**UNLOCK THE POWER OF DYNAMIC PROGRAMMING**

**USING 0/1 KNAPSACK**

**Dr S. Suriya1, Anne Merin Mathew2**

1Associate Professor, Department of Computer Science and Engineering, PSG College of Technology, Coimbatore, Tamil Nadu, India.

[suriyas84@gmail.com,](mailto:suriyas84@gmail.com,%20) [ss.cse@psgtech.ac.in](mailto:ss.cse@psgtech.ac.in)

2PG Scholar, Department of Computer Science and Engineering, PSG College of Technology, Coimbatore, Tamil Nadu, India

[23mz31@psgtech.ac.in](mailto:23mz31@psgtech.ac.in)

**ABSTRACT**

Dynamic Programming offers a powerful strategy for solving the 0/1 Knapsack Problem, which revolves around selecting items based on their values and weights to maximize overall value without surpassing a designated weight limit. By constructing a dynamic programming table to store optimal solutions, this method proves instrumental in various practical scenarios like project scheduling and financial portfolio optimization. Mastering dynamic programming for the Knapsack Problem equips individuals to tackle diverse optimization challenges efficiently and effectively.

**Keywords:** Dynamic program, Knapsack, Optimal solution.

1. **INTRODUCTION**

**A. Dynamic Programming**

An algorithm design approach for optimization issues, which frequently include minimizing or maximizing, is dynamic programming. An optimization issue is divided into smaller subproblems, and the solutions to each are stored such that each subproblem need only be solved once. With this method, temporal complexities are reduced from exponential to polynomial.

In two different ways we can store our values in which subproblem values can be stored such as:

• Bottom-Up approach – Begins with smallest subproblems and builds up to original program through iterative calculation.

• Top-Down approach – Begins with main problem and breaks into smaller subproblems through recursive method.

1. **0/1 Knapsack Problem**

In mathematics and computer science, the 0/1 Knapsack problem is an optimization problem where you have a set of objects, each with weight, and a value with a restricted weight capacity. The objective is to choose the goods to include in a way that maximizes their combined worth and to ensure that the overall weight of the items chosen does not exceed the knapsack's capacity. The value of xi in 0/1 Knapsack can be either fully or none at all, with no items being prohibited.

**Manual Tracing of 0/1 Knapsack problem**

Given N items, each with weight and profit. You have a bag with a maximum weight capacity of W. Goal is to select some items to put into the bag in such a way that their combined profits are maximized while ensuring that total weight of selected items not exceeding the bag capacity by following some steps.

**Example:**

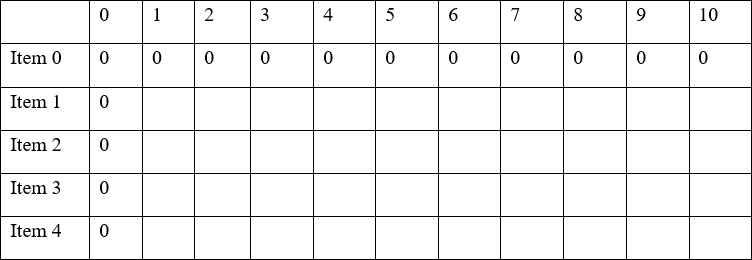
Here are the following items given are: Item 1: Weight =2, Value = 10

Item 2: Weight =3, Value = 5

Item 3: Weight =5, Value = 15

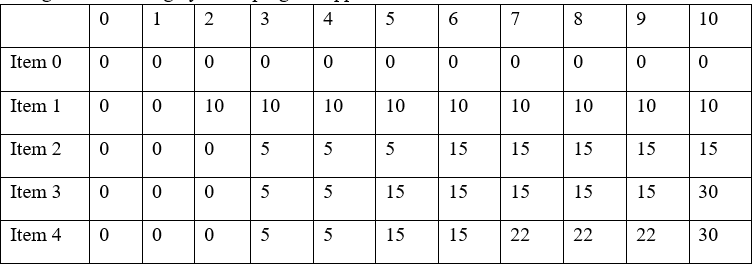
Item 4: Weight =7, Value = 7

The maximum weight capacity given is 10 .

