Creating a diet plan using Simplex Method and Google OR-Tools

CS5800 Group2, Cong(Cathy)Fu

Group members: Jieling Gong, Zilei Liu

Context and Problem Statement

Many of the problems we encountered in the Algorithms class are optimization tasks: the shortest path between two nodes, a spanning tree with a minimal total length, etc. [1]. We seek a solution to these problems that 1) satisfies certain constraints; and 2) is the best possible, with respect to some well-defined criterion, among all solutions. When both the constraints and the optimization criterion are linear functions, Linear Programming techniques (we use the Simplex Method here due to its effectiveness and practicality) can be used. In reality, an enormous number of problems can be expressed in this way.

The problem I chose to solve (I will call it Cathy’s Diet Problem) is to *create a diet plan that satisfies basic nutritional need constraints, with a minimal cost, using real-life data.* This topic is inspired by the famous “Stigler Diet Problem” proposed by Economics Nobel laureate George Stigler[2], and a paper called “The Diet Problem” created by George Dantzig, the father of linear programming, himself[3].

Introduction

The Stigler’s Diet Problem[2] was developed around the 1940s, before the Simplex Method was proposed by Dantzig in 1947. In Stigler’s Diet Problem, he considered 9 nutritional needs (Calories, Protein, Calcium, Iron, Vitamin A, Vitamin B1, Vitamin B2, Niacin, and Vitamin C) as constraints(Appendix 1-a) and considered the price and nutrient data of 77 kinds of food(Appendix 2-a). The objective function to minimize is the total cost of the necessary food that satisfies the nutrition constraints above. Stigler used the food price in 1939 in US cents, and the nutritional data in each cell is per dollar instead of per food unit, so the result will tell us how much US dollars we should spend on each food item, instead of the weight of the food item we should consume. This design makes it very hard to “compare apples to apples” in Canada or any other countries, in 2024. What’s more, each item on Stigler’s food list could have different units, for example, lb, oz, bunch, dozen, and pt.

Comparing to The Stigler's Diet Problem, Cathy's Diet Problem has several interesting features:

First of all, I have reliable and up-to-date sources of the food price and nutrient values data(Appendix 2-b), as well as the recommended daily values of nutritional needs(Appendix 1-b).

- The nutrient data of each food item is taken from the Canadian Nutrient (CNF) API : https://produits-sante.canada.ca/api/documentation/cnf-documentation-en.html
- The food price data is taken from "Raw materials price index for crop products, animals and animal products, monthly" (https://www150.statcan.gc.ca/t1/tbl1/en/tv.action?pid=1810026803) and I will be using the food price of May, 2024. All the food prices will be in Canadian Cents, per 100 gram of the food.
- The daily nutrient values recommended data is taken from
    - "Nutrition labelling – Table of daily values"(https://www.canada.ca/en/health-canada/services/technical-documents-labelling-requirements/table-daily-values/nutrition-labelling.html) and
    - "Dietary reference intakes tables: Reference values for macronutrients" (https://www.canada.ca/en/health-canada/services/food-nutrition/healthy-eating/dietary-reference-intakes/tables/reference-values-macronutrients.html).

Secondly, I have unified all the units of my food items and food price items (Appendix 2-b). I am using "100g" as the serving size for all food items, as all the nutrient data uses "100g" as the standard food amount. For some liquid items, I searched the average density of the liquid and converted volumes to weights. For the food prices, which are usually in "Canadian dollars/kilograms," I converted them to "Canadian cents/100g" so both the nutrient values and the food prices can be placed in the same cell of the table.

Thirdly, I am using an open-source software suite for optimization called Google OR-Tools. The solver I am using, Glop, is Google's in-house implementation of the primal and dual simplex methods. Glop is open-source and trusted for Google's production workloads[5]. The documentation of Google OR-Tools offers sample code and detailed explanations, making it very convenient for users. Glop solves my problem efficiently(it processes 15 iterations in less than one second).

Fourthly, readers of this report can use and modify my code according to their personal needs. For example, users can adjust the daily calorie/protein consumption according to their height, weight, age, and activity levels. Readers can also add more food choices and update the current prices of those foods. My code can also be used in developing mobile apps or websites, which could help users create personal diet plans online.

Analysis

The problem to solve

There are 20 food items available in unlimited quantities, each with a fixed price(Appendix 2-b). We can purchase and consume any amount of these items to meet our daily nutritional needs, which include 12 specific daily values(Appendix 1-b). It is considered healthy to consume more than the recommended values. We aim to determine the minimal cost required to achieve this.

