BỘ GIÁO DỤC VÀ ĐÀO TẠO

ĐẠI HỌC KINH TẾ TP HỒ CHÍ MINH

TRƯỜNG CÔNG NGHỆ VÀ THIẾT KẾ

ĐỒ ÁN MÔN HỌC

ĐỀ TÀI:

GIẢI BÀI TOÁN KNAPSACK PROBLEM BẰNG GIẢI THUẬT DI TRUYỀN

Học phần: Trí Tuệ Nhân Tạo

Nhóm Sinh Viên:

1. Đặng Xuân Cường
2. Trần Thọ
3. Dương Khải Nghiêm
4. Huỳnh Trần Bảo Việt

Chuyên Ngành: KHOA HỌC MÁY TÍNH
Khóa: K49

Giảng Viên: TS. Đặng Ngọc Hoàng Thành

TP. Hồ Chí Minh, Ngày 1 tháng 12 năm 2024

MỤC LỤC

CHƯƠNG 1. TỔNG QUAN

1.1. Giới Thiệu Về Knapsack problem

Trong thực tế luôn tồn tại các bài toán có thể biểu diễn dưới dạng một tập hợp các thực thể mà mỗi thực thể đều có một giá trị riêng của nó từ đó chọn ra nhiều tập con sao cho tổng giá trị các thực thể là lớn nhất và đồng thời cũng đáp ứng các điều kiện được cho trước. Điều kiện phổ biến đó là mỗi một thực thể đều có kích thước nhất định và tổng các kích thước không vượt quá giới hạn cho trước. Bài toán đó được gọi là bài toán cái túi. Trong thực tế, bài toán cái túi cũng được mô tả trong tình huống một người đi du lịch phải lựa chọn để nhét các vật phẩm vào túi sao cho đạt được độ thoải mái nhất. Các bài toán tương tự cũng xuất hiện trong các tình huống kinh doanh, hay các lĩnh vực toán tổ hợp, lý thuyết độ phức tạp tính toán, mật mã học và toán ứng dụng. Việc giải quyết các bài toán hiệu quả có thể giúp doanh nghiệp tối ưu hóa các chi phí và tăng trưởng, đồng thời có thể áp dụng trong các tình huống thực tế nơi mà tài nguyên có hạn và cần được phân bổ hợp lý.

1.2. Phát Biểu Bài Toán

Bài toán cái túi 0-1 được phát biểu như sau:

Ta có một cái túi có giới hạn trọng lượng là W và một tập hợp gồm n các đồ vật mà mỗi một đồ vật i có trọng lượng wi và giá trị vi.

Với mục tiêu chọn ra một tập hợp các đồ vật sao cho thỏa mãn điều kiện:

Tổng trọng lượng của các đồ vật được chọn không vượt quá giới hạn W và tổng giá trị các đồ vật được chọn là lớn nhất.

Bài toán được mô tả dưới dạng toán học:

Cực đại hóa $i = \sum_{i=1}^{n} v_i . x_i$

Với điều kiện: $\sum_{i=1}^{n} w_i . x_i \leq W$

Kí hiệu:

n: số lượng đồ vật

W: giới hạn trọng lượng của cái túi

w_i: trọng lượng của mỗi đồ vật

v_i: giá trị của đồ vật

1.3. Một Số Hướng Tiếp Cận Giải Quyết Bài Toán

Để giải quyết bài toán, các nhà nghiên cứu và kỹ sư đã phát triển nhiều phương pháp tiếp cận khác nhau, bao gồm:

Phương pháp quy hoạch động (Dynamic Programming) là một trong những cách tiếp cận hiệu quả nhất để giải quyết bài toán Knapsack, đặc biệt là với bài toán Knapsack 0-1. Ý tưởng cơ bản của phương pháp này là xây dựng một bảng để lưu trữ các giá trị tối ưu của các bài toán con. Mỗi giá trị trong bảng đại diện cho giá trị tối ưu mà ta có thể đạt được với một lượng trọng lượng cụ thể.

