Research on Knapsack Problem and its Algorithm

Peichi Li*

Stamford American International School Singapore, Singapore, 357684, Singapore

*Corresponding author's e-mail: lpts1011@gmail.com

Abstract. Resource allocation and constraint optimization problems are prevalent in real life and engineering, such as loading in logistics transportation, asset portfolio management in financial investment, and task scheduling in computer systems. The knapsack problem is an abstract model of these problems and is widely used to describe optimal choices under limited resource conditions. As one of the classic problems in combinatorial optimization, the knapsack problem not only has significant theoretical research value but also has wide applications in practice. This paper systematically introduces several typical knapsack problems, including the 0-1 knapsack problem, the complete knapsack problem, the two-dimensional knapsack problem. The paper primarily focuses on dynamic programming methods and analyzes the state transition equations and algorithmic approaches for each type of problem in detail. This research helps readers understand the different types of knapsack problems and their algorithmic frameworks, and provides a reference for further exploration of more efficient optimization methods.

Keywords: Knapsack problem, Dynamic programming, Optimization

1 Introduction

The Knapsack Problem is a classic problem in combinatorial optimization and dynamic programming, dating back to the mid-20th century. It was initially used by mathematicians and computer scientists to describe trade-offs in resource allocation. In its most basic form, the Knapsack Problem can be described as follows: given a knapsack with limited capacity and a set of items with weight and value, how can we select a set of items to maximize the total value without exceeding the knapsack's capacity? This simple and intuitive model can abstract many real-world optimization problems, such as cargo loading in logistics, budget allocation in project management, portfolio optimization in financial investment, and resource scheduling and task allocation in computer systems. However, the Knapsack Problem and its variations are often NP-complete, meaning that as the problem size increases, the computation time required for an exact solution grows exponentially, posing a significant challenge to algorithm design.

As research has deepened, it has been discovered that different real-world scenarios lead to various variations of the Knapsack Problem. For example, the 0-1 knapsack

J. Rak et al. (eds.), *Proceedings of the 2026 11th International Conference on Social Sciences and Economic Development (ICSSED 2026)*, Advances in Economics, Business and Management Research 391,
https://doi.org/10.2991/978-94-6239-701-9_40

problem requires each item to be chosen only once, while the unbounded knapsack problem allows items to be chosen an unlimited number of times; the two-dimensional knapsack problem introduces a second constraint besides weight (such as volume or time), while the multiple knapsack problem limits the quantity of each item; the grouped knapsack problem divides items into several groups, with each group choosing at most one item. These variations not only expand the application boundaries of the knapsack problem but also drive the development of corresponding algorithms, from classic dynamic programming to branch and bound methods, greedy heuristics, approximation algorithms, and the recently emerging AI-based hybrid optimization methods.

Studying the knapsack problem and its variations has significant academic and practical value. On the one hand, they are important touchstones for algorithm design and analysis, helping to understand core ideas such as dynamic programming, search strategies, pruning optimization, and approximation algorithms; on the other hand, different types of knapsack problems have wide applications in industry, finance, logistics, computer systems, and other fields, directly providing decision support for production scheduling, cost control, and resource allocation. Therefore, this paper aims to systematically analyze and compare common variations of the knapsack problem and their solution algorithms, including 0-1 knapsack, complete knapsack, two-dimensional knapsack, multiple knapsack, and grouped knapsack, and to explore their problem characteristics, algorithm principles, and performance differences, so as to provide a reference for subsequent optimization research and engineering practice.

2 Literature Review

Research on knapsack problems originated from the most fundamental one-dimensional form, among which the most representative types include the 0–1 knapsack problem and the unbounded (complete) knapsack problem. These early formulations laid the theoretical and methodological foundation for subsequent complex variants such as multidimensional, multiple, and grouped knapsack problems, and thus have received extensive attention in academia. As research progressed, scholars not only proposed various dynamic programming algorithms but also expanded the problem to address time complexity optimization and practical application scenarios. Based on existing research findings, this section reviews typical one-dimensional and multidimensional knapsack problems [1,2].

