KNAPSACK Problem

Outline

- Definitions of PTAS, FPTAS
- 2-approximation Algorithm for Knapsack
- PTAS/FPTAS for Knapsack

Recap

defn: (for minimization Problem)

ALG is a ρ-approximation, for some $\rho > 1$,

if $ALG(I) \leq \rho \, OPT(I)$ for all inputs I

We want ρ to close to 1.

For example,

VERTEX COVER

- 2-approximation algorithm is possible
- 1.361-approximation algorithm not possible unless P=NP.

  So we cannot get really close to $\rho = 1$.

But for some Problems we can get arbitrarily close to $\rho = 1$.

Polynomial-time approximation schemes (PTAS)

Algorithm Alg with input

- Instance I of the problem
- Parameter $\epsilon > 0$

Alg is PTAS if (for minimization problems)

① $Alg(I, \epsilon) \leq (1+\epsilon)\, OPT(I)$ for all inputs I, ϵ.

② running time is polynomial in size of input instance I for every fixed ϵ.

The running time of the algorithm should be polynomial in n, its dependency on ϵ can be exponential.

If the dependency on the parameter $1/\epsilon$ is also polynomial then we call it as fully polynomial time approximation scheme (FPTAS)

Eg: $O\left(\left(\frac{1}{\epsilon}\right)^4 n^2\right)$ or $O\left(2^{1/\epsilon} n \log n\right)$ or $O(n^{2/\epsilon})$ (allowed)

Polynomial in $1/\epsilon$ → FPTAS

not Polynomial in $1/\epsilon$

- if the dependency

$O\left(\frac{1}{\epsilon} 2^n\right)$ (not allowed)

Alg is PTAS if (for ~~minimization~~ maximization Problems)

① $Alg(I, \epsilon) \geq (1-\epsilon)\, OPT(I)$ for all inputs I, ϵ.

② running time is polynomial in size of input instance I.

Knapsack Problem (0-1)

Recap:

A thief is robbing a store and can carry a maximal weight of W into his knapsack. There are n items available in the store and weight of i^{th} item is w_i and its value is v_i. The problem is to choose a subset of the items of maximum total value that will fit in the knapsack.

Remark: We have looked at this problem while dealing with Dynamic Programming in Algorithms I.

Running time : $O(nW)$

Where n : # of items

W : knapsack capacity.

Example: $W = 45$

item	weight	value
1	10	20
2	20	65
3	30	50

Soln: Pick item 1 and item 2

total weight $= 10+20 = 30 \leq 45$

Total Value $= 20+65 = 85$

Remark: The Knapsack Problem is NP-hard.

(when W is arbitrary)

[SUBSET SUM Problem is a special case of Knapsack Problem, when $v_i = w_i$ for all i]

* we delete all items with weight $w_i > W$ as they cannot be considered by any solution.

Next we look at a 2-approximation algo for 0-1 knapsack.

This is a maximization Problem, we want to design an algorithm Alg such that

$Alg(I) \geq \frac{1}{2} OPT(I)$ for every i/p instance I

(a) Consider items in order of their value and add to the set I till the weight is not exceeded. (greedy idea)

Eg: $w_n = W$, $v_n = W$ [This greedy idea can be really bad]

all other items value $v_i = W-1$, $w_i = 1$

The item of maximum value is item n which alone fills the knapsack.

However, by taking many of the other items, we can take as many as $\min\{n-1, W\}$ items each of value $v_i = W-1$ reaching a much higher total value.

(b) Consider items in decreasing order of their density $d_i = \frac{v_i}{w_i}$ and add to the set I till the weight is not exceeded.

This greedy choice can also be very bad.

Eg: $w_1 = v_1 = W$

$w_2 = 1, \quad v_2 = 1+\varepsilon$

density $d_1 = 1, \quad d_2 = 1+\varepsilon$

$\therefore d_2 > d_1$

After fitting item 2 in the knapsack, item 1 no longer fits, so we endup with a total value $1+\varepsilon$, while optimal value is W.

Claim: Running both ⓐ and ⓑ greedy algorithm above, and taking the solution higher value is a 2-approximation algorithm.

Proof:- let V_a & V_b be the values achieved by greedy Algorithms ⓐ & ⓑ respectively.

let OPT denote the maximum possible value.

For algorithm ⓑ let I be the set of items Packed into the knapsack.

let j be the first item that didn't fit into the knapsack.

Clearly $\sum_{i \in I} v_i = V_b$ and $v_j \leq V_a$

↓ as algo ⓐ Starts by taking the single item of maximum value

Claim: $\sum_{i \in I} v_i + v_j \geq OPT$

(ie, $V_a + V_b \geq OPT$, so the larger of V_a & V_b must be at least $\frac{1}{2}$ of OPT)

Proof of the Claim :-

Suppose if we could cut a part of the last item so that it exactly fills the knapsack, that would be the optimum soln if taking a partial item was allowed.

