Algorithms and Data Structures: Dynamic Programming approach for 0/1 Knapsack PROBLEM

Pavel Ghazaryan

February 23 2023

1 Introduction

The knapsack problem is a classic optimization problem that seeks to determine the optimal way to pack items of varying weights and values into a knapsack with a limited capacity. The dynamic programming approach is a common algorithmic technique used to solve this problem, in which a two-dimensional array is filled with the best possible profit for a given number of items and knapsack capacity.

2 Hypothesis

In the dynamic programming approach, a two-dimensional array of size N (number of items) x C (capacity of knapsack) is utilized, where each cell corresponds to the best possible profit obtainable, given N and C as the row and column numbers, respectively. The time complexity of the dynamic programming approach to solve the knapsack problem is O(N x C). Notably, besides the number of items, the time complexity is also influenced by the capacity of the knapsack. Consequently, an increase in the capacity of the knapsack will result in a lengthier execution time for the dynamic programming approach. Thus, the hypothesis for the experiment would be to say that a larger capacity of the knapsack will increase the execution time of the problem, making it more challenging for the dynamic programming approach.

3 Design

For this experiment, 15 tests were generated using the provided kp_generate.py file. Each test contained 100 items with varying weights and values, and the capacity of the knapsack was increased for each test. The range of capacity values was chosen to ensure that each test presented a unique challenge for the dynamic programming approach. Capacity values range from 100 to 10000. The range for values and weights of items was chosen to be around the half of the capacity in order to keep knapsack algorithm logically correct and meaningful. What I mean is that if the weights range is too small for bigger knapsack capacities then we could just put all the items in the knapsack and that would be the best profit which is basically meaningless. A timer library was used to measure the execution time of the dynamic programming approach for each test which is the main output for measuring the correctness of the hypothesis. Please see the table below(Figure 1).

Nitems	Capacity	Time/s
100	100	0.0025555
100	200	0.004182
100	300	0.0063118
100	500	0.0106702
100	700	0.0153394
100	1000	0.0216828
100	1200	0.0258385
100	1800	0.0387857
100	2500	0.0546111
100	3000	0.065373
100	4000	0.0868224
100	5000	0.1056869
100	7000	0.1512424
100	8000	0.1731939
100	10000	0.214032

Figure 1: Number of items, Capacities and resulting execution time for each test

4 Results

The graph representing the dataset was generated by utilizing the x-axis to represent the capacity of the knapsack and the y-axis to depict the corresponding execution time. This visualization was produced using the Matplotlib library in the Jupyter Notebook environment. The resulting graphical representation is presented below(Figure 2).

Observing the graphical representation, it is evident that as the capacity of the knapsack increases, the execution time of the dynamic programming approach also increases. This relationship indicates that the larger the capacity value, the more challenging the problem becomes to solve utilizing dynamic programming.

5 Discussion

The results of this experiment support the hypothesis that the capacity of the knapsack has a significant impact on the execution time of the dynamic programming approach to the knapsack problem. As the capacity of the knapsack increases, the number of possible combinations that must be considered also increases, making the problem more difficult to solve with dynamic programming.

6 Conclusion

In conclusion, the results of this experiment confirm that the capacity of the knapsack has a significant impact on the execution time of the dynamic pro-

