Dim Sum Mania: A Knapsack-Based Dynamic Programming Approach for Optimizing Dim Sum Variety in Group Dining

Varistha Devi - 13524135
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: varistha.devi@gmail.com , 13524135@std.stei.itb.ac.id

***Abstract*—Optimization problems can often be found in everyday situations, including group dining scenarios where people seek to maximize food variety while adhering to budgetary and dietary constraints. This paper presents a Dynamic Programming-based approach for optimizing dim sum variety in group dining by modeling the problem as a modified 0/1 Knapsack Problem. In the proposed formulation, each dim sum variety is treated as a decision item, while menu prices are adjusted through an effective cost calculation that considers both group size and serving requirements. Additional constraints such as food allergies and dietary preferences are handled through a preprocessing stage that filters infeasible menu items prior to optimization. A Python-based implementation was developed to evaluate the proposed method using a predefined dataset derived from a publicly available dim sum menu. Experimental results demonstrate that the approach successfully maximizes the number of distinct dim sum varieties while satisfying budget limitations, serving requirements, and dietary restrictions. Furthermore, the system is capable of correctly identifying situations in which no feasible solution exists. The findings indicate that Dynamic Programming provides an effective and practical solution for combinatorial optimization problems encountered in everyday group decision-making scenarios.**

Keywords—Dynamic Programming, 0/1 Knapsack Problem, Combinatorial Optimization, Menu Optimization, Group Dining

I. Introduction

Various problems, including optimization ones, can be found in many aspects of everyday life where people often don't realize that they can be approached using concepts from computer science and mathematics. One of those examples can be observed in group dining situations. Let's consider a group of friends who've decided that they want to dine at a restaurant that specializes in dim sums. With a limited budget, the group may wish to try as any varieties of dim sum as possible while ensuring that every person can taste each of the items selected. At the same time, however, dietary restrictions and food allergies must also be considered in the decision-making. Determining the best combination of menu items to order under these situations can quickly become challenging.

The scenario described above can be seen as a combinatorial optimization problem, where the objective is to maximize the dim sum variety selected while satisfying a few constraints, including a fixed budget, serving quantity requirements, and dietary restrictions among the group. Since each menu item can either be chosen or not, and each selected item contributes to the overall item variety while consuming a portion of the available budget, this problem can be modeled as a variation of the Knapsack Problem. In order to ensure that every group member can taste each selected variety, the effective cost of a menu item is determined by both its price and the number of servings required based on the group size.

The Knapsack Problem is a well-known optimization problem in computer science where a subset of items is selected to maximize value while remaining within a limited capacity or resource constraint [1][2]. It exists with a few variants, including the Fractional Knapsack Problem, 0/1 Knapsack Problem, Bounded Knapsack, and Unbounded Knapsack [1]. Since each menu item can be either selected or not in this case study, the model most likely resembles the 0/1 Knapsack Problem.

There are various algorithmic strategies that can be used to solve Knapsack-based problems, and among them the most common are Greedy algorithms and Dynamic Programming. While Greedy may provide efficient solutions for a select few of knapsack variants, they don't always guarantee an optimal solution for the 0/1 Knapsack Problem [1]. On the other hand, Dynamic Programming is a problem-solving technique that decomposes the problem into overlapping subproblems and stores previously computed results to avoid redundancy in computations [3]. As a result, Dynamic Programming is the commonly used solution for the 0/1 Knapsack Problem optimally and therefore is selected as the primary approach in this study.

To help simulate and evaluate the proposed approach, a Python program will be developed using a predefined set of dim sum menu items and prices obtained from publicly available online sources. The program will then determine the optimal combination of menu items that maximizes dim sum variety while satisfying all defined constraints. Ultimately, this paper aims to demonstrate how Dynamic Programming can be applied to solve practical optimization problems encountered in everyday group decision-making scenarios, and with it, showing that we can somewhat effectively use concepts in computer

science to help us in our day-to-day lives for problems we are not aware of that can be solved this way.

