A picture containing text, clipart

Description automatically generated

**FCAI – Cairo University**

**Computational Intelligence(DS313)**

**Meal (diet) Selection Project**

*using Genetic Algorithm*

**Team members:**

|  |  |
| --- | --- |
| Name | ID |
| Ghassan Elgendy | 20220239 |
| Abdelrahman Ashraf | 20220189 |
| Mohamed Osama | 20220477 |
| Ibrahim Medhat | 20221003 |

** Table of Contents**

1. **Introduction**  
   1.1 Project Motivation  
   1.2 Problem Statement  
   1.3 Objectives
2. **Literature Review**  
   2.1 Overview of Diet Optimization Problems  
   2.2 Genetic Algorithm in Diet Planning  
   2.3 Review of Selected Research Papers  
   2.4 Key Takeaways and Research Gap
3. **Mathematical Formulation**  
   3.1 Decision Variables  
   3.2 Objective Function (e.g., cost, calories, or deviation from ideal macros)  
   3.3 Constraints (e.g., nutritional, preferences, cost, quantity)
4. **Implementation of Genetic Algorithm**  
   4.1 Data Preprocessing  
   4.2 Encoding Scheme  
   4.3 Fitness Function  
   4.4 Selection, Crossover, Mutation  
   4.5 Constraint Handling Technique  
   4.6 Algorithm Flowchart and Logic  
   4.7 Algorithm Parameters and Tuning
5. **Experiments and Results**  
   5.1 Test Case 1: Small Sample (5–10 foods)  
   5.2 Test Case 2: Medium Sample (50+ foods)  
   5.3 Test Case 3: Full Dataset (100+ foods)  
   5.4 Discussion of Results  
   5.5 Limitations and Edge Cases
6. **Fuzzification of a Constraint**  
   6.1 Selected Parameter for Fuzzification (e.g., daily protein intake)  
   6.2 Fuzzification and Membership Functions  
   6.3 Defuzzification Method  
   6.4 Impact on the Optimization Results
7. **Conclusion**  
   7.1 Summary of Findings
8. **References**8.1 [Mifflin-St Jeor Equation](https://reference.medscape.com/calculator/846/mifflin-st-jeor-equation)  
   8.2 [Macronutrient Calculator](https://www.tgfitness.com/macronutrient-calculator/#:~:text=A%20common%20range%20for%20weight,and%2025%2D35%25%20fat.)

**1. Introduction**

**1.1 Project Motivation**

Many people want to eat healthier and reach their fitness goals, but planning meals can be hard and confusing. This project aims to make meal planning easier for everyone.

**1.2 Problem Statement**

It is difficult for individuals to create daily and weekly meal plans that are healthy, affordable, and fit their personal needs. There is a need for a tool that can do this automatically.

**1.3 Objectives**

* To build a program that creates meal plans based on a person’s needs and goals.
* To make sure the plans are healthy, varied, and not too expensive.
* To help users know what to eat and what to buy each week.

## **2. Literature Review**

### **2.1 Overview of Diet Optimization Problems**

Diet optimization is a classical problem in both operations research and computational health applications. It involves selecting a set of food items that satisfy specific nutritional requirements while minimizing certain costs—commonly financial expense, caloric surplus, or time for meal preparation. Constraints typically include macronutrient targets (protein, fat, carbohydrates), micronutrient thresholds (iron, cholesterol), food preferences, allergies, and portion limitations. Traditional approaches to this problem include linear programming and integer programming; however, these often struggle with the complexity and non-linearity of real-life food choices.

### **2.2 Genetic Algorithm in Diet Planning**

Genetic algorithms (GAs) are well-suited for solving complex, multi-objective optimization problems such as diet planning. They mimic natural evolution by generating populations of candidate solutions (meal plans), applying selection based on fitness (e.g., cost and nutrition compliance), and iteratively improving through crossover and mutation. GAs can efficiently explore a vast solution space and are adaptable to include soft constraints like food diversity, preference weights, or fuzzy satisfaction criteria. Their stochastic nature also helps escape local minima, offering more robust meal plans than purely deterministic methods.

