*Exploring the Knapsack Problem*

Kiera Conway   
Dakota State University  
Seattle, Washington  
Kiera.Conway@trojans.dsu.edu

*Abstract*—The Knapsack Problem (KP) has been a subject of fascination since its first appearance in 1895, when Mathews' first laid the groundwork for combinatorial optimization problems like the KP. The primary goal of the KP focuses on optimizing resource allocation by selecting the most valuable combination of items to fit within a knapsack's weight capacity. Over time, the KP has not only inspired the exploration of related problems like the Traveling Salesman Problem and the Graph Coloring Problem, it has also exploded into a vast field of KP variants that address the practical complexities of real world uses. Notable problem variations include the 0/1 KP, Bounded Knapsack Problem, and the Unbounded Knapsack Problem. This report focuses on two prominent solution variations for the 1/0 KP: the Extended Greedy Algorithm and Dynamic Programming Algorithm. As the KP remains a compelling and universal problem, researchers and mathematicians continue to delve into new problem variations and discover additional solution variants; the significance and impact of the Knapsack Problem continues to command attention and drive innovation. (*Abstract*)

Keywords—Knapsack Problem, Resource Allocation, Unbounded, Bounded, Greedy, Dynamic Programming, 0/1, combinatorial optimization

# Introduction

Imagine a daring thief who has found himself amidst a legendary treasure trove, equipped with just a worn knapsack on his back. As his eyes dart across the most enticing items in the room, he is sorely reminded of the limitations of his bag. He finds himself forced to make strategic decision about which treasures to take and which to leave behind as each item contains distinct weights and values. The imposition of his knapsack capacity has placed him in the precise predicament that challenges some of the greatest minds for generations.

This modified version of the classic Thief example from Bhargava demonstrates an intriguing optimization challenge, known as the Knapsack Problem (KP), that has tested mathematicians, computer scientists, and even fictional thieves throughout history [2]. At its core, the KP focuses on optimal resource allocation by framing it into one basic question: How can we maximize the value of the items in the knapsack without exceeding a weight limit?

# Problem Statement

The classic KP challenge pertains to the efficient packing of items into a knapsack with a limited weight capacity. The problem includes a knapsack with a predetermined weight capacity and a variety of items, each of which is characterized by a corresponding weight and value. It is assumed that the weight capacity of the knapsack is at least as large as the heaviest item, and allows inclusion of as many items as possible without exceeding the knapsack's weight limit. As the challenge’s primary focus is on distribution of the items, real world constraints such as dimensions and size are ignored [1].

While there are many variations of the KP, one of the most discussed and researched is known as the 0/1 KP. Unlike some variations which allow for inclusion of fractional items to maximize value, the 0/1 KP asserts that each item can be either be entirely included in the knapsack or entirely excluded. It is important to note that while we often allude to items going in a knapsack, mathematically, this problem simplifies down to simply a "combination of binary decisions" [6]. In other words, for each item in the knapsack, it is either ‘in the bag’ (1) or ‘not in the bag’ (0). Throughout this report, we will primarily focus on the 0/1 KP, henceforth referred to as KP, unless explicitly specified otherwise.

## Mathematical Representation

In the formal representation of the KP, a combination of a given set of items, characterized by their corresponding weight () and value (), are selected to place in the knapsack, such that the value is maximized while the weight does not exceed the knapsack's capacity (). This representation can be expressed as follows:

(1)

where

* represents the profit/value of item
* corresponds to the weight of item
* denotes the knapsack's maximum weight capacity
* is a binary decision variable that takes the value if item is included in the knapsack and otherwise.

In equation (1), the function represents the total value/profit achieved by selecting items with their corresponding profits () .The constraint ensures that the total weight of the selected items does not exceed the knapsack's capacity () by restricting the combination of items that can be placed in the knapsack. Since this problem pertains to the 1/0 KP variation of the knapsack problem, the binary variable is used to represent the selection () or exclusion () of items. This representation enables the optimization process to find the optimal combination of items that maximizes the total profit while adhering to the weight constraint [6].

