COMP26120 Lab 5 Report

Rui Xu

1.1 Hypothesis

There are several factors that can make a Knapsack problem Hard for Dynamic Programming. The time complexity for 0/1 Knapsack problem solved using DP is **O(N*W)** where N denotes the number of items available and W denotes the capacity of the knapsack. In addition to the number of items in the backpack, a knapsack's capacity has a significant impact on how long a dynamic programming solution runs, with larger capacities making the problem harder to resolve than smaller capacities. Without increasing the number of items, run time varies linearly with capacity.

1.2 Design

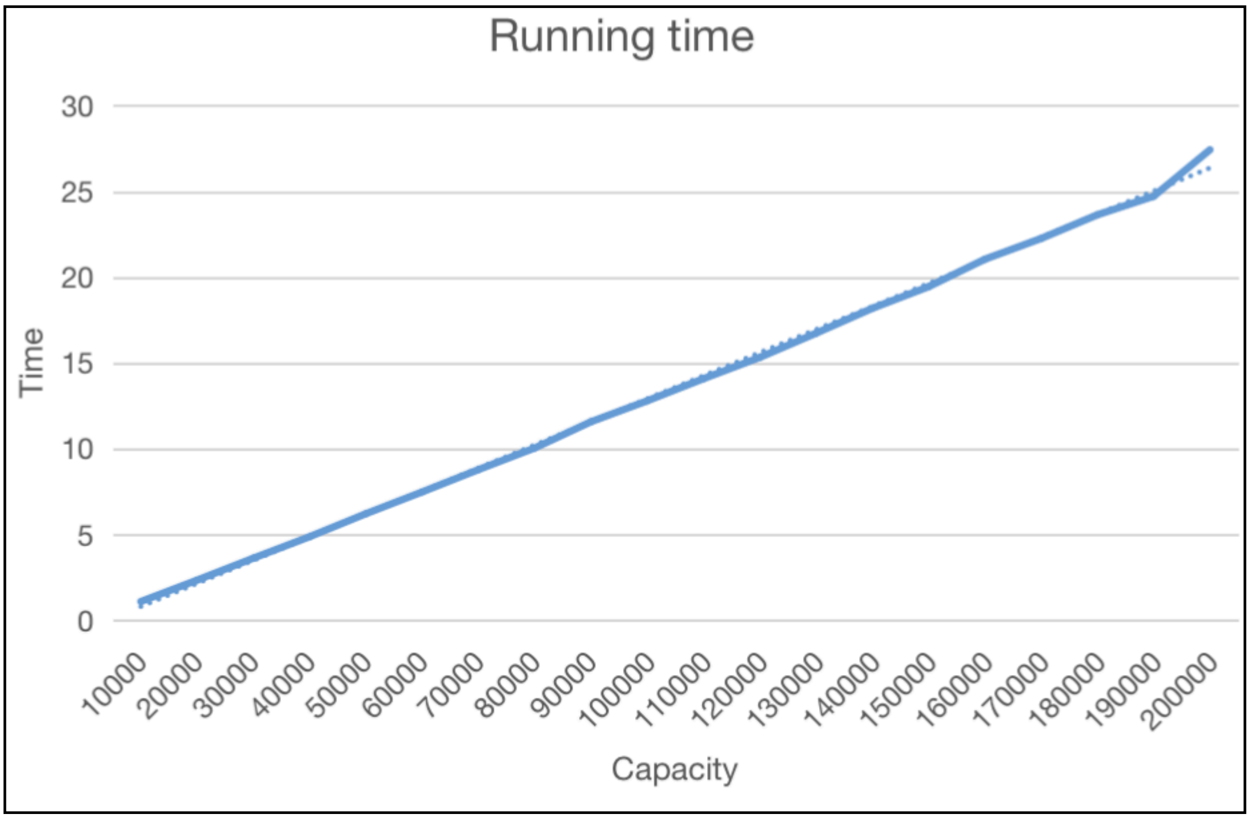
This experiment will modify the knapsack's capacity to see how it affects the dynamic programming solution's execution time. The experiment keeps the number of items and upper bound unchanged. Here it set the item number as 300, and the upper bound is 5000.