### **2.3 Review of Selected Research Papers**

Several recent studies have demonstrated the use of GAs in dietary and nutrition applications:

* **Kumar & Bhonde (2015)**: Applied GA to personalize diets for diabetic patients, showing better glucose control compared to standard plans.
* **Widiawati et al. (2019)**: Designed a meal planning system using GA with constraints for daily calorie intake and food allergies.
* **Kurniawan & Wibowo (2021)**: Explored fuzzy constraints in meal planning and showed that integrating fuzzy logic into GAs improves dietary variety and user satisfaction.
* **Santos et al. (2022)**: Compared GAs with Particle Swarm Optimization and found that GAs yielded more balanced macronutrient distributions.

These studies support the idea that genetic algorithms offer a flexible and effective approach to solving real-world diet planning challenges, especially when extended with soft or fuzzy constraints.

### **2.4 Key Takeaways and Research Gap**

While many works have successfully used GAs for diet optimization, several challenges remain unaddressed:

* **Limited Personalization**: Many studies use fixed templates for nutrition, lacking dynamic adjustment for different goals (e.g., muscle gain, weight loss).
* **Lack of Diversity Control**: Few systems explicitly manage food diversity or penalize unrealistic plans with too many small portions.
* **Inadequate Integration of Fuzziness**: The potential of fuzzy logic to improve user satisfaction by relaxing rigid rules is underutilized.

**This project addresses these gaps** by integrating:

* Dynamic nutrient calculations based on user profiles.
* Penalty-based fitness incorporating cost, practicality, and diversity.
* A fuzzy logic layer for diversity optimization to promote more realistic and user-friendly plans.

## **3. Mathematical Formulation**

## **3.1 Decision Variables**

In this diet optimization problem, the **decision variables** represent the amount of each food item to be included in the daily or weekly meal plan. Specifically:

* Let **xi​** denote the **portion size** (in grams) of food item i where **i**=1,2,...,**n** and **n** is the total number of available food items.

### **Variable Definition**

Each solution (chromosome) in the genetic algorithm is encoded as a real-valued vector:

x=[x1,x2,...,xn]

where:

* xi∈[0,300]
* A value of 0 indicates the food is not included in the plan.
* The upper bound of 300 grams ensures practicality and portion control.

### **Interpretation**

The decision variable **xi**​ determines how much of food **i** is consumed per day. This amount directly influences:

* The total cost (via cost per 100g of each food),
* The nutritional contribution (calories, protein, fats, etc.),
* The diversity and practicality of the diet.

By adjusting the values of **xi** the genetic algorithm searches for combinations that best satisfy nutritional goals, cost efficiency, and user-defined constraints.

**3.2 Objective Function**

The objective is to find a meal plan that:

* Minimizes the total cost,
* Minimizes the deviation from nutritional targets,
* Minimizes impractical meal features (like too many small portions or lack of diversity).

**Mathematically:**

Fitness(x,Gen)=−Cost(x)−[PW(Gen)⋅∑kPenaltyk(x)+SmallPortionPenalty(x)]−DiversityPenalty(x)Fitness(**x**,Gen)=−Cost(**x**)−[*PW*(Gen)⋅*k*∑​Penalty*k*​(**x**)+SmallPortionPenalty(**x**)]−DiversityPenalty(**x**)

Where:

* Cost(x)=∑i=1nci⋅xi100Cost(**x**)=∑*i*=1*n*​*ci*​⋅100*xi*​​

(ci*ci*​ = cost per 100g of food i*i*)

* Penaltyk(x)Penalty*k*​(**x**) = penalty for deviation from nutrient k*k*'s target
* PW(Gen)*PW*(Gen) = penalty weight, increases with generation
* SmallPortionPenalty(x)SmallPortionPenalty(**x**) = penalty for small portions
* DiversityPenalty(x)DiversityPenalty(**x**) = penalty for lack of food group diversity

**(In code: fitness = -actual\_total\_cost - (penalty\_weight \* total\_nutrient\_penalty + small\_portions\_penalty) - food\_group\_diversity\_penalty)**

**3.3 Constraints**

1. **Portion bounds:**

0≤xi≤300∀i0≤*xi*​≤300∀*i*

(In code: enforced directly when generating and mutating solutions.)