Abdel et al. (2022) [3] investigated the 0–1 knapsack problem and its multidimensional extensions, systematically analyzing dynamic programming–based methods for obtaining optimal solutions. They adopted a reverse-iteration dynamic programming algorithm, which ensures that each item is processed without redundant calculations while still achieving the maximum-value combination. Their work provides a clear algorithmic framework for understanding and implementing the 0–1 knapsack problem and establishes a basis for addressing multi-constraint problems. However, their method still suffers from high time complexity when dealing with large-scale item sets.

Beliakov et al. (2021) [4] studied the unbounded (complete) knapsack problem and proposed a forward-iteration dynamic programming approach that allows items to be selected repeatedly. This method is well suited for real-world scenarios in which items can be reused, such as warehouse replenishment and material procurement. Their findings show that this approach reduces redundant computation and improves efficiency while maintaining accuracy. Nevertheless, computation may still become intensive when dealing with a very large number of item types. Feng et al. (2022) [5] introduced a self-learning binary moth search algorithm for solving the 0–1 multidimensional knapsack problem. Li et al. (2024) [6] proposed an improved BQPSO algorithm for efficiently solving the 0–1 knapsack problem. Their improvements focus on three aspects: first, to address discretization challenges in quantum-behaved particle swarm optimization, they introduced a mapping strategy based on the average position of all particles; second, they used a transfer function to discretize the local attractor into a binary vector for subsequent crossover operations, guiding individuals in the discrete space toward the optimal region; additionally, they designed a new repair mechanism to handle infeasible solutions and implemented a diversity-maintenance strategy to alleviate premature convergence.

Building on research on one-dimensional knapsack problems (including both 0–1 and unbounded versions), scholars further extended the model to the two-dimensional knapsack problem, in which a second constraint (such as volume or time) is added in addition to the single capacity constraint. This extension enables the model to better reflect multi-resource optimization scenarios in real-world applications. To achieve this, researchers typically employ a two-dimensional dynamic programming array to maintain state transitions, allowing multiple constraints to be incorporated simultaneously during the computation.

Beliakov (2021) [7] examined the two-dimensional knapsack problem by introducing a second constraint (such as volume or time) alongside weight. They used a two-dimensional dynamic programming array to maintain states, enabling simultaneous consideration of multiple resource limitations. Their work expanded the application of knapsack problems in multi-resource scheduling, but the increase in dimensionality leads to larger space complexity and higher memory consumption.

Based on research into two-dimensional knapsack problems, scholars have also begun to explore multi-knapsack problems that more closely reflect practical applications. In many real-world scenarios, not only do multiple constraints exist, but multiple knapsacks or resource pools may also be available—for example, logistics companies must assign goods to multiple transport vehicles, or cloud computing platforms must schedule tasks across multiple servers. These situations have motivated researchers to propose multi-knapsack models that more accurately represent complex resource allocation tasks.

Christophe et al. (2021) [8] studied the multiple knapsack problem by restricting the quantity of each item to simulate practical scenarios. They used dynamic programming combined with binary splitting methods to solve the problem, enabling efficient processing of item selections with limited quantities. This approach is easy to understand and implement, but as the scale of available items increases, computation complexity remains an issue.

Overall, these studies focus on the design and application of dynamic programming algorithms, ensuring solution accuracy while providing practical methods for real-world use cases. However, large-scale or multi-constraint knapsack problems still face challenges related to time and space consumption, offering promising directions for future research on algorithm optimization and efficiency improvement.

3 Knapsack Problems and Their Algorithms

The knapsack problem is a classical model in combinatorial optimization. Its core idea is to select an optimal subset of items under a limited capacity constraint so that the total value is maximized. Owing to its simple yet highly representative formulation, the knapsack problem holds significant importance in theoretical computer science and has been widely applied in practical fields such as resource allocation, investment decision-making, and logistics management.

