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

Approximation Factor/Ratio

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

The approximation ratio or approximation factor of an algorithm A is defined as the ratio *'between'* value of output of A and value of $\textsc{opt}$

- For minimization problem it is $f\big(A(I)\big) / f\big(\textsc{opt}(I)\big)$
- For maximization problem it is $f\big(\textsc{opt}(I)\big) / f\big(A(I)\big)$

▷ Note: approximation factor is always bigger than 1

Generally, approximation factor is defined as $max \left\{ \frac{f\big(A(I)\big)}{f\big(\textsc{opt}(I)\big)}, \frac{f\big(\textsc{opt}(I)\big)}{f\big(A(I)\big)} \right\}$

Quality of Approximation: Types

Approximation Error

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

The approximation error of A is its approximation factor minus 1

- For a minimization problem it is

  $$f\big(A(I)\big)/f\big(\textsc{opt}(I)\big) - 1 \;=\; f\big(A(I)\big) - f\big(\textsc{opt}(I)\big)/f\big(\textsc{opt}(I)\big)$$

- For a maximization problem it is

  $$f\big(\textsc{opt}(I)\big)/f\big(A(I)\big) - 1 \;=\; f\big(\textsc{opt}(I)\big) - f\big(A(I)\big)/f\big(A(I)\big)$$

▷ Useful when approximation ratio is close to 1

Also called relative approximation error

Quality of Approximation: Types

Polynomial Time Approximation Scheme (PTAS)

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

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

- 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 could be exponential in $1/\epsilon$ $\triangleright$ e.g. $O(n^{1/\epsilon})$

Knapsack Problem

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

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

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: $C \in \mathbb{R}^+$

ID	weight	value
1	1	1
2	2	6
3	5	18
4	6	22
5	7	28
6	98	99

$C = 11$

Output:

- A subset $S \subset U$
- Capacity constraint: $\sum_{a_i \in S} w_i \leq C$
- Objective: Maximize $\sum_{a_i \in S} v_i$

- $\{1, 2\}$ weight 3, value 7
- $\{1, 2, 4\}$ weight 9, value 29
- $\{3, 4\}$ weight 11, value 40
- $\{4, 5\}$ weight 13, value 50

Knapsack Problem: Greedy Algorithms

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 $\sum_{a_i \in S} v_i$

Greedy by Value

- Select the most profitable item
- Check if its fits remaining capacity
- Repeat

ID	weight	value
1	51	51
2	50	50
3	50	50

$C = 100$

$\{1\}$ weight 51, value 51

Optimal $\{2, 3\}$ weight 100, value 100

Knapsack Problem: Greedy Algorithms

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 $\sum_{a_i \in S} v_i$

Greedy by weight

- Select the least weighted item
- Check if its fits remaining capacity
- Repeat

ID	weight	value
1	1	1
2	50	50
3	50	50

$C = 100$

$\{1, 2\}$ weight 51, value 51

Optimal $\{2, 3\}$ weight 100, value 100

Knapsack Problem: Greedy Algorithms

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 $\sum_{a_i \in S} v_i$

GREEDY-BY-RATIO

- Select item with highest v_i/w_i
- Check if its fits capacity
- Repeat

ID	weight	value	ratio
1	1	1	1
2	2	6	3
3	5	18	3.6
4	6	22	3.67
5	7	28	4
6	98	99	1.01

$C = 11$

$\{5, 2, 1\}$ weight 10, value 35

Optimal $\{3, 4\}$ weight 11, value 40

Knapsack Problem: GREEDY-BY-RATIO

The GREEDY-BY-RATIO algorithm is suboptimal but worth exploring

Algorithm GREEDY-BY-RATIO

if $\sum_{i=1}^{n} w_i \leq C$ **then** ▷ If all items fit in the sack, then take all
  return U
SORT items by v_i/w_i ▷ assume $v_1/w_1 \geq v_2/w_2 \geq \ldots \geq v_n/w_n$
$weight \leftarrow 0$ ▷ total weight collected so far
$value \leftarrow 0$ ▷ total value collected so far
$S \leftarrow \emptyset$ ▷ Initially the knapsack is empty
for $i = 1 \to n$ **do**
  if $weight + w_i < C$ **then**
    $S \leftarrow S \cup \{a_i\}$
    $value \leftarrow value + v_i$
    $weight \leftarrow weight + w_i$

Knapsack Problem: GREEDY-BY-RATIO

- We saw example where GREEDY-BY-RATIO algorithm was suboptimal
- The following example show that it could be arbitrarily bad
- The ratio v_i/w_i is called the density of item a_i
- Density is not necessarily a good measure of profitability

