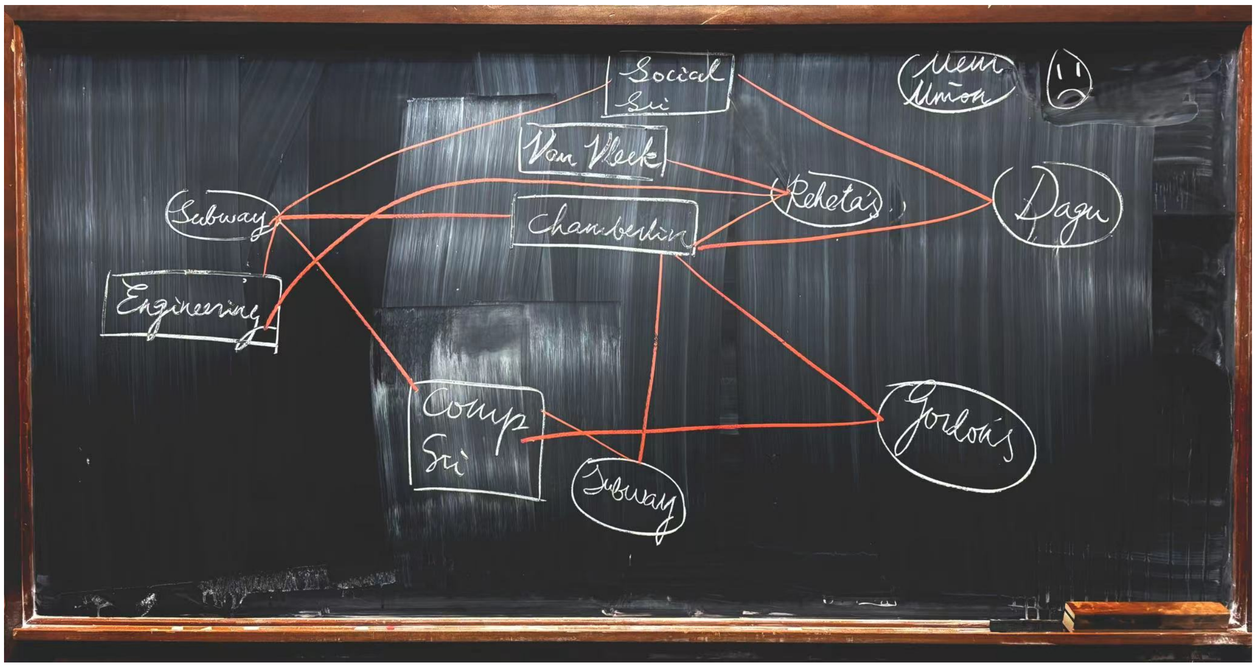
A Strategist's Guide to Campus Dining:
A Multifactor Optimization Approach

Yufei Shen, Harry Luo

June, 1925



A Typical Week of Lunch Choices; No one visits Mem Union :(

Contents

1 Introduction

A Chinese wisdom has been driving generations of innovators and entrepreneurs: “Time is money; efficiency is life.” This principle, first proclaimed during China’s economic reforms in 1981, resonates deeply with the daily reality of college students navigating campus life. Indeed, time and efficiency have become the constant pursuit of students hurrying between classes, strategically timing their meals to avoid crowded dining halls, and calculating the optimal schedules across campus to maximize precious minutes between lectures. This project is our answer to this challenge. We optimize for weekly lunch decisions given a fixed schedule and daily targets. The daily targets determine how we balance food cost and commuting time for the day. The best food plan would also satisfy our nutritional needs, which is used to check the validity of a solution. Our decision would also be affected by discounts and deals, as well as our enjoyment level from having different kinds of food.

We then utilize the solution to the optimization formulation, or the “optimal plan”, under different daily targets, to explore the trade-offs one needs to make if they are more concerned with commuting time or the cost of food, respectively. We are also interested in knowing if there exists a “Restaurant of the Week” that is chosen by a significant portion of people, regardless of their different daily targets. Finally, we also explore whether consistent optimal restaurant choices emerge over the week, and how these patterns are influenced by nutritional constraints

The main part of the project will be construct into 5 parts:

Part I – Problem Data: Setting the Stage. We assemble the inputs and parameterize the environment:

(i) Restaurant and food options with prices (including time-limited discounts) and basic nutrition;

(ii) Course schedule and transit times from varying starting points;

(iii) Discounts and deals as scenario parameters;

(iv) Enjoyment and agility scores capturing cuisine preferences and flexibility.

Part II – Modeling. We formalize the daily restaurant assignment as an optimization problem: define decision variables, objectives, and constraints. We then add a nutrition feasibility check as higher-level constraints.

Part III – Computational Solution. We implement the model in Julia (JuMP/Gurobi), describe the simulation setup, and outline code design choices that ensure reproducibility.

Part IV – Results and Discussion. We analyze solutions under three preference profiles (time-saving, budget-saving, balanced), quantify the cost of a healthy diet, and examine the emergence of an optimal plan. We then discuss the effects of assumptions on shaping the computational output.

Part V – Conclusion and Future Directions. We summarize practical takeaways for students with fixed schedules and sketch extensions.

2 Problem Data: Setting the Stage

2.1 Restaurant and Food Options

Based on a survey of a sample size = 2 (the two authors), a UW-Madison student might frequent the following list of restaurants (table 1), to which we denote from R1 to R6. For simplicity, we assume that each restaurant only serves one food option. Food prices and nutrition facts are gathered with the help of generative AI. In particular, there are two Subway stores near campus: R1 (Subway) and R2 (Subway 2). To make them distinct, we assume that Subway 2 only serves Tuna Sandwichs ($15).

Nutrient / $\mathbf{R_j}$	$\mathbf{R_1}$ Subway	$\mathbf{R_2}$ Subway2	$\mathbf{R_3}$ Dagu	$\mathbf{R_4}$ Rehta	$\mathbf{R_5}$ Gordon	$\mathbf{R_6}$ Union
Price ($)	11.59	15.00	19.50	10.00	13.25	17.25
Carb (g)	102	128	100	90	100	110
Protein (g)	41	323	26	45	25	34
Fat (g)	35	40	30	20	25	28
Total Calories (kCal)	975	1025	890	665	775	800

Table 1: Price and macronutrient content of meals offered at each restaurant(Union stand for Memorial Union).

2.2 Course Schedule and Transit Time

A schedule for a typothetical undergraduate student is generated below, with total time cost of dining at restaurant R_j and commuting to the subsequent building (Table 2).

Day / $\mathbf{R}_j$	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**
1 (Chamberlin$\rightarrow$Social Sci.)	13	13	26	14	26	19
2 (Van Vleck$\rightarrow$Chamberlin)	13	13	23	13	23	19
3 (Engineering$\rightarrow$CompSci.)	6	13	34	27	23	34
4 (Van Vleck$\rightarrow$Engineering)	13	15	30	21	25	27
5 (Social Sci.$\rightarrow$CompSci.)	9	16	29	19	23	23

Table 2: Daily transit times (min) between lecture halls and restaurants, according to a weekly schedule.

2.3 Discounts and Deals

A dollar saved is a dollar earned. We design a deal policy for certain restaurants, motivated by common discount events. These policies are divided into two categories: fixed daily discounts and conditional discounts that depend on the visit history.