Over time, the knapsack problem has evolved into multiple variants. Although these variants differ in their constraint settings and objective functions, the fundamental concept remains the same: to strike a balance between limited resources and potential benefits in order to find the optimal solution. This section begins with the basic 0–1 knapsack problem, followed by the unbounded (complete) knapsack problem, the two-dimensional knapsack problem, providing detailed explanations of their algorithmic ideas and solution methods.

In concrete algorithmic implementations, a one-dimensional array dp (short for *Dynamic Programming*, representing the state array used in dynamic programming) is commonly employed to store intermediate results. Here, dp[w] denotes the maximum achievable value when the capacity is w. By continually updating the dp array, redundant computations can be avoided, thereby improving the efficiency of the algorithm.

3.1 0-1 Knapsack Problem

Among the many variants of the knapsack problem, the 0–1 knapsack problem is the most fundamental and classical form. In this problem, we are given a set of items, each with a fixed weight and value, and each item can be selected at most once. The objective is to maximize the total value of the selected items without exceeding the capacity of the knapsack. This model can simulate numerous real-world scenarios, such as airline baggage loading where certain items, due to their uniqueness or scarcity, cannot be carried multiple times; project investment decisions in which each project can only be chosen once; or task scheduling where a task can be assigned to only one time slot and cannot be repeated.

Since each item can be selected only once, the algorithmic design for this problem typically employs a reverse-order dynamic programming (DP) iteration to avoid counting the contribution of the same item more than once. As shown in Table 1.

Table 1. Pseudocode for 0-1 Knapsack Problem

Section	Content
Input	A set of items (each with a weight and a value), and a knapsack with a maximum capacity
Main Pseudocode Body	**1. Initialize an array to record the maximum value obtainable at each capacity, with all initial entries set to 0.** **2. Consider each item in sequence:** ○ **Iterate from the maximum capacity downward to the weight of the item;** ○ **For each capacity, compare the following two choices:** 1. **If the item is included:** new value = maximum value at (capacity − item's weight) + item's value; 2. **If the item is not included:** total value remains unchanged; ○ **Select the larger of the two values and update the corresponding position in the array.**
Output	The value in the array at the "maximum capacity" position represents the maximum total value achievable under that capacity. This result indicates that among all feasible choices, the algorithm continuously updates the array to record the optimal comparison between "including or not including" each item, ultimately obtaining the global optimal solution.

3.2 Complete Knapsack Problem

The complete knapsack problem allows each item to be selected an unlimited number of times, which commonly appears in real-world situations involving bulk purchasing or production. For example, in warehouse restocking, the same type of goods can be purchased repeatedly; in cargo loading, if an item is abundantly available (such as ore or grain), it can be loaded continuously; and in the coin change problem, each coin denomination can be used infinitely many times.

Unlike the 0–1 knapsack problem, the implementation of the state transition in the complete knapsack typically uses forward iteration, because items may be used multiple times, and there is no risk of incorrectly selecting the same item more than once within a single iteration. As shown in Table 2.

Table 2. Pseudocode for Complete Knapsack Problem

Section	Content
Input	**A set of items** (each item has a weight and a value), and **a knapsack with a maximum capacity.**
Main Pseudocode Body	**Initialize an array** to record the maximum value obtainable at each capacity, with all initial values set to **0**. Consider each item in sequence: • **Start from the item's weight and iterate upward** to the maximum capacity; • **For each capacity, compare two choices:** 1. **Include the item** (can be placed multiple times): new value = value in the array at (current capacity − item's weight) + value of the item; 2. **Do not include the item:** new value = original value; • **Select the option with the larger value** and update the **corresponding position** in the array.
Output	The value stored in the array at the **maximum capacity** position, which represents the **maximum total value** achievable under this capacity. This result indicates that, under the condition that each item may be used **unlimited times,** the algorithm obtains the **global optimal** solution by continuously comparing the choices of"placing" vs. "not placing" the item at each step.