Phương pháp tham lam (Greedy Algorithm) là một cách tiếp cận đơn giản và hiệu quả, nhưng chỉ phù hợp với một số dạng bài toán Knapsack đặc biệt, chẳng hạn như bài toán Knapsack chia nhỏ (Fractional Knapsack). Trong bài toán này, các đối tượng có thể được chia nhỏ và lấy theo từng phần thay vì phải chọn toàn bộ. Ý tưởng chính của phương pháp tham lam là ưu tiên lựa chọn các đối tượng có tỷ lệ giá trị trên trọng lượng cao nhất trước, sau đó tiếp tục xử lý các đối tượng còn lại.

Phương pháp quay lui (Backtracking) là một phương pháp tiếp cận tổng quát, trong đó ta thử tất cả các khả năng của bài toán và loại bỏ những nhánh không hợp lý. Với bài toán Knapsack, ta có thể thử lựa chọn hoặc không lựa chọn một đối tượng vào ba lô, sau đó tiếp tục đệ quy để kiểm tra tất cả các khả năng.

Phương pháp thuật toán di truyền (Genetic Algorithm) là một phương pháp tối ưu hóa dựa trên các nguyên lý di truyền và chọn lọc tự nhiên. Phương pháp này sử dụng các cá thể (hay còn gọi là chromosome) để đại diện cho các lời giải khả thi và tiến hành lai ghép, đột biến để tạo ra thế hệ mới với các lời giải tối ưu hơn.

Phương pháp tìm kiếm mở rộng (Branch and bound) là một kỹ thuật giải quyết các bài toán tối ưu hóa bằng cách phân chia không gian tìm kiếm thành các nhánh và loại bỏ những nhánh không khả thi thông qua các biên giới. Trong bài toán Knapsack, biên giới được tính toán để xác định liệu ta có thể bỏ qua một nhánh nếu giá trị tối đa có thể có từ nhánh đó không lớn hơn giá trị tốt nhất hiện tại.

Phương pháp thuật toán Hill-Climbing là một thuật toán tìm kiếm cục bộ, cải thiện lời giải hiện tại bằng cách duyệt qua các lời giải lân cận. Thuật toán bắt đầu với một giải pháp ngẫu nhiên hoặc đơn giản (như chọn các vật phẩm nhẹ nhất). Sau đó, trong mỗi vòng lặp, thuật toán tạo các giải pháp lân cận thông qua việc thêm, bớt hoặc thay thế vật phẩm, đánh giá giá trị của chúng và chọn giải pháp tốt nhất. Quá trình lặp kết thúc khi không còn giải pháp lân cận nào tốt hơn, tức là đạt đến cực trị cục bộ. Thuật toán dừng khi không còn sự cải thiện trong các lần tìm kiếm tiếp theo.

Bài toán **Knapsack** problem là một bài toán có tính ứng dụng cao trong thực tế. Mỗi hướng tiếp cận để giải quyết bài toán luôn bao gồm có ưu và nhược điểm riêng của nó. Một

số phương pháp tiếp cận cho thấy phù hợp với lời giải tối ưu như quy hoạch động hoặc tìm kiếm mở rộng. Trong đó, các thuật toán tham lam hay heuristics như thuật toán di truyền phù hợp hơn cho các bài toán lớn, đòi hỏi tốc độ và tính linh hoạt. Việc lựa chọn phương pháp tối ưu phụ thuộc vào quy mô bài toán, yêu cầu về thời gian và độ chính xác của bài toán.

CHƯƠNG 2. GIẢI THUẬT DI TRUYỀN

2.1. Giới Thiệu Về Giải Thuật Di Truyền

Giải thuật di truyền (Genetic Algorithm) là một phương pháp tối ưu hóa được mô phỏng dựa trên quá trình tiến hóa và tự thích nghi của quần thể sinh học, dựa theo Học thuyết Tiến hóa của *Charles Darwin*. Phương pháp này mô phỏng quá trình thích nghi tự nhiên nhằm tìm kiếm lời giải tối ưu cho một vấn đề phức tạp trong không gian tìm kiếm rộng lớn. Bằng cách áp dụng các quy tắc di truyền và đấu tranh sinh tồn, giải thuật di truyền tiến hành tối ưu hóa ngẫu nhiên qua nhiều thế hệ liên tiếp, giúp tìm ra các lời giải có chất lượng cao.

