Math 381: Assignment 1: Knapsack Problem

Ella Kim

October 8 2021

1 Introduction

After learning about the common Knapsack Problem, I have been motivated to explore various scenarios of my own knapsack problem that relates to my own life. I have a large container that a client wants to fill up with different valuable works/books they want to buy off of my book store, but it can only hold so much and withstand so much weight. I have 100 unique types of books (i.e, valuable books, where I suppose valuable book i where $1 \leq$ i ≤ 100) with various book Values (a value based on a composite nutrition score), Weights, and Volumes defined with these functions:

$$value = floor(50 + 25 + 25cos(7i))$$

in thousands of dollars,

$$weight = floor(30 + 12cos(6i + 2))$$

in kg, and

$$volume = floor(40 + 8cos(5i + 4))$$

in liters,

where floor truncates to the largest integer $\leq$ real value x.

Similar to the constraints in the lecture, I will also suppose that my knapsack/container has a weight capacity of 400 kg, and volume capacity of 500 liters, which, in other words, would give me the constraints:

$$\sum_{n=1}^{100} w_i x_i \leq W_{max}$$

$$\sum_{n=1}^{100} v_i x_i \leq V_{max}$$

where w_i and v_i are the weight w and volume v of valuable book i with W_{max} and V_{max} of 400 and 500 corresponding to the values above and where x_i indicates if valuable book is put in my container ($x_i = 0$ if valuable book i is not in container, and $x_i = 1$ if valuable book i is in container)

2 Scenario 1

With these equations and constraints, I would like to know how to maximize total value in my container/knapsack with each valuable book i.

As I would like to explore this using lpsolve, I will first put my equations through Python to retrieve the values va_i, w_i and v_i, where va_i is the value of valuable book i. The code below demonstrates the retrieving of each variable type (value, weight, volume) for every valuable book, and printing it in a .txt file to run lpsolve with.

```
# imports required to solve trig functions
import math

# open pre-made txt file to write in
scen1_input = open("sen_one.txt", 'w')

# start LP line for maximizing value
scen1_input.write("max: ")
# initialize a range for i valuable books
ran = range(1, 101)
# for loop for 1<=i<=100 valuable books, solve and
# add each i valuable book x_i and corresponding val_i value
for i in ran:
    # solve value function defined in introduction section
    value = math.floor(50 + 25*math.cos(7*i))
    # add i valuable book and val_i to LP maximized
    scen1_input.write("+" + str(value) + "x" + str(i))
# end max LP equation
scen1_input.write("; \n")

# for style purposes, j valuable book is equivalent to i valuable
# book previously defined
for j in ran:
    # solve weight function defined in introduction
    weight = math.floor(30 + 12*math.cos(6*j + 2))
    # add j valuable book and w_i to weight constraint
    scen1_input.write("+" + str(weight) + "x" + str(j))
# end weight constraint inequality
scen1_input.write(" <= 400; \n")

# for style purposes, k valuable book is equivalent to i valuable
# book previously defined
for k in ran:
    # solve volume function defined in introduction
```

```
    volume = math.floor(40 + 8*math.cos(5*k + 4))
    # add k valuable book and v_i to volume constraint
    scen1_input.write("+" + str(volume) + "x" + str(k))
# end volume constraint inequality
scen1_input.write(" <= 500; \n")

# to follow format, print bin and x1 in binary variable constraint
scen1_input.write("bin x1")
# for loop for 2<=l<=100 valuable books, add each
# l valuable book x_l and corresponding val_l value
# for style purposes, l valuable book is equivalent to i valuable
# book previously defined
for l in range(2, 101):
    # add l valuable book and val_l to binary variables (all)
    scen1_input.write("," + "x" + str(l))
# end binary variable list
scen1_input.write(";")

# read in written in txt file from above
scen1_input = open("sen_one.txt", 'r')

# sanity check
print (scen1_input.read())
# ensure closure of txt file
scen1_input.close()
```

From the equations and inequalities above, the LP in this scenario is:
Maximize:

$$68x_1 + 53x_2 + 36x_3 + ... + 60x_{98} + 43x_{99} + 29x_{100}$$

which is subject to

$$28x_1 + 31x_2 + 34x_3 + ... + 39x_{98} + 37x_{99} + 34x_{100} \leq 400$$

$$32x_1 + 41x_2 + 47x_3 + ... + 34x_{98} + 33x_{99} + 41x_{100} \leq 500$$

$$x_1, x_2, x_3 + ... + x_{98}, x_{99}, x_{100} \in (0, 1)$$

where x_i indicates if valuable book is put in my container.