II. Theoretical Frameworks

A. Combinatorial Optimization

Combinatorial optimization is a subfield on the topic of optimization in mathematics, emerging at the forefront of combinatorics and theoretical computer science that aims to use combinatorial techniques in order to solve discreet optimization problems. In combinatorial optimization, we seek to determine the most optimal solution within a finite set of possible solutions of a problem [4][5].

In the perspective of computer science, combinatorial optimization aims to improve an algorithm by using mathematical methods to either reduce the size of the set of possibilities or to make the search process itself swifter. In the combinatorics field, however, it models complex problems using mathematical structures such as sets and graphs, allowing researchers to analyze the problem systematically and apply established solution methods [4].

Problems in combinatorial optimization have a broad applicability and come in a variety of different types. The most common ones includes routing problems, which aim to determine the most efficient path between locations; scheduling problems, which seek the optimal arrangement of tasks over time; knapsack problems, which maximize the value of selected items while adhering to resource constraints; matching problems, which identify the best possible pairings between elements; and minimum spanning tree problems, which find the lowest-cost way to connect all nodes within a network [4]. Among these, the knapsack problem is particularly relevant to this study, since it focuses on selecting an optimal subset of items under limited resource with restrictions.

B. Knapsack Problem

The Knapsack Problem is one of the most fundamental combinatorial optimization problems in computer science and operations research. It involves selecting a subset of items, where each is associated with a given weight and value, to maximize the total value obtained while also ensuring that the total weight doesn't exceed the capacity constraint set in place [1][2]. Due to its ability to model resource allocation and decision-making scenarios, it has been widely applied in various domains, including logistics, scheduling, transportation, and resource management [1].

In its classical form, the problem can be portrayed as follows. Given a knapsack with a maximum capacity of W and a set of N items, where each has an associated weight w_i and value v_i, the goal is to determine which items should be put into the knapsack so that the total value is maximized while keeping the total weight within the capacity limit [1].

The Knapsack Problem itself can be broken down into several types according to GeeksforGeeks, including the Fractional Knapsack Problem, 0/1 Knapsack Problem, Bounded Knapsack Problem, and Unbounded Knapsack Problem [1].

The Fractional Knapsack Problem allows an item to be divided into smaller pieces, meaning that a fraction of an item may be selected if doing so helps increase the overall value obtained [1][2]. In contrast, the 0/1 Knapsack Problem requires each item to be either selected or rejected it its whole, without any partial selection allowed [1][2]. The Bounded Knapsack Problem extends this concept by limiting the number of instances of each item that may be selected, while the Unbounded Knapsack allows an unlimited number of copies of an item to be chosen instead [1].

Among these, the 0/1 Knapsack Problem is particularly relevant to this study. Each dim sum variety is treated as a single decision unit that can either be selected or not. Although a selected variety may require multiple servings to accommodate all group members, the number of required servings is calculated before hand and incorporated into the item's effective cost. Consequently, the decision variable remains binary, preserving the characteristics of the 0/1 Knapsack Problem [1][2].

The Knapsack Problem is notable not only because of its practical applications but because of the assortments of algorithmic approaches that can be used to solve it. Common solution methods involve brute-force search, recursive approaches, greedy algorithms, branch-and-bound techniques, and dynamic programming [2]. Amid these, dynamic programming is widely recognized as one of the most effective approaches for obtaining optimal solutions for the 0/1 Knapsack Problem by systematically solving and storing the result of overlapping subproblems [2].

C. 0/1 Knapsack Problem

The 0/1 Knapsack Problem is a variant of the Knapsack Problem in which each item can only be selected once or not at all. Unlike the Fractional Knapsack, where items may be divided into smaller parts, the 0/1 Knapsack treats every item as a whole, meaning that partial selection is not allowed [1][2].

Given a set of N items and a knapsack with a maximum capacity of W, where each item i is associated with a value v_i and weight w_i, the goal is to determine the subset of items that will maximize the total value without making it exceed the knapsack's capacity [1].

