Optimizing Food Composition with a Genetic Algorithm

Cairo University
Faculty of Computers and Artificial Intelligence

Cairo University

Faculty of Computers and Artificial Intelligence

Term Research Project

Department: Operations Research and Decision Support

Course Name: Computational Intelligence

Course Code: DS313/DS351

Instructor: Assoc. Prof. Ayman Ghoneim - Assoc. Prof.

Sally Kassem

Name	ID
Youssef Hanaie Hussien	20220417
Ammar Mohsen Abdulaziz	20220214
Mohamed Gomaa Mohamed	20220449

Contents

Abstract

This code implements a genetic algorithm to optimize a daily meal plan based on nutritional constraints, aiming for a balanced diet. The GA uses real-valued encoding, roulette wheel selection, single-point crossover, mutation, and elitism, with penalties for constraint violations.

First identifying the **mathematical model**, we will use in this algorithm:

Data Model:

Table 1: Data Model for Decision Variables

Component	**Portion**	**Notation**
Vegetable Protein	0–3	x_1
Animal Protein	0–3	x_2
Carbohydrate	0–2	x_3
Vegetable	0–2	x_4
Fruit	0–2	x_5
Water	0.5–2	x_6
Healthy Fats	0–1.5	x_7
Dairy or Alternatives	0–3	x_8
Legumes	0–3	x_9
Whole Grains	0–2	x_{10}
One Day Consumption	~ 24	-

Objective Function:

The objective is to achieve a weighted nutritional score close to a target value of 24, representing a balanced diet. The nutritional score $S(x)$ is computed as a weighted sum of servings:

$$S(x) = 3x_1 + 3x_2 + 8x_3 + 5x_4 + 5x_5 + 8x_6 + 4x_7 + 3x_8 + 4x_9 + 6x_{10}$$

The objective function minimizes the deviation from perfect membership:

$$\text{Minimize } f(x) = 1 - s(x) \textit{ trying to make it equal 0}$$

This ensures the nutritional score is as close as possible to 24, maximizing dietary balance.

Constraints:

1. **Macronutrient Ratio Constraints**

Let T represent the total intake:

$$T = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10}$$

• **Protein Ratio (20% – 30%)**

$$0.20 \leq (x_1 + x_2 + x_8 + x_9) / T \leq 0.30$$

• **Carbohydrate Ratio (50% – 60%)**

$$0.50 \leq (x_3 + x_{10}) / T \leq 0.60$$

• **Fat Ratio (10% – 15%)**

$$0.10 \leq x_7 / T \leq 0.15$$

2. **Minimum Water Intake**

$$x_6 \geq 0.5$$

3. **Vitamin C Requirement**

$$20x_4 + 30x_5 + 5x_8 \geq 30$$

4. **Sodium Limit**

$$100x_2 + 80x_8 + 50x_{10} \leq 500$$

5. **Vegetarian Preference**

Animal-based food should not exceed 20% of the total intake:

$$(x_2 + x_8) / T \leq 0.20$$

6. **Dietary Variety**

At least 6 of the 10 food components must be non-zero:

$$\sum_{i=1}^{10} I(x_i > 0) \geq 6$$

$$\text{where } I(x_i > 0) = 1 \text{ if } x_i > 0, \text{ else } 0$$

Constraint Types:

• **Linear Constraints:** vitamin C requirement, sodium limit, and variable bounds are linear inequalities.

• **Nonlinear Constraints:** Macronutrient ratios and vegetarian preference are non- linear due to division by T. The variety constraint is discrete (count-based).

1 Encoding:

- **Representation:** Each solution (chromosome) is a list of **10 floating-point numbers**, representing servings of 10 food components (e.g., vegetable protein, carbohydrates, water). Values are rounded to two decimal places within defined ranges (e.g., 0–3 for most, 0.5–2 for water).
- **Justification: Floating-point** encoding allows fine-grained control over serving sizes, reflecting realistic dietary portions. The ranges ensure practical limits (e.g., water ≥ 0.5 servings) while maintaining flexibility**.**

2 Operators:

- **Selection:** Roulette wheel selection picks individuals based on fitness probabilities, with elitism preserving the top ELITE_SIZE (2) solutions. Chromosome Selection based on **Fitness Function** f[i] and P[i].

  $$\textit{Fitness Function} \rightarrow f[i] = \frac{1}{(f[i]+1)}$$

- **Crossover:** Single-point crossover swaps parts of two parent chromosomes with a probability of CROSSOVER_RATE (0.7).
- **Mutation:** Each gene may mutate with MUTATION_RATE (0.05), randomly resetting to a value within its range.
- **Justification:**
  - Roulette wheel balances exploration and exploitation, favoring fitter solutions while allowing diversity.
  - Crossover promotes recombination of good traits, with a high rate (0.7) to encourage mixing.
  - Mutation introduces randomness to prevent premature convergence, with a low rate to maintain stability.
  - Elitism ensures the best solutions persist, guaranteeing progress toward optimal solutions.