## History

The KP has a long history, dating back centuries to its first appearance in 1895 [6]. Although it lacked an official name at the time, Mathews' work on Partition Theory introduced the underlying connections to combinatorial optimization problems like the Knapsack problem. Partition Theory’s focus on dividing or representing a number as a sum of positive integers introduced the concept of dividing the knapsack's capacity into parts (weights) to maximize the overall value[6][4]. Combinatorial optimization is a branch of mathematics which deals with finding the best solution from a limited selection and a finite set of possible combinations. However, the KP is just one of many problems in this category; other well-known combinatorial optimization problems include the Traveling Salesman Problem, the Vehicle Routing Problem, and the Graph Coloring Problem, to name a few [11]. This optimization concept set the stage for future developments in the KP.

In 1980, the introduction of the 0/1 variant of the KP by Gallo, Hammer, and Simeone further expanded the problem's scope. With this variant came the inclusion of decision variables (), which brought an additional layer of complexity to the KP and opened new avenues for research and exploration [9]. Since then, researchers have been working to improve the algorithm efficiency to various instances of the KP and its variants, resulting in significant advancements that can now solve "nearly all standard instances [of the KP] found in the literature" [7]. These advancements have played an important role in advancing the study and application of the KP.

## Importance

The KP serves as a powerful tool across the fields of mathematics and computer science, enabling the exploration algorithmic workings and efficiency [1]. Computer scientists can use the KP to help in program partitioning and task allocation, where it is essential to prioritize critical tasks while optimizing computational resources. Furthermore, the KP demonstrates versatility and utility for solving diverse challenges in day-to-day scenarios. The KP can be used to help efficiently pack food for survival situations, with a goal to maximize nutrition while minimizing overall weight or help assist investors in maximizing their profits while adhering to their budget constraints [5]. It is also invaluable to the lumber industry, where the KP contributes to the optimization of log cutting processes by maximizing the value obtained from each log while minimizing waste. In fact, while KP is often considered a ‘packing’ problem, it also commonly referred to as a ‘cutting problem [6].

Whether efficiently packing food for survival, optimizing investment portfolios, or even cutting logs, the versatility of the Knapsack is evident in many real-world applications. However, the real world often presents additional complexities that the basic KP does not account for. Factors including nutritional sustainability, investment risk tolerance, time constraints, market price variations, and other practical factors, require a different approach. As such, there appeared a substantial “need for [an] extension of the basic knapsack model,” which led to the development of “various extensions and variations,” with each variant offering a solution to tackle a specific scenario [6]. These extensions further enhance the KP’s applicability and make it invaluable across various industries and applications.

## Variants

There have been numerous variations created to tackle the 0/1 KP. Among the earliest and most well-known variants are the Unbounded Knapsack Problem (UNP) and the Bounded Knapsack Problem (BNP), both of which focus on the limitation of item repetition. In the UNP, items can be included in the knapsack an unlimited number of times. This allows greater flexibility in item selection and the potential to achieve a higher total value, but increases the number of possible solutions. Alternatively, the BNP limits the number of times each item can be placed inside the bag. Since each item appears an equal number of times, the primary focus is on the specific combination of the items. As a result, the solutions tend to be simpler and more realistic, but the number of possible solutions is limited [6].

Amidst the variations in the BNP, two popular solution variants for solving the 0/1 KP are the Greedy and Dynamic Programming algorithms. The Greedy approach focuses on locally optimal choices but may disregard the overall consequences of these choices, resulting in efficient but approximate solutions. In contrast, the Dynamic Programming variant breaks the problem into smaller subproblems and systematically calculates optimal solutions, ensuring a more accurate solution but with a potentially less-efficient running time.

### Greedy Algorithm: The Greedy Algorithm is characterized by its prioritization of immediate choices over evaluating multiple options to find the most optimal result [8]. While often considered naïve and heuristic due to its focus on local gains rather than long-term consequences, it stands out as an intuitive method for solving the KP. This is due to the seemingly obvious approach: it assigns a value-to-weight ratio for each item and fills the knapsack with items having the largest ratio until it reaches as close to the maximum weight as possible [6]. This simplicity makes it more efficient and easier to understand and implement compared to other methods like dynamic programming [8].