(ie, optimal soln to the fractional knapsack)

ie, $\sum_{i \in I} v_i + v_j \geq$ OPT of fractional knapsack

$\geq OPT$.

DP Algorithm for Knapsack [Assume all i/p values are integers]

Input: $X = \{x_1, x_2, \ldots x_n\}$ of n items

- weight(x_i)
- Value(x_i)

} Positive integers

- W Positive integer

Goal: Find a subset $S \subseteq X$ whose total value is maximized under the condition that its total weight is at most W.

Notation: For a subset $S \subseteq X$

- $\text{weight}(S) = \sum_{x_i \in S} \text{weight}(x_i)$
- $\text{Value}(S) = \sum_{x_i \in S} \text{Value}(x_i)$
- $S_i = \{x_1 \ldots x_i\}$ = first i items in X.

$V_{tot} = \sum_{x_i \in X} \text{Value}(x_i)$ = total value of all items

For $1 \le i \le n$ and $0 \le j \le V_{tot}$ define

$$A[i,j] = \min\{ \text{weight}(S) : S \subset \{x_1, \ldots x_i\} \text{ and } \text{value}(S) = j \}$$

In Otherwords,

$A[i,j]$ denotes the minimum possible weight of any subset S of the first i items such that value (S) is exactly j.

When there is no subset $S \subset \{x_1, \ldots x_i\}$ of value exactly j then we define $A[i,j] = \infty$.

$$OPT := \max\{ j : 0 \le j \le V_{tot} \text{ and } A[n,j] \le W \}$$

if we can compute all values $A[i,j]$ then we can compute OPT.

As usual in DP, the values $A[i,j]$ are computed bottom-up by filling in a table.

Recursive formula for $A[i,j]$ is given as follows.

$$A[i,j] = \begin{cases} 0 & \text{if } j = 0 \\ \infty & \text{if } i = 0,\ j > 0 \\ A[i-1,j] & \text{if } i > 0 \ \&\ 0 < j < \text{value}(x_i) \\ \min\{A[i-1,j],\ A[i-1, j-\text{value}(x_i)] + \text{weight}(x_i)\} & \text{if } i > 0 \ \&\ j \geq \text{value}(x_i) \end{cases}$$

We call the above DP algorithm as

"Integer Value Knapsack (n, W)"

Theorem: Suppose all values in a KNAPSACK instance are integers. Then the Problem can be solved in $O(n \cdot V_{tot})$ time, where $V_{tot} = \text{Value}(X)$ is the total Value of all items.

An FPTAS for Knapsack

To apply the above result when Values are arbitrarily large and need not even be integers we apply the following strategy:

We replace each $Value(x_i)$ by a $Value^*(x_i)$ and then compute an optimal subset for these new values using the algorithm from the previous section.

To make the above idea work, values $Value^*$ should satisfy the following three conditions:

(i) Each $Value^*(x_i)$ should be an integer, so that the algorithm from the previous section can be applied.

(ii) the sum $\sum_i Value^*(x_i)$ should be sufficiently small ie, Polynomial in n, so that the algorithm from the previous section will run in Polynomial time

(iii) Each $Value^*(x_i)$ should be sufficiently close to $Value(x_i)$ so that the error we make by working with $Value^*$ is under control.

We know that each Value(x_i) is a Positive real number.

We Partition $\mathbb{R}^+$ into intervals of length Δ,

where Δ is a Parameter that we will select later.

Thus we Cover $\mathbb{R}^+$ with $(0, \Delta], (\Delta, 2\Delta], \ldots$ etc.

Then we replace each Value(x_i) by the Value j such that Value(x_i) lies in the j^{th} interval $((j-1)\Delta, j\Delta]$

Formally

$$\text{Value}^*(x_i) = \left\lceil \frac{\text{Value}(x_i)}{\Delta} \right\rceil$$

Next, we need to select Δ such that we also satisfy Conditions (ii) & (iii).

Hence the error in the individual values does not increase & intitutively Condition (iii) Satisfied

Condition (ii)

$$\text{Value}^*(x_i) \leq \left\lceil \frac{\max_i \text{Value}(x_i)}{\Delta} \right\rceil$$

$$\leq \left\lceil \frac{\max_i \text{Value}(x_i)}{\left(\frac{\varepsilon}{n}\right) LB} \right\rceil$$

$$\leq \left\lceil \frac{n}{\varepsilon} \right\rceil$$

ie, $\sum_i \text{Value}^*(x_i) = O\left(\frac{n^2}{\varepsilon}\right)$

∴ Condition (ii) is satisfied.

Picking Δ

To obtain PTAS we must compute a solution whose total value is at least $(1-\varepsilon)OPT$.

That is the total error of the solution should be at most $\varepsilon . OPT$.

If the error in an individual value is less than Δ, then the total error of the solution must be $n\Delta$ (as we have n items).

