Introduction to Algorithms

Topic 9-2 : Approximation Basics

Xiang-Yang Li and Haisheng Tan

School of Computer Science and Technology
University of Science and Technology of China (USTC)

Fall Semester 2023

Outline

History of Approximation

1966	**Graham:** First analyzed algorithms by approximation ratio
1971	**Cook:** Gave the concepts of NP-Completeness
1972	**Karp:** Introduced plenty NP-Hard combinatorial optimization problems
1970's	Approximation became a popular research area
1979	**Garey & Johnson:** Computers and Intractability: A guide to the Theory of NP-Completeness

NP Optimization Problem

An NP Optimization Problem P is a four tuple $(I, sol, m, goal)$ s.t.

- I is the set of the instances of P and is recognizable in polynomial time
- Given an instance x of I, $sol(x)$ is the set of short feasible solutions of x and $\forall x$ and $\forall y$ such that $|y| \leq p(|x|)$, it is decidable in polynomial time whether $y \in sol(x)$.
- Given an instance x and a feasible solution y of x, $m(x, y)$ is a polynomial time computable measure function providing a positive integer which is the value of y.
- $goal \in \{max, min\}$ denotes maximization or minimization.

An Example of NP Optimization Problem

Example: Minimum Vertex Cover

Given a graph $G = (V, E)$, the Minimum Vertex Cover problem (MVC) is to find a vertex cover of minimum size, that is, a minimum node subset $U \subseteq V$ such that, for each edge $(v_i, v_j) \in E$, either $v_i \in U$ or $v_j \in U$.

An Example of NP Optimization Problem

Example: Minimum Vertex Cover

Given a graph $G = (V, E)$, the Minimum Vertex Cover problem (MVC) is to find a vertex cover of minimum size, that is, a minimum node. subset $U \subseteq V$ such that, for each edge $(v_i, v_j) \in E$, either $v_i \in U$ or $v_j \in U$.

Justification $\rightarrow$ MVC is an NP Optimization Problem

- $I = \{G = (V, E) | Gisagraph\}$; poly-time decidable
- $sol(G) = \{U \subseteq V | \forall (v_i, v_j) \in E[v_i \in U \vee v_j \in U]\}$; short feasible solution set and poly-time decidable
- $m(G, U) = |U|$; poly-time computable function
- $goal = min$.

NPO Class

Definition: (NPO Class)
The class NPO is the set of all NP optimization problems.

Definition: (Goal of NPO Problem)
The goal of an NPO problem with respect to an instance x is to find an optimum solution, that is, a feasible solution y such that $m(x,y) = goal\{m(x,y') : y' \in sol(x)\}$.

What is Approximation Algorithm

Definition: Approximation Algorithm

Given an NP optimization problem $P = (I, sol, m, goal)$, an algorithm A is an approximation algorithm for P if, for any given instance $x \in I$, it returns an approximate solution, that is a feasible solution $A(x) \in sol(x)$ with guaranteed quality.

What is Approximation Algorithm

Definition: Approximation Algorithm

Given an NP optimization problem $P = (I, sol, m, goal)$, an algorithm A is an approximation algorithm for P if, for any given instance $x \in I$, it returns an approximate solution, that is a feasible solution $A(x) \in sol(x)$ with guaranteed quality.

Note:

- Guaranteed quality is the difference between approximation and heuristics.
- Approximation for PO, NPO and NP-hard Optimization.
- Decision, Optimization, and Constructive Problems.

$r-$Approximation

Definition: Approximation Ratio

Let P be an NPO problem. Given an instance x and a feasible solution y of x, we define the performance ratio of y with respect to x, we define the performance ratio of y with respect to x as

$$R(x,y) = \max\{\frac{m(x,y)}{opt(x)}, \frac{opt(x)}{m(x,y)}\}$$

Definition: $r-$Approximation

Given an optimization problem P and an approximation algorithm A for P, A is said to be an $r-approximation$ for P if, given any input instance x of P, the performance ratio of the approximate solution $A(x)$ is bounded by r, say, $R(x,A(x)) \leq r$.

APX Class

Definition: F-APX

Given a class of functions F, an NPO problem P belongs to the class F-APX if an $r-$approximation polynomial time algorithm A for P exists, for some function $r \in F$.

Example:

- F is constant functions $\rightarrow P \in APX$.
- F is $O(\log n)$ functions $\rightarrow P \in \log -APX$.
- F is $O(n^k)$ functions (polynomials) $\rightarrow p \in poly - APX$.
- F is $O(2^{n^k})$ functions $\rightarrow P \in exp - APX$.