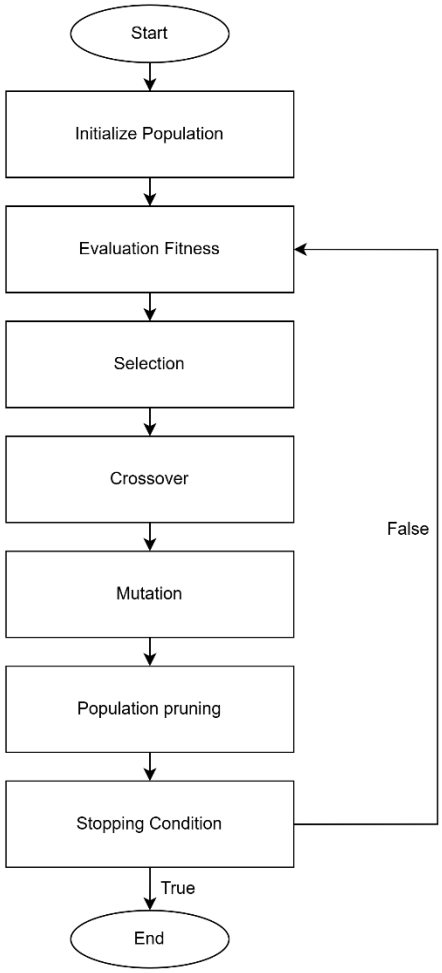
Giải thuật di truyền thường sử dụng các cá thể được biểu diễn dưới dạng chuỗi nhị phân chứa thông tin mã hóa tương tự như gene trong sinh học, đóng vai trò như các nhiễm sắc thể. Thuật toán hoạt động dựa trên các nguyên tắc sinh học như sau:

- Mỗi cá thể trong quần thể cạnh tranh về tài nguyên và cơ hội sinh sản.
- Cá thể khỏe mạnh hơn có khả năng sinh sản cao hơn, nhờ đó gene của chúng có cơ hội lan truyền rộng rãi qua thế hệ sau.
- Các cặp bố mẹ có gene tốt có thể sinh ra thế hệ con cái với các đặc điểm vượt trội, thậm chí tốt hơn cả bố mẹ.
- Qua từng thế hệ, quần thể tiến hóa để ngày càng thích nghi tốt hơn với môi trường của nó.

Giải thuật di truyền có ứng dụng rộng rãi, trong đó có bài toán Knapsack (bài toán Balo) một bài toán kinh điển trong toán học và khoa học máy tính. Sẽ được trình bày phần dưới.

2.2. Ứng Dụng Giải Thuật Di Truyền Cho Bài Toán Knapsack problem

Bài toán knapsack là một bài toán tối ưu tổ hợp kinh điển, làm sao để lựa chọn số vật phẩm có sẵn để trọng lượng không vượt quá sức chứa cái túi và giá trị của các vật phẩm được chọn phải lớn nhất. Với những giải pháp không gian lớn thì các phương pháp đơn giản tỏ ra thiếu hiệu quả trong việc đưa ra kết quả tốt nhất. Giải thuật Genetic Algorithm được sử dụng như một sự lựa chọn hiệu quả trong việc tìm lời giải gần tối ưu, ngay cả với những không gian giải pháp lớn.



Hình 2.2.1: Sơ đồ tổng quan về giải thuật di truyền.

Trước khi tìm hiểu về các chức năng và nhiệm vụ trong giải thuật di truyền ta sẽ đi xem xét về các thông thiết lập trước khi sử dụng và giải bài toán.