- **Fixed Daily Discounts:** These are promotions offered by specific restaurants on certain days of the week.
  - **Restaurant 1:** 50% discount on Thursday (*Thursday Madness*).
  - **Restaurant 3:** 20% discount on odd-numbered days (i.e., Day 1, 3, and 5).
  - **Restaurant 4:** 12% discount on even-numbered days (i.e., Day 2 and 4).
- **Conditional Visit-Based Discounts:** These are special deals earned based on the pattern of visits to a restaurant each week.
  - **Restaurant 2:** A 20% discount is applied on the second day of any two consecutive-day visits.
  - **Restaurant 5:** A 35% discount is applied on the *first* visit.
  - **Restaurant 6:** A 16% discount is applied on the *second* visit.

2.4 Enjoyment and Agility

Good food changes people. Different food options would also affect our happiness, with higher happiness in turn making us more agile (walks faster). To model this, we introduce a daily "Enjoyment Index," denoted by E_d, which is a state variable that evolves based on dining choices. The student begins on Day 1 with a baseline Enjoyment Index of $E_1 = 1$. Each day, the choice of restaurant modifies the index for the following day. The specific enjoyment values contributed by each restaurant are as follows:

- **Restaurant 3:** +2 (Dagu makes one happy)
- **Restaurant 4:** +1
- **Restaurant 2, 6:** 0
- **Restaurant 1, 5:** -1 (Subway and Gordon's make one sad)

The happier we are, the faster we walk (or even skip!). This Enjoyment Index, E_d, directly influences the student's agility through a time multiplier, ϕ_d. The effective travel time cost for day d is scaled by this multiplier, becoming $\phi_d \times \text{time}_{d,r}$. A lower ϕ_d represents a "speed buff," making travel feel shorter.

The relationship between E_d and ϕ_d is modeled as a continuous, piecewise linear function defined as:

$$\phi_d = \begin{cases} 1.0 & \text{if } 0 \leq E_d \leq 1, \\ 1.1 - 0.1 \cdot E_d & \text{if } 1 \leq E_d \leq 6. \end{cases} \tag{1}$$

This means that for every point of enjoyment gained above 1, the student receives a 10% "speed buff" (a 0.1 reduction in travel time), up to a maximum buff of 50% when the Enjoyment Index reaches 6.

3 Modeling

3.1 The Daily Restaurant Assignment Model

We set out to find the best weekly sequence of restaurant choices that balances monetary expenses and commuting time, while also accounting for dynamic factors like deals and enjoyment.

Conceptually, this problem can be visualized as finding an optimal path through a decision graph, as shown in Figure 1. Each node $D_{d,r}$ represents the choice of restaurant r on day d. The goal is to select one node for each of the five days, forming a "path" that represents the weekly dining plan.

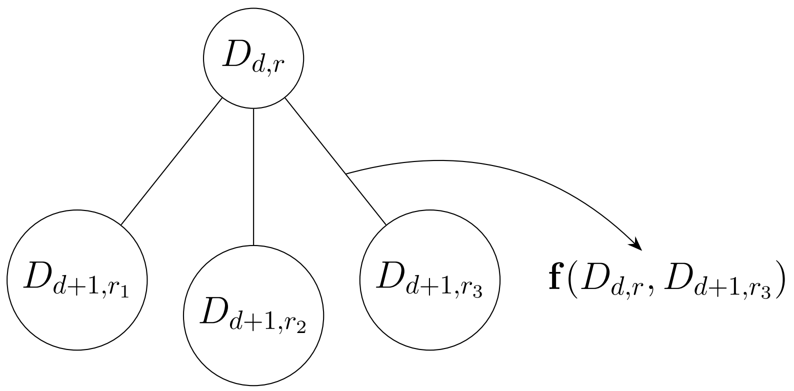


Figure 1: A segment of the restaurant decision graph, from day d to $d + 1$.

We formulate the problem algebraically using an assignment-based model that directly reflects the implementation.

Sets and Parameters

- $D = \{1, \dots, 5\}$: The set of days.
- $R = \{1, \dots, 6\}$: The set of available restaurants.
- $t_{d,r}$: The travel time to restaurant r on day d.
- p_r: The base price of a meal at restaurant r.
- $c_{d,r}$: The price of a meal at restaurant r on day d after applying any *fixed daily discounts.*
- e_r: The enjoyment value gained from visiting restaurant r.
- $\mathbf{W}$: A 5×2 matrix where $W_{d,1}$ and $W_{d,2}$ are the non-negative weights for time and money on day d, respectively.

Decision Variables

- $x_{d,r} \in \{0, 1\}$: A binary variable, equal to 1 if restaurant r is chosen on day d, and 0 otherwise.
- $E_d \in [0, 6]$: A continuous variable representing the Enjoyment Index on day d.
- $\phi_d \geq 0$: A continuous variable for the time multiplier on day d, defined in eq. 1
- $y_{2,d}, y_{5,d}, y_{6,d} \in \{0, 1\}$: Auxiliary binary variables to indicate if a specific conditional deal for restaurants 2, 5, or 6 is activated on day d.

Objective Function

The objective is to minimize the total weighted cost, which is the sum of weighted time and money costs, less the value of any activated conditional deals.

$$\min \quad \underbrace{\sum_{d \in D} \sum_{r \in R} W_{d,1}\, \phi_d\, t_{d,r}\, x_{d,r}}_{\text{Weighted Time Cost}} + \underbrace{\sum_{d \in D} \sum_{r \in R} W_{d,2}\, c_{d,r}\, x_{d,r}}_{\text{Weighted Money Cost}} - \Omega, \tag{2}$$

where Ω represents the total weighted value of all conditional deals, broken down by restaurant:

$$\begin{aligned} \Omega = \quad & \underbrace{\sum_{d=2}^{5} W_{d,2}(0.20 p_2) y_{2,d}}_{\text{R2 Deal: Consecutive Days}} \\ & + \underbrace{\sum_{d \in D} W_{d,2}(0.35 p_5) y_{5,d}}_{\text{R5 Deal: First Visit}} \\ & + \underbrace{\sum_{d \in D} W_{d,2}(0.16 p_6) y_{6,d}}_{\text{R6 Deal: Second Visit}} . \end{aligned}$$

Constraints

- **Daily Choice:** We must select exactly one restaurant each day.

$$\sum_{r \in R} x_{d,r} = 1 \quad \forall d \in D$$

- **Visit Limit:** A specific restaurant cannot be visited more than twice during the week.

$$\sum_{d \in D} x_{d,r} \leq 2 \quad \forall r \in R$$

- **Enjoyment Dynamics:** The Enjoyment Index starts at 1 and evolves based on daily choices. The time multiplier ϕ_d is linked to E_d via the piecewise linear function defined in Sec 2.4, which is enforced using SOS2 constraints.

$$\begin{aligned} E_1 &= 1 \\ E_{d+1} &= E_d + \sum_{r \in R} e_r x_{d,r} \quad \forall d \in \{1, \dots, 4\} \end{aligned}$$

- **Conditional Deal Logic:** The auxiliary deal variables are linked to the primary choice variables. For the "two days in a row" deal for Restaurant 2, the constraints are:

$$\begin{aligned} y_{2,d} &\leq x_{d,2} \quad \forall d \in \{2, \dots, 5\} \\ y_{2,d} &\leq x_{d-1,2} \quad \forall d \in \{2, \dots, 5\} \\ y_{2,d} &\geq x_{d,2} + x_{d-1,2} - 1 \quad \forall d \in \{2, \dots, 5\} \end{aligned}$$

  This set of constraints ensures $y_{2,d}$ is 1 if and only if both $x_{d,2}$ and $x_{d-1,2}$ are 1. Similar logic is applied to model the first-visit deal for R5 ($y_{5,d}$) and the second-visit deal for R6 ($y_{6,d}$).

3.2 The Nutrition Feasibility Check

Busy as we are, we still have great care for healthy diets. We predefine a set of nutritional goals for each week, and for any given path $\mathbf{p}$, we must verify if it meets the nutrition goals.

