Assignment 1

MATH 381 A - Winter 2022

Yixuan Liu

January 9, 2022

Introduction

For this assignment, let's suppose we have 100 unique objects which have various values, weights, and volumes, and we have a container that has a weight capacity of 300 kg and volume capacity of 300 liters. We define a binary variable x_i for each unique object which indicates whether the i^{th} object should be put into the container. When the i^{th} object should be put into the container, x_i equals to one.
Assign the 100 objects values a, weights w, and volumes v with the following equations:

$$\begin{aligned} a_i &= floor(50 + 25 \cdot \cos(5i) dollars \\ w_i &= floor(30 + 12 \cdot \cos(4i + 1)) kg \\ v_i &= floor(30 + 12 \cdot \cos(2i + 2)) liters \end{aligned}$$

where $floor$ is a function to truncates a real number to the largest integer less than or equal to that number and i representing the subscript of the variables.

1

First, we want to consider that to maximize the total value of objects, what objects can be put in the container while not exceeding the weight and volume capacity. To solve this problem, we formulate an LP with the objective:

$$\text{Maximize } \textstyle\sum_{i=1}^{100} a_i x_i$$

to maximize the total value in the container, and the following constraints:

$$\sum_{i=1}^{100} w_i x_i \leq 300 \tag{1.1}$$

$$\sum_{i=1}^{100} v_i x_i \leq 300 \tag{1.2}$$

$$x_i \in \{0, 1\} \text{ for all } 1 \leq i \leq 100 \tag{1.3}$$

The first constraint makes the total weight of the objects that should be put in the container not exceeding the weight capacity and the second constraint limit the total volume of the objects in the container below the volume capacity. The third constarint keeps all variables to be binary.
To solve this LP, we will use lpsolve and here is the code written in Java to generate an input file for lpsolve:

```
// This code will generate an lpsolve-formatted LP which solves a knapsack
    problem with unique objects.

import java.util.*;
import java.io.*;

public class A1 {
   // set up 100 objects and a container with specific capacity
   public static final int n = 100;
   public static final int[] value = new int[n];
   public static final int[] weight = new int[n];
   public static final int[] volume = new int[n];
   public static final int weightCapacity = 300;
   public static final int volumeCapacity = 300;

   public static void main(String[] args) throws FileNotFoundException {
      PrintStream input = new PrintStream(new File("a1_input.txt"));

      // assign value, weight, adn volume for each object
      for (int i = 1; i < n+1; i++) {
         value[i-1] = (int)(50+25*Math.cos(5*i));
         weight[i-1] = (int)(30+12*Math.cos(4*i+1));
         volume[i-1] = (int)(30+12*Math.cos(2*i+2));
      }

      // generate lpsolve input file
      // print objective function
      input.print("max: ");
      for (int i = 1; i < n+1; i++) {
         input.print("+" + value[i-1] + "*x_" + i);
      }
      input.println(";");

      // print constriants
      // the sum of objects' weight should not exceed the weight capacity
      for (int i = 1; i < n+1; i++) {
         input.print("+" + weight[i-1] + "*x_" + i);
      }
      input.println("<=" + weightCapacity + ";");
      // the sum of objects' volume should not exceed the volume capacity
```

```
        for (int i = 1; i < n+1; i++) {
           input.print("+" + volume[i-1] + "*x_" + i);
        }
        input.println("<=" + volumeCapacity + ";");

        // define variables to be bianary
        input.print("bin ");
        for (int i = 1; i < n+1; i++) {
           if (i < n) {
              input.print("x_" + i + ",");
           } else {
              input.print("x_" + i + ";");
           }
        }
     }
  }
```

We set up 100 unique objects and assign them with various weights and volumes and print out the objective and constraints. Then we have the output file of this code:

```
max: +57*x_1+...+27*x_100;
+33*x_1+...+35*x_100<=300;
+22*x_1+...+37*x_100<=300;
bin x_1,...,x_100;
```

Then run the lpsolve with the following command that only presenting non-zero variables:

```
lp_solve -ia a1_input.txt
```

and get the output:

```
Value of objective function: 920.00000000

Actual values of the variables:
x_10                          1
x_19                          1
x_24                          1
x_29                          1
x_35                          1
x_44                          1
x_49                          1
x_54                          1
x_63                          1
x_73                          1
x_79                          1
x_88                          1
x_98                          1
```

From the output we get the answer that the 10^{th}, 19^{th}, 24^{th}, 29^{th}, 35^{th}, 44^{th}, 49^{th}, 54^{th}, 63^{rd}, 73^{rd}, 79^{th}, 88^{th}, and 98^{th} objects should be put into the container to maximize the value,

which is 920 dollars. According to the code, we can compute the total weight of these objects is 298 kg and the total volume is 297 liters, which are both less than the capacity, so both these constraints are not binding.