2.2.1. Các tham số cho giải thuật di truyền

1. Kích thước quần thể (Population Size)

- **Ý nghĩa:** Đây là số lượng cá thể (solutions) có mặt trong mỗi thế hệ. Mỗi cá thể đại diện cho một lời giải khả dĩ của bài toán (một tổ hợp các vật phẩm được chọn).
- **Thiết lập:** Bạn có thể chọn bất kỳ giá trị nào, thường là từ **20 đến 200** cá thể, tùy thuộc vào kích thước không gian giải pháp và tài nguyên tính toán.
    - **Lớn:** Cải thiện sự đa dạng, nhưng tốn tài nguyên và thời gian tính toán.
    - **Nhỏ:** Tối ưu hóa nhanh hơn, nhưng có thể dẫn đến hội tụ sớm.

2. Số thế hệ (Number of Generations)

- **Ý nghĩa:** Đây là số vòng lặp mà thuật toán sẽ thực hiện trước khi dừng. Mỗi thế hệ gồm các bước: chọn lọc, lai ghép, đột biến và tính toán lại điểm số thích nghi.

- **Thiết lập:** Bạn cần xác định số thế hệ trước khi thuật toán dừng lại, thường là vài trăm đến vài nghìn thế hệ.
    - **Lớn:** Cho phép thuật toán khám phá không gian giải pháp nhiều hơn.
    - **Nhỏ:** Có thể dừng sớm, chưa đạt được lời giải tốt nhất.

3. Tỷ lệ lai ghép (Crossover Rate)

- **Ý nghĩa:** Tỷ lệ lai ghép xác định phần trăm cá thể trong quần thể sẽ được lai ghép để tạo ra cá thể con trong mỗi thế hệ.
- **Thiết lập:** Giá trị thường nằm trong khoảng **0.7 đến 0.9**.
    - **Cao:** Nhiều cá thể sẽ được lai ghép, tăng khả năng khám phá không gian giải pháp.
    - **Thấp:** Quần thể có xu hướng giữ lại các cá thể hiện tại, ít biến đổi.

4. Tỷ lệ đột biến (Mutation Rate)

- **Ý nghĩa:** Tỷ lệ đột biến xác định xác suất một bit trong chuỗi nhị phân của mỗi cá thể bị thay đổi (từ 0 thành 1 hoặc từ 1 thành 0).
- **Thiết lập:** Thường là một số rất nhỏ, từ **0.01 đến 0.05** (1% đến 5%).
    - **Cao:** Làm tăng sự đa dạng, nhưng có thể phá hỏng các giải pháp tốt.
    - **Thấp:** Giúp duy trì các giải pháp tốt, nhưng có nguy cơ hội tụ sớm vào giải pháp không tối ưu.

5. Phương pháp chọn lọc (Selection Method)

- **Ý nghĩa:** Đây là cách chọn cá thể cha mẹ cho quá trình lai ghép. Bạn có thể chọn một hoặc kết hợp nhiều phương pháp chọn lọc khác nhau.
- **Thiết lập:** Có nhiều phương pháp chọn lọc.

6. Phương pháp lai ghép (Crossover Method)

- **Ý nghĩa**: Crossover là quá trình kết hợp các đặc điểm từ hai cá thể cha mẹ để tạo ra cá thể con. Phương pháp lai ghép xác định cách thức kết hợp thông tin từ cha mẹ để tạo ra thế hệ con cháu.
- **Thiết lập:** Có nhiều phương pháp lai ghép.

7. Phương pháp đột biến (Mutation Method)

- **Ý nghĩa**: Đột biến là quá trình thay đổi ngẫu nhiên một phần trong chuỗi nhị phân của cá thể con (chẳng hạn như chuyển một bit từ 0 thành 1 hoặc ngược lại). Đột biến giúp tạo ra sự đa dạng và tránh quá trình hội tụ sớm vào một giải pháp không tối ưu.
- **Thiết lập:** Có nhiều phương pháp đột biến.

