CS102: Introduction to Algorithms – FPTAS for 0/1 Knapsack

Lecture Notes

Contents

1 Introduction

In many real-world problems, we want to choose the best option among many possibilities. Some examples include:

- Packing items in a backpack to maximize value (Knapsack)
- Selecting a subset of numbers whose sum is closest to a target (Subset Sum)
- Dividing tasks among machines efficiently (Scheduling)

Some of these problems are **NP-hard**, meaning that finding the exact optimal solution may take *exponentially long time* for large inputs. But in many cases, an **approximate solution** that is "good enough" is acceptable.

—

2 Approximation Algorithms and FPTAS

2.1 What is an Approximation Algorithm?

An **approximation algorithm** provides a solution close to optimal:

$$(1 - \varepsilon) \cdot OPT \leq S \leq (1 + \varepsilon) \cdot OPT$$

where:

- S = solution returned by the algorithm
- OPT = optimal solution
- ε = small number representing allowed error (e.g., 0.1 for 10% error)

2.2 Fully Polynomial-Time Approximation Scheme (FPTAS)

A **Fully Polynomial-Time Approximation Scheme (FPTAS)** is:

- An algorithm that can produce a $(1 - \varepsilon)$-approximation for *any* $\varepsilon > 0$
- Runs in polynomial time in both:
  - n = number of items
  - $1/\varepsilon$ = precision parameter

Key idea: Convert a *pseudo-polynomial algorithm* (fast if numbers are small) into a *truly polynomial* approximation algorithm by scaling.

—

3 0/1 Knapsack Problem – Step by Step

3.1 Problem Definition

Given:

- n items, each with value v_i and weight w_i
- Knapsack capacity W

Goal: Pick a subset of items to maximize total value without exceeding W:

$$\max \sum_{i=1}^{n} v_i x_i \quad \text{s.t.} \quad \sum_{i=1}^{n} w_i x_i \leq W, \quad x_i \in \{0, 1\}$$

3.2 Weight-based Dynamic Programming

We can define a DP table as:

$DP[i][w]$ = maximum value achievable using first i items with total weight $\leq w$

Recurrence:

$$DP[i][w] = \begin{cases} DP[i-1][w], & \text{if } w_i > w \\ \max(DP[i-1][w], v_i + DP[i-1][w-w_i]), & \text{otherwise} \end{cases}$$

Time Complexity: $O(nW)$ – called **pseudo-polynomial** because it depends on W, which may be very large.

Example:

$$(v, w) = \{(60, 10), (100, 20), (120, 30)\}, \quad W = 50$$

DP table size = 3×50 = manageable. But if $W = 10^6$, runtime = $O(3 \cdot 10^6)$, too slow.

—

3.3 Value-based Dynamic Programming

Alternative DP:

$DP[i][v]$ = minimum weight needed to achieve total value v using first i items

Recurrence:

$$DP[i][v] = \begin{cases} DP[i-1][v], & \text{if } v_i > v \\ \min(DP[i-1][v], w_i + DP[i-1][v-v_i]), & \text{otherwise} \end{cases}$$

We then choose the largest v s.t. $DP[n][v] \leq W$.

Time Complexity: $O(nV_{\max})$, where $V_{\max} = \sum_i v_i$

—

4 Why DP is Pseudo-Polynomial

The time depends on $V_{\max}$ (sum of item values) or W (sum of weights), not on the *number of bits needed to represent input*:

$$\text{Input size } \sim O(n \log W)$$

Hence, it's not truly polynomial.

—

Why the Standard (Weight-based) DP Cannot Be Turned into an FPTAS

The standard dynamic programming (DP) approach for the knapsack problem has runtime

$$O(nW),$$

where W is the numeric weight limit.

This makes it **pseudo-polynomial**, because W can be exponentially large compared to its bit-length $\log W$.

We cannot simply scale or round weights, because the constraint is a *hard limit*. A small rounding error could make the total weight exceed W, thus invalidating feasibility.

$$\text{Rounding weights } \Rightarrow \text{ breaks feasibility.}$$

Hence, there is **no known FPTAS** based on the weight-based DP.

Step 3: Why the Value-Based DP Works for FPTAS

The value-based DP computes:

$$dp[i][v] = \text{minimum weight required to achieve value } v,$$

and runs in

$$O(nV_{\max}),$$

where $V_{\max}$ is the maximum total value.