Special Case

Definition: Polynomial Time Approximation Scheme $\rightarrow$ PTAS

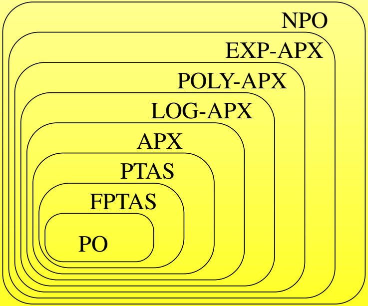
An NPO problem P belongs to the class PTAS if an algorithm A exists such that, for any rational value $\varepsilon > 0$, when applied A to input (x, ε), it returns an $(1+\varepsilon)-$approximate solution of x in time polynomial in $|x|$.

Definition: Fully PTAS $\rightarrow$ FPTAS

An NPO problem P belongs to the class FPTAS if an algorithm A exists such that, for any rational value $\varepsilon > 0$, when applied A to input (x, ε), it returns an $(1+\varepsilon)-$approximate solution of x in time polynomial both in $|x|$ and in $\frac{1}{\varepsilon}$.

Approximation Class Inclusion

If $P \neq NP$, then $FPTAS \subseteq PTAS \subseteq APX \subseteq Log-APX \subseteq Poly-APX \subseteq Exp-APX \subseteq NPO$



- Constant-Factor Approximation (APX)
  - Reduce App. Ratio
  - Reduce Time Complexity
- PTAS $((1+\varepsilon)-Appx)$
  - Test Existence
  - Reduce Time Complexity

Outline

1. Approximation Basics
   - History
   - NP Optimization
   - Definition of Approximation

2. The vertex-cover problem

3. The set cover problem

4. Knapsack

Vertex Cover Problem

Problem

Instance: Given an undirected graph $G = (V, E)$

Solution: A subset $V' \subseteq V$ that if $(u, v) \in E$, then $u \in V'$ or $v \in V'$ (or both)

Measure: The size which is the number of vertices in it.

Approximate Vertex-Cover

The following approximation algorithm takes as input an undirected graph G and returns a vertex cover whose size is guaranteed to be no more than twice the size of an optimal vertex cover.

APPROX-VERTEX-COVER(G)

$C = \emptyset$
$E' = G.E$
while $E' \neq \emptyset$ **do**
    Let(u, v) be an arbitrary edge of E'
    $C = C \cup \{u, v\}$
    remove from E' every edge incident on either u or v
return C

Approximate Vertex-Cover

The following approximation algorithm takes as input an undirected graph G and returns a vertex cover whose size is guaranteed to be no more than twice the size of an optimal vertex cover.

APPROX-VERTEX-COVER(G)

1: $C = \emptyset$
2: $E' = G.E$
3: **while** $E' \neq \emptyset$ **do**
4:     Let(u, v) be an arbitrary edge of E'
5:     $C = C \cup \{u, v\}$
6:     remove from E' every edge incident on either u or v
7: **return** C

Approximation Ratio?

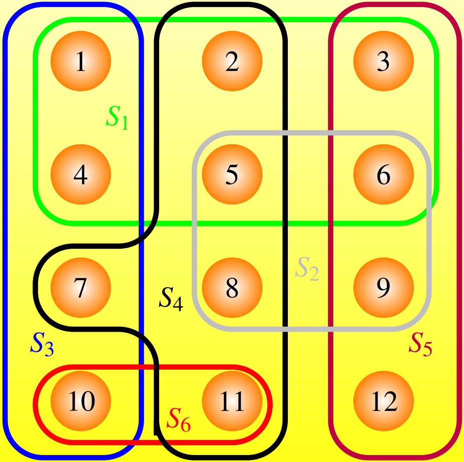
Outline

Set Cover Problem

Problem

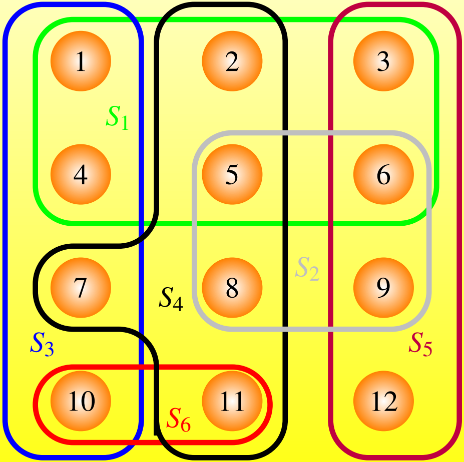
Instance: Given a finite set X and a family $\mathscr{F}$ of subsets of X, such that every element of X belongs to at least one subset in $\mathscr{F}$: $X = \bigcup_{S \in \mathscr{F}} S$.
Problem: Find a minimum-size subset $\mathscr{L} \subseteq \mathscr{F}$ whose members cover all of X: $X = \bigcup_{S \in \mathscr{L}} S$.