** Step1:** Create a table where the rows represent the items (0 to 4) and columns represent the knapsack’s weight capacity (0 to 10).

**Figure-1**

**Step 2:** Filling the table using dynamic program approach.



**Figure-2**

**For item 1 (weight = 2, value = 10):**

If knapsack’s capacity is less than 2, the value can’t include in item 1, So the value remains 0.For the other knapsack capacities 2,3,4,5,6,7,8,9,10 the item 1 with a value of 10 can be included so the value remains 10 as there is no other item to consider at this point.

**For Item 2(weight = 3 , value = 5):**

If knapsack’s capacity is less than 3, the value can’t include item 2,so value remains 0.For other knapsack capacities 3,4,5, if capacity is 3 you can include item 2 with value 5. For the capacities the value remains 5.

**For Item 3 (weight = 5, value = 15):**

If Knapsack’s capacity is less than 5, you can’t include item 3, so value remains 0.For other capacities 5,6,7,8,9,10, If capacity is 5 you can include item 3 with value of 15. For other capacities the value remains 15.

**For item 4(weight = 7, value = 7):**

If knapsack’s capacity is less than 7, you can’t include item 4, so value remains 0.For other capacities of 7,8,9,10, if capacity is 7 you can include item 4 with a value of 7. For other capacities the value remains 7.

**Step 3:**

To Determine the optimal solution from the table. The value in the bottom right side of table represents the maximum value that can be obtained with Knapsack’s Capacity of 10. In this case the maximum value is 30.

**Step 4:**

Backtrack to find the optimal solution where we move from DP[4][10] and move from backward.

* Since DP [4][10] is equal to DP [3][10], it means that item 4 was not included in the optimal solution.
* We move to DP [3][10], which is equal to DP [3][10], the same value, so item 3 was not included.
* Next DP [3][10], which is equal to DP [2][10], so item 2 was not included.
* At last DP [2][10]is not equal to DP [1][10], which means that item 1 was included in optimal solution.

So, the optimal solution is to included item 1 in Knapsack resulting in total value of 30.

1. **LITERATURE SURVEY**

# **Related Works – Dynamic Programming**

In paper [1], gives a thorough summary of research articles that solve deterministic machine scheduling challenges using dynamic programming and its variants. This paper discusses the applicability of dynamic programming and its variants as problem- solving techniques for deterministic machine scheduling.

In paper [2], It is mostly concerned with discrete time step dynamic programming problems, both stochastic and deterministic. Bellman's equation and the principle of optimality are first compared. This paper's application is the management of multiple apps at once.

In Paper [3], provides a dynamic programming method for figuring out the most illuminating measurement order under various conditions. Gaussian processes and Markov decision processes are compared. This paper applies to autonomous robotics as well as security and surveillance.

In paper [4], outlines a few common issues, including the optimum scheme problem and the shortest path problem. Effectiveness and dependability are contrasted. the use of dynamic programming to solve environmental concerns related to system optimization.

In Paper [5], investigates the theoretical underpinnings and historical context of dynamic programming. The consistency solution and the inconsistency solution are compared. One of the applications is backward induction, which is based on Bellman's research.

In paper [6], a methodological paradigm for applying dynamic programming to large-scale stochastic control problems. A comparison is shown between a reasonably straightforward approximation architecture and feature extraction. The applications shown here illustrate the drawbacks of some straightforward methods by showing the challenges in integrating concise representations.

In paper [7], around three of these sequencing issues. The traveling salesman problem and describing the IBM 7090 program's computational experience are compared. Applications such as the assembly line balance problem, the traveling salesman problem, and arbitrary cost functions are included here.

In Paper [8], A strategy for dealing with the problems caused by the curse of dimensionality while handling large-scale stochastic problems. The quality of approximation is compared with the state relevance weights Applications of this include queuing network control experimental results, which offer methodological support from empirical data.

In Paper [9], the dynamic programming surveys using the proper approach and relevant process control issue. Neuro-Dynamic Programming (NDP) and Reinforcement Learning (RL) are compared. Approximate dynamic programming methods, specifically Reinforcement Learning and Neuro- Dynamic Programming, are used to process control issues.