The idea is that instead of iterating over all possible values, we can *scale down the values* so that $V_{\max}$ becomes small and the runtime becomes polynomial.

The Scaling Trick (Core of FPTAS)

Let K be a scaling factor. We define new scaled values:

$$v_i' = \left\lfloor \frac{v_i}{K} \right\rfloor.$$

Now, solve the scaled instance using the value-based DP. Its runtime becomes

$$O(nV'_{\max}),$$

where

$$V'_{\max} = \sum_i v'_i \approx \frac{V_{\max}}{K}.$$

We choose K cleverly so that:

$$K = \frac{\varepsilon\, v_{\max}}{n},$$

which ensures runtime polynomial in n and $1/\varepsilon$, and also guarantees that the total loss due to rounding is within the ε margin.

5 Developing FPTAS for Knapsack

5.1 Step 1: Scaling Item Values

Choose:

$$K = \frac{\varepsilon \cdot v_{\max}}{n}$$

and define scaled values:

$$v'_i = \left\lfloor \frac{v_i}{K} \right\rfloor$$

We then run value-based DP on v'_i.

5.2 Step 2: Why This Scaling Works

- Each item loses at most K due to flooring:

$$0 \leq v_i - K \cdot \lfloor v_i/K \rfloor < K$$

- Total loss over n items $\leq nK = \varepsilon v_{\max} \leq \varepsilon OPT$

Hence, final solution $\geq (1-\varepsilon)OPT$.

—

6 Toy Problem Example

We have four items with their values and weights as follows:

Item	**Value** (v_i)	**Weight** (w_i)
1	120	10
2	100	20
3	60	15
4	80	25

Knapsack capacity: $W = 40$.

Total value $V_{\max} = 120+100+60+80 = 360$. Maximum single-item value $v_{\max} = 120$. We will construct an FPTAS with error parameter $\varepsilon = 0.25$ (i.e., 25% tolerance).

—

. Choosing the Scaling Factor K

We use the common scaling formula:

$$K = \frac{\varepsilon \cdot v_{\max}}{n}$$

where $n = 4$.

So,

$$K = \frac{0.25 \times 120}{4} = \frac{30}{4} = 7.5$$

This $K > 1$ — meaning we will *shrink* values, which visibly reduces DP table size and is easy to understand.

—

3. Scaled Values

We compute:

$$v'_i = \left\lfloor \frac{v_i}{K} \right\rfloor$$

$$v'_1 = \lfloor 120/7.5 \rfloor = 16,$$
$$v'_2 = \lfloor 100/7.5 \rfloor = 13,$$
$$v'_3 = \lfloor 60/7.5 \rfloor = 8,$$
$$v'_4 = \lfloor 80/7.5 \rfloor = 10.$$

So our scaled instance becomes:

Item	**Value** (v_i)	**Scaled Value** (v'_i)	**Weight** (w_i)
1	120	16	10
2	100	13	20
3	60	8	15
4	80	10	25

Now the maximum total scaled value $V'_{\max} = 16 + 13 + 8 + 10 = 47$, while original $V_{\max} = 360$. Thus, the DP table will have roughly 47 columns instead of 360 — a major reduction.

—

. Value-Based Dynamic Programming (on Scaled Values)

We define:

$DP[i][v']$ = minimum total weight required to achieve scaled value v' using first i items.

Initialization:

$$DP[0][0] = 0, \quad DP[0][v' > 0] = \infty$$

Recurrence:

$$DP[i][v'] = \min(DP[i-1][v'],\ w_i + DP[i-1][v' - v'_i])$$

—

. Manual Computation (Simplified Illustration)

For brevity, consider a few key combinations:

- Item 1 ($v'_1 = 16, w_1 = 10$): $DP[1][16] = 10$
- Item 2 ($v'_2 = 13, w_2 = 20$): $DP[2][13] = 20, \quad DP[2][29] = DP[1][16] + 20 = 30$
- Item 3 ($v'_3 = 8, w_3 = 15$): New combinations: $DP[3][8] = 15$, $DP[3][24] = DP[2][16] + 15 = 25$, etc.
- Item 4 ($v'_4 = 10, w_4 = 25$): Possible new sums such as $DP[4][18] = 25, DP[4][26] = DP[3][16] + 25 = 35$, etc.