GREEDY-BY-RATIO

ID	weight	value
1	1	2
2	C	C

C : is the capacity

Ouput: $\{1\}$ weight 1, value 2

Optimal: $\{2\}$ weight C, value C

Knapsack Problem: MODIFIED-GREEDY-BY-RATIO

- Can improve GREEDY-BY-RATIO with a simple trick
- Run another algorithm in parallel- chooses the first item this one skips
- Return the best of the above two algorithms

Algorithm MODIFIED-GREEDY-BY-RATIO

SORT items by v_i/w_i ▷ assume $v_1/w_1 \geq v_2/w_2 \geq \ldots \geq v_n/w_n$

$weight \leftarrow 0$ ▷ total weight collected so far

$value \leftarrow 0$ ▷ total value collected so far

$S \leftarrow \emptyset$ ▷ initially the knapsack is empty

for $i = 1 \to n$ **do**

  if $weight + w_i < C$ **then**

    • $S \leftarrow S \cup \{a_i\}$ • $value \leftarrow value + v_i$ • $weight \leftarrow weight + w_i$

$k \leftarrow$ index of first item skipped above

if $value \geq v_k$ **then return** S

else **return** $\{a_k\}$

Knapsack Problem: MODIFIED-GREEDY-BY-RATIO

MODIFIED-GREEDY-BY-RATIO algorithm is 2-approximate

- Let S be the output of $\mathcal{A} = $ MODIFIED-GREEDY-BY-RATIO
- Let k be the index of first item skipped by $\mathcal{A}$
- $v_1 + v_2 + \ldots + v_{k-1} \leq \text{OPT}$ **why?**
- $v_1 + v_2 + \ldots + v_{k-1} + v_k \geq \text{OPT}$
- Actually, $v_1 + v_2 + \ldots + v_{k-1} + c \cdot v_k \geq \text{OPT}$ $\triangleright$ $c = \frac{C-(w_1+w_2+\ldots+w_{k-1})}{w_k}$
  - numerator is remaining capacity after packing the first $k-1$ items
  - c-fraction of a_k can be packed (if fractional packing is allowed)
  - suppose we packed $\{a_1, \ldots, a_{k-1}\}$ and c-fraction of a_k
  - we consumed whole C it is optimal as we took largest density
- The two red statements implies that either

  $v_1 + v_2 + \ldots + v_{k-1} \geq \text{OPT}/2$ OR $v_k \geq \text{OPT}/2$

$f(S) = \max\left\{v_1 + v_2 + \ldots + v_{k-1}\,,\, v_k\right\}$

Knapsack Problem: MODIFIED-GREEDY-BY-RATIO

MODIFIED-GREEDY-BY-RATIO algorithm is 2-approximate

We show that this analysis is tight

Consider the instance

- $U = \{a_1, a_2, a_3\}$
- $v_1 = 1 + \epsilon/2, \;\; v_2 = v_3 = 1$
- $w_1 = 1 + \epsilon/3, \;\; w_2 = w_3 = 1$
- $C = 2$

$S = \{a_1\}$

$\text{OPT} = \{a_2, a_3\}$

- Let S be the output of $\mathcal{A} =$ MODIFIED-GREEDY-BY-RATIO
- Approximation ratio achieved is arbitrarily close to 2
- Runtime of $\mathcal{A}$ is $O(n \log n)$ (pseudo-polynomial)
  - each density computation takes $\log(C \cdot \sum_{i=1}^{n} v_i)$
- Recall runtime of dynamic programming algorithm is $O(n \cdot C)$

A pseudo-polynomial time algorithm for KNAPSACK

MODIFIED-GREEDY-BY-RATIO algorithm for KNAPSACK is

- pseudo polynomial in runtime
- 2-approximate

We identify cases where its output is even better

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

Lemma 2: If for some $0 < \epsilon < 1/2$, all $v_i \leq \epsilon \mathrm{OPT}$, then MODIFIED-GREEDY-BY-RATIO is $(1-\epsilon)$-approximate

We will use these lemmas to obtain a PTAS 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

- Items sorted by $v./w.$ $\implies \forall\, 1 \leq i \leq k,\ \frac{v_i}{w_i} \geq \frac{v_k}{w_k} \implies v_i \geq w_i \frac{v_k}{w_k}$
- Adding up all these inequalities:

$$v_1 + v_2 + \cdots + v_k \ \geq\ (w_1 + w_2 + \cdots + w_k)\,\frac{v_k}{w_k}$$

$$\implies w_k \cdot \frac{v_1 + v_2 + \cdots + v_k}{w_1 + w_2 + \cdots + w_k} \ \geq\ v_k$$

