Group 18 Assignment 1
The Great Bitwise Bake Off

Xinqi Phillip SHEN
LIACS
Leiden University
2333CZ Leiden, the Netherlands
x.shen.4@umail.leidenuniv.nl

Xueyun Liu
LIACS
Leiden University
2333CZ Leiden, the Netherlands
x.liu.27@umail.leidenuniv.nl

Abstract

We use a recipe collection from an open-source repository as the inspiring set for our creative system. The system has a custom genetic algorithm specifically designed for recipe generation, with tailored fitness functions, crossover operations, and mutation mechanisms adapted to the unique constraints of baking recipes. The algorithm run for 200 generations and return the best recipe found thus far. We evaluate the generated recipes' creativity based on three key criteria: novelty, usefulness, and diversity.

Introduction

This assignment explores computational creativity through cookie recipe generation. We use a cookie recipe collection from Average_Cookie (O'Brien) on GitHub as our inspiring set, which forms the foundation of our knowledge base. The dataset is provided in CSV format, containing 209 cookie recipes with detailed ingredient information and user ratings.

Our approach focuses on a custom genetic algorithm tailored specifically for baking recipes. Unlike general-purpose genetic algorithms, our implementation includes specialized functions that respect the structural constraints of baking—for instance, ensuring that recipes contain appropriate ratios of flour, fat, and leavening agents. The algorithm iterates through 200 generations, progressively refining recipes through selection, crossover, and mutation operations. At the end of this process, the system outputs the highest-rated recipe found.

To assess the creativity of our system, we carry out a multi-factor evaluation that considers novelty (measuring divergence from the inspiring set), usefulness (assessing whether generated recipes would produce edible cookies), and diversity (examining the variety of outputs).

Methodology

In this section we provide detailed solutions for each component of the recipe generation system.

Inspiring Set Preparation

We picked the 2_Scaled_Units_Cleaned.csv file from the Average_Cookie repository as our inspiring set. This dataset was chosen because it provides standardized measurements across all recipes. The file contains 209 cookie recipes, each decomposed into individual ingredient entries. Each row in the CSV represents a single ingredient within a recipe and includes the following fields:

- Ingredient: The ingredient name (e.g., all purpose flour, butter, brown sugar)
- Text: The original quantity and unit in string format as it appeared in the source
- Recipe_Index: A unique identifier linking ingredients to their parent recipe
- Rating: User ratings from three recipe websites, providing a quality metric
- Quantity: The numerical amount of the ingredient (normalized to standard units)
- Unit: The measurement unit (e.g., cup, teaspoon, gram)

Setting Up Knowledgebase

To support efficient genetic operations, we transformed the flat CSV structure into a hierarchical JSON format organized by complete recipes. We grouped all ingredient entries sharing the same Recipe_Index, creating a recipes array where each recipe object contains a name field and an ingredients array with the ingredient details (name, amount, unit, and rating).

We also added a categories section that groups ingredients by function: flours, fats, sugar, eggs, leavening, liquid, flavorings, and add-ins. This categorization serves four purposes in our genetic algorithm. First, it enables fitness evaluation based on recipe balance—a good cookie recipe typically needs appropriate proportions from each category. Second, it guides mutation operations to maintain recipe viability by ensuring essential ingredients remain present. Third, it facilitates meaningful crossover by swapping ingredient groups rather than random individual ingredients. Last but not least, the creativity evaluation also uses category information. The final knowledge base preserves all 209 recipes from the original dataset.

Recipe Generator

Our recipe generator implements a custom genetic algorithm tailored to the constraints and requirements of cookie

recipes. The algorithm evolves recipes over multiple generations by selecting high-quality parent recipes, combining them through crossover, and introducing variations through mutation.

Fitness Calculation The fitness function evaluates recipe quality through multiple criteria combined into a single score. The overall fitness is calculated as:

$$\text{fitness} = \text{rating} \times \text{balance} \times \text{diversity} \times \text{complexity} \times 10$$

The *rating* represents the average rating of all ingredients in the recipe, derived from the original dataset's user ratings. This provides a foundation based on historically successful ingredient choices.

