac{1}{m+1} = 1 + \epsilon$. Hence *Approx_Partition* is a $(1+\epsilon)$-approximation for Partition. $\square$

Natural Vertex Cover Approximation

Here we describe *Approx_Vertex_Cover_Natural*, a different approximation algorithm for Vertex Cover. Start with an empty set V'. While there are still edges left in E, pick the vertex $v \in V$ that has maximum degree and add it to V'. Then remove v and all incident edges from E. Repeat until no more edges left in E. In the end, return V'.

The following example shows a bad-case example for *Approx_Vertex_Cover_Natural*. In the example, the optimal cover will pick the $k!$ vertices at the top.



Approx_Vertex_Cover_Natural could possibly pick all the bottom vertices from left to right in order. Hence the cost could be $k! \cdot (\frac{1}{k} + \frac{1}{k-1} + \cdots + 1) \approx k! \log k$. Which is a factor of $\log k$ worse than optimal.

Claim: *Approx_Vertex_Cover_Natural* is a $(\log n)$-approximation.

Proof: Let G_k be the graph after iteration k of the algorithm. And let n be the number of edges in the graph, i.e. $|G| = n = |E|$. With each iteration, the algorithm selects a vertex and deletes it along with all incident edges. Let $m = C_{opt}$ be the number of vertices in the optimal vertex cover for G. Then let's look at the first m iterations of the algorithm: $G_0 \to G_1 \to G_2 \to \cdots \to G_m$.

Let d_i be the degree of the maximum degree vertex of G_{i-1}. Then the algorithm deletes all edges incident on that vertex to get G_i. Therefore:

$$|G_m| = |G_0| - \sum_{i=1}^{m} d_i$$

Also:

$$\sum_{i=1}^{m} d_i \geq \sum_{i=1}^{m} \frac{|G_{i-1}|}{m}$$

This is true because given $|G_{i-1}|$ edges that can be covered by m vertices, we know that there is a vertex with degree at least $\frac{|G_{i-1}|}{m}$. Then:

$$\sum_{i=1}^{m} \frac{|G_{i-1}|}{m} \geq \sum_{i=1}^{m} \frac{|G_m|}{m} = |G_m|$$

This is true since $|G_i| \leq |G_{i-1}|$ for all i. Then, it follows:

$$|G_0| - |G_m| \geq |G_m|$$

Because $|G_m| \leq \sum_{i=1}^{m} d_i$. Hence after m iterations, the algorithm will have deleted half or more edges from G_0. And generally, since every m iterations it will halve the number of edges in the graph, in $m \cdot \log |G_0|$ iterations, it will have deleted all the edges. And since with each iteration it addes 1 vertex to the cover, it will end up with a vertex cover of size $m \cdot \log |G_0| = m \cdot \log n$. Since we assumed that m was the size of the optimal vertex cover, $\frac{C}{C_{opt}} = \frac{m \log n}{m} = \log n$. Hence *Approx_Vertex_Cover_Natural* is a $(\log n)$-approximation. □

Note that since $n \approx k! \log k$ in the example of Figure , the worst-case example is $\log k \approx \log \log n$ worse, but we have only shown an $O(\log n)$ approximation.