TTIC 31290: Machine Learning for Algorithm Design (Fall 2025)

Avrim Blum and Dravyansh Sharma

Lecture 1: 09/30/25 Lecturer: Avrim Blum

Outline for today

- Course overview
- Approximation algorithms overview
- Case study #1: k-center clustering
- Case study #2: Set cover
- Case study #3: Vertex cover

1 Course Overview

[See material on course webpage]

2 Approximation Algorithms Overview

Many computational problems we'd like to solve are NP-hard, so we don't expect to be able to find an algorithm that can optimally solve arbitrary instances in polynomial time. [If needed: brief discussion of polynomial time, NP-hardness] One classic approach to addressing this is to study approximation algorithms.

We say an algorithm is an α-approximation for a *minimization* problem Π if for any instance I, the algorithm finds a solution of size at most α times the minimum. We say an algorithm is an α-approximation for a *maximization* problem Π if for any instance I, the algorithm finds a solution of size at least α times the maximum.

3 Case study #1: k-center clustering

Given n points $x_1, \ldots, x_n$ in a metric space M, the goal of k-center clustering is to find k "cluster centers" $c_1, \ldots, c_k \in M$ that minimize:

$$\max_i \min_j d(x_i, c_j)$$

i.e., minimize the maximum distance between any x_i and its nearest cluster center. Equivalently, we want to find centers $c_1, \ldots, c_k$ such that the balls of radius r around each center contain all the x_i's, for r as small as possible.

This problem is NP-hard, but there is a simple algorithm that gives a 2-approximation (i.e., if r^* is the minimum possible radius, then this finds a solution that works with radius $2r^*$).

Algorithm (Farthest Point Algorithm)

1. Pick $c_1 = x_1$ (or any of the x_i; it doesn't matter).
2. Pick c_2 to be the point x_i that is farthest from c_1.
3. For $j = 3, \ldots, k$, pick c_j to be the point x_i that is farthest from $\{c_1, \ldots, c_{j-1}\}$, specifically:

$$c_j = \arg\max_{x_i} \min_{j'<j} d(x_i, c_{j'})$$

Theorem 1. *The Farthest Point Algorithm is a 2-approximation for the k-center problem.*

Proof. Let r be the radius of the solution found by the algorithm:

$$r = \max_i \min_j d(x_i, c_j)$$

Let $x_i = \arg\max_i \min_j d(x_i, c_j)$. Notice that the $k+1$ points $\{c_1, c_2, \ldots, c_k, x_i\}$ all have distance at least r from each other [do you see why?]. This means there cannot exist a solution with radius $r^* < r/2$. Indeed, any ball of radius less than $r/2$ cannot contain more than one of these $k+1$ points (by the triangle inequality), and so there is no way to cover all $k+1$ points with only k such balls. □

Note: it is NP-hard to get any constant approximation less than 2.

4 Case study #2: Set Cover

The set cover problem is defined as follows: you have a universe X of n points $\{x_1, \ldots, x_n\}$ and m subsets $S_1, \ldots, S_m \subseteq X$. Assume that each x_i is in at least one subset S_j. You want to find the fewest subsets needed to cover all of X.

Notice that set cover is very similar to k-center clustering if instead of approximately minimizing the radius r, you fix r and aim to approximately minimize the number of centers. In particular, given a k-center problem in a finite metric space M with m total points, we can create one set for each ball of radius r around a point in M. In the other direction, given a set cover instance, we can create a bipartite graph with $x_1, \ldots, x_n$ on one side and $S_1, \ldots, S_m$ on the other, with an edge of length 1 between x_i and S_j if S_j contains x_i, and then look at $r = 1$ in the shortest-path metric.

Set cover is NP-hard, but there is an $O(\log n)$-approximation.

Greedy algorithm for set cover

Until done, choose the set that covers the most new points.

Theorem 2. *The greedy algorithm is an $O(\log n)$-approximation for set cover.*

Proof. Let k be the size of the minimum set cover. At any step, there must be at least one available set that covers at least a $1/k$ fraction of the points remaining. Thus, the algorithm chooses one that covers at least this fraction. After t steps, the number of uncovered points is at most:

$$n\left(1-\frac{1}{k}\right)^t$$

Using $(1+x) \leq e^x$ (true for all x, equality at $x=0$), we have:

$$n\left(1-\frac{1}{k}\right)^t \leq ne^{-t/k}$$

After $t = k\ln(n/k)$ steps, at most k points remain; after at most k more steps we are done. So the total number of sets chosen is at most:

$$k\left[1+\ln\left(\frac{n}{k}\right)\right] = O(k\log n)$$

□

Note: It is NP-hard to get even a $(1-\varepsilon)\ln n$-approximation, for any constant $\varepsilon > 0$.

5 Case study #3: Vertex Cover

The vertex cover problem is: given a graph G, find the fewest vertices needed to cover all the edges (i.e., pick at least one endpoint of every edge). Vertex cover is a special case of set cover: each edge is an "item" and each vertex is a set covering the edges it touches.

The greedy set cover algorithm, if applied to vertex cover, corresponds to picking the vertex covering the most new edges, giving an $O(\log n)$-approximation. However, there is a different greedy algorithm that gives a 2-approximation.

Greedy algorithm for vertex cover

Until done, find an uncovered edge and choose *both* endpoints.

Theorem 3. *This algorithm is a 2-approximation for vertex cover.*

Proof. Let $e_1, e_2, \ldots, e_k$ be the edges chosen in sequence. The algorithm picks $2k$ vertices. These edges share no endpoints, because each e_i is uncovered after choosing endpoints of previous edges. Any vertex cover must include at least one endpoint from each e_i, so it must have size at least k. Thus the algorithm's solution is at most twice optimal. □

Note: It is UGC-hard to approximate vertex cover within any factor better than $2-\varepsilon$.

6 Additional Resources

- Julia Chuzhoy's Approximation Algorithms course. `https://canvas.uchicago.edu/courses/51962`
- Chandra Chekuri's Approximation Algorithms course. `https://courses.grainger.illinois.edu/cs598csc/sp2011/`