The knapsack's capacity will be changed by creating a number of input sets, each with a different capacity. The capacity range used is large enough to span from 10000 to 200000 data at a time of 1000. To make sure that findings are not limited to a particular input configuration, the program will make numerous input sets for each potential value of the capacity. A set of items with varied weights and values as well as a knapsack with the required capacity will serve as the inputs. This experiment creates the item weights and values using a python script called "kp_generate.py", which will automatically generate knapsack instance files.

For each set of inputs, a shell script called "kp_capacity_output.sh"(Appendix A) will track how long the dynamic programming process takes to run and record the time into CSV files(Please check 1.5 Data Statement). It will track the running time for each capacity level. This procedure will be repeated five times to make sure that the conclusion is accurate and reliable. The knapsack issue will be solved using a Python dynamic programming approach named "dp_kp.py", and the time module from UNIX/Linux will be used to calculate the program's execution time.

1.3 Results

Five data sets were used in the experiment and by using these five sets of data the average time was derived and drawn as a line graph as shown below:



THE TIME IN THIS GRAPH IS IN SECONDS

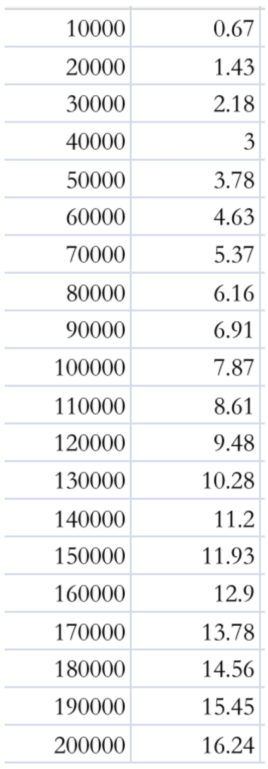
Overall, the running time tends to be linear, the solid blue line is the real data, and the dashed blue line is representative of the linear model.

1.4 Discussion

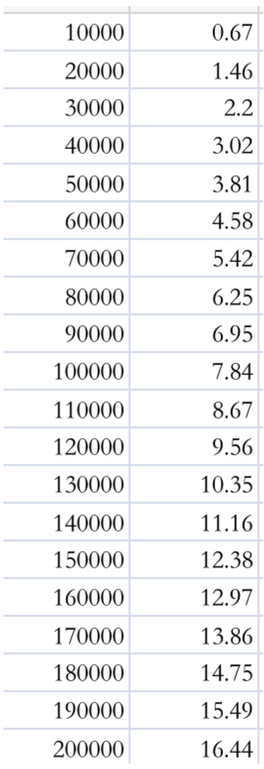
In general, the conclusion is consistent with the hypothesis. As the capacity of the knapsack increases, the number of possible combinations of items also increases, which could result in longer running times.

1.5 Data Statement

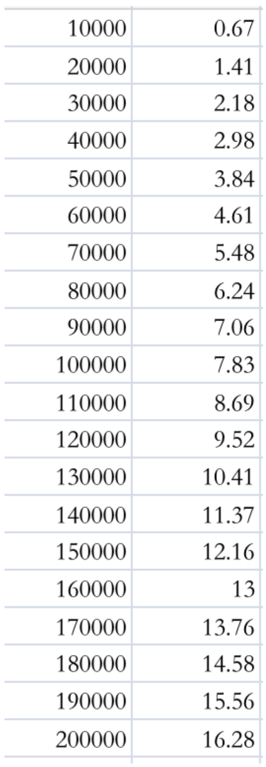
MULTIPLE INPUT SETS THAT VARY IN THEIR CAPACITY



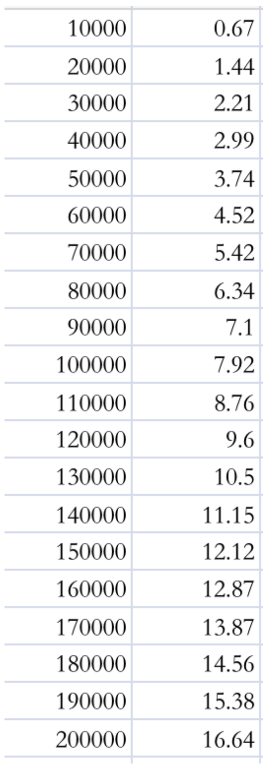
10000	0.67
20000	1.43
30000	2.18
40000	3
50000	3.78
60000	4.63
70000	5.37
80000	6.16
90000	6.91
100000	7.87
110000	8.61
120000	9.48
130000	10.28
140000	11.2
150000	11.93
160000	12.9
170000	13.78
180000	14.56
190000	15.45
200000	16.24