3 Constraint Handling Technique: Penalty Method:

How It Works:

1. **Penalty Calculation (**calculate penalties**):** • Constraints (e.g., protein ratio 20–30%, calories 400–600 kcal, at least 6 nonzero components) are checked.

   • Violations add penalties, scaled by a factor of 10:

   – Protein ratio < 0.20: penalty = $10 \times (0.20 - \text{ratio})$.

   – Calories < 400: penalty = $10 \times \frac{(400-\text{calories})}{100}$.

   • Penalties are normalized (e.g., per 100 kcal for calories, per 30 mg for vitamin C) for balance.

2. **Fitness Function:**

$$\text{fitness} = \frac{1}{\text{objective function}(x) + \text{penalties} + 1}$$

- Adds penalties to the objective function (deviation from target score 24).
- Lower fitness for solutions with violations.

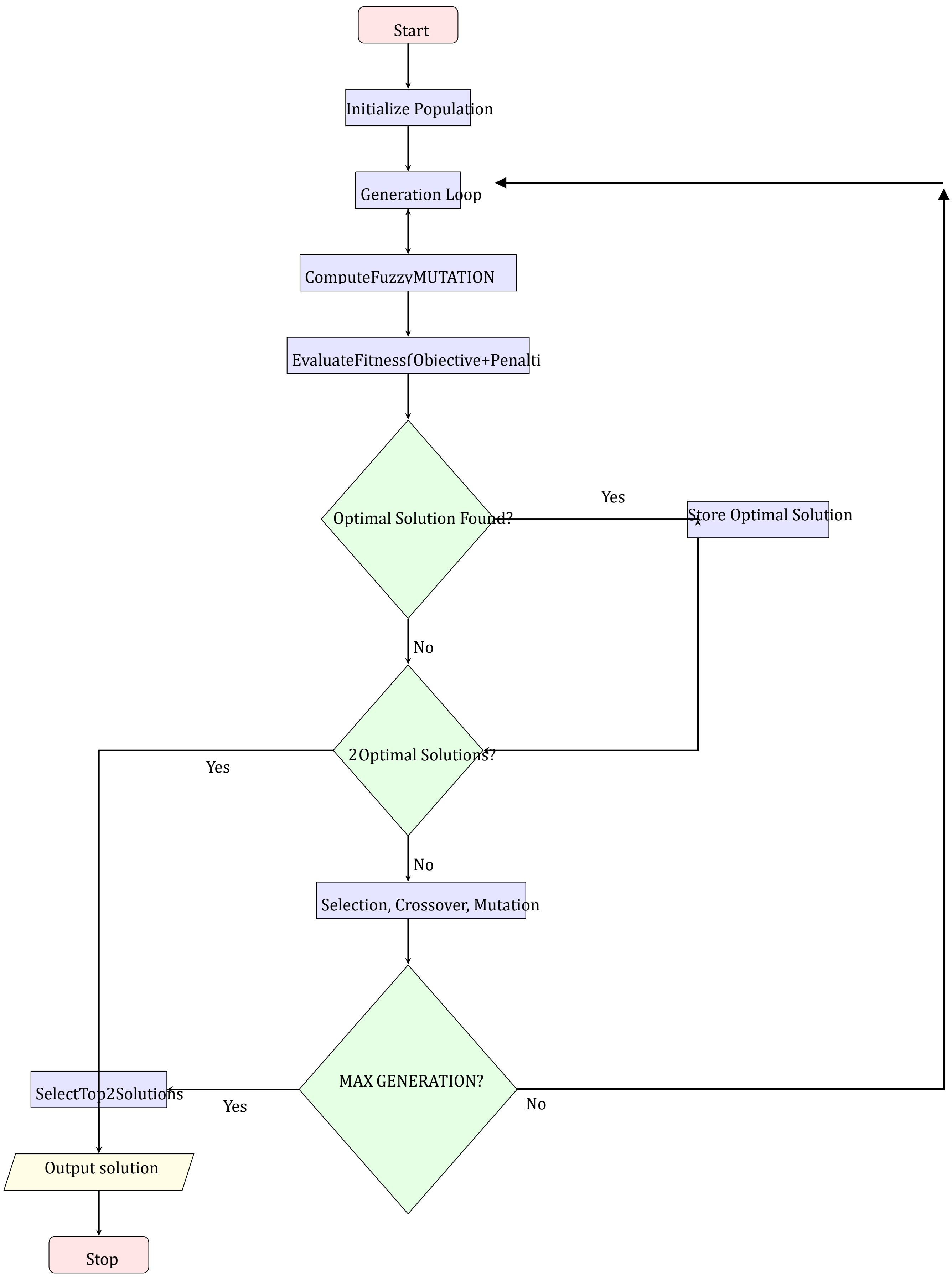
- **Justification:**
  - Allows exploration of infeasible regions while guiding the search toward feasible solutions.
  - Penalty factor (10) and normalization ensure balanced constraint enforcement.
  - Detailed violation feedback in main helps identify issues (e.g., "Protein Ratio: 0.15 (should be 0.20–0.30)").

