Design and Analysis of Algorithms (VIII)

An Introduction to Approximation Algorithms

Guoqiang Li

School of Software, Shanghai Jiao Tong University

NP-Hard and Optimization Problems

Combinatorial optimization is a topic that consists of finding an optimal object from a finite set of objects.

Most natural optimization problems, including those arising in application areas, are NP-hard. Exhaustive search is not feasible.

Under the widely believed conjecture that $P \neq NP$, their exact solution is prohibitively time consuming.

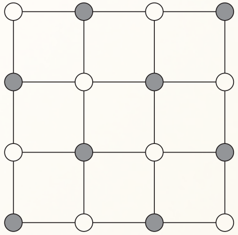
Approximability of these problems becomes a compelling subject of scientific inquiry in computer science and mathematics.

Vertex cover

Vertex cover

Given an undirected graph $G = (V, E)$, and a cost function on vertices $c : V \to \mathbb{Q}^+$, find a minimum cost vertex cover, i.e., a set $V' \subseteq V$ such that every edge has at least one endpoint incident at V'.

The special case, in which all vertices are of unit cost, will be called the cardinality vertex cover problem.



NP-Optimization Problem

An NP-optimization problem Π is either a minimization or a maximization problem.

Each valid instance I of Π comes with a nonempty set of feasible solutions, each of which is assigned a nonnegative rational number called its objective function value.

There exist polynomial time algorithms for determining validity, feasibility, and the objective function value.

A feasible solution that achieves the optimal objective function value is called an optimal solution.

NP-Optimization Problem

$OPT_{\Pi}(I)$ denotes the objective function value of an optimal solution to instance I. OPT is used when there is no ambiguity.

An approximation algorithm, $\mathcal{A}$, for Π is in polynomial time. A feasible solution of objective function value is "close" to the optimal.

By "close" we mean within a guaranteed factor of the optimal.

A Dilemma

To establish the approximation guarantee, the cost of the solution produced by the algorithm needs to compare with an optimal solution.

For such problems, not only is it NP-hard to find an optimal solution, but it is also NP-hard to compute the cost of an optimal solution.

In fact, computing the cost of an optimal solution is precisely the difficult core of such problems.

How do we establish the approximation guarantee? The answer provides a key step in the design of approximation algorithms.

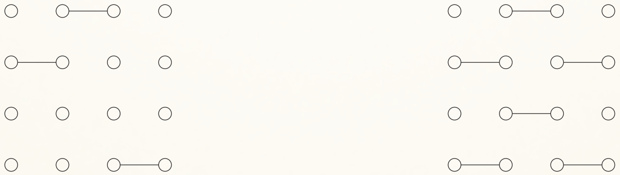
Cardinality Vertex Cover

Matching

Given a graph $G = (V, E)$, a subset of the edges $M \subseteq E$ is said to be a matching if no two edges of M share an endpoint.

A matching of maximum cardinality in G is called a maximum matching.

A matching that is maximal under inclusion is called a maximal matching.



Matching

Given a graph $G = (V, E)$, a subset of the edges $M \subseteq E$ is said to be a matching if no two edges of M share an endpoint.

A matching of maximum cardinality in G is called a maximum matching.

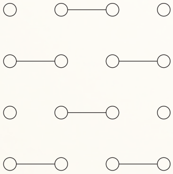
A matching that is maximal under inclusion is called a maximal matching.

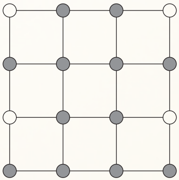
A maximal matching can clearly be computed in polynomial time by simply greedily picking edges and removing endpoints of picked edges. More sophisticated means lead to polynomial time algorithms for finding a maximum matching as well.

Approximation for Cardinality VC

Algorithm

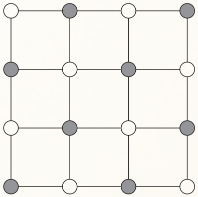
Find a maximal matching in G and output the set of matched vertices.