The *balance* measures how well ingredient proportions match ideal ratios for cookie recipes. We define approximate ideal ratios based on common baking principles: flour (2.5 tsp), fat (1.0 tsp), sugar (1.0 tsp), eggs (2.0 count), and leavening (1.0 tsp). For each category, we calculate the ratio between actual and ideal amounts, penalizing significant deviations. Missing required categories receive a 0.5 penalty multiplier.

The *diversity* encourages recipes with reasonable ingredient counts:

- 0.7 for fewer than 5 ingredients (too simple)
- 1.0 for 5-8 ingredients (standard)
- 1.2 for 8-12 ingredients (good variety)
- 0.8 for more than 15 ingredients (overly complex)

The *complexity* rewards recipes with varied ingredient categories rather than multiple items from the same category. It is calculated as the ratio of unique categories to total ingredients, preventing recipes dominated by a single category (e.g., five different types of chocolate chips).

Finally, all recipes must pass a validation check ensuring the presence of five essential categories: flour, fat, sugar, eggs, and leavening. Recipes missing any required category receive a fitness of 0.0 and are eliminated from the population. The factor of 10 scales the final fitness values to a more interpretable range for demonstration purposes.

The First Generation The initial population consists of 50 recipes randomly selected from the inspiring set of 209 original recipes. This provides a diverse starting point with known viable recipes.

To select parents for reproduction, we implement tournament selection with tournament size $k = 10$. In each tournament, we randomly sample 10 individuals from the current population and select the one with the highest fitness. This process repeats until we have selected enough parents to produce the next generation.

How to Crossover Our crossover operation preserves the categorical structure of parent recipes to maintain baking validity. Rather than randomly swapping individual ingredients, we perform category-aware crossover:

First, we group all ingredients in both parent recipes by their functional categories (flour, fat, sugar, etc.). Then, for each category present in either parent, we randomly choose to inherit ingredients from parent 1 or parent 2 with equal probability (0.5). If a category exists in only one parent, we inherit it directly. This approach ensures that the offspring recipe maintains coherent ingredient groups rather than arbitrary combinations.

For example, if parent 1 uses all-purpose flour and parent 2 uses bread flour and cake flour, the offspring will inherit either just all-purpose flour or both bread flour and cake flour, but not a random mix. This preserves the functional relationships between ingredients within categories.

As a safety measure, if the crossover produces an empty recipe (which can occur in rare cases), we add a random ingredient from parent 1 to ensure the offspring is non-empty.

How to Mutate Mutation introduces variation while maintaining recipe validity. We apply five different mutation operators, each selected randomly with equal probability:

Amount adjustment (type 0): Randomly select an ingredient and multiply its amount by a factor between 0.8 and 1.2. This creates subtle variations in ingredient proportions, with a minimum threshold of 0.1 to prevent ingredients from disappearing.

Ingredient substitution (type 1): Replace an ingredient with another from the same functional category. For example, butter might be replaced with shortening, or all-purpose flour with bread flour. The amount remains unchanged to preserve recipe proportions.

Category addition (type 2): If the recipe is missing any ingredient categories, randomly select a missing category and add an ingredient from it. This helps incomplete recipes evolve toward validity.

Non-essential removal (type 3): Remove a random ingredient from non-essential categories (add-ins, flavoring, liquid, other), but only if the recipe has more than 5 ingredients. This prevents recipes from becoming unnecessarily complex.

Add-in duplication (type 4): Duplicate a random add-in ingredient (chocolate chips, nuts, etc.). This allows popular add-ins to appear in greater quantities.

The mutation probability is set to 0.5 after empirical testing. Higher values (above 0.6) caused population instability, while lower values (below 0.1) reduced diversity. Despite these variations, the quality of the output is always within an acceptable range, with all fitness values reaching convergence after a moderate number of generations.

How to Form a New Generation After crossover and mutation, recipes go through normalization to ensure realistic proportions. The normalization process first combines duplicate ingredients by summing their amounts, then scales the entire recipe to a target total volume.

We convert all ingredient amounts to a common unit (teaspoons) using standard conversion factors: 1 cup = 48 tsp, 1 tablespoon = 3 tsp, 1 ounce $\approx$ 6 tsp (for dry ingredients), and 1 egg $\approx$ 12 tsp (approximate volume). The target total volume is set to 240 teaspoons (approximately 5 cups), which represents a typical cookie batch. Each ingredient is then scaled proportionally to meet this target.