8. Điều kiện dừng (Stopping Condition)

- **Ý nghĩa:** Điều kiện để dừng thuật toán, không nhất thiết phải dựa vào số thế hệ.

– **Thiết lập:** Đạt được một giá trị thích nghi tối ưu.

```
POPULATION_SIZE = 200  # Kích thước quần thể
GENERATIONS = 200  # Số thế hệ
CROSSOVER_RATE = 0.8  # Tỷ lệ lai ghép
MUTATION_RATE = 0.04  # Tỷ lệ đột biến
```

Hình 2.2.1: Các thông số quan trọng trong thuật toán di truyền.

2.2.2. Mô tả

Định nghĩa chuỗi nhị phân:

– 0: không chọn
– 1: chọn

Bước 1: Khởi tạo quần thể ban đầu. (Initialization)

Mục đích:

– **Đa dạng hóa giải pháp ban đầu**: Việc khởi tạo ngẫu nhiên giúp tạo ra nhiều giải pháp khác nhau ngay từ đầu, tránh việc toàn bộ quần thể tập trung vào một số giải pháp cụ thể. Điều này giúp đảm bảo sự đa dạng trong quá trình tiến hóa, từ đó tăng khả năng tìm ra lời giải tốt nhất.
– **Tránh rơi vào cực trị cục bộ**: Nếu tất cả các cá thể ban đầu đều giống nhau hoặc rất gần nhau, thuật toán có thể nhanh chóng rơi vào cực trị cục bộ, khiến quá trình tìm kiếm dừng lại mà chưa tìm ra giải pháp tối ưu toàn cục.

Xử lí ngoại lệ:

– Weight > Maximum capacity.

Ví dụ:

Cho bài toán Knapsack 0/1 với đề bài như sau

Item	Weight	Value
1	1	2
2	2	8
3	4	12
4	5	10

Maximum capacity = 10 (Balo có sức chứa tối đa là 10).

```
items = [] # Vật phẩm chứa weights và values
max_capacity = 0 # maximum capacity
fitness_history = [] # Chứa lịch sử fitness qua các thế hệ
```

Hình 2.2.2: Các giá trị dữ liệu của bài toán.

Xác định được dữ liệu bài toán từ bảng trên ta triển khai thuật toán.

Khởi tạo với quần thể **ngẫu nhiên**. (Ví dụ kích thước quần thể là 5)

Ví dụ: Khởi tạo với quần thể ban đầu là 5.

Cá thể	Chuỗi nhị phân	Weight	Value
1	1001	6	12
2	1101	8	20
3	1100	3	10
4	1010	5	14
5	0111	11	20

```
def initialize_population(num_items):
    return [[random.randint(0, 1) for _ in range(num_items)] for _ in range(POPULATION_SIZE)]
```

Hình 2.2.3: Hàm khởi tạo quần thể.

Bước 2: Đánh giá. (Evaluation)

1. Định nghĩa điểm số thích nghi:

- Điểm số thích nghi (fitness score) của mỗi cá thể được tính dựa trên tổng giá trị của các vật phẩm được chọn, miễn là tổng trọng lượng không vượt quá sức chứa tối đa của ba lô. (weight <= maximum capacity)
- Nếu tổng trọng lượng vượt quá sức chứa, điểm số thích nghi có thể được thiết lập thấp (ví dụ, bằng 0) để đánh giá cá thể đó không khả thi.

2. Cách tính toán:

- Với mỗi cá thể, xác định các vật phẩm nào được chọn (dựa trên chuỗi nhị phân).
- Tính tổng trọng lượng và tổng giá trị của các vật phẩm được chọn.
- So sánh tổng trọng lượng với sức chứa tối đa. Nếu không vượt quá, ghi nhận tổng giá trị là điểm số thích nghi. Nếu vượt quá, gán điểm số fitness score = 0

Cá thể được lấy theo thứ tự từ Bước 1

Cá thể	Chuỗi nhị phân	Item được chọn	Weight	Value	Fitness Score
1	1001	1,4	6	12	12
2	1101	1,2,4	8	20	20
3	1100	1,2	3	10	10
4	1010	1,3	5	14	14
5	0111	2,3,4	11	20	0

```
def fitness(individual):
    total_weight = sum(individual[i] * items[i][0] for i in range(len(items)))
    total_value = sum(individual[i] * items[i][1] for i in range(len(items)))
    return total_value if total_weight <= max_capacity else 0
```

Hình 2.2.4: Hàm đánh giá fitness.

