Convex Optimization II

Lecture 3: Approximation Algorithms

Hamed Shah-Mansouri

Department of Electrical Engineering

Sharif University of Technology

1400-2

OUTLINE

- Preliminaries
- Complexity Theory Basics
- Approximation Algorithms
- Set Cover Problem Analysis
- Summary

PRELIMINARIES

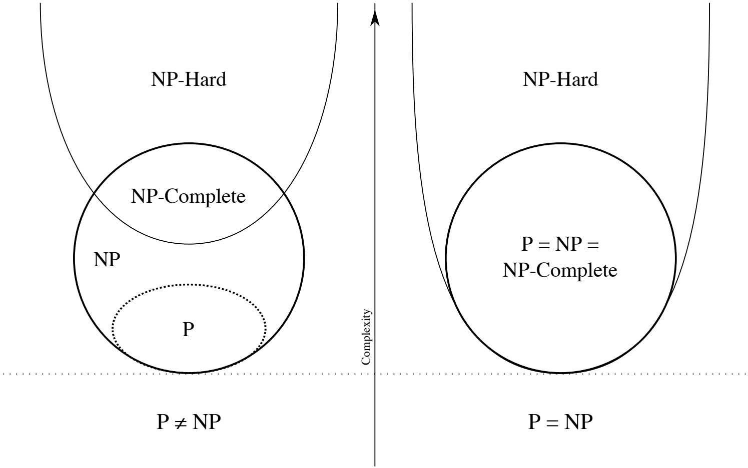
- Convex programming and linear programming (LP) problems can be solved in polynomial time to the size of input.
- Integer programming problems are often hard to solve. They can take exponential time to the size of input.
- NP-hard (non-deterministic polynomial-time hard), in computational complexity theory, is a class of problems that are, informally, at least as hard as the hardest problems in NP.
- A common mistake is thinking that the NP in "NP-hard" stands for "non-polynomial"

V. Vazirani, Approximation Algorithms, Springer-Verlag, 2001.

Computation Complexity Theory

- Class P: It contains all decision problems that can be solved in a polynomial amount of computation time, or polynomial time.
- Class NP: A complexity class that represents the set of all decision problems for which the solution can be verified in polynomial time.
- Class NP-Complete: A complexity class which represents the set of all problems X in NP for which it is possible to reduce any other NP problem Y to X in polynomial time.
  - If we had a polynomial time algorithm for an NP-complete, we could solve all problems in NP in polynomial time.
- NP-hard: The problems that are at least as hard as the NP-complete problems.
- P=NP? An open question

Computation Complexity Theory



Approximation Algorithms

Goal

- Find good, but approximate solutions for difficult problems (NP-hard).
- Be able to quantify the goodness of the given solution.

An approximation algorithm produces

- in polynomial time
- a feasible solution
- whose objective function value is close to the optimal value (within a guaranteed factor).

Set Cover Problem

Set Cover Problem

- Given a universe U of n elements
- Given a collection of subsets of U, $S = \{S_1, \dots, S_k\}$, and a cost c_j for each subset $j = 1, \dots, k$.
- Objective: find a minimum cost subcollection of S that covers all elements of U.

Set Cover Problem

Let $x_j \in \{0,1\}$ for each set $j = 1, \dots, k$ indicate whether set j is selected.

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{k} c_j x_j \\ \text{subject to} \quad & \sum_{j: e \in S_j} x_j \geq 1, \forall e \in U \\ & x_j \in \{0,1\}, \quad j = 1, \dots, k. \end{aligned}$$

We can model numerous classical problems as special cases of set cover:

- vertex cover, minimum cost shortest path, ...

The set cover problem is NP-hard.

Approximation Algorithms for Set Cover

Algorithms:

- Greedy algorithm
- Rounding techniques
  - LP rounding
  - Randomized rounding
- Dual fitting
- Primal-dual technique
- $\cdots$

Approximation ratio:

- The ratio between the cost obtained from the approximate algorithm and optimal cost (OPT).

Greedy Algorithm

Idea: 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.
- Define the cost-effectiveness of a set S to be the average cost at which it covers new elements:
$$\alpha = \frac{\text{cost}(S)}{|S - C|},$$
where $\text{cost}(S)$ is the cost associated with set S.
- Define the price of an element to be the average cost at which it is covered. Equivalently, when a set S is picked, we can think of its cost being distributed equally among the new elements covered, to set their prices.

Greedy Algorithm

Greedy set cover algorithm *

1. $C \leftarrow \emptyset$
2. While $C \neq U$ do

   Find the most cost-effective set in the current iteration, say S.

   Let $\alpha = \frac{\mathrm{cost}(S)}{|S-C|}$, i.e., the cost-effectiveness of S.

   Pick S, and for each $e \in S - C$, set $\mathrm{price}(e) = \alpha$.

   $C \leftarrow C \cup S$.
3. Output the picked sets.

*V. Vazirani, Approximation Algorithms, Springer-Verlag, 2001.

Analysis of Greedy Algorithm

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.

Lemma 1

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

Theorem 1

The greedy algorithm is an H_n factor approximation algorithm for the minimum set cover problem, where $H_n = 1 + \frac{1}{2} + \ldots + \frac{1}{n}$.

The approximation ratio of the greedy algorithm is of order $O(\log n)$.