Let:

- $\mathbf{n}_{d,r}$: A vector of nutritional values (e.g., calories, protein) associated with the transition on day d to r on day $d+1$.
- $\mathbf{N}_{\text{lower}}$, $\mathbf{N}_{\text{upper}}$: Bound vectors for the total nutritional values over the 5-day period.

The total nutritional value for a path $\mathbf{p}$ is the sum of the nutrition vectors of its constituent arcs. Let $A(\mathbf{p})$ be the set of arcs corresponding to path $\mathbf{p}$.

$$\mathbf{N}_{\text{total}}(\mathbf{p}) = \sum_{(d,r) \in A(\mathbf{p})} \mathbf{n}_{d,r}.$$

The path $\mathbf{p}$ is **nutritionally feasible** if and only if:

$$\mathbf{N}_{\text{lower}} \leq \mathbf{N}_{\text{total}}(\mathbf{p}) \leq \mathbf{N}_{\text{upper}}.$$

This is a component-wise vector inequality.

4 Computational Solution

4.1 Simulation and Code Design

We consider two degrees of freedom in this project.

Firstly, we consider students of different dining preferences, namely: people who only prioritize having food with the least budget (*Money first*), people who only prioritize having food that takes the least amount of time (*Time first*), and those who have relatively balanced preferences (*Arbitrary*), whose preference profile will be generated at random.

Secondly, we consider the effect of nutrition constraints. One can certainly have healthy diets on campus, but we ask the question "What, then, must we give in return?". We explore this by running the model with the same *Arbitrary* profile from above, but first without considering nutrition constraints, and then using Nutrition Feasibility Check discussed in Sec 3.2.

Finally, we explore a restaurant pattern across 1,000 simulations initiated with random profiles and then analyze it for an optimal dining plan.

The mathematical model was implemented in the Julia programming language, using the `JuMP` package for algebraic modeling and `Gurobi` as the Mixed-Integer Linear Programming (MILP) solver. We outline the implementation below, with the complete code available in the Appendix.

4.1.1 The Core Model

The core script solves the primary optimization problem **without** considering nutritional constraints. The implementation closely mirrors the mathematical formulation. Full code is attached in Appendix: Code 1: No Nutrition Check.

We begin by initializing the model. In particular, we first initiate the 5-by-2 weight matrix $\mathbf{W}$ that governs the preference between time and money cost over a week using seeded random. (see 2.4).

```julia
Random.seed!(1)
baseW  = [rand() for _ in 1:DAYS]

# The weight of each day will sum up to 1
weights = hcat( round.(baseW; digits=2),
                round.(1 .- baseW; digits=2) )
```

We then load the fixed discounts and deals for each restaurant:

```
price_base = fill(0.0, DAYS, RESTAURANTS)
for d in 1:DAYS, r in 1:RESTAURANTS
    f = 1.0  # default: no discount
    # If restaurant 1 on day 4, apply 50% price
    f = (r == 1 && d == 4)  ? 0.50 : f
    # If restaurant 3 on an odd-numbered day, apply 20% off (pay 80%)
    f = (r == 3 && isodd(d)) ? 0.80 : f
    # If restaurant 4 on an even-numbered day, apply 12% off (pay 88%)
    f = (r == 4 && iseven(d)) ? 0.88 : f
    # Final price for this (day, restaurant)
    price_base[d, r] = price[r] * f
end
```

We then introduce the decision variables:

```
@variable(m, x[1:DAYS, 1:RESTAURANTS], Bin) # dine / day,rest.
@variable(m, 0 <= E[1:DAYS] <= 6)
@variable(m, phi[1:DAYS] >= 0)
@variable(m, y2[2:DAYS], Bin)  # r2 two days in a row
@variable(m, y5[1:DAYS], Bin)  # first r5 visit
@variable(m, y6[1:DAYS], Bin)  # second r6 visit
```

The objective function is then constructed from expressions representing weighted time, weighted base money cost, and weighted deal discounts (Eq. 2).

```
@expression(m, expr_time,
    sum(weights[d,1] * phi[d] * time[d,r] * x[d,r] for d=1:DAYS, r=1:RESTAURANTS))

# Direct cost (fixed deal included in constraint)
@expression(m, expr_money_base,
    sum(weights[d,2] * price_base[d,r] * x[d,r] for d=1:DAYS, r=1:RESTAURANTS))

# Conditional deal
@expression(m, expr_money_deals,
      sum(weights[d,2] * 0.20 * price[2] * y2[d] for d=2:DAYS) +
      sum(weights[d,2] * 0.35 * price[5] * y5[d] for d=1:DAYS) +
      sum(weights[d,2] * 0.16 * price[6] * y6[d] for d=1:DAYS))

@objective(m, Min, expr_time + expr_money_base - expr_money_deals)
```

Key constraints are implemented to enforce daily choices, restaurant visit limits, and the dynamics of the Enjoyment Index and its effect on the time multiplier (as discussed in sec 2.4).

```
@constraint(m, [d=1:DAYS], sum(x[d,r] for r=1:RESTAURANTS) == 1) # Choose one arc
↪  for each day
@constraint(m, [r=1:RESTAURANTS], sum(x[d,r] for d=1:DAYS) <= 2) # Restrict each
↪  restaurant 2 time per week

@constraint(m, E[1] == 1) # Initial enjoyment

for d in 1:DAYS-1 # Inventory balance for enjoyment
    @constraint(m, E[d+1] == E[d] + sum(enjoy_val[r] * x[d,r] for r=1:RESTAURANTS))
end

# SOS-2 approximation for phi(E)
@variable(m, 0 <= lambda[1:DAYS, 1:length(knot_E)] <= 1)

for d in 1:DAYS
    @constraint(m, sum(lambda[d,:]) == 1)
    @constraint(m, E[d] == sum(knot_E[k] * lambda[d,k] for k in 1:length(knot_E)))
    @constraint(m, phi[d] == sum(knot_factor[k] * lambda[d,k] for k in
    ↪  1:length(knot_E)))
    @constraint(m, lambda[d,:] in SOS2())
end
```

The conditional deal logic for ‘y2’, ‘y5’, and ‘y6’ variables is implemented using a combination of linear constraints to activate the binary deal variables.

Restaurant 2: two days in a row discount is implemented as follows.

```
for d in 2:DAYS
    @constraint(m, y2[d] <= x[d,2])  # z2 can be 1 only if we choose r=2 on day d
    @constraint(m, y2[d] <= x[d-1,2])  # and only if we chose r=2 on day d-1
    # if both days chose r=2, force y2[d]=1
    @constraint(m, y2[d] >= x[d,2] + x[d-1,2] - 1)
end
```

Restaurant 5: first visit discount is implemented as follows.

```
@constraint(m, sum(y5) <= 1)  # allow marking at most one first-visit day
for d in 1:DAYS
    @constraint(m, y5[d] <= x[d,5])
    if d > 1
        @constraint(m, y5[d] + sum(x[k,5] for k=1:d-1) <= 1)# y5 can only be 1 on
        ↪  the first day r=5 appears
    end
end
```

Restaurant 6: second visit discount is implemented as follows.

```
@constraint(m, sum(y6) <= 1) # can mark at most one second-or-later day, caps total
↪  visits at <= 2
for d in 1:DAYS
    @constraint(m, y6[d] <= x[d,6])
    @constraint(m, y6[d] <= sum(x[k,6] for k=1:d-1))
    if d > 1
        @constraint(m, y6[d] >= x[d,6] + sum(x[k,6] for k=1:d-1) - 1)
    else
        @constraint(m, y6[d] == 0) # cannot be the "second" on day 1
    end
end
```

4.1.2 The Nutritious Addition

We now consider the nutrition validation as an addition to the core model.