The above LP will be solved in lpsolve using the text input file written in the Python code. In other words, this LP with its constraints and binary variables will be written as such in the .txt files (without the parenthesis comments):

```
(i 1-100 valuable books with val_i values: LP)
max: +68x_1+53x_2+36x_3+ ... +60x_98+43x_99+29x_100;
(ensure that total weight for for x_i valuable books is less than 400kg)
```

```
+28x_1+31x_2+34x_3+ ... +39x_98+37x_99+34x_100 <= 400;
(ensure that total volume for x_i valuable books is less than 500 liters)
+32x_1+41x_2+ 47x_3+ ... +34x_98+33x_99+41x_100 <= 500
(ensure binary in values for x_i)
bin x1,x2,x3,x4, ... ,x97,x98,x99,x100;
```

I now start my computation of the LP in lpsolve with the line of code below (where -ia represents the removal of $x_i = 0$ variables uses the Python txt file):

```
lp_solve -ia sen_one.txt
```

Which has a resulting output:

```
Value of objective function: 1000.00000000

Actual values of the variables:
x1                              1
x9                              1
x17                             1
x26                             1
x35                             1
x44                             1
x45                             1
x61                             1
x69                             1
x70                             1
x78                             1
x79                             1
x88                             1
x89                             1

All other variables are zero.
```

Therefore, from the output x_i variables with value 1, I would conclude that those i valuable books would yield the max value with the two constraints on weight and volume of my container. In other words, the 1st, 9th, 17th, 26th, 35th, 44th, 45th, 61th, 69th, 70th, 78th, 79th, 88th, and 89th would be what I should fill the container with to get the most amount of money/profit.

3 Scenario 2

Using the same functions for 100 unique valuable books (valuable book i where $1 \leq i \leq 100$) for value, weight, and volume in the Introduction section that I need to move, I will now consider whether the addition of a lower bound for the types of valuable books that I put into my container will affect the output I had in Scenario 1. I want to do this because, while they are valuable and need to be handled with care, I want to clear up some space in my almost-full database of works for newer books, so I want to at least ensure n count number of valuable books will be bought, but also not go over the weight, as recommended by the transit company.

There are several scenarios: if all the books chosen are the smallest, largest, lightest, or heaviest. In Python, I have computed the max possible number of valuable books can be added with those min/max sorted attributes.

```
# imports for using dataframe
import pandas as pd

# initalize list
data = []
# itialize range for the i-th valueable book
ran = range(1, 101)

# for loop: calculate value, weight, and volume
for i in ran:
    value = math.floor(50 + 25*math.cos(7*i))
    weight = math.floor(30 + 12*math.cos(6*i + 2))
    volume = math.floor(40 + 8*math.cos(5*i + 4))
    # after calucalting the i-th valueable book value,
    # weight, volume, add to list as list
    data.append([value, weight, volume])

# change object type to data frame, columns are each attribute
df = pd.DataFrame(data, columns=['Value', 'Weight', 'Volume'])
# sort data frame in descending order by weight
df_w_d = df.sort_values(by=['Weight'], ascending=False)
# sort data frame in ascending order by weight
df_w_a = df.sort_values(by=['Weight'], ascending=True)
# initialize total weight variable
total_weight_d = 0
# initialize total count of books variable
count_d = 0

# for loop, start at heaviest weight
for wei in list(df_w_d["Weight"]):
```

```
    # add to total if under max weight condition
    if total_weight_d <= 400:
        total_weight_d = total_weight_d + wei
        # also increase count of valuable books by 1
        count_d = count_d + 1
# print count of number of heaviest books before hitting the max
# weight
print(count_d-1)


# initialize total weight variable
total_weight_a = 0
# initialize total count of books variable
count_a = 0

# for loop, start at lightest weight
for wei in list(df_w_a["Weight"]):
    # add to total if under max weight condition
    if total_weight_a <= 400:
        total_weight_a = total_weight_a + wei
        # also increase count of valuable books by 1
        count_a = count_a + 1
# print count of number of lightest books before hitting the max
# weight
print(count_a-1)

# sort data frame in descending order by volume
df_v_d = df.sort_values(by=['Volume'], ascending=False)
# sort data frame in ascending order by volume
df_v_a = df.sort_values(by=['Volume'], ascending=True)
# initialize total volume variable
total_vol_d = 0
# initialize total count of books variable
count_d = 0

# for loop, start at largest volume
for vol in list(df_v_d["Volume"]):
    # add to total if under max size condition
    if total_vol_d <= 500:
        total_vol_d = total_vol_d + vol
        # also increase count of valuable books by 1
        count_d = count_d + 1
# print count of number of largest books before hitting the max
# size
print(count_d-1)
```

```
# initialize total volume variable
total_vol_a = 0
# initialize total count of books variable
count_a = 0

# for loop, start at smallest volume
for vol in list(df_v_a["Volume"]):
    # add to total if under max size condition
    if total_vol_a <= 500:
        total_vol_a = total_vol_a + vol
        # also increase count of valuable books by 1
        count_a = count_a + 1
# print count of number of smallest books before hitting the max
# size
print(count_a-1)
```