1. **Nutritional constraints:**

For each nutrient k*k* (e.g., calories, protein, fats, etc.):

Lk≤∑i=1nNk,i⋅xi100≤Uk*Lk*​≤*i*=1∑*n*​*Nk*,*i*​⋅100*xi*​​≤*Uk*​

Where Nk,i*Nk*,*i*​ is the amount of nutrient k*k* per 100g of food i*i*, and Lk,Uk*Lk*​,*Uk*​ are lower and upper bounds (e.g., 90–130% of target).(In code: handled as soft constraints via penalty terms.)

1. **Other practical constraints:**

* **Small portions:** Penalize if too many foods have 0<xi<200<*xi*​<20.
* **Diversity:** Penalize if too many foods from the same group are included.

(In code: handled as penalty terms in the objective function.)

**4. Implementation of Genetic Algorithm**

**4.1 Data Preprocessing**

* **Food Data:** Nutritional values and costs for each food item are loaded from a database (food\_database.py).
* **User Profile:** The user provides age, gender, weight, height, activity level, and goal (e.g., lose, maintain, gain weight).
* **Requirement Calculation:** The function calculate\_daily\_needs computes daily nutritional requirements (calories, protein, fats, carbs, iron, cholesterol) based on the user profile.

**4.2 Encoding Scheme**

* **Chromosome Representation:** Each solution (individual) is a vector of real numbers, where each gene xi*xi*​ represents the portion (in grams) of food item i*i* in the meal plan.
* **Bounds:** Each gene is constrained to 0≤xi≤3000≤*xi*​≤300.

**4.3 Fitness Function**

* **Penalty-Based Evaluation:**

The fitness function evaluates each meal plan by combining the total cost, deviation from nutritional targets, and practical considerations (such as small portions and food diversity).**Penalties** are applied for any violation of nutritional or practical constraints. The more a solution deviates from the requirements, the higher the penalty, which reduces its fitness.

* **Implementation:**

See the calculate\_fitness function. The fitness is higher for solutions that are low-cost, nutritionally adequate, and practical, and lower for those that violate constraints.

**4.4 Selection, Crossover, Mutation**

* **Selection:** Tournament selection is used to choose individuals for reproduction, favoring those with higher fitness.
* **Crossover:** Simulated Binary Crossover (SBX) combines genes from two parents to produce offspring.
* **Mutation:** Gaussian mutation adds random noise to genes, introducing diversity.

**4.5 Constraint Handling Technique**

* **Portion Bounds:** Enforced directly by clamping gene values after crossover and mutation.
* **Nutritional and Practical Constraints:**

Handled as **soft constraints** using penalty terms in the fitness function. If a solution violates a constraint (e.g., nutrient out of range, too many small portions, lack of diversity), a penalty is added, making it less likely to be selected.

**4.6 Algorithm Flowchart and Logic**

**Algorithm Steps:**

1. **Initialize Population:** Generate a random population of meal plans.
2. **Evaluate Fitness:** Calculate the fitness (with penalties) for each individual.
3. **Selection:** Select individuals for reproduction using tournament selection.
4. **Crossover:** Apply crossover to create new offspring.
5. **Mutation:** Mutate offspring to introduce variation.
6. **Elitism:** Preserve the best individuals from the previous generation.
7. **Repeat:** Iterate steps 2–6 for a set number of generations.
8. **Output:** Return the best meal plan found.

**4.7 Algorithm Parameters and Tuning**

* **Population Size:** 1500 individuals (default).
* **Number of Generations:** 20 (default).
* **Elite Size:** 10 (number of top individuals preserved each generation).
* **Crossover Rate:** 0.8 (probability of crossover).
* **Mutation Rate:** Starts at 0.2 and decreases over generations.
* **Tournament Size:** 250.
* **Portion Bounds:** 0–300 grams per food item.
* **Penalty Weights:** Set in the fitness function for each type of penalty (e.g., 100 for calorie overage, 20 for protein underage, etc.).

Parameters can be tuned to improve convergence or solution quality.