Mathematically, the 0/1 Knapsack Problem can be expressed as follows [1]:

$$max \sum_{i=1}^{N} v_i x_i$$

subject to

$$\sum_{i=1}^{N} w_i x_i \leq W$$

where

$$x_i \in \{0,1\}$$

In this formulation, x_i = 1 indicates that item i is selected, while x_i = 0 means otherwise. The objective is therefore to

maximize the total value of the chosen items while ensuring that the total weight still remains within the limit.

This problem is commonly solved using Dynamic Programming because the problem exhibits two important properties: optimal substructure and overlapping subproblems. Optimal substructure implies that an optimal solution to the overall problem can be constructed from optimal solutions of its smaller subproblems. Meanwhile, overlapping subproblems occur when the same intermediate calculations are repeatedly required during the solution process. By storing previously computed results, Dynamic Programming avoids redundant calculations and significantly improves computational efficiency [2].

In the context of this study, the dim sum selection problem can be modeled as a variation of the 0/1 Knapsack Problem, where each dim sum variety is treated as a single item that can either be chosen or not. Since a menu item cannot be partially ordered, the binary decision structure of the 0/1 Knapsack Problem is preserved. Furthermore, the number of servings required for each selected variety is calculated prior to optimization based on the group size and the number of pieces provided per serving. This preprocessing step determines the effective cost associated with selecting a menu item while maintaining a binary selection decision.

The correspondence between the classical 0/1 Knapsack Problem and the proposed dim sum selection problem is shown in the table below.

Table 1. Mapping of 0/1 Knapsack Components to the Dim Sum Selection Problem

Classical 0/1 Knapsack Component	**Dim Sum Selection Problem**
Item	Dim Sum Variety
Weight ($\boldsymbol{w_i}$)	Effective Cost of a Dim Sum Variety
Capacity ($\boldsymbol{W}$)	Available group budget
Value ($\boldsymbol{v_i}$)	Contribution to Menu Variety
Decision Variable ($\boldsymbol{x_i}$)	Whether the Menu Item is Selected or Not

Under this formulation, the goal of the problem is to maximize the number of dim sum variety that can be sampled by the group while satisfying the available budget and dietary constraints. Additional constraints, such as ensuring that sufficient servings of each selected variety are available for all group members and excluding menu items that violate dietary restrictions, further adapt the classical 0/1 Knapsack Problem to the group dining scenario examined in this paper.

D. Dynamic Programming

Dynamic programming is a problem-solving and optimization technique that breaks a complex problem into smaller, simpler subproblems whose solutions are stored and reused for constructing the solution of the original problem [3][6]. By avoiding repeated computations of identical subproblems, dynamic programming can significantly improve the computational efficiency compared to that of a naïve recursion approach [3][6].

Unlike divide-and-conquer algorithms, dynamic programming is particularly suitable for problems that have overlapping subproblems. Rather than solving the same subproblems multiple times, it will store and reuse when needed the previously computed results, reducing both computation time and redundancy [3][6].

A problem is generally considered suitable for dynamic programming if it exhibits two vital properties: optimal substructure and overlapping subproblems [3][6].

- Optimal Substructure

A problem exhibits optimal substructure when its optimal solution can be constructed from it's optimal solutions of its smaller subproblems [6]. In other words, solving the overall problem optimally requires solving each of its constituent subproblems optimally. This property allows a large problem to be recursively decomposed into smaller and more manageable components.

For instance, in a shortest-path problem, the shortest path between two vertices can be obtained by combining shorter optimal paths that connect intermediate vertices [6].

- Overlapping Subproblems

A problem exhibits overlapping subproblems when the same subproblem appears multiple of times during computation [3][6]. In such cases, repeatedly solving identical subproblems makes it inefficient. Dynamic programming addresses this issue by storing previously computed solutions and reusing them whenever the same subproblem is encountered again.

A common example of this is the Fibonacci sequence, where when computing a large Fibonacci number recursively, the same intermediate Fibonacci values are often recalculated multiple times. By storing these intermediate results, dynamic programming eliminates unnecessary recomputation and improves efficiency [3][6].

- Dynamic Programming Approach

Dynamic programming can generally be implemented using two approaches: top-down and bottom-up [6].