However, the Greedy Algorithm has its limitations. It may not always yield the most optimal solution compared to dynamic programming, as it does not consider all possible combinations of items [8]. As such, it should only be used for scenarios where "good enough" solutions are acceptable, such as finding the least amount of coins needed to return change or solving certain variants of the shortest path problem [6][8]. While it may not be “as broadly applicable as other algorithm design approaches [like] dynamic programming”, the Greedy Algorithm remains a valuable tool for solving specific instances of the Knapsack Problem efficiently and intuitively [8].

### Dynamic Programming Algorithm: The Dynamic Programming (DP) emerges as one of the best techniques for solving the challenging 0/1 KP [7]. It involves breaking a larger problem down into smaller subproblems and solving them iteratively. Instead of dealing with an entire problem at once, DP starts with the simplest cases and gradually builds up to the original problem. For the KP, DP operates in two main phases: the forward phase and the backtracking phase.

In the forward phase, DP employs the Bellman Recursion to compute optimal solution values for knapsack subproblems [10][6]. This process involves creating a grid of items and knapsack weights, starting from the smallest weight capacity and progressively moving through each capacity and item, until it reaches the capacity of the original knapsack [2]. By iteratively calculating the optimal solutions for smaller subproblems, DP efficiently determines the maximum value achievable at each weight capacity.

In the backtracking phase, DP starts from the final cell of the table generated during the forward phase and identifies the items that were included in each knapsack to achieve the maximum value. This step ensures that the algorithm provides not only the maximum value, but also which specific combination of items achieves that value.

The DP approach is known for its efficiency and ability to find the optimal solution for the 0/1 KP, even for large and complex instances of the problem, with a guaranteed worst-case running time [7]. It is considered one of the most effective techniques for solving the KP, providing a reliable and accurate solution that accounts for all possible item combinations.

# Algorithms

To better understand the previously discussed algorithm variations and how they work, we will analyze the pseudocode of both the Greedy Algorithm and the Dynamic Algorithm. By analyzing the step-by-step procedures of each approach, we can visualize their methods for solving the Knapsack Problem. This analysis will provide valuable insights into the strengths and limitations of each technique, with the goal to highlight the distinct features and performance characteristics of each algorithm, showcasing their significance in the realm of solving the KP.

## Extended Greedy Algorithm

Before observing the pseudocode of the Greedy Algorithm for KP, it is essential to address a special case which may prevent the standard Greedy Algorithm from returning an optimal solution [6]. Consider following values for our Knapsack instance:

Table I. Knapsack Instance

|  |  |  |
| --- | --- | --- |
| Number of items | |  |
| Knapsack capacity | |  |
| Item 1 | *weight* |  |
| *profit* |  |
| *efficiency* |  |
| Item 2 | *weight* |  |
| *profit* |  |
| *efficiency* |  |

Given the values in Table 1, the Greedy Algorithm selects Item 1 due to its higher efficiency () compared to Item 2 (). It then places Item 1 into the knapsack, leaving a remaining capacity of 2. As no additional items fit within this space, the function concludes that the knapsack is full. However, the optimal solution in this case would have been to pack Item 2, which has a value of 3, resulting in a higher profit.

To address this limitation, I have implemented Kellerer’s "Extended Greedy" algorithm. This enhanced version incorporates an additional step to account for the possibility of selecting a single item with the highest profit value. This is accomplished by comparing the solution value obtained from the standard Greedy Algorithm to the largest individual profit value, of which the larger value is added to the knapsack [6]. This modification will improve the algorithm's performance and ensure a more reliable and accurate approximation for the Knapsack problem.

The pseudocode for the Extended Greedy Algorithm is as follows:

Table II. Extended Greedy Algorithm Pseudocode

|  |
| --- |
| 1. Initialize total weight and profit variables to 0  2. Sort items by efficiency in decreasing order (highest first).  3. Iterate through the sorted items, and for each item:  - If adding item does not exceed the weight capacity:  - Include the item  - Update the total weight  - Update the total profit  - Else:  - skip adding the item to knapsack |

With the implementation of this special case, the Extended Greedy Algorithm aims to provide a promising alternative to the standard Greedy approach by offering improved results for certain instances of the KP.

