Approximation Algorithms

- Approximation Algorithms for Optimization Problems: Types
- Absolute Approximation Algorithms
- Inapproximability by Absolute Approximate Algorithms
- Relative Approximation Algorithm
- InApproximability by Relative Approximate Algorithms
- Polynomial Time Approximation Schemes
- Fully Polynomial Time Approximation Schemes

Imdad ullah Khan

Quality of Approximation: Types

Fully Polynomial Time Approximation Scheme (FPTAS)

Given an optimization problem P with value function f on solution space

A family of algorithms $A(\epsilon)$ is called a **fully polynomial time approximation scheme** if for a given ϵ, on any instance I, $A(\epsilon)$ achieves an approximation error ϵ and runtime of A is polynomial in $|I| = n$ and $1/\epsilon$

- For a minimization problem this means $f(A(I)) \leq (1+\epsilon) \cdot f(\text{OPT}(I))$
- For a maximization problem this means $f(A(I)) \geq (1-\epsilon) \cdot f(\text{OPT}(I))$

Runtime of A cannot be exponential in $1/\epsilon$ ▷ e.g. $O(1/\epsilon^2 n^3)$

Constant factor decrease in ϵ increases runtime by a constant factor

Knapsack Problem

Input:

- Items: $U = \{a_1, \ldots, a_n\}$ ▷ Fixed order
- Weights: $w : U \to \mathbb{Z}^+$ ▷ $(w_1, \ldots, w_n)$
- Values: $v : U \to \mathbb{R}^+$ ▷ $(v_1, \ldots, v_n)$
- Capacity: $C \in \mathbb{R}^+$

Output:

- A subset $S \subset U$
- Capacity constraint:

$$\sum_{a_i \in S} w_i \leq C$$

- Objective: Maximize

$$\sum_{a_i \in S} v_i$$

FPTAS for KNAPSACK

Lemma 1: If for some $0 < \epsilon < 1/2$, all $w_i \leq \epsilon C$, then MODIFIED-GREEDY-BY-RATIO is $(1-\epsilon)$-approximate

1. Scale down all weights to meet above requirement
2. Run $(1-\epsilon)$-approximate MODIFIED-GREEDY-BY-RATIO
3. Scale up resulting solution

▷ Scaling up may violate capacity constraint

Develop scaling friendly solution using dynamic programming

Scaling w.r.t. desired ϵ,

we get a $(1-\epsilon)$-approximate solution polynomial in both n and $\frac{1}{\epsilon}$ (FPTAS)

Recall that for the items subset $\{a_1, \cdots, a_i\}$ and capacity c

$$\text{OPT}(i,c) = \max \begin{cases} 0 & \text{if } c \leq 0 \\ 0 & \text{if } i = 0 \\ \text{OPT}(i-1, c-w_i) + v_i & \\ \text{OPT}(i-1, c) & \end{cases}$$

Runtime is $\mathcal{O}(nC)$ ▷ not polynomial unless C is in unary

For above solution, the question is:

What is the maximum value achievable if capacity is c?

Now, the question is transformed to:

What is the minimum weight needed to gain a value of p?

Note: all values are integers

Scaling Friendly Dynamic Programming

Let $\widehat{\text{OPT}}(i, v)$ be the min capacity needed to get value v from items $\{a_1, , \cdots, a_i\}$

Let $P = \sum_{i}^{n} v_i$ ▷ maximum achievable value

We need $\widehat{\text{OPT}}(i, v)$ for $0 \leq i \leq n$ and $0 \leq v \leq P$ ▷ $n \cdot P$ subproblems

If v_m is the max value of an item, then $P \leq nv_m$ ▷ $O(n \cdot nv_m)$ subproblems

$$\widehat{\text{OPT}}(i, v) = \begin{cases} 0 & \text{if } v = 0 \\ \infty & \text{if } i = 0 \text{ and } v > 0 \\ \widehat{\text{OPT}}(i-1, v) & \text{if } i \geq 1 \text{ and } 1 \leq v < v_i \\ \min\left\{\widehat{\text{OPT}}(i-1, v), \widehat{\text{OPT}}(i-1, p - v_i) + w_i\right\} & \text{if } i \geq 1 \text{ and } v \geq v_i \end{cases}$$