The top-down approach, commonly referred to as memoization, begins by solving the original problem recursively and storing the results of subproblems as they are computed. When the same subproblem is later encountered, the stored result is retrieved instead of being recalculated [6].

The bottom-up approach, also known as tabulation, starts from the smaller subproblems before gradually building solutions to larger subproblems until the final solution is obtained. This approach avoids recursive function calls and is often more memory-efficient than the top-down method [6].

- Dynamic Programming for the 0/1 Knapsack

The 0/1 Knapsack Problem satisfies both the optimal substructure and overlapping subproblem properties, making it well-suited for Dynamic Programming [3][6]. The optimal

solution for a knapsack with a given capacity can be derived from optimal solutions to smaller capacities and subsets of items. Furthermore, many intermediate states are repeatedly evaluated during the optimization process.

As a result, Dynamic Programming is widely used to solve the 0/1 Knapsack Problem by storing solutions to previously computed states and reusing them when necessary. This enables the algorithm to efficiently determine an optimal set of items while satisfying the specified capacity constraints [1][2][3].

III. Methodology

A. Problem Representation

This study models the group dim sum selection scenario as a variation of the 0/1 Knapsack Problem, where each dim sum variety is treated as a single item that may either be chosen or not. The goal is to maximize the number of distinct dim sum varieties that can be sampled by the group while satisfying the available budget and dietary constraints.

Each menu item is represented by the following attributes:

- Price (P_i)
- Number of pieces per serving (S_i)
- Dietary information or allergen content
- Variety contribution (V_i)

Since every member within the group must be able to taste each selected dim sum variety, the number of servings required for a menu item depends on both the group size and the number of pieces provided per serving.

The required number of servings for menu item i is calculated as:

$$RequiredServings_i = \left\lceil \frac{GroupSize}{S_i} \right\rceil$$

where:

- $GroupSize$ = number of diners in the group
- S_i = number of pieces provided in one serving of menu item i

The effective cost of selecting menu item i is then defined as:

$$EffectiveCost_i = Price_i \times RequiredServings_i$$

This effective cost is used as the weight component in the proposed knapsack formulation.

B. Objective Function

The primary objective of this study is to maximize the number of distinct dim sum varieties that can sampled by the group.

Let:

$$x_i = f(x) = \begin{cases} 1, & if\ menu\ item\ i\ is\ selected \\ 0, & otherwise \end{cases}$$

The objective function can therefore be expressed as:

$$max \sum_{i=1}^{N} x_i$$

where N represents the total number of available dim sum varieties.

C. Constraints

The optimization process is subject to several constraints:

1. Budget Constraint

The total effective cost of all selected menu items must not exceed the available budget, as formulated below:

$$\sum_{i=1}^{N} EffeciveCost_i x_i \leq Budget$$

2. Dietary Restriction Constraint

Menu items containing allergens that conflict with the dietary restrictions of any group member are excluded from the candidate item set before the optimization process is performed.

3. Serving Requirement Constraint

Each selected menu item must provide enough servings so that everyone can receive at least one piece of the selected dim sum variety. This requirement is incorporated into the effective cost calculation through the RequiredServings variable.

D. Dynamic Programming Formulation

To solve the optimization problem, a Dynamic Programming approach is employed.

A 2-D DP table is constructed where $DP[i][b]$ represents the maximum number of dim sum varieties that can be obtained by considering the first i menu items under budget b.

For each menu item, two decisions are possible:

- Exclude the item from the solution
- Include the item if the remaining budget permits it

The recurrence relation is defined as:

$$\max(DP[i-1][b], 1 + DP[i-1][b - EffectiveCost_i)$$

for

$$EffectiveCost_i \leq b$$

Otherwise:

$$DP[i-1][b]$$

The final solution is obtained from:

$$DP[N][Budget]$$

which represents the maximum achievable dim sum variety under all specified constraints.

E. Experimental Setup

To help evaluate and simulate the proposed approach, a predefined dim sum menu will be constructed using publicly available menu information collected from online sources. Each menu item will contain information regarding its price, quantity of pieces per serving, and allergen content.

