CS-GY 6043 Lecture 10 Approximation Algorithm II

By Dan Mao

updated on 12/17/2023

Introduction

Building upon our previous lecture, we continue to explore more approximation algorithms, concentrating on the following four problems:

1. Use the Greedy Algorithm to approach the Set Cover problem.
2. Use the Gonzalez Approximation Algorithm to approach the k-center problem.
3. Use Maximal Matching to approach Maximum Matching.
4. Use LP Rounding technique to approximate the Lightest Vertex Cover.

We then moved with a brief comparative overview of deterministic and randomization algorithms.

Use the Greedy Algorithm to approach the Set Cover problem

Problem Definition

Given a collection S of sets over a universe U, a set cover $C \subseteq S$ is a subcollection of the sets whose union is U. The set-cover problem is, given S, to find a smallest (minimum-cardinality) set cover.

Implementing the Greedy Algorithms

The greedy algorithm for unweighted set cover builds a cover by repeatedly choosing a set S that covers the maximum number of element left uncovered, denoted as n_i, after picking the i^{th} set. It stops and returns the chosen sets when they form a cover.

Greedy Set Cover Algorithm

Initialize $C \leftarrow \emptyset$. Define $f(C) = |\cup_{s \in C} s|$.

While $f(C) < f(S)$:

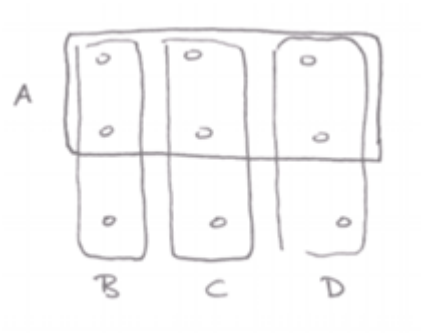
$s \leftarrow \arg\max_{s \in S} f(s - C)$

Update $C \leftarrow C \cup s$

Return C

Here is an example demonstrating how it works:

The optimal choice is to cover the following nine point with sets B, C, and D, the greedy algorithm will choose A and then be forced to select the rest of the sets. [1]



What is the α of the approximation?

As we discussed in our previous class, finding an approximation is not our only interest, we are also interested in demonstrating that approximation algorithms can yield solutions with provable guarantees concerning their closeness to the optimal solution.

In this context, we examine the set cover obtained from the greedy algorithm compared to the optimal minimum set cover. It's important to note that with the greedy algorithm, acquiring one set at each step effectively means that counting the size of the set cover is equivalent to counting the steps taken to cover the entire set U.

Claim: The greedy algorithm provides a $\ln(n)$-approximation for the set cover problem.

Proof:

Let $n = |U|$ represent the total number of elements, k denote the number of sets in the optimal minimum set cover, and n_i signify the number of elements remaining uncovered after selecting the i^{th} set.

Initially, $n_0 = n$. After the i^{th} step of selecting a subset from S, n_i elements remain, which are covered by the optimal minimum set. According to the Pigeonhole Principle, for any natural numbers k and m, if $n = km + 1$ objects are distributed among m sets, at least one of these sets must contain at least $k + 1$ objects. Consequently, there exists a set in the optimal minimum set cover that covers at least $\frac{n_i}{k}$ elements. Therefore, the greedy algorithm selects at least $\frac{n_i}{k}$ elements.

This leads to the following inequality:

$$n_{i+1} \leq n_i - \frac{n_i}{k} = n_i \left(1 - \frac{1}{k}\right)$$

By induction, this inequality simplifies to:

$$n_i \leq n_0 \left(1 - \frac{1}{k}\right)^i$$

Since $n_i \geq 0$ and n_i is an integer by define, $n_i < 1$ implies $n_i = 0$, so we found a cover. From the previous inequality, we aim to establish a relationship between i and k for when $n_i < 1$. To simplify this relationship, we use the inequality $1 - x \leq e^{-x}$, which leads to:

$$n_i \leq n_0 \left(1 - \frac{1}{k}\right)^i < n_0 e^{-\frac{i}{k}} = n e^{-\frac{i}{k}}$$

Given that the stopping criterion for the greedy algorithm is when $n_i < 1$, setting $n_0 e^{-\frac{i}{k}} = 1$ yields:

$$-\frac{i}{k} = \ln\left(\frac{1}{n}\right)$$
$$i = k\ln(n)$$

This shows that the greedy algorithm will provide a set cover within $k\ln(n)$ steps, and thus we have $k \leq c \leq k\ln(n)$, where c is the number of set we obtained from the greedy algorithm.

