1 Approximation Algorithms

Karp (see Figure 1) introduced NP-complete problems. Unless P = NP, the optimization versions of these problems admit no algorithms that simultaneously (1) find optimal solution (2) in polynomial time (3) for all instances.

Definition 1. Let $A$ be an algorithm for an optimization problem. Let $A(I)$ and $\text{Opt}(I)$ be the solution returned by $A$ and the optimal solution for the instance $I$ respectively.

Then Algorithm $A$ is $\alpha$-approximation if

- $\text{Opt}(I) \leq A(I) \leq \alpha \cdot \text{Opt}(I)$ (for minimization problem)
- $\alpha \cdot \text{Opt}(I) \leq A(I) \leq \text{Opt}(I)$ (for maximization problem).

Here $\alpha$ is referred to as the approximation ratio, approximation factor or the performance guarantee of $A$.

Different problems have different approximability. Some problem such as TSP are inapproximable and some other problems such as Bin Packing admit PTAS.

Definition 2. A polynomial time approximation scheme (PTAS) is a family of algorithms $\{A_\epsilon\}$, where for each $\epsilon > 0$ there is an algorithm $A_\epsilon$ that is $(1 + \epsilon)$-approximation algorithm (for minimization problems) or $(1 - \epsilon)$-approximation algorithm (for maximization problems).

So, for any small constant $\epsilon$ (say $0.0001$), we can get a 1.0001 approximation for any problem that admit PTAS.
Why study approximation algorithms?

- We need fast solution for practical problems.
- Provides mathematical rigor to study and analyze heuristics.
- Gives a metric for difficulty of different discrete optimization problems.
- It's cool!

We will study approximation algorithms for three such NPC problems:

- Set Cover
- *Metric* Traveling Salesman Problem
- Max Cut.

2 Set Cover

In the *Set Cover problem*, we are given a ground set of $n$ elements $E = \{e_1, e_2, \ldots, e_n\}$ and a collection of $m$ subsets of $E$: $S := \{S_1, S_2, \cdots, S_m\}$ and a nonnegative weight function $\text{cost} : S \to \mathbb{Q}^+$. We will sometimes use $w_j = \text{cost}(S_j)$. The goal is to find a minimum weight collection of subsets that covers all elements in $E$. Formally we want to find a set cover $C$ that minimizes $\Sigma_{S_j \in C} w_j$ subject to $\bigcup_{S_j \in C} S_j = E$. If $w_j = 1$ for all $j$, then the problem is called the *unweighted set cover problem*.

- Set Cover is a problem *whose study has led to the development of fundamental techniques for the entire field of approximation algorithms* [2].
- It is a generalization of many other important NPC problems such as *vertex cover* and *edge cover*.
- It is used in the development of antivirus products, VLSI design and many other practical problems.

3 A Greedy Algorithm for Set Cover

3.1 Algorithm

1. Initialize $C \leftarrow \phi$.

2. While $C$ does not cover all elements in $E$ do

   (a) Define cost-effectiveness of each set $S \in S$ as $\alpha_S = \frac{\text{cost}(S)}{|S \setminus C|}$
   (b) Find $S$, the most cost-effective set in the current iteration.
   (c) Pick $S$ and for all newly covered elements $e \in S \setminus C$, set $\text{price}(e) = \alpha_S$.
   (d) $C \leftarrow C \cup S$.

3. Output $C$. 


3.2 Analysis

- Returns a valid set cover in polynomial time.
- In any iteration, leftover sets of the optimal solution can cover the remaining elements \( E \setminus C \) at a cost of \( \text{Opt} \).
- Among these sets one must have cost-effectiveness \( \leq \frac{\text{Opt}}{|E \setminus C|} \).
- W.l.o.g. assume that the elements are numbered in the order in which they were covered by the algorithm, resolving ties arbitrarily. Let \( e_1, e_2, ..., e_n \) be this numbering in the order they are covered by the greedy algorithm.
- Assume element \( e_k \) was covered by the most cost-effective set at some iteration \( i \leq k \). At most \( (k-1) \) items were covered before the iteration \( i \). Thus at least \( n - (k-1) \) elements were not covered before the iteration \( i \) and \( |E \setminus C| \geq (n - k + 1) \).
- \( \text{price}(e_k) \leq \frac{\text{Opt}}{|E \setminus C|} \leq \frac{\text{Opt}}{n-k+1} = \frac{\text{Opt}}{p} \) where \( p = (n-k+1) \).
- \( \text{price} \) is just distribution of set weights into the items. So the total cost of set cover \( \sum_{S_i \in C} \text{cost}(S_i) = \sum_{e_i \in E} \text{price}(e_i) \).
- Now, \( \sum_{e_k \in E} \text{price}(e_k) \leq \sum_{k=1}^{n} \frac{\text{Opt}}{n-k+1} \leq \sum_{p=1}^{n} \frac{\text{Opt}}{p} \leq H_n \cdot \text{Opt} \).
- Thus the greedy algorithm has \( H_n \) or \( O(\log n) \) approximation ratio where \( H_n \) is the \( n \)'th Harmonic number.
- Note: Finding a good lower bound on \( \text{Opt} \) is a basic starting point in the design of an approximation algorithm for a minimization problem.

3.3 Tight Example for Analysis:

- See Figure 3.3 for a tight example for the greedy algorithm for the set cover.
- Optimal solution has only one set of cost \( (1 + \epsilon) \) where \( \epsilon(<< 1) \) is a very small constant close to 0.
- The greedy algorithm will return \( n \) singleton sets with total cost = \( \frac{1}{n} + \frac{1}{(n-1)} + \ldots + 1 = H_n \).
- So, approximation ratio for this example is \( H_n/(1 + \epsilon) \approx H_n \) as we can take \( \epsilon \) to be arbitrarily small.
- Thus the analysis gave a upper bound of \( H_n \) and this example gave a lower bound of \( H_n \) for the greedy algorithm. As the upper and lower bound matches, we call it a tight example and the analysis is tight.

![Figure 2: Tight example for Greedy Algorithms for Set Cover](image)
3.4 Hardness:

- Is there any other polynomial time algorithm that achieves $(1 - o(1)) \ln n$-approximation assuming $P \neq NP$?

- No! [Feige 1998]. The proof is quite complex and use probabilistic checkable proof systems (PCPs).

4 Resources:

I am following chapter 1 of [1] for the lectures. The book is freely available online: [http://www.designofapproxalgs.com/](http://www.designofapproxalgs.com/). You can also see chapter 2 (Set Cover) from [2].

References