Several test scenarios will be evaluated by varying:

- Available budget
- Group size
- Dietary restrictions

The resulting combination of the selected menu items, total effective cost, and achieved variety count will then be analyzed to assess the effectiveness of the proposed Dynamic Programming approach.

IV. Results and Discussions

A. Test Case Results

The proposed Dynamic Programming approach was evaluated using four test scenarios with varying group sized, budgets, and dietary restrictions. The objective of each test was to determine the maximum numbers of dim sum varieties that could be selected while satisfying all specified constraints.

Table 2. The Results of Each Test Case

Test Case	Group Size	Budget (IDR)	Restrictions	Variety Count	Total Cost (IDR)	Remaining Budget (IDR)
1	4	250000	None	6	249997	3
2	6	300000	Shrimp Allergy	6	292722	7278
3	3	150000	Vegan	2	49999	100001
4	15	30000	Vegan + Gluten Intolerance	0	0	30000

B. Discussion

1. Test Case 1: No Dietary Restrictions

In the first scenario, the group consisted of four diners with an available budget of IDR 250,000 and no dietary restrictions. The algorithm successfully selected six different dim sum varieties while utilizing nearly the entire budget.

The selected menu consisted of Siomay Udang Ayam, Siomay Ayam, Hakau, Xiao Long Bao Ayam, Pao Telur Asin, and Mantao Kukus HAKA. The total effective cost reached IDR 249,997, leaving only IDR 3 unused.

This result demonstrates that the Dynamic Programming approach effectively maximizes menu variety under budget constraints. Since no dietary restrictions were imposed, the algorithm was able to choose from the entire dataset and identify a combination that nearly fully utilized the available budget while achieving the highest variety count.

2. Test Case 2: Shrimp Allergy Restriction

The second scenario introduced a shrimp allergy restriction while increasing the group size to six diners and the budget to IDR 300,000.

All menu items containing shrimp were automatically excluded during the preprocessing stage. Despite this restriction, the algorithm was still able to obtain six distinct dim sum varieties. However, the selected menu differed significantly from the first test case, consisting primarily of non-shrimp alternatives such as Siomay Ayam, Xiao Long Bao Ayam, Lumpia Steam Saus Tiram, Pao Telur Asin, Bao Ayam Panggang Merah, and Dimsum Asparagus Tio Chiu.

Although the variety count remained unchanged, the total effective cost increased due to the larger group size, which required additional servings of each selected item. This result demonstrates that the preprocessing stage successfully handled allergen restrictions without affecting the correctness of the optimization process.

3. Test Case 3: Vegan Restriction

The third scenario evaluated the algorithm under a vegan dietary restriction with a group size of three diners and a budget of IDR 150,000.

Only two menu items in the dataset satisfied the vegan requirement: Dimsum Asparagus Tio Chiu and Dimsum Panggang Daging Vegetarian. Consequently, the maximum achievable variety was limited to two menu items regardless of the available budget.

The total cost of the selected items was only IDR 49,999, leaving a substantial portion of the budget unused. This outcome highlights an important characteristic of the proposed model: the optimal solution is constrained not only by budget limitations but also by the availability of feasible menu items after dietary filtering.

4. Test Case 4: No Feasible Solution

The final scenario was designed to evaluate the system's behavior when no feasible solution exists. The group size was increased to fifteen diners while the budget was limited to IDR 30,000. Additionally, both vegan and gluten-free restrictions were applied.

Although at least one menu item satisfied the dietary requirements, the effective cost calculation significantly increased the cost of each candidate item due to the large group size. As a result, every feasible menu item exceeded the available budget.

The program correctly reported that no feasible solution could be found. This demonstrates that the proposed approach is capable of identifying situations in which the specified constraints make the problem unsatisfiable rather than returning an invalid recommendation.

C. Overall Analysis

The experimental results indicate that the proposed Dynamic Programming approach successfully solves the modified 0/1 Knapsack formulation for group dim sum selection. Across all test cases, the algorithm consistently produced solutions that

satisfied the specified budget, serving requirements, and dietary constraints.