Therefore, we conclude that the greedy algorithm is a $\ln(n)$-approximation for the unweighted set cover problem.

Use the Gonzalez Approximation Algorithm for the k-center problem

Problem Definition

Based on Jeff Erickson's definition in his chapter on approximation algorithms: [2]

Overview:
The k-center clustering problem involves finding a set of k circles that collectively enclose a given set $P = \{p_1, p_2, \ldots, p_n\}$ of n points in the plane. The goal is to minimize the radius of the largest circle.

Formal Definition:
We aim to compute a set of center points $C = \{c_1, c_2, \ldots, c_k\}$ such that the following cost function is minimized:

$$\text{cost}(C) := \max_i \min_j |p_i c_j|.$$

Here, $|p_i c_j|$ denotes the Euclidean distance between input point p_i and center point c_j. Intuitively, each input point is assigned to its closest center point; the points assigned to a given center c_j comprise a cluster. The distance from c_j to the farthest point in its cluster is the radius of that cluster; the cluster is contained in a circle of this radius centered at c_j. The k-center clustering cost $\text{cost}(C)$ is precisely the maximum cluster radius.

Notation Clarification: [3]
While the original notation might be confusing, it can be decomposed as follows:

Denoting the distance of p_i to its center as:

$$d(p_i, C) = \min_{c \in C} d(p_i, c)$$

Given our cluster centers C, we will also denote the radius of C as the following:

$$r = \max_{p_i \in P} d(p_i, C)$$

We can then restate our objective as finding C that minimizes the radius of C. That is, we find C subject to

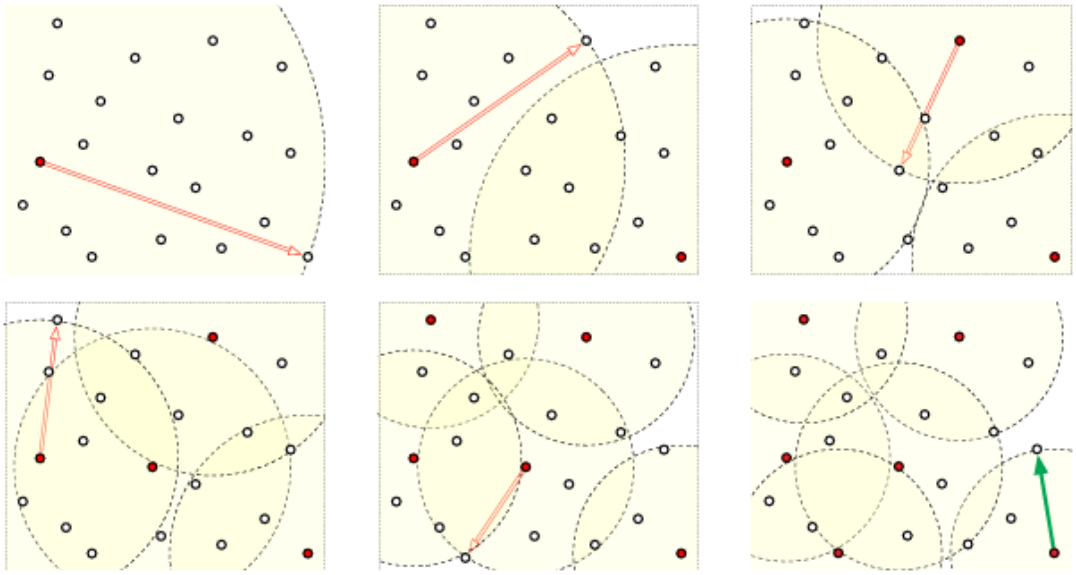
$$\min_{C \subseteq P : |C| = k} \max_{p_i \in P} d(p_i, C)$$

and output the $\text{cost}(C)$.