The feasible combinations with $DP[v'] \leq W(= 40)$ include, for instance, Items {1,2,3} $\rightarrow$ total scaled value $16 + 13 + 8 = 37$ and weight $10 + 20 + 15 = 45$ (too heavy). Items {1,3,4} $\rightarrow$ scaled value $16 + 8 + 10 = 34$, weight $10 + 15 + 25 = 50$ (too heavy). Items {1,2} $\rightarrow$ scaled value 29, weight 30 $\rightarrow$ feasible.

Hence the best feasible scaled sum within capacity is approximately $v^* = 29$.

—

. Mapping Back to Real Value

The selected scaled sum $v^* = 29$ corresponds to original items {1,2} with total actual value:

$$A = v_1 + v_2 = 120 + 100 = 220$$

The exact optimal solution (by enumeration) also gives items {1,2} with value $OPT = 220$. So our approximation is in fact optimal here.

—

. Approximation Guarantee

In general, due to rounding:

$$K \cdot v'_i \leq v_i < K \cdot (v'_i + 1)$$

$$\Rightarrow v_i - K \cdot v'_i < K$$

Summing across items gives total loss $< nK = n \cdot \frac{\varepsilon v_{\max}}{n} = \varepsilon v_{\max} \leq \varepsilon \cdot OPT$.

Hence,

$$(1 - \varepsilon) \cdot OPT \leq A \leq OPT$$

In this example, $A/OPT = 1.0 \geq (1 - \varepsilon) = 0.75$.

—

. Observation

- Original value space: $0 \ldots 360$ (large DP table) - Scaled value space: $0 \ldots 47$ (much smaller DP table) - Runtime reduces roughly 8-fold. - The solution remains within 25% of the true optimum in worst case.

—

. Key Takeaways

- Scaling compresses the value space while maintaining a bounded error.
- $K > 1$ makes the reduction visible and easy to understand.
- Value-based DP is the foundation for FPTAS — efficient and accurate.

7 Proof Sketch for $(1-\varepsilon)$-Approximation

For each item i:

$$v_i < K(v_i' + 1) \implies v_i - Kv_i' < K.$$

Summing over items in the optimal set:

$$OPT = \sum_{i \in OPT} v_i < K \sum_{i \in OPT} v_i' + nK.$$

Thus,

$$OPT' > \frac{OPT}{K} - n.$$

Since the DP finds the optimal scaled solution ($A' \geq OPT'$):

$$A' \geq \frac{OPT}{K} - n.$$

Multiplying by K:

$$A \geq KA' \geq OPT - nK.$$

Choose $K = \frac{\varepsilon v_{\max}}{n}$. Since $v_{\max} \leq OPT$,

$$nK = n \cdot \frac{\varepsilon v_{\max}}{n} = \varepsilon v_{\max} \leq \varepsilon OPT.$$

Therefore:

$$A \geq OPT - \varepsilon OPT = (1-\varepsilon)OPT.$$

Runtime Analysis

After scaling:

$$v_i' \leq \frac{v_i}{K} \leq \frac{v_{\max}}{K} = \frac{v_{\max}}{\varepsilon v_{\max}/n} = \frac{n}{\varepsilon}.$$

Hence:

$$V_{\max}' = \sum_i v_i' \leq n \cdot \frac{n}{\varepsilon} = \frac{n^2}{\varepsilon}.$$

Runtime of the value-based DP:

$$T(n, \varepsilon) = O(n \cdot V_{\max}') = O\left(\frac{n^3}{\varepsilon}\right).$$

With optimization, it can be improved to:

$$T(n, \varepsilon) = O\left(\frac{n^2}{\varepsilon}\right),$$

which is polynomial in both n and $1/\varepsilon$.

8 Intuition Behind $K = \varepsilon v_{\max}/n$

Let K be a scaling factor.

We define new scaled values:

$$v'_i = \left\lfloor \frac{v_i}{K} \right\rfloor .$$

Now, solve the scaled instance using the value-based DP. Its runtime becomes:

$$O(nV'_{\max}),$$

where

$$V'_{\max} = \sum_i v'_i \approx \frac{V_{\max}}{K}.$$

We choose K cleverly so that:

$$K = \frac{\varepsilon\, v_{\max}}{n},$$

which ensures that the runtime is polynomial in n and $1/\varepsilon$, and also guarantees that the total loss due to rounding is within the ε-margin.

