SinglePurchaseOptimizer

July 14, 2020

1 Optimizing Stardew Valley Purchases with Linear Programming

1.1 Introduction

Stardew Valley is a managerial game in which the protagonist is the owner of a farm. Here, far far away from the chaothic city, the player can plant different seeds and harvest all the goods that is able to produce. The player can also manage a ranch or some other activities. While playing, at some point during the game play, in everyone's mind an eternal question popped: "*How can i become the richest farmer in Pellican Town and marry that hot chick, Leah?*"

In this notebook we will see how to optimize all the purchases that the player will make while buying seeds, applying some sweet sweet **Linear Programming**.

1.2 Modelization

In stardew valley, every plant can be bought in an emporium for a given cost. Every crop has a different price and different growing time. When a crop has completed its growing cycle, it can be harvested and sold, generating a revanue. In this context, different resources must be considered. - *money* - Someone said somewhere: >The person who doesn't know where his next dollar is coming from usually doesn't know where his last dollar went. > Of course money is a resource. We have little money, and we want to make a lot of money. The game starts with a very very small amount, and it's our duty to harvest it and become richer and richer (and marry Leah).

- *space* - The space that we have at our disposal in the farm it's not infinite. Every crop will use one tile, and sometimes this space must be sacrified for other objects like scarerows or splinkers. Sadly, every penny in the world is not enough when no space can be used.
- *time* - I imagine that somewhere in the internet there's another good quote, but i'm too lazy. In Stardew Calley, time is essential, since every crop needs several days to complete its growth cycle. However, there is another limitation. The year is divided in 4 months of 28 days, one for each season. Every season has a set of allowed plants that can grow; if a green cabbage plant lasts till the end of the spring, it will be atrociously killed by the mighty spears of the summer Sun.

1.2.1 Elements of the modelization

Constants P : the set of all the cultures. p_i is the i-th crop N : the number of available tiles D : number of days available M : total amount of money available

t_i: time needed by the i-th culture to grow c_i: cost of the i-th culture r_i: revenue of the i-th culture β_i : days needed for products to regrow. If the crop does not regraw, it's set to -1

if $\beta_i \neq -1$ $\alpha_i = \lfloor \frac{D}{\beta_i} \rfloor$: number of times that a crop will produce in the D days **else** $\alpha = 1$

max_i: maximum number of seeds allowed for the culture min_i: minimum number of seeds allowed for the culture These two can be set to $+\infty$ and 0, but can also be used in order to have some diversifiaction.

Variables x_i: amount of the i-th culture that needs to be purchased.

Objective function $maximize\ \pi = \sum_{i\in P} \alpha_i r_i x_i - \sum_{i\in P} c_i x_i$

Constaints **tile availablity**; *the number of products cannot exceed the number of plantable tiles* $\sum_{i\in P} x_i \leq N$

time horizon constraint; *If there is not enough time for this culture, it should not be planted* **if** $t_i > D \rightarrow x_i = 0$

total cost constraint; *The total cost cannot exceed the total amount of money owned* $\sum_{i\in P} c_i x_i \leq M$

the real problem is integer, but can be relaxed to linear if the amount of money is much larger $\forall i \in P \quad x_i \geq 0, \in NaturalNumbers$

1.3 Code of the optimizer

The following class is ued to store all the information regarding a crop, such buy cost, sell cost, growth time and season in which the seed can be planted. The following lines of code retrieve these information from a JSON file.

[1]:
```
class Crop:

    def __init__(self, name, buyCost, sellCost, growthTime):
        self.name = name
        self.buyCost = buyCost
        self.sellCost=sellCost
        self.growthTime = growthTime
        self.seasons = []
        self.regrow=-1

    def addSeason(self, season):
        self.seasons= self.seasons +(season)

    def setRegrowTime(self, regrow):
        self.regrow = regrow;

    def __str__(self):
        stringa ="Name={}\nBuy={}\nSell={}\ngrothTime={}".format(self.name,␣
↪self.buyCost, self.sellCost, self.growthTime)
        if(self.regrow != -1):
            stringa= stringa +("\nregrow=" + str(self.regrow))
        stringa = stringa + "\nseasons:" + str(self.seasons)
        return stringa
```

[2]:
```
import json
from math import *
from pulp import *
```

[3]:
```
jsonFile = open("resources/stardewConfig.json", "r")
data = json.load(jsonFile)

crops = data.get("crops")
```

[4]:
```
cropList = []
for crop in crops:
    x = Crop(crop.get("name"), crop.get("buy"), crop.get("sell"), sum([x for x␣
↪in crop.get("stages")]))
    if(crop.get("regrow")):
        x.setRegrowTime(crop.get("regrow"))
    x.addSeason(crop.get("seasons"))
    cropList.append(x)
    #print(x)
    #print("\n")
```

The followings are the input of the program.