$\therefore$ we want to choose Δ such that

$$n\Delta \leq \varepsilon . OPT \Rightarrow \Delta \leq \frac{\varepsilon}{n} OPT$$

$\therefore$

$$\text{Pick} \quad \Delta := \frac{\varepsilon}{n} . OPT \quad -(1)$$

However, we do not know OPT, so our algorithm cannot set Δ according to (1).

To overcome this Problem, we use suitable lower bound LB instead of OPT

we have

$$OPT \geq \max_i \text{Value}(x_i)$$

$\therefore$ we set $\Delta := \frac{\varepsilon}{n} LB \leq \frac{\varepsilon}{n} OPT$

where $LB = \max_i \text{Value}(x_i)$.

By using LB instead of OPT, the interval size Δ can only become Smaller.

Algorithm

Knapsack-FPTAS(X, W, ε)

1: Set LB $\leftarrow \max_{1 \leqslant i \leqslant n} value(x_i)$ and $\Delta \leftarrow (\varepsilon/n) \cdot$ LB.
2: For all $1 \leqslant i \leqslant n$, let $value^*(x_i) \leftarrow \left\lceil \frac{value(x_i)}{\Delta} \right\rceil$.
3: Compute a subset $S^* \subset X$ of maximum value and total weight at most W with algorithm *IntegerValueKnapsack*, using the new values $value^*(x_i)$ instead of $value(x_i)$.
4: **return** S^*

To Prove that algorithm is a PTAS we need to show

① Output is Valid : Selected Subset S must have total weight $\leq W$ [Trivial : We did not change the weight of the items]

② output of the Algorithm $\geq (1-\epsilon)OPT$

③ Running time is Polynomial in n

Analysis of the Approximation Ratio

S = Computed subset, which is optimal for value*

S_{opt} = optimal subset for original values

To show

Goal: $\text{value}(S) \geq (1-\epsilon)\ \text{value}(S_{opt})$

$= (1-\epsilon)\ OPT.$

Few observations

① Computed subset S is optimal for $Value^*$:

$$Value^*(S) \geq Value^*(S_{opt})$$

② Relation between $Value$ & $Value^*$

$$Value^*(x_i) = \left\lceil \frac{Value(x_i)}{\Delta} \right\rceil$$

Where $\Delta = \frac{\varepsilon}{n} \cdot LB$ and $LB = \max_i Value(x_i) \leq OPT$

$$\frac{Value(x_i)}{\Delta} \leq Value^*(x_i) \leq \frac{Value(x_i)}{\Delta} + 1$$

$$\text{Value}(S) = \sum_{x_i \in S} \text{value}(x_i)$$

$$\geq \sum_{x_i \in S} \left(\Delta \, \text{value}^*(x_i) - 1\right)$$

$$= \Delta \left(\sum_{x_i \in S} \text{Value}^*(x_i)\right) - |S| \cdot \Delta$$

$$\geq \Delta \cdot \left(\sum_{x_i \in S_{opt}} \text{value}^*(x_i)\right) - n \cdot \Delta$$

$$\geq \sum_{x_i \in S_{opt}} \text{value}(x_i) - n\Delta$$

$$= OPT - n \cdot \frac{\varepsilon}{n} LB$$

$$= OPT - \varepsilon \, LB \qquad \left[\text{we know } LB \leq OPT\right]$$

$$\geq OPT - \varepsilon \, OPT$$

$$\text{Value}(S) = (1-\varepsilon) OPT$$

Running time

Step 1: Const time to compute LB

Step 2: $O(n)$ time to compute Value^*

Step 3: int value knapsack DP runs in $O(n V_{tot})$

$$\Delta = \frac{\varepsilon}{n} \cdot LB$$

$$V_{tot} = \sum_{x_i \in X} \text{value}^*(x_i)$$

$$\leq n \cdot \max_i \text{value}^*(x_i)$$

$$\leq n \cdot \max_i \left\lceil \frac{\text{value}(x_i)}{\Delta} \right\rceil$$

$$= n \cdot \left\lceil \max_i \frac{\text{value}(x_i)}{\Delta} \right\rceil$$

$$= n \cdot \left\lceil \max_i \frac{\text{value}(x_i)}{\frac{\varepsilon}{n} \cdot LB} \right\rceil$$

$$= \frac{n^2}{\varepsilon} \frac{1}{LB} \left\lceil \max_i \text{value}(x_i) \right\rceil$$

$$= O(n^2/\varepsilon)$$

Step4: $O(n)$; to output the items computed in Step3 but with original values

Running time: $O(n^3/\varepsilon)$

Running time is Polynomial in n & $1/\varepsilon$

$\therefore$ FPTAS

Summary

- PTAS : has an additional Parameter.

  $Alg(I) \leq (1+\epsilon)\ OPT$ (minimization)

- Running time is Polynomial in input size, for FPTAS also Polynomial in $1/\epsilon$

- General strategy to design a PTAS
  - develop exact algorithm that is efficient when input values are "small" integers
  - Replace actual input values by "small" integers