We will represent this by a linear program, using matrix-vector notation:

Objective function	$\min \mathbf{c}^T\mathbf{x}$
Constraints	$\mathbf{Ax} \geq \mathbf{b}$
	$\mathbf{x} \geq 0$

$\mathbf{c}^T$ is the vector of food prices per serving (100g).This is given.

$$\mathbf{c}^T = \left[price_1, price_2, price_3, \ldots, price_{20}\right]$$

$\mathbf{x}$ is the vector of the amount (how many servings) we should purchase and consume, for each food item. This is part of our solution.

$$\mathbf{x} = \begin{bmatrix} amount_1 \\ amount_2 \\ \ldots \\ amount_{20} \end{bmatrix}$$

A is the matrix of nutrient value in 100g of each food item. This is given.

$n_1 f_1$ denotes nutrient 1 in 100g of food 1.

$$\mathbf{A} = \begin{bmatrix} n_1 f_1, n_1 f_2, \dots, n_1 f_{20} \\ n_2 f_1, n_2 f_2, \dots, n_2 f_{20} \\ \dots \\ n_{12} f_1, n_{12} f_2, \dots, n_{12} f_{20} \end{bmatrix}$$

b is the vector of nutrient needs for an average person per day. This is given.

$$\mathbf{b} = \begin{bmatrix} nutrient_1 \\ nutrient_2 \\ \dots \\ nutrient_{12} \end{bmatrix}$$

To summarize, we want to minimize

$$\text{total\_cost} = \begin{bmatrix} price_1, price_2, price_3, \dots, price_{20} \end{bmatrix} \begin{bmatrix} amount_1 \\ amount_2 \\ \dots \\ amount_{20} \end{bmatrix}$$

So that

$$\begin{bmatrix} n_1 f_1, n_1 f_2, \dots, n_1 f_{20} \\ n_2 f_1, n_2 f_2, \dots, n_2 f_{20} \\ \dots \\ n_{12} f_1, n_{12} f_2, \dots, n_{12} f_{20} \end{bmatrix} \begin{bmatrix} amount_1 \\ amount_2 \\ \dots \\ amount_{20} \end{bmatrix} \geq \begin{bmatrix} nutrient_1 \\ nutrient_2 \\ \dots \\ nutrient_{12} \end{bmatrix}$$

Our solution will be the total cost and the amount of each food item we should consume.

A python solution

Below I will display my code blocks and explain the steps I take to solve this problem using glop. The complete python program can be found in a Github repo in Appendix 3.

Step 1: Import the linear solver wrapper

```
from ortools.linear_solver import pywraplp
```