2

Now we want to add a lower bound on the number of objects that we put into the container. Our objective is still the same:

$$\text{Maximize } \sum_{i=1}^{100} a_i x_i$$

which maximize the total value in the container, and we need one more constraint:

$$\sum_{i=1}^{100} w_i x_i \leq 300 \tag{2.1}$$

$$\sum_{i=1}^{100} v_i x_i \leq 300 \tag{2.2}$$

$$\sum_{i=1}^{100} x_i \geq lowerbound \tag{2.3}$$

$$x_i \in \{0, 1\} \text{ for all } 1 \leq i \leq 100 \tag{2.4}$$

The first two constraints still limit the total weight and volume of the objects in the container, and the last constraint makes all variables to be binary. The new constraint makes the total number of objects that should be put into the container greater than a specific number of lower bound.
From the previous part, we found that we need 13 unique objects to maximize the value, so we first choose 14 to be the lower bound to find out whether the new LP is feasible. Here is the new Java code based on the previous one to generate an lpsolve input file to solve the LP:

```
// This code will generate an lpsolve-formatted LP which solves a knapsack
    problem with unique objects and a lower bound.

import java.util.*;
import java.io.*;

public class A1b {
   // set up 100 objects and a container with specific capacity
   public static final int n = 100;
   public static final int[] value = new int[n];
   public static final int[] weight = new int[n];
   public static final int[] volume = new int[n];
   public static final int weightCapacity = 300;
   public static final int volumeCapacity = 300;
   // set up a lower bound
```

```
public static final int low = 14;

public static void main(String[] args) throws FileNotFoundException {
   PrintStream input = new PrintStream(new File("a1b_input.txt"));

   // assign value, weight, adn volume for each object
   for (int i = 1; i < n+1; i++) {
      value[i-1] = (int)(50+25*Math.cos(5*i));
      weight[i-1] = (int)(30+12*Math.cos(4*i+1));
      volume[i-1] = (int)(30+12*Math.cos(2*i+2));
   }

   // generate lpsolve input file
   // print objective function
   input.print("max: ");
   for (int i = 1; i < n+1; i++) {
      input.print("+" + value[i-1] + "*x_" + i);
   }
   input.println(";");

   // print constriants
   // the sum of objects' weight should not exceed the weight capacity
   for (int i = 1; i < n+1; i++) {
      input.print("+" + weight[i-1] + "*x_" + i);
   }
   input.println("<=" + weightCapacity + ";");
   // the sum of objects' volume should not exceed the volume capacity
   for (int i = 1; i < n+1; i++) {
      input.print("+" + volume[i-1] + "*x_" + i);
   }
   input.println("<=" + volumeCapacity + ";");

   // set a lower bound on the total number of objects
   for (int i = 1; i < n+1; i++) {
      input.print("+x_" + i);
   }
   input.println(">=" + low + ";");

   // define variables to be bianary
   input.print("bin ");
   for (int i = 1; i < n+1; i++) {
      if (i < n) {
         input.print("x_" + i + ",");
      } else {
         input.print("x_" + i + ";");
      }
   }
```

```
    }
  }
```

In this revised code we add a lower bound 14 and use this lower bound to print an additional constraints on the total number of objects should be put into the container. Here is the output of lower bound 14:

```
max: +57*x_1+...+27*x_100;
+33*x_1+...+35*x_100<=300;
+22*x_1+...+37*x_100<=300;
+x_1+...++x_100>=14;
bin x_1,...,x_100;
```

Then run the lpsolve with the following command that only presenting non-zero variables:

```
lp_solve -ia a1b_input.txt
```

and get the output:

```
Value of objective function: 907.00000000

Actual values of the variables:
x_4                           1
x_10                          1
x_19                          1
x_24                          1
x_29                          1
x_35                          1
x_38                          1
x_54                          1
x_60                          1
x_63                          1
x_73                          1
x_79                          1
x_82                          1
x_98                          1
```

