BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG CÔNG NGHỆ VÀ THIẾT KẾ UEH

KHOA CÔNG NGHỆ THÔNG TIN KINH DOANH

KHÓA LUẬN TỐT NGHIỆP

Đề tài:

Tìm hiểu và phát triển ứng dụng dòng lệnh đơn giản về công thức nấu ăn thuần thực vật, sử dụng thuật toán di truyền (GA – Genetic Algorithm) để tạo ra công thức nấu ăn từ nguyên liệu sẵn có.

Giảng viên hướng dẫn : TS. Hoàng Anh

Sinh viên thực hiện : Huỳnh Thị Cẩm Nhung

MSSV : 31201024508

Lớp : DS001

Hệ : CQ

Khóa : K46

Thành Phố Tam Kỳ - Năm 2024

MỤC LỤC NỘI DUNG

MỤC LỤC HÌNH ẢNH VÀ BẢNG BIỂU

1. Giới thiệu

1.1. Lý do chọn đề tài

Trong bối cảnh hiện nay, chế độ ăn thuần chay đang ngày càng thu hút sự quan tâm của cộng đồng, không chỉ từ những người ủng hộ bảo vệ động vật mà còn từ những người quan tâm đến sức khỏe và bảo vệ môi trường. Một chế độ ăn thuần chay khoa học có thể giúp giảm nguy cơ mắc các bệnh mãn tính, cải thiện sức khỏe và góp phần bảo vệ môi trường thông qua việc giảm tiêu thụ sản phẩm từ động vật.

Tuy nhiên, việc duy trì chế độ ăn thuần chay một cách bền vững không hề đơn giản, đặc biệt trong bối cảnh xã hội hiện đại với nhịp sống bận rộn và nhu cầu về sự tiện lợi ngày càng tăng cao. Một trong những thách thức lớn mà người theo đuổi chế độ ăn thuần thực vật gặp phải là tìm kiếm các công thức nấu ăn vừa phong phú, vừa phù hợp với nguyên liệu có sẵn.

Để giải quyết vấn đề này, đề tài khóa luận tập trung vào việc xây dựng một ứng dụng dòng lệnh có khả năng đề xuất công thức nấu ăn thuần chay dựa trên một số lượng nguyên liệu cụ thể mà người dùng có, lựa chọn hương vị mong muốn và số lượng khẩu phần (servings). Sử dụng tính toán tiến hóa Evolutionary Computation (EC), cụ thể là thuật toán di truyền (Genetic Algorithm), ứng dụng này sẽ tìm kiếm và tối ưu hóa các công thức nấu ăn dựa trên 117 công thức của Dr. Greger – một chuyên gia nổi tiếng về dinh dưỡng thuần chay.

1.2. Mục tiêu nghiên cứu

Mục tiêu chính của nghiên cứu này là phát triển một ứng dụng dòng lệnh sử dụng thuật toán di truyền (Genetic Algorithm – GA) – một loại thuật toán tối ưu hóa thuộc nhóm tính toán tiến hóa Evolutional Computation (EC) dựa trên nguyên lý chọn lọc tự nhiên và di truyền học – để tạo ra các công thức nấu ăn thuần chay dựa trên nguyên liệu có sẵn do người dùng nhập vào. Ứng dụng được tạo ra với mong muốn giúp người dùng dễ dàng tạo ra các món ăn phù hợp với khẩu vị và số lượng khẩu phần mà họ mong muốn.

1.3. Phạm vi nghiên cứu

Phạm vi của nghiên cứu này giới hạn trong việc sử dụng 117 công thức nấu ăn từ Dr.Greger làm cơ sở dữ liệu gốc. Người dùng sẽ được yêu cầu cung cấp ít nhất 3 nguyên liệu có sẵn, chọn hương vị mong muốn và xác định số lượng khẩu phần (servings) cần thiết. Thuật toán di truyền (Genetic Algorithm) sẽ sử dụng các ràng buộc này để tìm kiếm và tạo ra công thức tối ưu nhất trong phạm vi dữ liệu gốc.

1.4. Cấu trúc báo cáo

Báo cáo này được chia thành các phần chính sau:

- Cơ sở lý thuyết: Tổng quan về bộ dữ liệu và các thuật toán tính toán tiến hóa.

- Lý do lựa chọn Evolutionary Computation cho bài toán.

- Phương pháp nghiên cứu: Bao gồm mô tả bài toán (đầu vào và đầu ra) và đi sâu vào việc thiết kế các thuật toán di truyền (chia làm 2 thuật toán di truyền, một cho nguyên liệu và một cho hướng dẫn nấu ăn).

- Phát triển ứng dụng: Môi trường phát triển (Python và các thư viện), cấu trúc chương , giao diện dòng lệnh CLI.

- Kết quả thực nghiệm: Các công thức được tạo ra, biểu đồ fitness và đánh giá tổng quan.

- Thảo luận chung và các hạn chế của ứng dụng.

- Hướng cải tiến trong tương lai.

- Kết luận, tài liệu tham khảo và mã nguồn

2. Cơ sở lý thuyết

2.1. Giới thiệu về các công thức nấu ăn thuần thực vật của Dr. Greger

Dr. Michael Greger, một bác sĩ y khoa nổi tiếng và là tác giả của nhiều cuốn sách về dinh dưỡng, đã trở thành một trong những gương mặt tiêu biểu trong phong trào ăn thuần chay nhờ những nghiên cứu sâu sắc và các bài giảng khoa học về dinh dưỡng. Ông nổi tiếng với cuốn sách **"How Not to Die"** và ứng dụng thực phẩm vào việc phòng ngừa và điều trị các bệnh mãn tính. Các công thức nấu ăn của Dr. Greger không chỉ nhấn mạnh việc sử dụng thực phẩm từ thực vật mà còn chú trọng vào việc chọn lựa những thực phẩm giàu dưỡng chất, nhằm cung cấp một chế độ ăn uống lành mạnh, cân bằng và đầy đủ dinh dưỡng.

Bộ sưu tập 117 công thức nấu ăn của Dr.Greger và các cộng sự được đăng công khai trên website nutritionfacts.org/ không chỉ ngon miệng, dễ thực hiện mà còn đảm bảo các tiêu chí về dinh dưỡng cần thiết cho một chế độ ăn thuần thực vật. Mỗi công thức đều sử dụng các nguyên liệu giàu dinh dưỡng và tốt cho sức khỏe đi kèm với các hướng dẫn chi tiết giúp người dùng dễ dàng thực hiện ngay cả khi họ không phải là đầu bếp chuyên nghiệp.

Việc sử dụng các công thức của Dr. Greger trong ứng dụng này không chỉ nhằm tạo ra các món ăn ngon mà còn là để cung cấp các lựa chọn thực phẩm lành mạnh, dựa trên nền tảng dinh dưỡng khoa học đã được chứng minh. Đây là một bước tiến quan trọng trong việc giúp người tiêu dùng dễ dàng tiếp cận với chế độ ăn thuần chay mà không cần lo lắng về chất lượng dinh dưỡng.

2.2. Tính toán tiến hóa (Evolutional Computation) và thuật toán di truyền (Genetic Algorithm)

2.2.1. Tổng quan về thuật toán tính toán tiến hóa (Evolutionary Computation)

Evolutionary Computation (EC) là một nhánh chuyên sâu của trí tuệ nhân tạo (AI) và học máy (Machine Learning), áp dụng các thuật toán lấy cảm hứng từ nguyên lý tiến hóa tự nhiên để giải quyết các vấn đề phức tạp. Evolutionary Computation tái hiện các khái niệm như chọn lọc tự nhiên, giao phối, đột biến, lai ghép, và thích nghi. Mục tiêu của phương pháp này là tìm kiếm giải pháp tốt nhất hoặc tối ưu nhất cho một vấn đề thông qua việc mô phỏng quá trình tiến hóa của sinh vật trong tự nhiên.

EC bao gồm một loạt các kỹ thuật tối ưu hóa, với các thuật toán tiến hóa (**Evolutionary Algorithms - EA**) là công cụ cốt lõi. Những thuật toán này sử dụng các nguyên lý tiến hóa để dần dần cải thiện các giải pháp, từ đó giúp tìm ra kết quả tối ưu. Một số thuật toán tiến hóa phổ biến nhất trong lĩnh vực này bao gồm:

- Thuật toán di truyền (Genetic Algorithm – GA): Đây là phương pháp sử dụng các nguyên tắc chọn lọc tự nhiên để tìm kiếm giải pháp tốt nhất. GA được phát minh vào những năm 1950 để giải quyết các bài toán liên quan đến điều khiển tự động.

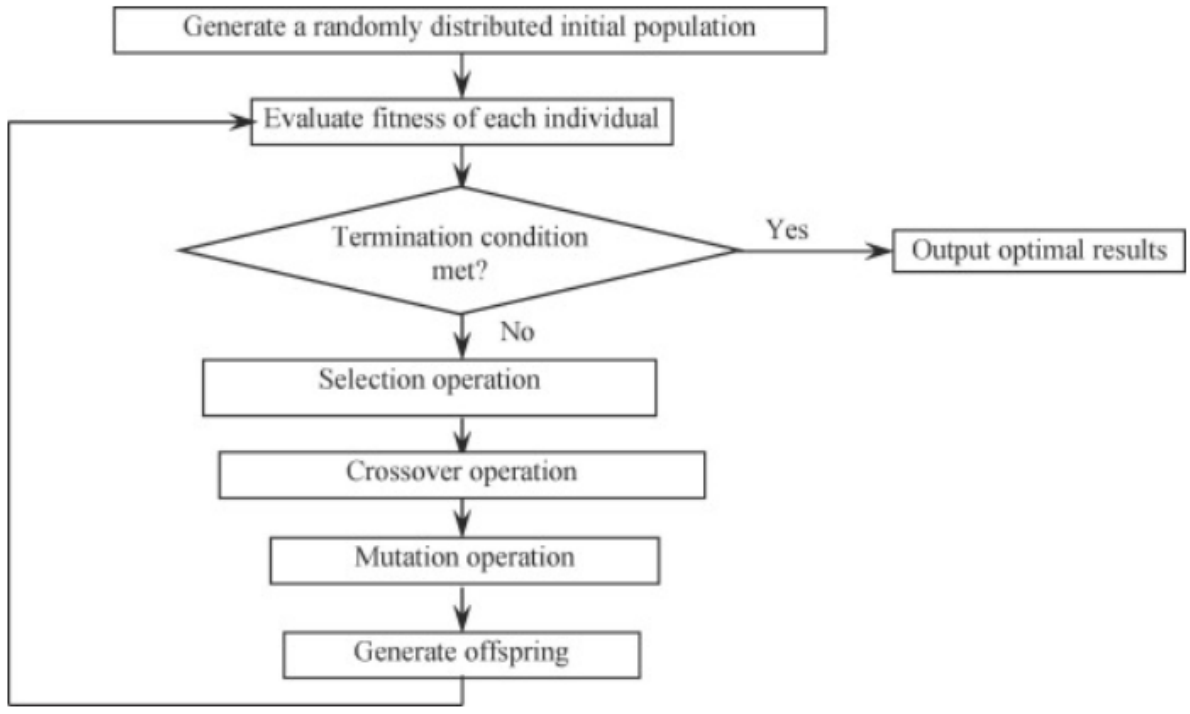
- Chiến lược tiến hóa (Evolutionary Strategies – ES): Phương pháp này tập trung vào việc giải quyết các bài toán tối ưu hóa số học, tức là tìm ra cách tính toán tốt nhất cho các bài toán liên quan đến số liệu.

- Lập trình di truyền (Genetic Programming – GP): Vào những năm 1990, dựa trên GA, một nhà nghiên cứu người Mỹ đã phát triển GP để tìm cách tạo ra các chương trình máy tính một cách tự động. Phương pháp này sử dụng các cấu trúc cây để biểu diễn chương trình máy tính hoặc các biểu thức toán học và phát triển chúng qua các thế hệ, trong đó các cá thể là các chương trình hoặc biểu thức thay vì chuỗi nhị phân.

- Thuật toán tiến hóa vi phân (Differential Evolution – DE): Một thuật toán tối ưu hóa khác dựa trên sự khác biệt giữa các cá thể trong quần thể để hướng dẫn việc tìm kiếm. Nó sử dụng cập nhật vector để tìm ra lời giải tối ưu và được đánh giá cao nhờ tốc độ hội tụ nhanh cùng với khả năng tìm kiếm giải pháp tốt trong không gian giải pháp rộng.

2.2.2. Thuật toán di truyền (GA)

Genetic Algorithm (GA) là một trong những thuật toán tiến hóa phổ biến nhất và được sử dụng rộng rãi. Sơ đồ luồng (flowchart) cơ bản của GA được thể hiện dưới đây:



Hình 1: Flowchart của thuật toán di truyền (Genetic Algorithm)

1. Generate a randomly distributed initial population: Bước đầu tiên là tạo ra một quần thể gồm các cá thể ban đầu phân bố ngẫu nhiên, tức là tập hợp các giải pháp khả thi ban đầu. Các cá thể được biểu diễn dưới dạng mã hóa, cách mã hóa phổ biến là sử dụng chuỗi nhị phân, nhưng có thể là chuỗi số nguyên hoặc số thực tùy thuộc vào bài toán.

- Mã hóa nhị phân (Binary Encoding) Sử dụng chuỗi các bit (0 và 1) để biểu diễn các cá thể.

- Ví dụ: Chuỗi nhị phân 110101 có thể đại diện cho một giải pháp cụ thể.

- Mã hóa số thực (Real-valued Encoding): Sử dụng các số thực để biểu diễn các cá thể, phù hợp với các bài toán cần độ chính xác cao.

- Ví dụ: Một chuỗi các số thực như [3.14,2.71,1.62].

- Mã hóa thứ tự (Permutation Encoding): Sử dụng các chuỗi hoán vị để biểu diễn các cá thể, thường sử dụng trong các bài toán như bài toán đường đi của người bán hàng (Traveling Salesman Problem – TSP)

- Ví dụ: Dãy các số nguyên thể hiện vị trí hoặc thứ tự [5,2,4,1,3].

2. Evaluate fitness of each individual: Tiến hành đánh giá mức độ "phù hợp" của từng cá thể. Mỗi cá thể sẽ được đánh giá dựa trên một hàm thích nghi (Fitness Function) để xem giải pháp đó tốt đến đâu.

3. Termination condition met? Yes or No: Kiểm tra xem điều kiện dừng đã đạt được chưa. Nếu có, quy trình kết thúc và đưa ra kết quả tốt nhất.

- Yes: Nếu điều kiện dừng đã thỏa mãn, quy trình sẽ dừng lại và xuất ra kết quả tốt nhất. Tiêu chí dừng để kết thúc thuật toán có thể là: Số thế hệ (Generation) đạt mức tối đa, không có cải thiện nào về giá trị thích nghi trong một số thế hệ liên tiếp hoặc giá trị của hàm thích nghi đạt ngưỡng mong muốn.

- No: Nếu chưa đạt điều kiện dừng, quy trình sẽ tiếp tục với bước tiếp theo.

4. Selection operation: Chọn lọc các cá thể tốt nhất (Selection) dựa trên mức độ phù hợp để tiếp tục. Các cá thể có "điểm số" tốt hơn sẽ có cơ hội cao hơn để được chọn. Một số phương pháp chọn lọc phổ biến là:

- Roulette Wheel Selection: Dựa trên xác suất, cá thể có giá trị thích nghi cao hơn sẽ có nhiều khả năng được chọn.

- Hàm chọn lọc: $P_i = \frac{f(x_i)}{\sum_{j=1}^{N} f(x_i)}$

P_i là xác suất chọn cá thể i.

$f(x_i)$ là giá trị thích nghi của cá thể i.
N là tổng số cá thể trong quần thể.

- Tournament Selection: Chọn một nhóm nhỏ các cá thể ngẫu nhiên và chọn cá thể tốt nhất trong nhóm.

- Rank Selection: Sắp xếp các cá thể dựa trên thứ hạng của chúng và xác suất chọn phụ thuộc vào thứ hạng.

5. Crossover operation: Thực hiện phép lai (crossover), nơi hai hoặc nhiều cá thể được kết hợp lại với nhau để tạo ra cá thể con mới. Đây là quá trình trao đổi thông tin giữa các cá thể để tạo ra những giải pháp mới.

- Lai ghép hai cá thể cha mẹ P_1 và P_2 để tạo ra cá thể con: $O = crossover(P_1, P_2)$

- Có nhiều kiểu lai ghép như Single-point crossover, Two-point crossover, và Uniform crossover.

- Ví dụ:
    - P_1 = 1100|1101
    - P_2 = 1011|0001
    - Kết quả:
        - O_1 = 1100|0001
        - O_2 = 1011|1101

6. Mutation operation: Thực hiện đột biến (mutation), thay đổi ngẫu nhiên một số phần tử của các cá thể con để tạo ra sự đa dạng. Điều này giúp tránh việc rơi vào các cực trị địa phương.

- Đột biến với một xác suất Pm:

$$x_{new} = x + Pm \cdot (random\_value)$$

- Xác suất đột biến thường được chọn nhỏ (như 0.01 – 0.1) để tránh phá hủy cấu trúc tốt của cá thể.

7. Generate offspring: Tạo ra thế hệ con mới từ các cá thể được chọn, lai và đột biến. Thế hệ mới này sẽ thay thế thế hệ cũ trong lần lặp tiếp theo.

Quá trình này sẽ được lặp lại liên tục cho đến khi điều kiện dừng được thỏa mãn, sau đó sẽ đưa ra kết quả tối ưu nhất tìm được.

2.2.3. Ứng dụng của thuật toán di truyền

Trong những năm gần đây, các thuật toán GA được ứng dụng rộng rãi trong nhiều lĩnh vực như:

- Tối ưu hóa thiết kế kỹ thuật: Tìm các thiết kế kỹ thuật tối ưu.

- Machine Learning: Tối ưu hóa cấu trúc mạng neural hoặc chọn tham số tốt nhất.

- Tối ưu hóa hệ thống giao thông: Giải quyết các bài toán như TSP hoặc tối ưu hóa lộ trình.

- Robot Learning: Tìm kiếm chiến lược tốt nhất cho các robot tự động.

- Tối ưu hóa tài chính: Phân tích và tối ưu hóa danh mục đầu tư.

Trong lĩnh vực công nghệ thực phẩm, GA có thể được sử dụng để tối ưu hóa công thức nấu ăn bằng cách tìm kiếm sự kết hợp tốt nhất giữa các nguyên liệu sao cho món ăn có hương vị ngon miệng, giá trị dinh dưỡng cao và đáp ứng yêu cầu của người tiêu dùng. Các ứng dụng phổ biến của GA trong lĩnh vực này bao gồm:

- Tối ưu hóa công thức thực phẩm: GA có thể giúp các nhà phát triển sản phẩm thực phẩm tìm ra công thức tối ưu cho một loại thực phẩm mới bằng cách kết hợp các thành phần khác nhau và thử nghiệm trên một số lượng lớn các biến thể.

- Phát triển thực phẩm chức năng: Với nhu cầu ngày càng tăng về các sản phẩm thực phẩm có lợi cho sức khỏe, GA có thể được sử dụng để xác định sự kết hợp tối ưu của các nguyên liệu để tạo ra các sản phẩm thực phẩm chức năng, giàu chất dinh dưỡng và phù hợp với các chế độ ăn đặc biệt như thuần chay.

- Cá nhân hóa công thức: GA có thể giúp cá nhân hóa công thức nấu ăn theo yêu cầu riêng của từng người tiêu dùng, dựa trên sở thích cá nhân, nhu cầu dinh dưỡng, hoặc các giới hạn về nguyên liệu. Điều này đặc biệt quan trọng trong các ứng dụng dinh dưỡng và chăm sóc sức khỏe, nơi mà yêu cầu về sự chính xác và phù hợp của công thức là rất cao.

3. Lý do lựa chọn Evolutionary Computation cho bài toán

3.1. Modeling vs. Creativity trong Học Máy

Trong học máy, hai khái niệm “Modeling” (mô hình hóa) và “Creativity” (tính sáng tạo) thường được sử dụng để phân biệt hai cách tiếp cận chính đối với việc giải quyết các bài toán.

- Modeling: Phần lớn AI hiện nay đều là modeling, đây là cách tiếp cận dựa trên việc xây dựng các mô hình từ dữ liệu đã biết để dự đoán và đưa ra quyết định trong tương lai. Các thuật toán như Neural Networks (Mạng Nơ-ron) và Reinforcement Learning dựa vào dữ liệu giám sát hoặc các ví dụ cụ thể để tìm ra giải pháp. Điều này hiệu quả cho các bài toán mà giải pháp đúng đã được xác định rõ. Những phương pháp này chủ yếu hoạt động dựa trên leo đồi “Hill Climbing”.

- Creativity, mặt khác, không có dữ liệu định hướng cụ thể và đòi hỏi sự khám phá trong không gian tìm kiếm chưa được biết đến. Đây là nơi Evolutionary Computation nổi bật, vì nó không chỉ dựa vào dữ liệu quá khứ mà còn có khả năng tạo ra các giải pháp mới mẻ, độc đáo.

3.2. Leo đồi (Hill Climbing) trong các phương pháp Học Máy hiện đại

Phần lớn các phương pháp học máy hiện đại như Neural Networks và Deep Learning đều dựa trên khái niệm "leo đồi" (hill climbing), nơi các thuật toán tối ưu hóa dựa vào gradient được tính toán từ dữ liệu đã biết. Quá trình này cho thấy cách mà một mô hình nên được điều chỉnh để cải thiện dần dần. Tuy nhiên, nó có các hạn chế quan trọng:

- Thiếu cái nhìn tổng thể: Các phương pháp học máy truyền thống như Deep Learning thường chỉ dựa vào việc cải thiện cục bộ từ một điểm ban đầu. Gradient chỉ cho biết cách điều chỉnh nhỏ để đạt hiệu quả tốt hơn mà không có cái nhìn tổng quan về toàn bộ không gian tìm kiếm, tức là không biết nên bắt đầu từ đâu hoặc leo đồi nào.

- Học Tăng Cường (Reinforcement Learning) cũng gặp khó khăn tương tự. Nó bắt đầu với một giải pháp cá nhân và tìm kiếm xung quanh để điều chỉnh, nhưng quá trình này vẫn dựa vào việc khám phá cục bộ từ điểm xuất phát ban đầu và dễ dàng bị mắc kẹt trong các đỉnh cực trị cục bộ.