4. Logical flow of program:

The program follows a standard genetic algorithm structure tailored for meal plan optimization:

1. **Initialize Population:** Creates POP_SIZE (20) random chromosomes with servings within specified ranges.
2. **Evaluate Fitness:** Computes the objective function (deviation from target score) and penalties for constraint violations, then calculates fitness as 1/(objective+penalties+1).
3. **Check for Best Solution:** If a chromosome has zero objective value and penalties, it's stored. Stops if two perfect solutions are found.
4. **Sort and Elitism:** Sorts the population by combined objective and penalty scores, preserving the top ELITE_SIZE (2) solutions.
5. **Selection:** Uses roulette wheel selection to choose parents for the next generation (excluding elites).
6. **Crossover:** Applies single-point crossover with CROSSOVER_RATE (0.7) to create children.
7. **Mutation:** Mutates genes with MUTATION_RATE (0.05), respecting ranges.
8. **New Population:** Forms the next generation and repeats from step 2 until termination.
9. **Output:** Returns and prints the top two solutions with their details and constraint violations.

Flowchart:



Start
Initialize Population
Generation Loop
ComputeFuzzyMUTATION
EvaluateFitness(Objective+Penalti
Optimal Solution Found?
Yes
Store Optimal Solution
No
2 Optimal Solutions?
Yes
No
Selection, Crossover, Mutation
MAX GENERATION?
Yes
No
SelectTop2Solutions
Output solution
Stop

5. Parameters:

- **POP_SIZE=20**
  Justification: A population size of 20 provides a balance between diversity and computational efficiency. For a problem with 6 to 10 variables (depending on the problem size), a small population ensures quick evaluation while maintaining enough genetic variation to explore the solution space effectively. Larger populations could slow down the algorithm without significant improvement for this problem's scale.
- **CROSSOVER_RATE=0.7**
  Justification: A 70% crossover rate encourages frequent recombination of genetic material, promoting the mixing of good traits between parents. This high rate accelerates convergence toward better solutions, which is crucial for a problem with multiple constraints (e.g., macronutrient ratios, caloric limits). It ensures the algorithm exploits promising solutions while still allowing some exploration.
- **MUTATION_RATE=0.05**
  Justification: A 5% mutation rate introduces small random changes to maintain diversity and prevent the algorithm from getting stuck in local optima. This low rate ensures stability in the population while allowing occasional exploration of new regions, which is important for escaping suboptimal solutions in a constrained optimization problem like meal planning.
- **MAX_GENERATIONS=100**
  Justification: Setting the maximum number of generations to 100 provides sufficient iterations for the algorithm to converge on good solutions without excessive computation. Given the early stopping condition (finding two perfect solutions), 100 generations act as a reasonable upper limit for this problem's complexity, ensuring the algorithm doesn't run indefinitely while still allowing enough time to refine solutions.
- **ELITE_SIZE = 2**
  Justification: Preserving the top 2 solutions each generation ensures the best meal plans are retained, aligning with the goal of returning two optimal solutions. This small elite size balances exploitation (keeping the best solutions) with exploration (evolving the rest of the population), making it effective for maintaining high-quality solutions throughout the optimization process.

6.fuzzification and de-fuzzification:

The MUTATION_RATE parameter, originally a crisp value (0.05), was converted to a fuzzy variable to adaptively adjust based on the generation number.

Fuzzification Steps

1. Input: Use the normalized generation number (current generation / MAX_GENERATIONS, range [0, 1]).

2. Fuzzy Sets for Generation:

   - Early: Peaks at 0, range [0, 0, 0.5].

   - Middle: Peaks at 0.5, range [0, 0.5, 1].

   - Late: Peaks at 1, range [0.5, 1, 1].

   - Compute membership degrees using triangular functions.

3. Fuzzy Sets for MUTATION_RATE:

    - Low: Peaks at 0.02, range [0, 0.02, 0.03].
    - Medium: Peaks at 0.05, range [0.03, 0.05, 0.07].
    - High: Peaks at 0.11, range [0.07, 0.11, 0.15].
4. Rules:
    - If generation is Early, MUTATION_RATE is High.
    - If generation is Middle, MUTATION_RATE is Medium.
    - If generation is Late, MUTATION_RATE is Low.