Approximation Factor

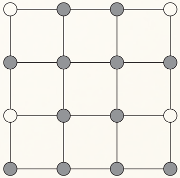
The Algorithm is a factor 2 approximation algorithm for the cardinality vertex cover problem.



Proof.

- No edge can be left uncovered by the set of vertices picked.
- Let M be the matching picked. As argued above,

$$|M| \leq OPT$$



- The approximation factor is at most $2 \cdot OPT$.

Lower Bounding OPT

The approximation algorithm for vertex cover was very much related to, and followed naturally from, the lower bounding scheme. This is in fact typical in the design of approximation algorithms.

Can the Guarantee be Improved?

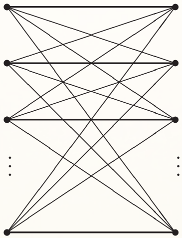
Can the approximation guarantee of Algorithm be improved by a better analysis?

Can an approximation algorithm with a better guarantee be designed using the lower bounding scheme of Algorithm?

Is there some other lower bounding method that can lead to an improved approximation guarantee for Vertex cover?

A Better Analysis?

Consider the infinite family of instances given by the complete bipartite graphs $K_{n,n}$.



When run on $K_{n,n}$, Algorithm will pick all $2n$ vertices, whereas picking one side of the bipartition gives a cover of size n.

Tight Example

$K_{n,n}$ shows that the analysis is tight, by giving an infinite family of instances in which the solution is twice the optimal.

An infinite family of instances showing that the analysis of an approximation algorithm is tight, is referred to as a tight example.

Tight examples for an approximation algorithm give critical insight into the functioning of the algorithm.

They have often led to ideas for obtaining algorithms with improved guarantees.

A Better Guarantee?

The lower bound, of size of a maximal matching, is half the size of an optimal vertex cover for the following infinite family of instances. Consider the complete graph K_n, where n is odd. The size of any maximal matching is $(n-1)/2$, whereas the size of an optimal cover is $n-1$.

A Better Algorithm?

Still Open!

Set Cover

Set cover

Set cover

Given a universe U of n elements, a collection of subsets of U, $\mathcal{S} = \{S_1, \ldots, S_k\}$, and a cost function $c : \mathcal{S} \to \mathbb{Q}^+$, find a minimum cost sub-collection of $\mathcal{S}$ that covers all elements of U.

Some Remarks

Define the frequency f of an element to be the number of sets it is in.

The various approximation algorithms for set cover achieve one of two factors: $O(\log n)$ or f.

Clearly, neither dominates the other in all instances.

The special case of set cover with $f = 2$ is essentially the vertex cover problem, for which we gave a factor 2 approximation algorithm.

Cardinality Set Cover

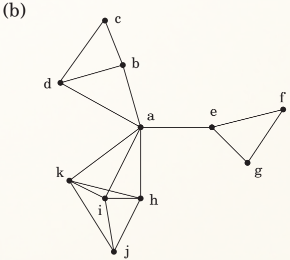
The Problem

A county is in its early stages of planning and is deciding where to put schools.

There are only two constraints:

- each school should be in a town,
- and no one should have to travel more than 30 miles to reach one of them.

Q: What is the minimum number of schools needed?



The Problem

This is a typical (cardinality) set cover problem.

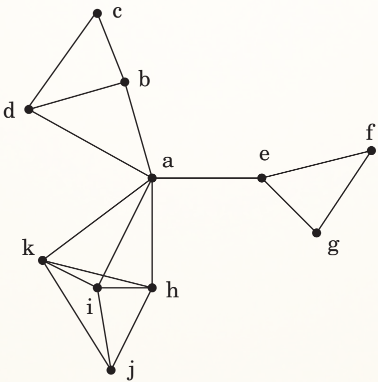
- For each town x, let S_x be the set of towns within 30 miles of it.
- A school at x will essentially "cover" these other towns.
- The question is then, how many sets S_x must be picked in order to cover all the towns in the county?