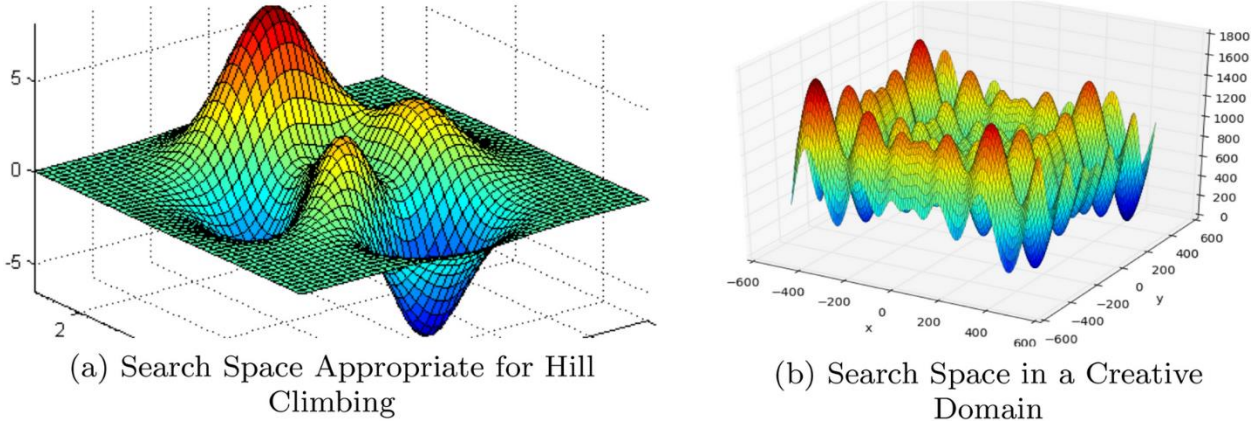
3.3. Những thách thức trong không gian tìm kiếm của các nhiệm vụ sáng tạo

Các nhiệm vụ sáng tạo không thuận lợi cho việc leo đồi truyền thống vì không gian tìm kiếm của chúng có những đặc điểm sau:

- Không gian rất lớn: Trong các bài toán phức tạp như dinh dưỡng thuần chay, không gian tìm kiếm bao gồm vô số giải pháp có thể. Điều này khiến việc khám phá toàn bộ không gian trở nên khó khăn, ngay cả khi thực hiện nhiều lần khởi động lại từ các điểm khác nhau.

- Không gian nhiều chiều: Bài toán tối ưu hóa dinh dưỡng yêu cầu tìm kiếm giá trị tốt nhất cho nhiều biến số cùng một lúc, ví dụ như nguyên liệu, hương vị, và số khẩu phần. Điều này tạo ra một không gian tìm kiếm cao chiều, làm cho việc tối ưu hóa trở nên phức tạp hơn.

- Không gian lừa dối (deceptive): Không gian tìm kiếm có thể chứa nhiều đỉnh và thung lũng, làm cho việc tìm kiếm cục bộ trở nên khó khăn. Các thuật toán truyền thống dễ bị lừa vào các đỉnh cực trị cục bộ và bỏ lỡ các giải pháp tốt hơn nằm ở các vùng khác của không gian.



(a) Search Space Appropriate for Hill Climbing

(b) Search Space in a Creative Domain

Hình 2: Không gian tìm kiếm trong Hill Climbing và trong các nhiệm vụ sáng tạo.

Hình (a): Không gian tìm kiếm đơn giản, có một số ít các đỉnh và thung lũng nên các thuật toán Hill Climbing có thể hoạt động hiệu quả, vì chúng chỉ cần đi theo gradient để đạt được đỉnh cao nhất. Khi một giải pháp tốt cục bộ được tìm thấy, hill climbing có thể dễ dàng điều hướng đến điểm cao hơn nếu không có các rào cản hoặc đỉnh cực trị khác ngăn cản. Không gian này ít chứa các điểm lừa dối, nên việc leo đồi để tìm giải pháp tối ưu không gặp quá nhiều khó khăn.

Hình (b) Không gian tìm kiếm phức tạp hơn, với nhiều đỉnh và thung lũng nằm rải rác. Không gian này phản ánh những bài toán sáng tạo, nơi mà các giải pháp tốt nhất có thể bị "ẩn giấu" giữa các điểm cực trị cục bộ và không dễ dàng để tìm ra bằng các thuật toán cục bộ như hill climbing.

3.4. Evolutionary Computation (EC) – Giải pháp cho các thách thức sáng tạo

Để đối mặt với những thách thức về không gian tìm kiếm cho các nhiệm vụ sáng tạo, Evolutionary Computation cung cấp một cách tiếp cận khác biệt và hiệu quả hơn so với các phương pháp truyền thống:

- Tìm kiếm dựa trên quần thể: Thay vì chỉ dựa vào một giải pháp cá nhân như trong hill climbing, EC tìm kiếm dựa trên một quần thể các cá thể. Điều này cho phép nó khám phá đồng thời nhiều khu vực khác nhau của không gian tìm kiếm, giúp giảm thiểu khả năng bị mắc kẹt trong các đỉnh cực trị cục bộ.

- Khả năng chia sẻ thông tin giữa các tìm kiếm song song: Trong EC, các tìm kiếm song song có thể cải thiện tuyến tính thông qua sự chia sẻ thông tin. Nếu một giải pháp tốt được tìm thấy ở một phần của quần thể, các cá thể khác có thể ngay lập tức tận dụng thông tin này. Điều này giúp EC tạo ra các giải pháp tốt hơn thông qua việc tìm kiếm các "khối xây dựng" (schemata) hoặc các bước nhảy tạm thời (stepping stones), sau đó kết hợp chúng để tạo ra các giải pháp tối ưu hơn.

  - Ví dụ về Multiplexer: EC đã được chứng minh là hiệu quả trong việc giải quyết bài toán multiplexer, nơi không gian tìm kiếm lớn và phức tạp với các giá trị logic. EC có thể phát hiện các cấu trúc hữu ích trong không gian này và tận dụng chúng để tìm ra các giải pháp hiệu quả mà các thuật toán khác bỏ qua.

- Khả năng xử lý không gian cao chiều: EC có thể giải quyết các bài toán với không gian nhiều chiều phức tạp, nơi mà các thuật toán truyền thống gặp khó khăn. Ví dụ:

  - Bài toán đúc kim loại: Để tối ưu hóa việc sản xuất với hàng tỷ biến số, EC sử dụng các kỹ thuật đặc biệt để khai thác cấu trúc của bài toán và tìm ra các giải pháp tối ưu, ngay cả khi số lượng biến rất lớn.

- Khả năng thích ứng với không gian lừa dối:

  - Đa mục tiêu: EC có thể tối ưu hóa đa mục tiêu, cho phép nó tìm kiếm theo nhiều hướng khác nhau cùng lúc. Nếu một chiều tìm kiếm bị mắc kẹt, nó có thể chuyển sang các chiều khác để thoát khỏi các đỉnh cực trị cục bộ.
  - Nhấn mạnh sự đa dạng và mới lạ: EC có thể được điều chỉnh để nhấn mạnh sự khác biệt và sáng tạo trong giải pháp, không chỉ tối ưu hóa các mục tiêu cụ thể. Điều này giúp EC tránh bị lừa bởi không gian tìm kiếm và phát hiện ra những giải pháp mới mà các thuật toán khác có thể bỏ qua.
  - Ví dụ về tính mới lạ trong EC: Trong các phương pháp kết hợp tính mới lạ, EC không chỉ tập trung vào việc tối ưu hóa các mục tiêu chính như hiệu suất mà còn khuyến khích các giải pháp mới lạ và khác biệt so với những gì đã tồn tại trong

quần thể trước đó. Điều này giúp tránh sự lừa dối của không gian tìm kiếm, nơi mà các giải pháp tốt nhất có thể bị bỏ qua nếu chỉ tối ưu hóa dựa trên hiệu suất.

Chính vì vậy, Evolutionary Computation là một giải pháp mạnh mẽ cho các bài toán đòi hỏi tính sáng tạo cao, đặc biệt là trong không gian tìm kiếm phức tạp và lừa dối như bài toán dinh dưỡng thuần chay. Trong khi các phương pháp học máy truyền thống thường gặp giới hạn trong việc leo đồi, EC mở rộng khả năng tìm kiếm với cách tiếp cận dựa trên quần thể, cho phép khám phá các giải pháp mới mẻ và tối ưu hơn. Điều này giúp nó trở thành lựa chọn lý tưởng cho các bài toán yêu cầu sự sáng tạo và khả năng thích ứng linh hoạt với không gian tìm kiếm phức tạp.

4. Phương pháp nghiên cứu

4.1. Mô tả bài toán

Mục tiêu của bài toán là xây dựng một ứng dụng dựa trên dòng lệnh, cho phép người dùng tạo ra công thức nấu ăn thuần chay dựa trên những nguyên liệu đầu vào mà họ cung cấp. Ứng dụng này sử dụng thuật toán Genetic Algorithm (GA) để tối ưu hóa việc lựa chọn nguyên liệu và các bước hướng dẫn nấu ăn, nhằm tạo ra một công thức tối ưu về mặt hương vị và khẩu phần.

4.1.1. Yếu tố đầu vào (Các ràng buộc của người dùng)

- Nguyên liệu:

- Người dùng cần cung cấp ít nhất 3 nguyên liệu mà họ muốn có trong món ăn
- Nguyên liệu có thể bao gồm rau củ, gia vị hoặc các thành phần dinh dưỡng khác mà người dùng muốn sử dụng

- Hương vị ưu tiên và không thích:

- Người dùng có thể chỉ định các hương vị mà họ muốn món ăn có nhiều hơn hoặc ít hơn, bao gồm các tùy chọn như Sweet (Ngọt), Bitter (Đắng), Savory (Đậm đà), Spicy (Cay), và Sour (Chua)
- Điều này sẽ ảnh hưởng đến quá trình tối ưu hóa nguyên liệu và gia vị được chọn.

- Số khẩu phần ăn:

- Người dùng nhập số lượng khẩu phần mong muốn (servings) từ 1 đến 10, giúp điều chỉnh lượng nguyên liệu phù hợp với số người ăn.

4.1.2. Yếu tố đầu ra

- Danh sách nguyên liệu tối ưu:

- Danh sách nguyên liệu sẽ được chọn trên các nguyên liệu mà người dùng cung cấp, kết hợp với các gia vị và nguyên liệu bổ sung nhằm đạt được hương vị tối ưu theo yêu cầu.
- Lượng nguyên liệu sẽ được điều chỉnh phù hợp với số khẩu phần ăn.

- Các bước hướng dẫn nấu ăn: Một chuỗi các bước hướng dẫn nấu ăn được tối ưu hóa sẽ được tạo ra dựa trên các nguyên liệu đã chọn, nhằm đảm bảo rằng món ăn được chế biến dễ dàng và đạt được hương vị mong muốn.

- Thông tin dinh dưỡng: Ứng dụng sẽ cung cấp các thông tin dinh dưỡng như lượng calorie, protein, carbohydrate và chất béo của món ăn được tạo ra, dựa trên API dinh dưỡng của Edamam.

4.2. Thiết kế thuật toán di truyền (Genetic Algorithm – GA)

Việc thiết kế thuật toán di truyền được chia làm hai phần chính, một là thuật toán di truyền để tối ưu hóa tạo ra các nguyên liệu đầu ra (GA Ingredients) và hai là thuật toán di truyền để tối ưu hóa tạo ra hướng dẫn nấu ăn dựa trên các nguyên liệu đầu ra vừa được tối ưu hóa (GA Directions).

4.2.1. Thuật toán di truyền tối ưu nguyên liệu (GA Ingredients)

4.2.1.1. Khởi tạo quần thể ban đầu

Mỗi cá thể là một danh sách nguyên liệu ban đầu, bao gồm nguyên liệu của người dùng và các thảo mộc hoặc gia vị được chọn ngẫu nhiên.

```python
# Genetic Algorithm Functions
# Hàm tạo cá thể cho GA cho Ingredients
def create_individual(user_ingredients, herbs_spices):
  base_ingredients = set(user_ingredients.copy())
  additional_herbs_spices = sample([herb for herb in herbs_spices if herb not in base_ingredients], randint(2, 5))
  return list(base_ingredients.union(additional_herbs_spices))
```

Hình 3: Hàm tạo cá thể cho thuật toán di truyền tối ưu nguyên liệu

- user_ingredients: Nguyên liệu mà người dùng có sẵn.

- herbs_spices: Danh sách thảo mộc/gia vị

- additional_herbs_spices: Lựa chọn ngẫu nhiên các gia vị để thêm vào cá thể.

4.2.1.2. Các hàm của thuật toán di truyền (GA Ingredients)

a. Hàm đánh giá fitness

Hàm evaluate_individual dùng để tính điểm fitness của mỗi cá thể dựa trên việc các nguyên liệu phù hợp với **khẩu vị** của người dùng.

```python
def evaluate_individual(individual, flavor_preferences, user_ingredients, diversity_bonus=2):
    """Evaluate the fitness of an individual based on user preferences."""
    score = 0
    preferred_weight = 6
    disliked_weight = -5
    user_ingredient_weight = 8

    for ingredient in user_ingredients:
      if ingredient in individual:
        score += user_ingredient_weight

    for ingredient in individual:
        # Evaluate if the ingredient matches the preferred flavors
        for flavor in flavor_preferences['more']:
            if synchronize_df.loc[synchronize_df['Ingredient Name'] == ingredient, flavor].any():
                score += preferred_weight
        # Penalize if the ingredient matches the disliked flavors
        for flavor in flavor_preferences['less']:
            if synchronize_df.loc[synchronize_df['Ingredient Name'] == ingredient, flavor].any():
                score += disliked_weight

    unrelated_ingredients = set(individual) - set(user_ingredients)
    if len(unrelated_ingredients) > len(user_ingredients) / 2:
      score -= 10

    # Bonus điểm dựa trên độ đa dạng của nguyên liệu
    unique_herbs_spices = set([i for i in individual if i in herbs_and_spices_keywords])
    score += len(unique_herbs_spices) * diversity_bonus
```

Hình 4: Hàm đánh giá fitness của các cá thể nguyên liệu

Hàm evaluate_individual đánh giá các cá thể nguyên liệu (individual) dựa trên các yếu tố sau:

- Điểm số score: bắt đầu từ 0.

- Trọng số:

- preferred_weight = 6: cộng điểm nếu nguyên liệu có hương vị yêu thích.
- disliked_weight = -5: trừ điểm nếu nguyên liệu có hương vị không thích.
- user_ingredient_weight = 8: cộng điểm nếu công thức có nguyên liệu người dùng yêu cầu.
- diversity_bonus = 2: thưởng điểm cho mỗi loại thảo mộc/gia vị độc đáo.

- Cách xử lý:

- Cộng điểm: Nếu công thức có nguyên liệu trong user_ingredients, cộng user_ingredient_weight vào score.
- Thêm/Trừ điểm dựa trên sở thích hương vị trong flavor_preferences:
    - Nếu nguyên liệu có trong more, cộng preferred_weight.
    - Nếu có trong less, trừ disliked_weight.
- Phạt điểm nếu có quá nhiều nguyên liệu không liên quan (-10).

- Thưởng điểm đa dạng: với mỗi thảo mộc/gia vị trong herbs_and_spices_keywords, cộng diversity_bonus.
- Hàm trả về tổng score.

b. Hàm lai ghép tùy chỉnh

Hàm custom_crossover thực hiện lai ghép giữa hai cá thể, giữ lại nguyên liệu chính của người dùng.

```python
def custom_crossover(ind1, ind2, user_ingredients, swap_prob=0.5):
    """Hàm crossover tùy chỉnh sử dụng phương pháp Blend nhưng bảo vệ nguyên liệu của người dùng."""
    min_length = min(len(ind1), len(ind2))

    for i in range(min_length):
        # Chỉ hoán đổi nếu gene không phải là nguyên liệu của người dùng
        if ind1[i] not in user_ingredients and ind2[i] not in user_ingredients:
            # Với xác suất `swap_prob`, hoán đổi các gene giữa ind1 và ind2
            if random.random() < swap_prob:
                ind1[i], ind2[i] = ind2[i], ind1[i]

    # Đảm bảo các cá thể trả về là dạng `creator.Individual`
    ind1 = creator.Individual(dict.fromkeys(ind1))
    ind2 = creator.Individual(dict.fromkeys(ind2))

    return ind1, ind2
```

Hình 5: Hàm lai ghép custom_crossover cho tối ưu nguyên liệu

Hàm này thực hiện việc lai chéo (crossover) giữa hai cá thể ind1 và ind2, với các quy tắc sau:

- Bảo vệ nguyên liệu của người dùng: Nếu một gene (nguyên liệu) thuộc user_ingredients, nó sẽ không bị hoán đổi giữa ind1 và ind2. Điều này đảm bảo giữ nguyên những nguyên liệu mà người dùng muốn duy trì trong cả hai cá thể.

- Xác suất hoán đổi: Với mỗi gene không thuộc user_ingredients, hàm có 50% xác suất (hoặc giá trị swap_prob đã chỉ định) để hoán đổi giữa ind1 và ind2.

- Chuyển đổi dữ liệu: Cuối cùng, ind1 và ind2 được chuyển thành dạng creator.Individual, có thể là một lớp tùy chỉnh để hỗ trợ các thao tác di truyền. Việc sử dụng dict.fromkeys() loại bỏ các phần tử trùng lặp trong các cá thể.

Hàm bảo vệ các nguyên liệu của người dùng trong quá trình crossover và tạo ra các cá thể mới với các gene có thể hoán đổi với nhau một cách ngẫu nhiên nhưng có kiểm soát.

c. Hàm đột biến dựa trên sở thích hương vị

Hàm flavor_based_mutation thực hiện đột biến dựa trên hương vị của từng cá thể (individual) trong thuật toán di truyền. Nó thay thế các nguyên liệu trong cá thể hiện tại bằng các lựa chọn gia vị hoặc nguyên liệu phù hợp với sở thích hương vị đã được xác định.

```python
#Đột biến trên hương vị
def flavor_based_mutation(individual, flavor_preferences, herbs_spices, user_ingredients, indpb=0.1):
  for i in range(len(individual)):
    if individual[i] not in user_ingredients and random.random() < indpb:
      possible_choices = [herb for herb in herbs_spices if herb not in individual]
      selected_flavor = random.choice(flavor_preferences['more'])
      filtered_choices = [choice for choice in possible_choices if synchronize_df.loc[synchronize_df['Ingredient Name'] == choice, selected_flavor].any()]
      if filtered_choices:
        individual[i] = random.choice(filtered_choices)
      else:
        individual[i] = random.choice(possible_choices)

  individual = creator.Individual(dict.fromkeys(individual))
  return individual,
```

Hình 6: Hàm đột biến dựa trên sở thích hương vị cho tối ưu nguyên liệu

- Duyệt từng nguyên liệu trong individual: Nếu nguyên liệu này không có trong user_ingredients (nghĩa là nguyên liệu không phải do người dùng bắt buộc) và một số ngẫu nhiên nhỏ hơn indpb, quá trình đột biến sẽ áp dụng lên nguyên liệu này.

- Lựa chọn các gia vị có thể thay thế (possible_choices): Lấy danh sách gia vị và thảo mộc từ herbs_spices nhưng không có trong individual hiện tại, giúp tạo ra sự đa dạng mà không bị lặp lại nguyên liệu.

- Chọn hương vị ưu tiên: Một hương vị từ flavor_preferences['more'] được chọn ngẫu nhiên để tăng thêm phù hợp với sở thích người dùng.

- Lọc các lựa chọn phù hợp với hương vị ưu tiên: Từ possible_choices, chỉ giữ lại những nguyên liệu có hương vị phù hợp với selected_flavor. Bộ dữ liệu về nguyên liệu synchronize_df được dùng để xác nhận hương vị của từng gia vị trong bộ dữ liệu.

- Thay thế nguyên liệu: Nếu filtered_choices có chứa các nguyên liệu phù hợp, chọn một nguyên liệu ngẫu nhiên từ đó để thay thế. Nếu không, chọn một nguyên liệu bất kỳ từ possible_choices.

- Sử dụng dict.fromkeys(individual) để loại bỏ các nguyên liệu trùng lặp trong cá thể. Sau đó, trả về cá thể đã qua đột biến, sẵn sàng cho vòng đánh giá fitness tiếp theo.

d. Hàm chọn lọc

Hàm rank_based_selection thực hiện quá trình chọn lọc dựa trên thứ hạng và elitism trong thuật toán di truyền, nhằm đảm bảo rằng các cá thể có fitness cao hơn có cơ hội được chọn cao hơn, đồng thời vẫn giữ lại những cá thể tốt nhất (elite).

```python
# Hàm rank-based selection mới
def rank_based_selection(population, k, elitism_rate=0.15):
    """Rank-based Selection with Elitism."""
    # Xếp hạng các cá thể dựa trên fitness của chúng (từ cao xuống thấp)
    ranked_population = sorted(population, key=lambda ind: ind.fitness.values[0], reverse=True)

    # Giữ lại các cá thể tốt nhất dựa trên elitism rate
    elite_count = int(len(population) * elitism_rate)
    elite_individuals = ranked_population[:elite_count]

    # Tính tổng số thứ hạng
    total_ranks = sum(range(1, len(ranked_population) + 1))

    # Xác suất chọn dựa trên thứ hạng
    probabilities = [(len(ranked_population) - rank + 1) / total_ranks for rank in range(1, len(ranked_population) + 1)]

    # Chọn cá thể ngẫu nhiên dựa trên xác suất từ thứ hạng
    selected_individuals = random.choices(ranked_population, weights=probabilities, k=k - elite_count)

    # Kết hợp các cá thể elite với các cá thể được chọn
    return elite_individuals + selected_individuals
```

Hình 7: Hàm chọn lọc rank_based_selection cho tối ưu nguyên liệu

- Xếp hạng cá thể theo fitness: Quần thể (population) được sắp xếp giảm dần dựa trên fitness, từ cá thể có fitness cao nhất đến thấp nhất.

- Chọn cá thể tốt nhất (Elitism): Số lượng cá thể "elite" được xác định dựa trên elitism_rate. elite_individuals chứa các cá thể hàng đầu này và sẽ được giữ nguyên qua thế hệ sau.

- Tính tổng thứ hạng: total_ranks để tính tổng thứ hạng của tất cả cá thể, các thứ hạng này sẽ dùng để tính xác suất chọn lọc.

- Tạo xác suất dựa trên thứ hạng (probabilities): Cá thể xếp hạng cao nhất có xác suất cao nhất, và xác suất giảm dần cho các cá thể có xếp hạng thấp hơn.

- Dùng random.choices để chọn cá thể từ ranked_population với trọng số xác suất từ thứ hạng, sao cho có k - elite_count cá thể được chọn ngẫu nhiên.

- Trả về quần thể mới, gồm các cá thể "elite" cùng với các cá thể được chọn ngẫu nhiên theo xác suất từ thứ hạng.