N : the number of available tiles D : number of days available M : total amount of money available

[5]:
```
D = 30
N = 120
M = 10000
```

if $\beta_i \neq -1$ $\alpha_i = \lfloor \frac{D}{\beta_i} \rfloor$: number of times that a crop will produce in the D days **else** $\alpha = 1$

```
[6]: alpha= {}
     for crop in cropList:
         if crop.regrow == -1:
             alpha[crop.name] =1
         else:
             alpha[crop.name] = floor(D/crop.regrow)
```

Creation of the variables. The name of the variable is the name of the crop.

```
[7]: variables = {}
     for crop in cropList:
         variables[crop.name]= LpVariable(crop.name, 0, cat="Integer")
```

```
[8]: prob = LpProblem("ProfitMaximization", LpMaximize)
```

maximize $\pi = \sum_{i \in P} \alpha_i r_i x_i - \sum_{i \in P} c_i x_i$

```
[9]: prob += lpSum([alpha[x.name]*x.sellCost*variables[x.name] for x in cropList])␣
      ↪-lpSum([x.buyCost*variables[x.name] for x in cropList])
```

tile availablity $\sum_{i \in P} x_i \leq N$

```
[10]: prob += lpSum([variables[x.name] for x in cropList]) <= N
```

time horizon constraint if $t_i > D \rightarrow x_i = 0$

```
[11]: for crop in cropList:
          if(crop.growthTime > D):
              prob += variables[crop.name] == 0
```

total cost constraint $\sum_{i \in P} c_i x_i \leq M$

```
[12]: prob += lpSum([x.buyCost*variables[x.name] for x in cropList]) <= M
```

```
[13]: status = prob.solve()
```

```
[14]: status
```

```
[14]: 1
```

```
[15]: LpStatus[status]
```

```
[15]: 'Optimal'
```

[16]:
```
for v in prob.variables():
    if value(v)> 0:
        print(v.name,"=", value(v))
```

```
Ancient_Fruit = 4.0
Hops = 116.0
```

[17]:
```
print("profit =",value(prob.objective))
```

```
profit = 86040.0
```

[18]:
```
prob
```

[18]:
```
ProfitMaximization:
MAXIMIZE
1500*Ancient_Fruit + 270*Blueberry + 195*Coffee_Bean + 200*Corn + 690*Hops +
360*Hot_Pepper + 170*Melon + 40*Poppy + 50*Radish + 160*Red_Cabbage +
350*Starfruit + 40*Summer_Spangle + -45*Sunflower + 370*Tomato + 15*Wheat + 0
SUBJECT TO
_C1: Ancient_Fruit + Blueberry + Coffee_Bean + Corn + Hops + Hot_Pepper
 + Melon + Poppy + Radish + Red_Cabbage + Starfruit + Summer_Spangle
 + Sunflower + Tomato + Wheat <= 120

_C2: 700 Ancient_Fruit + 80 Blueberry + 30 Coffee_Bean + 150 Corn + 60 Hops
 + 40 Hot_Pepper + 80 Melon + 100 Poppy + 40 Radish + 100 Red_Cabbage
 + 400 Starfruit + 50 Summer_Spangle + 125 Sunflower + 50 Tomato + 10 Wheat
 <= 10000

VARIABLES
0 <= Ancient_Fruit Integer
0 <= Blueberry Integer
0 <= Coffee_Bean Integer
0 <= Corn Integer
0 <= Hops Integer
0 <= Hot_Pepper Integer
0 <= Melon Integer
0 <= Poppy Integer
0 <= Radish Integer
0 <= Red_Cabbage Integer
0 <= Starfruit Integer
0 <= Summer_Spangle Integer
0 <= Sunflower Integer
0 <= Tomato Integer
0 <= Wheat Integer
```