Bước 3: Chọn lọc. (Selection)

Ta sẽ sử dụng một phương án chọn lọc để thực hiện.

Note:

Sử dụng chọn lọc ưu tú. **(Elitism Selection)**

Số lượng cá thể được chọn:

- Bằng với kích thước quần thể hiện tại.
- Trong ví dụ: 5 cá thể.

Đặc điểm quan trọng:

- Cá thể có thể được chọn nhiều lần.
- Cá thể có fitness score cao có cơ hội được chọn nhiều lần hơn.

Kết quả:

- Danh sách các cá thể cha mẹ tiềm năng cho quá trình lai ghép tiếp theo.
- Số lượng cá thể trong danh sách này bằng với kích thước quần thể.

Lưu ý:

- Không loại bỏ bất kỳ cá thể nào khỏi quá trình chọn lọc.
- Việc một cá thể được chọn nhiều lần là bình thường.
- Danh sách cá thể được chọn không nhất thiết phải gồm tất cả các cá thể khác nhau từ quần thể ban đầu. (Tức là có sự trùng lặp)

Chọn lọc ưu tú (Elitism Selection): Trong mỗi thế hệ, giữ lại ít nhất một cá thể có điểm số *fitness* cao nhất. Điều này giúp đảm bảo rằng giải pháp tốt nhất hiện tại không bị mất đi.

Thực hiện chọn lọc:

Danh sách ban đầu:

Cá thể	Chuỗi nhị phân	Item được chọn	Weight	Value	Fitness Score
1	1001	1,4	6	12	12
2	1101	1,2,4	8	20	20
3	1100	1,2	3	10	10
4	1010	1,3	5	14	14
5	0111	2,3,4	11	20	0

Danh sách sau khi chọn lọc:

- **Cá thể 2** (1101, fitness score = 20): Được chọn đầu tiên do có điểm cao nhất.
- **Cá thể 4** (1010, fitness score = 14): Là cá thể có điểm fitness cao thứ hai sau cá thể 2.
- **Cá thể 1** (1001, fitness score = 12): Chọn tiếp cá thể có điểm cao tiếp theo.
- **Cá thể 3** (1100, fitness score = 10): Tiếp tục chọn cá thể có fitness cao tiếp theo để đảm bảo sự đa dạng.
- **Cá thể 5** (0111, fitness score = 20): chọn tiếp cá thể 5.

Kết quả:

Vòng	Cá thể được chọn	Fitness score	Cách thức
1	2	20	Chọn lọc ưu tú
2	4	14	Chọn theo thứ tự giảm dần fitness
3	1	12	Chọn theo thứ tự giảm dần fitness
4	3	10	Chọn theo thứ tự giảm dần fitness
5	5	0	Chọn theo thứ tự giảm dần fitness

```
def select_population(population):
    sorted_population = sorted(population, key=lambda x: fitness(x), reverse=True)
    return sorted_population[:POPULATION_SIZE]
```

Hình 2.2.5: Hàm chọn lọc.

Bước 4: Lai ghép. (Crossover)

Sử dụng một phương án lai ghép cho giải thuật.

Mục đích:

- Kết hợp đặc điểm: Lai ghép giúp kết hợp các đặc điểm tốt từ các cá thể cha mẹ, từ đó tạo ra các cá thể con có khả năng tốt hơn.

- Tăng cường sự đa dạng: Việc lai ghép có thể tạo ra các tổ hợp mới từ các cá thể cha mẹ, giúp duy trì sự đa dạng trong quần thể.