Solution to an instance $[U, w, v, C]$ is the maximum v s.t. $\widehat{\text{OPT}}(n, v) \leq C$

$\widehat{\text{OPT}}(n, P)$ can be computed in $O(n^2 v_m)$ (pseudo-polynomial) ▷ bottom-up DP

If v_m is polynomial in n (e.g. n^k), then runtime is polynomial

Solution to an instance $[U, w, v, C]$ is the maximum v s.t. $\widehat{\text{OPT}}(n, v) \leq C$

$\widehat{\text{OPT}}(n, P)$ can be computed in $O(n^2 v_m)$ (pseudo-polynomial) ▷ bottom-up DP

▷ If v_m is polynomial in n (e.g. n^k), then runtime is polynomial

If item values are not polynomial. To get an approximate solution

1. Scale down values so they are not too large and round to integers

▷ Error introduced as exact values are unknown (not used)

We bound the error due to scaling to $\leq \epsilon \cdot \text{OPT}$ to get a $(1-\epsilon)$-approximation

Let $b = \frac{\epsilon}{n}\text{OPT}$ and let $v_i' = \left\lceil \frac{v_i}{b} \right\rceil$ i.e. v_i' is the smallest integer s.t. $v_i \leq v_i' \cdot b$

▷ Note: If $v_i \leq v_j$, then $v_i' < v_j'$ for $1 \leq i, j \leq n$

$$\text{OPT} \geq v_m \implies v_m' = \left\lceil \frac{v_m}{b} \right\rceil = \left\lceil \frac{v_m}{\epsilon/n \cdot \text{OPT}} \right\rceil \leq \left\lceil \frac{n \cdot v_m}{\epsilon \cdot v_m} \right\rceil = \left\lceil \frac{n}{\epsilon} \right\rceil$$

Run scaling-friendly dynamic programming with values v_i'

Get opt solution S' w.r.t v_i' in $O(n^2 v_m) = O(n^3 \cdot \frac{1}{\epsilon})$ time ▷ poly$(n, 1/\epsilon)$

▷ $w(S') < C$ as capacity and weights were unchanged

What is the error?

Let S be the optimal solution using v_i, i.e. $\text{OPT} = \sum_{i \in S} v_i$

Let $v'(S) = \sum_{i \in S} v_i'$ and $v'(S') = \sum_{i \in S'} v_i'$

1. $v'(S') \geq v'(S)$ ▷ Since S' is optimal w.r.t. v_i'
2. $v_i/b \leq v_i' \leq v_i/b + 1$ ▷ By definition,

Use above observations to upper bound on OPT in terms of $v(S')$ and ϵ

FPTAS for KNAPSACK

$$\text{OPT} = \sum_{i \in S} v_i \leq \sum_{i \in S} b \cdot v_i' \leq b \cdot \sum_{i \in S} v_i' \leq b \cdot v'(S) \leq b \cdot v'(S')$$

$$\leq b \cdot \sum_{i \in S'} v_i' \leq b \cdot \sum_{i \in S'} v_i/b + 1 = b \sum_{i \in S'} \frac{v_i + b}{b}$$

$$= \sum_{i \in S'} v_i + b \cdot |S'| \leq v(S') + n \cdot b = v(S') + \epsilon \cdot \text{OPT}$$

$$v(S') \geq (1 - \epsilon) \cdot \text{OPT} \implies S' \text{ is } (1 - \epsilon)\text{-approximate}$$

- The value of OPT (used in b) is unknown
- Use lower bound $\text{OPT} \geq v_m$ for $b = \frac{\epsilon}{n} \cdot v_m$
- Above analysis results in $\text{OPT} \leq v(S') + \epsilon \cdot v_m \leq v(S') + \epsilon \cdot \text{OPT}$