The results also reveal that dietary restrictions and group size significantly influence the achievable menu variety. While larger budgets generally allow more menu items to be selected, strict dietary restrictions may reduce the candidate item set and consequently limit the maximum attainable variety. Furthermore, increasing the group size increases the effective cost of each menu item due to the additional servings required, reducing the number of varieties that can be selected within a fixed budget.

Overall, the findings demonstrate that Dynamic Programming provides an effective method for optimizing menu variety in shared dining scenarios while respecting practical real-world constraints.

V. Conclusion

This paper proposed a Dynamic Programming-based approach for optimizing dim sum variety in group dining scenarios under budgetary and dietary constraints. By modeling the problem as a modified 0/1 Knapsack Problem, the study demonstrated how a real-world decision-making scenario can be transformed into a combinatorial optimization problem and solved algorithmically.

To accommodate the requirement that every diner must be able to sample each selected menu item, an effective cost formulation was introduced to adjust menu prices according to group size and serving quantity requirements. Dietary restrictions and food allergies were incorporated through a preprocessing stage that filtered infeasible menu items before optimization was performed. The resulting model preserved the binary decision structure of the classical 0/1 Knapsack Problem while adapting it to the shared dining context.

Experimental results showed that the proposed approach successfully maximized the number of dim sum varieties that could be selected while satisfying all specified constraints. The algorithm consistently generated valid recommendations across different budgets, group sizes, and dietary restrictions. Additionally, the system correctly identified cases where no feasible solution existed, demonstrating its robustness in handling edge cases.

Overall, the findings confirm that Dynamic Programming is an effective strategy for solving the proposed dim sum selection problem. More broadly, this study illustrates how concepts from algorithm design and combinatorial optimization can be applied to practical situations encountered in everyday life. Future work may explore additional factors such as individual food preferences, menu popularity, multi-objective optimization, or larger restaurant datasets to further improve the realism and applicability of the model.

VI. Appendix

GitHub Repo Link

https://github.com/Beeziebloop/DimsumMania

Acknowledgment

First and foremost, the author would like to express her deepest gratitude to Allah SWT, for His grace and guidance, without which the completion of this paper would not have been possible. Secondly, the author would like to extend her sincere thanks to Dr. techn. Muhammad Zuhri Catur Candra, S.T, M.T., Tricya Esterina Widagdo, S.T., M.Sc., and Dr. Fazat Nur Azizah, S.T., M.Sc. for their guidance, patience, and dedication throughout the IF2211 Strategy of Algorithms course, which greatly contributed to the development of this work. The author would also like to thank her family and friends for their continuous support, understanding, and encouragement, especially during challenging moments throughout the writing process. The author is truly grateful for all those who have contributed, directly or indirectly, to the completion of this work.

References

[1] GeeksforGeeks, "Introduction to Knapsack Problem, Its Types and How to Solve Them." Available: https://www.geeksforgeeks.org/dsa/introduction-to-knapsack-problem-its-types-and-how-to-solve-them/

[2] P. Yadav, "Understanding the Knapsack Problem: A Guide for Beginners," Medium. Available: https://medium.com/@prekshayadav0819/understanding-the-knapsack-problem-a-guide-for-beginners-d0146a59e9

[3] Brilliant.org, "Dynamic Programming." Available: https://brilliant.org/wiki/problem-solving-dynamic-programming

[4] Brilliant.org, “Combinatorial Optimization.” Available: https://brilliant.org/wiki/combinatorial-optimization/

[5] QuEra, “Combinatorial Optimization.” Available: https://www.quera.com/glossary/combinatorial-optimization

[6] Spiceworks, "What is Dynamic Programming." Available: https://www.spiceworks.com/soft-tech/what-is-dynamic-programming/

STATEMENT

With this, I hereby declare that this paper that I have written is my own, not an adaptation or translation of someone else’s work, and not plagiarized.

Bandung, 19 June 2026

Varistha Devi 13524135