When the optimal path $\mathbf{p}^*$ from the cost minimization problem is found to be **nutritionally infeasible**, it must be discarded. We then need to find the next-best path and repeat the check. This creates an iterative algorithm where we solve the optimization problem, check for feasibility, and if the path fails, add a constraint to exclude it from future consideration.

This iterative process, which incorporates the nutrition feasibility check and “no-good“ cuts, is implemented using a ‘while’ loop that repeatedly solves the MILP and adds a constraint if the nutrition targets are not met (Algorithm 1). This part of the code corresponds to the second script provided in Appendix: Code 2: *With* Nutrition Check. A snippet is converted from 1 to Julia below.

Algorithm 1 Iterative Path Finding with Nutritional Feasibility Check

```
1: while a nutritionally feasible path has not been found do
2:     Solve the current MILP formulation to find the optimal path, $\mathbf{p}_k$.
3:     if the MILP is infeasible then
4:         Terminate: No solution exists that satisfies all constraints.
5:         return "No feasible path found"
6:     end if
7:     Let $S_k$ be the set of arcs corresponding to path $\mathbf{p}_k$.
8:     Calculate the total nutrition vector $\mathbf{N}_{\text{total}}(\mathbf{p}_k) = \sum_{(d,r,r')\in S_k} \mathbf{n}_{d,r,r'}$.
9:     if $\mathbf{N}_{\text{lower}} \leq \mathbf{N}_{\text{total}}(\mathbf{p}_k) \leq \mathbf{N}_{\text{upper}}$ (component-wise) then
10:        Terminate: Path $\mathbf{p}_k$ is nutritionally feasible.
11:        return $\mathbf{p}_k$ as the final, optimal, and feasible solution.
12:    else
13:        Add "no-good" cut to the current solution to exclude path $\mathbf{p}_k$: $\sum_{(d,r)\in S_k} x_{d,r} \leq 4$
14:                                        ▷ This cut ensures $\mathbf{p}_k$ will not be chosen again.
15:    end if
16: end while
```

```julia
cut_cnt   = 0
tot_nutr  = zeros(4)
while cut_cnt < 30 # Avoid infinite loop
    optimize!(m)
    if termination_status(m) != MOI.OPTIMAL
        println("m infeasible or not optimal.")
        break
    end
    # We will find the suboptimal solution if the optimal solution does not fulfill
    ↪  the nutrition constraints, repeat the process to find the optimal solution
    ↪  fulfill the constraints.
    chosen = [(d,r) for d in 1:DAYS, r in 1:RESTAURANTS if value(x[d,r]) > 0.5]
    tot_nutr = sum(nutrition[r,:] for (_ ,r) in chosen)
    viol_low  = max.(0.0, nutr_lower .- tot_nutr)
    viol_high = max.(0.0, tot_nutr .- nutr_upper)
    violated  = any(viol_low  .> 1e-6) || any(viol_high .> 1e-6)
    if violated
        cut_cnt += 1
        @constraint(m, sum(x[d,r] for (d,r) in chosen) <= DAYS - 1)
    else
        println("----------Nutrition satisfied after cut_cnt cut(s)----------")
        break
    end
end
```

5 Results and Discussion

5.1 Three Different Dining Preferences

5.1.1 Raw Results

We begin exploring the results by first comparing the optimal plan **without nutrition constraints**, with three different dining preferences:

- **Profile 1: Only Prioritize Money over Time (Money First)**

  We manually set the preference weight matrix $\mathbf{W}_{i,:} = [0 \quad 1]$, $\forall i$ (100% emphasis on money).

  Raw results:

```
Day-by-day details:
  Day 1 -> R4   (E = 1, Speed buff = 0.0%)  |  Time = 14.0 min  Money = $10.0
  Day 2 -> R4   (E = 2, Speed buff = 10.0%)  |  Time = 11.7 min  Money = $8.8
  Day 3 -> R1   (E = 3, Speed buff = 20.0%)  |  Time = 4.8 min  Money = $11.75
  Day 4 -> R1   (E = 2, Speed buff = 10.0%)  |  Time = 11.7 min  Money = $5.88
  Day 5 -> R5   (E = 1, Speed buff = 0.0%)  |  Time = 23.0 min  Money = $8.61

WEIGHTED objective value = 45.04
Weekly nutrition [kcal carb prot fat] = [4055, 484, 197, 135]

---- Unweighted totals ----
Total travel time  : 65.2 min
Total money spent  : $45.04
```

- **Profile 2: Only Prioritize Time over Money (Time First)**

  We manually set the preference weight matrix $\mathbf{W}_{i,:} = [1 \quad 0]$, $\forall i$ (100% emphasis on time).

  Raw results:

```
Day-by-day details:
  Day 1 -> R4   (E = 1, Speed buff = 0.0%)  |  Time = 14.0 min  Money = $10.0
  Day 2 -> R4   (E = 2, Speed buff = 10.0%)  |  Time = 11.7 min  Money = $8.8
  Day 3 -> R1   (E = 3, Speed buff = 20.0%)  |  Time = 4.8 min  Money = $11.75
  Day 4 -> R2   (E = 2, Speed buff = 10.0%)  |  Time = 13.5 min  Money = $15.0
  Day 5 -> R1   (E = 2, Speed buff = 10.0%)  |  Time = 8.1 min  Money = $11.75

WEIGHTED objective value = 52.1
Weekly nutrition [kcal carb prot fat] = [4055, 484, 197, 135]

---- Unweighted totals ----
Total travel time  : 52.1 min
Total money spent  : $57.3
```

- **Profile 3:** An arbitrary preference matrix is generated with `seed = 1`.

  Raw results:

```
---- FINAL PLAN ----
Daily weight vector (time  money):
  Day 1 -> (0.07, 0.93)
  Day 2 -> (0.35, 0.65)
  Day 3 -> (0.7, 0.3)
  Day 4 -> (0.63, 0.37)
  Day 5 -> (0.91, 0.09)

Day-by-day details:
  Day 1 -> R4   (E = 1, Speed buff = 0.0%)  |  Time = 14.0 min  Money = $10.0
  Day 2 -> R4   (E = 2, Speed buff = 10.0%)  |  Time = 11.7 min  Money = $8.8
  Day 3 -> R2   (E = 3, Speed buff = 20.0%)  |  Time = 10.4 min  Money = $15.0
  Day 4 -> R1   (E = 3, Speed buff = 20.0%)  |  Time = 10.4 min  Money = $5.88
  Day 5 -> R1   (E = 2, Speed buff = 10.0%)  |  Time = 8.1 min  Money = $11.75

WEIGHTED objective value = 49.03
Weekly nutrition [kcal carb prot fat] = [4305, 512, 204, 150]

---- Unweighted totals ----
Total travel time  : 54.6 min
Total money spent  : $51.42
```

5.1.2 Comparison of Different Preferences

We summarize the weekly optimal plan given three different preferences below (Table 3).

Preference	Restaurants of Choice	Money ($)	Time (min)	[kcal, carb, prot, fat]
Money First	R4 R4 R1 R1 R5	45.04	65.2	[4055, 484, 197, 135]
Time First	R4 R4 R1 R2 R1	57.3	52.1	[4305, 512, 204, 150]
Arbitrary	R4 R4 R2 R1 R1	51.42	54.6	[4305, 512, 204, 150]

Table 3: Comparison of dining plans under different preferences, without nutrition constraints.

At first glance, the restaurant choices in Table 3 show only subtle differences across preferences, with sequences often featuring R1, R2, R4, or R5 from Wednesday to Friday. However, these small changes lead to significant shifts in time and money costs, highlighting how the *sequence* of choices critically influences outcomes due to factors like daily deals and enjoyment dynamics.