The output yields the the max number of valuable books before the constraints for weight/volume are hit for:
Heaviest books: 19 (absolute min)
Lightest books: 21 (absolute max)
Largest books: 10 (absolute min)
Smallest books: 15 (absolute max)

While any other combination of 10 books (other than the 10 heaviest) will not go over the weight limit, any other combination of 22 books (other than the 22 most lightest) will go over the weight limit. Similarly, while any other combination of 11 books (other than the 11 largest) will not go over the volume limit, any other combination of 16 books (other than the 16 most smallest) will go over the volume limit. These conditions would explain the absolute max/min notes I have written next to the count outputs. So when focusing purely on weight or volume constraints, the possible lower bounds would be n count, where $10 \leq n \leq 22$ and $11 \leq n \leq 16$ intervals correspond do the weight and volume constraints respectively.

A side note: comparing the absolute minimums (10 vs 11), observing the sorted Python data frames, it can be seen that the heaviest books are not also the largest books, so the absolute minimum/lower bound of the number of valuable books that I can fill the container with is 10. That would also guarantee that I can at least sell 10 books and free up 10 spaces in my database.

These two max lower bounds can be tested in lpsolve, whether they are feasible or not. Constructing similar .txt files as the first one except removing the non-used constraint can be observed here:

Weight constraint: Testing if max 21 is true, Python code for .txt input file:

```
# imports required to solve trig functions
```

```
import math

# open pre-made txt file to write in
scen_input = open("sen_two.txt", 'w')

# start LP line for maximizing value
scen_input.write("max: ")
# initialize a range for i valuable books
ran = range(1, 101)
# for loop for 1<=i<=100 valuable books, solve and
# add each i valuable book x_i and corresponding val_i value
for i in ran:
    # solve value function defined in introduction section
    value = math.floor(50 + 25*math.cos(7*i))
    # add i valuable book and val_i to LP maximized
    scen_input.write("+" + str(value) + "x" + str(i))
# end max LP equation
scen_input.write("; \n")

# for style purposes, j valuable book is equivalent to i valuable book previously
# defined
for j in ran:
    # solve weight function defined in introduction
    weight = math.floor(30 + 12*math.cos(6*j + 2))
    # add j valuable book and w_i to weight constraint
    scen_input.write("+" + str(weight) + "x" + str(j))
# end weight constraint inequality
scen_input.write(" <= 400; \n")

# for style purposes, m valuable book is equivalent to i valuable book previously
# defined
for m in ran:
    # add m valuable book to minimum count constraint
    scen_input.write("+" + "x" + str(m))
scen_input.write(" >= 21; \n")

# to follow format, print bin and x1 in binary variable constraint
scen_input.write("bin x1")
# for loop for 2<=l<=100 valuable books, add each
# l valuable book x_l and corresponding val_l value
# for style purposes, l valuable book is equivalent to i valuable book previously
# defined
for l in range(2, 101):
    # add l valuable book and val_l to binary variables (all)
    scen_input.write("," + "x" + str(l))
# end binary variable list
```

```
scen_input.write(";")

# read in written in txt file from above
scen_input = open("sen_two.txt", 'r')

# sanity check
print (scen_input.read())
# ensure closure of txt file
scen_input.close()
```