Set Cover Problem

Set Cover

- Input: A set of elements B, sets $S_1, \ldots, S_m \subseteq B$
- Output: A selection of the S_i whose union is B.
- Cost: Number of sets picked.

The Example



Performance Ratio

Lemma

Suppose B contains n elements and that the optimal cover consists of OPT sets. Then the greedy algorithm will use at most $\ln n \cdot OPT$ sets.

Proof:

Let n_t be the number of elements still not covered after t iterations of the greedy algorithm (so $n_0 = n$).

Since these remaining elements are covered by the optimal OPT sets, there must be some set with at least n_t/OPT of them.

Therefore, the greedy strategy will ensure that

$$n_{t+1} \leq n_t - \frac{n_t}{OPT} = n_t(1 - \frac{1}{OPT})$$

which by repeated application implies

$$n_t \leq n_0(1 - \frac{1}{OPT})^t$$

Performance Ratio

A more convenient bound can be obtained from the useful inequality

$$1 - x \leq e^{-x} \text{ for all } x$$

with equality if and only if $x = 0$,

Thus

$$n_t \leq n_0\left(1 - \frac{1}{OPT}\right)^t < n_0\left(e^{-\frac{1}{OPT}}\right)^t = ne^{-\frac{t}{OPT}}$$

At $t = \ln n \cdot OPT$, therefore, n_t is strictly less than $ne^{-\ln n} = 1$, which means no elements remain to be covered.

Set Cover

The Greedy Algorithm

The Greedy Strategies

Iteratively pick the most cost-effective set and remove the covered elements, until all elements are covered.

Let C be the set of elements already covered at the beginning of an iteration.

During this iteration, define the cost-effectiveness of a set S to be the average cost at which it covers new elements, i.e., $\dfrac{c(S)}{|S - C|}$.

Define the price of an element to be the average cost at which it is covered.

When a set S is picked, we can think of its cost being distributed equally among the new elements covered, to set their prices.

The Greedy Algorithm

1. $C \leftarrow \emptyset$.
2. While $C \neq U$ do
   - Find the most cost-effective set in the current iteration, say S.
   - Let $\alpha = \frac{c(S)}{|S-C|}$, i.e., the cost-effectiveness of S.
   - Pick S, and for each $e \in S - C$, set $price(e) = \alpha$.
   - $C \leftarrow C \cup S$.
3. Output the picked sets.

The Lemma

Lemma

Number the elements of U in the order in which they were covered by the algorithm, resolving ties arbitrarily. Let $e_1, \ldots, e_n$ be this numbering.

For each $k \in \{1, \ldots, n\}$, $price(e_k) \leq OPT/(n-k+1)$.

The Lemma

Proof.
In any iteration, the leftover sets of the optimal solution can cover the remaining elements at a cost of at most OPT.

Therefore, among these sets, there must be one having cost-effectiveness of at most $OPT/|\overline{C}|$.

$$price(e_k) \leq \frac{OPT}{|\overline{C}|}$$

In the iteration in which element e_k was covered, $\overline{C}$ contained at least $n - k + 1$ elements.

$$price(e_k) \leq \frac{OPT}{|\overline{C}|} \leq \frac{OPT}{n - k + 1}$$

The Approximation Factor

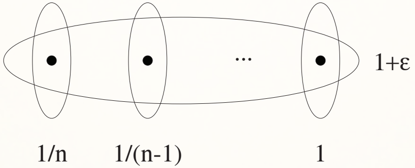
Theorem

The greedy algorithm is an H_n factor approximation algorithm for the minimum set cover problem, where

$$H_n = 1 + \frac{1}{2} + \ldots + \frac{1}{n}$$

A Tight Example

The following is a tight example



When run on this instance the greedy algorithm outputs the cover consisting of the n singleton sets, since in each iteration some singleton is the most cost-effective set. Thus, the algorithm outputs a cover of cost

$$= \frac{1}{n} + \frac{1}{n-1} + \ldots + 1 = H_n$$

On the other hand, the optimal cover has a cost of $1 + \varepsilon$.

