CS221 Final Project: Hybrid Packing Algorithm: Optimizing Space Utilization and Suitcase Efficiency

Gonzales,China Mae C - Javier, April Kate M.- Labay, Caryll L.

May 19, 2023

1 Introduction

The Best Fit strategy aims to minimize wasted space by selecting the most suitable suitcase with the smallest available capacity that can accommodate each item. This strategy maximizes the utilization of existing suitcases, reducing the need for creating additional ones. However, in cases where an item cannot fit into any existing suitcases, the algorithm utilizes the Next Fit strategy. It creates a new suitcase and places the item in it, ensuring that all items are packed without leaving any behind. This strategy maintains the packing process's continuity and avoids the inefficiency of creating partially filled suitcases solely for individual items. By combining the Best Fit and Next Fit strategies, the algorithm optimizes both space utilization and the number of required suitcases. Its overarching goal is to minimize wasted space while minimizing the total number of suitcases, resulting in an efficient and compact packing solution for the items.

2 Problem Statement

The problem is to determine the minimum number of suitcases required to pack a given set of items, considering the size of each item and the capacity of the suitcases.

3 Algorithm Design

Inputs:
itemSize: A list of numbers representing the size of each item to be packed.
: An integer indicating the number of items to be packed.
: An integer specifying the capacity of each suitcase.

Output:
minSuitcase: An integer representing the minimum number ofsuitcases required to pack all the items.

Constraints:
The values in the size list are positive and represent the size of each item, with 0 ¡ size[i] ¡= c.

The values of n and c are positive integers.
The length of the size list is equal to n.

1. Initialize an empty list of suitcases.
2. Iterate through each item in the given set of items.
3. For each item, perform the following steps:
   a. Check if the item can fit into any existing suitcases using the Best Fit strategy.
   b. If a suitable suitcase is found, place the item in that suitcase.
   c. If the item cannot fit into any existing suitcases, create a new suitcase and place the item in it using the Next Fit strategy.
4. Repeat steps 2 and 3 until all items have been processed.
5. Return the list of suitcases with the packed items.

The algorithm starts by initializing an empty list of suitcases. Then, it iterates through each item and tries to fit it into the existing suitcases using the Best Fit strategy. If a suitable suitcase is found, the item is placed in that suitcase. If not, a new suitcase is created, and the item is placed in it using the Next Fit strategy to maintain continuity in the packing process. This process continues until all items have been processed. Finally, the algorithm returns the list of suitcases with the packed items. The algorithm's hybrid approach of combining the Best Fit and Next Fit strategies aims to optimize both space utilization and the number of suitcases required for efficient packing.]

4 Implementation

```
FUNCTION hybridSuitcasePacking(size, n, c)
   suitcaseCapacity = [c]
   FOR i = 0 TO n-1
    placed = False
   itemSize = size[i]
   // First, try to fit the item in an existing suitcase using Best Fit strategy
   FOR j = 0 TO length(suitcaseCapacity) - 1
        IF itemSize ¡= suitcaseCapacity[j] THEN
         suitcaseCapacity[j] -= itemSize
         placed = True
         BREAK
      END IF
    END FOR
    // If the item couldn't be placed in an existing suitcase, create a new suitcase (Next Fit strategy)
   IF NOT placed THEN suitcaseCapacity.append(c - itemSize)
      END IF
    END FOR
```

```
  RETURN length(suitcaseCapacity)
END FUNCTION
size = [0.5, 0.7, 0.5, 0.2, 0.4, 0.2, 0.5, 0.5, 0.6]
suitcaseCapacity = 3
numItems = length(size)
minSuitcase = hybridSuitcasePacking(size, numItems, suitcaseCapacity)
PRINT ”Number of suitcase required:”, minSuitcase
```

5 Analysis

The Customized Packing Algorithm implements the hybridSuitcasePacking function, which aims to determine the minimum number of suitcases required to pack a given list of items. The algorithm follows a hybrid approach that combines the Best Fit and Next Fit strategies to optimize the packing process. The algorithm begins by initializing a list called suitcaseCapacity to keep track of the remaining capacity of each suitcase. It then proceeds to iterate through the items, attempting to fit each item into an existing suitcase using the Best Fit strategy. For each item, the algorithm searches for the suitcase with the smallest available capacity that can accommodate the item's size. If such a suitcase is found, the item's size is subtracted from the corresponding suitcase's capacity in suitcaseCapacity. In cases where an item cannot fit into any existing suitcase, the algorithm creates a new suitcase with the remaining capacity after accommodating the item. The remaining capacity is determined by subtracting the item's size from the specified suitcase capacity, c. The new suitcase's remaining capacity is then appended to the suitcaseCapacity list. Once all items have been processed, the algorithm returns the length of the suitcaseCapacity list. This value represents the minimum number of suitcases required to pack all the items efficiently. The code example provided demonstrates the usage of the algorithm by initializing a list of item sizes (size), setting the suitcase capacity (suitcaseCapacity), and computing the minimum number of suitcases needed using the hybridSuitcasePacking function. The resulting minimum number of suitcases is then printed as output. All in all, the hybridSuitcasePacking algorithm efficiently packs items into suitcases by employing the Best Fit and Next Fit strategies. The provided code example showcases the algorithm's functionality by determining the minimum number of suitcases required for a specific set of item sizes.

Time Complexity:
The algorithm consists of an outer loop that iterates n times, where n is the number of items in the input list. Within each iteration, there is an inner loop that iterates up to the number of suitcases created so far. In the worst-case scenario, where each item needs to be placed in a new suitcase, the number of iterations for the inner loop is proportional to the number of items processed so far, resulting in an average time complexity of O(n^2).
However, if the items can be efficiently placed in existing suitcases using the Best Fit strategy, the time complexity can be significantly better than O(n^2).