4.2.1.3. Thuật toán GA cho nguyên liệu

Đây là phần chính của thuật toán di truyền tối ưu nguyên liệu dựa trên sở thích hương vị của người dùng. Mã sẽ trải qua nhiều thế hệ, chọn lọc, lai ghép và đột biến để tìm ra danh sách nguyên liệu tối ưu.

a. Khởi tạo Toolbox và đăng ký các hàm cơ bản cho thuật toán di truyền

```python
def genetic_algorithm_ingredients(user_ingredients, herbs_spices, flavor_preferences, ngen=50, pop_size=150, cxpb=0.6, mutpb=0.15):
    toolbox = base.Toolbox()
    toolbox.register("individual", tools.initIterate, creator.Individual, lambda: create_individual(user_ingredients, herbs_spices))
    toolbox.register("population", tools.initRepeat, list, toolbox.individual)
    toolbox.register("mate", custom_crossover, user_ingredients=user_ingredients, swap_prob=0.5)
    toolbox.register("mutate", flavor_based_mutation, flavor_preferences=flavor_preferences, herbs_spices=herbs_spices, user_ingredients=user_ingredients, indpb=0.2)
    toolbox.register("select", rank_based_selection)
    toolbox.register("evaluate", lambda ind: evaluate_individual(ind, flavor_preferences, user_ingredients, diversity_bonus=2))
```

Hình 8: Khởi tạo toolbox đăng ký hàm cho GA Ingredients

- toolbox là một công cụ từ thư viện DEAP để chứa các hàm phục vụ thuật toán di truyền. Ở đây, chúng ta đăng ký hàm individual, đại diện cho một danh sách nguyên liệu.

- Hàm create_individual tạo một cá thể chứa danh sách nguyên liệu kết hợp từ user_ingredients và herbs_spices (những gia vị tùy chọn), tạo ra tính phong phú trong các lựa chọn ban đầu cho quần thể.

- population: Hàm population tạo một quần thể với số lượng cá thể nhất định để bắt đầu thuật toán di truyền, tạo nên sự đa dạng ban đầu trong các giải pháp.

- Lai ghép (mate): custom_crossover là một hàm lai ghép tùy chỉnh, cho phép kết hợp hai cá thể (danh sách nguyên liệu) để tạo ra cá thể con. Việc truyền vào user_ingredients và swap_prob nhằm bảo đảm rằng các nguyên liệu người dùng chọn sẽ ít bị thay đổi trong quá trình lai ghép.

- Đột biến (mutate): Hàm flavor_based_mutation thực hiện đột biến dựa trên sở thích hương vị (flavor_preferences), tăng khả năng đột biến cho các gia vị và nguyên liệu tùy chọn, giúp danh sách nguyên liệu đáp ứng tốt hơn sở thích của người dùng.

- Chọn lọc (select): Sử dụng rank_based_selection, ưu tiên các cá thể có điểm fitness cao hơn.

- Đánh giá (evaluate): Hàm evaluate_individual tính điểm fitness cho mỗi cá thể dựa trên mức độ phù hợp với sở thích hương vị của người dùng, với các tham số như diversity_bonus để khuyến khích sự đa dạng trong các cá thể.

b. Tạo quần thể và cập nhật Fitness với Fitness Sharing

```
population = toolbox_directions.population(n=pop_size)
invalid_ind = [ind for ind in population if not ind.fitness.valid]
update_population_fitness_with_sharing_directions(invalid_ind, relevant_ingredients, synchronize_df, process_data, sharing_radius=3.0)
adjust_fitness(invalid_ind)
```

Hình 9: Tạo quần thể và cập nhật fitness cho tối ưu nguyên liệu

- Quần thể (population): population tạo một số lượng cá thể nhất định dựa trên pop_size, đại diện cho nhiều tổ hợp nguyên liệu khác nhau.

- Cập nhật fitness với Fitness Sharing: Hàm update_population_fitness_with_sharing giảm điểm fitness cho các cá thể quá giống nhau trong một bán kính chia sẻ (sharing_radius), khuyến khích đa dạng và tránh việc thuật toán rơi vào cực trị cục bộ. Điều này có vai trò quan trọng trong tối ưu hóa bởi giúp duy trì nhiều hướng giải pháp khác nhau trong không gian tìm kiếm.

c. Vòng lặp qua các thế hệ

```
for gen in range(ngen):
  mutpb = adaptive_mutation_rate(population, gen, ngen)
  offspring = toolbox.select(population, len(population) - len(hof))
  offspring = list(map(toolbox.clone, offspring))
```

Hình 10: Vòng lặp qua các thế hệ (GA Ingredients)

- Vòng lặp (ngen): Thuật toán sẽ chạy qua một số lượng thế hệ xác định (ngen), mỗi thế hệ là một lần cải tiến dần dần trong quần thể.

- Điều chỉnh tỷ lệ đột biến (mutpb): adaptive_mutation_rate điều chỉnh tỷ lệ đột biến dựa trên mức độ hội tụ của quần thể, giúp tăng đột biến nếu quần thể quá đồng nhất để tránh rơi vào cực trị cục bộ.

- Chọn lọc (offspring): Lựa chọn các cá thể con qua toolbox.select dựa trên điểm fitness, duy trì cá thể tốt nhất từ Hall of Fame (hof), đồng thời tạo bản sao của cá thể con bằng toolbox.clone để sử dụng cho các bước tiếp theo.

d. Lai ghép và đột biến

```
# Lai ghép và đột biến
for child1, child2 in zip(offspring[::2], offspring[1::2]):
  if random.random() < cxpb:
    toolbox.mate(child1, child2)
    del child1.fitness.values
    del child2.fitness.values

for mutant in offspring:
  if random.random() < mutpb:
    if random.random() < 0.5:
      toolbox.mutate(mutant)
    del mutant.fitness.values
```

Hình 11: Vòng lặp lai ghép và đột biến (GA Ingredients)

- Lai ghép (cxpb): Với xác suất cxpb, hai cá thể con được lai ghép với nhau qua toolbox.mate để tạo ra tổ hợp nguyên liệu mới.

- Đột biến (mutpb): Mỗi cá thể có xác suất mutpb để đột biến, có thể giúp thay đổi các nguyên liệu nhằm đa dạng hóa các giải pháp. Điểm fitness của cá thể sau lai ghép và đột biến bị xóa (del fitness) để đảm bảo chúng được đánh giá lại.

e. Cập nhật Fitness và thêm các cá thể tốt nhất từ Hall of Fame

```
# Đánh giá lại các cá thể con sau khi lai ghép và đột biến
invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
update_population_fitness_with_sharing(invalid_ind, flavor_preferences, user_ingredients, sharing_radius=2.0)

# Cập nhật quần thể với các cá thể tốt nhất từ thế hệ trước
population[:] = offspring + list(hof)

# Cập nhật Elitism
hof.update(population)
```

Hình 12: Cập nhật fitness và thêm các thể tốt nhất từ HOF (GA Ingredients)

- Cập nhật fitness: Đánh giá lại fitness cho các cá thể mới (invalid_ind) bằng update_population_fitness_with_sharing.

- Kết hợp quần thể: Các cá thể con sau khi lai ghép, đột biến, và được điều chỉnh fitness sẽ được kết hợp vào quần thể cùng với các cá thể tốt nhất từ Hall of Fame để tạo thành quần thể mới cho thế hệ tiếp theo.

- Cập nhật Hall of Fame: Lưu giữ các cá thể tốt nhất qua từng thế hệ để đảm bảo không mất đi các giải pháp tối ưu.

f. Thống kê, điều chỉnh tỷ lệ đột biến dựa trên độ đa dạng và lấy cá thể tốt nhất cuối cùng

```
    # Thu thập số liệu thống kê
    record = stats.compile(population)
    print(f"Generation {gen}: {record}")

    # Điều chỉnh tỷ lệ đột biến dựa trên sự đa dạng sau mỗi 10 thế hệ
    if gen % 10 == 0:
      diversity = calculate_population_diversity(population)
      if diversity < 0.2:
        mutpb = min(mutpb + 0.05, 0.5)
      else:
        mutpb = max(mutpb - 0.05, 0.1)

  # Trả về cá thể tốt nhất từ Hall of Fame
  best_individual = tools.selBest(population, 1)[0]
  return best_individual
```

Hình 13: Thống kê, điều chỉnh và trả kết quả cuối cùng (GA Ingredients)

- Thống kê (stats): Ghi nhận các chỉ số như điểm fitness trung bình, tối thiểu, và tối đa của từng thế hệ, giúp người dùng theo dõi quá trình cải tiến.

- Điều chỉnh tỷ lệ đột biến: Mỗi 10 thế hệ, thuật toán đo độ đa dạng trong quần thể; nếu quần thể có độ đa dạng thấp (< 0.2), tỷ lệ đột biến được tăng lên để duy trì các lựa chọn đa dạng hơn trong quần thể.

- Kết quả cuối cùng: Sau khi hoàn thành các thế hệ, thuật toán sẽ trả về cá thể có điểm fitness cao nhất từ population, đại diện cho danh sách nguyên liệu tối ưu đáp ứng sở thích hương vị của người dùng.

4.2.2. Thuật toán di truyền tối ưu hướng dẫn nấu ăn (GA Directions)

4.2.2.1. Khởi tạo quần thể ban đầu

Hàm create_direction_individual tạo ra một cá thể mới cho các bước hướng dẫn (directions) trong thuật toán di truyền. Mục tiêu là chọn ngẫu nhiên một tập hợp các bước hướng dẫn từ danh sách all_steps, đồng thời loại bỏ các bước trùng lặp hoặc quá tương tự nhau dựa trên một ngưỡng nhất định (threshold).

```python
# Direction Handling and Genetic Algorithm
def create_direction_individual(all_steps, threshold=0.8):
  """Create an individual for recipe directions."""
  individual = []
  while len(individual) < randint(5,8):
    new_step = random.choice(all_steps)
    if not any(are_steps_similar(new_step, existing_step, threshold) for existing_step in individual):
      individual.append(new_step)
  individual = remove_duplicates(individual)
  return individual
```

Hình 14: Hàm tạo cá thể ban đầu cho thuật toán di truyền tối ưu hướng dẫn nấu ăn.

- all_steps là danh sách chứa tất cả các bước hướng dẫn nấu ăn (instructions) liên quan đến các nguyên liệu đã chọn. Hàm này thu thập và lọc các bước hướng dẫn từ hai nguồn dữ liệu (synchronize_df và recipes_instructions_df) dựa trên các nguyên liệu đầu vào (ingredients) và loại bỏ các bước không liên quan.

- Ở đây, sẽ tiến hành khởi tạo các cá thể rỗng: individual sẽ chứa các bước hướng dẫn được chọn.

- Sau đó, tiến hành lặp để thêm bước hướng dẫn ngẫu nhiên. While lặp cho đến khi individual có độ dài ngẫu nhiên từ 5 đến 8 bước. Biến new_step được chọn ngẫu nhiên từ all_steps, chứa tất cả các bước hướng dẫn có sẵn.

- are_steps_similar kiểm tra xem new_step có quá giống với bất kỳ bước nào đã có trong individual (dựa trên threshold). Nếu new_step không quá tương tự với các bước đã có, nó được thêm vào individual.

- Hàm remove_duplicates đảm bảo không có bước nào bị trùng lặp hoàn toàn trong individual.

4.2.2.2 Các hàm của thuật toán di truyền (GA Directions)

a. Hàm đánh giá fitness

Hàm evaluate_direction_individual đánh giá điểm số fitness cho một cá thể individual đại diện cho một chuỗi các bước hướng dẫn nấu ăn. Điểm số này phản ánh mức độ tối ưu của chuỗi hướng dẫn dựa trên tiêu chí như tính duy nhất của các bước, mức độ bao phủ nguyên liệu, và tính liên tục của các bước.

```
def evaluate_direction_individual(individual, relevant_ingredients, synchronize_df, process_data):
        """Evaluate the fitness of an individual direction."""
        unique_individual = remove_duplicates(individual)
        filtered_steps = filter_directions_by_ingredients_nlp(unique_individual, relevant_ingredients)
        sorted_steps = categorize_steps_with_nlp(filtered_steps)
        score = 0
        unique_steps = set(sorted_steps)  # Sử dụng tập hợp để kiểm tra trùng lặp
        # Nếu số lượng các bước duy nhất nhỏ hơn số bước tổng cộng, phạt điểm
        if len(unique_steps) < len(sorted_steps):
          score -= (len(sorted_steps) - len(unique_steps)) * 3  # Phạt cho mỗi bước trùng lặp

        covered_ingredients = set()
        for step in sorted_steps:
          for ingredient in relevant_ingredients:
            if ingredient in step:
              covered_ingredients.add(ingredient)
              score +=4

        if len(covered_ingredients) < len(relevant_ingredients) / 2:
          score -= 3

        previous_step = None
        for step in unique_individual:
          if previous_step and are_steps_similar(previous_step, step):
            score -= 2
          previous_step = step

        return score,
```

Hình 15: Hàm đánh giá fitness cho thuật toán tối ưu hướng dẫn nấu ăn

Gọi hàm remove_duplicates(individual): Xóa các bước trùng lặp trong chuỗi hướng dẫn để tạo danh sách unique_individual.

Gọi hàm filter_directions_by_ingredients_nlp để tạo danh sách filtered_steps: Lọc các bước có chứa nguyên liệu trong relevant_ingredients (nguyên liệu tối ưu hóa đã được tìm ra sau khi thuật toán GA Ingredients kết thúc).

Gọi hàm categorize_steps_with_nlp để tạo danh sách sorted_steps: Đây là hàm phân loại và sắp xếp các bước theo thứ tự hợp lý như chuẩn bị, nấu chính, và hoàn thiện

Phạt cho các bước trùng lặp:

- Tạo tập hợp unique_steps từ sorted_steps để loại bỏ trùng lặp. Nếu có sự khác biệt giữa độ dài của sorted_steps và unique_steps, nghĩa là có các bước trùng lặp.

- Điểm bị trừ dựa trên số lượng bước trùng lặp, với mức phạt 3 điểm cho mỗi bước lặp lại.

Bao phủ nguyên liệu:

- Duyệt qua các bước sorted_steps và kiểm tra xem mỗi bước có đề cập đến các nguyên liệu trong relevant_ingredients.
- Mỗi nguyên liệu được bao phủ tăng điểm 4 điểm.
- Nếu dưới một nửa nguyên liệu trong relevant_ingredients được đề cập, sẽ có một mức phạt 3 điểm.

Phạt cho các bước tương tự liên tiếp: Kiểm tra xem hai bước liên tiếp có tương tự nhau không, thông qua hàm are_steps_similar. Nếu đúng, phạt 2 điểm cho mỗi cặp bước tương tự.

Trả về điểm số fitness: Trả về điểm số tổng score của cá thể individual, với các điểm cộng hoặc phạt dựa trên mức độ tối ưu của chuỗi hướng dẫn.

b. Hàm lai ghép hai điểm (two-point crossover)

Hàm two_point_crossover thực hiện phép lai ghép hai điểm (two-point crossover) trên hai cá thể ind1 và ind2 để tạo ra hai cá thể con mới, giữ lại sự đa dạng trong quá trình tiến hóa.

```python
def two_point_crossover(ind1, ind2):
    """Thực hiện two-point crossover."""
    if len(ind1) < 3 or len(ind2) < 3:
        return ind1, ind2  # Nếu cá thể quá ngắn, không thực hiện crossover

    # Chọn hai điểm crossover ngẫu nhiên
    point1 = random.randint(1, min(len(ind1), len(ind2)) - 2)
    point2 = random.randint(point1 + 1, min(len(ind1), len(ind2)) - 1)

    # Tạo ra hai con bằng cách trao đổi phần giữa hai điểm crossover
    child1 = ind1[:point1] + ind2[point1:point2] + ind1[point2:]
    child2 = ind2[:point1] + ind1[point1:point2] + ind2[point2:]

    return creator.DirectionIndividual(child1), creator.DirectionIndividual(child2)
```

Hình 16: Hàm lai ghép two-point crossover cho GA Directions

Đầu tiên kiểm tra xem ind1 và ind2 có độ dài đủ dài không (ít nhất 3 phần tử). Nếu không, hàm sẽ trả về ind1 và ind2 mà không thực hiện lai ghép.

Chọn hai điểm lai ghép ngẫu nhiên:

- point1: Một vị trí ngẫu nhiên được chọn trong phạm vi [1, len(ind) - 2] để tránh việc chọn điểm đầu hoặc cuối, giúp bảo toàn cấu trúc của cá thể.

- point2: Điểm thứ hai được chọn ngẫu nhiên trong phạm vi [point1 + 1, len(ind) - 1], đảm bảo point2 luôn nằm sau point1.

Tạo ra cá thể con bằng cách trao đổi phần tử giữa các điểm lai ghép:

- child1: Được tạo từ phần đầu của ind1 (từ đầu đến point1), phần giữa của ind2 (từ point1 đến point2), và phần cuối của ind1 (từ point2 đến hết).
- child2: Tương tự, nhưng phần đầu và cuối đến từ ind2 và phần giữa từ ind1.

Cuối cùng trả child1 và child2 dưới dạng DirectionIndividual để đảm bảo cá thể có cùng kiểu dữ liệu như trong thuật toán di truyền.

Tóm lại, hàm two_point_crossover tạo ra hai cá thể con từ hai cá thể bố mẹ bằng cách hoán đổi các phần tử giữa hai điểm ngẫu nhiên. Phương pháp này duy trì đa dạng của quần thể và giúp tăng khả năng tìm kiếm giải pháp tối ưu hơn trong thuật toán di truyền.

c. Hàm đột biến

```
toolbox_directions.register("mutate", tools.mutShuffleIndexes, indpb=0.2)
```

Hình 17: Hàm đột biến mutShuffleIndexes cho GA Directions

Đăng ký hàm đột biến với tên "mutate" trong toolbox_directions, một công cụ chứa các hàm được sử dụng trong thuật toán di truyền.

tools.mutShuffleIndexes: Đây là hàm đột biến mutShuffleIndexes của thư viện DEAP, dùng để thực hiện phép hoán đổi ngẫu nhiên thứ tự của các gene trong cá thể.

indpb=0.2: indpb (individual probability) là xác suất để mỗi gene (phần tử) trong cá thể bị hoán đổi vị trí. Với indpb=0.2, mỗi gene có 20% khả năng tham gia vào hoán đổi.

Cách hoạt động: Hàm mutShuffleIndexes sẽ chọn ngẫu nhiên một số gene trong cá thể và hoán đổi vị trí của chúng, giúp tăng tính đa dạng trong quần thể bằng cách tạo ra các tổ hợp gene mới từ các cá thể hiện có.

d. Hàm chọn lọc

```
toolbox_directions.register("select", tools.selRoulette)
```

Hình 18: Hàm chọn lọc selRoulette cho GA Directions

Đăng ký một hàm lựa chọn (selection function) có tên "select" trong toolbox_directions. Điều này giúp gọi phương thức select trực tiếp từ toolbox_directions trong quá trình tiến hóa.

tools.selRoulette: selRoulette là một hàm lựa chọn từ thư viện DEAP thực hiện chọn lọc bánh xe Roulette:

- Mỗi cá thể sẽ có xác suất được chọn tương ứng với điểm fitness của nó.
- Cá thể có điểm fitness cao hơn sẽ có khả năng được chọn lớn hơn, nhưng cá thể có điểm fitness thấp hơn vẫn có cơ hội nhỏ được chọn, giúp duy trì một phần đa dạng trong quần thể.

Cách hoạt động của phương pháp Roulette: Trong phương pháp bánh xe Roulette, các cá thể được chọn ngẫu nhiên nhưng dựa trên trọng số tỷ lệ thuận với điểm số fitness. Quá trình này tương tự như xoay bánh xe có phân vùng, với mỗi vùng đại diện cho một cá thể và kích thước tỷ lệ với điểm fitness của nó.

4.2.2.3 Thuật toán GA cho hướng dẫn nấu ăn

Hàm genetic_algorithm_directions được thiết kế để tối ưu hóa một chuỗi các bước hướng dẫn nấu ăn, đảm bảo rằng chúng phù hợp với các nguyên liệu đã chọn và tuân theo yêu cầu về hương vị của người dùng. Đây là cách áp dụng thuật toán di truyền (Genetic Algorithm - GA) để cải thiện từng thế hệ của chuỗi hướng dẫn, giúp đạt được phiên bản tối ưu nhất.

a. Khởi tạo các công cụ cho GA

```
def genetic_algorithm_directions(all_relevant_steps, relevant_ingredients, ngen=10, pop_size=30, cxpb=0.6, mutpb=0.2, threshold=0.85):
    """Chạy thuật toán GA để tối ưu hóa hướng dẫn nấu ăn."""
    toolbox_directions = base.Toolbox()
    toolbox_directions.register("individual", tools.initIterate, creator.DirectionIndividual,
                                lambda: create_direction_individual(all_relevant_steps, threshold=0.85))
    toolbox_directions.register("population", tools.initRepeat, list, toolbox_directions.individual)
    toolbox_directions.register("mate", two_point_crossover)
    toolbox_directions.register("mutate", tools.mutShuffleIndexes, indpb=0.2)
    toolbox_directions.register("select", tools.selRoulette)
    toolbox_directions.register("evaluate", lambda ind: evaluate_direction_individual(ind, relevant_ingredients, synchronize_df, process_data))
```

Hình 19: Khởi tạo toolbox đăng ký hàm cho GA Directions

individual: Định nghĩa cách tạo một cá thể (chuỗi các bước) từ all_relevant_steps dựa trên ngưỡng threshold. Mỗi cá thể là một chuỗi các bước hướng dẫn được lấy ngẫu nhiên và lọc để tránh trùng lặp.

population: Tạo quần thể gồm pop_size cá thể (ở đây pop_size sẽ là 30 để đảm bảo thuật toán hoạt động nhanh chóng và tối ưu đối với hướng dẫn nấu ăn)

mate: Hàm two_point_crossover thực hiện hoán đổi hai đoạn trong hai cá thể khác nhau. Điều này cho phép các bước trong chuỗi hướng dẫn được tái kết hợp, tạo ra những chuỗi mới từ các phần đã có.

mutate: Hàm mutShuffleIndexes để hoán đổi vị trí ngẫu nhiên của các bước với xác suất 0.2, làm tăng tính đa dạng trong quần thể.

select: Phương pháp chọn lọc bánh xe Roulette (selRoulette) được sử dụng. Các cá thể có điểm số Fitness cao hơn có xác suất được chọn lớn hơn, nhưng tất cả các cá thể đều có cơ hội, giúp duy trì đa dạng trong quần thể.

evaluate: Hàm tính fitness (evaluate_direction_individual) đánh giá mức độ tối ưu của từng cá thể dựa trên sự phù hợp của chuỗi hướng dẫn với các nguyên liệu liên quan và yêu cầu hương vị.

b. Khởi tạo quần thể ban đầu và điều chỉnh Fitness:

```
population = toolbox_directions.population(n=pop_size)
invalid_ind = [ind for ind in population if not ind.fitness.valid]
update_population_fitness_with_sharing_directions(invalid_ind, relevant_ingredients, synchronize_df, process_data, sharing_radius=3.0)
adjust_fitness(invalid_ind)
```

Hình 20: Khởi tạo quần thể và điều chỉnh fitness cho GA Directions

Khởi tạo quần thể ban đầu với pop_size cá thể.

update_population_fitness_with_sharing_directions: Sử dụng fitness sharing để chia sẻ điểm số giữa các cá thể tương tự, giúp giữ đa dạng và tránh hiện tượng các cá thể giống nhau chiếm ưu thế trong quần thể.

adjust_fitness: Điều chỉnh fitness để không có giá trị âm, đảm bảo rằng phương pháp chọn lọc (như selRoulette cần fitness dương) hoạt động trơn tru.

c. Lặp qua các thế hệ:

```
# Bắt đầu các thế hệ GA
for gen in range(ngen):
  mutpb = adaptive_mutation_rate(population, gen, ngen)

  # Chọn các cá thể con từ quần thể hiện tại bằng Roulette Selection
  offspring = toolbox_directions.select(population, len(population) - len(hof))
  offspring = list(map(toolbox_directions.clone, offspring))
```

Hình 21: Vòng lặp qua các thế hệ (GA Directions)

Vòng lặp qua ngen thế hệ: Tạo ra các thế hệ con từ quần thể hiện tại để dần tìm ra giải pháp tối ưu.

adaptive_mutation_rate: Tỉ lệ đột biến mutpb được điều chỉnh dựa trên mức độ hội tụ của quần thể. Khi quần thể thiếu đa dạng, tỷ lệ đột biến tăng lên để tạo ra nhiều cá thể mới hơn.