Step2: Gather data for the problem (12 constraints and 20 variables).
The nutrients array contains our constraint vector **b**.
The foods_data array contains both our nutrient-food matrix **A** and price vector $\mathbf{c}^T$**.**
(Actually this array contains $\mathbf{A}^T$, so both the price data and the nutrient data can stay in the same array, sharing the same row.)

```
# Nutrient minimums for an average person

nutrients = [
        ["Calories (cal)", 2000],
        ["Protein (g)", 46],
        ["Calcium (g)", 1.3],
        ["Iron (mg)", 18],
        ["Vitamin B6 (mg)", 1.7],
        ["Vitamin B12 (ug)", 2.4],
        ["Niacin (mg)", 16],
        ["Vitamin C (mg)", 90],
        ["Fibre (g)", 28],
        ["Sodium (mg)", 2300],
        ["Potassium (mg)", 3400],
        ["Vitamin D (ug)", 20],
    ]
```

```
# The foods_data is in the following format:
# food_name, unit, May 2024 price, Calories (cal), Protein (g),Calcium (g),
# Iron (mg),  Vitamin B6 (mg), Vitamin B12 (ug), Niacin (mg),
# Vitamin C (mg), Fibre (g), Sodium (mg),Potassium (mg), Vitamin D (ug)

    foods_data = [
        ["Wheat Flour", "100g", 20.88, 369, 14.639, 26.441, 3.396, 0.213,
0, 4.93, 0, 2.7, 2, 107, 0],
        ["White Rice", "100g", 64.66, 357, 14.73, 21, 1.96, 0.391, 0,
6.733, 0, 0.4, 1, 35, 0],
        ["Milk", "100g", 30.10, 61, 3.15, 113, 0.03, 0.036, 0.45, 0.089, 0,
0, 44, 150, 0.025],
        ["Butter", "100g", 127.75, 717, 0.85, 24, 0.16, 0.003, 0.13, 0.042,
0, 0, 11, 24, 1.5],
        ["Bacon", "100g", 126.8, 417, 12.62, 5, 0.41, 0.266, 0.5, 4.022, 0,
0, 1717, 565, 1.05],
        ["Apple", "100g", 54.7, 52, 0.26, 6, 0.12, 0.041, 0, 0.091, 4.6,
2.4, 1, 107, 0],
        ["Carrots", "100g", 30.96, 41, 0.93, 33, 0.3, 0.138, 0, 0.983, 5.9,
2.8, 69, 320, 0],
        ["Ground Beef", "100g", 125.5, 207, 19.575, 10.2, 1.8, 0.2375,
2.35, 5.375, 0, 0, 67, 218, 0.125],
        ["Salmon", "100g", 265.2, 142, 19.84, 12, 0.8, 0.818, 3.18, 7.86,
0, 0, 59, 363, 13.15],
        ["Pork", "100g", 88.4, 255, 26.29, 5, 0.92, 0.31, 0.44, 4.311, 0.3,
0, 62, 423, 1.325],
        ["Orange", "100g", 40.8, 49, 1.04, 40, 0.09, 0.063, 0, 0.274, 48.5,
2.4, 0, 181, 0],
        ["Potato", "100g", 49.8, 78, 2.38, 8, 1.01, 0.256, 0, 1.68, 14.2,
1.7, 3, 484, 0],
        ["Cabbage", "100g", 28.4, 16, 1.2, 77, 0.31, 0.232, 0, 0.4, 27,
1.2, 18, 170, 0],
        ["tomato", "100g", 43, 16, 1.16, 5, 0.47, 0.06, 0, 0.593, 16, 0.9,
42, 212, 0],
        ["Banana", "100g", 16.5, 89, 1.09, 5, 0.26, 0.367, 0, 0.665, 8.7,
1.74, 1, 358, 0],
        ["Green beans", "100g", 56.13, 39, 1.977, 31.117, 0.738, 0.115, 0,
0.558, 29.321, 2.376, 20.133, 171.592, 0],
        ["Chicken", "100g", 60.1, 124, 17.88, 10, 5.86, 0.42, 11.41, 6.662,
16.2, 0, 77, 228, 0.05],
        ["Cheese", "100g", 135.8, 371, 23.24, 674, 0.43, 0.065, 1.26,
0.118, 0, 0, 560, 136, 0.5],
        ["grapes", "100g", 92.3, 67, 0.063, 14, 0.29, 0.11, 0, 0.3, 4, 0.9,
2, 191, 0],
        ["Yogurt", "100g", 69, 50, 4.6, 147, 0.12, 0.032, 0.24, 0.075, 0.5,
0, 40, 174, 0],
    ]
```

Step3: Instantiate a glop solver

We name the solver, and declare an array to hold our solutions (vector **x**)

```
solver = pywraplp.Solver.CreateSolver("GLOP")
    if not solver:
        return

    food_amounts = [solver.NumVar(0.0, solver.infinity(), item[0]) for item
in foods_data]
```

Step4: Set up the constraints

For each nutrient we need to consume at least the amount of the daily value (We do not consider an upper bound here, for simplicity).

```
constraints = []

    for i, nutrient in enumerate(nutrients):

        constraints.append(solver.Constraint(nutrient[1],
solver.infinity()))

        for j, item in enumerate(foods_data):
            # item[i + 3] is the amount of nutrient i in the food item j
per 100g.
            constraints[i].SetCoefficient(food_amounts[j], item[i + 3])
```

Step5: Setup the objective function

we multiply the price per 100g of the food item and the amount we need to purchase.

```
objective = solver.Objective()

    food_prices = [item[2] for item in foods_data]

    for i, food_amount in enumerate(food_amounts):
        objective.SetCoefficient(food_amount, food_prices[i])

    objective.SetMinimization()
```