Scope of Applicability:
The k-center problem is applicable in any metric space, and the approximation analysis presented here is valid across these spaces. However, specific analyses in following proof require the points to be in the Euclidean plane.

Implementing the Gonzalez Approximation Algorithm

Choose the k center points one at a time, starting with an arbitrary input point as the first center. In each iteration, choose the input point that is farthest from any earlier center point to be the next center point. [2]



The first six iterations of Gonzalez's k-center clustering algorithm.

Gonzalez Approximation Algorithm (Modified from Ida Huihan Liu's Note) [3]

Pick arbitrary $p \in P$ and initialize $C = \{p\}$. Do while $|C| < k$:

$p \leftarrow \text{argmax}_{p \in P} d(p, C)$

Update $C \leftarrow C \cup \{p\}$

return $\text{cost}(C)$

Approximation Analysis

Claim: The Gonzalez Approximation algorithm yields a 2-approximation for the k-center problem.

Proof of 2-Approximation for the k-Clustering Problem by Gonzalez's Algorithm:

Let OPT denote the optimal k-center clustering radius for a set P. For any index i, let c_i and r_i denote the i-th center point and i-th clustering radius computed by the Gonzalez K-Center Algorithm.

- **Observation 1**: By construction, each center point c_j is at least r_{j-1} distance away from any other center point c_i, where $i < j$. Moreover, for any $i < j$, we have $r_i \geq r_j$. Therefore, for all indices i and j, $d(c_i, c_j) \geq r_k$.
- **Observation 2**: For any two nodes i and j within the same cluster of the optimal solution, their distance is at most $2 \cdot \text{OPT}$. By the triangle inequality:

$$d(i,j) \leq d(i,c) + d(j,c) \leq 2 \cdot \text{OPT}$$

where c is the center of the cluster.

Recall that the cost function, $\text{cost}(C) = r_k = \max_{p_i \in P} d(p_i, C)$, determines r_k by finding the $(k+1)$-th center and measuring its distance to the current set C. After the algorithm terminates, selecting an additional center results in $k+1$ points distributed across k optimal clusters. The pigeonhole principle implies that at least one optimal cluster contains two or more of these points, guaranteeing at least one pair of c_i and c_j within the same optimal cluster.

From Observation 2, if two nodes c_i and c_j are in the same optimal cluster, then $d(c_i, c_j) \leq 2 \cdot \text{OPT}$. Observation 1 implies $r_k \leq d(c_i, c_j)$ for all pairs $\{c_i, c_j\} \subset C$. Hence, we conclude:

$$r_k \leq d(c_i, c_j) \leq 2 \cdot \text{OPT}$$

This implies that the Gonzalez Approximation algorithm is indeed a 2-approximation for the k-clustering problem.

Use Maximal Matching to approach Maximum Matching and Minimum Cardinality Maximal Matching

Problem Definition

Consider an undirected graph $G = (V, E)$.

- **Matching:** A matching, M, of G is a subset of the edges E, such that no vertex in V is incident to more than one edge in M.

  Intuitively, we can say that no two edges in M have a common vertex.

- **Maximal Matching:** A matching M is said to be maximal if M is not properly contained in any other matching. Formally, $M \not\subset M'$ for any matching M' of G.

  Intuitively, this is equivalent to saying that a matching is maximal if we cannot add any edge to the existing set.

- **Maximum Matching:** A matching M is said to be Maximum if for any other matching M', $|M| \geq |M'|$. $|M|$ is the maximum size matching.
- **Minimum Cardinality Maximal Matching**: A matching M is said to be minimum if for any other maximal matching M', $|M| \leq |M'|$. $|M|$ is the Minimum size maximal matching.

Approximating Maximum Matching and Minimal Matching with Maximal Matching

A maximal matching for a given graph can be found using the following simple greedy algorithm:

Maximal Matching

$M = \emptyset$

While (no more edges can be added):

Select an edge e, not sharing any vertex with edges in M

$M = M \cup e$

Remove the end points of e and all edges incident to them

return M

Approximation Analysis

To assess the quality of approximation of maximal matching for Maximum Matching and Minimal Cardinality Matching, consider the following:

Let M_1, M_2 be two maximal matchings.

