COMP26120 Lab 5 Report

Sourabh Roy

February 16, 2024

1 Experiment – Difficult 0/1 Knapsack Problem for Branch and Bound

Hypothesis If the profit and weight of each item in a 0/1 Knapsack problem are strongly correlated via an equation of circle, then that problem will be difficult for **branch and bound** (BNB) algorithm to solve. In other words, for each weight, corresponding profit is:

$$profit = \sqrt{4r^2 - (weight - 2r)^2} \tag{1}$$

where r is the upper bound on weight. We will refer to such instances as **circle instances**[1].
In 0/1 Knapsack problems, given x_i a binary variable indicating whether item i is selected ($x_i = 1$) or not ($x_i = 0$), we are maximizing:

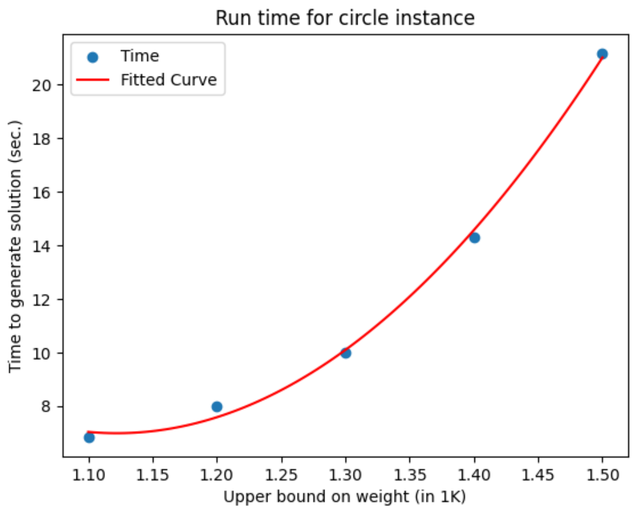
$$\sum_{i=1}^{n} profit_i \cdot x_i$$

subject to the constraint on the total capacity (C):

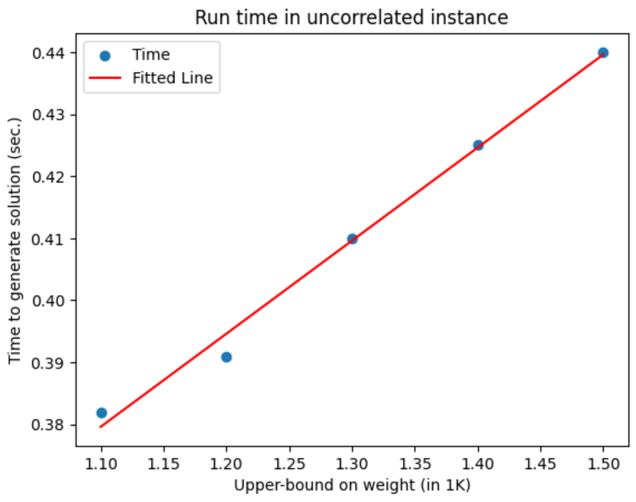
$$\sum_{i=1}^{n} weight_i \cdot x_i \leq C$$

Therefore, introducing additional non-linear constraint like (1) brings in non-convexity to the initial problem and makes the problem difficult to solve.

Experimental Design The independent variable is the upper bound (r) on weight of items. The dependent variable is the time taken to solve the given problem. The maximum capacity of Knapsack was fixed at 10000. The total number of items was fixed at 50. We compare the run-times of BNB in circle and uncorrelated instances.
5 circle instances were created. The upper limit on weight, r was varied as 1100, 1200, 1300, 1400, 1500. 5 uncorrelated instances with same variation in r were created; where weights and profits were both randomly generated.
Every instance was run 3 times on `bnb_kp.java`. All data has been generated using `kp_generate.py`. For the circle instances, the file was modified (see B). For uncorrelated instances, the default version of the file was used.
The time taken to generate solutions was measured by adding the `user` and `sys` values of `UNIX time` command. The average of 3 run-times for each instance, was taken for plotting.



(a) Time plot for circle instance



(b) Time plot for uncorrelated instance

Figure 1: BNB run-times for different instances

Results Figures 1-2 illustrate the results of the experiment. `Python Matplotlib` was used for making the plots. All best-fit curves were generated using `poly_fit` and `poly1D` functionalities of `Python numpy`.