Comparing "Money First" and "Time First" reveals clear trade-offs. The "Money First" plan prioritizes savings, such as the 50% discount at R1 on Thursday, resulting in the lowest money cost ($45.04) but higher travel time (65.2 min). In contrast, "Time First" sacrifices these savings for minimal travel, opting for a closer Subway (R2) on Thursday instead of R1—boosting the money cost to $57.3 while cutting time to 52.1 min.

The "Arbitrary" profile, with its balanced daily weights, yields costs ($51.42 and 54.6 min) that fall neatly between the extremes, aligning with expectations for a mixed-preference scenario. Across all plans, nutrition profiles vary but remain unconstrained, sometimes falling short of ideal weekly targets (e.g., low carbs in "Money First").

5.2 The Cost of a Healthey Diet

We then compare the results of not applying nutrition validation to one with nutrition validation, using the same preference matrix generated from `seed = 1`

5.2.1 The Unhealthy Optimal

The optimal solution for **W** generated from `seed = 1` without nutrition validation was already found in **Profile 3** from the last subsection.

5.2.2 The Healthy Optimal

An optimal solution for **W** generated from `seed = 1`, with the additional nutrition validation, was found below. We notice that the optimizer had rejected 5 plans due to unsatisfactory nutrition content before a validated optimal plan was found.

```
  Nutrition not met (cut #1)
  Objective     : 48.985249065399174
  Total nutrient: [4305, 512, 204, 150]
  Below lower by: [0.0, 0.0, 0.0, 0.0]
  Above upper by: [0.0, 0.0, 0.0, 10.0]

  Nutrition not met (cut #2)
  Objective     : 49.401099065399166
  Total nutrient: [4305, 512, 204, 150]
  Below lower by: [0.0, 0.0, 0.0, 0.0]
  Above upper by: [0.0, 0.0, 0.0, 10.0]

  Nutrition not met (cut #3)
  Objective     : 51.470099065399175
  Total nutrient: [4355, 538, 195, 155]
  Below lower by: [0.0, 0.0, 0.0, 0.0]
  Above upper by: [0.0, 0.0, 0.0, 15.0]

  Nutrition not met (cut #4)
  Objective     : 52.03024681651802
  Total nutrient: [4530, 522, 185, 160]
  Below lower by: [0.0, 0.0, 0.0, 0.0]
  Above upper by: [0.0, 0.0, 0.0, 20.0]

  Nutrition not met (cut #5)
  Objective     : 52.35714746322631
  Total nutrient: [4305, 512, 204, 150]
  Below lower by: [0.0, 0.0, 0.0, 0.0]
  Above upper by: [0.0, 0.0, 0.0, 10.0]

----------Nutrition satisfied after 5 cut(s)----------

-------- FINAL PLAN --------
Daily weight vector (time  money):
  Day 1 -> (0.07, 0.93)
  Day 2 -> (0.35, 0.65)
  Day 3 -> (0.7, 0.3)
  Day 4 -> (0.63, 0.37)
  Day 5 -> (0.91, 0.09)

Day-by-day details:
  Day 1 -> R3   (E = 1, Speed buff = 0.0%)  |  Time = 26.0 min  Money = $15.6
  Day 2 -> R4   (E = 3, Speed buff = 20.0%)  |  Time = 10.4 min  Money = $8.8
  Day 3 -> R1   (E = 4, Speed buff = 30.0%)  |  Time = 4.2 min  Money = $11.59
  Day 4 -> R4   (E = 3, Speed buff = 20.0%)  |  Time = 16.8 min  Money = $8.8
  Day 5 -> R1   (E = 4, Speed buff = 30.0%)  |  Time = 6.3 min  Money = $11.59

WEIGHTED objective value = 52.72
Weekly nutrition [kcal carb prot fat] = [4170, 484, 198, 140]

-------- Unweighted totals --------
Total travel time  : 75.0 min
Total money spent  : $56.38
```

5.2.3 Comparison on the Effect of Nutrition Validation

We summarize the weekly optimal plan with and without nutrition considerations into Table 4:

Preference	R of Choice	Money ($)	Time (min)	[kcal, carb, prot, fat]
W.O. Nu Constraint	R4 R4 R2 R1 R1	51.42	54.6	[4305, 512, 204, 150]
With Nu Constraint	R3 R4 R1 R4 R1	56.38	75.0	[4170, 484, 198, 140]

Table 4: Comparison of dining plans, with and without nutrition considerations

The *"cost of nutrients"*—the price paid in time and money to satisfy dietary goals—is clearly quantified by our results in Table 4. Enforcing these constraints is not free; it increases the total time cost by over 20 minutes (from 54.6 to 75.0) and adds nearly $5 to the weekly budget. The primary driver for this shift in restaurant choices is the need to satisfy the fat constraint; the unconstrained optimal solution contains 150g of fat, exceeding the upper bound of 140g by 10g.

Further analysis of the iterative solution process (as shown in the raw output above) reveals that every "no-good" cut was triggered by a violation of the fat upper bound. This consistent violation prompts a reflection: is the fat upper bound overly restrictive, or does this reflect a common dietary challenge in daily life?

We contend that this observation points to the latter scenario. Achieving a diet with diverse flavors, especially one incorporating richer spices and varied cooking styles beyond campus dining staples like salads and Subway sandwiches, often presents a challenge in meeting strict nutritional targets. Such preferences can indeed lead to potential dietary imbalances, particularly concerning fat intake. Therefore, maintaining tight nutrition bounds is crucial for the model to serve its purpose of guiding healthier choices, acknowledging that achieving this balance is often a gradual process in real-world dietary habits.

5.3 The Emergence of an Optimal Plan

An optimal plan for a single week is useful, but what constitutes a good dining strategy over time? To answer this, we moved beyond single-shot optimization to uncover recurring patterns in restaurant selection. We exploit our simulation to generate 1000 unique, randomized preference profiles, and for each one, we solved for the optimal weekly dining plan.

This process was conducted under two scenarios: one without nutritional constraints and one with them. By aggregating the optimal choices from all 1000 runs for each scenario, we can visualize the most frequently selected restaurants. The resulting patterns are displayed in the heatmap in Figure 2.

We observe a clear emergence of consistent optimal restaurant choices across the week. Without the nutrition constraint, Restaurant R4 (Subway) is predominantly chosen on Days 1 and 2, while Restaurant R1 is the primary choice for Days 3, 4, and 5, with some optimal plans also including R2 and R5. The introduction of the nutrition constraint maintains these dominant daily preferences for R4 and R1. However, it significantly alters the secondary choices, leading to a marked decrease in the selection of R2 and a noticeable increase in the frequency of R5 across Days 3, 4, and 5. This indicates that the nutrition constraint compels the optimization to diversify choices, favoring other restaurants to meet dietary requirements even

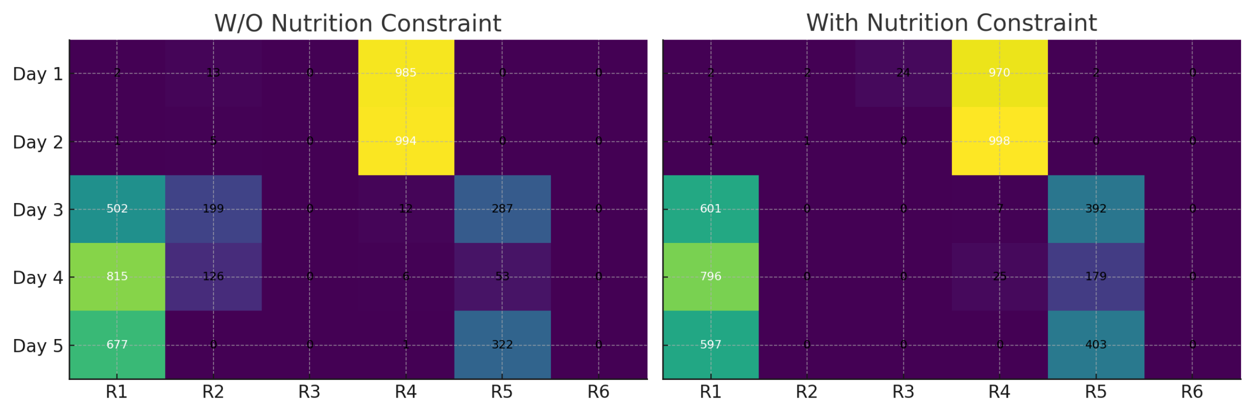


Figure 2: Heat Map of Weekly Restaurant Choices, with and without nutrition constraints. No one visits Mem Union :(

if they are not the top preference based solely on other criteria. We also notice that not one instance among our 1000 simulations chose to visit Memorial Union (R6).