Chọn lọc: Sử dụng chọn lọc bánh xe Roulette để chọn cá thể cho thế hệ con, tạo ra một thế hệ offspring có kích thước nhỏ hơn một chút so với quần thể ban đầu để duy trì một số cá thể tốt nhất.

d. Thực hiện lai ghép và đột biến

```
# Lai ghép (crossover) và đột biến (mutation)
for child1, child2 in zip(offspring[::2], offspring[1::2]):
  if random.random() < cxpb:
    toolbox_directions.mate(child1, child2)
    del child1.fitness.values
    del child2.fitness.values
for mutant in offspring:
  if random.random() < mutpb:
    toolbox_directions.mutate(mutant)
    del mutant.fitness.values
```

Hình 22: Vòng lặp lai ghép và đột biến của GA Directions

Lai ghép (Crossover): Cặp các cá thể con (offspring) và thực hiện lai ghép với xác suất cxpb. Sau khi lai ghép, fitness của các cá thể con bị xóa để chúng được đánh giá lại.

Đột biến (Mutation): Các cá thể có xác suất mutpb sẽ bị đột biến (hoán đổi thứ tự các bước trong hướng dẫn). Việc đột biến làm tăng sự khác biệt trong quần thể, giúp tránh hiện tượng tối ưu cục bộ.

e. Đánh giá và cập nhật Fitness sau đột biến và lai ghép:

```
# Đánh giá lại các cá thể không hợp lệ sau đột biến và lai ghép
invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
if gen % 10 == 0 and invalid_ind:
  update_population_fitness_with_sharing_directions(invalid_ind, relevant_ingredients, synchronize_df, process_data, sharing_radius=3.0)
  adjust_fitness(invalid_ind)  # Điều chỉnh fitness nếu cần thiết cho selRoulette

# Kết hợp offspring với các cá thể elitist
population[:] = tools.selBest(population + offspring, pop_size)

# Cập nhật Elitism
hof.update(population)
```

Hình 23: Đánh giá và cập nhật fitness sau đột biến và lai ghép (GA Directions)

Mỗi 10 thế hệ, hàm update_population_fitness_with_sharing_directions được gọi lại để đảm bảo rằng các cá thể giống nhau không chiếm ưu thế và đa dạng được duy trì trong quần thể.

adjust_fitness được gọi để tránh giá trị fitness âm.

Kết hợp population với offspring và chọn ra các cá thể tốt nhất để đảm bảo quần thể giữ nguyên kích thước pop_size với các cá thể chất lượng cao nhất.

hof.update(population): Cập nhật Hall of Fame (HOF) với các cá thể có fitness cao nhất, giúp lưu trữ các giải pháp tốt nhất qua các thế hệ.

f. Trả về kết quả tối ưu cho GA Directions

```
        best_directions = tools.selBest(population, 1)[0]
        best_directions = remove_duplicates(best_directions)
        return best_directions
```

Hình 24: Trả về kết quả hướng dẫn nấu ăn tối ưu nhất.

Sau khi tất cả các thế hệ hoàn tất, tools.selBest chọn ra cá thể có fitness cao nhất từ quần thể cuối cùng, đại diện cho chuỗi hướng dẫn tối ưu.

remove_duplicates: Xóa bất kỳ bước nào bị trùng lặp trong chuỗi hướng dẫn để đảm bảo tính hợp lý và mạch lạc.

4.3. Xử lý dữ liệu từ các công thức có sẵn

4.3.1. Thu thập và trích xuất dữ liệu từ website nutritionfacts.org

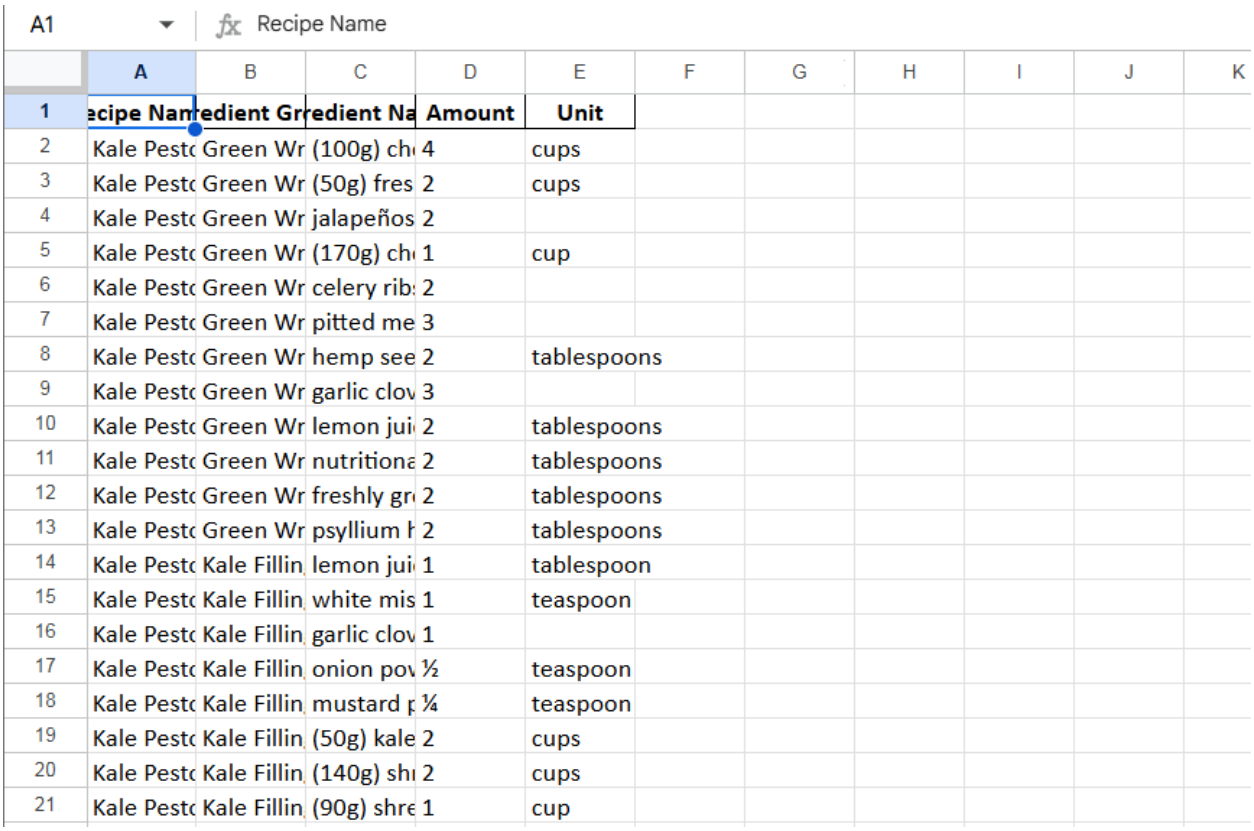
Việc thu thập dữ liệu nguyên liệu nấu ăn và các hướng dẫn nấu trên trang web nutrionfacts.org được thực hiện bằng sự hỗ trợ của hai thư viện requests và BeautifulSoup.

```
import requests
from bs4 import BeautifulSoup
import pandas as pd

# Base URL của trang công thức
base_url = "https://nutritionfacts.org/recipes/"
response = requests.get(base_url)
soup = BeautifulSoup(response.content, 'html.parser')
```

Hình 25: Khởi tạo các biến trích xuất dữ liệu từ website nutritionfacts.org

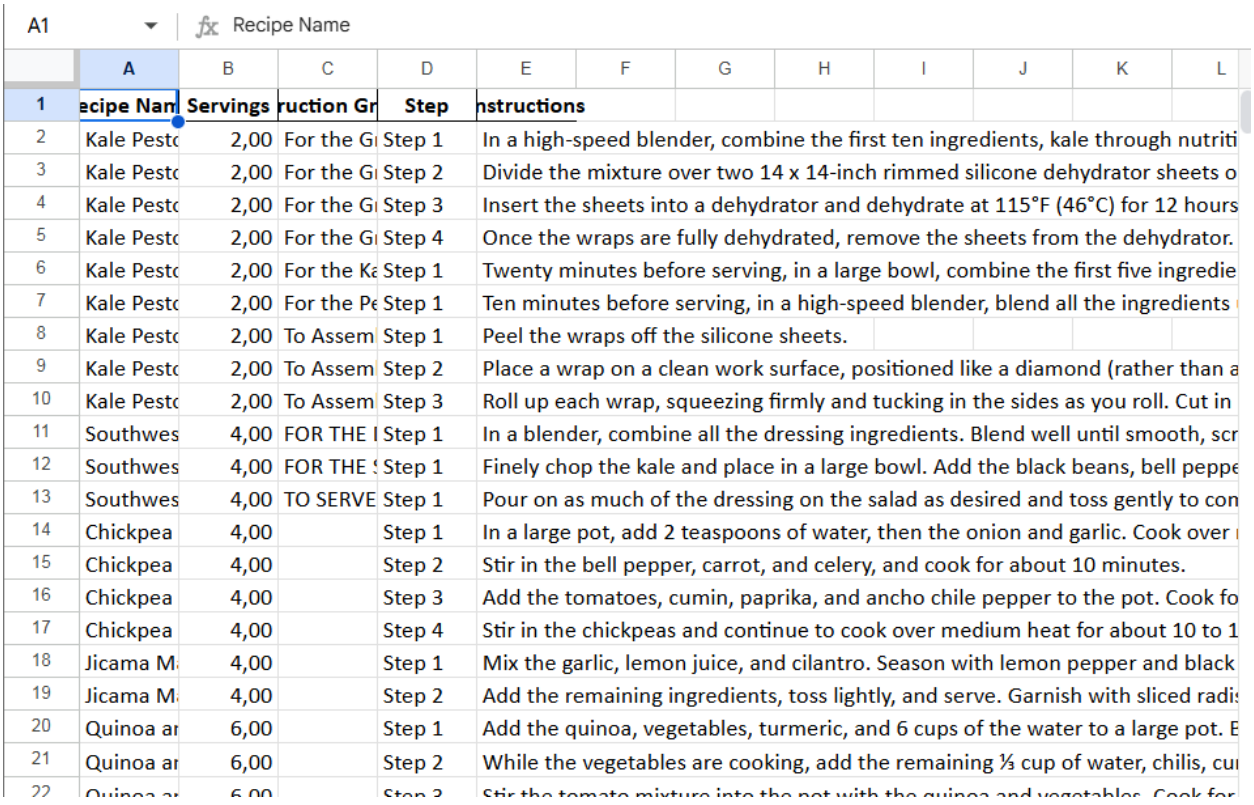
requests là một thư viện giúp gửi yêu cầu HTTP dễ dàng đến các trang web và nhận phản hồi từ chúng. BeautifulSoup là thư viện phân thích cú pháp HTML/XML, giúp xử lý và trích xuất thông tin từ nội dung HTML của trang web. Thư viện này cung cấp các phương thức để tìm kiếm và truy cập các thành phần (như thẻ div, a, span, v.v.) trong trang web một cách dễ dàng.

Dưới đây là hai bộ dữ liệu thu thập được sau khi chạy mã trích xuất – một bộ là về nguyên liệu nấu, một bộ là về các hướng dẫn nấu ăn.



	A	B	C	D	E
1	ecipe Nan	edient Gr	edient Na	Amount	Unit
2	Kale Pesto	Green Wr	(100g) ch	4	cups
3	Kale Pesto	Green Wr	(50g) fres	2	cups
4	Kale Pesto	Green Wr	jalapeños	2	
5	Kale Pesto	Green Wr	(170g) ch	1	cup
6	Kale Pesto	Green Wr	celery rib	2	
7	Kale Pesto	Green Wr	pitted me	3	
8	Kale Pesto	Green Wr	hemp see	2	tablespoons
9	Kale Pesto	Green Wr	garlic clov	3	
10	Kale Pesto	Green Wr	lemon jui	2	tablespoons
11	Kale Pesto	Green Wr	nutritiona	2	tablespoons
12	Kale Pesto	Green Wr	freshly gr	2	tablespoons
13	Kale Pesto	Green Wr	psyllium h	2	tablespoons
14	Kale Pesto	Kale Fillin	lemon jui	1	tablespoon
15	Kale Pesto	Kale Fillin	white mis	1	teaspoon
16	Kale Pesto	Kale Fillin	garlic clov	1	
17	Kale Pesto	Kale Fillin	onion pov	½	teaspoon
18	Kale Pesto	Kale Fillin	mustard p	¼	teaspoon
19	Kale Pesto	Kale Fillin	(50g) kale	2	cups
20	Kale Pesto	Kale Fillin	(140g) shi	2	cups
21	Kale Pesto	Kale Fillin	(90g) shre	1	cup

Hình 26: Bộ dữ liệu nguyên liệu gốc sau khi trích xuất từ website



	A	B	C	D	E
1	ecipe Nan	Servings	ruction Gr	Step	nstructions
2	Kale Pesto	2,00	For the G	Step 1	In a high-speed blender, combine the first ten ingredients, kale through nutriti
3	Kale Pesto	2,00	For the G	Step 2	Divide the mixture over two 14 x 14-inch rimmed silicone dehydrator sheets o
4	Kale Pesto	2,00	For the G	Step 3	Insert the sheets into a dehydrator and dehydrate at 115°F (46°C) for 12 hours
5	Kale Pesto	2,00	For the G	Step 4	Once the wraps are fully dehydrated, remove the sheets from the dehydrator.
6	Kale Pesto	2,00	For the Ka	Step 1	Twenty minutes before serving, in a large bowl, combine the first five ingredie
7	Kale Pesto	2,00	For the Pe	Step 1	Ten minutes before serving, in a high-speed blender, blend all the ingredients
8	Kale Pesto	2,00	To Assem	Step 1	Peel the wraps off the silicone sheets.
9	Kale Pesto	2,00	To Assem	Step 2	Place a wrap on a clean work surface, positioned like a diamond (rather than a
10	Kale Pesto	2,00	To Assem	Step 3	Roll up each wrap, squeezing firmly and tucking in the sides as you roll. Cut in
11	Southwes	4,00	FOR THE I	Step 1	In a blender, combine all the dressing ingredients. Blend well until smooth, scr
12	Southwes	4,00	FOR THE S	Step 1	Finely chop the kale and place in a large bowl. Add the black beans, bell peppe
13	Southwes	4,00	TO SERVE	Step 1	Pour on as much of the dressing on the salad as desired and toss gently to con
14	Chickpea	4,00		Step 1	In a large pot, add 2 teaspoons of water, then the onion and garlic. Cook over
15	Chickpea	4,00		Step 2	Stir in the bell pepper, carrot, and celery, and cook for about 10 minutes.
16	Chickpea	4,00		Step 3	Add the tomatoes, cumin, paprika, and ancho chile pepper to the pot. Cook fo
17	Chickpea	4,00		Step 4	Stir in the chickpeas and continue to cook over medium heat for about 10 to 1
18	Jicama M	4,00		Step 1	Mix the garlic, lemon juice, and cilantro. Season with lemon pepper and black
19	Jicama M	4,00		Step 2	Add the remaining ingredients, toss lightly, and serve. Garnish with sliced radis
20	Quinoa ar	6,00		Step 1	Add the quinoa, vegetables, turmeric, and 6 cups of the water to a large pot. E
21	Quinoa ar	6,00		Step 2	While the vegetables are cooking, add the remaining ⅓ cup of water, chilis, cu
22	Quinoa ar	6,00		Step 3	Stir the tomato mixture into the pot with the quinoa and vegetables. Cook for

Hình 27: Bộ dữ liệu hướng dẫn nấu ăn gốc sau khi trích xuất từ website.

4.3.2. Đồng bộ dữ liệu từ hai bộ dữ liệu

Sau khi thu thập được, nhận thấy việc dữ liệu khá sạch nên không cần xử lý quá nhiều. Nhưng việc cần làm là kết hợp các cột "Step" và "Instructions" của bộ dữ liệu hướng dẫn nấu với cột "Ingredient Name" của bộ dữ liệu nguyên liệu. Tức là làm sao cho thuật toán nhận thấy được rằng các ingredients nào sẽ thuộc về steps và instructions nào. Việc này cần thiết cho thuật toán di truyền của hướng dẫn nấu (GA Directions) có thể hoạt động được. Dưới đây, chính là cách thực hiện với khá nhiều bước phức tạp kết hợp cùng với các mô hình NER để nhận diện.

Bước 1: Gộp nhóm nguyên liệu và hướng dẫn vào cùng một "Recipe Name"

```
# Adjusting to ensure all ingredients from the Ingredient Group are included in the corresponding Instruction Group
import json
# Initialize an empty dictionary to store structured data by recipe and group
final_recipes_data = {}

# Merge Ingredient Group and Instruction Group under a common key and merge their data
for _, row in df1.iterrows():
    recipe_name = row['Recipe Name']
    group_name = row['Ingredient Group'] if pd.notna(row['Ingredient Group']) else "No Group"

    # Initialize structure if recipe not already in the dictionary
    if recipe_name not in final_recipes_data:
        final_recipes_data[recipe_name] = {}

    if group_name not in final_recipes_data[recipe_name]:
        final_recipes_data[recipe_name][group_name] = {'ingredients': [], 'instructions': []}

    # Append ingredient to the group's ingredient list
    ingredient_info = f"{row['Amount']} {row['Unit']} {row['Ingredient Name']}".strip()
    final_recipes_data[recipe_name][group_name]['ingredients'].append(ingredient_info)
```

Hình 28: Gộp nhóm nguyên liệu vào cùng một "Recipe Name"

- final_recipes_data: Một dictionary rỗng được khởi tạo để lưu trữ dữ liệu có cấu trúc của công thức, phân nhóm theo tên công thức và nhóm nguyên liệu.

- Vòng lặp for _, row in df1.iterrows(): Duyệt qua từng hàng của bảng df1, nơi chứa thông tin về nguyên liệu. Mỗi hàng (row) đại diện cho một nguyên liệu của một công thức nấu ăn cụ thể.

- recipe_name: Lấy tên của công thức nấu ăn từ cột Recipe Name của row.
- group_name: Lấy tên nhóm nguyên liệu (Ingredient Group) từ hàng hiện tại (row). Nếu không có tên nhóm (tức là giá trị bị thiếu hoặc NaN), nó sẽ gán tên nhóm là "No Group" để xác định các nguyên liệu không thuộc nhóm nào.
- Sau đó, kiểm tra và khởi tạo công thức, nếu recipe_name chưa tồn tại trong final_recipes_data, tạo một dictionary rỗng cho công thức này. Điều này đảm bảo rằng mỗi công thức sẽ có một mục riêng trong final_recipes_data.

- Tiếp theo, kiểm tra và khởi tạo nhóm nguyên liệu, nếu group_name chưa tồn tại trong recipe_name hiện tại, thêm một dictionary mới cho nhóm này với các khóa ingredients và instructions. ingredients sẽ là danh sách chứa nguyên liệu của nhóm này, còn instructions sẽ lưu các bước hướng dẫn tương ứng (sẽ được thêm vào sau).
- ingredient_info: Biến này tạo một chuỗi chứa thông tin về nguyên liệu, gồm số lượng (Amount), đơn vị (Unit), và tên nguyên liệu (Ingredient Name). Chuỗi này được định dạng sao cho gọn gàng, ví dụ: "1 cup flour".
- Thêm nguyên liệu vào nhóm: ingredient_info được thêm vào danh sách ingredients của group_name trong recipe_name hiện tại.

- Phần này sẽ tạo ra một cấu trúc dictionary (final_recipes_data) có dạng như sau:

```
{
    "Recipe Name A": {
        "Group 1": {
            "ingredients": ["1 cup flour", "2 tbsp sugar", ...],
            "instructions": []
        },
        "Group 2": {
            "ingredients": ["1 tsp salt", ...],
            "instructions": []
        }
    },
    "Recipe Name B": {
        "No Group": {
            "ingredients": ["1 egg", "200ml milk", ...],
            "instructions": []
        }
    },
    ...
}
```

Hình 29: Dữ liệu dưới dạng dictionary sau khi gộp nhóm nguyên liệu

- Mỗi công thức nấu ăn (Recipe Name A, Recipe Name B, ...) chứa các nhóm nguyên liệu (như Group 1, No Group), và trong mỗi nhóm có danh sách ingredients với các nguyên liệu tương ứng.
- Kết quả là chúng ta có một cấu trúc dữ liệu được tổ chức rõ ràng, dễ truy cập và xử lý cho các bước sau.

Sau khi đã gộp được nguyên liệu, tiến hành gộp thông tin hướng dẫn (Instructions) từ df2 vào final_recipes_data. Dùng vòng lặp for _, row in df2.iterrows() tương tự.

```python
# Process instructions and merge them into the corresponding group
for _, row in df2.iterrows():
    recipe_name = row['Recipe Name']
    group_name = row['Instruction Group'] if pd.notna(row['Instruction Group']) else "No Group"

    # Initialize structure if recipe not already in the dictionary
    if recipe_name not in final_recipes_data:
        final_recipes_data[recipe_name] = {}

    #if group_name.startswith("For the "):
        #group_name = group_name.replace("For the ", "")

    # Initialize group if not already in the recipe
    if group_name not in final_recipes_data[recipe_name]:
        final_recipes_data[recipe_name][group_name] = {'ingredients': [], 'instructions': []}

    # Append instruction to the group's instruction list
    step_info = f"{row['Step']}: {row['Instructions']}"
    final_recipes_data[recipe_name][group_name]['instructions'].append(step_info)
```

Hình 30: Gộp nhóm hướng dẫn vào cùng một "Recipe Nam"

- Phần này sẽ thu được kết quả tương tự như dưới đây:

```
{
    "Recipe Name A": {
        "Group 1": {
            "ingredients": ["1 cup flour", "2 tbsp sugar", ...],
            "instructions": [
                "1: Mix flour and sugar",
                "2: Add water gradually",
                ...
            ]
        },
        "Group 2": {
            "ingredients": ["1 tsp salt", ...],
            "instructions": [
                "1: Add salt to the mixture",
                ...
            ]
        }
    },
    "Recipe Name B": {
        "No Group": {
            "ingredients": ["1 egg", "200ml milk", ...],
            "instructions": [
                "1: Beat egg and milk together",
                ...
            ]
        }
    },
    ...
}
```

Hình 31: Dữ liệu dưới dạng dictionary sau khi gộp nhóm hướng dẫn

- Mỗi nhóm nguyên liệu trong từng công thức (Recipe Name A, Recipe Name B, ...) giờ đã có danh sách các bước hướng dẫn (instructions) phù hợp với nguyên liệu của nhóm đó.
- Kết quả này giúp tổ chức dữ liệu của công thức nấu ăn theo từng nhóm nguyên liệu và bước hướng dẫn, hỗ trợ quá trình phân tích hoặc xử lý dữ liệu sau này, như tự động phân loại nguyên liệu hoặc xây dựng quy trình nấu ăn hoàn chỉnh.