In Paper [10], methods for approximate dynamic programming for the single-vehicle routing problem with stochastic needs. With a random base sequence, comparisons between two-step lookahead rollouts were made. This program is built on effective rollout techniques.

In paper [11], explains how to enhance the dynamic programming-related model predictive control (MPC) methodology. Stochastic constrained nonlinear and linear systems are compared. This paper's applications include the efficient ADP method.

In Paper [12], explains how the idea of dynamic programming can be used to the resolution of different kinds of boundary value issues. The dynamic programming approach is compared with a family of problems and a single problem with a fixed time. This article has several applications, ranging from basic optimization problems to stochastic problems, variational problem calculus, and differential equation numerical solution.

In paper [13], explains how to approach dynamic programming (ADP) using mathematics. The use of time scales calculus in ADP and its possible importance in diverse applications are compared. In addition to telecommunications, robotics and autonomous systems are employed.

In paper [14], explains the stochastic dynamic programming model, which uses the best estimate of the input for the current period. Applications such as Transferrable Knowledge and Stochastic Dynamic Programming are utilized.

In Paper [15], discusses the introduction to multistage decision problems by Richard Bellman and the rationale for the term "dynamic programming" Bellman's interest in multistage decision issues is the basis for the applications.

In Paper [16], best strategies for dynamic programming problems with discounts. Existing semi-Markov ϵ-optimal plans and issues with no known solution for positive dynamic programming are compared. A generalization of Dubbin’s and Savage's selection theorem serves as the main instrument in this paper's applications.

In paper [17], Mathematical techniques like linear programming can be used to solve optimization problems and determine the largest benefits or lowest costs. The applications utilized in this work include the condition of the system at time step t, which offers enough details to forecast the system's states at that point.

In paper [18], regarding the step-by-step explanations of the statistical dynamic programming method used in system design. The applications employed in this study aim to improve and maximize the system's overall utilization beforehand.

In Paper [19] offers an overview of minimum spanning tree techniques and conducted an empirical investigation. In combinatorial optimization, one of the most common and well-known issues is the minimum-weight spanning tree problem.

In paper [20], regarding the investigation of the integer multiple criteria knapsack issue and suggest methods based on dynamic programming. Other, more complicated model issues beyond the knapsack and dynamic method problems are compared. The binary multiple criteria knapsack problem with multiple constraints and multiperiod time dependent models is one of the applications of this work.

In Paper [21], Present a novel category of algorithms that can be used to resolve a variety of optimization issues, such as determining the best storage strategy. Every deliberative choice is a scheduling issue with a limited temporal horizon.

**Related Works –0/1 Knapsack Problem**

In paper [22], the cooperative evolutionary method for solving the bi-objective quadratic multiple knapsack problem, an NP-Hard combinatorial optimization problem. It begins by employing the constraint operator-based approach to create an initial approximation of the Pareto front.

In Paper [23], gives Numerous real-world applications require handling of inaccurate data. The paper suggests expanding the standard solution to allow for numerous instances of the same object in the solution with knapsack and to determine whether an acceptable degree of profit can be obtained.

In paper [24], reports on the successful application of ant colony optimization to solve the 0/1 knapsack problem. Tests are conducted on examples to show how effective the suggested algorithm is.

In Paper [25], presents a deep learning and neural network based heuristic solver for the knapsack problem. When there is a correlation between the values and weights of the items, the solver neural model outperforms the greedy approach in obtaining nearly optimal solutions. This ability to generalize was evaluated on instances including up to 200 items.

In Paper [26], Analyze how three different evolutionary multi-objective optimization (EMO) algorithm classes behave when dealing with numerous objective knapsack issues. Coefficients are specified at random to create test problems.

In paper [27], A project to solve the 0-1 Knapsack Problem via a genetic algorithm. A combinational optimization problem is the knapsack problem. The selection functions employed in this case are Roulette-Wheel, Tournament Selection, and Stochastic Selection. The resulting populations are examined for fitness value in an effort to find the right answer, and the anticipated outcomes were noted.

In Paper [28], The traditional approach to solving the Knapsack Problem was enhanced, and an algorithm for automatically creating paper based on the KP was introduced. Paper prioritizes the necessary requirements, classifies those goal parameters, and relaxes the restrictions on the parameters of generating paper.