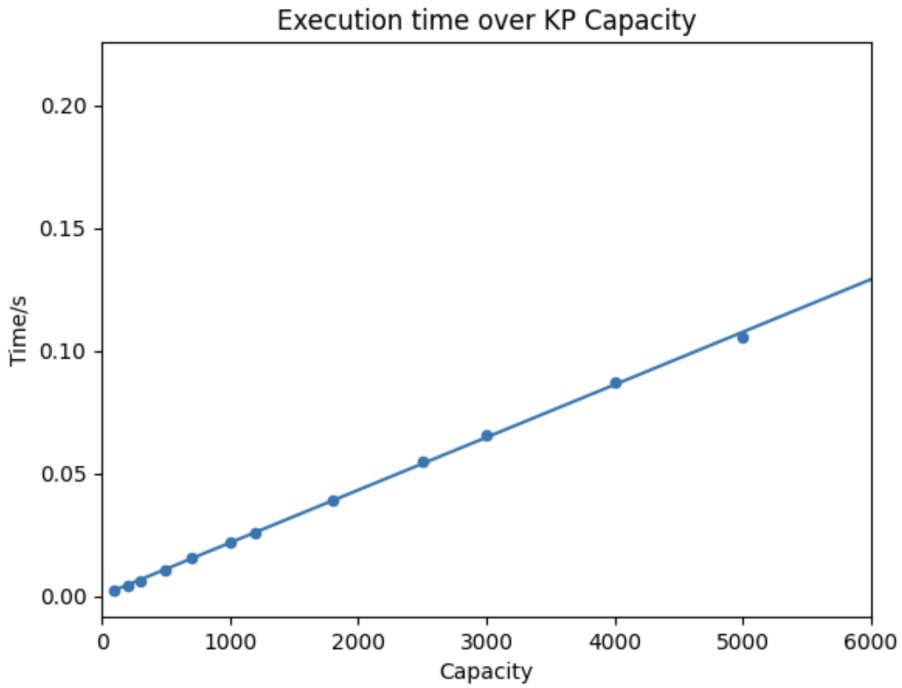

Figure 2: Execution time of dynamic programming with respect to capacity of Knapsack

gramming approach to the knapsack problem. This finding has implications for the development of more efficient algorithms for solving the knapsack problem, particularly in situations where the capacity of the knapsack is expected to be large. Further research could explore alternative approaches to solving the knapsack problem, such as branch and bound algorithm, and compare their performance to that of the dynamic programming approach.

7 Data Statement

All generated test inputs and results can be found in gitlab repository in data/tests and data/results folders. Some are also in the appendix.

Appendices

```
N: 100
1 31 6
2 44 40
3 40 46
4 40 25
5 19 4
6 13 1
7 14 47
8 30 35
9 27 38
10 11 17
11 50 41
12 14 5
13 43 37
14 38 46
15 11 14
16 7 22
17 43 32
18 37 41
19 26 35
20 35 43
21 1 18
22 13 44
23 16 32
24 10 15
25 25 2
26 32 8
27 26 10
28 39 1
29 25 19
30 18 1
31 25 47
32 38 49
33 9 3
34 36 39
35 4 11
36 27 28
37 8 21
38 43 16
39 9 33
40 31 7
41 49 28
42 45 15
43 22 29
44 33 33
45 40 21
46 36 36
47 15 9
48 41 27
49 3 40
```

50 23 3
51 34 18
52 18 4
53 31 28
54 4 34
55 1 41
56 45 35
57 29 14
58 17 37
59 47 43
60 7 38
61 44 15
62 48 35
63 41 29
64 10 32
65 43 22
66 38 33
67 4 27
68 19 41
69 49 11
70 45 50
71 17 42
72 41 19
73 9 34
74 5 45
75 7 47
76 26 13
77 39 43
78 8 29
79 46 10
80 35 32
81 21 38
82 50 22
83 31 18
84 11 50
85 26 49
86 14 12
87 38 8
88 11 18
89 14 17
90 41 49
91 12 34
92 40 25
93 33 46
94 10 1
95 35 47
96 30 42
97 25 22
98 22 5
99 36 22

```
100 35 38
C: 100

N: 100
1 50 65
2 23 46
3 56 34
4 41 88
5 92 19
6 75 100
7 4 53
8 59 38
9 37 85
10 9 4
11 83 84
12 53 1
13 62 51
14 71 26
15 59 75
16 49 94
17 65 79
18 72 79
19 40 18
20 14 48
21 18 16
22 71 49
23 98 53
24 88 28
25 86 61
26 29 96
27 11 89
28 20 51
29 7 86
30 46 2
31 73 54
32 81 36
33 54 65
34 39 59
35 56 72
36 29 47
37 69 60
38 15 63
39 60 57
40 30 6
41 83 91
42 6 29
43 39 42
44 60 72
45 55 40
46 42 97
```

47 22 73
48 52 32
49 70 66
50 89 3
51 76 73
52 98 66
53 73 96
54 19 4
55 10 87
56 30 42
57 33 23
58 100 31
59 29 82
60 73 19
61 92 40
62 19 1
63 6 20
64 25 3
65 60 73
66 96 48
67 27 53
68 51 64
69 29 24
70 82 59
71 97 81
72 42 84
73 38 77
74 83 39
75 38 49
76 100 17
77 16 56
78 11 83
79 20 95
80 53 61
81 65 69
82 41 18
83 9 61
84 58 98
85 88 90
86 88 87
87 18 68
88 72 43
89 63 11
90 88 91
91 60 27
92 27 18
93 66 65
94 1 70
95 72 3
96 76 15

```
97 13 52
98 14 26
99 15 33
100 35 84
C: 200
```