Claim: $\frac{1}{2}|M_1| \leq |M_2| \leq 2|M_1|$

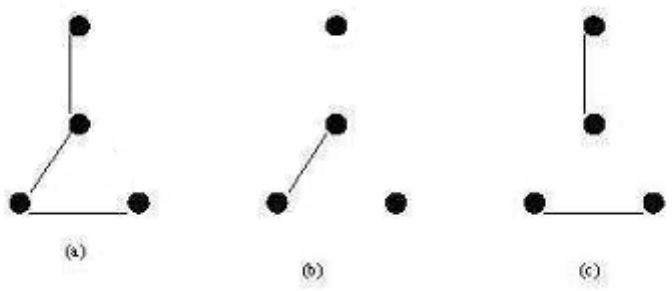
Intuition: *(Please be aware this is not a formal proof!)*

Consider a basic structure (e.g., a path with 4 nodes) that illustrates the size difference between a minimum and a maximum matching. In this structure: [4]

(a) represents the original graph.

(b) shows a minimum cardinality maximal matching.

(c) shows a maximum matching.



Removing the edges shared by M_1 and M_2, we observe that every edge in M_2 not in M_1 can intersect at most two edges in M_1. In scenarios where the graph comprises only this structure, the inequality $\frac{1}{2}|M_1| \leq |M_2| \leq 2|M_1|$ holds.

Importantly, a maximum matching is a form of maximal matching, as so does a minimum cardinality maximal matching. Therefore, if the inequality holds, the cardinality of a minimum cardinality maximal matching is bounded by $2|M_1|$, indicating that maximal matching is a 2-approximation for minimum cardinality maximal matching. Similarly, the inequality suggests that maximal matching is a 2-approximation for maximum matching.

Use LP Rounding technique to approximate the Minimum Weight Vertex Cover

Problem Definition

A **vertex cover** of a graph $G(E, V)$ is a set of vertices that includes at least one endpoint of every edge of the graph. Given a weight function $W : V \to R^+$, want to find a vertex cover X which minimizes $\sum_{v \in X} w(v)$.

The problem can be rephrased as the following integer linear programming problem (a mathematical optimization problem in which some or all of the variables are restricted to be integers):

Define a indicator function $1_X : V \to \{0, 1\}$, where $1_X(v) = 1$ if $v \in X$, and $1_X(v) = 0$ otherwise.

For a given graph G, the problem aims to minimize $\sum_{v \in X} w(v) \cdot 1_X(v)$, subject to $x_u + x_v \geq 1$ for every edge uv in the graph.

Apply LP Rounding technique [5]

Solving integer linear programs is NP-Hard. Instead, we use Linear Programming (LP) to approximate the optimal solution, `OPT(Input)`, for the integer program. In general, this approach allow us solve a linear programming relaxation to get a fractional solution. Then converting this fractional solution into a feasible solution by appropriate rounding.

In this particular case, we will first relax the constraints of the indicator function to $x_v \in [0, 1]$. Thus, a linear programming formulation for Vertex Cover is:

$$\text{minimize} \sum_{v \in V} w_v \cdot x_v$$

subject to

$$x_u + x_v \geq 1 \quad \forall e = (u, v) \in E$$
$$0 \leq x_v \leq 1$$

In the second step, we will convert this fractional solution into a feasible solution use the following algorithm:

Vertex Cover via LP:

Solve LP to obtain an optimal fractional solution x^*.

Let $x'_v = 1$ if $x^*_v \geq \frac{1}{2}$, otherwise $x'_v = 0$

$S = \{v \mid x'_v = 1\}$

Output S

S need to be a vertex cover in order for the algorithm work. So here we make the following claim:

Claim 1: S is a vertex cover.