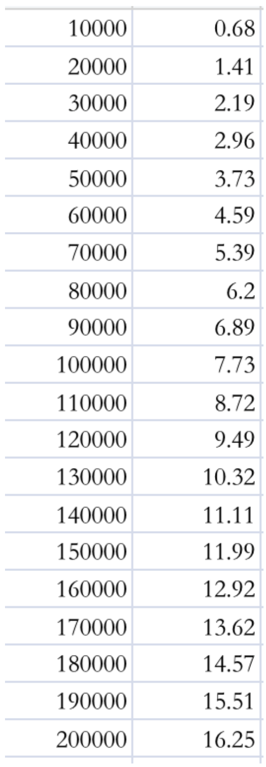
10000	0.67
20000	1.46
30000	2.2
40000	3.02
50000	3.81
60000	4.58
70000	5.42
80000	6.25
90000	6.95
100000	7.84
110000	8.67
120000	9.56
130000	10.35
140000	11.16
150000	12.38
160000	12.97
170000	13.86
180000	14.75
190000	15.49
200000	16.44



10000	0.67
20000	1.41
30000	2.18
40000	2.98
50000	3.84
60000	4.61
70000	5.48
80000	6.24
90000	7.06
100000	7.83
110000	8.69
120000	9.52
130000	10.41
140000	11.37
150000	12.16
160000	13
170000	13.76
180000	14.58
190000	15.56
200000	16.28



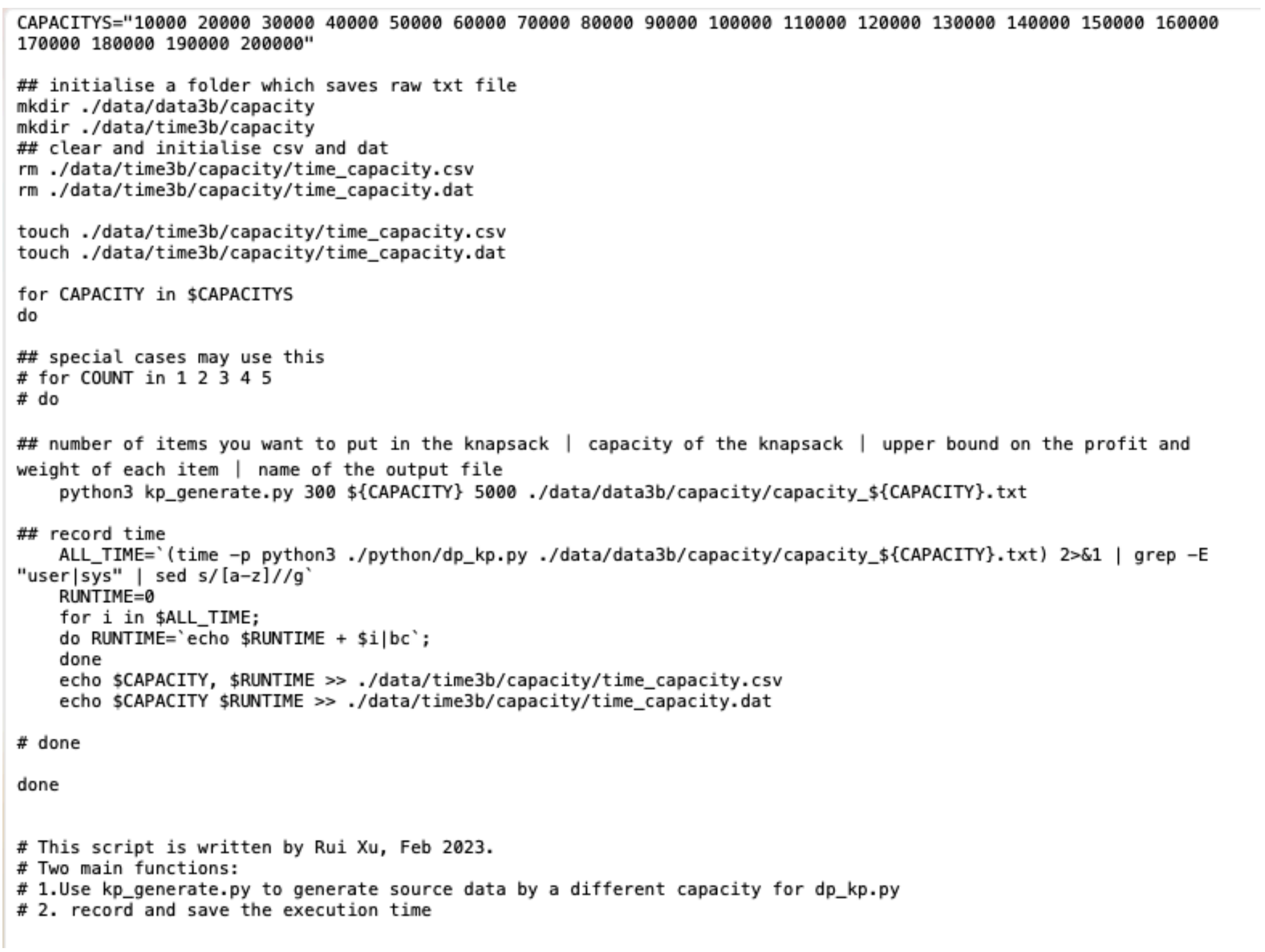
10000	0.67
20000	1.44
30000	2.21
40000	2.99
50000	3.74
60000	4.52
70000	5.42
80000	6.34
90000	7.1
100000	7.92
110000	8.76
120000	9.6
130000	10.5
140000	11.15
150000	12.12
160000	12.87
170000	13.87
180000	14.56
190000	15.38
200000	16.64