We now have 14 different objects with a total maximum value of 907 dollars, a total weight of 299 kg, and a total volume of 285 liters. We can see that compare to the solution with 13 objects, the 10^{th}, 19^{th}, 24^{th}, 29^{th}, 35^{th}, 54^{th}, 63^{rd}, 73^{rd}, 79^{th}, and 98^{th} objects are still in the container, the 44^{th}, 49^{th}, and 88^{th} objects have been replaced by the 4^{th}, 38^{th}, 60^{th}, and 82^{nd} objects.
Next, we set the lower bound to 15 and run the code again. Here is the new output of the code:

```
max: +57*x_1+...+27*x_100;
+33*x_1+...+35*x_100<=300;
+22*x_1+...+37*x_100<=300;
+x_1+...++x_100>=15;
bin x_1,...,x_100;
```

Then run the lpsolve with the following command that only presenting non-zero variables:

```
lp_solve -ia a1b_input.txt
```

and get the output:

```
Value of objective function: 875.00000000

Actual values of the variables:
x_10                          1
x_13                          1
x_19                          1
x_24                          1
x_29                          1
x_32                          1
x_35                          1
x_38                          1
x_54                          1
x_57                          1
x_60                          1
x_73                          1
x_79                          1
x_82                          1
x_98                          1
```

The value of 15 unique objects is 875 dollars, their weight is 299 kg, and volume is 299 liters. Comparing to the solution with 14 objects, the 10^{th}, 19^{th}, 24^{th}, 29^{th}, 35^{th}, 38^{th}, 54^{th}, 60^{th}, 73^{rd}, 79^{th}, 82^{nd}, and 98^{th} objects are still in the container, and the 4^{th} object and the 63^{rd} object are replaced by the 13^{rd}, 32^{nd}, and 57^{th} objects. But comparing to the solution with 13 objects, only the 10^{th}, 19^{th}, 24^{th}, 29^{th}, 35^{th}, 54^{th}, 73^{rd}, 79^{th}, and 98^{th} objects are still in the container, and the 44^{th}, 49^{th}, 63^{rd}, and 88^{th} objects are replaced by the 13^{rd}, 32^{nd}, 38^{th}, 57^{th}, 60^{th}, and 82^{nd} objects.
Now we run the code with a lower bound 16:

```
max: +57*x_1+...+27*x_100;
+33*x_1+...+35*x_100<=300;
+22*x_1+...+37*x_100<=300;
+x_1+...++x_100>=16;
bin x_1,...,x_100;
```

Then run the lpsolve with the following command that only presenting non-zero variables:

```
lp_solve -ia a1b_input.txt
```

and get the output:

```
This problem is infeasible
```

Starting from 16, the LP begins to be infeasible, so we can conclude that the maximum lower bound for this problem to be feasible is 15.
By implementing the code, we found that the lower bound can only be less or equal to 15.

To interpreting this result, we go back to the weights and volumes of the objects. We found that the minimum of weights and volumes are both 18 kg or 18 liters, and 17 objects have weight 18 kg while 14 objects have volume 18 liters. Suppose we want to put as much objects as we can into the container by not exceeding the volume capacity. Since there are 9 objects that are both 18 kg and 18 liters, and other 8 objects with 18 kg weight have a volume of 41 liters, we only put the 9 objects with both minimum weight and volume in the container. Next, we can find that the only object with a weight of 19 kg has a volume of 41 liters. Since there are 8 objects with weight 18 kg and volume 41 liters, we will not put it in the container until we put all 18 kg objects in. The third smallest value of weight is 21 kg, and there are 4 objects have a 21 kg weight and a 18 liters volume, the other 3 objects that are 21 kg have both 41 liters volume, so we only put the 4 objects with the minimum volume in the container. Now the container has 13 objects with a total weight of 246 kg and a total volume of 234 liters. At this point, we can only put at most 54 kg of objects, which indicates that each object would have an average weight of 18 kg. Previously we found that the remaining objects with weights of 18 kg all have volumes of 41 liters. However, we only have 66 liters of room left in the container, so there is no way to put 16 objects into the container without not exceeding the weight capacity. So 15 is the biggest lower bound such that the LP is feasible. For lower bound less than 13, the solution of 13 unique objects always satisfies the lower bound, so for all lower bound less than 13, the LPs are feasible. There is no way to put more than 15 objects in the container.