3.3 Two-Dimensional 0-1 Knapsack Problem

The two-dimensional 0–1 knapsack problem introduces a second constraint (such as volume, time, or energy consumption) in addition to the single capacity constraint (e.g., weight), and each item can be selected at most once. This model is often used in scenarios requiring consideration of multiple resource limitations simultaneously. For example, unmanned aerial vehicle (UAV) transportation must account for both maximum payload weight and battery endurance; data center task scheduling must simultaneously consider processing capacity and energy consumption; construction planning must adhere to both material constraints and time limits.

The two-dimensional 0–1 knapsack problem belongs to the general family of 0–1 knapsack problems, but requires maintaining a two-dimensional dynamic programming state array, making the transition process dependent on the interaction of the two constraints. As shown in Table 3.

Table 3. Pseudocode for Two-Dimensional 0-1 Knapsack Problem

Section	Content
Input	A set of items (each with weight, volume, and value), and a knapsack with maximum weight capacity and maximum volume capacity.
Main Pseudocode Body	**Initialize a two-dimensional array** to record the maximum value under different weight and volume capacities, with all initial values set to **0**. **Consider each item in sequence:** ● First, **iterate downward from the maximum weight** to the item's weight; ● Meanwhile, **iterate downward from the maximum volume** to the item's volume; ● For each state, compare two choices: 1. **Include the item once** (because this is a 0–1 knapsack problem): new value = value in the array at (weight − item's weight, volume − item's volume) item's value; 2. **Do not include the item:** keep the original value; ● **Select the larger value** and update the corresponding position in the two-dimensional array.
Output	The value stored at the position (*maximum weight, maximum volume*) represents the maximum total value achievable under the two constraints. This result indicates that since each item can be selected at most once, the algorithm repeatedly updates the two-dimensional array until reaching the optimal solution under both constraints.

3.4 Two-Dimensional Complete Knapsack Problem

The two-dimensional complete knapsack problem differs from the two-dimensional 0–1 knapsack problem in that each item can be selected an unlimited number of times. In practice, this situation often appears in multi-resource scenarios involving bulk procurement. For example, industrial raw materials may need to be considered simultaneously in terms of warehouse storage capacity, yet raw materials can be purchased repeatedly; transportation scheduling may be constrained by vehicle load limits and

container volume restrictions, but goods can still be repeatedly loaded. In algorithmic implementation, the state transition typically adopts forward iteration over both dimensions, allowing the same item to be reused within a single iteration. As shown in Table 4.

Table 4. Pseudocode for Two-Dimensional Complete Knapsack Problem

Section	Content
Input	A set of items (each item has weight, volume, and value), and a knapsack with maximum weight capacity and maximum volume capacity.
Main Pseudocode Body	**Initialize a two-dimensional array** to record the maximum value under different weight and volume capacities, with all initial values set to **0**. **Consider each item in sequence:** ● **Iterate upward from the item's weight** to the maximum weight capacity; ● **Iterate upward from the item's volume** to the maximum volume capacity; ● For each state, compare the following two choices: 1. **Include the item** (can be selected multiple times): new value = value in the array at *(weight − item's weight, volume − item's volume)* ● *item's value;* 2. **Do not include the item:** keep the original value; ● **Select the better option** and update the corresponding position in the two-dimensional array.
Section	Content
Output	The value stored at the position **(maximum weight capacity, maximum volume capacity)** represents the maximum total value achievable under the two constraints. This result indicates that, under the condition that items can be used **unlimited times**, the algorithm achieves the global optimal solution by continuously updating the two-dimensional array.

4 Conclusions and Future Work

4.1 Conclusions

This paper systematically introduces the knapsack problem and its various classical variants, including the 0–1 knapsack problem, the complete knapsack problem, the two-dimensional 0–1 knapsack problem, the two-dimensional complete knapsack problem. For each type of problem, detailed explanations are provided regarding its background, constraints, and real-world application scenarios, such as logistics transportation, project investment, task scheduling, and resource allocation. Moreover, the corresponding dynamic programming principles, state transition equations, and pseudocode are presented to help readers clearly understand the core logic and implementation of each algorithm.