Space Complexity: The algorithm utilizes a list, suitcaseCapacity, to store the remaining capacity of each suitcase. The length of the suitcaseCapacity list can grow as new suitcases are created. In the worst-case scenario, where each item requires a new suitcase, the length of suitcaseCapacity can be up to n, resulting in a space complexity of O(n). Additionally, the algorithm uses a few variables with constant space requirements.

Next Fit Algorithm

The Next Fit algorithm and the algorithm share a similar goal, which is to solve the suitcase packing problem by packing items into suitcases based on their sizes.

Time Complexity:
The Next Fit algorithm has an O(n) time complexity, where n is the quantity to be packed. This is due to the algorithm iterating over each item once. It executes constant-time actions on each item to determine whether it can fit in the current suitcase, update the capacity, and determine whether it can. The iteration over the objects is linear, and the operations carried out inside the loop have constant time complexity. As a result, the overall time complexity grows linearly as the number of objects increases.

Space Complexity:
The space complexity of the Next Fit algorithm is O(1), meaning it requires constant space. The algorithm doesn't require any additional data structures besides the input array that holds the item sizes. It iterates over the items consecutively without saving any intermediate data structures and uses variables to keep track of the current suitcase's capacity. As a result, the amount of space needed is constant and unchanged by the number of items.

Best Fit Decreasing Algorithm

The Best Fit Decreasing algorithm is another approach to solving the suitcase packing problem.

Time Complexity:
The Best Fit algorithm has a time complexity of O(n * m), where n is the number of items and m is the number of suitcases. For each item, the algorithm needs to compare its size with the remaining capacity of each suitcase to find the best fit. This comparison is performed for each item in the list, resulting in a nested loop structure that iterates through both the items and suitcases. Therefore, the time complexity is proportional to the number of items multiplied by the number of suitcases.

Space Complexity:
The space complexity of the Best Fit algorithm is O(m), where m is the number of suitcases. It requires space to store the remaining capacity of each suitcase. This storage is typically implemented as an array or a list that keeps track of the available space in each suitcase. The space complexity is determined by the number of suitcases since the algorithm needs to maintain this information for each suitcase.

Comparison:

The customized algorithm has a worst-case time complexity of O(n2). While the Best Fit algorithm has a time complexity of O(n * m), and the Next Fit algorithm has a time complexity of O(n). In terms of space complexity, all three algorithms have the same complexity of O(m), as they all require storing the remaining capacity of each suitcase. Overall, the Best Fit algorithm has a more favorable time complexity compared to the customized algorithm. The Next Fit algorithm has the lowest time complexity among the three algorithms but may lead to less efficient space utilization compared to the Best Fit strategy. However, it's important to consider that the customized algorithm combines elements from both the Best Fit and Next Fit strategies, potentially achieving a balance between space utilization and the number of suitcases used. The algorithm provides customization and adaptability based on the specific characteristics of the items and problem instance. Furthermore, the algorithm is flexible and can handle a wide range of item sizes. It adapts to the size of each item and dynamically adjusts the packing strategy accordingly. The effectiveness of the customized algorithm would depend on the specific characteristics of the items and the problem instance.

In summary, the Best Fit algorithm offers a better time complexity, while the algorithm introduces a customized approach that balances space utilization and the number of suitcases used. The choice between the algorithms would depend on the specific requirements and constraints of the packing problem at hand.

6 Conclusion

The customized packing algorithm presents several advantages in certain scenarios when compared to other algorithms. Its simplicity and focus solely on item size make it easy to understand and implement, which can be beneficial for developers looking for a straightforward solution. Additionally, by eliminating the consideration of item weights, the algorithm reduces computational overhead and becomes more flexible for situations where weight differences among items are insignificant or irrelevant to the packing problem. Although the customized packing algorithm may not always yield the optimal number of suitcases used, it can still improve packing efficiency compared to algorithms like Next Fit. By distributing items evenly and optimizing space utilization, it aims to achieve a better overall packing result. This can be particularly advantageous in scenarios where efficient space usage is crucial, such as when packing items for shipping or organizing inventory in a warehouse. However, it's important to note that the effectiveness of the customized packing algorithm heavily depends on the specific characteristics and requirements of the packing problem at hand. For instance, if the packing problem involves items with significant weight disparities, considering item weights in the algorithm may become important to ensure stability and prevent damage. Similarly, if the packing problem requires certain constraints or has additional factors to consider, such as item fragility or stacking limitations, the algorithm may need to be further modified or extended to accommodate these requirements. The packing algorithm can be improved by incorporating enhanced optimization techniques such as genetic algorithms, simulated annealing, or particle swarm optimization, enabling more efficient exploration of the solution space and potentially better packing results. Additional constraints, like weight limitations, fragile item handling, or stacking restrictions, can be included to make the algorithm more versatile. Adaptive resizing techniques can dynamically adjust suitcase sizes based on item sizes, optimizing space utilization and reducing unused gaps. Benchmarking and comparisons with other algorithms would provide insights into strengths and weaknesses, guiding algorithm selection. Real-time optimization as new items are added would enable efficient and adaptive packing in dynamic environments like e-commerce or production line packaging. In summary, while the customized packing algorithm offers advantages in terms of simplicity, computational efficiency, and space utilization, its effectiveness depends on the specific characteristics and requirements of the packing problem. Further improvements and future work can focus on enhancing optimization techniques, incorporating additional constraints, exploring adaptive resizing, conducting comparative studies, and enabling real-time optimization.