COMP26120 Lab 5 Report

Yaoxuan Ju

February 18, 2023

1 What makes a problem hard for Dynamic Programming?

1.1 Hypothesis

According to the Python code, the theoretical time complexity of the dynamic programming approach to solving the knapsack problem should be $O(nW)$, where n is the number of items and W is the maximum weight capacity of the knapsack. This is because the algorithm breaks the problem down into a set of smaller problems. There are n small problems and each small problem has W possible weights that it can take on. Since the algorithm needs to compute the optimal solution for each of these small problems, the total number of operations is proportional to $n \times W$.

Based on this analysis, a hypothesis can be formulated that, assuming the number of items remains constant, increasing the capacity of the knapsack will lead to a linear increase in the running time of dynamic programming.

1.2 Design

Variable change: The capacity of the knapsack will be changed. Specifically, the capacity of the knapsack will vary across multiple runs of the dynamic programming algorithm.

Input creation: To test the hypothesis, random knapsack instance files are generated. Each file will have a constant of 200 items, and the value and weight of each items are randomly assigned within the range of 1 to 500. Ten different sizes of capacity knapsack files are created, with sizes of 1K, 2K, 3K, 4K, 5K, 6K, 7K, 8K, 9K and 10K. Each size of knapsack instance forms 5 files, which is aimed to calculating the average running time in 5 files to reduce the chance of random generation and get a relatively typical average case.

Measurement: The running time of the dynamic programming algorithm for each input will be measured using a UNIX time command. This command

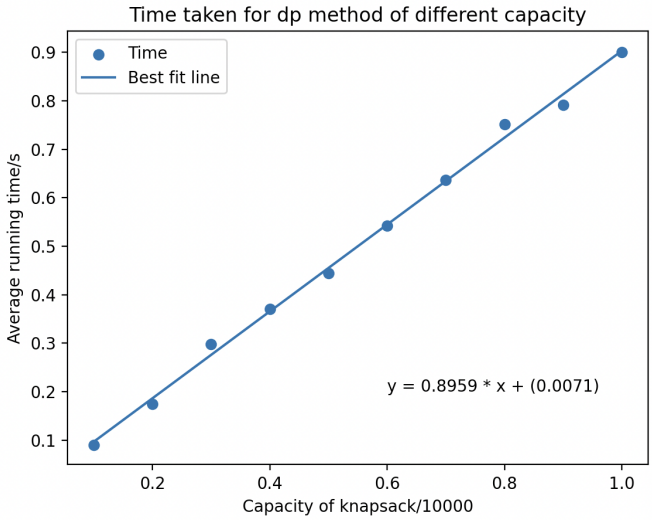

Figure 1: Time taken for dynamic programming method with different capacity knapsack with best fit line $y = 0.8959 \times x + 0.0071$ shown

returns the running time of each knapsack instance file with the dynamic programming method. For each capacity value, the average running time over all inputs will be calculated and taken as the final measurement. A graph of the average running time and the knapsack capacity will be plotted.

The shell scripts used to create instances and calculate time automatically can be found in Appendix A. They are also available in documents named generate.sh and compute.sh.

1.3 Results

The results were plotted using the matplotlib library in Python. A function named "linregress" in the Python scipy library was used to fit a line of the form $y = mx + q$, where m is the slope of the line and q is the intercept. These values were calculated in the process. The results can be seen in Figure 1, and the raw data can be found in Appendix B. The code for plotting the graph can be found in the "plot.py" Python file. As the capacity are large, the numbers on the x-axis were divided by 10,000 to make the result of the slope more reasonable. The graph clearly shows that the line $y = mx + q$ fits the data well, and the values of m and q are 0.8959 and 0.0071, respectively. Therefore, we can predict that the running time of the dynamic programming method in the knapsack problem is given by $0.8959x + 0.0071$ seconds for a capacity of $10,000x$.

1.4 Discussion

The results supports the hypothesis that increasing the capacity of the knapsack leads to a linear increase in the running time of dynamic programming, assuming the number of items remains constant. This is because the number of possible weight values that need to be considered increases with the capacity of the knapsack. As a result, the algorithm needs to compute the optimal solution for a larger number of subproblems, which increases the computational complexity and running time of the algorithm.

On the other hand, if the capacity of the knapsack is very small, the running time of the dynamic programming algorithm may be relatively fast. This is because there are fewer weight values that need to be considered, and the algorithm needs to compute the optimal solution for a smaller number of subproblems.

Overall, the capacity of the knapsack can be an important factor in the running time of dynamic programming algorithms for the knapsack problem. In practical applications, it is important to consider the trade-off between the capacity of the knapsack and the running time of the algorithm, and to choose an appropriate capacity that balances computational complexity and practical constraints.

1.5 Data Statement

The shell scripts for generating and computing knapsack instance can be found in Appendix A or in the file named generate.sh and computer.sh. The raw data for experiment can be found in Appendix B or the file named data_python.csv.

A Shell Scripts for Experiment

A.1 Generating Knapsack Instance

```
SIZES="1000 2000 3000 4000 5000 6000 7000 8000 9000 10000"

for SIZE in $SIZES
do
echo $SIZE

for COUNT in 1 2 3 4 5
do
python3 kp_generate.py 200 $SIZE 500 ./data2/test_${SIZE}_$COUNT.tex
done
done
```

A.2 Computing Run Times

```
SIZES="1000 2000 3000 4000 5000 6000 7000 8000 9000 10000"
```

```
rm data_python.csv


for SIZE in $SIZES
do
for COUNT in 1 2 3 4 5
do
# Debugging statement to check program calls working as expected
python3 ./python/dp_kp.py ./data2/test_${SIZE}_$COUNT.tex

ALL_TIME=‘(time -p python3 ./python/dp_kp.py ./data2/test_${SIZE}_$COUNT.tex) 2>&1 | grep -E

RUNTIME=0
for i in $ALL_TIME;
do RUNTIME=‘echo $RUNTIME + $i|bc‘;
done
echo $SIZE, $RUNTIME >> data_python.csv

done

done
```

B Raw Data for Experiment

Random Data		Random Data	
Size	Time (s)	Size	Time (s)
1000	0.09	6000	0.52
1000	0.09	6000	0.58
1000	0.09	6000	0.54
1000	0.09	6000	0.55
1000	0.09	6000	0.52
2000	0.17	7000	0.65
2000	0.19	7000	0.65
2000	0.17	7000	0.64
2000	0.17	7000	0.61
2000	0.17	7000	0.63
3000	0.33	8000	0.86
3000	0.29	8000	0.75
3000	0.3	8000	0.75
3000	0.29	8000	0.69
3000	0.28	8000	0.71
4000	0.37	9000	0.8
4000	0.36	9000	0.77
4000	0.39	9000	0.77
4000	0.35	9000	0.79
4000	0.38	9000	0.83
5000	0.46	10000	0.9
5000	0.44	10000	0.89
5000	0.45	10000	0.88
5000	0.43	10000	0.92
5000	0.44	10000	0.91