Bước 2: Sử dụng mô hình NER để nhận diện nguyên liệu

Sau khi gộp nhóm được dữ liệu, tiến hành sử dụng mô hình Named Entity Recognition (NER) để trích xuất tên nguyên liệu từ các bước hướng dẫn trong công thức. Cụ thể, nó áp dụng mô hình distilbert-for-food-extraction để xác định các từ hoặc cụm từ đại diện cho nguyên liệu trong văn bản, giúp phát hiện và ghi nhận các nguyên liệu được nhắc đến trong từng câu hướng dẫn. Kết quả sau khi tự động nhận diện sẽ là một danh sách các nguyên liệu xuất hiện trong từng câu hướng dẫn, giúp liên kết các nguyên liệu với các steps và instructions cụ thể.

```
from transformers import pipeline
from collections import defaultdict
from fuzzywuzzy import fuzz

# Khởi tạo pipeline cho mô hình NER
nlp = pipeline("token-classification", model="chambliss/distilbert-for-food-extraction", aggregation_strategy="simple")
```

Hình 32: Khởi tạo mô hình NER sử dụng pipeline

- Khởi tạo mô hình NER: Sử dụng pipeline từ thư viện transformers để khởi tạo mô hình NER. Mô hình distilbert-for-food-extraction được chọn do khả năng nhận diện nguyên liệu trong văn bản mô tả thức ăn.

- aggregation_strategy="simple": Chiến lược này giúp hợp nhất các từ liên quan, tạo ra các cụm từ hoàn chỉnh cho nguyên liệu, tránh tách rời các từ trong tên nguyên liệu dài.

```python
# Hàm nhận diện nguyên liệu từ câu hướng dẫn bằng mô hình NER
def extract_ingredients_from_instructions(step_text):
    entities = nlp(step_text)
    ingredients = []
    current_ingredient = ""

    for entity in entities:
        if entity['entity_group'] in ['LABEL_0', 'LABEL_1']:
            current_ingredient += entity['word'].replace("##", "")
        else:
            if current_ingredient:
                ingredients.append(current_ingredient.strip())
                current_ingredient = ""

    if current_ingredient:
        ingredients.append(current_ingredient.strip())

    return ingredients
```

Hình 33: Nhận diện và trích xuất nguyên liệu từ câu hướng dẫn sử dụng NER

- Hàm extract_ingredients_from_instructions: Hàm này trích xuất các nguyên liệu từ câu hướng dẫn sử dụng mô hình NER.

- entities = nlp(step_text): Áp dụng mô hình nlp lên câu hướng dẫn (step_text) để xác định các từ hoặc cụm từ là nguyên liệu. Kết quả entities sẽ là một danh sách chứa các đối tượng đã được mô hình NER nhận diện.

- ingredients là danh sách lưu các nguyên liệu đã nhận diện từ câu hướng dẫn.

- Ghép các từ thành nguyên liệu hoàn chỉnh sử dụng vòng lặp for entity in entities:

- LABEL_0 và LABEL_1: Đây là các nhãn của mô hình NER, dùng để phân biệt các từ thuộc về nguyên liệu.
- Nếu entity_group thuộc LABEL_0 hoặc LABEL_1, current_ingredient sẽ tiếp tục cộng thêm các từ mới, ghép thành cụm từ đại diện cho tên nguyên liệu.
- Khi mô hình gặp một nhãn khác, hàm sẽ coi cụm từ hiện tại (current_ingredient) là một nguyên liệu hoàn chỉnh, thêm nó vào ingredients.

- Thêm nguyên liệu cuối cùng: Nếu có một cụm nguyên liệu còn sót lại trong current_ingredient sau khi kết thúc vòng lặp, nó sẽ được thêm vào ingredients.

- Trả về danh sách ingredients chứa tất cả các nguyên liệu đã được nhận diện từ câu hướng dẫn.

- Ví dụ minh họa: Giả sử, step_text = "Add 2 cups of flour and 1 teaspoon of salt to the bowl.", thì extract_ingredients_from_instructions(step_text) có thể trả về: ["flour", "salt"].

Bước 3: So khớp mờ với nguyên liệu và lọc thông tin

Sau khi đã nhận diện nguyên liệu, tiến hành sử dụng so khớp mờ với nguyên liệu và lọc thông tin.

```
# Hàm cải tiến so khớp fuzzy
def fuzzy_match_ingredient(detected_ingredient, ingredient_list):
    best_match, best_score = None, 0
    for ingredient in ingredient_list:
        match_ratio = fuzz.token_set_ratio(detected_ingredient.lower(), ingredient.lower())
        if match_ratio > best_score and match_ratio >= 90:  # Ngưỡng 90
            best_match, best_score = ingredient, match_ratio
    return best_match
```

Hình 34: Hàm fuzzy_match_ingredient (so khớp mờ và lọc thông tin)

- fuzzy_match_ingredient: Hàm này sử dụng phương pháp so khớp mờ (fuzzy matching) để đối chiếu tên nguyên liệu được nhận diện từ bước hướng dẫn với danh sách nguyên liệu đầy đủ của công thức với độ chính xác cao (≥ 90%). Phương pháp này giúp đối phó với các tình huống khi tên nguyên liệu trong hướng dẫn có thể khác một chút so với tên trong danh sách gốc (ví "sugar" và "brown sugar").

- Tham số:

- detected_ingredient: Tên nguyên liệu đã nhận diện từ mô hình NER trong các bước hướng dẫn.
- ingredient_list: Danh sách nguyên liệu chuẩn của công thức, nơi mà detected_ingredient sẽ được đối chiếu.

- Khởi tạo các biến best_match và best_score:

- best_match: Giữ lại tên nguyên liệu có độ tương đồng cao nhất với detected_ingredient.
- best_score: Lưu điểm số tương đồng cao nhất giữa detected_ingredient và các nguyên liệu trong ingredient_list.

- Vòng lặp qua danh sách nguyên liệu (ingredient_list):

- fuzz.token_set_ratio: So sánh detected_ingredient với từng ingredient trong ingredient_list, tính điểm số khớp (match_ratio). Điểm số này dựa trên mức độ tương đồng giữa hai chuỗi: Nếu detected_ingredient = "brown sugar" và ingredient

= "sugar", match_ratio có thể ở mức cao (trên 90), vì "sugar" và "brown sugar" có các từ tương tự nhau.

- Ngưỡng 90: Hàm chỉ xem xét một nguyên liệu là tương đồng nếu match_ratio đạt từ 90 trở lên, để đảm bảo độ chính xác cao trong so khớp.
- Cập nhật best_match: Nếu match_ratio lớn hơn best_score, hàm sẽ cập nhật best_match và best_score bằng tên nguyên liệu hiện tại và điểm số của nó.
- Kết quả: Hàm trả về best_match - tên nguyên liệu trong danh sách ingredient_list có độ tương đồng cao nhất với detected_ingredient. Nếu không có kết quả phù hợp, hàm sẽ trả về None.

- Ví dụ minh họa: Giả sử ta có detected_ingredient = "brown sugar" và ingredient_list = ["sugar", "salt", "flour"]. Khi fuzzy_match_ingredient("brown sugar", ingredient_list) chạy:

- Hàm sẽ tính match_ratio giữa "brown sugar" và từng nguyên liệu trong ingredient_list.
- Nếu "sugar" có match_ratio trên 90, nó sẽ được chọn làm best_match và được trả về.

Bước 4: Phân loại nguyên liệu theo ngữ cảnh của các bước hướng dẫn

Tiếp theo, sử dụng thông tin từ các bước hướng dẫn để xác định nguyên liệu được sử dụng trong từng bước, dựa trên danh sách nguyên liệu trong công thức và các cụm từ đặc biệt trong câu hướng dẫn.

```
# Hàm phân loại nguyên liệu từ các bước hướng dẫn
def classify_ingredients_with_context(group_data, df1):
    instructions = group_data['instructions']
    ingredients = group_data['ingredients']
    used_ingredients = []
    classified_steps = {}

    for instruction in instructions:
        step_number = instruction.split(":")[0].strip()
        step_text = instruction.split(":")[1].strip()

        classified_steps[step_number] = {"step": step_text, "ingredients": []}

        # Sử dụng mô hình NER để nhận diện nguyên liệu
        detected_ingredients = extract_ingredients_from_instructions(step_text)

        # So khớp fuzzy với danh sách nguyên liệu trong group_data
        for detected_ingredient in detected_ingredients:
            match = fuzzy_match_ingredient(detected_ingredient, ingredients)
            if match:
                classified_steps[step_number]['ingredients'].append(match)
                used_ingredients.append(match)
```

Hình 35: Xác định nguyên liệu được sử dụng trong từng bước hướng dẫn

```
# Xử lý các câu hướng dẫn đặc biệt
for instruction in instructions:
    step_number = instruction.split(":")[0].strip()
    step_text = instruction.split(":")[1].strip()

    if "the first ten ingredients" in step_text.lower():
        first_ten = get_first_n_ingredients(ingredients, 10)
        classified_steps[step_number]['ingredients'].extend(first_ten)
        used_ingredients.extend(first_ten)

    if "the first five ingredients" in step_text.lower():
        first_five = get_first_n_ingredients(ingredients, 5)
        classified_steps[step_number]['ingredients'].extend(first_five)
        used_ingredients.extend(first_five)

    if any(phrase in step_text.lower() for phrase in [
        "all ingredients", "all the ingredients", "all of the ingredients", "all the dressing ingredients",
        "all the salad ingredients", "all the chimichurri sauce ingredients", "the tahini sauce ingredients",
        "mix ingredients", "all of the cheese sauce ingredients", "all of the queso ingredients"]):
        all_ingr = get_all_ingredients(ingredients)
        classified_steps[step_number]['ingredients'].extend(all_ingr)
        used_ingredients.extend(all_ingr)

    if any(phrase in step_text.lower() for phrase in [
        "remaining ingredients", "all the remaining tempeh ingredients",
        "all the remaining loaf ingredients", "the remaining gravy ingredients", "the rest of the ingredients"]):
        remaining = get_remaining_ingredients(used_ingredients, ingredients)
        classified_steps[step_number]['ingredients'].extend(remaining)
        used_ingredients.extend(remaining)
```

Hình 36: Xử lý các trường hợp đặc biệt trong việc xác định nguyên liệu

- Hàm chính trong bước này là classify_ingredients_with_context, có nhiệm vụ phân loại các nguyên liệu liên quan cho từng bước hướng dẫn trong một công thức, dựa vào ngữ cảnh và từ khóa đặc biệt.

- Tham số đầu vào:

- group_data: Dữ liệu của một nhóm nguyên liệu trong công thức, bao gồm cả instructions và ingredients.
- df1: Bảng dữ liệu nguyên liệu chuẩn từ đầu.

- Biến khởi tạo:

- used_ingredients: Danh sách lưu trữ các nguyên liệu đã được xác định là dùng trong các bước trước đó.
- classified_steps: Một dictionary dùng để lưu các bước hướng dẫn với các nguyên liệu tương ứng.

- Xác định nguyên liệu cho từng bước hướng dẫn:

- Tách số bước và nội dung bước hướng dẫn:
    - step_number: Lấy số thứ tự các steps từ chuỗi instruction.
    - step_text: Phần văn bản mô tả của bước.

- Nhận diện nguyên liệu từ mô tả bước:
    - detected_ingredients: Lấy ra các nguyên liệu trong câu hướng dẫn (step_text) bằng hàm NER từ Bước 3.
- So khớp nguyên liệu bằng fuzzy matching:
    - Dùng hàm fuzzy_match_ingredient trước đó để đối chiếu các nguyên liệu đã phát hiện (detected_ingredient) với danh sách ingredients của nhóm.
    - Nếu tìm thấy match phù hợp, thêm match vào ingredients của bước hiện tại và đánh dấu vào used_ingredients.

- Xử lý các từ khóa đặc biệt: Một số bước hướng dẫn chứa các cụm từ như "all ingredients" hoặc "remaining ingredients". Những câu này yêu cầu xử lý đặc biệt:

- "first ten ingredients": Lấy 10 nguyên liệu đầu tiên.
- "all ingredients": Lấy toàn bộ nguyên liệu.
- "remaining ingredients": Lấy tất cả nguyên liệu chưa dùng.

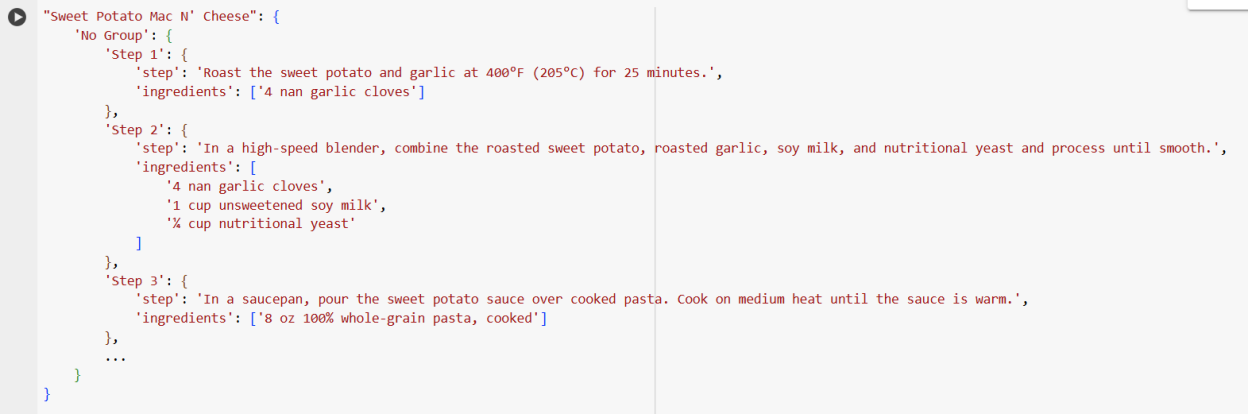
Bước 5: Lặp qua toàn bộ công thức và nhóm để phân loại

```
# Hàm lặp qua final_recipes_data và phân loại tự động
def classify_all_recipes(final_recipes_data, df1):
    classified_data = {}

    for recipe_name, groups in final_recipes_data.items():
        classified_data[recipe_name] = {}

        for group_name, group_data in groups.items():
            result = classify_ingredients_with_context(group_data, df1)
            classified_data[recipe_name][group_name] = result

    return classified_data

classified_data = classify_all_recipes(final_recipes_data, df1)
```

Hình 37: Hàm lặp toàn bộ công thức và xác định các nguyên liệu xuất hiện ở từng bước

- classify_all_recipes: Hàm này lặp qua tất cả công thức và nhóm để phân loại nguyên liệu cho từng bước hướng dẫn, lưu lại kết quả phân loại theo từng nhóm và từng công thức.

Kết quả của việc đồng bộ sẽ tương tự như thế này, khi từng steps và instructions sẽ tương ứng với nguyên liệu.



```
"Sweet Potato Mac N' Cheese": {
    'No Group': {
        'Step 1': {
            'step': 'Roast the sweet potato and garlic at 400°F (205°C) for 25 minutes.',
            'ingredients': ['4 nan garlic cloves']
        },
        'Step 2': {
            'step': 'In a high-speed blender, combine the roasted sweet potato, roasted garlic, soy milk, and nutritional yeast and process until smooth.',
            'ingredients': [
                '4 nan garlic cloves',
                '1 cup unsweetened soy milk',
                '¼ cup nutritional yeast'
            ]
        },
        'Step 3': {
            'step': 'In a saucepan, pour the sweet potato sauce over cooked pasta. Cook on medium heat until the sauce is warm.',
            'ingredients': ['8 oz 100% whole-grain pasta, cooked']
        },
        ...
    }
}
```

Hình 38: Kết quả dữ liệu dạng dictionary tổng kết của việc đồng bộ hóa

Bước 6: Gắn thông tin bước hướng dẫn (Step) và hướng dẫn cụ thể (Instructions) vào từng nguyên liệu trong bộ dữ liệu nguyên liệu df1 dựa vào dữ liệu đã được gộp nhóm và phân loại trước đó (Ở đây sẽ thêm trực tiếp vào cột dựa theo dữ liệu dạng dictionary đã được phân loại từ trước)

```
# Khởi tạo các cột cần thêm
df1['Step'] = None
df1['Instructions'] = None

# Duyệt qua từng dòng trong df1
for index, row in df1.iterrows():
    recipe_name = row['Recipe Name']
    group_name = row['Ingredient Group'] if pd.notna(row['Ingredient Group']) else "No Group"
    amount = row['Amount']
    unit = row['Unit']
    ingredient_name = row['Ingredient Name']

    # Tạo chuỗi để khớp với dữ liệu phân loại trong classified_data (cấu trúc: Amount Unit Ingredient Name)
    ingredient_string = f"{amount} {unit} {ingredient_name}".strip()

    # Kiểm tra nếu công thức và group có trong `classified_data`
    if recipe_name in classified_data and group_name in classified_data[recipe_name]:
        steps = classified_data[recipe_name][group_name]

        # Duyệt qua các bước trong group và tìm ingredient_string tương ứng
        for step_number, step_data in steps.items():

          for ingredient in step_data['ingredients']:
            ingredient_from_step = ingredient.strip()

            if ingredient_string == ingredient_from_step:
              df1.at[index, 'Step'] = step_number
              df1.at[index, 'Instructions'] = step_data['step']
              break
```

Hình 39: Thêm cột dữ liệu Steps và Instructions tương ứng với nguyên liệu

- Khởi tạo các cột "Step" và "Instructions" trong df1 để lưu thông tin về số bước và nội dung hướng dẫn.

- Duyệt qua từng hàng trong df1:

- Lấy tên công thức, nhóm nguyên liệu, số lượng, đơn vị, và tên nguyên liệu.
- Tạo chuỗi ingredient_string từ thông tin nguyên liệu (Amount Unit Ingredient Name).

- Kiểm tra công thức và nhóm trong classified_data (đây là dữ liệu dạng dictionary đã được gộp nhóm và phân loại ở các bước trên – các nguyên liệu đã được tương ứng với các steps và instructions trong dữ liệu này):

- Nếu có, duyệt qua các bước và so khớp từng nguyên liệu với ingredient_string.
- Nếu khớp, gán step_number vào Step và step_data['step'] vào Instructions cho nguyên liệu đó trong df1.

- Kết quả sẽ tương tự như bảng dưới đây:

Recipe Name	Ingredient Group	Amount	Unit	Ingredient Name	Step	Instructions
Lentil–Quinoa Tacos	No Group	1	cup	dry green lentils	1	Rinse the lentils and combine them with ...
Sweet Potato Mac N' Cheese	No Group	4	nan	garlic cloves	1	Roast the sweet potato and garlic at 400°F (205°C) ...

Bảng 1: Bảng dữ liệu minh họa cho việc tạo cột Steps và Instructions tương ứng với nguyên liệu sau khi phân loại chúng cùng nhau.

4.3.3. Phân loại nguyên liệu theo các hương vị cụ thể (Spicy, Bitter, Sweet, Sour, Savory)

Việc phân loại theo hương vị cụ thể để hỗ trợ và tích hợp vào thuật toán di truyền tối ưu cho nguyên liệu (GA Ingredients) khi người dùng lựa chọn các hương vị yêu thích và hương vị không muốn.

Bước 1: Khởi tạo các danh sách nguyên liệu theo hương vị:

```
# Danh sách các nguyên liệu cho từng hương vị
spicy_ingredients = ['jalapenos', 'chili', 'paprika', 'chilies', 'ginger', 'garlic', 'black pepper',
                     'serrano', 'chipotle', 'mustard', 'curry powder', 'onion', 'turmeric',
                     'Healthy Hot Sauce', 'chile']
bitter_ingredients = ['kale', 'brussels sprouts', 'spinach', 'arugula', 'fenugreek seeds', 'cacao', 'bok choy',
                      'watercress', 'oregano', 'cocoa', 'mustard', 'lettuce']
sweet_ingredients = ['bell pepper', 'ancho chile pepper', 'banana', 'apple', 'mango', 'red pepper', 'pumpkin pie spice',
                     'cinnamon', 'sweet potato', 'vanilla', 'cashew', 'coconut', 'molasses', 'apricot', 'dates', 'butter',
                     'medjool', 'milk', 'syrup', 'sugar', 'berries', 'raisin', 'peach', 'pomegranate', 'white miso paste',
                     'Cashew Cream', 'prune', 'watermelon', 'nectarine', 'plantain']
sour_ingredients = ['juice', 'juiced', 'lemon pepper', 'vinegar', 'plum', 'tomato', 'berries', 'guacamole', 'marinara',
                    'lime', 'lemon']
savory_ingredients = ['vegetable broth', 'garlic', 'onion', 'mushrooms', 'caps', 'basil', 'dill', 'rosemary',
                      'miso paste', 'chinese five-spice', 'asafoetida', 'thyme', 'sunflower', 'chickpeas', 'beans',
                      'tomato paste', 'sauce', 'nutritional yeast', 'Savory spice blend', 'butter',
                      'Super-Charged Spice Blend', 'lentils', 'Nutty Parm', 'guacamole', 'Umami', 'marinara', 'seasoning']

# Các từ khóa loại trừ (nếu có, sẽ trả về False cho mọi cột)
exclude_keywords = ['optional', 'garnish', 'desired','additional']
```

Hình 40: Khởi tạo các danh sách keyword nguyên liệu theo hương vị của chúng

- Tạo thủ công danh sách các nguyên liệu có hương vị tương ứng. Sau đó định nghĩa các từ khóa loại trừ như: “optional”, “garnish”, “additional” để loại trừ các nguyên liệu không liên quan đến việc đánh giá hương vị, nếu xuất hiện các từ khóa này, nó sẽ không được xem xét cho bất kỳ hương vị nào và sẽ trả về False cho toàn bộ hương vị.

Bước 2: Xác định hương vị của nguyên liệu bằng cách tạo hàm check_flavour. Hàm này kiểm tra xem một nguyên liệu có thuộc một danh sách hương vị nào không.

```
# Hàm kiểm tra nguyên liệu có thuộc hương vị không và loại trừ các từ khóa
def check_flavour(ingredient_name, flavour_list):
    ingredient_name_lower = ingredient_name.lower()

    # Nếu chứa từ khóa loại trừ, trả về False
    if any(exclude in ingredient_name_lower for exclude in exclude_keywords):
        return False

    # Nếu không chứa từ khóa loại trừ, kiểm tra hương vị
    return any(flavour in ingredient_name_lower for flavour in flavour_list)
```

Hình 41: Hàm kiểm tra nguyên liệu có thuộc hương vị hay không

Bước 3: Thêm các cột boolean (giá trị True/False) vào bộ dữ liệu nguyên liệu.