3

Starting now, with no lower bound anymore, we will now treat each uique object as a class of objects in unlimited supply. We can still use the first code, but re-define the variables: x_i is now non-binary and can be any non-negative integer, which represents the amount of the i^{th} type of the objects. For our LP, the objective is still the same:

$$\text{Maximize } \sum_{i=1}^{100} a_i x_i$$

which maximize the total value in the container, and we rewrite the constraints:

$$\sum_{i=1}^{100} w_i x_i \leq 300$$

$$\sum_{i=1}^{100} v_i x_i \leq 300$$

$$x_i \in \mathbb{N}_0 \text{ for all } 1 \leq i \leq 100$$

The first two constraints still limit the total weight and volume of the objects in the container, but the last constraint makes all variables to be non-negative integers. Now the new Java code to generate the new lpsolve input file should be:

```
// This code will generate an lpsolve-formatted LP which solves a knapsack
   problem with non-unique objects.
```

```java
import java.util.*;
import java.io.*;

public class A1c {
   // set up 100 objects and a container with specific capacity
   public static final int n = 100;
   public static final int[] value = new int[n];
   public static final int[] weight = new int[n];
   public static final int[] volume = new int[n];
   public static final int weightCapacity = 300;
   public static final int volumeCapacity = 300;

   public static void main(String[] args) throws FileNotFoundException {
      PrintStream input = new PrintStream(new File("a1c_input.txt"));

      // assign value, weight, adn volume for each object
      for (int i = 1; i < n+1; i++) {
         value[i-1] = (int)(50+25*Math.cos(5*i));
         weight[i-1] = (int)(30+12*Math.cos(4*i+1));
         volume[i-1] = (int)(30+12*Math.cos(2*i+2));
      }

      // generate lpsolve input file
      // print objective function
      input.print("max: ");
      for (int i = 1; i < n+1; i++) {
         input.print("+" + value[i-1] + "*x_" + i);
      }
      input.println(";");

      // print constriants
      // the sum of objects' weight should not exceed the weight capacity
      for (int i = 1; i < n+1; i++) {
         input.print("+" + weight[i-1] + "*x_" + i);
      }
      input.println("<=" + weightCapacity + ";");
      // the sum of objects' volume should not exceed the volume capacity
      for (int i = 1; i < n+1; i++) {
         input.print("+" + volume[i-1] + "*x_" + i);
      }
      input.println("<=" + volumeCapacity + ";");

      // define variables to be any non-negative integer
      input.print("int ");
      for (int i = 1; i < n+1; i++) {
         if (i < n) {
            input.print("x_" + i + ",");
```

```
            } else {
               input.print("x_" + i + ";");
            }
         }
      }
   }
```

Instead of printing the last constraint as to make all variables to be either 0 or 1, we now allow x_i to be any non-negative integer. We get the output:

```
max: +57*x_1+...+27*x_100;
+33*x_1+...+35*x_100<=300;
+22*x_1+...+37*x_100<=300;
int x_1,...,x_100;
```

Then run the lpsolve with the following command that only presenting non-zero variables:

```
lp_solve -ia a1c_input.txt
```

and get the output:

```
Value of objective function: 1184.00000000

Actual values of the variables:
x_54                          16
```

The lpsolve shows that the solution is to put 16 of the 54^{th} type of objects into the container to maximize the value, which is 1184 dollars, which is totally different from what we have talked about previously. Why would this happen? By examining the values of the 100 types of objects, the maximum value is 74 dollars. Among all the objects that are 74 dollars, the 10^{th}, 54^{th}, and 98^{th} types of objects are the three types of objects that have both the minimum weight and minimum volume. It is interesting why the lpsolve solve this problem with x_{54}, not x_{10} nor x_{98}. So I decide to rearrange the sequnce of the constraint to see whether it will make a difference. However, the result was still the same. According to Dr. Matthew Conroy, "the complexities of the lp-solving process are such that the choice is quite unpredictable; lpsolve will always only give one optimal solution, even if there are many, and which one it gives is not really predictable." So I add a line to the lpsolve input file:

```
max: +57*x_1+...+27*x_100;
+33*x_1+...+35*x_100<=300;
+22*x_1+...+37*x_100<=300;
int x_1,...,x_100;
thisisalabel: x_54=0;
```

After running the lpsolve again, the new solution is:

```
Value of objective function: 1184.00000000

Actual values of the variables:
x_98                          16
```

Now our solution is to have 16 of the 98^{th} type of objects with the same total value. If we solve this LP by hand, whether to choose 16 of the 10^{th} type of objects, 16 of the 54^{th} type of objects, 16 of the 98^{th} type of objects, or any combination of them can always lead to a solution.