## Dynamic Programming Algorithm

As previously mentioned, the Dynamic Programming approach solves the 0/1 Knapsack Problem by breaking it down into smaller subproblems and iteratively finding each optimal solution. To illustrate the steps of the Dynamic Programming algorithm variant, consider the following pseudocode:

|  |
| --- |
| 1. Initialization  - Initialize total weight and profit variables to 0  - Create DP array[n+1, capacity+1] and initialize all cells to 0  *Note: The DP array will contain the Bellman Recursion Table*  2. Bellman Recursion:  - Iterate from 1 to n, For each item:  - Iterate from 0 to capacity, For each sub-capacity:  - If Adding Current Item Exceeds Capacity  - Set cell to the previous best solution (cell above)  - Else  - Set current cell to max(cell above current cell, best value for remaining capacity) |
| 3. Reconstruction:  - Initialize empty list to store knapsack items  - While able to reverse iterate through DP table:  - If current cell is different from above cell:  - Include item in bag  - Update knapsack weight  - Continue to next item |

The Dynamic Programming algorithm considers all possible combinations of items to achieve the maximum profit while adhering to the knapsack's weight capacity. Through its iterative nature and efficient subproblem solving, DP guarantees the most optimal solution, making it a powerful algorithm for solving the 0/1 Knapsack Problem [6].

# Time Complexity

Now

## Equations

The equations are an exception to the prescribed specifications of this template. You will need to determine whether or not your equation should be typed using either the Times New Roman or the Symbol font (please no other font). To create multileveled equations, it may be necessary to treat the equation as a graphic and insert it into the text after your paper is styled.

Number equations consecutively. Equation numbers, within parentheses, are to position flush right, as in (1), using a right tab stop. To make your equations more compact, you may use the solidus ( / ), the exp function, or appropriate exponents. Italicize Roman symbols for quantities and variables, but not Greek symbols. Use a long dash rather than a hyphen for a minus sign. Punctuate equations with commas or periods when they are part of a sentence, as in:

*a**b* 

Note that the equation is centered using a center tab stop. Be sure that the symbols in your equation have been defined before or immediately following the equation. Use “(1)”, not “Eq. (1)” or “equation (1)”, except at the beginning of a sentence: “Equation (1) is . . .”

## Some Common Mistakes

* The word “data” is plural, not singular.
* The subscript for the permeability of vacuum **0, and other common scientific constants, is zero with subscript formatting, not a lowercase letter “o”.
* In American English, commas, semicolons, periods, question and exclamation marks are located within quotation marks only when a complete thought or name is cited, such as a title or full quotation. When quotation marks are used, instead of a bold or italic typeface, to highlight a word or phrase, punctuation should appear outside of the quotation marks. A parenthetical phrase or statement at the end of a sentence is punctuated outside of the closing parenthesis (like this). (A parenthetical sentence is punctuated within the parentheses.)
* A graph within a graph is an “inset”, not an “insert”. The word alternatively is preferred to the word “alternately” (unless you really mean something that alternates).
* Do not use the word “essentially” to mean “approximately” or “effectively”.
* In your paper title, if the words “that uses” can accurately replace the word “using”, capitalize the “u”; if not, keep using lower-cased.
* Be aware of the different meanings of the homophones “affect” and “effect”, “complement” and “compliment”, “discreet” and “discrete”, “principal” and “principle”.
* Do not confuse “imply” and “infer”.
* The prefix “non” is not a word; it should be joined to the word it modifies, usually without a hyphen.
* There is no period after the “et” in the Latin abbreviation “et al.”.
* The abbreviation “i.e.” means “that is”, and the abbreviation “e.g.” means “for example”.

An excellent style manual for science writers is [7].

# Using the Template

After the text edit has been completed, the paper is ready for the template. Duplicate the template file by using the Save As command, and use the naming convention prescribed by your conference for the name of your paper. In this newly created file, highlight all of the contents and import your prepared text file. You are now ready to style your paper; use the scroll down window on the left of the MS Word Formatting toolbar.