After scaling, we enforce minimum amounts to prevent impractically small quantities: 0.25 for cups and teaspoons, 0.5 for tablespoons, and at least 1 whole egg (rounded to the nearest integer). This normalization prevents the algorithm from generating recipes with unrealistic measurements like 0.01 cups of flour or 1000 cups of milk.

The new generation maintains the same population size as the previous generation (50 individuals), with offspring replacing the current population entirely (generational replacement strategy).

For creativity consideration, the seemingly strange unit after convert like 0.81 cup of powder is kept rather than rounding. Converting to weight might be a solution but it is less commonly used in a recipe so we reserve the commonly-used units.

Creativity Evaluation Evaluation of creativity is based upon three factors: novelty, value and typicality.

Novelty measures how different the generated recipe is from the training dataset. We compute two complementary metrics: *ingredient novelty* and *combination novelty*.

Ingredient novelty uses Jaccard similarity to find the most similar recipe in the original set:

$$\text{Novelty}_{\text{ing}} = 1 - \max_{r \in R} \frac{|I_g \cap I_r|}{|I_g \cup I_r|} \tag{1}$$

where I_g represents the ingredient set of the generated recipe and I_r represents ingredients from recipe r in the original set R.

Combination novelty examines pairwise ingredient relationships, calculating the proportion of ingredient pairs that never appeared together in the training data:

$$\text{Novelty}_{\text{comb}} = \frac{|P_g \setminus P_R|}{|P_g|} \tag{2}$$

where P_g contains all ingredient pairs from the generated recipe and P_R contains all pairs observed in the training set.

Value evaluates the quality and usefulness of the generated recipe through multiple fitness criteria. We aggregate four components: average ingredient rating, nutritional balance, diversity, and structural validity. The balance score assesses whether ingredient category proportions fall within acceptable ranges, while diversity rewards recipes that have 8–12 ingredients. The overall value score is computed as:

$$\text{Value} = \frac{\text{Rating} + \text{Balance} + \text{Diversity}}{3} \tag{3}$$

Following Boden's computational creativity framework, we define creativity as the combination of novelty and value, modulated by typicality. Typicality ensures the recipe remains recognizable as a cookie by comparing ingredient category proportions to original data averages. The final score is:

$$\text{Creativity} = \frac{(\text{Novelty} + \text{Value}) \cdot \phi}{2} \tag{4}$$

where ϕ is a typicality penalty ($\phi = 1.0$ if typicality $>$ 0.3, otherwise $\phi = 0.5$). This formulation rewards recipes that are both innovative and high-quality while penalizing those that deviate too far from regular cookies.

Stopping Criteria The algorithm terminates after 200 generations, returning the recipe with the highest fitness score encountered during the entire evolutionary process. We chose a fixed generation count rather than fitness-based convergence criteria, the reason for which would be discussed in Reflections section.

We implemented a non-interactive stopping criteria without user intervention during evolution. This decision was made because users typically cannot form accurate mental images of cookies from textual ingredient lists, nor can they reliably evaluate recipe quality without actually baking and tasting the results.

Experiment

In this section we provide several experiment results.

Fitness Over Generation

Initially we experimented with three different evolution generations.

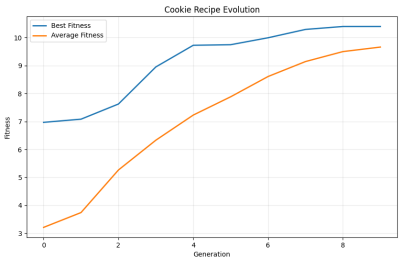

Figure 1: Fitness after 10 generations

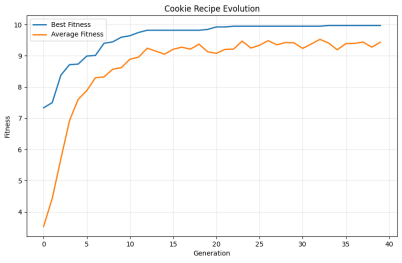

Figure 2: Fitness after 40 generations

As is shown in Figure 1, 10 generations are not enough for average fitness to converge. When it comes to 40, average fitness seems to converge at 9 on Figure 2. Then we used a much lager amount of generations as 200, results shown on Figure 3 where fitness converges around 11.