U	V	W	X	Y	Z	AA
Spicy	**Bitter**	**Sweet**	**Sour**	**Savory**	**Step**	**nstruction**
FALSE	TRUE	FALSE	FALSE	FALSE	Step 1	In a high-s
FALSE	FALSE	FALSE	FALSE	TRUE	Step 1	In a high-s
FALSE	FALSE	FALSE	FALSE	FALSE	Step 1	In a high-s
FALSE	FALSE	FALSE	FALSE	FALSE	Step 1	In a high-s
FALSE	FALSE	FALSE	FALSE	FALSE	Step 1	In a high-s
FALSE	FALSE	TRUE	FALSE	FALSE	Step 1	In a high-s
FALSE	FALSE	FALSE	FALSE	FALSE	Step 1	In a high-s
TRUE	FALSE	FALSE	FALSE	TRUE	Step 1	In a high-s
FALSE	FALSE	FALSE	TRUE	FALSE	Step 1	In a high-s
FALSE	FALSE	FALSE	FALSE	TRUE	Step 1	In a high-s
FALSE	FALSE	FALSE	FALSE	FALSE	Step 1	In a high-s

Hình 42: Giá trị Boolean cho các cột hương vị trong bộ dữ liệu chính

5. Phát triển ứng dụng

5.1. Môi trường phát triển

Ứng dụng này được phát triển bằng ngôn ngữ lập trình chính là Python cùng với các thư viện hỗ trợ mạnh mẽ cho xử lý dữ liệu, mô hình hóa, và xây dựng thuật toán di truyền. Các thư viện chính bao gồm:

- Pandas: Xử lý và quản lý dữ liệu dạng bảng, dùng để thao tác với tập dữ liệu nguyên liệu và công thức nấu ăn.

- NumPy: Cung cấp các công cụ xử lý mảng và tính toán, được sử dụng để lưu trữ quần thể và thực hiện các phép toán cần thiết trong thuật toán di truyền.

- DEAP (Distributed Evolutionary Algorithms in Python): Thư viện hỗ trợ thuật toán di truyền, cho phép chúng ta định nghĩa các cá thể, quần thể và các thao tác di truyền như chọn lọc, lai ghép và đột biến.

- Transformers: Sử dụng các mô hình ngôn ngữ BERT và T5 để nhận diện thực thể trong bước hướng dẫn nấu ăn và tóm tắt chúng. Điều này giúp phân tích và sắp xếp các bước nấu ăn theo thứ tự hợp lý..

- Requests: Dùng để gửi yêu cầu API đến các dịch vụ bên ngoài để lấy thông tin dinh dưỡng về các công thức nấu ăn đã tạo ra.

Môi trường phát triển trên Google Colab giúp truy cập nhanh đến các thư viện mới nhất và hỗ trợ xử lý dữ liệu lớn cũng như các yêu cầu tính toán chuyên sâu.

5.2. Xây dựng cấu trúc chương trình

Ứng dụng được chia thành các thành phần chính như sau:

a. Nhập dữ liệu từ người dùng:

Chương trình bắt đầu với việc yêu cầu người dùng nhập ít nhất ba nguyên liệu để tạo công thức món ăn.

Ngoài ra, người dùng chọn hương vị ưu tiên và không mong muốn (ngọt, cay, đắng, v.v.) cũng như số lượng khẩu phần ăn.

Nếu người dùng không cung cấp đủ nguyên liệu, chương trình gợi ý một số nguyên liệu phổ biến.

```python
def main():
    print("===ỨNG DỤNG TÌM KIẾM CÔNG THỨC NẤU ĂN VỚI NGUYÊN LIỆU CÓ SẴN CỦA BẠN")
    def suggest_user_ingredients():
      """Gợi ý một số nguyên liệu phổ biến nếu người dùng không biết chọn gì."""
      common_ingredients = ["broccoli", "cabbage", "tomato", "carrot", "potato", "red pepper", "berries", "lentils", "chickpeas"]
      print("Gợi ý nguyên liệu phổ biến:", ", ".join(common_ingredients))
    # Đọc dữ liệu từ người dùng
    while True:
      user_ingredients = input("Nhập ít nhất 3 nguyên liệu mà bạn muốn tạo món ăn (phân cách bởi dấu phẩy): ").split(',')
      user_ingredients = [i.strip() for i in user_ingredients if i.strip()]
      if len(user_ingredients) >= 3:
        break
      else:
        print("Bạn chưa nhập đủ 3 nguyên liệu. Vui lòng nhập lại.")
        suggest_user_ingredients()
```

Hình 43: Khởi tạo hàm main() cho ứng dụng dòng lệnh để chạy các hàm và thuật toán di truyền.

b. Thực thi thuật toán di truyền (GA):

Ứng dụng sử dụng hai thuật toán di truyền để tối ưu nguyên liệu và hướng dẫn nấu ăn. Cả hai được thiết kế để tìm ra danh sách nguyên liệu và thứ tự các bước nấu ăn tối ưu dựa trên sở thích của người dùng.

- GA cho nguyên liệu: Tạo ra quần thể ban đầu với các nguyên liệu kết hợp từ người dùng và thảo mộc/gia vị ngẫu nhiên. Qua các thế hệ, thuật toán thực hiện chọn lọc, lai ghép và đột biến để tìm ra danh sách nguyên liệu tối ưu nhất dựa trên khẩu vị người dùng.

- GA cho hướng dẫn: Dựa vào danh sách nguyên liệu đã chọn, thuật toán lọc ra các bước hướng dẫn liên quan từ cơ sở dữ liệu. Sau đó, các bước được sắp xếp và tối ưu bằng GA để tạo thành quy trình nấu ăn hợp lý và ngắn gọn nhất.

c. Hiển thị kết quả

Sau khi hoàn tất GA, chương trình hiển thị danh sách nguyên liệu tối ưu với số lượng chính xác cho số khẩu phần người dùng chọn. Quy trình nấu ăn cũng được hiển thị với từng bước cụ thể.

Ngoài ra, chương trình sử dụng API Edamam để lấy thông tin dinh dưỡng cho danh sách nguyên liệu cuối cùng.

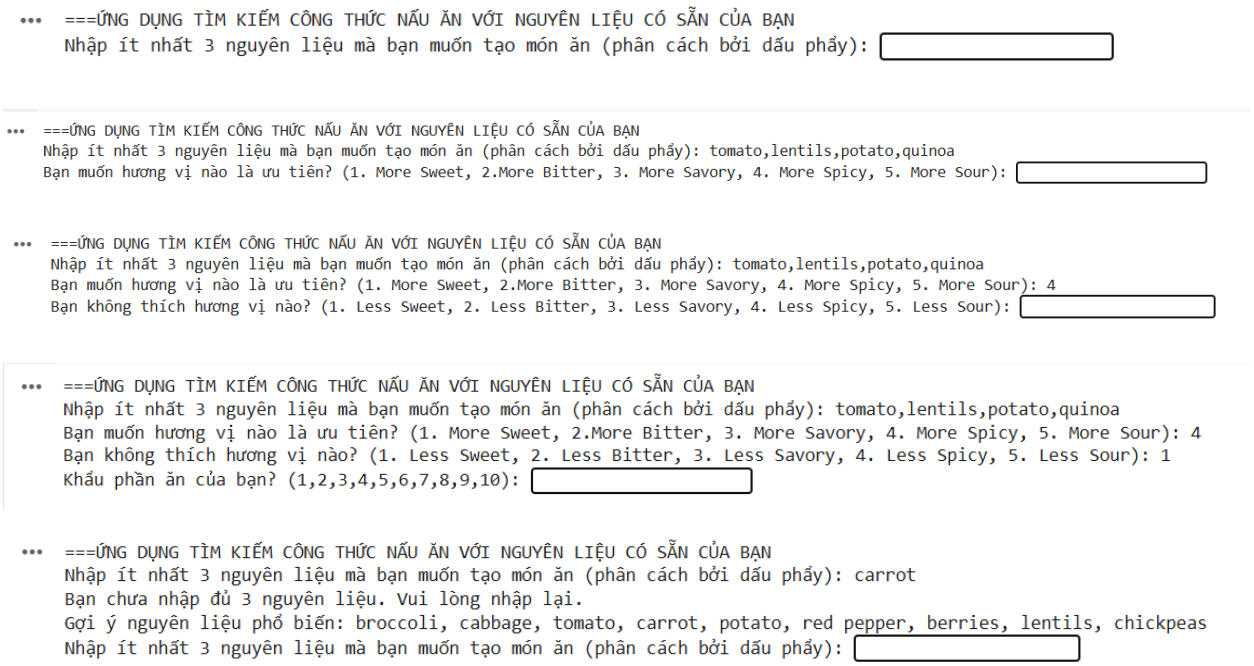
5.3. Giao diện dòng lệnh (CLI)

Ứng dụng sử dụng giao diện dòng lệnh (CLI) để người dùng nhập thông tin và tương tác trực tiếp. Các bước tương tác chính bao gồm:

a. Khởi động chương trình: Người dùng chạy file Python chính để khởi động CLI.

b. Nhập thông tin nguyên liệu và hương vị ưu tiên: Ứng dụng bắt đầu với thông báo chào mừng và yêu cầu người dùng nhập nguyên liệu, kèm theo lựa chọn hương vị ưu tiên.

Nếu người dùng nhập chưa đủ số lượng nguyên liệu, chương trình sẽ gợi ý một danh sách nguyên liệu phổ biến.

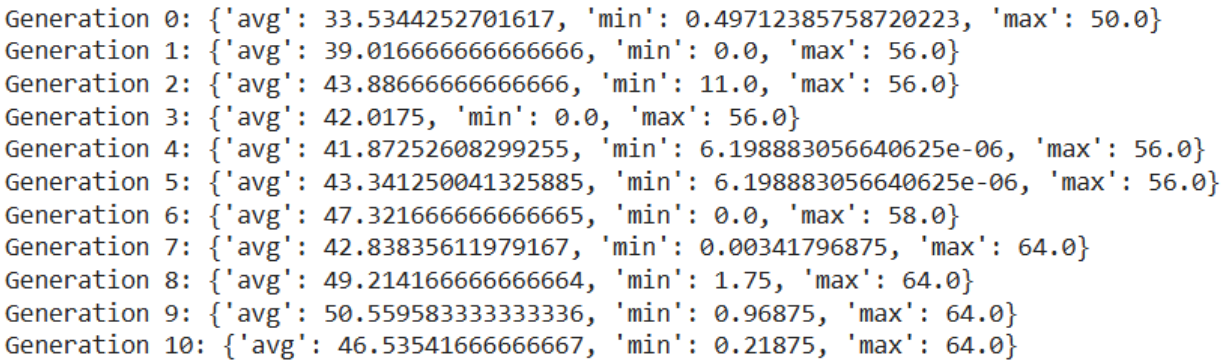


Hình 44: Giao diện dòng lệnh

Ở đây người dùng sẽ nhập ít nhất 3 nguyên liệu, nhập số tương ứng với hương vị ưu tiên và hương vị không thích. Cuối cùng là nhập khẩu phần mong muốn từ 1 đến 10 người.

c. Chạy thuật toán GA và xử lý: CLI sẽ hiển thị tiến trình xử lý của GA, thông báo số thế hệ và mức độ tối ưu của công thức theo thời gian thực.

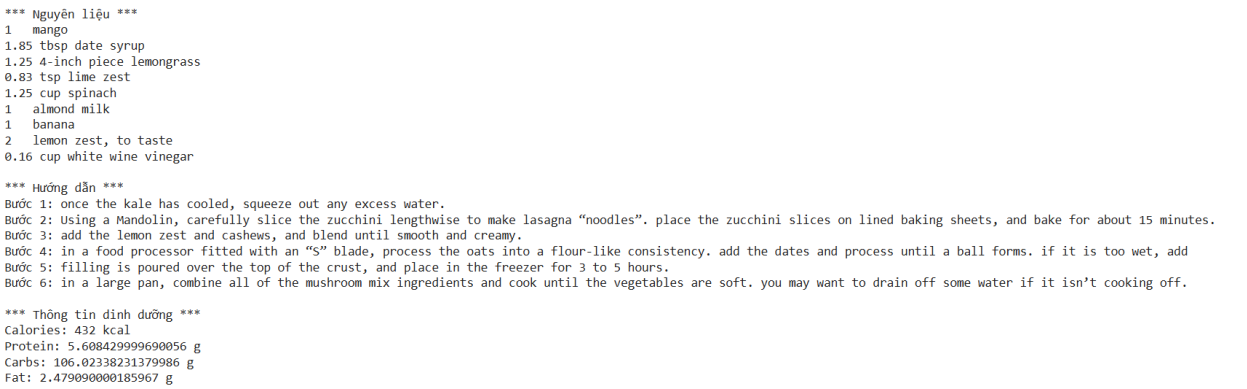
Giao diện minh họa:



Hình 45: Giao diện minh họa tiến trình xử lý GA để xem xét số thế hệ và điểm fitness

d. Hiển thị kết quả cuối cùng: Sau khi GA hoàn tất, ứng dụng xuất ra công thức tối ưu nhất với danh sách nguyên liệu, số lượng, đơn vị, hướng dẫn các bước nấu ăn và thông tin dinh dưỡng.

Giao diện minh họa:



Hình 46: Giao diện minh họa cho kết quả cuối cùng in ra nguyên liệu, công thức và thông tin dinh dưỡng.

6. Kết quả thực nghiệm

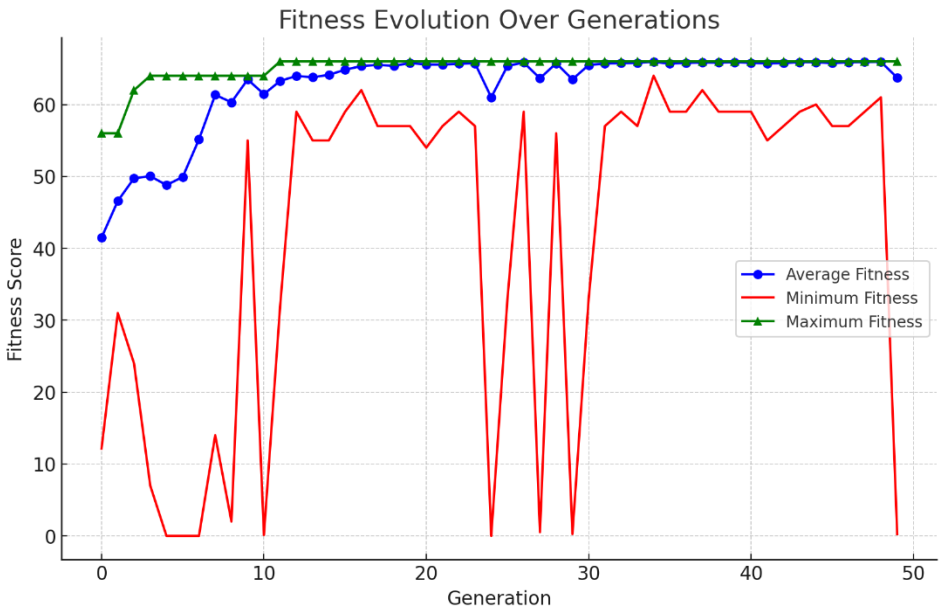
6.1. Ví dụ về công thức được tạo ra

Đầu vào người dùng nhập là kale, tomato, potato, carrot, lemon juice với hương vị ưu tiên là Savory và hương vị không muốn là Sweet.

```
*** Nguyên liệu ***
1   potato
2.14   garlic cloves, minced
1   tomato
1.33 tsp garlic
0.67 sprig rosemary
1.59 cup umami sauce
2.0 cups jarred or homemade marinara sauce
1.46 tbsp lemon juice
1.0   carrot
1   kale

*** Hướng dẫn ***
Bước 1: in a large bowl, combine all the salad ingredients, toss lightly with the dressing.
Bước 2: cashew mixture is very thick, stirring constantly. let the mixture cool down a few minutes before serving.
Bước 3: molasses, date syrup, tomato paste, black pepper, and black pepper. reduce the heat to low and simmer for 1 minute.
Bước 4: Using a Mandolin, carefully slice the zucchini lengthwise to make lasagna “noodles”. place the zucchini slices on lined baking sheets, and bake for about 15 minutes.
Bước 5: high-speed blender combines all ingredients with 2 cups of water. transfer to large glasses and serve.
Bước 6: mix in the miso mixture and stir to combine. cook until the tomatoes and greens are softened.

*** Thông tin dinh dưỡng ***
Calories: 546 kcal
Protein: 18.102170932456254 g
Carbs: 98.64409920135736 g
Fat: 11.381966008460523 g
```

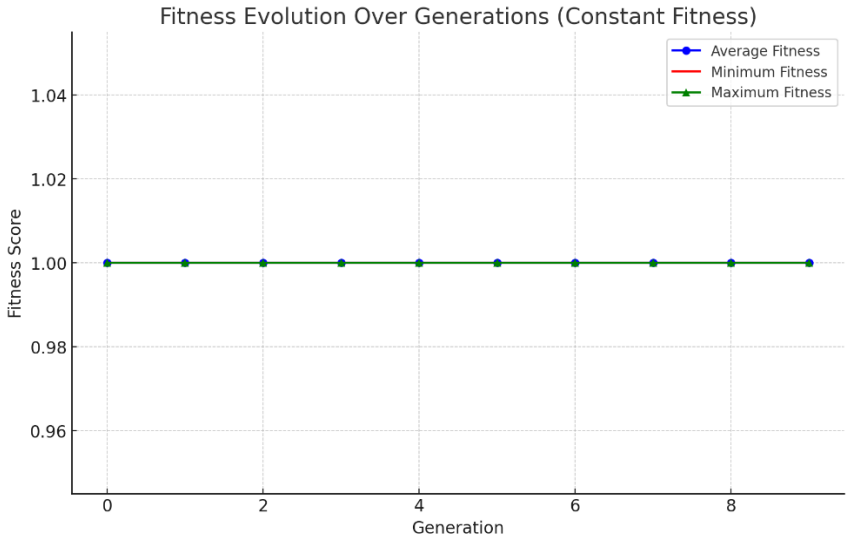
Hình 47: Kết quả thực nghiệm về công thức nấu ăn được tạo ra

Với fitness cho GA Ingredients:



Hình 48: Biểu đồ fitness GA Ingredients.

Với fitness cho GA Directions:



Hình 49: Biểu đồ fitness cho GA Directions

Đánh giá tổng quan:

- Ở fitness cho GA Ingredients:

- Tăng trưởng của Fitness Trung Bình: Có sự tăng trưởng về fitness trung bình qua các thế hệ, cho thấy quá trình tiến hóa đã giúp cải thiện giải pháp tổng thể. Tuy nhiên, xu hướng này không phải lúc nào cũng đều đặn và có một số giai đoạn trung bình giảm nhẹ. Điều này có thể cho thấy quá trình tối ưu hóa chưa hoàn toàn ổn định.
- Fitness Tối Thiểu Thấp ở Một Số Thế Hệ: Fitness tối thiểu có những giá trị rất thấp tại các thế hệ nhất định (ví dụ: thế hệ 4, 5, 24 và 29), điều này có thể chỉ ra rằng vẫn tồn tại một số giải pháp kém hiệu quả hoặc chưa hội tụ hoàn toàn. Những giá trị tối thiểu này cho thấy quần thể có thể có một số giải pháp không đáp ứng được mức fitness cao
- Fitness Tối Đa Ổn Định Cao: Fitness tối đa duy trì ở mức cao (64-66) từ rất sớm (từ thế hệ 9 trở đi) và ổn định, cho thấy thuật toán đã tìm ra một số giải pháp chất lượng từ sớm nhưng chưa hoàn toàn cải thiện sau mốc này. Đây là dấu hiệu của hội tụ, nhưng cũng có thể là chỉ dấu cho việc thuật toán có thể đã "mắc kẹt" tại một vùng giải pháp tốt mà không tiếp tục tối ưu thêm.
- Khách quan mà nói, biểu đồ này cho thấy một xu hướng cải thiện rõ rệt nhưng cũng tồn tại một số dấu hiệu chưa ổn định.

- Ở fitness cho GA Directions:

- Không có tiến triển: Fitness không thay đổi qua các thế hệ, điều này có thể là dấu hiệu của sự ngưng trệ trong quá trình tối ưu hóa. Thuật toán hoặc không có sự đa dạng để tiến hóa hoặc đã mắc kẹt ở một giá trị fitness cố định.

- Hội tụ sớm: Các giá trị fitness này có thể cho thấy thuật toán đã hội tụ rất sớm về một giải pháp, nhưng không nhất thiết là giải pháp tốt nhất có thể. Nếu đây là giá trị tối ưu thực sự, thì đây là kết quả thành công. Nhưng nếu bạn kỳ vọng fitness cao hơn, thì có thể cần điều chỉnh thuật toán.
- Giảm độ đa dạng: Điều này cũng có thể phản ánh rằng thuật toán thiếu cơ chế để duy trì hoặc tăng cường sự đa dạng giữa các cá thể, chẳng hạn như thông qua đột biến hoặc chọn lọc không hoàn toàn.

- Đánh giá sự phù hợp: Nguyên liệu phù hợp và đáp ứng sở thích của người dùng, thuật toán cũng đã lấy được thông tin dinh dưỡng dựa theo khẩu phần. Tuy nhiên logic hướng dẫn nấu ăn rời rạc và không ứng dụng thực tế được.

6.2. Phân tích ưu và nhược điểm của ứng dụng

a. Ưu điểm:

- Ứng dụng có thể thực hiện được việc tối ưu hóa nguyên liệu, tìm ra các giải pháp phù hợp với khẩu vị và tính toán được giá trị dinh dưỡng theo khẩu phần ăn.
- Ứng dụng tận dụng các thư viện phổ biến và mạnh mẽ như pandas, numpy, DEAP, và transformers, giúp dễ dàng xử lý dữ liệu, tối ưu hóa bằng thuật toán di truyền, và thực hiện các tác vụ NLP (Natural Language Processing).
- Ứng dụng kết nối với API dinh dưỡng từ Edamam để lấy thông tin dinh dưỡng của công thức đã tạo. Điều này rất hữu ích cho người dùng quan tâm đến lượng calo, protein, carbs, và chất béo trong khẩu phần ăn của mình.
- Sử dụng mô hình T5-base từ thư viện transformers để tóm tắt hướng dẫn nấu ăn, làm cho các bước trở nên dễ hiểu và ngắn gọn hơn.

b. Nhược điểm:

- Ứng dụng này có thể gặp vấn đề về hiệu suất, đặc biệt khi kích thước của dữ liệu lớn hoặc khi số thế hệ và kích thước quần thể trong GA lớn. Quá trình này có thể tốn nhiều thời gian và tài nguyên tính toán, đặc biệt trên môi trường Colab hoặc các thiết bị có cấu hình thấp.

- Trong các lần gọi API hoặc mô hình NLP, nếu có lỗi hoặc trục trặc, ứng dụng có thể dừng lại mà không cung cấp cách xử lý lỗi chi tiết, ảnh hưởng đến trải nghiệm người dùng.

