# COMP26120 Lab 5 Report

Ayush Gupta

## 1 Experiment

**Hypothesis** *The time required by the branch-and-bound algorithm to solve the knapsack instance exhibits a significant increase when the mean of the weights of items is less than half of the maximum weight capacity, contrasting with notably reduced times observed when the mean exceeds half of the maximum weight capacity.*

The hypothesis can be elucidated by considering the mechanics of branch and bound. Pruning within branch and bound occurs when partial solutions that are deemed infeasible are rejected. Accordingly, in scenarios where the distribution of item weights is skewed towards higher values, the algorithm is likely to encounter a greater number of infeasible partial solutions compared to situations where the mean weight is lower. Consequently, a larger proportion of branches will be pruned in the former case, leading to a notable improvement in the efficiency of the implementation.

**Experimental Design** To investigate the hypothesis, the experimental design involved generating knapsack instances with consistent parameters such as maximum weight capacity and the number of items. The variation among the instances was introduced through the mean of weights of the items, along with their profits which exhibited a strong correlation with the weights.

To maintain data consistency with respect to the mean, the random.normal function from the NumPy library was utilized in the data generation script. This function facilitated the random sampling of data from a normal distribution characterized by the mean and standard deviation passed as arguments. To enhance data concentration around the mean, the standard deviation was set to maxWeightCapacity / 50. The maxWeightCapacity was fixed to 10000 and there were 17 data Samples generated with mean 1000, 1500, 2000,.....,9000,and the experiments were performed twice with number of items being 30 and 50.

In the experiment, the mean of weights served as the independent variable, while the time taken by the branch-and-bound implementation to solve the knapsack instance acted as the dependent variable (the time was measured using the UNIX time command). The primary aim was to provide insights into the relationship between the distribution of weights over the items and the corresponding time required for solution. This approach facilitated a quantitative evaluation of the hypothesis, enabling a thorough examination of the proposed relationship.

**Results**

The results shown in Figure 1 clearly indicate a noticeable trend: when the mean values drop below 5000, the time it takes for computation significantly increases, reaching about 10-20 seconds. However, once the mean values go beyond this point, the computation time drastically decreases to less than 1 second. These findings strongly support our hypothesis. After running the Branch and Bound algorithm on each set of data, the results were plotted using Matplotlib.

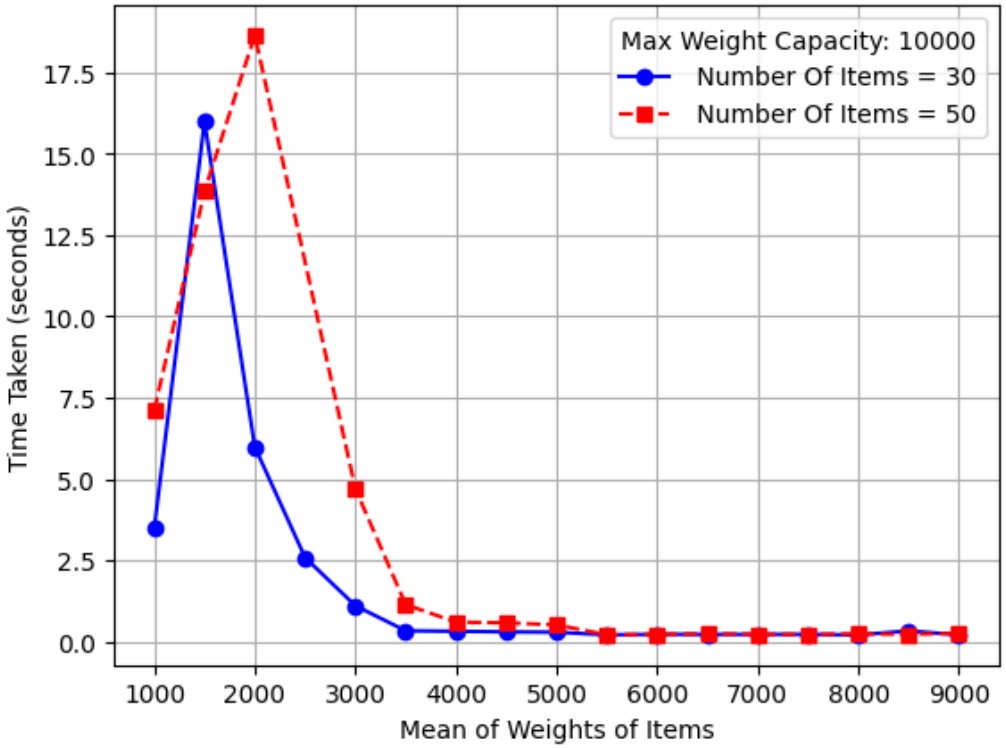


Figure 1: Effect of Weight Distribution Over the Items on Knapsack Solution Time

# 2 Data Statement

The raw data for the Experiment is given in Appendix A. The process of generating da and computing the time for this experiment was done using the shell scripts given in Appendix B. The main functions in the python script to generate data is given in Appendix C.

# A Raw Data for Experiment

| **Number of Items = 30** | |
|---|---|
| Means | Avg_Time (s) |
| 1000 | 3.477 |
| 1500 | 16.000 |
| 2000 | 5.985 |
| 2500 | 2.583 |
| 3000 | 1.103 |
| 3500 | 0.333 |
| 4000 | 0.317 |
| 4500 | 0.299 |
| 5000 | 0.289 |
| 5500 | 0.207 |
| 6000 | 0.224 |
| 6500 | 0.221 |
| 7000 | 0.221 |
| 7500 | 0.223 |
| 8000 | 0.205 |
| 8500 | 0.330 |
| 9000 | 0.207 |

| **Number of Items = 50** | |
|---|---|
| Means | Avg_Time (s) |
| 1000 | 7.106 |
| 1500 | 13.851 |
| 2000 | 18.645 |
| 3000 | 4.698 |
| 3500 | 1.136 |
| 4000 | 0.599 |
| 4500 | 0.576 |
| 5000 | 0.515 |
| 5500 | 0.208 |
| 6000 | 0.225 |
| 6500 | 0.240 |
| 7000 | 0.225 |
| 7500 | 0.223 |
| 8000 | 0.241 |
| 8500 | 0.223 |
| 9000 | 0.257 |

# B Shell Commands for Experiment

## B.1 Generating Data

```
python kp_generate.py $Number of Items$  $Weight Capacity$ $Name of Folder$
```

## B.2 Computing Run Times

```
time java comp26120.bnb_kp ../$Name of Folder$/$Mean$.txt
```

# C Script For Generating Data

## C.1 Generating KnapSack Instance

```
def generate_knapsack_instance(n, mean, std_dev, upper_bound):
    data = np.random.normal(mean, std_dev, n)
```

```
    data = np.clip(data, a_min=1, a_max=upper_bound)

    return np.round(data).astype(int)
```

## C.2 Loading the Instance in File

```
def load_knapsack_instance(n, c, name, mean, data):
    folder_path = name
    file_name = os.path.join(folder_path, f"{mean}.txt")
    os.makedirs(folder_path, exist_ok=True)
    with open(file_name, 'w') as fileObj:
        fileObj.write(str(n) + "\n")
        for i, weight in enumerate(data, start=1):
            profit = int(min(0.8 * weight + (mean + std_dev) / 5, (mean + std_dev)))
            fileObj.write(f"{i} {profit} {weight}\n")
        fileObj.write(f"{c}\n")
```