5.4 A Discussion on Assumptions

After exploring the emergence of an optimal plan, it is important to discuss the assumptions and simplifications in the modeling process that shape the simulation output and the specific results we see above.

- **Simplified Restaurant Offerings:** We assume each restaurant serves only one fixed food option with predefined prices and nutritional content. In reality, restaurants offer diverse menus, allowing for varied choices within a single establishment. This simplification might lead to a less flexible optimal plan than a student could achieve in practice, potentially overestimating the "cost" of adhering to certain constraints if a healthier or cheaper option exists at a preferred restaurant.

- **Deterministic Environment:** Transit times between locations are assumed to be fixed and known. Similarly, prices, deals, and enjoyment values are static parameters. This deterministic view does not account for real-world variability such as traffic delays, unexpected restaurant closures, fluctuating prices, or individual variations in mood and preference. The model thus provides an "ideal" plan under perfect conditions, which may require adjustments in a dynamic environment.

- **Quantified and Linear Enjoyment:** A bold attempt was made to quantify *happiness.* The abstract concept of "happiness" is measured with a linear "Enjoyment Index" (E_d), which evolves based on fixed values assigned to each restaurant. The "speed buff" (ϕ_d) is then a piecewise linear function of this index. This linear accumulation and effect might oversimplify the complex, non-linear nature of human emotions and their impact on physical agility. It assumes a direct, predictable relationship that may not hold true for all individuals or situations.

- **Weekly Nutrition Aggregation:** Nutritional targets are set as aggregate bounds for the entire week. The model does not enforce daily nutritional requirements or consider meal-to-meal balance. This means that while the weekly totals may be met, the daily distribution of nutrients could be highly imbalanced, potentially leading to sub-optimal daily dietary habits despite meeting the overall weekly goals.

6 Conclusion and Future Directions

6.1 Conclusion

This project developed and implemented a mixed-integer linear programming model to optimize the daily lunch decisions for a university student. By integrating real-world complexities such as dynamic discounts, state-dependent travel times via an "Enjoyment Index," and nutritional requirements, our framework provides a powerful tool for exploring the trade-offs inherent in everyday planning.

Our analysis yielded several key insights:

First, we demonstrated that even subtle changes in the sequence of restaurant choices can lead to significant differences in weekly time and money costs, highlighting the value of a holistic, week-long optimization approach.

Second, by comparing preference profiles, we quantified the trade-offs between prioritizing time and money, showing how the model logically sacrifices deals for speed, and vice-versa.

Third, and most significantly, we calculated the tangible "cost of a healthy diet," revealing that enforcing nutritional constraints increased weekly travel time by over 20 minutes and costs by nearly $5 in our case study, primarily driven by the need to limit fat intake. One can further ask the question of whether it is a necessary sacrifice to have a healthier diet on campus.

Finally, through simulation, we identified robust daily dining patterns, showing that while certain restaurants (R1 and R4) are consistently optimal choices, nutritional constraints play a crucial role in diversifying the selection of secondary options.

6.2 Future Directions

The framework developed in this project can be extended in several compelling directions to enhance its realism and utility. Future work could involve scaling the model to include a more comprehensive set of restaurant choices and diversified menus, introducing a new layer of meal-specific decision-making. Incorporating dynamic route optimization for commutes and introducing dining times corresponding to restaurant types and the impact of restaurant traffic on ordering and dining times can provide more accurate time costs. From a behavioral perspective, adding constraints for meal variety and repetition aversion would better reflect real-world preferences. Furthermore, the model could be enhanced by integrating preference learning to automatically adapt to a user's dining history, or by employing stochastic optimization to account for uncertainties like unpredictable restaurant deals or travel times, thereby generating more robust and personalized weekly plans.