—

9 Other Problems Admitting FPTAS

- **Subset Sum:** Scale numbers; DP tracks sums target
- **Partition Problem:** Special case of Subset Sum
- **0/1 Knapsack:** Values scaled as shown

Not applicable: Problems without pseudo-polynomial DP (e.g., general TSP, Bin Packing).

Reason: The Core Idea Behind FPTAS Scaling

A **Fully Polynomial Time Approximation Scheme (FPTAS)** works by scaling down large numerical parameters (typically values or costs) to make a pseudo-polynomial dynamic programming (DP) algorithm run in true polynomial time.

This works only for problems that:

- Already have pseudo-polynomial algorithms (e.g., DP with numeric limits), and
- Are optimization problems with bounded numeric inputs.

Thus, the scaling trick converts a pseudo-polynomial algorithm $\rightarrow$ true polynomial time, at the cost of a small approximation error.

2. Where the Trick Works

Let's check a few famous problems and whether the scaling trick (FPTAS) applies:

booktabs, array, graphicx

cc

Problem	Has pseudo-poly DP?	FPTAS possible?	What we scale?	Remarks
0/1 Knapsack	Yes ($O(nW)$ or $O(nV_{\max})$)	Yes	Values v_i	Canonical example
Subset Sum	Yes ($O(n \cdot T)$)	Yes	Target sum / subset sums	Essentially same as knapsack
Partition Problem	Yes (special case of Subset Sum)	Yes	Subset sums	FPTAS via same scaling trick
Scheduling ($1\|\|L_{\max}$)	Exact poly algo	Not needed	—	Already polynomial
Scheduling ($1\|\|\sum T_i$)	NP-hard, no pseudo-poly DP	No	—	No FPTAS possible
Travelling Salesman (metric)	No pseudo-poly DP	No	—	Only PTAS for special cases (e.g., Euclidean)
Bin Packing	No pseudo-poly DP	No	—	Only Asymptotic PTAS (APTAS)

3. Why It Works for Some and Not Others

The key structure enabling FPTAS is the *pseudo-polynomial DP form*:

$$DP[i][x] = \text{optimum value using first } i \text{ items and parameter } x,$$

where x runs up to a numeric limit such as W or $V_{\max}$.

This dependence on numeric magnitude (not bit-length) is what makes the algorithm pseudo-polynomial.

If we can scale those numeric parameters (like v_i or weights) by a factor related to ε, we can make the DP's range polynomial in n and $1/\varepsilon$, yielding a truly polynomial-time approximation algorithm.

4. Example: Subset Sum Problem

Problem: Given numbers $a_1, a_2, \ldots, a_n$ and a target T, find a subset whose sum is $\leq T$ and as close to T as possible.

DP (Pseudo-polynomial): We maintain all possible subset sums up to T. Time complexity $= O(nT)$.

Scaling Trick for FPTAS:

$$a_i' = \left\lfloor \frac{a_i}{K} \right\rfloor, \qquad K = \frac{\varepsilon\, a_{\max}}{n}.$$

Then the maximum sum becomes approximately $O(n/\varepsilon)$, so the runtime is $O(n^2/\varepsilon)$ — polynomial in n and $1/\varepsilon$.

The recovered subset sum gives a $(1-\varepsilon)$-approximation to the true optimum.

$$\Rightarrow \text{Hence, Subset Sum admits an FPTAS (and so does Partition).}$$

5. When It Fails

For problems like Job Scheduling, TSP, or Vertex Cover, the input cannot be expressed as a DP bounded by a numeric sum. Thus, there's no pseudo-polynomial structure to "scale down."

Hence:

$$\text{FPTAS} \neq \text{general-purpose trick;}$$

it applies only to problems already having pseudo-polynomial DPs.

6. Quick tips to check for FPTAS

c

Property	Needed for FPTAS?
Has pseudo-polynomial DP	Required
Numeric objective (value/weight)	Helps define scaling
NP-hardness due to numeric encoding	Usually true
Additive / combinatorial structure	Required
Continuous optimization	Not suitable

—

10 Summary

- Pseudo-polynomial algorithms are polynomial in numeric values, not input size
- FPTAS converts pseudo-poly DP into a true polynomial-time approximate algorithm
- Scaling factor $K = \varepsilon v_{\max}/n$ balances precision and speed
- Approximation error is controlled: solution $\geq (1-\varepsilon)OPT$
- Works for Knapsack, Subset Sum, Partition