The resulting .txt input file is then written as such below (without the parenthesis comments):

```
(i 1-100 valuable books with val_i values: LP)
max: +68x_1+53x_2+36x_3+ ... +60x_98+43x_99+29x_100;
(ensure that total weight for for x_i valuable books is less than 400kg)
+28x_1+31x_2+34x_3+ ... +39x_98+37x_99+34x_100 <= 400;
(ensure total count is lower bounded at 21)
+x1+x2+x3+ ... +x98+x99+x100 >= 21;
(ensure x_i objects/books are binary in count)
bin x1,x2,x3,x4, ... ,x97,x98,x99,x100;
```

Then using lpsolve on the resulting .txt file outputs:

```
lp_solve -ia sen_two.txt

Value of objective function: 1189.00000000

Actual values of the variables:
x16                           1
x17                           1
x18                           1
x19                           1
x20                           1
x39                           1
x40                           1
x41                           1
x42                           1
x43                           1
x44                           1
x60                           1
x61                           1
x62                           1
x63                           1
x64                           1
x84                           1
x85                           1
```

```
x86                             1
x87                             1
x88                             1

All other variables are zero.
```

Because of this output, I therefore conclude that at the max count of weighted valuable books, valuable book 16, 17, 18, 19, 20, 39, 40, 41, 42, 43, 44, 60, 61, 62, 63, 64, 84, 85, 86, 87, and 88 is the max value combination of books to earn the greatest profit.

Tweaking the very last Python code to a ≥ 22 gives the .txt file for testing 22 valuable book counts gives the input file like such (without the parenthesis comments):

```
(i 1-100 valuable books with val_i values: LP)
max: +68x_1+53x_2+36x_3+ ... +60x_98+43x_99+29x_100;
(ensure that total weight for for x_i valuable books is less than 400kg)
+28x_1+31x_2+34x_3+ ... +39x_98+37x_99+34x_100 <= 400;
(ensure total count is lower bounded at 22)
+x1+x2+x3+ ... +x98+x99+x100 >= 21;
(ensure x_i objects/books are binary in count)
bin x1,x2,x3,x4, ... ,x97,x98,x99,x100;
```

The .txt input file then results in a run of lpsolve output like this:

```
lp_solve -ia sen_two.txt

This problem is infeasible
```

The LP is infeasible, meaning there is no combination that will satisfy those constraints, confirming my previous claim of a max lower bound.

Similarly, when replacing the weight constraint with the volume one, and checking the absolute max of 15, it uses this Python for .txt input file (without the parenthesis comments):

```
(i 1-100 valuable books with val_i values: LP)
max: +68x_1+53x_2+36x_3+ ... +60x_98+43x_99+29x_100;
(ensure that total volume for x_i valuable books is less than 500 liters)
+32x_1+41x_2+ 47x_3+ ... +34x_98+33x_99+41x_100 <= 500;
(ensure total count is lower bounded at 22)
+x1+x2+x3+ ... +x98+x99+x100 >= 15;
(ensure x_i objects/books are binary in count)
bin x1,x2,x3,x4, ... ,x97,x98,x99,x100;

lp_solve -ia sen_two.txt
```

```
Value of objective function: 1026.00000000

Actual values of the variables:
x1                              1
x10                             1
x16                             1
x25                             1
x26                             1
x35                             1
x44                             1
x45                             1
x54                             1
x60                             1
x69                             1
x70                             1
x79                             1
x88                             1
x89                             1

All other variables are zero.
```

Because of this output, I therefore conclude that at the max count of size-based valuable books, valuable book 1, 10, 16, 25, 26, 35, 44, 54, 60, 69, 70, 79, 88, and 89 are the max value combination of various sized books to earn the greatest profit.

