Extending Genetic Algorithm

Introduction

In this project, I explored ways to enhance a genetic algorithm for optimizing candy production, focusing on maximizing profit while introducing practical production constraints. Specifically, I implemented two extensions to the algorithm:

1. Adding a second production line with no repeated candies between lines.
2. Restricting one line to produce only chocolate-type candies and the other only fruit-type candies.

These changes simulate real-world production requirements where lines might be limited to certain candy types, or restrictions could prevent the same candy from being produced on both lines. The goal was to see how these modifications would influence the algorithm's performance and profitability.

Extensions and How They Were Implemented

1. **Adding a Second Production Line with No Repeat Candies**

- **Objective**: I wanted to simulate having two distinct production lines, each responsible for a unique set of candies to maximize variety and avoid redundancy.
- **Implementation**:
    - In the Population class, I created two Line objects per individual to represent the two production lines—one for chocolate and one for fruit candies.
    - To ensure no candy was repeated across lines, I set a condition during line initialization so that any candy selected by one line was excluded from the options of the other. This was achieved by filtering out already-chosen candies from the other line's selection pool.

    - **Code Details**: In the initialization of each line, the code filters out candies that the other line has selected. This way, each line independently optimizes its unique candy list without overlap.
- **Effect on Results**:
    - Adding this second line with non-repeating candies diversified the candy selection, leading to different profit outcomes in each experiment. Some configurations resulted in higher profits when both lines balanced demand more effectively, suggesting the added diversity helped the algorithm adapt better to fluctuating conditions.
    - This change also encouraged the algorithm to spread profits across both the chocolate and fruit lines, leading to more robust, adaptable solutions across generations.

2. Restricting Each Line by Candy Type (Chocolate or Fruit Only)

- **Objective**: My goal here was to mimic real-world production limits, where a line might only be set up to produce either chocolate or fruit-based candies.
- **Implementation**:
    - To set this up, I modified the Line class to accept a candy_type parameter, indicating if the line would produce chocolate or fruit candies only.
    - During initialization, the line would filter available candies to include only those that matched its designated type. For example, if a line was restricted to chocolate candies, only chocolate options would be available for selection. This was done by checking if each candy name contained the word "chocolate" or "fruit," thereby narrowing the options based on the type.
    - **Code Details**: The filter was applied at the start of line creation, so each production line adhered strictly to its designated candy type, either chocolate or fruit. Any candies not matching the specified type were excluded from that line’s selection pool.
- **Effect on Results**:

    - This restriction led to each line becoming specialized in its type of candy, which seemed to create a more balanced and optimized production setup.
    - Interestingly, I noticed that configurations with a higher demand for chocolate tended to yield slightly higher profits on the chocolate line compared to the fruit line. This constraint limited flexibility, but it also forced the algorithm to optimize within the bounds of each candy type, highlighting trends in demand and profitability specific to chocolate or fruit.

Experiment Results

I ran three main configurations to see how these extensions would perform:

- **Population Size & Generations**: I tested the algorithm with three different setups: (10, 15), (15, 10), and (20, 20) for population size and generations, respectively.
- **Findings**:
    - **Higher Population and Generations**: The configuration (20, 20) was the most effective, yielding the highest average profits. The larger population allowed more exploration, and the extra generations helped refine the results further.
    - **Balanced Profit Across Lines**: When using a larger population and more generations, I noticed that profits between the chocolate and fruit lines became more balanced. This setup seemed to allow the algorithm to find better solutions, making the most out of both lines by balancing selection across candy types.

Overall, the results showed that these extensions—adding a second line with no repeats and restricting lines by candy type—helped improve the algorithm's adaptability and optimization for profit. The varying demand conditions across experiments also emphasized how these constraints could be useful in real-world settings, making production lines more specialized yet complementary.

Conclusion

This project demonstrated how adding practical constraints to a genetic algorithm can simulate real-world production setups more closely. By adding a second line and restricting each line to a specific type of candy, the algorithm was able to generate more realistic, optimized selections. While these changes limited flexibility in some ways, they also led to more strategic, balanced selections across candy types. The experiments provided interesting insights into the impact of these constraints, and I believe there's room to add further complexity—such as price fluctuations or changeover penalties—to explore additional production dynamics.