10000	0.68
20000	1.41
30000	2.19
40000	2.96
50000	3.73
60000	4.59
70000	5.39
80000	6.2
90000	6.89
100000	7.73
110000	8.72
120000	9.49
130000	10.32
140000	11.11
150000	11.99
160000	12.92
170000	13.62
180000	14.57
190000	15.51
200000	16.25

1.6 Appendix

A



```
CAPACITYS="10000 20000 30000 40000 50000 60000 70000 80000 90000 100000 110000 120000 130000 140000 150000 160000
170000 180000 190000 200000"

## initialise a folder which saves raw txt file
mkdir ./data/data3b/capacity
mkdir ./data/time3b/capacity
## clear and initialise csv and dat
rm ./data/time3b/capacity/time_capacity.csv
rm ./data/time3b/capacity/time_capacity.dat

touch ./data/time3b/capacity/time_capacity.csv
touch ./data/time3b/capacity/time_capacity.dat

for CAPACITY in $CAPACITYS
do

## special cases may use this
# for COUNT in 1 2 3 4 5
# do

## number of items you want to put in the knapsack | capacity of the knapsack | upper bound on the profit and
weight of each item | name of the output file
    python3 kp_generate.py 300 ${CAPACITY} 5000 ./data/data3b/capacity/capacity_${CAPACITY}.txt

## record time
    ALL_TIME=`(time -p python3 ./python/dp_kp.py ./data/data3b/capacity/capacity_${CAPACITY}.txt) 2>&1 | grep -E
"user|sys" | sed s/[a-z]//g`
    RUNTIME=0
    for i in $ALL_TIME;
    do RUNTIME=`echo $RUNTIME + $i|bc`;
    done
    echo $CAPACITY, $RUNTIME >> ./data/time3b/capacity/time_capacity.csv
    echo $CAPACITY $RUNTIME >> ./data/time3b/capacity/time_capacity.dat

# done

done


# This script is written by Rui Xu, Feb 2023.
# Two main functions:
# 1.Use kp_generate.py to generate source data by a different capacity for dp_kp.py
# 2. record and save the execution time
```