Tweaking the very last Python code to a ≥ 16 gives the .txt file for testing 16 valuable book counts as below (without the parenthesis comments):

```
(i 1-100 valuable books with val_i values: LP)
max: +68x_1+53x_2+36x_3+ ... +60x_98+43x_99+29x_100;
(ensure that total volume for x_i valuable books is less than 500 liters)
+32x_1+41x_2+ 47x_3+ ... +34x_98+33x_99+41x_100 <= 500;
(ensure total count is lower bounded at 22)
+x1+x2+x3+ ... +x98+x99+x100 >= 16;
(ensure x_i objects/books are binary in count)
bin x1,x2,x3,x4, ... ,x97,x98,x99,x100;
```

This .txt input file then results in a run of lpsolve output like this:

```
lp_solve -ia sen_two.txt

This problem is infeasible
```

The LP is infeasible, meaning there is no combination that will satisfy those constraints, confirming my previous claim of a max lower bound.

Adding a lower bound on the value of the objective function with both conditions did not change the final i valuable books, as it stayed as 15 in count. However, changing the lower bound to 16 made the LP infeasible, so 15 would be the absolute max while maximizing value.

4 Scenario 3

The last scenario I would like to consider, is if I have many copies of an i valuable book, whether that would increase my profit. Using the same functions for 100 unique valuable books (valuable book i where $1 \leq i \leq 100$) for value, weight, and volume in the Introduction section, I will utilize the same Python code as my first scenario, except now x_i is no longer restricted to 0 or 1 in value (binary), and can now many any whole number count (integer), as shown below:

```
# open pre-made txt file to write in
scen_input = open("sen_three.txt", 'w')

# start LP line for maximizing value
scen_input.write("max: ")
# initialize a range for i valuable books
ran = range(1, 101)
# for loop for 1<=i<=100 valuable books, solve and
# add each i valuable book x_i and corresponding val_i value
for i in ran:
    # solve value function defined in introduction section
    # add i valuable book and val_i to LP maximized
    value = math.floor(50 + 25*math.cos(7*i))
    scen_input.write("+" + str(value) + "x" + str(i))
# end max LP equation
scen_input.write("; \n")

# for style purposes, j valuable book is equivalent to i valuable book previously
# defined
for j in ran:
    # solve weight function defined in introduction
    weight = math.floor(30 + 12*math.cos(6*j + 2))
    # add j valuable book and v_i to volume constraint
    scen_input.write("+" + str(weight) + "x" + str(j))
# end volume constraint inequality
scen_input.write(" <= 400; \n")

# for style purposes, k valuable book is equivalent to i valuable book previously
# defined
```

```
for k in ran:
    # solve volume function defined in introduction
    volume = math.floor(40 + 8*math.cos(5*k + 4))
    # add k valuable book and v_i to volume constraint
    scen_input.write("+" + str(volume) + "x" + str(k))
# end volume constraint inequality
scen_input.write(" <= 500; \n")

# to follow format, print int and x1 in binary variable constraint
# for loop for 2<=l<=100 valuable books, add each
# l valuable book x_l and corresponding val_l value
# for style purposes, l valuable book is equivalent to i valuable book previously
# defined
scen_input.write("int x1")
for l in range(2, 101):
    # add l valuable book and val_l to binary variables (all)
    scen_input.write("," + "x" + str(l))
# end binary variable list
scen_input.write(";")

# read in written in txt file from above
scen_input = open("sen_three.txt", 'r')

#sanity check
print (scen_input.read())
# ensure closure of txt file
scen_input.close()
```

This results in a .txt input file (without the parenthesis comments) as below:

```
(i 1-100 valuable books with val_i values: LP)
max: +68x_1+53x_2+36x_3+ ... +60x_98+43x_99+29x_100;
(ensure that total weight for for x_i valuable books is less than 400kg)
+28x_1+31x_2+34x_3+ ... +39x_98+37x_99+34x_100 <= 400;
(ensure that total volume for x_i valuable books is less than 500 liters)
+32x_1+41x_2+ 47x_3+ ... +34x_98+33x_99+41x_100 <= 500;
(ensure x_i objects are whole number in count)
int x1,x2,x3,x4, ... ,x97,x98,x99,x100;
```

This .txt input file then results in a run of lpsolve output like this:

```
lp_solve -ia sen_three.txt

Value of objective function: 1110.00000000
```

```
Actual values of the variables:
x35                            10
x88                             5
```

The consideration of having many copies of the same valuable book resulted in an output with 10 copies of valuable book 35, and 5 copies of valuable book 88. I would therefore conclude that this is the max profit I would be able to earn if I were to sell multiple copies of the same valuable book while fitting the weight and volume constraints of my container.