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

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 minimization problems this means $f\left(A(I)\right) \leq (1+\epsilon) \cdot f\left(\mathrm{OPT}(I)\right)$
- For maximization problems this means $f\left(A(I)\right) \geq (1-\epsilon) \cdot f\left(\mathrm{OPT}(I)\right)$
- 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

Input:

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

Output:

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

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

- Objective: Maximize

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

- Recall that if for some $0 < \epsilon < 1/2$ all $w_i \leq \epsilon C$, then we have a $(1-\epsilon)$-approximation
- One possible way:
  - Scale down all weights to meet above requirement
  - Run $(1-\epsilon)$-approximate MODIFIED-GREEDY-BY-RATIO
  - Scale up resulting solution
- Scaling up may violate capacity constraint
- Develop scaling friendly solution using dynamic programming
- Scaling w.r.t. desired ϵ, we can 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

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

- Optimal solution found in $\mathcal{O}(nC)$ time
- Runtime is not polynomial unless C is represented in unary system
- 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 that all values are integers

Scaling Friendly Dynamic Programming

- Let min capacity needed to get value v from items $\{a_1, a_2, \cdots, a_i\}$
- Maximum achievable value is $P = \sum_i^n v_i$, for which $\overline{\text{OPT}}(i, v)$ must be computed $\forall\ 0 \leq i \leq n$ and $0 \leq v \leq P$

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

- Solution to the instance I is the maxmimum v s.t. $\overline{\text{OPT}}(i, v) \leq C$
- Let v_m be the maximum value of any item, then $P \leq nv_m$
- Total number of sub-problems are at most $O(n \cdot nv_m)$
- Solve recurrence using bottom-up iterative dynamic programming procedure that computes $\text{OPT}(n, C)$ in $\mathcal{O}(n^2 v_m)$ (pseudo-polynomial)

- If v_m is polynomial in n (e.g. n^k), then dynamic programming solution can be used as it is
- If item values are larger (not polynomial), then above solution i not polynomial (can not be used directly)
- To get an approximate solution
  - scale down values so they are not too large
  - round values to integers
- Error introduced as exact values are unknown (not used)
- Bound error to $\leq \epsilon \cdot \text{OPT}$ to get a $(1-\epsilon)$-approximation
- Let $b = \epsilon/n \cdot \text{OPT}$
- Let $v_i' = \lceil v_i/b \rceil$ 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 $v_i' < v_j'$ $\forall\ 1 \leq i,j \leq n$ and since $\text{OPT} \geq v_m$

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

- Run scaling-friendly dynamic programming with values v_i'
- Get optimal solution S' w.r.t v_i' in $\mathcal{O}(n^2 v_m) = \mathcal{O}(n^3 \cdot \frac{1}{\epsilon})$ time
- Runtime is polynomial in n and $\frac{1}{\epsilon}$
- What is the error?
- Let S be the optimal solution using v_i, i.e. $\text{OPT} = \sum_{i \in S} v_i$
- $w(S') < C$ as capacity and weights were unchanged
- Let $v'(S) = \sum_{i \in S} v_i'$ and $v'(S') = \sum_{i \in S'} v_i'$.
- Then $v'(S') \geq v'(S)$ since S' is optimal w.r.t. v_i'
- By definition, $\frac{v_i}{b} \leq v_i' \leq \frac{v_i}{b} + 1$
- Use above observations to compute an upper bound on OPT in terms of $v(S')$ and ϵ

$$
\begin{aligned}
\mathrm{OPT} &= \sum_{i \in S} v(i) \\
&\le \sum_{i \in S} b \cdot v'_i \\
&\le b \cdot \sum_{i \in S} v'_i \\
&\le b \cdot v'(S) \\
&\le b \cdot v'(S') \\
&\le b \cdot \sum_{i \in S'} v'_i \\
&\le b \cdot \sum_{i \in S'} (\frac{v_i}{b} + 1)
\end{aligned}
$$

$$
\begin{aligned}
&= b \cdot \sum_{i \in S'} \frac{v_i + b}{b} \\
&= b \cdot \frac{1}{b} \sum_{i \in S'} (v_i + b) \\
&= \sum_{i \in S'} v_i + b \cdot |S'| \\
&\leq v(S') + n \cdot b \\
&= v(S') + \epsilon \cdot \text{OPT}
\end{aligned}
$$

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

- 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 $\text{OPT} \leq v(S') + \epsilon \cdot v_m \leq v(S') + \epsilon \cdot \text{OPT}$