- Quá trình tối ưu hóa và tổ chức các bước nấu ăn phức tạp, và nếu có nhiều bước giống nhau hoặc không liên quan đến nguyên liệu, sẽ có thể làm giảm hiệu quả của các thuật toán GA và mô hình NLP. Điều này có thể dẫn đến các hướng dẫn nấu ăn không hoàn toàn logic.

- Hạn chế về mức độ hội tụ trong thuật toán di truyền: Việc sử dụng thuật toán di truyền với các tham số cố định có thể gây ra hội tụ sớm hoặc không đạt được kết quả tối ưu trong

mọi trường hợp. Điều này có thể dẫn đến việc thuật toán không tìm ra danh sách nguyên liệu và hướng dẫn tốt nhất.

7. Thảo luận và các hạn chế của nghiên cứu

7.1. Thảo luận chung và các hạn chế của nghiên cứu

Mặc dù nghiên cứu này đã đạt được một số thành công trong việc sử dụng thuật toán di truyền để tối ưu hóa danh sách nguyên liệu và đáp ứng các ràng buộc về sở thích hương vị, nghiên cứu vẫn còn nhiều hạn chế cần được khắc phục để đạt được mục tiêu tổng quát là tạo ra công thức nấu ăn hoàn chỉnh với các bước hướng dẫn chi tiết và logic.

Việc chỉ sử dụng thuật toán di truyền là chưa đủ để xây dựng được các bước hướng dẫn nấu ăn có tính logic và ứng dụng cao. Thuật toán di truyền rất hiệu quả trong việc tối ưu hóa các nguyên liệu theo các ràng buộc và sở thích, nhưng nó không thể tự tạo ra các hướng dẫn có ý nghĩa và logic giống như cách con người hướng dẫn. Để tạo ra hướng dẫn nấu ăn hoàn chỉnh, cần có sự can thiệp của các mô hình ngôn ngữ tiên tiến như các mô hình NLP hiện đại (ví dụ: GPT-4, BERT) có khả năng hiểu ngữ cảnh và tổ chức các bước theo trình tự hợp lý.

Một trong những hạn chế quan trọng của nghiên cứu này là sự thiếu tích hợp với các mô hình ngôn ngữ tự nhiên (NLP) được huấn luyện tốt để đảm bảo các bước hướng dẫn được tạo ra có tính thống nhất và dễ hiểu. Các mô hình ngôn ngữ có khả năng lập trình tư duy logic có thể giúp đảm bảo rằng các bước hướng dẫn không chỉ phù hợp với nguyên liệu mà còn hợp lý về mặt ngữ cảnh và trật tự. Nếu không có sự hỗ trợ từ các mô hình này, các bước hướng dẫn có thể trở nên rời rạc và thiếu tính thực tiễn, mặc dù danh sách nguyên liệu đã được tối ưu hóa tốt.

Một yếu tố quan trọng khác là chất lượng của bộ dữ liệu huấn luyện. Để mô hình có thể tạo ra các công thức nấu ăn hoàn chỉnh, bộ dữ liệu cần phải đủ tinh tế và đa dạng, bao gồm không chỉ các thành phần nguyên liệu mà còn các bước hướng dẫn chi tiết, rõ ràng và có tính logic. Nếu bộ dữ liệu huấn luyện thiếu sót hoặc không đủ chi tiết, các bước hướng dẫn được tạo ra có thể không đảm bảo tính thống nhất và có thể dẫn đến các hướng dẫn không có ý nghĩa hoặc khó thực hiện.

Trong nghiên cứu này, phần lớn sự chú tâm được đặt vào tối ưu hóa nguyên liệu và đáp ứng các yêu cầu về khẩu phần và hương vị. Tuy nhiên, do không có các cơ chế kiểm soát để đảm bảo tính logic của các bước hướng dẫn, kết quả là các bước nấu ăn có thể không liên kết chặt chẽ với nhau hoặc không tuân theo trình tự hợp lý. Điều này dẫn đến một hạn chế lớn, khiến cho công thức cuối cùng có thể không thể áp dụng được trong thực tế, ngay cả khi danh sách nguyên liệu đã được tối ưu hóa.

Xây dựng một công thức nấu ăn không chỉ đơn giản là kết hợp các nguyên liệu phù hợp mà còn đòi hỏi sự hiểu biết sâu sắc về cách chế biến, thứ tự các bước, thời gian nấu nướng và các kỹ thuật chế biến. Do đó, nếu chỉ dựa vào thuật toán di truyền, nghiên cứu khó có thể đáp ứng được các yếu tố này. Để có thể tạo ra một công thức nấu ăn hoàn chỉnh, cần

tích hợp các phương pháp mô phỏng hành vi con người hoặc các mô hình học sâu chuyên biệt có khả năng hiểu và tạo ra các hướng dẫn chi tiết.

Do những hạn chế trên, khả năng ứng dụng của nghiên cứu trong thực tế còn nhiều hạn chế. Các bước hướng dẫn nấu ăn rời rạc có thể khiến người dùng gặp khó khăn khi làm theo công thức, dẫn đến trải nghiệm không như mong đợi. Để cải thiện điều này, cần phát triển thêm các phương pháp kiểm tra tính hợp lý của hướng dẫn và đảm bảo rằng chúng có thể áp dụng được trong thực tế.

7.2. Các cải tiến trong tương lai

a. Tích hợp các mô hình ngôn ngữ tự nhiên tiên tiến

Để cải thiện chất lượng và tính logic của các bước hướng dẫn nấu ăn, nghiên cứu trong tương lai có thể tích hợp các mô hình ngôn ngữ tự nhiên tiên tiến như GPT-4, PaLM 2 hoặc các mô hình fine-tune chuyên biệt dành cho hướng dẫn và quy trình. Các mô hình này có khả năng hiểu ngữ cảnh và tạo ra các chỉ dẫn rõ ràng, giúp cho các bước hướng dẫn không chỉ phù hợp với nguyên liệu mà còn hợp lý về mặt trình tự và logic.

Fine-tune mô hình ngôn ngữ trên các bộ dữ liệu nấu ăn có thể giúp hệ thống hiểu sâu hơn về quy trình nấu ăn, từ đó tạo ra các bước hướng dẫn chi tiết, dễ hiểu và phù hợp hơn với thực tế.

b. Sử dụng phân tích ngữ cảnh và hiểu ngữ nghĩa cho hướng dẫn nấu ăn

Sử dụng các kỹ thuật phân tích ngữ cảnh và hiểu ngữ nghĩa để kiểm tra và điều chỉnh các bước hướng dẫn. Điều này có thể được thực hiện thông qua các mô hình NLP có khả năng phân tích và phát hiện các lỗi logic trong chuỗi hành động. Chẳng hạn, nếu một bước hướng dẫn yêu cầu cho thêm muối sau khi món ăn đã hoàn tất, mô hình có thể phát hiện lỗi và đề xuất điều chỉnh lại.

Xây dựng cây quyết định hoặc đồ thị ngữ cảnh để xác định thứ tự các bước dựa trên yêu cầu nguyên liệu và phương pháp nấu, từ đó giúp đảm bảo tính hợp lý của các hướng dẫn.

c. Phát triển bộ dữ liệu huấn luyện đa dạng và chi tiết hơn

Để tăng khả năng tạo ra các công thức nấu ăn có tính ứng dụng cao, cần một bộ dữ liệu phong phú và chi tiết hơn, bao gồm không chỉ các thành phần nguyên liệu mà còn có các bước hướng dẫn, kỹ thuật chế biến và các tình huống thực tế trong nấu ăn. Bộ dữ liệu có thể được mở rộng để bao gồm các kỹ thuật nấu ăn chuyên biệt (như chiên, nướng, hấp).

d. Tối ưu hóa thuật toán di truyền

Cải tiến thuật toán di truyền để đảm bảo không chỉ tối ưu hóa nguyên liệu mà còn xem xét tính logic của các bước hướng dẫn. Có thể thiết kế thêm các hàm fitness để đo lường tính mạch lạc và độ khả thi của các hướng dẫn nấu ăn.

Thử nghiệm các chiến lược đột biến và lai ghép mới để đảm bảo sự đa dạng của quần thể, tránh hiện tượng hội tụ sớm và tăng cường khả năng khám phá các công thức mới, độc đáo.

e. Áp dụng kiểm tra trải nghiệm người dùng và phản hồi

Triển khai chương trình kiểm tra trải nghiệm người dùng để thu thập phản hồi và cải tiến hệ thống dựa trên các góp ý thực tế. Những thử nghiệm này sẽ giúp điều chỉnh các bước hướng dẫn, đảm bảo rằng công thức nấu ăn dễ thực hiện và phù hợp với người dùng.

Phân tích phản hồi từ người dùng để đánh giá hiệu quả của các cải tiến, xác định các điểm yếu và đưa ra điều chỉnh cụ thể cho hệ thống.

8. Kết luận

8.1 Tổng kết lại các kết quả chính đạt được

Nhìn chung, nghiên cứu đã đạt được các kết quả khả quan trong việc tối ưu hóa nguyên liệu, nhưng để đạt đến một hệ thống tạo công thức nấu ăn hoàn chỉnh, cần thêm những cải tiến về công nghệ và phương pháp xử lý ngôn ngữ tự nhiên. Với những bước phát triển tiếp theo, hệ thống này có tiềm năng trở thành một trợ lý nấu ăn thông minh và hữu ích cho người dùng, giúp họ sáng tạo và tận hưởng trải nghiệm nấu ăn một cách dễ dàng và hiệu quả hơn.

8.2 Các kiến thức đã ứng dụng và mở rộng

- Áp dụng thuật toán di truyền trong tối ưu hóa thực phẩm và dinh dưỡng: Đề tài này mở rộng ứng dụng của thuật toán di truyền (Genetic Algorithm) vào lĩnh vực tối ưu hóa nguyên liệu và quy trình nấu ăn, từ đó góp phần vào nghiên cứu về tối ưu hóa trong lĩnh vực dinh dưỡng. Việc sử dụng các thuật toán tính toán tiến hóa để tối ưu danh sách nguyên liệu và thứ tự các bước nấu ăn sẽ là một bước tiến trong lĩnh vực công nghệ thực phẩm nhờ vào khả năng sáng tạo và tìm kiếm giải pháp của nó. Tuy nhiên sự sáng tạo này cần được ràng buộc tinh tế để đảm bảo các logic thông thường trong ăn uống không bị phá vỡ, dẫn đến việc các công thức không thể ứng dụng vào thực tế được.

- Kết hợp xử lý ngôn ngữ tự nhiên (NLP) với tối ưu hóa công thức nấu ăn: Việc ứng dụng các mô hình NLP để phân loại và tối ưu các bước hướng dẫn nấu ăn là một cách tiếp cận mới và cần thiết trong nghiên cứu về xử lý ngôn ngữ trong ngữ cảnh ẩm thực. Hiện nay các mô hình ngôn ngữ tốt nhất có thể kể đến như GPT-4, PaLM 2 (Google), Claude 2 (Anthropic), LlaMA 2 (Meta)

- Phát triển mô hình tối ưu hóa đa mục tiêu cho chế độ dinh dưỡng: việc phát triển các hệ thống khuyến nghị đa mục tiêu trong lĩnh vực dinh dưỡng và chăm sóc sức khỏe rất tiềm năng và đã được biết đến rộng rãi. Đề tài đã sử dụng thuật toán di truyền với các tiêu chí đa mục tiêu (dinh dưỡng, khẩu vị, sở thích cá nhân) để tối ưu hóa công thức.

- Ứng dụng là một ví dụ về cách sử dụng các thuật toán AI để phát triển hệ thống gợi ý thực đơn thông minh, kết hợp giữa sở thích cá nhân và yêu cầu dinh dưỡng. Điều này có ý nghĩa quan trọng trong các nghiên cứu về hệ thống khuyến nghị, đặc biệt trong ngữ cảnh ẩm thực và dinh dưỡng.

- Đề tài tích hợp API dinh dưỡng để truy xuất thông tin thành phần dinh dưỡng theo thời gian thực. Phương pháp này giúp làm giàu dữ liệu của ứng dụng và đồng thời cung cấp một ví dụ điển hình về cách kết hợp dữ liệu từ nhiều nguồn để phục vụ mục tiêu tối ưu hóa.

Tài liệu tham khảo

[1] Research: Creative AI Through Evolutionary Computation: Principles and Examples, Risto Miikkulainen, SN Computer Science (2021). https://arxiv.org/abs/2008.04212

[2] https://en.wikipedia.org/wiki/Computational_creativity

[3] Research: A comprehensive review on identification of the geomaterial constitutive model using the computational intelligence method, Wei Gao, in Advanced Engineering Informatics (2018)

[4] https://en.wikipedia.org/wiki/Evolutionary_computation

[5] Research: Cooking recipes generator utilizing a deep learning-based language model, Michał Bień, Michał Gilski, Martyna Maciejewska, Wojciech Taisner

[6] A review on genetic algorithm: past, present, and future, Sourabh Katoch, Sumit Singh Chauhan, Vijay Kumar

[7] Forrest S, Mitchell M. Relative building-block ftness and the building-block hypothesis. In: Whitley LD, editor. Foundations of genetic algorithms. New York: Elsevier; 1993. p. 109–26.9

[8] Salimans T, Ho J, Chen X, Sutskever I. Evolution strategies as a scalable alternative to reinforcement learning. 2017. https://arxiv.org/abs/1703.03864

Mã nguồn

In [8]:
```
import pandas as pd
import numpy as np
import json
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from deap import base, creator, tools, algorithms
from transformers import pipeline
import random
from random import sample, uniform, randint, choice
import requests
import re
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
import time
from collections import defaultdict
import math
```

In [9]:
```
file_path1 = '/content/drive/MyDrive/sqlite/synchronize.xlsx'
file_path2 = '/content/drive/MyDrive/sqlite/recipes_instructions_grouped.xlsx'
```

In [10]:
```
synchronize_df = pd.read_excel(file_path1)
recipes_instructions_df = pd.read_excel(file_path2)
```

In [11]:
```
synchronize_df
```

In [14]:
```
herbs_and_spices_keywords = [
    "basil", "jalapeno", "garlic", "black pepper", "cumin", "oregano", "cilantro", "parsley", "paprika", "ancho chile",
    "lemon pepper", "zest of 1 orange", "lemon zest", "powder", "ground", "mustard", "chile", "asafoetida", "fenugreek",
    "cayenne", "garam masala", "lime zest", "ginger", "crushed red pepper", "red pepper flakes", "dill", "cinnamon",
    "chili", "thyme", "turmeric", "mint", "curry", "italian seasoning", "vanilla", "pumpkin pie spice", "sage", "bay",
    "celery seeds", "rosemary", "serrano", "coriander seeds", "lemongrass", "spices", "vinegar", "yeast", "miso",
    "Savory Spice Blend", "tahini", "Super-Charged Spice Blend", "Umami Sauce", "syrup", "blackstrap molasses",
    "Ranch Dressing", "Balsamic Date Glaze", "chlorella", "marinara"
]
```

In [15]:
```
pattern = re.compile(r'\b(' + '|'.join(re.escape(word) for word in herbs_and_spices_keywords) + r')\b', re.IGNORECASE)
```

In [16]:
```
# Load JSON Data
def load_json_data(file_path):
    with open(file_path, 'r') as file:
        return [json.loads(line) for line in file]

action_data = load_json_data('/content/drive/MyDrive/sqlite/Action.json')
food_entity_data = load_json_data('/content/drive/MyDrive/sqlite/Food_Entity.json')
process_data = load_json_data('/content/drive/MyDrive/sqlite/Process.json')
```

In [17]:
```
synchronize_df['Herbs and Spices'] = synchronize_df['Ingredient Name'].apply(lambda x: bool(pattern.search(str(x))))
```

In [17]:
```
synchronize_df['Herbs and Spices'] = synchronize_df['Ingredient Name'].apply(lambda x: bool(pattern.search(str(x))))
```

In [18]:
```
def add_servings_to_synchronize(synchronize_df, recipes_instructions_df):
    # Tạo một từ điển để tra cứu các giá trị 'Servings' từ `recipes_instructions_df`
    recipe_servings = recipes_instructions_df[['Recipe Name', 'Servings']].drop_duplicates('Recipe Name').set_index('Recipe

    # Thêm cột 'serving' vào `synchronize_df` dựa trên từ điển 'recipe_servings'
    synchronize_df['serving'] = synchronize_df['Recipe Name'].map(recipe_servings)

    return synchronize_df
```

In [19]:
```
synchronize_df = add_servings_to_synchronize(synchronize_df, recipes_instructions_df)
```

In [22]:
```
# Sử dụng mô hình nhận diện thực thể (NER)
ner = pipeline("ner", model="dbmdz/bert-large-cased-finetuned-conll03-english")
```

In [23]:
```
# Sử dụng mô hình NLP để phân loại các bước
classifier = pipeline("text-classification", model="distilbert-base-uncased-finetuned-sst-2-english")
```

In [24]:
```
def get_nutrition(ingredients):
    """Lấy thông tin dinh dưỡng sử dụng API Edamam."""
    APP_ID = 'dfc9551d'
    APP_KEY = '4086f5c1929cda0a712d09af7f377da9'
    url = 'https://api.edamam.com/api/nutrition-details'

    headers = {'Content-Type': 'application/json'}
    data = {"title": "Generated Recipe", "ingr": ingredients}

    try:
        response = requests.post(f'{url}?app_id={APP_ID}&app_key={APP_KEY}', headers=headers, json=data)
        response.raise_for_status()  # Kiểm tra xem yêu cầu có thành công không
        nutrition_data = response.json()

        # Trích xuất thông tin dinh dưỡng nếu có
        calories = nutrition_data.get('calories', 'Không có thông tin')
        protein = nutrition_data.get('totalNutrients', {}).get('PROCNT', {}).get('quantity', 'Không có thông tin')
        carbs = nutrition_data.get('totalNutrients', {}).get('CHOCDF', {}).get('quantity', 'Không có thông tin')
        fat = nutrition_data.get('totalNutrients', {}).get('FAT', {}).get('quantity', 'Không có thông tin')

        # Đơn vị của mỗi giá trị (g hoặc kcal)
        protein_unit = nutrition_data.get('totalNutrients', {}).get('PROCNT', {}).get('unit', 'g')
        carbs_unit = nutrition_data.get('totalNutrients', {}).get('CHOCDF', {}).get('unit', 'g')
        fat_unit = nutrition_data.get('totalNutrients', {}).get('FAT', {}).get('unit', 'g')

        # In thông tin dinh dưỡng
        print("\n*** Thông tin dinh dưỡng ***")
        print(f"Calories: {calories} kcal")
        print(f"Protein: {protein} {protein_unit}")
        print(f"Carbs: {carbs} {carbs_unit}")
        print(f"Fat: {fat} {fat_unit}")

        return nutrition_data
    except requests.exceptions.HTTPError as http_err:
        print(f"Lỗi HTTP khi gọi API: {http_err}")
        print(f"Nội dung phản hồi từ server: {response.text}")
    except requests.exceptions.RequestException as req_err:
        print(f"Lỗi khi gọi API: {req_err}")
    except Exception as e:
        print(f"Lỗi không xác định: {e}")
        return {}
```

In [25]:
```
def compute_average_amount(ingredient, target_servings):
  """Compute average amount for ingredients based on servings."""
  recipes_with_ingredient = synchronize_df[synchronize_df['Ingredient Name'] == ingredient]

  if recipes_with_ingredient.empty or recipes_with_ingredient['Amount'].isnull().all():
    return 1

  avg_amount = recipes_with_ingredient['Amount'].dropna().mean()
  avg_servings = recipes_with_ingredient['serving'].dropna().mean() if not recipes_with_ingredient['serving'].isnull().all(
```

```
  if avg_servings > 0:
    return (avg_amount / avg_servings) * target_servings

  return 1
```

In [26]:
```python
# Khởi tạo môi trường GA cho Ingredients
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)
```

In [27]:
```python
# Genetic Algorithm Functions
# Hàm tạo cá thể cho GA cho Ingredients
def create_individual(user_ingredients, herbs_spices):
  base_ingredients = set(user_ingredients.copy())
  additional_herbs_spices = sample([herb for herb in herbs_spices if herb not in base_ingredients], randint(2, 5))
  return list(base_ingredients.union(additional_herbs_spices))
```

In [28]:
```python
#Đột biến trên hương vị
def flavor_based_mutation(individual, flavor_preferences, herbs_spices, user_ingredients, indpb=0.1):
  for i in range(len(individual)):
    if individual[i] not in user_ingredients and random.random() < indpb:
      possible_choices = [herb for herb in herbs_spices if herb not in individual]
      selected_flavor = random.choice(flavor_preferences['more'])
      filtered_choices = [choice for choice in possible_choices if synchronize_df.loc[synchronize_df['Ingredient Name'] == c
      if filtered_choices:
        individual[i] = random.choice(filtered_choices)
      else:
        individual[i] = random.choice(possible_choices)

  individual = creator.Individual(dict.fromkeys(individual))
  return individual,
```

In [29]:
```python
def custom_crossover(ind1, ind2, user_ingredients, swap_prob=0.5):
    """Hàm crossover tùy chỉnh sử dụng phương pháp Blend nhưng bảo vệ nguyên liệu của người dùng."""
    min_length = min(len(ind1), len(ind2))

    for i in range(min_length):
        # Chỉ hoán đổi nếu gene không phải là nguyên liệu của người dùng
        if ind1[i] not in user_ingredients and ind2[i] not in user_ingredients:
            # Với xác suất `swap_prob`, hoán đổi các gene giữa ind1 và ind2
            if random.random() < swap_prob:
                ind1[i], ind2[i] = ind2[i], ind1[i]

    # Đảm bảo các cá thể trả về là dạng `creator.Individual`
    ind1 = creator.Individual(dict.fromkeys(ind1))
    ind2 = creator.Individual(dict.fromkeys(ind2))

    return ind1, ind2
```

In [30]:
```python
def evaluate_individual(individual, flavor_preferences, user_ingredients, diversity_bonus=2):
    """Evaluate the fitness of an individual based on user preferences."""
    score = 0
    preferred_weight = 6
    disliked_weight = -5
    user_ingredient_weight = 8

    for ingredient in user_ingredients:
      if ingredient in individual:
        score += user_ingredient_weight

    for ingredient in individual:
        # Evaluate if the ingredient matches the preferred flavors
        for flavor in flavor_preferences['more']:
            if synchronize_df.loc[synchronize_df['Ingredient Name'] == ingredient, flavor].any():
                score += preferred_weight
        # Penalize if the ingredient matches the disliked flavors
        for flavor in flavor_preferences['less']:
            if synchronize_df.loc[synchronize_df['Ingredient Name'] == ingredient, flavor].any():
                score += disliked_weight

    unrelated_ingredients = set(individual) - set(user_ingredients)
    if len(unrelated_ingredients) > len(user_ingredients) / 2:
      score -= 10

    # Bonus điểm dựa trên độ đa dạng của nguyên liệu
    unique_herbs_spices = set([i for i in individual if i in herbs_and_spices_keywords])
    score += len(unique_herbs_spices) * diversity_bonus

    return score,
```