Step6: Invoke the solver and display the result

In the print statements, we convert the total price to Canadian dollars and the amount to kilograms.

```
status = solver.Solve()

       if status != solver.OPTIMAL:
        print("The problem does not have an optimal solution!")
        if status == solver.FEASIBLE:
            print("A potentially suboptimal solution was found.")
        else:
            print("The solver could not solve the problem.")
            exit(1)

    nutrients_result = [0] * len(nutrients)

    print("\nDaily Foods consumption (kg):")
for i, food_amount in enumerate(food_amounts):
        if food_amount.solution_value() > 0.0:
            print("{}: {:.4f} kg".format(foods_data[i][0], 1/10 *
food_amount.solution_value()))
            print("Costs: ${:.2f}".format(food_prices[i]/ 100 *
food_amount.solution_value()))
            for j, _ in enumerate(nutrients):
                nutrients_result[j] += foods_data[i][j + 3] *
food_amount.solution_value()
```

```
    print("\nOptimal daily price: ${:.2f}".format(objective.Value()/100))

    print("\nNutrients per day:")
    for i, nutrient in enumerate(nutrients):
        print(
            "{}: {:.2f} (min {})".format(nutrient[0], nutrients_result[i],
nutrient[1])
        )
```

Step7 : Read and interpret the result.

```
Daily Foods consumption (kg):
Wheat Flour: 0.4074 kg
Costs: $0.85
Bacon: 0.1280 kg
Costs: $1.62
Salmon: 0.1419 kg
Costs: $3.76
Orange: 0.0137 kg
Costs: $0.06
Banana: 0.9580 kg
Costs: $1.58

Optimal daily price: $7.87

Nutrients per day:
Calories (cal): 3098.25 (min 2000)
Protein (g): 114.54 (min 46)
Calcium (g): 184.55 (min 1.3)
Iron (mg): 18.00 (min 18)
Vitamin B6 (mg): 5.89 (min 1.7)
Vitamin B12 (ug): 5.15 (min 2.4)
Niacin (mg): 42.80 (min 16)
Vitamin C (mg): 90.00 (min 90)
Fibre (g): 28.00 (min 28)
Sodium (mg): 2300.00 (min 2300)
Potassium (mg): 5129.01 (min 3400)
Vitamin D (ug): 20.00 (min 20)
```

Conclusion

In this report, I addressed a real-life diet problem involving 20 food variables and 12 nutrient constraints. Utilizing the knowledge from our Algorithms class and the powerful Google OR-Tools, I formulated and solved this problem as a linear programming model. This process allowed me to review course materials, deepen my understanding of linear programming, enhance my data-processing skills, learn a new tool, and improve my overall problem-solving abilities. The experience was both enjoyable and rewarding.

I also acknowledge several limitations in the data collection and analysis process. Some missing data were supplemented using Google search, and certain data points are approximate due to variations in product types, such as different apple brands and rice varieties. The price data, sourced from the “Raw Materials Price Index,” may differ from actual supermarket prices. Additionally, only 12 constraints were considered, though more nutritional factors should ideally be included. Due to time constraints, the food item list is limited, but it can be expanded with advanced data methods. My future work should address these limitations to enhance the accuracy and comprehensiveness of the analysis. Moreover, I am eager to utilize my knowledge on web/mobile development and create an app using this optimization algorithm to help my users create their own diet plans.

Appendix

1-a) Stigler's 9 Constraints

Nutrient	Daily Recommended Intake
Calories	3,000 calories
Protein	70 grams
Calcium	.8 grams
Iron	12 milligrams
Vitamin A	5,000 IU
Thiamine (Vitamin B1)	1.8 milligrams
Riboflavin (Vitamin B2)	2.7 milligrams
Niacin	18 milligrams
Ascorbic Acid (Vitamin C)	75 milligrams

1-b) Cathy's 12 Constraints

Daily Values

Nutrient	Recommended Daily Intake	
Calories (cal)	2000	
Protein (g)	46	# if male, 56
Calcium (g)	1.3	
Iron (mg)	18	
Vitamin B6 (mg)	1.7	
Vitamin B12 (ug)	2.4	
Niacin (mg)	16	
Vitamin C (mg)	90	
Fibre (g)	28	
Sodium (mg)	2300	
Potassium (mg)	3400	
Vitamin D (ug)	20	

2-a Stigler’s food items (partial)