## Authors and Affiliations

**The template is designed for, but not limited to, six authors.** A minimum of one author is required for all conference articles. Author names should be listed starting from left to right and then moving down to the next line. This is the author sequence that will be used in future citations and by indexing services. Names should not be listed in columns nor group by affiliation. Please keep your affiliations as succinct as possible (for example, do not differentiate among departments of the same organization).

### For papers with more than six authors: Add author names horizontally, moving to a third row if needed for more than 8 authors.

### For papers with less than six authors: To change the default, adjust the template as follows.

#### Selection: Highlight all author and affiliation lines.

#### Change number of columns: Select the Columns icon from the MS Word Standard toolbar and then select the correct number of columns from the selection palette.

#### Deletion: Delete the author and affiliation lines for the extra authors.

## Identify the Headings

Headings, or heads, are organizational devices that guide the reader through your paper. There are two types: component heads and text heads.

Component heads identify the different components of your paper and are not topically subordinate to each other. Examples include Acknowledgments and References and, for these, the correct style to use is “Heading 5”. Use “figure caption” for your Figure captions, and “table head” for your table title. Run-in heads, such as “Abstract”, will require you to apply a style (in this case, italic) in addition to the style provided by the drop down menu to differentiate the head from the text.

Text heads organize the topics on a relational, hierarchical basis. For example, the paper title is the primary text head because all subsequent material relates and elaborates on this one topic. If there are two or more sub-topics, the next level head (uppercase Roman numerals) should be used and, conversely, if there are not at least two sub-topics, then no subheads should be introduced. Styles named “Heading 1”, “Heading 2”, “Heading 3”, and “Heading 4” are prescribed.

## Figures and Tables

#### Positioning Figures and Tables: Place figures and tables at the top and bottom of columns. Avoid placing them in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert figures and tables after they are cited in the text. Use the abbreviation “Fig. 1”, even at the beginning of a sentence.

1. Table Type Styles

| Table Head | Table Column Head | | |
| --- | --- | --- | --- |
| Table column subhead | Subhead | Subhead |
| copy | More table copya |  |  |

1. Sample of a Table footnote. (*Table footnote*)
2. Example of a figure caption. (*figure caption*)

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an example, write the quantity “Magnetization”, or “Magnetization, M”, not just “M”. If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write “Magnetization (A/m)” or “Magnetization {A[m(1)]}”, not just “A/m”. Do not label axes with a ratio of quantities and units. For example, write “Temperature (K)”, not “Temperature/K”.

# References

|  |  |
| --- | --- |
| [1] | R. Hurbans, Grokking Artificial Intelligence Algorithms, Shelter Island, NY: Manning Publications Co, 2020. |
| [2] | A. Bhargava, Grokking Algorithms, Shelter Island, NY: Manning Publications Co, 2016. |
| [3] | G. Mathews, "On the Partition of Numbers," *Proceedings of the London Mathematical Society,* June 1897. |
| [4] | M. La Rocca, Advanced Algorithms and Data Structures, Shelter Island, NY: Manning Publications Co, 2021. |
| [5] | D. Pisinger, "Where are the hard knapsack problems?," Copenhagen, 2003. |
| [6] | H. Kellerer, U. Pferschy and D. Pisinger, Knapsack Problems, Berlin: Springer, 2004. |
| [7] | D. Zingaro, Algorithmic Thinking, San Francisco, CA: No Starch Press, 2020. |
| [8] | A. Billionnet and F. Calmels, "Linear programming for the 0–1 quadratic knapsack problem," *European Journal of Operational Research,* vol. 92, no. 2, pp. 310-325, 19 July 1996. |
| [9] | R. Andonov, V. Poirriez and S. Rajopadhye, "Unbounded knapsack problem: Dynamic programming revisited," *European Journal of Operational Research,* vol. 123, no. 2, pp. 394-407, June 2000. |
| [10] | Q. Wang, K. H. Lai and C. Tang, "Solving combinatorial optimization problems over graphs with BERT-Based Deep Reinforcement Learning," *Information Sciences,* vol. 619, pp. 930-946, January 2023. |