- Recall a_k is the first item skipped by $\mathcal{A} =$ MODIFIED-GREEDY-BY-RATIO
- Plugging $w_1 + w_2 + \cdots + w_k > C$ in above inequality:

$$v_k \ \leq\ \frac{w_k}{C} \cdot v_1 + v_2 + \cdots + v_k$$

PTAS for the KNAPSACK Problem

- Since all $w_i \leq \epsilon C$, plugging $w_k \leq \epsilon C$ in above inequality:

$$v_k \leq \epsilon \cdot (v_1 + v_2 + \cdots + v_k) \leq \frac{\epsilon}{1-\epsilon} \cdot (v_1 + v_2 + \cdots + v_{k-1})$$

- If $(v_1 + v_2 + \cdots + v_{k-1}) \geq (1-\epsilon)\text{OPT}$, then we have $(1-\epsilon)$-approximation
- If $(v_1 + v_2 + \cdots + v_{k-1}) < (1-\epsilon)\text{OPT}$, then $v_k \leq \epsilon\text{OPT}$

  Combining these two:

$$v_1 + v_2 + \cdots + v_{k-1} + v_k < (1-\epsilon)\text{OPT} + \epsilon\text{OPT} < \text{OPT}$$

  which contradicts the fact that a_{k-1} is the last item chosen

Thus, either $(v_1 + v_2 + \cdots + v_{k-1}) \geq (1-\epsilon)\text{OPT}$ or $v_k \geq (1-\epsilon)\text{OPT}$, giving a $(1-\epsilon)$-approximation

PTAS for the KNAPSACK Problem

Lemma 2: If for some $0 < \epsilon < 1/2$, all $v_i \leq \epsilon\text{OPT}$, then MODIFIED-GREEDY-BY-RATIO is $(1-\epsilon)$-approximate

- Since a_k is the first item skipped by A

$$(v_1 + v_2 + \cdots + v_{k-1} + v_k) \geq \text{OPT}$$

- Since $v_k \leq \epsilon\text{OPT}$, then

$$(v_1 + v_2 + \cdots + v_{k-1}) \geq (1-\epsilon)\text{OPT}$$

- This gives a $(1-\epsilon)$-approximation

PTAS for the KNAPSACK Problem

In any optimal solution with total value OPT and any $0 < \epsilon < 1$,
there are $\leq \lceil 1/\epsilon \rceil$ items with values $\geq \epsilon\text{OPT}$

▷ This follows from basic counting

We use this fact to design a PTAS for KNAPSACK problem

1. First, get 'heavier' items of the OPT-SOLUTION values $> \epsilon\text{OPT}$
2. Use MODIFIED-GREEDY-BY-RATIO for lighter items among remaining

Problem: How to get the heavier items of the OPT-SOLUTION

▷ OPT is unknown, only a bound on number of heavier items is known

1. Try all $n^{\lceil 1/\epsilon \rceil + 1}$ subsets of U of sizes $\leq \lceil 1/\epsilon \rceil$
2. and select the most valuable feasible subset

PTAS for the KNAPSACK Problem

For a set $S \subseteq U$, $w(S) = \sum_{i \in S} w_i$ and $v(S) = \sum_{i \in S} v_i$

Algorithm : KNAPSACK-PTAS

$h \leftarrow \lceil 1/\epsilon \rceil$

max-tot-value-heavy-items $\leftarrow 0$

max-value-heavy-set $\leftarrow \emptyset$

for each $H \subseteq U$, such that $|H| \leq h$ and $w(H) \leq C$ **do**

$v_m \leftarrow \mathrm{argmin}_{i \in H}(v_i)$

$U' \leftarrow \{a_i \in U \setminus H : v_i < v_m\}$ ▷ lighter items in $U \setminus H$

$S \leftarrow$ MODIFIED-GREEDY-BY-RATIO$(U', C - w(H))$

if max-tot-value-heavy-items $< v(H) + v(S)$ **then**

max-tot-value-heavy-items $\leftarrow v(H) + v(S)$

max-value-heavy-set $\leftarrow H \cup S$

Runtime:

- For each of the $O(n^{\lceil 1/\epsilon \rceil + 1})$ subsets, linear work is done before calling MODIFIED-GREEDY-BY-RATIO
- Sorting done only once but dominated by loop
- Total time polynomial in n and exponential in $1/\epsilon$

Approximation Ratio:

- Consider the iteration where the set H is part of optimal solution

  ▷ since all subsets of size at most h are checked

  H can not have more than h items of value $> \epsilon \cdot \text{OPT}$
- Let $U' \leftarrow \{a_i \in U \setminus H : v_i < v_m\}$ and OPT' be optimal value for U'

  ▷ $\text{OPT} = v(H) + \text{OPT}'$
- Since $\forall\ i \in U', v_i \leq \epsilon \cdot \text{OPT}$, solution for U' has value $\geq (1-\epsilon)\text{OPT}'$
- value of KNAPSACK-PTAS output is $v(H) + (1-\epsilon)\text{OPT}' \geq (1-\epsilon)\text{OPT}$

▷ $v(H)$ may be 0, i.e. optimal solution may not include any heavy item