Layering

Layering

The algorithm design technique of layering is also best introduced via set cover. However, that this is not a very widely applicable technique.

We will give a factor 2 approximation algorithm for vertex cover, assuming arbitrary weights.

The idea in layering is to decompose the given weight function on vertices into convenient functions, called degree-weighted, on a nested sequence of subgraphs of G.

The Lemma

Let $\omega : V \to \mathbb{Q}^+$ be the function assigning weights to the vertices of the given graph $G = (V, E)$.

A function assigning vertex weights is degree-weighted if there is a constant $c > 0$ such that the weight of each vertex $v \in V$ is $c \cdot deg(v)$.

Lemma

In Vertex Cover, let $\omega : V \to \mathbb{Q}^+$ be a degree-weighted function. Then $\omega(V) \leq 2 \cdot OPT$.

The Lemma

Lemma

In Vertex Cover, let $\omega : V \to \mathbb{Q}^+$ be a degree-weighted function. Then $\omega(V) \leq 2 \cdot OPT$.

Proof.

Let c be the constant such that $\omega(v) = c \cdot deg(v)$, and let U be an optimal vertex cover in G.

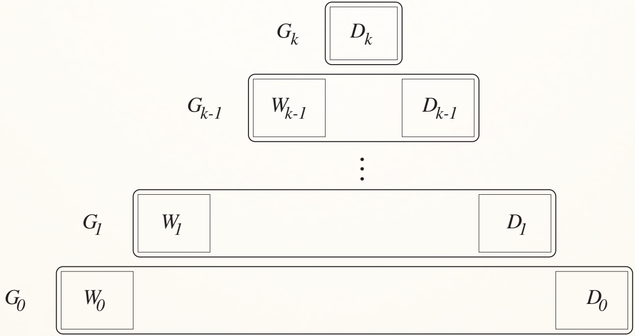
Since U covers all the edges, $\sum_{v \in U} deg(v) \geq |E|$.

Therefore, $\omega(U) \geq c|E|$. Since $\sum_{v \in V} deg(v) = 2|E|$, $\omega(V) = 2c|E|$. The lemma follows.

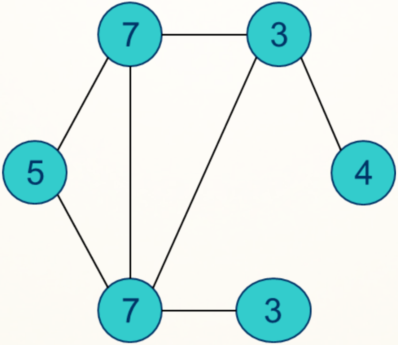
The Layer Algorithm

1. $G_0 = G$, $C = \varnothing$, $i = 0$.
2. Remove degree zero vertices from G_i, say this set is D_i.
3. Compute $c = \min\{w(v)/\deg(v)\}$ for all $v \in G_i$.
4. Let $t_i(v) = c \cdot \deg(v)$ and $w(v) = w(v) - t_i(v)$ for all $v \in G_i$.
5. Let $W_i = \{v \in G_i \mid w(v) = 0\}$, $C = C \cup W_i$.
6. Let G_{i+1} be the graph induced by $V_i - (D_i \cup W_i)$. Increase i by 1 and goto step 2 until G_i is empty graph.