In paper [29], emphasizes the topic's theoretical and practical applications while discussing the many Knapsack problem kinds. In order to avoid going over a specified maximum limit L, KP aims to choose a portion of the available goods with the maximum total weight.

In Paper [30], is to examine several paradigms for algorithm creation applied to the 0/1 Knapsack Problem. The goal of the Knapsack Problem is to maximize the usefulness of the items within a knapsack while staying within its carrying capacity. It is a combinatorial optimization problem. It is an NP-complete problem that can be solved with heuristic and exact methods.

1. **ALGORITHM**

The dynamic programming approach for solving the 0/1 Knapsack Problem involves initializing a 2D table to store intermediate results, with rows representing items and columns representing the knapsack capacity. The base cases are set, indicating that the maximum value for no items or zero capacity is zero. The algorithm iterates through each item and capacity, calculating the optimal value for each subproblem by considering whether to include the current item. The recurrence relation is established based on the comparison of weights and values. If the weight of the current item is less than or equal to the current capacity, the maximum value is determined by either including or excluding the item. If the weight exceeds the capacity, the maximum value remains the same as the value obtained without including the current item. By iteratively filling the table in a bottom-up manner, the algorithm efficiently computes the maximum value achievable with the given items and knapsack capacity. The final result is found in the bottom-right cell of the table, representing the optimal solution to the 0/1 Knapsack Problem.

1. **APPLICATIONS OF DYNAMIC PROGRAMMING**

# **Basic applications of dynamic programming**

# **Coin Change Problem:**

Dynamic programming is employed to solve the coin change problem, where the goal is to determine the minimum number of coins needed to make a specific amount using a given set of coin denominations. The Dynamic programming approach considers each coin denomination and calculates the minimum number of coins required to make different amounts, eventually solving the problem optimally.

# **Shortest Path Problems:**

Dynamic programming is applied to solve problems like finding the shortest path in a graph, for instance, using Dijkstra's algorithm or the Bellman Ford algorithm. It works by breaking down the problem into subproblems and calculating the shortest path from the source to each node in the graph. This dynamic programming approach optimizes the overall computation of shortest paths, particularly in scenarios like route planning and network optimization.

# **Fibonacci Sequence:**

The Fibonacci sequence is a classic example of dynamic programming. Dynamic programming is used to efficiently calculate the nth Fibonacci number by avoiding redundant calculations. Instead of calculating Fibonacci numbers from scratch, dynamic programming stores previously computed Fibonacci numbers and reuses them to compute larger Fibonacci numbers, reducing the time complexity from exponential to linear.

1. **Research related applications for dynamic programming**

**Genome Sequence Alignment:**

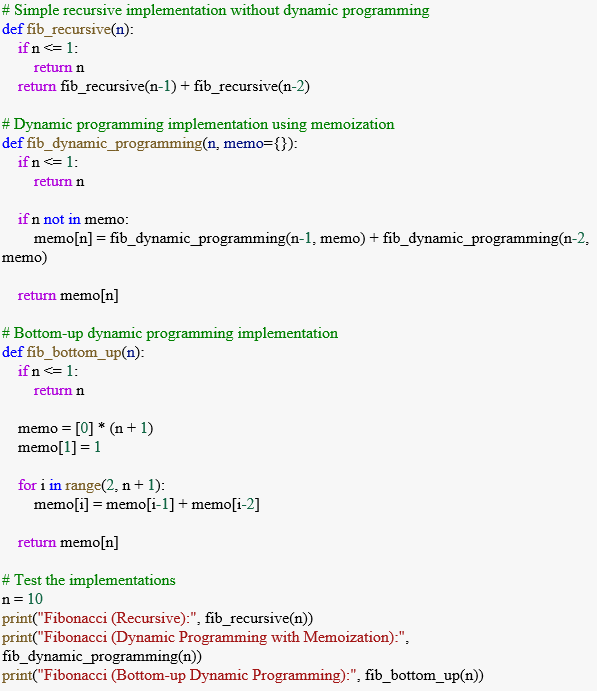
Dynamic programming is used in bioinformatics to align DNA and protein sequences for the purpose of identifying genetic similarities and differences. Researchers develop advanced algorithms and heuristics to handle large-scale sequence alignment efficiently and accurately.

