Analysis of a Fully Polynomial-Time Approximation Scheme (FPTAS) for the 0-1 Knapsack Problem

Abstract

The 0-1 Knapsack Problem is a fundamental NP-hard problem in combinatorial optimization. While it admits a pseudo-polynomial time solution via dynamic programming, this becomes infeasible for large input values. This report presents a Fully Polynomial-Time Approximation Scheme (FPTAS) for the problem. We provide a detailed theoretical foundation, a step-by-step description of the value-scaling algorithm, and a comprehensive analysis of its approximation ratio, time complexity, and space complexity. The algorithm guarantees a solution within a factor of $(1-\epsilon)$ of the optimal value and runs in time polynomial in both the input size and $1/\epsilon$.

1 Introduction and Problem Definition

1.1 The 0-1 Knapsack Problem

Given a set of n items, where each item i has a weight w_i and a value v_i, and a knapsack with maximum weight capacity W, the 0-1 Knapsack Problem seeks to determine a subset of items $S \subseteq \{1, 2, ..., n\}$ that maximizes the total value $\sum_{i \in S} v_i$ subject to the constraint $\sum_{i \in S} w_i \leq W$. The "0-1" constraint means each item can be taken at most once.

1.2 Computational Complexity

The decision version of the 0-1 Knapsack problem is NP-Complete. However, it is considered *weakly* NP-Complete because it can be solved in pseudo-polynomial time $O(nW)$ using dynamic programming. This time is polynomial in the numerical value of W, but not in the length of its binary representation. When W or item values are large, this solution becomes impractical, necessitating efficient approximation schemes.

2 Approximation Schemes: PTAS and FPTAS

2.1 Definitions

A **Polynomial-Time Approximation Scheme (PTAS)** is an algorithm that, for any fixed accuracy parameter $\epsilon > 0$, produces a solution within a factor $(1+\epsilon)$ of optimal (for minimization) or $(1-\epsilon)$ (for maximization) in time polynomial in the input size n. The exponent of the polynomial may depend heavily on ϵ, e.g., $O(n^{1/\epsilon})$.

A **Fully Polynomial-Time Approximation Scheme (FPTAS)** is a stricter, more efficient class of algorithm. Its running time must be polynomial in *both* the input size n and the reciprocal of the precision parameter $1/\epsilon$. An FPTAS provides a robust guarantee that increasing accuracy (decreasing ϵ) leads to a predictable and manageable increase in computation time.

3 Foundations: Dynamic Programming Solution

The standard pseudo-polynomial dynamic programming (DP) solution constructs a table $dp[i][c]$, representing the maximum value achievable using the first i items with a total weight of exactly c. Its recurrence is:

$$dp[0][c] = 0 \quad \text{for } c = 0, \ldots, W$$
$$dp[i][c] = \max(dp[i-1][c],\ dp[i-1][c-w_i] + v_i) \quad \text{if } w_i \leq c$$

The final answer is $\max_{0 \leq c \leq W} dp[n][c]$. The time and space complexity are $O(nW)$. This DP on weights is the foundation upon which the FPTAS is built.

4 FPTAS for 0-1 Knapsack via Value Scaling

The core idea to transform the pseudo-polynomial DP into an FPTAS is to reduce the range of the value dimension, making it polynomial in n and $1/\epsilon$.

4.1 Algorithm Description

Algorithm 1 FPTAS for 0-1 Knapsack via Value Scaling

Require: Items with weights w_i, values v_i, capacity W, precision $\epsilon \in (0, 1]$.
Ensure: An approximate maximum knapsack value V_{approx} s.t. $V_{approx} \geq (1 - \epsilon) \cdot OPT$.

1: Find the maximum item value: $v_{max} = \max_i v_i$
2: Compute the scaling factor: $K = \frac{\epsilon \cdot v_{max}}{n}$
3: **for** each item i **do**
4: Compute scaled value: $v'_i = \lfloor v_i / K \rfloor$
5: **end for**
6: Let $V'_{total} = \sum_i v'_i$
7: Run a modified DP: Build table $F[i][s]$ = minimum weight to achieve scaled value s with first i items.
8: Find $s_{max} = \max\{s \mid F[n][s] \leq W\}$
9: **return** V_{approx} = The **original** total value corresponding to achieving scaled value s_{max}.