An Example



$U = \{1, 2, \ldots, 12\}$
$\mathbf{S} = \{S_1, S_2, \ldots, S_6\}$
$S_1 = \{1, 2, 3, 4, 5, 6\}$
$S_2 = \{5, 6, 8, 9\}$
$S_3 = \{1, 4, 7, 10\}$
$S_4 = \{2, 5, 7, 8, 11\}$
$S_5 = \{3, 6, 9, 12\}$
$S_6 = \{10, 11\}$

An Example



$U = \{1, 2, ..., 12\}$
$\mathbf{S} = \{S_1, S_2, ..., S_6\}$
$S_1 = \{1, 2, 3, 4, 5, 6\}$
$S_2 = \{5, 6, 8, 9\}$
$S_3 = \{1, 4, 7, 10\}$
$S_4 = \{2, 5, 7, 8, 11\}$
$S_5 = \{3, 6, 9, 12\}$
$S_6 = \{10, 11\}$

OptimalSolution :
$\mathbf{S'} = \{S_3, S_4, S_5\}$

Greedy Algorithm

GREEDY-SET-COVER(X,$\mathscr{F}$)

$U = X$

$\mathscr{L} \leftarrow \emptyset$

while $U \neq \emptyset$ **do**

    select an $S \in \mathscr{F}$ that maximizes $|S \cap U|$.

    $U = U - S$.

    $\mathscr{L} = \mathscr{L} \cup \{S\}$

return $\mathscr{L}$.

Analysis

Theorem 1

Greedy-Set-Cover is a polynomial-time $\rho(n)-$approximation algorithm, where $\rho(n) = H(\max\{|S| : S \in \mathscr{F}\})$. (We denote the dth harmonic number $H_d = \sum_{i=1}^{d} 1/i$ by $H(d)$.)

Analysis

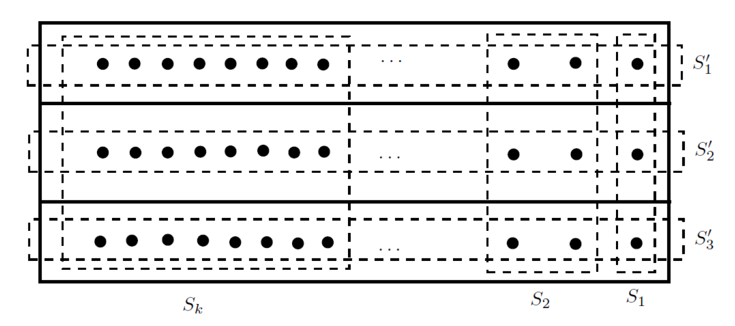
Theorem 1

Greedy-Set-Cover is a polynomial-time $\rho(n)-$approximation algorithm, where $\rho(n) = H(\max\{|S| : S \in \mathscr{F}\})$. (We denote the dth harmonic number $H_d = \sum_{i=1}^{d} 1/i$ by $H(d)$.)

Corollary 2

Greedy-Set-Cover is a polynomial-time $(\ln|X| + 1)$-approximation algorithm.

Greedy Performs Badly



Outline

Knapsack

Problem

Instance: Given a set of n items, each with profit p_i and size s_i, and a knapsack with size bound $B (B > s_i)$.
Solution: A subset of items $S \subset [n]$ that subject to the constraint $\sum_{i \in S} s_i \leq B$.
Measure: Total profit of the chosen subset, $\sum_{i \in S} p_i$.

Greedy Algorithm

Greedy Algorithm?

1. Sort items in non-increasing order of $\frac{P_i}{S_i}$
2. Greedily pick items in above order.

Greedy Algorithm

Greedy Algorithm?

1. Sort items in non-increasing order of $\frac{P_i}{S_i}$
2. Greedily pick items in above order.

Consider the following input:

- An item with size 1 and profit 2
- An item with size B and profit B

Greedy Algorithm

Greedy Algorithm?

1. Sort items in non-increasing order of $\frac{P_i}{S_i}$
2. Greedily pick items in above order.

Consider the following input:

- An item with size 1 and profit 2
- An item with size B and profit B

Our greedy algorithm will only pick the small item, making this a pretty bad approximation algorithm

Greedy Algorithm

Greedy Algorithm Redux

1. Sort items in non-increasing order of $\frac{P_i}{S_i}$
2. Greedily add items until we hit an item a_i that is too big. $(\sum_{k=1}^{i} s_i > B)$
3. Pick the better of $\{a_1, a_2, ..., a_{i-1}\}$ and a_i.

Greedy Algorithm

Greedy Algorithm Redux

1. Sort items in non-increasing order of $\frac{P_i}{S_i}$
2. Greedily add items until we hit an item a_i that is too big. $(\sum_{k=1}^{i} s_i > B)$
3. Pick the better of $\{a_1, a_2, ..., a_{i-1}\}$ and a_i.

Greedy Algorithm Redux is a $2-$*approximation* for the knapsack problem.
Actually, we can achieve $(1+\varepsilon)$-approximation for any $\varepsilon > 0$ based on Dynamic Programming.