Creativity Score

Figure 4 shows the creativity score after evaluation along with the corresponding fitness. The higher the fitness, the higher the corresponding creativity.

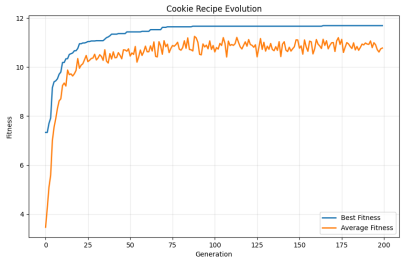

Figure 3: Fitness after 200 generations

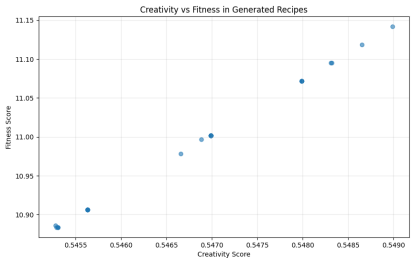

Figure 4: Creativity after 200 generations

Generated Recipes ad Cookbook

An example of output recipe is:

- addins 0.25 cup chocolate chip
- eggs 2.00 egg egg
- fat 1.00 cup margarine
- flavoring 0.53 teaspoon vanilla
- flour 2.27 cup all purpose flour
- leavening 1.00 teaspoon baking soda
- other 0.90 teaspoon salt
- sugar 1.00 cup sugar

The recipe's fitness is 11.142, creativity score being 0.549. The cookbook contains more details and AI generated information to make it resemble an authentic recipe, including a recipe name, comprehensive baking directions and a background tale that adds narrative depth and atmosphere.

Reflections

In this section we analyze the experiment results and Generative AI usage for cookbook.

Fitness and Creativity Score

As shown in Figure 3, the fitness score converges to approximately 11 after 80 generations. However, fitness values continue to fluctuate by ± 0.5, indicating incomplete stabilization. Therefore, we extend evolution to 200 generations to ensure robust convergence.

Figure 4 demonstrates the relationship between fitness and creativity scores within the evolved population. The results show that recipes with higher fitness values (around 11) tend to achieve higher creativity scores (approximately 0.545). However, the relationship exhibits diminishing returns—the marginal improvement in creativity score decreases as fitness increases beyond a certain threshold. This suggests that the algorithm has reached a practical optimum where further fitness optimization yields minimal creativity gains. The convergence to an intermediate creativity score indicates that our generator successfully balances the competing objectives of producing novel recipes while maintaining baking validity. Recipes with extremely high creativity scores often sacrifice usefulness (becoming too unusual to bake successfully), while recipes with very low creativity scores simply reproduce existing patterns from the inspiring set. The observed plateau around 0.545 represents a desirable outcome where recipes are novel and diverse yet remain practical and well-balanced.

Generative AI usage for Cookbook

For the final cookbook, we use the generator's output as prompt words being fed to ChatGPT and ask the language model to generate baking directions for a consumable cookie. Then with ingredients and baking directions, the language model is asked to create new prompt to be used as input for Stable Diffusion to generate an image for the recipe. Taking image, ingredients and baking directions as input, a meaningful name along with a short artificial story is made up for the recipe by ChatGPT to make the recipe resemble an authentic one.

Generative AI does great job in prompt generation, producing professional-level details like "close-up photograph" and "crispy edges and soft centers" that typically require both culinary and photographic expertise. It also performs well in creative tasks like story narratives and recipe naming when complex tasks were broken into smaller components. However, multi-modal capability became a bottleneck in image generation - the visuals didn't accurately represent all specified ingredients despite detailed prompts. For the baking directions, we lack the expertise to verify them and simply trust the AI's output since it appears logical and well-structured. This shows that AI-generated technical content would seem very convincing with their logic and structure, and the importance of domain expertise for validation is significantly needed.

References

O'Brien, E. $Average_{c}ookierepository.$ $https://github.com/elleobrien/Average_{C}ookie.$