4.2 Key Implementation Details

The algorithm uses a different dynamic programming formulation for efficiency after scaling. Instead of tracking maximum value for a given weight, it tracks the **minimum weight** required to achieve a given scaled value s. This is because the sum of scaled

values V'_{total} is polynomially bounded. Define $F[i][s]$ as the minimum weight needed to achieve a total scaled value of *exactly* s using a subset of the first i items. The recurrence is:

$$F[0][0] = 0; \quad F[0][s] = \infty \text{ for } s > 0$$
$$F[i][s] = \min \begin{cases} F[i-1][s], \\ F[i-1][s - v'_i] + w_i \quad \text{if } s \geq v'_i \text{ and } w_i \leq W \end{cases}$$

After filling the table, we find the largest s such that $F[n][s] \leq W$. The corresponding original value (sum of v_i for items in the solution) is the algorithm's output.

5 Theoretical Analysis

5.1 Approximation Guarantee Proof

Let:

- OPT: Optimal value for the original problem.
- O: The set of items in the optimal solution.
- S: The set of items found by the FPTAS algorithm.
- $v'(X) = \sum_{i \in X} v'_i$: Total scaled value of a set X.

The floor function in scaling ensures $K \cdot v'_i \leq v_i < K \cdot (v'_i + 1)$. Therefore, for any set X:

$$v(X) - K \cdot v'(X) < |X| \cdot K$$

Applying this to the optimal set O (where $|O| \leq n$):

$$v(O) - K \cdot v'(O) < n \cdot K$$
$$\text{Since } v(O) = OPT, \quad OPT < K \cdot v'(O) + nK \tag{1}$$

Our dynamic programming finds the set S that is optimal for the *scaled* instance, so $v'(S) \geq v'(O)$. Multiplying by K:

$$K \cdot v'(S) \geq K \cdot v'(O)$$

From (1), we have $K \cdot v'(O) > OPT - nK$. Combining and using the fact that $v(S) \geq K \cdot v'(S)$ (as values are only floored down):

$$\begin{aligned} v(S) &\geq K \cdot v'(S) \\ &\geq K \cdot v'(O) \\ &> OPT - nK \end{aligned}$$

Substituting the definition of $K = (\epsilon \cdot v_{max})/n$ and noting $v_{max} \leq OPT$:

$$\begin{aligned} v(S) &> OPT - n \cdot \left(\frac{\epsilon \cdot v_{max}}{n}\right) \\ &> OPT - \epsilon \cdot v_{max} \\ &\geq OPT - \epsilon \cdot OPT \\ &\geq (1 - \epsilon) \cdot OPT \end{aligned}$$

Thus, $v(S) \geq (1 - \epsilon) \cdot OPT$, proving the approximation ratio.

5.2 Time Complexity Analysis

- **Step 1 (Find v_{max})**: $O(n)$.
- **Step 2-4 (Scale values)**: $O(n)$.
- **Step 5 (Dynamic Programming)**: This is the dominant step. The table F has dimensions $(n+1) \times (V'_{total}+1)$, where $V'_{total} = \sum v'_i$. Since $v'_i = \lfloor v_i/K \rfloor \leq \lfloor v_i/((\epsilon v_{max})/n) \rfloor \leq \lfloor n/\epsilon \rfloor$, we have $V'_{total} \leq n \cdot (n/\epsilon) = n^2/\epsilon$. Each cell update is $O(1)$, so the total time for DP is $O(n \times V'_{total}) = O(n \times (n^2/\epsilon)) = O(n^3/\epsilon)$. More careful implementation leads to a bound of $O(n^2/\epsilon)$.

The overall time complexity is $O(n^2/\epsilon)$ or $O(n^3/\epsilon)$, which is polynomial in both n and $1/\epsilon$, satisfying the FPTAS requirement.

5.3 Space Complexity Analysis

The space required is dominated by the DP table F, which has size $O(n \times V'_{total})$. Given $V'_{total} = O(n^2/\epsilon)$, the space complexity is $O(n^3/\epsilon)$. This can often be optimized to $O(n^2/\epsilon)$ using a space-optimized DP that only keeps the previous row.

6 Conclusion

This report has detailed the construction and analysis of an FPTAS for the 0-1 Knapsack problem. By intelligently scaling down the item values by a factor proportional to ϵ, we transform the standard pseudo-polynomial dynamic programming algorithm into one whose running time is fully polynomial in n and $1/\epsilon$. The provided proof confirms that the solution value is guaranteed to be at least $(1-\epsilon)$ times the optimal value. This FPTAS effectively bridges the gap between theoretical intractability and practical necessity for this canonical optimization problem.