Commodity	Unit	1939 price (cents)	Calories (kcal)	Protein (g)	Calcium (g)	Iron (mg)	Vitamin A (KIU)	Thiamine (mg)	Riboflavin (mg)	Niacin (mg)	Ascorbic Acid (mg)
Wheat Flour (Enriched)	10 lb.	36	44.7	1411	2	365	0	55.4	33.3	441	0
Macaroni	1 lb.	14.1	11.6	418	0.7	54	0	3.2	1.9	68	0
Wheat Cereal (Enriched)	28 oz.	24.2	11.8	377	14.4	175	0	14.4	8.8	114	0
Corn Flakes	8 oz.	7.1	11.4	252	0.1	56	0	13.5	2.3	68	0
Corn Meal	1 lb.	4.6	36.0	897	1.7	99	30.9	17.4	7.9	106	0
Hominy Grits	24 oz.	8.5	28.6	680	0.8	80	0	10.6	1.6	110	0
Rice	1 lb.	7.5	21.2	460	0.6	41	0	2	4.8	60	0
Rolled Oats	1 lb.	7.1	25.3	907	5.1	341	0	37.1	8.9	64	0

2-b Cathy’s food items

Food , price, and nutrient values

Commodity	Unit	Price/Unit (Canadian cents/ 100g)	Calories (cal)	Protein (g)	Calcium (g)	Iron (mg)	Vitamin B6 (mg)	Vitamin B12 (ug)	Niacin (mg)	Vitamin C (mg)	Fibre (g)	Sodium (mg)	Potassium (mg)	Vitamin D (ug)
Wheat Flour	100g	20.88	369	14.639	26.441	3.396	0.213	0	4.93	0	2.7	2	107	0
White Rice	100g	64.66	357	14.73	21	1.96	0.391	0	6.733	0	0.4	1	35	0
Milk	100g	30.10	61	3.15	113	0.03	0.036	0.45	0.089	0	0	44	150	0.025
Butter	100g	127.75	717	0.85	24	0.16	0.003	0.13	0.042	0	0	11	24	1.5
Bacon	100g	126.8	417	12.62	5	0.41	0.266	0.5	4.022	0	0	1717	565	1.05
Apple	100g	54.7	52	0.26	6	0.12	0.041	0	0.091	4.6	2.4	1	107	0
Carrots	100g	30.96	41	0.93	33	0.3	0.138	0	0.983	5.9	2.8	69	320	0
Ground Beef	100g	125.5	207	19.575	10.2	1.8	0.2375	2.35	5.375	0	0	67	218	0.125
Salmon	100g	265.2	142	19.84	12	0.8	0.818	3.18	7.86	0	0	59	363	13.15
Pork	100g	88.4	255	26.29	5	0.92	0.31	0.44	4.311	0.3	0	62	423	1.325
Orange	100g	40.8	49	1.04	40	0.09	0.063	0	0.274	48.5	2.4	0	181	0
Potato	100g	49.8	78	2.38	8	1.01	0.256	0	1.68	14.2	1.7	3	484	0
Cabbage	100g	28.4	16	1.2	77	0.31	0.232	0	0.4	27	1.2	18	170	0
tomato	100g	43	16	1.16	5	0.47	0.06	0	0.593	16	0.9	42	212	0
Banana	100g	16.5	89	1.09	5	0.26	0.367	0	0.665	8.7	1.74	1	358	0
Green beans	100g	56.13	39	1.977	31.117	0.738	0.115	0	0.558	29.321	2.376	20.133	171.592	0
Chicken	100g	60.1	124	17.88	10	5.86	0.42	11.41	6.662	16.2	0	77	228	0.05
Cheese	100g	135.8	371	23.24	674	0.43	0.065	1.26	0.118	0	0	560	136	0.5
grapes	100g	92.3	67	0.063	14	0.29	0.11	0	0.3	4	0.9	2	191	0
Yogurt	100g	69	50	4.6	147	0.12	0.032	0.24	0.075	0.5	0	40	174	0

3 full code and my GitHub repository:

https://github.com/cathyfu1215/foodPlannerProject/blob/main/cathy_diet.py

Reference

[1]S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani, *Algorithms*. London: McGraw-Hill Publishing, 2006. Pp. 202-203

[2]"The Stigler Diet Problem | OR-Tools," *Google Developers*. https://developers.google.com/optimization/lp/stigler_diet

[3] Dantzig, G. B. (1990). The Diet Problem. *Interfaces*, *20*(4), 43–47. http://www.jstor.org/stable/25061369

[4]"OR-Tools Examples," *Google for Developers*. https://developers.google.com/optimization/examples (accessed Jul. 31, 2024).

[5]"Advanced LP Solving | OR-Tools," Google for Developers. https://developers.google.com/optimization/lp/lp_advanced (accessed Aug. 02, 2024).