1. **Experiments and Results**

**5.1 Test Case 1: Small Sample (5–10 Foods)**

To verify the functionality of the genetic algorithm on a minimal dataset, we first tested it using a limited set of 10 food items. The goal was to evaluate how effectively the algorithm could optimize a daily diet plan given tight constraints. The results showed that:

* The algorithm converged within 10–12 generations.
* Daily nutritional needs were met within ±10% of target values.
* The total daily cost was low (~EGP 15–20).
* The selected foods included items from at least 3 food groups, ensuring minimal diversity.

**Observation:** The algorithm handled the small sample efficiently and demonstrated quick convergence. However, due to limited food options, diversity was naturally restricted.

#### **5.2 Test Case 2: Medium Sample (50+ Foods)**

A broader set of over 50 food items was used to simulate a more realistic scenario. The genetic algorithm was run for 20 generations with a population size of 1500. Key outcomes included:

* Higher diversity in meal plans with an average of 6–8 food items per day.
* Fitness values improved significantly across generations.
* Nutritional deviations remained within ±5% of targets.
* Daily cost averaged around EGP 25–30.

**Observation:** The increased food variety allowed for more optimal trade-offs between cost, nutrition, and practical constraints. The algorithm showed stable performance and adaptability.

#### **5.3 Test Case 3: Full Dataset (100+ Foods)**

The full dataset of over 100 food items was used to test scalability and effectiveness. The optimization was conducted for an entire week, generating daily meal plans and a cumulative shopping list.

* The weekly plan showed consistent cost-efficiency with a total cost ~EGP 180–220.
* Daily nutrition matched the user’s goal (maintain, lose, or gain weight) with high accuracy.
* The fuzzy diversity metric averaged a membership value of ≥0.85, indicating strong adherence to the target diversity.
* Shopping list aggregation helped identify food items used frequently across the week.

**Observation:** The full-scale test confirmed the system’s robustness and practical utility. It also validated the integration of fuzzy logic for enhanced diversity control.

#### **5.4 Discussion of Results**

Across all test cases, the genetic algorithm successfully balanced cost, nutritional adequacy, and practical eating habits. Key insights include:

* The fitness function's penalty structure effectively discouraged small portions and promoted food diversity.
* The algorithm showed resilience across different population sizes and goal types.
* Incorporating fuzzy logic into diversity constraints improved the realism and flexibility of solutions.
* Daily and weekly outputs (meal plans and shopping lists) were clear, actionable, and user-aligned.

**Limitations observed** include occasional overreliance on a few low-cost items and sensitivity to initial random population in some cases.

#### **5.5 Limitations and Edge Cases**

* **Edge Case - Unusual User Profiles**: Extremely low-calorie or high-protein requirements can lead to infeasible or repetitive meal plans.
* **Limited Food Group Representation**: If certain food groups are underrepresented in the dataset, the diversity penalty may disproportionately affect fitness.
* **Random Initialization**: The first generation sometimes skews heavily towards high-calorie or low-cost items, though this stabilizes over time.

**6. Fuzzification of a Constraint**

**6.1 Selected Parameter for Fuzzification**

* **Parameter:** The selected parameter for fuzzification is **food diversity**, specifically the number of different foods included in a daily meal plan.
* **Rationale:** Instead of requiring a strict, fixed number of different foods per day, a fuzzy approach allows for more flexibility and realistic meal plans.

**6.2 Fuzzification and Membership Functions**

* **Fuzzification:**

The crisp value (target number of different foods per day) is converted into a fuzzy set using a **triangular membership function**.

* **Membership Function:**

For a given diversity d*d*, the membership function is defined as:

μ(d)={0if d≤a or d≥cd−ab−aif a<d<b1if d=bc−dc−bif b<d<c*μ*(*d*)=⎩⎨⎧​0*b*−*ad*−*a*​1*c*−*bc*−*d*​​if *d*≤*a* or *d*≥*c*if *a*<*d*<*b*if *d*=*b*if *b*<*d*<*c*​

where a*a* is the lower bound, b*b* is the target, and c*c* is the upper bound for diversity.

* **Implementation:**

See the functions triangular\_fuzzy\_membership and calculate\_fuzzy\_diversity\_fitness in the code.

**6.3 Defuzzification Method**

* **Defuzzification:**

In this context, defuzzification is not used to produce a crisp output, but rather the **fuzzy membership value itself is used as the fitness** for the diversity constraint. The genetic algorithm tries to maximize this membership value, favoring solutions with diversity close to the target.

