Cookie recipe generator

Dinu Catalin-Viorel

28-10-2022

1 Knowledgebase

This cookie generator requires a knowledgebase to create the initial populations and use it to develop them to better results. For this project, we took the data from Fruit Flavor Pairing Chart and The Flavour Guide and saved them as two CSV files.
pairings.csv stores the data of ingredient pairing, and ingredients.csv saves the ingredient's name, class, and quantity.

pairings.csv include:

- Ingredient name
- Good pairing ingredient
- Thing Three

ingredients.csv include:

- Ingredient name
- Class: binding, core, fat, flavor, flour, fruit, nut, rising, spirits, and sugars.
- Quantity: maximum, minimum, and scale.

2 Taste function

To compute how good a recipe it is we use concepts from Food Pairing Theory. So, we create a huge graph (pairing graph) with all the possible ingredients we want to use in our recipes, and we add an edge between elements so that their taste pairs well together(for example apple-caramel, apple-mango).
For a recipe, we will also assign a weight to the edges between elements to have a stronger connection between elements that ratio near 1 to have a bigger weight than a ratio near 0. Then, we want to maximise this value with our algorithm. We also want to consider recipes with fewer ingredients as better than ones with more ingredients, so our function will decrease in value with the increasing

number of ingredients. We want to assign different values to a recipe than to the same one with an extra ingredient that doesn't match the other ingredients.

$$recipe = [ing_1, ing_2, ..., ing_n],\ ing_i = \text{recipe's ingredient}$$
$$qt_i = \text{ the quantity of } ing_i \text{ used in the recipe}$$
$$ratio_{i,j} = \begin{cases} \frac{qt_i}{qt_j}, qt_i \leq qt_j \\ \frac{qt_j}{qt_i}, qt_i > qt_j \end{cases}$$
$$\mathbf{1}_{i,j} = \begin{cases} 1, ing_i \text{ and } ing_j \text{ are connected in the pairing graph} \\ 0, \text{otherwise} \end{cases}$$
$$taste(recipe) = \frac{\sum_{i=1}^{n} \sum_{j=i+1}^{n} ratio_{i,j} * \mathbf{1}_{i,j}}{n}$$

3 Genetic algorithm

3.1 Representation

For our algorithm, we used a list of ingredients to be our representation of a recipe. For every ingredient from that list, we know the class it belongs to, the range(min, max, step) of the quantity and the actual quantity.
Even though we work with our recipes as a list, for the rest of the report, we will consider the list as the reunion of the lists of ingredients from every class:

$$recipe = [ing_1, ing_2, ..., ing_n]$$
$$classes = \text{all the ingredients classes}$$
$$cl_i = \text{the ingredients from class i}, i \in classes$$
$$cl_i = \{ing | ing \in recipe, \text{ the class of the ingredient is i}, i \in classes\}$$
$$recipe = \bigcup_{i \in classes} cl_i$$

3.2 Parent selection

For parent selection, we used a standard roulette algorithm (recipes with higher values for our taste function has a higher probability of being chosen).

$$population = [recipe_1, recipe_2, ..., recipe_m]$$
$$losses = [loss_1, loss_2, ..., loss_m],\ loss_i = taste(recipe_i)$$
$$p_i = \text{the probability that i-th recipe is chosen}$$
$$p_i = \frac{loss_i - min(losses)}{\sum_{i=1}^{m}(loss_i - min(losses))}$$

3.3 Crossover

For our algorithm, we adapt a classic uniform crossover to work with our list of various lengths.

The following algorithm is applied to every class of ingredients from a recipe (cl_i).

$$
\begin{aligned}
parent^1 &= \bigcup_{i \in classes} cl_i^1 \\
parent^2 &= \bigcup_{i \in classes} cl_i^2 \\
cl_i^1 &= [ing_1^1, ing_2^1, ..., ing_{n_1}^1] \\
cl_i^2 &= [ing_1^2, ing_2^2, ..., ing_{n_2}^2] \\
crossover\_mask &= [b_1, b_2, ..., b_{max(n_1,n_2)})] \\
\mathbb{P}(b_i = 1) &= 0.5 \\
\mathbb{P}(b_i = 0) &= 0.5 \\
cl_i'^1 &= \bigcup_{i < max(n_1,n_2)} \{ \begin{cases} ing_i^1, \text{ if } b_i = 0 \text{ and } i \leq n_1 \\ ing_i^2, \text{ if } b_i = 1 \text{ and } i \leq n_2 \\ \emptyset, \text{ otherwise} \end{cases} \} \\
cl_i'^2 &= \bigcup_{i < max(n_1,n_2)} \{ \begin{cases} ing_i^2, \text{ if } b_i = 0 \text{ and } i \leq n_2 \\ ing_i^1, \text{ if } b_i = 1 \text{ and } i \leq n_1 \\ \emptyset, \text{ otherwise} \end{cases} \} \\
offspring^1 &= \bigcup_{i \in classes} cl_i'^1 \\
offspring^2 &= \bigcup_{i \in classes} cl_i'^1
\end{aligned}
$$

After those steps, we will modify the offspring, so they don't have duplicate ingredients.

3.4 Mutation

The following mutation operators are applied to every class of ingredients from a recipe (cl_i).

1. **Remove ingredient from a class**
   With probability p_1 we will choose a random element from cl_i to remove from the recipe.

2. **Change the quantity of an ingredient from a class**
   For every ingredient from cl_i with probability p_2 we will change the ingredient quantity by a new random value sampled from the range of that ingredient.

3. **Add new ingredient from a class**
   With probability p_3 we will add a new ingredient to cl_i that doesn't already exist in the recipe and it is from the same class as the elements from cl_i

3.5 Environment selection

To select the population used in the next iteration, we will select top n recipes by the value of taste function from the reunion of the current population and the offspring population (n = the population size).

$$\begin{aligned} P(t) &= \text{the population at t-th epoch} \\ n &= \text{size of } P(t) \\ P'(t) &= \text{the population of offspring at t-th} \\ P(t+1) &= \text{top n from } (P(t) \cup P'(t)) \end{aligned}$$

4 Creativity and novelty

In this project, the data of ingredients we use do not only contain ingredients commonly used in cookies, so some results are peculiar compared to most of the cookie recipes. However, we limit the diversity and quantity of ingredients by setting the range of the variety of ingredients and default amount. Also, those recipes are unique and individual because every time we run the application, it will randomly initialize the populations.

To assess the creativity of our system, We consider the quality and novelty of the output to be an essential indicators. Based on our Taste function, we have a high probability of getting a tasty cookie, meaning the value of our recipes will be influenced by the data we use. Additionally, our cookie recipe generator can be more than just an application dedicated to generating cookie recipes. For example, if we increase the range of class 'fruit' and set the class 'flour' to 0, we will get fruit salad recipes instead. It shows our algorithm has high flexibility.