**Optimal Control in Robotics:**

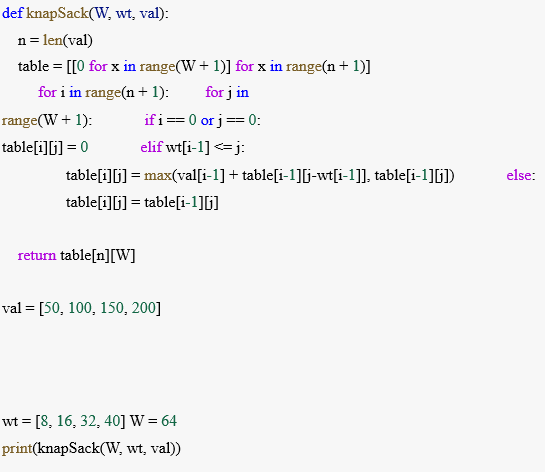
Developing control algorithms for robots and autonomous systems that involve making dynamic decisions in real-time, taking into account complex, changing environments. Dynamic programming helps in optimizing control policies for tasks like path planning and obstacle avoidance.

**Wireless Sensor Network Optimization:**

In the context of wireless sensor networks, researchers may use the 0/1 Knapsack Problem to optimize the selection of sensors for deployment. Each sensor has associated costs (weight) and benefits (value), and the objective is to maximize the information gathered while staying within budget constraints.

1. **EXPERIMENTAL RESULTS - AND DISCUSSION – 0/1 KNAPSACK PROBLEM**
2. **dynamic programming using python program:**

**OUTPUT**

1. **implementation of 0/1 knapsack problem:**

**OUTPUT**

# **Complexity analysis of dynamic programming using 0 /1 knapsack**

The time complexity of the bottom-up dynamic programming approach is O(n \* capacity), where 'n' is the number of items and 'capacity' is the capacity of the knapsack. This is because we fill up a 2D table of size (n+1) x (capacity+1) in a nested loop, and each cell takes constant time to compute.

The space complexity is also O(n \* capacity) because we use a 2D array (dp) of size (n+1) x (capacity+1) to store the intermediate results. This space is used to memorize the solutions to subproblems, allowing us to avoid redundant calculations. If memory optimization is a concern, it's possible to reduce the space complexity to O(capacity) by using a 1D array since we only need the results of the previous row during computation.

1. **CONCLUSION**

To sum up, the 0/1 Knapsack Problem is an important optimization problem having many applications in computer science, mathematics, and real-world decision-making situations. We can effectively tackle this problem by dividing it into smaller subproblems and storing the best solutions thanks to dynamic programming. Dynamic programming allows us to effectively address a variety of optimization problems and minimize temporal complexity, regardless of whether the Bottom-Up or Top-Down technique is used. It also provides individuals with useful skills that they can apply in real-world scenarios, such as project scheduling, financial portfolio optimization, and resource allocation.