Appendix

Code 1: No Nutrition Check

```
# No Nutrition Check
# A Strategist's Guide to Campus Dining

using JuMP
using Gurobi
using Random
import MathOptInterface # Check the status of our Model
const MOI = MathOptInterface

const DAYS, RESTAURANTS = 5, 6

# ------------------------------ DATA --------------------------------------
#        r1  r2 r3  r4 r5 r6
time = [13  13  26  14  26  19;  # D1
        13  13  23  13  23  19;  # D2
        6   13  34  27  23  34;  # D3
        13  15  30  21  25  27;  # D4
        9   16  29  19  23  23]  # D5

price = [11.75, 15.00, 19.50, 10.00, 13.25, 17.25]

Random.seed!(1)
baseW  = [rand() for _ in 1:DAYS]                # taste weights
weights = hcat( round.(baseW;        digits=2),
                round.(1 .- baseW;   digits=2) )
# [kcal  carb  prot  fat]  (per meal)
nutrition = [ 975 102 41 35;
             1025 128 32 40;
              890 100 26 30;
              665  90 45 20;
              775 100 25 25;
              800 110 34 28 ]

nutr_lower = [3000, 484,  130,  20]
nutr_upper = [5250, 900, 300, 140]

# Enjoyment of restaurant (-1, 0, +1)
enjoy_val = [-1, 0, 2, 1, -1, 0]
R_plus    = [r for r in 1:RESTAURANTS if enjoy_val[r] == 1]

# ------------------------------ SET MODEL -----------------------------------
m = Model(Gurobi.Optimizer)
set_silent(m)
@variable(m, x[1:DAYS, 1:RESTAURANTS], Bin)                # dine / day,rest.
@constraint(m, [d=1:DAYS], sum(x[d,r] for r=1:RESTAURANTS) == 1)
@constraint(m, [r=1:RESTAURANTS], sum(x[d,r] for d=1:DAYS) <= 2)

#------------------------------ enjoyment inventory --------------------------
@variable(m, 0 <= E[1:DAYS] <= 6)
@constraint(m, E[1] == 1)
```

```
for d in 1:DAYS-1
    @constraint(m, E[d+1] == E[d] + sum(enjoy_val[r] * x[d,r] for r=1:RESTAURANTS))
end

@variable(m, lowEnjoy[1:DAYS], Bin)
bigM = 10
@constraint(m, [d=1:DAYS], E[d] - 1   <=  bigM*(1 - lowEnjoy[d]))
@constraint(m, [d=1:DAYS], E[d] - 0.1 >= -bigM*lowEnjoy[d])

# Must pick a +1 restaurant when lowEnjoy == 1
@constraint(m, [d=1:DAYS], sum(x[d,r] for r in R_plus) >= lowEnjoy[d])

# ------------------ time bonus phi(E) via SOS-2 approximation ---------------
knot_E      = 0.0:6.0
knot_factor = [1.0, 1.0, 0.9, 0.8, 0.7, 0.6, 0.5] # When enjoyment not a dominant factor
#knot_factor = [1.0, 1.0, 0.8, 0.6, 0.4, 0.2, 0.01] # When enjoyment is dominant factor

# when using the dominant enjoyment, the optimal solution will tend to go to the restaurant that
# can boost their enjoyment, because in this case "time" is only a dummy, the huge time reduce caused
# by enjoyment has already compensate the long time cost of some restaurant e.g. restaurant 3 have
# huge time cost on each day, never been choosed usually, but the enjoyment dominant
# condition will let it be reconsider.


@variable(m, 0 <= lambda[1:DAYS, 1:length(knot_E)] <= 1)
@variable(m, phi[1:DAYS] >= 0)

for d in 1:DAYS
    @constraint(m, sum(lambda[d,:]) == 1)
    @constraint(m, E[d]         == sum(knot_E[k]      * lambda[d,k] for k in 1:length(knot_E)))
    @constraint(m, phi[d]         == sum(knot_factor[k] * lambda[d,k] for k in 1:length(knot_E)))
    @constraint(m, lambda[d,:] in SOS2())
end

# ----------------------------- dynamic pricing ------------------------------
price_base = fill(0.0, DAYS, RESTAURANTS)
for d in 1:DAYS, r in 1:RESTAURANTS
    f = 1.0
    f = (r == 1 && d == 4)  ? 0.50 : f   # r1 Thu 50 %
    f = (r == 3 && isodd(d)) ? 0.80 : f  # r3 odd-days 20 %
    f = (r == 4 && iseven(d)) ? 0.88 : f # r4 even-days 12 %
    price_base[d,r] = price[r] * f
end

@variable(m, z2[2:DAYS], Bin)  # r2 two days in a row   20 % discount
@variable(m, y5[1:DAYS], Bin)  # first r5 visit         35 % discount
@variable(m, y6[1:DAYS], Bin)  # second r6 visit        16 % discount

for d in 2:DAYS
    @constraint(m, z2[d] <= x[d,2])
    @constraint(m, z2[d] <= x[d-1,2])
    @constraint(m, z2[d] >= x[d,2] + x[d-1,2] - 1)
end
```

```
@constraint(m, sum(y5) <= 1)
for d in 1:DAYS
    @constraint(m, y5[d] <= x[d,5])
    if d > 1
        @constraint(m, y5[d] + sum(x[k,5] for k=1:d-1) <= 1)
    end
end

@constraint(m, sum(y6) <= 1)
for d in 1:DAYS
    @constraint(m, y6[d] <= x[d,6])
    @constraint(m, y6[d] <= sum(x[k,6] for k=1:d-1))
    if d > 1
        @constraint(m, y6[d] >= x[d,6] + sum(x[k,6] for k=1:d-1) - 1)
    else
        @constraint(m, y6[d] == 0)   # cannot be \second" on day 1
    end
end

# ---------------------------- objective parts ------------------------------
@expression(m, expr_time,
    sum(weights[d,1] * phi[d] * time[d,r] * x[d,r] for d=1:DAYS, r=1:RESTAURANTS))

@expression(m, expr_money_base,
    sum(weights[d,2] * price_base[d,r] * x[d,r] for d=1:DAYS, r=1:RESTAURANTS))

@expression(m, expr_money_deals,
      sum(weights[d,2] * 0.20 * price[2] * z2[d] for d=2:DAYS) +
      sum(weights[d,2] * 0.35 * price[5] * y5[d] for d=1:DAYS) +
      sum(weights[d,2] * 0.16 * price[6] * y6[d] for d=1:DAYS))

@objective(m, Min, expr_time + expr_money_base - expr_money_deals)
optimize!(m)
# ----------------------------- report --------------------------------------
if termination_status(m) == MOI.OPTIMAL
    println("\n---- FINAL PLAN ----")
    println("Daily weight vector (time  money):")
    for d in 1:DAYS
        println("  Day $d -> (", weights[d,1], ", ", weights[d,2], ")")
    end

    daily_time  = zeros(DAYS)
    daily_money = zeros(DAYS)

    println("\nDay-by-day details:")
    for d in 1:DAYS
        # chosen restaurant
        r_idx = findfirst(r -> value(x[d,r]) > 0.5, 1:RESTAURANTS)

        # ---------- unweighted time ----------
        daily_time[d] = time[d, r_idx]        # raw travel minutes

        # ---------- unweighted money ----------
        # base dynamic price (Thu 50 %, etc.)
        cost = price_base[d, r_idx]

        # extra deal discounts realised in the solution
        if r_idx == 2 && value(z2[d]) > 0.5
            cost -= 0.20 * price[2]
        elseif r_idx == 5 && value(y5[d]) > 0.5
            cost -= 0.35 * price[5]
        elseif r_idx == 6 && value(y6[d]) > 0.5
            cost -= 0.16 * price[6]
        end

        daily_money[d] = cost

        println("  Day $d -> R$r_idx   (E = ",
                round(Int, value(E[d])), ", phi = ",
                round(value(phi[d]), digits=2), ")  |  ",
                "Time = ", round(daily_time[d] * value(phi[d]), digits=2), " min",
                "  Money = \$", round(daily_money[d], digits=2))
    end
    result = objective_value(m)
    println("\nWEIGHTED objective value = ", round(result, digits=2))

    println("\n---- Unweighted totals ----")
    println("Total travel time  : ", sum(daily_time), " min")
    println("Total money spent  : \$", round(sum(daily_money), digits=2))
else
    println("\nNo optimal nutrition-feasible plan found.")
end
```