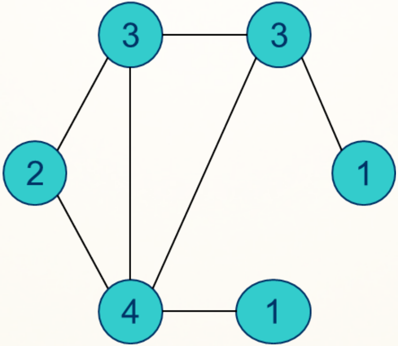
The Layer Algorithm



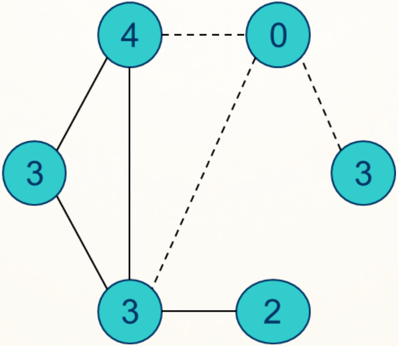
An Example



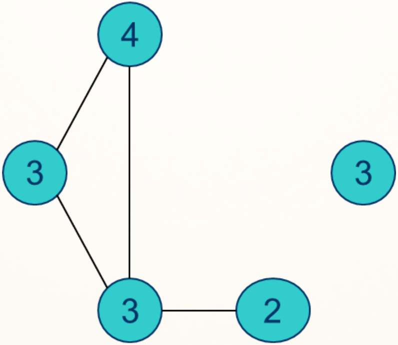
An Example



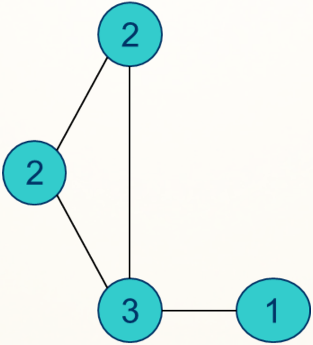
An Example



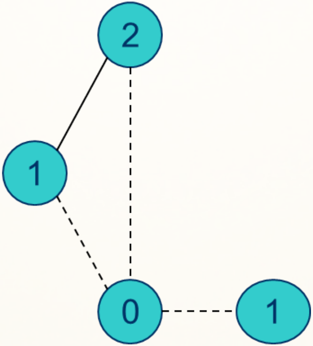
An Example



An Example



An Example



Analysis

Theorem

The layer algorithm achieves an approximation guarantee of factor 2 for the vertex cover problem, assuming arbitrary vertex weights.

Proof.

- Firstly, C is a vertex cover for G.
- Otherwise, there must be some $(u, v) \in E$ with $u \in D_i$ and $v \in D_j$.
- Assume $i \leq j$, then (u, v) is in G_i contradicting the fact that u is of degree zero.

Analysis (cont'd)

Then we show $\omega(c) \leq 2 \cdot \mathbf{OPT}$. Let C^* be an optimal vertex cover.

- For $v \in C$, if $v \in W_j$
- $$\omega(v) = \sum_{i \leq j} t_i(v)$$
- For $v \in V - C$, if $v \in D_j$, then
- $$\omega(v) \geq \sum_{i < j} t_i(v)$$

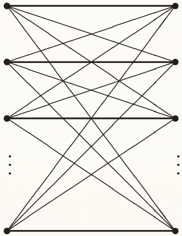
Analysis (cont'd)

- In each layer i, $C^* \cap G_i$ is a vertex cover for G_i.
- Thus by previous lemma, $t_i(C \cap G_i) \le 2 \cdot t_i(C^* \cap G_i)$.
- Therefore,

$$\omega(C) = \sum_{i=0}^{k-1} t_i(C \cap G_i) \le 2 \sum_{0}^{k-1} t_i(C^* \cap G_i) \le 2 \cdot \omega(C^*)$$

A Tight Example

A tight example is provided by the family of complete bipartite graphs, $K_{n,n}$, with all vertices of unit weight. The layering algorithm will pick all $2n$ vertices of $K_{n,n}$ in the cover, whereas the optimal cover picks only one side of the bipartition.



Referred Materials

Content of this lecture comes from Chapter 1 and 2 in [Vaz04].

Suggest to read the rest part of Chapter 1 and 2 in [Vaz04].