Jonathan Calderon Chavez and Huiran Lin
Dr.Gurman Gill
CS415: Analysis of Algorithms
May 13, 2020

Approaches to the KnapSack problem

In this report, our group will discuss the efficiencies of the different algorithms that find or approximate the solution to the Knapsack problem.

We will start with the contrasting algorithms that fall under the dynamic programming design technique. We implemented the traditional and memory function approach to solving the knapsack problem.

The total number of basic operations to find optimal value and path in a knapsack of the capacity size of W with n items is $\Theta(Wn + n)$ using the traditional approach. However, the implementation of the memory function can reduce the number of basic operations by only computing a subset of the table values made in the traditional approach. In some cases, like in case id 0, 2, and 4 in figure 1, the number of basic operations computed by both algorithms is small. However, in examples such as case id 7 and 8 in figures 1 and 2, the number of basic operations performed by the memory function approach is significantly lower than that of the traditional method.

The memory function approach is the most efficient algorithm between the two, as in the worst case, the number of operations can be equal to the traditional approach. But In other cases, the number of operations can be lower than the traditional approach, making the memory function approach more efficient.

We next contrast the two different strategies of the “greedy” approach to solve the knapsack problem.

Using the sorting technique requires a total of $\Theta(n + nLog(n) + n)$ basic operations. n operations to compute the ratios. Since sorting the ratios has a run time of Θ(n log n) and finding the optimal subsets and value on average do not require traversing through every element in the ratio vector - around n operations - the total running time of the sorting technique is Θ(n log n). However, the technique using a max-heap improves upon it by requiring a slightly less total of basic operations in some cases. In the cases like in ID 7 and 8 in figure 3, the running time is better than that of the approach using the best sorting technique. But in cases where k in Θ(k log n) is close to n, the difference becomes negligible for example in case ids 2-6 in figure 3 .

The Greedy approach using max-heap is more efficient overall, since in the best cases where k is very small compared to n the running time is better than that of the approach using the best sorting algorithm. But in other cases, the gap between the efficiencies of the two techniques is small.

Figure 1

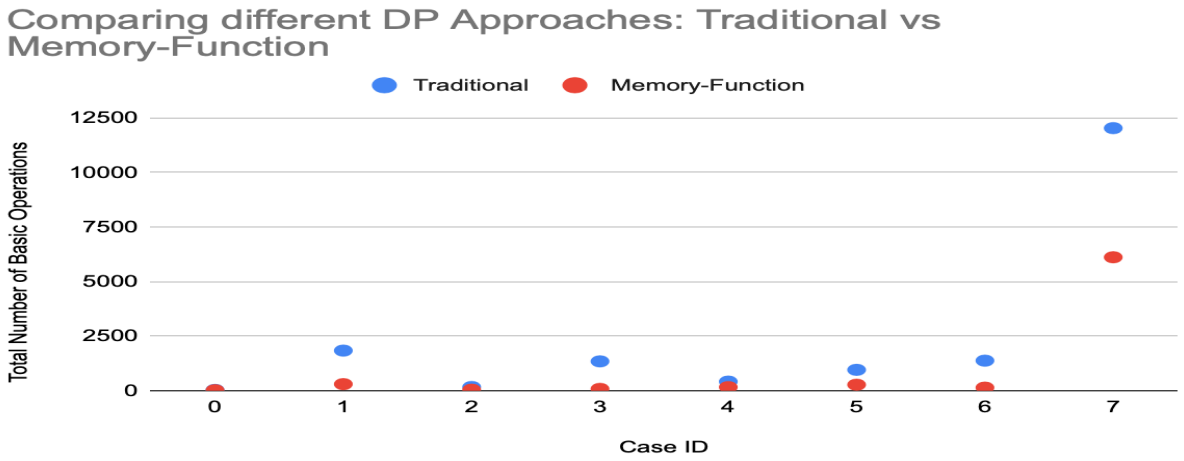

Figure 2

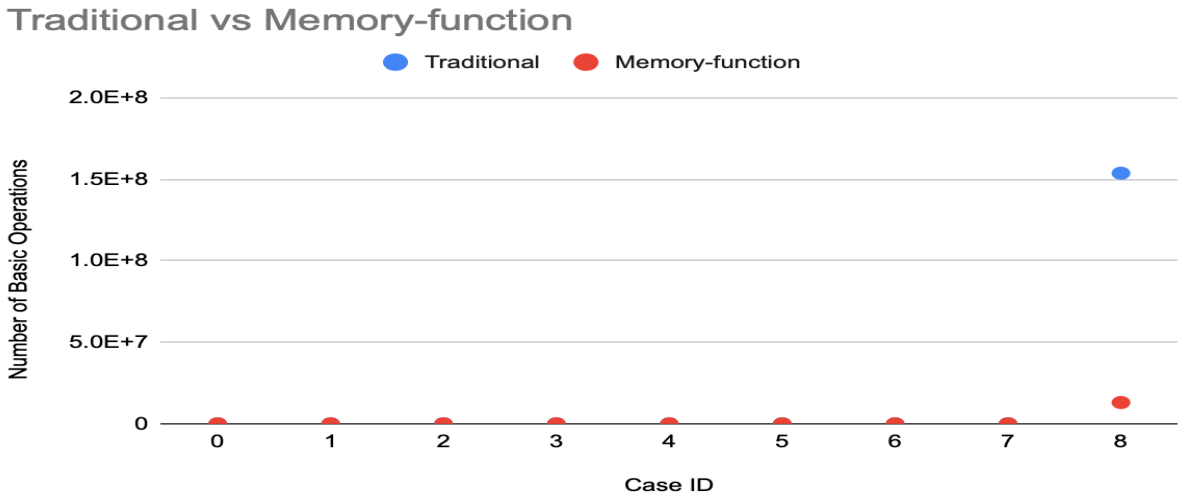

Figure 3