Proof of Lemma 1

- In any iteration, the leftover elements can be covered at cost at most OPT.
- Therefore, among these sets, there must be one having cost-effectiveness of at most $\frac{\text{OPT}}{|\bar{C}|}$.
- In the iteration in which element e_k was covered, $\bar{C}$ contained at least $n-k+1$ elements.
- Since e_k was covered by the most cost-effective set in this iteration, it follows that
$$\text{price}(e_k) \leq \frac{\text{OPT}}{n-k+1}.$$

Proof of Theorem 1

- The total cost of the set cover picked is equal to

$$\sum_{k=1}^{n} \text{price}(e_k).$$

- By Lemma 1, this is at most

$$\left(1 + \frac{1}{2} + \ldots + \frac{1}{n}\right) \text{OPT}.$$

- Therefore,

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

which approaches $\log n$ when $n \to \infty$.

Rounding Techniques

Relaxing the integer variables of set cover problem

Integer Programming

$$\begin{array}{ll} \text{minimize} & \sum_{j=1}^{k} c_j x_j \\ \text{subject to} & \sum_{j: e \in S_j} x_j \geq 1, \forall e \in U \\ & x_j \in \{0,1\}, \quad j = 1, \ldots, k. \end{array}$$

Linear Programming

$$\begin{array}{ll} \text{minimize} & \sum_{j=1}^{k} c_j x_j \\ \text{subject to} & \sum_{j: e \in S_j} x_j \geq 1, \forall e \in U \\ & 0 \leq x_j \leq 1, \quad j = 1, \ldots, k. \end{array}$$

$$\mathrm{OPT}_{\mathrm{IP}} \geq \mathrm{OPT}_{\mathrm{LP}}.$$

Question: How we can interpret the result when some x_j are fractions?

LP-ROUNDING

Definitions:

- Define the frequency of an element to be the number of sets it belongs to.
- Denote the frequency of the most frequent element as f.

LP-rounding Algorithm

1. Relax the integer program into linear program.
2. Solve the linear program to obtain optimal solution $\bar{\boldsymbol{x}} = (x_1, \ldots, x_k)$.
3. Determine the maximum frequency (i.e., f).
4. Output deterministic rounding $\bar{\boldsymbol{x}'} = (x'_1, \ldots, x'_k)$, where

$$x'_j = \begin{cases} 1, & \text{if } x_j \geq \frac{1}{f} \\ 0, & \text{otherwise.} \end{cases}$$

Question: Is the obtained solution a feasible solution of the integer programming problem?

LP-ROUNDING FEASIBILITY

To show that $\bar{\boldsymbol{x}}'$ is a feasible solution of integer programming problem

- Examine constraints for all elements.
- Consider element e and suppose

$$x_1 + x_2 + \ldots + x_l \geq 1,$$

  is the corresponding constraint of linear program.
- By definition, we have $l \leq f$.
- To satisfy the above constraint, there must be at least one i, such that

$$x_i \geq \frac{1}{l} \geq \frac{1}{f}.$$

- In this way, x_i will be rounded to $x_i' = 1$, hence, the above constraint is satisfied for integer program.

Approximation ratio of LP-rounding

Theorem 2

LP-rounding algorithm for set cover problem is f-approximate.

Proof. We can easily check that $x'_i \leq f x_i$. So,

$$\sum_{j=1}^{k} c_j x'_j \leq \sum_{j=1}^{k} c_j f x_j = f \sum_{j=1}^{k} c_j x_j = f \mathrm{OPT}_{\mathrm{LP}} \leq f \mathrm{OPT}_{\mathrm{IP}}.$$

Randomized rounding

Idea: View the fractions as probabilities, flip coins with these biases, and round accordingly.

Randomized rounding Algorithm

1. Relax the integer program into linear program.
2. Solve the linear program to obtain optimal solution $\bar{\boldsymbol{x}} = (x_1, \ldots, x_k)$.
3. Set $x'_i = 1$ with probability x_i.
4. Repeat Step 3 for $O(\log n)$ times.

Question: Is the obtained solution a feasible solution of the integer programming problem?

Approximation ratio of Randomized Rounding

Theorem 3

Using randomized rounding algorithm, we achieve a set cover with high probability.

Theorem 4

Randomized rounding algorithm for set cover problem is $O(\log n)$-approximate.

PROOF OF THEOREM 4

Let $\mathcal{C}$ be the collection of sets picked in Phase 3. The expected cost of $\mathcal{C}$ is

$$\mathbb{E}[\text{cost}(\mathcal{C})] = \sum_{i=1}^{k} \mathbf{Pr}[\text{set } i \text{ is pciked}]c_k = \sum_{i=1}^{k} x_k c_k = \text{OPT}_{\text{LP}}.$$

Thus, through Phase 4, the expected total cost of the cover becomes

$$O(\log n)\text{OPT}_{\text{LP}}.$$

Proof of Theorem 3

Now, let us find the probability that an element i is covered by $\mathcal{C}$.

$$\mathbf{Pr}[i \text{ is covered by } \mathcal{C}] = 1 - (1 - x_1) \cdots (1 - x_k) \geq 1 - \left(1 - \frac{1}{k}\right)^k > 1 - \frac{1}{\mathrm{e}}$$

Again, through Phase 4, we independently pick $c \log n$ covers, say $\mathcal{C}'$, where c is a constant. Thus,

$$\mathbf{Pr}[i \text{ is not covered by } \mathcal{C}'] \leq \left(\frac{1}{e}\right)^{c \log n}$$

Summing over all elements

$$\mathbf{Pr}[\mathcal{C}' \text{ is not valid set cover }] \leq n \left(\frac{1}{e}\right)^{c \log n}.$$

Summary

- Mixed integer programming problems are often hard to solve.
- Set cover problem is NP-hard.
- Greedy algorithm for set cover problem can achieve approximation ratio of H_n.
- LP-rounding for set cover problem can achieve approximation ratio of f.
- Randomized rounding for set cover problem can asymptotically achieve approximation ratio of $O(\log n)$.
- Reading: Sections 2, 12, and 14 of Approximation Algorithms by V. Vazirani.