Note:

- Sử dụng **Uniform crossover.** (Thay đổi tuỳ vào mục đích người sử dụng)
- Chúng ta sẽ chọn từng gen của cá thể ngẫu nhiên từ cha hoặc mẹ và thực hiện Uniform crossover cho từng cặp.
- Số lượng cá thể được chọn để lai ghép: (Số lượng cá thể) x (Tỷ lệ lai ghép).
- Số lượng cặp cha mẹ: (Số lượng cá thể được chọn để lai ghép) / 2.
- **Tỷ lệ lai ghép** = 80% = 0.8 (Thiết lập).

Thực hiện: Tuân thủ điều kiện ở mục Note.

Danh sách cá thể cha mẹ tiềm năng sau bước 3 (Elitism Selection).

Đầu tiên, thiết lập tỷ lệ lai ghép lấy 80% tổng số cá thể gốc.

Cá thể	Chuỗi nhị phân	Fitness Score
1	1001	12
2	1101	20
3	1100	10
4	1010	14

Sử dụng Uniform Crossover (phương pháp tung đồng xu)**:**

- Random(0,1) ngẫu nhiên cho hai số 0,1. (Như hai mặt úp, ngửa)
- Nếu ra 0 thì chọn gen mẹ, 1 thì chọn gen cha cho mỗi bit (Có thể ngược lại).

Cặp cha mẹ 1 :
Cha : 1001
Mẹ : 1101

Thực hiện Uniform Crossover:

- **Con 1 :**
  [Gen 1: Chọn từ cha (1), Gen 2: Chọn từ mẹ (1), Gen 3: Chọn từ cha (0), Gen 4: Chọn từ mẹ (1)]
  Kết quả con 1: 1101
- **Con 2 :**
  [Gen 1: Chọn từ mẹ (1), Gen 2: Chọn từ cha (0), Gen 3: Chọn từ cha (0), Gen 4: Chọn từ mẹ (1)]
  Kết quả con 2: 1001

Cặp cha mẹ 2 :
Cha : 1100
Mẹ : 1010

Thực hiện Uniform Crossover:

- **Con 3:**
  [Gen 1: Chọn từ cha (1), Gen 2: Chọn từ mẹ (0), Gen 3: Chọn từ cha (0), Gen 4: Chọn từ mẹ (0)]
  Kết quả Con 3: 1000
- **Con 4:**
  [Gen 1: Chọn từ mẹ (1), Gen 2: Chọn từ cha (1), Gen 3: Chọn từ mẹ (1), Gen 4: Chọn từ cha (0)]
  Kết quả Con 4 : 1110

Từ *Kết Quả* trên ta thu được các cá thể con là:

Con	Chuỗi nhị phân
1	1101
2	1001
3	1000
4	1110

Thực hiện tính toán fitness score cho các cá thể con:

Cá thể	Chuỗi nhị phân	Item được chọn	Weight	Value	Fitness Score
1	1101	1, 2, 4	8	12	12
2	1001	1, 4	6	12	10
3	1000	1	1	2	2
4	1110	1, 2, 3	7	22	22

Xử lý ngoại lệ: Nếu weight > Maximum capacity => fitness score = 0.

```python
def crossover(parent1, parent2):
    child1, child2 = parent1[:], parent2[:]
    for i in range(len(parent1)):
        coin_flip = random.choice([0, 1])
        if coin_flip == 0:
            child1[i] = parent1[i]
            child2[i] = parent2[i]
        else:
            child1[i] = parent2[i]
            child2[i] = parent1[i]
    return child1, child2
```

Hình 2.2.6: Hàm lai ghép.

Bước 5: Mutate. (Đột biến)

Sử dụng đột biến **Bit-Flip Mutation.** (Thay đổi tuỳ vào mục đích người sử dụng)

Tỷ lệ đột biến (Mutation Rate) = 0.04 (Ví dụ cho thông số bằng 0.04)

Mục đích của đột biến:

- Duy trì sự đa dạng di truyền trong quần thể.
- Ngăn chặn sự hội tụ sớm vào các cực trị cục bộ.
- Khám phá các vùng mới trong không gian tìm kiếm.