In [31]:
```python
def update_population_fitness_with_sharing(population, flavor_preferences, user_ingredients, sharing_radius=3.0):
    """Update fitness of the population using Fitness Sharing."""
    for ind in population:
        base_fitness = evaluate_individual(ind, flavor_preferences, user_ingredients)[0]
        shared_fitness = base_fitness

        for other in population:
            if other != ind:
                distance = len(set(ind) ^ set(other))  # Đo sự khác biệt giữa hai cá thể
                if distance < sharing_radius:
                    # Giảm điểm fitness nếu khoảng cách nhỏ hơn sharing_radius
                    shared_fitness *= (1 - (sharing_radius - distance) / sharing_radius)

        ind.fitness.values = (shared_fitness,)
```

In [32]:
```python
def adaptive_mutation_rate(population, current_gen, max_gen):
    """Điều chỉnh tỷ lệ đột biến dựa trên mức độ hội tụ của quần thể."""
    diversity = calculate_population_diversity(population)
    base_mutation_rate = 0.1  # Tỷ lệ đột biến cơ bản
    max_mutation_rate = 0.5   # Tỷ lệ đột biến tối đa

    # Nếu đa dạng quần thể thấp, tăng tỷ lệ đột biến
    if diversity < 0.3:  # Ngưỡng đa dạng thấp
        adaptive_mutation = base_mutation_rate + (max_mutation_rate - base_mutation_rate) * (1 - (current_gen / max_gen))
    else:
        adaptive_mutation = base_mutation_rate

    if diversity < 0.1:
        adaptive_mutation += 0.3
    return min(adaptive_mutation, max_mutation_rate)  # Giới hạn tỷ lệ đột biến
```

In [33]:
```python
def merge_ingredient_variations(ingredients_list):
    """Hợp nhất các biến thể nguyên liệu trùng lặp và cộng số lượng, không lặp mô tả."""
    # Sử dụng defaultdict để lưu trữ thông tin về mỗi nguyên liệu
    ingredient_dict = defaultdict(lambda: {"amount": 0, "unit": "", "description": set()})

    for ingredient in ingredients_list:
        base_name = ingredient['ingredient'].lower()
        amount = ingredient['amount']
        unit = ingredient['unit']
        description = ingredient.get('description', base_name)  # Dùng tên cơ bản nếu không có mô tả

        # Cộng dồn số lượng cho nguyên liệu trùng tên
        ingredient_dict[base_name]["amount"] += amount
        ingredient_dict[base_name]["unit"] = unit if ingredient_dict[base_name]["unit"] == "" else ingredient_dict[base_name
        ingredient_dict[base_name]["description"].add(description)

    # Kết hợp lại thành danh sách tối ưu
    merged_ingredients = []
    for base_name, details in ingredient_dict.items():
        merged_description = ", ".join(sorted(details["description"]))
        merged_ingredient = f"{details['amount']} {details['unit']} {merged_description}".strip()
        merged_ingredients.append(merged_ingredient)

    return merged_ingredients
```

In [34]:
```python
# Hàm rank-based selection mới
def rank_based_selection(population, k, elitism_rate=0.15):
    """Rank-based Selection with Elitism."""
    # Xếp hạng các cá thể dựa trên fitness của chúng (từ cao xuống thấp)
    ranked_population = sorted(population, key=lambda ind: ind.fitness.values[0], reverse=True)

    # Giữ lại các cá thể tốt nhất dựa trên elitism rate
    elite_count = int(len(population) * elitism_rate)
    elite_individuals = ranked_population[:elite_count]

    # Tính tổng số thứ hạng
    total_ranks = sum(range(1, len(ranked_population) + 1))

    # Xác suất chọn dựa trên thứ hạng
    probabilities = [(len(ranked_population) - rank + 1) / total_ranks for rank in range(1, len(ranked_population) + 1)]

    # Chọn cá thể ngẫu nhiên dựa trên xác suất từ thứ hạng
    selected_individuals = random.choices(ranked_population, weights=probabilities, k=k - elite_count)

    # Kết hợp các cá thể elite với các cá thể được chọn
    return elite_individuals + selected_individuals
```

In [35]:
```python
def calculate_population_diversity(population):
    """Tính độ đa dạng của quần thể dựa trên độ khác biệt giữa các cá thể."""
    unique_individuals = set(tuple(ind) for ind in population)
    diversity_ratio = len(unique_individuals) / len(population)

    differences = [len(set(ind1) ^ set(ind2)) for ind1 in population for ind2 in population if ind1 != ind2]
    avg_difference = np.mean(differences) if differences else 0

    return diversity_ratio
```

In [53]:
```python
def genetic_algorithm_ingredients(user_ingredients, herbs_spices, flavor_preferences, ngen=50, pop_size=150, cxpb=0.6, mutpb
  toolbox = base.Toolbox()
  toolbox.register("individual", tools.initIterate, creator.Individual, lambda: create_individual(user_ingredients, herbs_sp
  toolbox.register("population", tools.initRepeat, list, toolbox.individual)
  toolbox.register("mate", custom_crossover, user_ingredients=user_ingredients, swap_prob=0.5)
  toolbox.register("mutate", flavor_based_mutation, flavor_preferences=flavor_preferences, herbs_spices=herbs_spices, user_
  toolbox.register("select", rank_based_selection)
  toolbox.register("evaluate", lambda ind: evaluate_individual(ind, flavor_preferences, user_ingredients, diversity_bonus=2

  population = toolbox.population(n=pop_size)
  invalid_ind = [ind for ind in population if not ind.fitness.valid]
  update_population_fitness_with_sharing(invalid_ind, flavor_preferences, user_ingredients, sharing_radius=3.0)

  hof = tools.HallOfFame(10)
  stats = tools.Statistics(lambda ind: ind.fitness.values)
  stats.register("avg", np.mean)
  stats.register("min", np.min)
  stats.register("max", np.max)

  for gen in range(ngen):
    mutpb = adaptive_mutation_rate(population, gen, ngen)
    offspring = toolbox.select(population, len(population) - len(hof))
    offspring = list(map(toolbox.clone, offspring))

    # Lai ghép và đột biến
    for child1, child2 in zip(offspring[::2], offspring[1::2]):
      if random.random() < cxpb:
        toolbox.mate(child1, child2)
        del child1.fitness.values
        del child2.fitness.values
```

```
    for mutant in offspring:
      if random.random() < mutpb:
        if random.random() < 0.5:
          toolbox.mutate(mutant)
        del mutant.fitness.values

    # Đánh giá lại các cá thể con sau khi lai ghép và đột biến
    invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
    update_population_fitness_with_sharing(invalid_ind, flavor_preferences, user_ingredients, sharing_radius=2.0)

    # Cập nhật quần thể với các cá thể tốt nhất từ thế hệ trước
    population[:] = offspring + list(hof)

    # Cập nhật Elitism
    hof.update(population)

    # Thu thập số liệu thống kê
    record = stats.compile(population)
    print(f"Generation {gen}: {record}")

    # Điều chỉnh tỷ lệ đột biến dựa trên sự đa dạng sau mỗi 10 thế hệ
    if gen % 10 == 0:
      diversity = calculate_population_diversity(population)
      if diversity < 0.2:
        mutpb = min(mutpb + 0.05, 0.5)
      else:
        mutpb = max(mutpb - 0.05, 0.1)

  # Trả về cá thể tốt nhất từ Hall of Fame
  best_individual = tools.selBest(population, 1)[0]
  return best_individual
```

In [54]:

```
from difflib import SequenceMatcher

def are_steps_similar(step1, step2, threshold=0.8):
    """Kiểm tra nếu hai bước hướng dẫn tương tự nhau dựa trên một ngưỡng."""
    similarity = SequenceMatcher(None, step1, step2).ratio()
    return similarity >= threshold
```

In [55]:

```
def adjust_fitness(population):
    """Điều chỉnh fitness để không có giá trị âm khi sử dụng selRoulette."""
    min_fitness = min(ind.fitness.values[0] for ind in population)
    if min_fitness < 0:
        adjustment = abs(min_fitness) + 1  # Thêm lượng bù dương
        for ind in population:
            ind.fitness.values = (ind.fitness.values[0] + adjustment,)
```

In [56]:

```
tokenizer = AutoTokenizer.from_pretrained("t5-base")
model = AutoModelForSeq2SeqLM.from_pretrained("t5-base")

def refine_instructions_with_model(steps):
    """Sử dụng mô hình NLP để làm sáng tỏ và tóm tắt hướng dẫn."""
    refined_steps = []
    for step in steps:
        input_ids = tokenizer.encode(f"summarize: {step}", return_tensors="pt")
        summary_ids = model.generate(input_ids, max_length=50, min_length=10, length_penalty=2.0, num_beams=4, early_stoppi
        summary = tokenizer.decode(summary_ids[0], skip_special_tokens=True)
        refined_steps.append(summary)

    return refined_steps
```

In [57]:

```
# Genetic Algorithm for Directions
creator.create("FitnessDirection", base.Fitness, weights=(1.0,))
creator.create("DirectionIndividual", list, fitness=creator.FitnessDirection)
```

In [58]:
```
def remove_duplicates(directions):
    """Remove duplicate steps from directions."""
    seen = set()
    unique_directions = []
    for step in directions:
        if not any(are_steps_similar(step, seen_step) for seen_step in seen):
            unique_directions.append(step)
            seen.add(step)
    return unique_directions
```

In [59]:
```
# Direction Handling and Genetic Algorithm
def create_direction_individual(all_steps, threshold=0.8):
  """Create an individual for recipe directions."""
  individual = []
  while len(individual) < randint(5,8):
    new_step = random.choice(all_steps)
    if not any(are_steps_similar(new_step, existing_step, threshold) for existing_step in individual):
      individual.append(new_step)
  individual = remove_duplicates(individual)
  return individual
```

In [60]:
```
def categorize_steps_with_nlp(steps):
      """Phân loại và sắp xếp các bước nấu ăn theo thứ tự hợp lý."""
      categories = {'preparation': [], 'cooking': [], 'finishing': []}
      for step in steps:
        # Phân loại bước dựa trên mô hình NLP
        prediction = classifier(step)[0]['label'].lower()

        if 'prep' in prediction:
          categories['preparation'].append(step)
        elif 'cook' in prediction:
          categories['cooking'].append(step)
        elif 'finish' in prediction:
          categories['finishing'].append(step)
        else:
          categories['cooking'].append(step)

        # Sắp xếp theo thứ tự: Chuẩn bị -> Nấu chính -> Hoàn thiện
      sorted_directions = categories['preparation'] + categories['cooking'] + categories['finishing']
      return sorted_directions
```

In [61]:
```
def filter_directions_by_ingredients_nlp(directions, relevant_ingredients):
      """Lọc các bước hướng dẫn dựa trên các nguyên liệu chính."""
      filtered_directions = []

      for step in directions:
        entities = ner(step)
        detected_ingredients = [entity['word'].lower() for entity in entities if entity['entity'] == 'B-INGREDIENT']

        if sum(1 for ingredient in relevant_ingredients if ingredient in detected_ingredients) >= len(relevant_ingredients)
          filtered_directions.append(step)

      return filtered_directions
```

In [62]:
```
def extract_relevant_processes(relevant_ingredients, process_data):
      """Extract relevant processes for evaluation based on selected ingredients."""
      relevant_process = {}
      for entry in process_data:
        if isinstance(entry, dict):
            recipe_name = list(entry.keys())[0]
            steps = entry[recipe_name]

            processes = []

            if isinstance(steps, dict):
                for step_key, step_data in steps.items():
                    if isinstance(step_data, list):
                        processes.extend(step_data)
                    else:
                        processes.append(step_data)
            relevant_process[recipe_name] = processes
      return relevant_process
```

In [63]:

```
def evaluate_direction_individual(individual, relevant_ingredients, synchronize_df, process_data):
        """Evaluate the fitness of an individual direction."""
        unique_individual = remove_duplicates(individual)
        filtered_steps = filter_directions_by_ingredients_nlp(unique_individual, relevant_ingredients)
        sorted_steps = categorize_steps_with_nlp(filtered_steps)
        score = 0
        unique_steps = set(sorted_steps)  # Sử dụng tập hợp để kiểm tra trùng lặp
        # Nếu số lượng các bước duy nhất nhỏ hơn số bước tổng cộng, phạt điểm
        if len(unique_steps) < len(sorted_steps):
          score -= (len(sorted_steps) - len(unique_steps)) * 3  # Phạt cho mỗi bước trùng lặp

        covered_ingredients = set()
        for step in sorted_steps:
          for ingredient in relevant_ingredients:
            if ingredient in step:
              covered_ingredients.add(ingredient)
              score +=4

        if len(covered_ingredients) < len(relevant_ingredients) / 2:
          score -= 3

        previous_step = None
        for step in unique_individual:
          if previous_step and are_steps_similar(previous_step, step):
            score -= 2
          previous_step = step

        return score,
```

In [64]:

```
 # Extract relevant steps based on ingredients
def extract_relevant_steps(ingredients, synchronize_df, recipes_instructions_df, food_entity_data):
        """Extract relevant steps for cooking based on the selected ingredients."""
        all_steps = []
        food_entities = set()

        for entry in food_entity_data:
          recipe_name = entry.get("Recipe Name", "")
          entities = entry.get("Food Entity", [])
          food_entities.update(entities)

        for ingredient in ingredients:
            related_steps = synchronize_df[synchronize_df['Ingredient Name'] == ingredient]
            related_steps = related_steps[['Step', 'Instructions', 'Recipe Name']].dropna(subset=['Step', 'Instructions'])

            if related_steps.empty:
              continue

            all_steps.extend(related_steps['Instructions'].tolist())

            related_recipes = related_steps['Recipe Name'].unique()

            for recipe_name in related_recipes:
              full_recipe_instructions = recipes_instructions_df[recipes_instructions_df['Recipe Name'] == recipe_name][['St

              filtered_instructions = []
              for _, row in full_recipe_instructions.iterrows():
                step = row['Instructions']
                words = step.split()

                contains_relevant_ingredient = any(ingredient in step for ingredient in ingredients)

                contains_unrelated_entity = any(word in food_entities and word not in ingredients for word in words)


            if not contains_unrelated_entity:
              filtered_instructions.append(step)

          all_steps.extend(filtered_instructions)

      return all_steps
```

In [65]:
```python
def two_point_crossover(ind1, ind2):
    """Thực hiện two-point crossover."""
    if len(ind1) < 3 or len(ind2) < 3:
        return ind1, ind2  # Nếu cá thể quá ngắn, không thực hiện crossover

    # Chọn hai điểm crossover ngẫu nhiên
    point1 = random.randint(1, min(len(ind1), len(ind2)) - 2)
    point2 = random.randint(point1 + 1, min(len(ind1), len(ind2)) - 1)

    # Tạo ra hai con bằng cách trao đổi phần giữa hai điểm crossover
    child1 = ind1[:point1] + ind2[point1:point2] + ind1[point2:]
    child2 = ind2[:point1] + ind1[point1:point2] + ind2[point2:]

    return creator.DirectionIndividual(child1), creator.DirectionIndividual(child2)
```

In [66]:
```python
def update_population_fitness_with_sharing_directions(population, relevant_ingredients, synchronize_df, process_data, sharin
    """Update fitness of the population in GA directions using Fitness Sharing."""
    for ind in population:
        # Tính fitness cơ bản cho từng cá thể
        base_fitness = evaluate_direction_individual(ind, relevant_ingredients, synchronize_df, process_data)[0]
        shared_fitness = base_fitness

        # Điều chỉnh fitness dựa trên sự tương đồng của cá thể với các cá thể khác trong quần thể
        for other in population:
            if other != ind:
                # Tính khoảng cách giữa các cá thể dựa trên số bước khác biệt
                distance = len(set(ind) ^ set(other))  # Đo sự khác biệt giữa hai cá thể
                if distance < sharing_radius:
                    # Giảm điểm fitness nếu khoảng cách nhỏ hơn sharing_radius
                    shared_fitness *= (1 - (sharing_radius - distance) / sharing_radius)

        # Cập nhật lại giá trị fitness đã được điều chỉnh
        ind.fitness.values = (shared_fitness,)
```

In [67]:
```python
def genetic_algorithm_directions(all_relevant_steps, relevant_ingredients, ngen=10, pop_size=30, cxpb=0.6, mutpb=0.2, thresh
        """Chạy thuật toán GA để tối ưu hóa hướng dẫn nấu ăn."""
        toolbox_directions = base.Toolbox()
        toolbox_directions.register("individual", tools.initIterate, creator.DirectionIndividual,
                                    lambda: create_direction_individual(all_relevant_steps, threshold=0.85))
        toolbox_directions.register("population", tools.initRepeat, list, toolbox_directions.individual)
        toolbox_directions.register("mate", two_point_crossover)
        toolbox_directions.register("mutate", tools.mutShuffleIndexes, indpb=0.2)
        toolbox_directions.register("select", tools.selRoulette)
        toolbox_directions.register("evaluate", lambda ind: evaluate_direction_individual(ind, relevant_ingredients, synchr

        population = toolbox_directions.population(n=pop_size)
        invalid_ind = [ind for ind in population if not ind.fitness.valid]
        update_population_fitness_with_sharing_directions(invalid_ind, relevant_ingredients, synchronize_df, process_data, s
        adjust_fitness(invalid_ind)

        hof = tools.HallOfFame(10)
        stats = tools.Statistics(lambda ind: ind.fitness.values)
        stats.register("avg", np.mean)
        stats.register("min", np.min)
        stats.register("max", np.max)
```

```
    # Bắt đầu các thế hệ GA
    for gen in range(ngen):
      mutpb = adaptive_mutation_rate(population, gen, ngen)

      # Chọn các cá thể con từ quần thể hiện tại bằng Roulette Selection
      offspring = toolbox_directions.select(population, len(population) - len(hof))
      offspring = list(map(toolbox_directions.clone, offspring))

      # Lai ghép (crossover) và đột biến (mutation)
      for child1, child2 in zip(offspring[::2], offspring[1::2]):
        if random.random() < cxpb:
          toolbox_directions.mate(child1, child2)
          del child1.fitness.values
          del child2.fitness.values
      for mutant in offspring:
        if random.random() < mutpb:
          toolbox_directions.mutate(mutant)
          del mutant.fitness.values

      # Đánh giá lại các cá thể không hợp lệ sau đột biến và lai ghép
      invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
      if gen % 10 == 0 and invalid_ind:
        update_population_fitness_with_sharing_directions(invalid_ind, relevant_ingredients, synchronize_df, process_dat
        adjust_fitness(invalid_ind)  # Điều chỉnh fitness nếu cần thiết cho selRoulette

      # Kết hợp offspring với các cá thể elitist
      population[:] = tools.selBest(population + offspring, pop_size)

      # Cập nhật Elitism
      hof.update(population)

      # Thu thập số liệu thống kê cho thế hệ hiện tại
      record = stats.compile(population)
      print(f"Generation {gen}: {record}")
```

In [72]:

```
# Hàm chính
def main():
    print("===ỨNG DỤNG TÌM KIẾM CÔNG THỨC NẤU ĂN VỚI NGUYÊN LIỆU CÓ SẴN CỦA BẠN")
    def suggest_user_ingredients():
      """Gợi ý một số nguyên liệu phổ biến nếu người dùng không biết chọn gì."""
      common_ingredients = ["broccoli", "cabbage", "tomato", "carrot", "potato", "red pepper", "berries", "lentils", "chick
      print("Gợi ý nguyên liệu phổ biến:", ", ".join(common_ingredients))
    # Đọc dữ liệu từ người dùng
    while True:
      user_ingredients = input("Nhập ít nhất 3 nguyên liệu mà bạn muốn tạo món ăn (phân cách bởi dấu phẩy): ").split(',')
      user_ingredients = [i.strip() for i in user_ingredients if i.strip()]
      if len(user_ingredients) >= 3:
        break
      else:
        print("Bạn chưa nhập đủ 3 nguyên liệu. Vui lòng nhập lại.")
        suggest_user_ingredients()

    more_flavor = int(input("Bạn muốn hương vị nào là ưu tiên? (1. More Sweet, 2.More Bitter, 3. More Savory, 4. More Spicy,
    less_flavor = int(input("Bạn không thích hương vị nào? (1. Less Sweet, 2. Less Bitter, 3. Less Savory, 4. Less Spicy, 5

    target_servings = int(input("Khẩu phần ăn của bạn? (1,2,3,4,5,6,7,8,9,10): "))

    # Lựa chọn hương vị ưu tiên và không thích
    flavors = ['Sweet', 'Bitter', 'Savory', 'Spicy', 'Sour']
    flavor_preferences = {
        'more': [flavors[more_flavor - 1]],
        'less': [flavors[less_flavor - 1]]
    }

    # Lấy danh sách các loại thảo mộc/gia vị từ bộ dữ liệu synchronize
    herbs_spices = synchronize_df[synchronize_df['Herbs and Spices'] == True]['Ingredient Name'].tolist()
```

```
optimized_ingredients = []
for ingredient in best_ingredients:
    amount = round(compute_average_amount(ingredient, target_servings), 2)
    ingredient_unit_series = synchronize_df[synchronize_df['Ingredient Name'] == ingredient]['Unit']
    unit = ingredient_unit_series.iloc[0] if not ingredient_unit_series.empty else ' '

    if isinstance(unit, float) and math.isnan(unit):
        unit = ' '

    optimized_ingredients.append({
        'amount': amount,
        'unit': unit,
        'ingredient': ingredient
    })

optimized_ingredients = merge_ingredient_variations(optimized_ingredients)
for item in optimized_ingredients:
    print(item)
```

```
print("\n*** Nguyên liệu tối ưu ***")
for ingredient in optimized_ingredients:
    print(ingredient)

# Tạo hướng dẫn nấu ăn bằng GA
start_time = time.time()
relevant_ingredients = best_ingredients
print(f"Thời gian chạy: {time.time() - start_time} giây")

start_time = time.time()
all_relevant_steps = extract_relevant_steps(relevant_ingredients, synchronize_df, recipes_instructions_df, food_entity_
print(f"Thời gian chạy ra các bước hướng dẫn liên quan: {time.time() - start_time} giây")

start_time = time.time()
best_directions = genetic_algorithm_directions(all_relevant_steps, relevant_ingredients, ngen=10, pop_size=30, cxpb=0.6
print(f"Thời gian chạy best_directions: {time.time() - start_time} giây")

#print("\n*** Hướng dẫn ***")
#for i, step in enumerate(best_directions, 1):
#    print(f"Bước {i}: {step}")

start_time = time.time()
refined_direction = refine_instructions_with_model(best_directions)
print(f"Thời gian chạy: {time.time() - start_time} giây")

# Hiển thị kết quả

print("\n*** Nguyên liệu ***")
for ingredient in optimized_ingredients:
    print(ingredient)

print("\n*** Hướng dẫn ***")
for i, step in enumerate(refined_direction, 1):
    print(f"Bước {i}: {step}")
```

```
nutrition = get_nutrition(optimized_ingredients)
```

In [74]:

```
if __name__ == "__main__":
    main()
```