motivation for the problem, formulation, methodology used to solve the problem, and a few lessons that you learned while working on the project

Dietary Planning System for Optimization Problem

Zeren Li, Bomin Zhang

# Abstract

# Introduction

# Motivation

Have you ever tried of planning for what you should eat for today’s lunch? Yes. it happens to us a lot. Sometimes, we wonder whether there exists some dietary planning application that could directly gives us at least some suggestion of the food to make for just one meal or three meals for a day. However, the most of the related application in the market only provides service of searching recipes along with the ingredients and time needed, and nutritional value, etc. But, they do not provide a personalized recipe suggestion for one. The reason is not only that the preference of a dish for one cannot be predetermined, but also that the time, and cost expected to make a dish can vary a lot for different one. Thus, this gives us a huge motivation for making a dietary planning system subject to some constraints like time, cost, etc.

# Formulation

We constructed the table as following:

Recipe table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | x1 | x2 | ... | xi | ... | xn |
| R1 | Q11 | Q12 |  |  |  | Q1n |
| R2 |  |  |  |  |  |  |
| ... |  |  |  |  |  |  |
| Rk |  |  |  |  |  |  |
| ... |  |  |  |  |  |  |
| Rd | Qd1 |  |  |  |  | Qdn |

, where {x1,...,xn} is set of raw ingredient, {R1,...,Rd} is set of recipe.

Nutrition table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | x1 | x2 | ... | xi | ... | xn |
| v1 | M11 |  |  |  |  | M1n |
| v2 |  |  |  |  |  |  |
| ... |  |  |  |  |  |  |
| vj |  |  |  |  |  |  |
| ... |  |  |  |  |  |  |
| Vm |  |  |  |  |  | Mmn |

, where {V1,...,Vm} is set of nutrition to consider.

Cost table

|  |  |
| --- | --- |
|  | Cost |
| x1 | c1 |
| x1 | c2 |
| ... | ... |
| xi | ci |
| ... | ... |
| xn | cn |

we can therefore deride Matrix A which describes recipes and their corresponding nutritional values by multiply the , that is Matrix, and has form:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | v1 | v2 | ... | vj | ... | vm |
| R1 | A11 | A12 |  |  |  | A1n |
| R2 |  |  |  |  |  |  |
| ... |  |  |  |  |  |  |
| Rk |  |  |  |  |  |  |
| ... |  |  |  |  |  |  |
| Rd | Ad1 |  |  |  |  | Adn |

Secondly, we can multiply the nutrition table Q and matrix Cost table to get recipes’ cost. Let capital letter Ck denotes cost of recipe k.

Thus, we can construct the following Matrix that shows relationship of recipes and their cooking time, cost, and preferences.

|  |  |  |  |
| --- | --- | --- | --- |
|  | time | Cost | Preference |
| R1 | t1 | C1 | p1 |
| R2 |  |  |  |
| ... |  |  |  |
| Rk |  |  |  |
| ... |  |  |  |
| Rd | td | Cd | pd |

Each recipe has its own time to make, the preference is rated from 1-5 and should be different according to one’s preference.

Lastly, we have a nutritional requirement vector called that specify how much we need daily for each category of nutrition.

Thus, system has

Input: preference, total cost I, total cooking time T, daily nutritional requirement W, and the matrix A that shows recipes’ nutritional value.

Output: a vector , denotes the quantity of every recipe to make.

Finally, optimization can be formulated as follows:

**objective:**

we want to

, where = is a vector denotes the quantity of the recipe to make and denotes preference for recipe k.

**Subject to:**

1. Cost constraint:

, where

Explanation: let capital letter denotes cost of recipe k, and denotes cost of ingredient i. So, each recipe costs and total cost for all recipe has to be less than letter I(investment).

1. Nutritional constraint:

above constraint equivalent to

where W= is mX1 vector, denoting the minimum nutritional requirement. , , and .

Explanation: W specify the nutritional requirements. gave us recipes’ nutritional value. We multiple that with the output vector to yield the result for every nutritional category. Make sure that value is greater than a threshold wj.

1. time constraint:

, where T denotes total time. we have to make all dishes within appropriate time. is time required for making dish k.

This is a convex optimization problem because the objective function is affine. The constraints are affine as well.

Furthermore, we also added some complexity into the vanilla problem by considering

1. We made the prediction for breakfast, lunch, dinner(three meals). In addition, we set constraint for the time of breakfast, lunch, dinner separately.
2. if we add exponential decay to the preference. That is, the preference gain for additional duplicated dish decreases exponentially.

For case 1), we can reformulate the problem as follows:

Obj: we want to

, where denote the outputs for three meals respectively.

subject to:

1. Cost constraint:
2. Nutritional constraint:
3. time constraint:

,where are the time limit for three meals respectively.

For case 2), we want to add a exponential decay for the preference gain. In the formula of exponential decay, we have

f(x)=a , where

f(x) denotes the value after the decay, a is initial amount.

d is called decay factor

(1-d) is the decay factor multiplier

x is the number of time interval. In our case, it is the number of dishes that have already appeared.

For example, suppose the preference for a cake is 5, decay factor is 0.4, and we have it continuous for three times. Initial preference gain is 5, preference gain for duplicated cake is 5\*(1-0.4) and preference gain for the third cake is 5\*(1-0.4)2. So, the total preference gain is a 5+5\*0.6+5\*0.62, which is a geometric series.

From geometric series,

This suggests that with exponential decay our objective function in the optimization problem became:

, where d\_m denotes the decay factor multiplier.

The rest of the constraints are unchanged.

However, this means our optimization problem become a MINLP (mixed integer non-linear program), since the objective function is no longer linear. So, it cannot be solved easily from linear solver then.

# Methodology

## Pre-processing

## Implementation

The optimization process is implemented using python language. For our problem, we have two notebook files to run the solver and they are called linear\_prog\_solver.ipynb and Non-linear-Solver-Gekko.ipynb. The linear solver is imported from python library called PULP and for the non-linear problem we used another library called Gekko to solve. Gekko is a software specifically designed to solve problem for IP, LP, NLP, MILP, MINLP. And it’s very accurate and time efficient.

For the nutritional vector W, it specifies the nutritional requirement for a day not for a meal. To solve the problem for a meal, we directly divide the W vector by 3 to get the average nutritional requirement for just one meal.

# Experiments

in the vanilla problem, we loaded three tables into the workspace and set some input variables like time, cost. The

# Conclusion

We learned that an optimization problem might be considered easy first, but become more complicated in reality. Just like in our problem, the vanilla problem is an easy LP program. However, in reality, there are many factors that could essentially affect the way we construct the problem. That is, if we consider the preference gain decreases as dishes get duplicated, which it does in reality, then the problem is no longer an LP program and can provide a more interesting and valuable results when we actually use it. Therefore, the simulation of an optimization in reality might be really complicated after taking into account what other factors that can come into play. However, after all, we are very satisfied with the performance of our dietary planning system, since it met our expectation of designing such a software.

# Reference

Beal, L.D.R., Hill, D.C., Martin, R.A., & Hedengren, J.D. (2018). GEKKO Optimization Suite. Processes, 6(8), 106. [https://doi.org/10.3390/pr6080106](https://doi.org/10.3390/pr6080106" \t "https://chatgpt.com/c/_new)