Code 2: *With* Nutrition Check

```
# With Nutrition Feasibility Check
# A Strategist's Guide to Campus Dining

using JuMP
using Gurobi
using Random
import MathOptInterface # Check the status of our Model
const MOI = MathOptInterface

const DAYS, RESTAURANTS = 5, 6

# ------------------------------ DATA ---------------------------------------
#        r1  r2 r3  r4 r5 r6
time = [13  13  26  14  26  19;  # D1
        13  13  23  13  23  19;  # D2
        6   13  34  27  23  34;  # D3
        13  15  30  21  25  27;  # D4
        9   16  29  19  23  23 ]  # D5

price = [11.59, 15.00, 19.50, 10.00, 13.25, 17.25]

Random.seed!(1)
baseW  = [rand() for _ in 1:DAYS]                # taste weights
weights = hcat( round.(baseW;        digits=2),
                round.(1 .- baseW;   digits=2) )
# [kcal  carb  prot  fat]  (per meal)
nutrition = [ 975 102 41 35;
             1025 128 32 40;
              890 100 26 30;
              665  90 45 20;
              775 100 25 25;
              800 110 34 28 ]

nutr_lower = [3000, 484,  130,  20]
nutr_upper = [5250, 900, 300, 140]

# Enjoyment of restaurant (-1, 0, +1)
enjoy_val = [-1, 0, 2, 1, -1, 0]
R_plus    = [r for r in 1:RESTAURANTS if enjoy_val[r] == 1]

# ------------------------------ SET MODEL ----------------------------------
m = Model(Gurobi.Optimizer)
set_silent(m)
@variable(m, x[1:DAYS, 1:RESTAURANTS], Bin)                 # dine / day,rest.
@constraint(m, [d=1:DAYS], sum(x[d,r] for r=1:RESTAURANTS) == 1)
@constraint(m, [r=1:RESTAURANTS], sum(x[d,r] for d=1:DAYS) <= 2)

#------------------------------ enjoyment inventory --------------------------
@variable(m, 0 <= E[1:DAYS] <= 6)
@constraint(m, E[1] == 1)
for d in 1:DAYS-1
    @constraint(m, E[d+1] == E[d] + sum(enjoy_val[r] * x[d,r] for r=1:RESTAURANTS))
```

```
end

@variable(m, lowEnjoy[1:DAYS], Bin)
bigM = 10
@constraint(m, [d=1:DAYS], E[d] - 1   <=  bigM*(1 - lowEnjoy[d]))
@constraint(m, [d=1:DAYS], E[d] - 0.1 >= -bigM*lowEnjoy[d])

# Must pick a +1 restaurant when lowEnjoy == 1
@constraint(m, [d=1:DAYS], sum(x[d,r] for r in R_plus) >= lowEnjoy[d])

# ------------------ time bonus phi(E) via SOS-2 approximation ---------------
knot_E      = 0.0:6.0
knot_factor = [1.0, 1.0, 0.9, 0.8, 0.7, 0.6, 0.5] # When enjoyment not a dominant factor
#knot_factor = [1.0, 1.0, 0.8, 0.6, 0.4, 0.2, 0.01] # When enjoyment is dominant factor

# when using the dominant enjoyment, the optimal solution will tend to go to the restaurant that
# can boost their enjoyment, because in this case "time" is only a dummy, the huge time reduce caused
# by enjoyment has already compensate the long time cost of some restaurant e.g. restaurant 3 have
# huge time cost on each day, never been choosed usually, but the enjoyment dominant
# condition will let it be reconsider.


@variable(m, 0 <= lambda[1:DAYS, 1:length(knot_E)] <= 1)
@variable(m, phi[1:DAYS] >= 0)

for d in 1:DAYS
    @constraint(m, sum(lambda[d,:]) == 1)
    @constraint(m, E[d]         == sum(knot_E[k]      * lambda[d,k] for k in 1:length(knot_E)))
    @constraint(m, phi[d]       == sum(knot_factor[k] * lambda[d,k] for k in 1:length(knot_E)))
    @constraint(m, lambda[d,:] in SOS2())
end

# ----------------------------- dynamic pricing ------------------------------
price_base = fill(0.0, DAYS, RESTAURANTS)
for d in 1:DAYS, r in 1:RESTAURANTS
    f = 1.0
    f = (r == 1 && d == 4)  ? 0.50 : f   # r1 Thu 50 %
    f = (r == 3 && isodd(d)) ? 0.80 : f  # r3 odd-days 15 %
    f = (r == 4 && iseven(d)) ? 0.88 : f # r4 even-days 12 %
    price_base[d,r] = price[r] * f
end

@variable(m, z2[2:DAYS], Bin)  # r2 two days in a row   20 % discount
@variable(m, y5[1:DAYS], Bin)  # first r5 visit         35 % discount
@variable(m, y6[1:DAYS], Bin)  # second r6 visit        16 % discount

for d in 2:DAYS
    @constraint(m, z2[d] <= x[d,2])
    @constraint(m, z2[d] <= x[d-1,2])
    @constraint(m, z2[d] >= x[d,2] + x[d-1,2] - 1)
end

@constraint(m, sum(y5) <= 1)
for d in 1:DAYS
```

```
    @constraint(m, y5[d] <= x[d,5])
    if d > 1
        @constraint(m, y5[d] + sum(x[k,5] for k=1:d-1) <= 1)
    end
end

@constraint(m, sum(y6) <= 1)
for d in 1:DAYS
    @constraint(m, y6[d] <= x[d,6])
    @constraint(m, y6[d] <= sum(x[k,6] for k=1:d-1))
    if d > 1
        @constraint(m, y6[d] >= x[d,6] + sum(x[k,6] for k=1:d-1) - 1)
    else
        @constraint(m, y6[d] == 0)   # cannot be \second" on day 1
    end
end

# ---------------------------- objective parts -------------------------------

@expression(m, expr_time,
    sum(weights[d,1] * phi[d] * time[d,r] * x[d,r] for d=1:DAYS, r=1:RESTAURANTS))

@expression(m, expr_money_base,
    sum(weights[d,2] * price_base[d,r] * x[d,r] for d=1:DAYS, r=1:RESTAURANTS))

@expression(m, expr_money_deals,
      sum(weights[d,2] * 0.20 * price[2] * z2[d] for d=2:DAYS) +
      sum(weights[d,2] * 0.35 * price[5] * y5[d] for d=1:DAYS) +
      sum(weights[d,2] * 0.16 * price[6] * y6[d] for d=1:DAYS))

@objective(m, Min, expr_time + expr_money_base - expr_money_deals)

# ---------------- nutrition-loop with \no-good-cuts" ------------------------
cut_cnt   = 0
tot_nutr  = zeros(4)

while cut_cnt < 30          # guard against infinite loop
    optimize!(m)
    if termination_status(m) != MOI.OPTIMAL
        println("m infeasible or not optimal.")
        break
    end

    chosen = [(d,r) for d in 1:DAYS, r in 1:RESTAURANTS if value(x[d,r]) > 0.5]
    tot_nutr = sum(nutrition[r,:] for (_ ,r) in chosen)

    viol_low  = max.(0.0, nutr_lower .- tot_nutr)
    viol_high = max.(0.0, tot_nutr .- nutr_upper)
    violated  = any(viol_low  .> 1e-6) || any(viol_high .> 1e-6)

    if violated
        cut_cnt += 1
        println("  Nutrition not met (cut #$cut_cnt)")
        println("  Objective     : ", objective_value(m))
        println("  Total nutrient: ", tot_nutr)
        println("  Below lower by: ", viol_low)
        println("  Above upper by: ", viol_high)
        println("")
        @constraint(m, sum(x[d,r] for (d,r) in chosen) <= DAYS - 1)
    else
        println("----------Nutrition satisfied after $cut_cnt cut(s)----------")
        break
    end
end

# ---------------------------- report -------------------------------------
if termination_status(m) == MOI.OPTIMAL
    println("\n---- FINAL PLAN ----")
    println("Daily weight vector (time  money):")
    for d in 1:DAYS
        println("  Day $d -> (", weights[d,1], ", ", weights[d,2], ")")
    end

    daily_time  = zeros(DAYS)
    daily_money = zeros(DAYS)

    println("\nDay-by-day details:")
    for d in 1:DAYS
        # chosen restaurant
        r_idx = findfirst(r -> value(x[d,r]) > 0.5, 1:RESTAURANTS)

        # ---------- unweighted time ----------
        daily_time[d] = time[d, r_idx]        # raw travel minutes

        # ---------- unweighted money ----------
        # base dynamic price (Thu 50 %, etc.)
        cost = price_base[d, r_idx]

        # extra deal discounts realised in the solution
        if r_idx == 2 && value(z2[d]) > 0.5
            cost -= 0.20 * price[2]
        elseif r_idx == 5 && value(y5[d]) > 0.5
            cost -= 0.35 * price[5]
        elseif r_idx == 6 && value(y6[d]) > 0.5
            cost -= 0.16 * price[6]
        end

        daily_money[d] = cost

        println("  Day $d -> R$r_idx   (E = ",
                round(Int, value(E[d])), ", Speed buff = ",
                round(100*(1 - round(value(phi[d]), digits=2)), digits=2), "%)  |  ",
                "Time = ", round(daily_time[d] * value(phi[d]), digits=2), " min",
                "  Money = \$", round(daily_money[d], digits=2))
    end
    result = objective_value(m)
    println("\nWEIGHTED objective value = ", round(result, digits=2))
    println("Weekly nutrition [kcal carb prot fat] = ", tot_nutr)
    println("\n---- Unweighted totals ----")
    println("Total travel time  : ", sum(daily_time), " min")
    println("Total money spent  : \$", round(sum(daily_money), digits=2))
else
    println("\nNo optimal nutrition-feasible plan found.")
end
```