Recipe allocation algorithm
Rui Xin Lee
18th February 2021

1. The class `RecipeAllocation` is a genetic algorithm that searches for a feasible recipe allocation across default orders within a constrained space, The constrains are:
    a. there is no same recipe in each order (condition A)
    b. vegetarian orders are only limited to vegetarian recipes, while gourmet orders can take both vegetarian and gourmet recipes (condition B)
    c. there is sufficient stock for each recipe (condition C)

2. Business implications of this algorithm are the following:
    a. Execution time:
    The evaluation time of the genetic algorithm is proportionate to the size of default orders and choices of recipes. The search is not instantaneous. Hence it cannot be used for real time business decisions making. Instead, this algorithm can be used to generate different scenarios of feasible default orders allocation. It can then be used to establish different scenarios of excessive stock to be sold to transactional customers. These scenarios have to be recalibrated after a number of arrivals of new orders.
    b. Optimal solutions:
    The algorithm only searches for a feasible solution for default orders. This solution may not be optimal in providing a good variety of recipes options to transactional customers. Further algorithms or development on the current objective functions has to be done to satisfy this purpose

2. Each order can be a combination of
    a. either vegetarian or gourmet
    b. either two, three or four recipes a week
    c. either two or four portions

3. After initiating the class, the method `load` loads in orders.json and stock.json as specified in the task.

4. The method `search_feasibility` returns
    a. `True` if a feasible allocation is found. It stores a feasible solution as `self.feasible_parent` and sets `self.is_feasible` as `True`
    b. `False` if the profit function does not improve in 5 successive steps. It sets `self.is_feasible` as `False`

5. The algorithm begins by creating a random 100 candidates of recipes allocations for each specific order.
    a. To ensure that condition A is satisfied, a non replacement sampling is used to generat these candidates.
    b. To ensure that condition B is satisfied, vegetarian orders for a candidate are sampled from vegetarian recipes, while gourmet orders are sampled from both vegetarian and gourmet recipes.

6. The profit function that the algorithm maximises is the sum of differences between total recipe allocation and the recipe stock, for each recipe. The recipe allocations take into consideration the number of recipes and the number of portions for each order.

$$argmax\, f(allocation) := \{ \sum_{i}[min(0,\ stock_i\ -\ \sum_{orders} allocation_i)]\}\ ,\ i \in \{recipes_{1,} \ldots, recipes_{22}\}$$

When this profit function reaches 0, a solution is found.
If the profit function does not improve in 5 successive steps it is assumed that there is no feasible solution. The number of successive steps can be tweak as a trade off between the execution time and the confidence of the algorithm output.

7. The algorithm creates a log.txt in the data folder recording
   a. time taken for each generation
   b. best and worst profit for each generation
   c. best allocation in each generation

8. Future algorithm improvement:
   a. The non replacement sampling within the genetic algorithm can be weighted for the distribution of recipes in stock.
   b. The mutation rate and mutation gene count decreases as the profit gets closer to 0.
   c. To improve execution speed, the genetic algorithm can run on distributed computers.