Clearly, BNB performs worse on circle instances than in uncorrelated instances. This confirms the hypothesis that: given a Knapsack problem, if the profits and weights of each item are strongly correlated via the equation of a circle, it will make that problem more difficult for the BNB algorithm to solve.

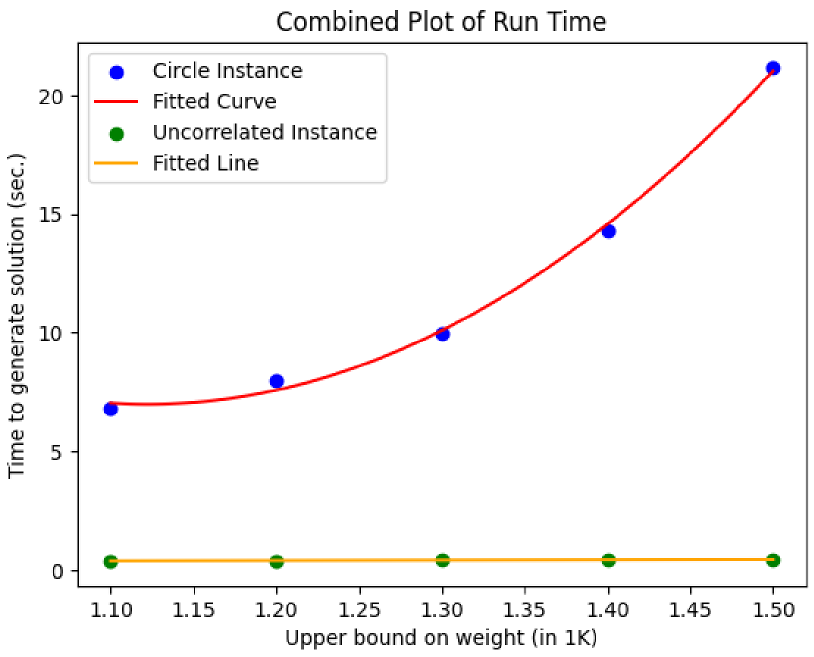


Figure 2: BNB takes significantly longer time for circle instances

2 Data Statement

The raw data for experiment can be found in Appendix A. The process of generating circle instances and computing the run-times for this experiment was done using the scripts shown in Appendix B. These can also be found in the `comp26120_2023_r67083sr` gitlab repository in the `lab5` directory.

References

[1] David Pisinger. Where are the hard knapsack problems? *Computers and Operations Research*, 32, 2005. `https://doi.org/10.1016/j.cor.2004.03.002`.

A Raw Data for Experiment

Circle Instance	
Supremum of weight	Time (s)
1100	7.01
1100	6.67
1100	6.85
1200	8.17
1200	7.87
1200	7.94
1300	9.89
1300	9.99
1300	10.11
1400	14.38
1400	14.42
1400	14.08
1500	20.72
1500	21.21
1500	21.59

Table 1: Raw data

Uncorrelated Instance	
Supremum of weight	Time (s)
1100	0.379
1100	0.387
1100	0.382
1200	0.389
1200	0.393
1200	0.393
1300	0.405
1300	0.425
1300	0.4
1400	0.427
1400	0.425
1400	0.425
1500	0.462
1500	0.429
1500	0.44

Table 2: Raw Data

B Scripts for Experiment

B.1 Generating circle instances

```
import random
import math
import sys
n = int(sys.argv[1])
c = sys.argv[2]
b = int(sys.argv[3])
name = sys.argv[4]
fileName = name
fileObj = open(fileName,'w')
fileObj.write(str(n) + "\n")
for i in range(1, n + 1):
    weight = int(random.uniform(1, b))
    circular_val = (2/3)*(math.sqrt((4*b**2) - (weight - 2*b)**2))
    profit = int(circular_val)
    fileObj.write(str(i) + " " + str(profit) + " " + str(weight) + "\n")
fileObj.write(str(c))
```

```
python kp_generate.py 50 10000 b [file-name].py
```

B.2 Computing Run Times

```
cd java
time java comp26120.bnb_kp ../data/[file-name].txt
```