Through comparative analysis of different knapsack problem variants, this paper highlights their differences in constraint structures, selection strategies, and algorithmic complexities. It also explains why certain variants employ reverse-order iteration or forward iteration, and why two-dimensional state arrays are required in certain cases. These discussions not only deepen the understanding of the inherent nature of the knapsack problem but also demonstrate the universality and flexibility of dynamic programming in solving combinatorial optimization problems. Overall, this paper provides a systematic and multi-level learning framework for knapsack algorithms—from basic formulations to advanced variants—helping readers understand the connection between problem modeling, algorithm design, and real-world applications, thereby laying a solid foundation for further research and engineering practice.

4.2 Future Work

Looking ahead, future research on knapsack problems will involve both algorithmic optimization and the continual expansion of real-world application scenarios. For each type of knapsack problem, practical use cases can be naturally mapped—for example, the 0–1 knapsack problem can be applied to investment portfolio selection, where investors must choose projects under a limited budget to maximize returns; the complete knapsack problem can be mapped to coin change or warehouse replenishment scenarios where items can be selected repeatedly; the two-dimensional knapsack problem is suitable for situations such as three-dimensional packing or logistics transportation involving simultaneous constraints on weight and volume; the multi-knapsack problem corresponds to blockchain task allocation or multi-server resource scheduling; and the grouped knapsack problem has practical significance in corporate investment decisions, such as mutually exclusive investment choices among departments. These real-world cases further validate the effectiveness and applicability of the algorithms and strengthen the bridge between theoretical research and practical engineering.

Research on knapsack problems continues to hold broad development potential and application prospects. On the one hand, as problem sizes grow and constraint structures become more complex, traditional algorithms may encounter limitations in

computational efficiency and memory usage. Therefore, exploring more efficient solution strategies remains of significant value, including approximate algorithms, heuristic algorithms, branch-and-bound methods, and hybrid intelligent optimization techniques. On the other hand, with the rapid development of artificial intelligence and large language models, future research may incorporate intelligent approaches to enhance the solving capabilities and expand the application boundaries of knapsack-related optimization problems.

References

1. Christophe Wilbaut, et al. “The Knapsack Problem and Its Variants: Formulations and Solution Methods.” Springer EBooks, 1 Jan. 2022, pp. 105–151
2. D’Ambrosio, Ciriaco, et al. “The Knapsack Problem with Forfeit Sets.” Computers & Operations Research, vol. 151, 23 Nov. 2022, p. 106093, www.sciencedirect.com/science/article/abs/pii/S0305054822003239, .
3. Abdel-Basset, Mohamed, et al. “Binary Light Spectrum Optimizer for Knapsack Problems: An Improved Model.” Alexandria Engineering Journal, vol. 67, 9 Jan. 2023, pp. 609–632, www.sciencedirect.com/science/article/pii/S1110016822008080, .
4. Agrawal, Prachi, et al. “Solving Knapsack Problems Using a Binary Gaining Sharing Knowledge-Based Optimization Algorithm.” Complex & Intelligent Systems, 4 Apr. 2021, .
5. Feng, Yanhong, and Gai-Ge Wang. “A Binary Moth Search Algorithm Based on Self-Learning for Multidimensional Knapsack Problems.” Future Generation Computer Systems, vol. 126, Jan. 2022, pp. 48–64, . Accessed 26 May 2022.
6. Li, Xiaotong, et al. “An Improved Binary Quantum-Behaved Particle Swarm Optimization Algorithm for Knapsack Problems.” Information Sciences, vol. 648, 1 Nov. 2023, pp. 119529–119529, . Accessed 24 May 2024.
7. Beliakov, Gleb. “Knapsack Problems with Dependencies through Non-Additive Measures and Choquet Integral.” European Journal of Operational Research, vol. 301, no. 1, Aug. 2022, pp. 277–286, . Accessed 9 Apr. 2022.
8. Cacchiani, Valentina, et al. “Knapsack Problems - an Overview of Recent Advances. Part II: Multiple, Multidimensional, and Quadratic Knapsack Problems.” Computers & Operations Research, vol. 143, Feb. 2022, p. 105693