Proof: Consider any edge, $e = (u, v)$. By feasibility of x^*, $x^*_u + x^*_v \geq 1$, and thus either $x^*_u \geq \frac{1}{2}$ or $x^*_v \geq \frac{1}{2}$, or both. Therefore, at least one of u and v will be in S.

Approximation Analysis

Claim 2: $w(S) \leq 2 \cdot \text{OPT}_{LP}(I)$

Proof:
Firstly, observe that $\frac{1}{2} x'_v \leq x^*_v$ for all v. Given this, consider the following inequality:

$$\text{OPT}_{LP}(\text{Input}) = \sum_v w_v x^*_v \geq \frac{1}{2} \sum_{v \in S} w_v = \frac{1}{2} w(S)$$

This inequality arises from the fact that the optimal solution to the Linear Programming (LP) version of the problem, $\text{OPT}_{LP}(\text{Input})$, is at least half of $w(S)$ due to the relaxation of constraints compared to the integer linear programming version.

Since the original ILP problem is a special case of the LP problem, the LP version can only yield a smaller or equal minimum weight, as it contains a broader range of feasible solutions. Thus, we can deduce:

$$\mathrm{OPT}(\mathrm{Input}) \geq \mathrm{OPT}_{LP}(\mathrm{Input}) \geq \frac{1}{2}w(S)$$

Therefore, we conclude that the weight of the approximated result set $w(S)$ is at most twice the optimal solution of the original problem. This implies that the algorithm provides a 2-approximation to the problem.

A brief Intro to Randomized Algorithm

A randomized algorithm is an algorithm that employs a degree of randomness as part of its logic or procedure. This approach aims to optimize performance in the "average case," considering all potential random inputs, or sometimes can succeed with high probability. As a result, the running time, the output, or both, become random variables. [6]

Here is a brief overview of the distinctions between randomized algorithm and deterministic Algorithm.

	Algorithm	**Randomized Algorithm**
Termination	Always terminates	Some always terminated, some terminated with high probability to terminated
Correctness	Always produces a correct result	Some always produce a correct result, some produces a correct result with high probability Key types include: - Monte Carlo algorithms, which have a bounded error probability (e.g., the bloom filter discussed in lecture 12). - Las Vegas algorithms, which either produce the correct result or report failure.
Performances	Evaluated based on worst-case time and space	Evaluated based on expected time and expected space

Summary

In conclusion, we explore two primary methods for approximating NP-hard problems, specifically focusing on greedy algorithms and LP (Linear Programming) Rounding.

1. **Greedy Algorithms**: These are straightforward, yet powerful techniques used for problem-solving and optimization. We delve into their application in several key NP-hard problems:
    - **Set Cover Problem**: This problem involves finding the smallest subset of sets that covers all elements.
    - **K-Center Problem**: Here, the objective is to choose 'k' centers to minimize the maximum distance to the nearest center.
    - **Maximum Matching**: This entails finding the largest matching in a graph, where a matching is a set of edges without common vertices.

2. **LP Rounding**: This approach involves solving a relaxation of an NP-hard problem through linear programming and then rounding the solution to get an integer result. We illustrate this method through its application in:
    - **Vertex Cover**: The task is to find the minimum number of vertices in a graph that touches all edges.

In addition, we briefly go through the basic idea of randomized algorithm, highlights their key differences from deterministic algorithms in terms of termination, correctness, and performance.

Reference

1. https://www.cs.ucr.edu/~neal/Young08SetCover.pdf "Set Cover Problem" ↩
2. https://ugtcs.berkeley.edu/src/approx-sp19/scribe-notes-2.pdf "Jeff Erickson's Algorithm Book" ↩ ↩
3. https://ugtcs.berkeley.edu/src/approx-sp19/scribe-notes-2.pdf "K-center Problem" ↩ ↩
4. https://www.cs.dartmouth.edu/~ac/Teach/CS105-Winter05/Notes/kavathekar-scribe.pdf "Maximum Matching" ↩
5. https://courses.engr.illinois.edu/cs598csc/sp2011/Lectures/lecture_4.pdf "Vertex Cover via LP" ↩
6. https://en.wikipedia.org/wiki/Randomized_algorithm "Randomized Algorithm" ↩