Defuzzification Steps

1. Inference: Apply rules using the minimum operator to combine generation memberships with MUTATION_RATE memberships.
2. Centroid Method: Compute a crisp MUTATION_RATE by discretizing the range [0, 0.15] into 100 points, calculating the weighted average of MUTATION_RATE values based on their membership degrees.

7.Results:

Small Problem Size

- Parameters: POP_SIZE=10, MAX_GENERATIONS=50

Solution 1

- Components:
    - Vegetable Protein: 0.16 | Animal Protein: 0.33 | Carbohydrate: 0.23 | Vegetable: 1.02 | Fruit: 0.67
    - Water: 0.95 | Healthy Fats: 0.44 | Dairy/Alt: 0.18 | Legumes: 0.25 | Whole Grains: 0.22
- Sum Value: 23.98
- Objective Function: 0.02
- Penalties: 4.00
- Violations:
    - Carbohydrate Ratio: 0.10 (should be 0.50–0.60)
    - Fat Ratio: 0.10 (should be 0.10–0.15)
    - Calories: 193.50 kcal (should be 400–600)

Solution 2

- Components: Identical to Solution 1
- Sum Value: 23.98
- Objective Function: 0.02

- Penalties: 4.00
- Violations: Same as Solution 1

Medium Problem Size

- Parameters: POP_SIZE=20, MAX_GENERATIONS=100

Solution 1

- Components:
  - Vegetable Protein: 0.08 | Animal Protein: 0.63 | Carbohydrate: 0.75 | Vegetable: 0.75 | Fruit: 0.58
  - Water: 0.51 | Healthy Fats: 0.21 | Dairy/Alt: 0.03 | Legumes: 0.14 | Whole Grains: 0.64
- Sum Value: 24.19
- Objective Function: 0.19
- Penalties: 2.30
- Violations:
  - Carbohydrate Ratio: 0.32 (should be 0.50–0.60)
  - Fat Ratio: 0.05 (should be 0.10–0.15)
  - Calories: 234.15 kcal (should be 400–600)

Solution 2

- Components: Identical to Solution 1
- Sum Value: 24.19
- Objective Function: 0.19
- Penalties: 2.30
- Violations: Same as Solution 1

Large Problem Size

- Parameters: POP_SIZE=50, MAX_GENERATIONS=200

Solution 1

- Components:
  - Vegetable Protein: 0.03 | Animal Protein: 0.13 | Carbohydrate: 0.23 | Vegetable: 1.23 | Fruit: 0.25

    - Water: 0.78 | Healthy Fats: 0.36 | Dairy/Alt: 0.08 | Legumes: 0.62 | Whole Grains: 0.65
- Sum Value: 24.02
- Objective Function: 0.02
- Penalties: 3.18
- Violations:
    - Protein Ratio: 0.20 (should be 0.20–0.30)
    - Carbohydrate Ratio: 0.20 (should be 0.50–0.60)
    - Fat Ratio: 0.08 (should be 0.10–0.15)
    - Calories: 188.45 kcal (should be 400–600)

Solution 2

- Components: Identical to Solution 1
- Sum Value: 24.02
- Objective Function: 0.02
- Penalties: 3.18
- Violations: Same as Solution 1

Summary

- **Small Problem**: Solutions are near the target sum (23.98), but high penalties (4.00) indicate violations in carbohydrate ratio, fat ratio, and calories.
- **Medium Problem**: Solutions slightly overshoot the target sum (24.19) with lower penalties (2.30), but still violate carbohydrate ratio, fat ratio, and calories.
- **Large Problem**: Solutions are very close to the target sum (24.02), with penalties (3.18) due to violations in protein, carbohydrate, fat ratios, and calories.

  - **Observation**: All solutions are suboptimal due to constraint violations; particularly low calorie counts and improper macronutrient ratios. Larger problem sizes improve sum accuracy but struggle with constraint satisfaction, suggesting a need for stronger constraint enforcement or parameter tuning.

References:

1 - Optimization using Genetic Algorithm in Food Composition Article in International Journal of Computing and Digital Systems · November 2021

Link: (PDF) Optimization using Genetic Algorithm in Food Composition

2 – Towards automatically generating meal plan based on genetic algorithm

Nan Jia1,4 · Jie Chen2 · Rongzheng Wang3 ·Mingliang Li1,4

Link: https://doi.org/10.1007/s00500-023-09556-0

3 - IJFANS INTERNATIONAL JOURNAL OF FOOD AND NUTRITIONAL SCIENCES ISSN PRINT 2319 1775 Online 2320 7876 Research Paper

PERSONALIZED NUTRITION PLANS USING GENETIC ALGORITHMS: OPTIMIZING DIETS BASED ON INDIVIDUAL GENOMIC DATA