1. **REFERENCES**
2. Algorithms for dynamic programming and their uses in machine scheduling Edson Marcelo Seido Nagano Journal of ELSIEVER, Volume 190, 116180; Antônio Gonçalves de Souza.
3. Fundamental Ideas and Uses of Dynamic Programming, Conference paper by K. Neumann, published in Springler, pages 31–56.
4. An Algorithm for Dynamic Programming to Determine the Best Order of Informative Measurements Ka Wai Cheung and Peter N. Loxley, Entropy 2023, 25(2), 251.
5. Using dynamic programming to optimize systems for environmental problems Fuxiang Weic, Baoyou Liub, and Jingjing Zhao a (ICSEM-13)
6. John Rust, New Palgrave Dictionary of Economics, "Dynamic Programming for Stationary, Markovian, Infinite Horizon Problems."
7. Methods Based on Features for Extensive Dynamic Programming Benjamin Van Roy & John N. Tsitsi Klis, 59–94
8. "A Dynamic Programming Approach to Sequencing Problems," by Richard M. Karp and Michael Held, published in Conférence paper Vol. 10.
9. Using Linear Programming to Estimate Dynamic Programming De Farias and Van Roy, D. P., Vol. 51, No. 6, Published Online: December 1, 2003
10. A review and recommendations for further work on approximate dynamic programming strategies and their suitability for process control Jong Min Lee and Jay H. Lee, International Journal of Automation, Control, and Systems, 2(3), 263-278, vol.
11. A truck routing issue with stochastic needs using approximation dynamic programming Robert Storer and Clara Novoa, Volume 196, Issue 2, July 16, 2009, Pages 509–515.
12. Process control using an approximate dynamic programming technique Volume 20, Issue 9, October 2010, Pages 1038-1048, Jay H. Lee and Weechin Wong
13. Dynamic programming using functional equations”. E. Stanley Lee and Richard Bellman, Aequationes Mathematicae 17 1-18
14. John Seiffertt, Suman Sanyal, and Donald C. Wagen, "Hamilton-Jacobi-Bellman Equations and Approximate Dynamic Programming on Time Scales," IEEE Transactions on Systems, Man, and Cybernetics, Part B Cybernetics Volume: 38.
15. Jery R. Stedinger, Bola F. Sule, and Daniel P. Loucks, "Stochastic dynamic programming models for reservoir operation optimization," Volume 20, Issue 11, Pages 1499–1505, 2021.
16. Stuart Dreyfus, "Richard Bellman on the Birth of Dynamic Programming," Operations Research 50(1):48–51.
17. F. S. VanVleck, T. Parthasarathy, C. J. Himmelberg, and "Optimal Plans for Dynamic Programming Problems," Vol. No. 1, 4.
18. Simplifying complex decisions: an introduction to stochastic dynamic programming Eric Marboutin and Oliver Gimenez, September 2013, Volume 4, Issue 9, pages 872–884.
19. Garud N. Iyengar, "Robust Dynamic Programming," Vol. 30, No. 2. Posted online on May 1, 2005.
20. Reduced Dynamic Scheduling pp. 226-235 in David Blackwell, Vol. 36, No. 1.
21. Approximate Dynamic Programming using Linear Programming, Vol. 51, No. 6, D. P. de Farias and B. Van Roy, published online December 1, 2003.
22. Quick dynamic programming applied to storage scheduling Robin Girard, RTE; Robin.girard@mines-paristech.fr; Vincent Barbesant, Center PERSEE of MINES ParisTech.
23. Learning methods' impact on an evolutionary approach: the biobjective quadratic multiple knapsack problem case study Oussama Gacem, Méziane Aïder, pp. 1183–12 09 (2023).
24. A research report on using imprecise data to solve the 0-1 Knapsack problem,
25. 2017 International Conference on Computer Communication and Informatics January 5– Jayashree Padmanabhan, Swagath S.
26. A Novel Method for Resolving the 0/1 Knapsack DilemmaZne-Jung Lee and Chou-Yuvan Lee International Conference on Informatics, Man, and Cybernetics, IEEE, 2006.
27. Neural Knapsack: An approach to solving the Knapsack Problem based on neural networks, A. A. HAZEM, KHALID ALNOWIBET, ACCESS.2020.3044005, IEEE Access.
28. Evolutionary Algorithms' Behavior in Multiobjective on Many-Objective Knapsack Problems, IEEE Fellow Hisao Ishibuchi, Member Yusuke Nojima, and Fellow Naoya Akedo. Vol. 19, No. 2.In 2014, Dutta, S., Shankar, H., Patra, D., and Alok Verma.
29. Using Genetic Algorithms to Solve the 0-1 Knapsack Problem, Rattan Preet Singh, University School of Information Technology, IEEE access.
30. Online Paper Generation Algorithm Using the Knapsack Problem, Peiguang Lin, Mei Sun, 2008 International Conference on Future Information Technology.
31. Lebanese American University online conference access, Maram Assi and Ramzi A. Haraty, A Survey of the Knapsack Problem.
32. In the 2012 Second International Conference on Advanced Computing & Communication Technologies, Ritika Mahajan and Sarvesh Chopra presented an analysis of the 0/1 Knapsack Problem utilizing deterministic and probabilistic techniques.