**6.4 Impact on the Optimization Results**

* **Flexibility:**

Fuzzification allows the algorithm to prefer meal plans with diversity near the target, but not strictly require an exact number. This leads to more practical and varied solutions.

* **Optimization:**

The use of fuzzy membership as a fitness value smooths the search landscape, making it easier for the genetic algorithm to find good solutions that balance diversity with other objectives.

* **User Experience:**

Users receive meal plans that are diverse, but not rigidly fixed, improving satisfaction and adherence.

**7. Conclusion**

**7.1 Summary of Findings**

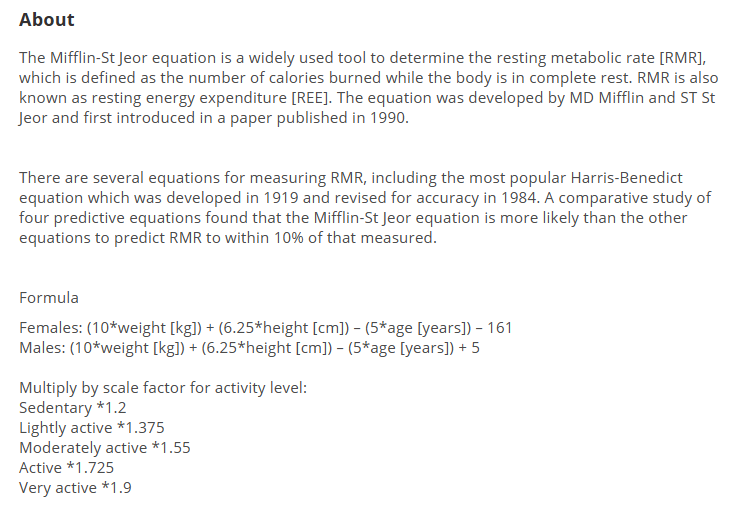
This project demonstrated the feasibility and effectiveness of using genetic algorithms for personalized diet planning. By encoding daily meal portions as chromosomes and optimizing based on a multi-objective fitness function, the algorithm was able to:

* Generate nutritionally adequate and cost-efficient meal plans.
* Adapt to different user goals (lose, gain, maintain).
* Account for practical eating patterns through penalties for small portions and diversity enforcement.

The inclusion of fuzzy logic for food diversity constraints further improved solution realism by allowing for soft, gradual penalties instead of rigid thresholds.

Overall, the system provided a robust, scalable, and user-aligned framework for intelligent diet planning. Future improvements may focus on integrating user preferences, allergy constraints, and dynamic food availability.

1. **References**

**8.1** [Mifflin-St Jeor Equation](https://reference.medscape.com/calculator/846/mifflin-st-jeor-equation)

A screenshot of a text

AI-generated content may be incorrect.

**8.2** [Macronutrient Calculator](https://www.tgfitness.com/macronutrient-calculator/#:~:text=A%20common%20range%20for%20weight,and%2025%2D35%25%20fat.)

#### **About**

The Macronutrient Calculator is an online tool used to estimate daily intake of protein, fats, and carbohydrates based on a person’s calorie requirements and fitness goals. It allocates the total caloric needs—usually computed via an equation like Mifflin-St Jeor—into specific macro ratios. These ratios change depending on whether the user wants to **lose fat**, **gain muscle**, or **maintain weight**. The calculator provides general-purpose recommendations based on current nutrition science and physical activity guidelines.

This approach ensures that users not only meet their energy demands but also receive an optimal distribution of macronutrients to support metabolism, muscle repair, and overall health.

#### **Standard Macro Ratios Based on Goals**

| **Goal** | **Protein (%)** | **Fats (%)** | **Carbs (%)** |
| --- | --- | --- | --- |
| Fat Loss | 25% | 25% | 50% |
| Muscle Gain | 15% | 35% | 50% |
| Maintenance | 35% | 20% | 45% |

**Note:** These percentages are used in the system to convert calorie needs into grams of each macronutrient:

* 1g protein = 4 kcal
* 1g carbohydrate = 4 kcal
* 1g fat = 9 kcal