Quy trình đột biến

- Duyệt qua từng bit trong mỗi chuỗi nhị phân của các cá thể con sau khi lai ghép
- Với mỗi bit, tạo một số ngẫu nhiên từ 0 đến 1
- Nếu số ngẫu nhiên nhỏ hơn hoặc bằng tỷ lệ đột biến, thực hiện đột biến trên bit đó
- Đột biến được thực hiện bằng cách đảo ngược giá trị của bit (0 thành 1 hoặc 1 thành 0)

Cách hoạt động:

Với mỗi bit trong cá thể (Duyệt qua từng cá thể con => duyệt qua từng bit của cá thể con => random bộ số **ngẫu nhiên**), nếu một số **ngẫu nhiên** nhỏ hơn tỷ lệ đột biến, bit đó sẽ được thay đổi từ '0' thành '1' hoặc ngược lại.

Con	Trước	Sau	Item	Weight	Value	Fitness score
1	1101	1101	1, 2, 4	8	20	20
2	1001	1111	1, 2, 3 ,4	12	32	0
3	1000	1001	1, 4	6	12	12
4	1110	1011	1, 3, 4	10	24	24

Lưu ý:

- Ngẫu nhiên bộ tạo số tốt để có kết quả chính xác.
- Xử lí ngoại lệ: Weight > Maximum capacity => gán fitness score = 0.

```
def mutate(individual):
    for i in range(len(individual)):
        if random.random() < MUTATION_RATE:
            individual[i] = 1 - individual[i]
    return individual
```

Hình 2.2.7: Hàm đột biến.

Bước 6: Cắt tỉa quần thể. (Population pruning)

- Các cá thể con mới được tạo ra từ quá trình lai ghép và đột biến sẽ được thêm vào danh sách quần thể ban đầu.
- Kiểm tra kích thước quần thể: Nếu tổng số lượng cá thể (bao gồm cả cá thể cũ và cá thể con) vượt quá kích thước quần thể thiết lập ban đầu, tiến hành loại bỏ các cá thể.
- Loại bỏ các cá thể có fitness score thấp:
    - Sắp xếp theo thứ tự điểm fitness score giảm dần.
    - Loại bỏ cho đến khi kích thước quần thể bằng kích thước quần thể được khởi tạo ban đầu.

Như ví dụ: Số lượng cá thể cần loại bỏ = Tổng cá thể hiện tại - Kích thước cá thể thiết lập ban đầu.

```
# Cắt tỉa quần thể để đảm bảo kích thước không vượt quá POPULATION_SIZE
population.extend(offspring)
population = sorted(population, key=lambda ind: fitness(ind), reverse=True)
population = population[:POPULATION_SIZE]
```

Hình 2.2.8: Cắt tỉa quần thể.

Bước 7: Kiểm tra điều kiện dừng. (Stopping Condition)

Sau khi đạt số thế hệ tối đa = **Số thế hệ quần thể** thì dừng thuật toán và ghi lại:

- Lời giải tốt nhất (cá thể có điểm fitness score cao nhất).
- Tổng số thế hệ đã chạy.
- Giá trị fitness tốt nhất đạt được.

Nếu không, quay lại **Bước 3** và thực hiện lại các bước từ Bước 3 đến Bước 6. (Vòng lặp có giá trị = số thế hệ tối đa)

```python
def genetic_algorithm():
    global fitness_history

    population = initialize_population(len(items))
    best_individual = max(population, key=lambda ind: fitness(ind))

    for gen in range(GENERATIONS):
        selected_population = select_population(population)

        num_to_crossover = int(POPULATION_SIZE * CROSSOVER_RATE)
        selected_for_crossover = selected_population[:num_to_crossover]

        offspring = []
        for i in range(0, num_to_crossover, 2):
            if i + 1 < len(selected_for_crossover):
                child1, child2 = crossover(selected_for_crossover[i], selected_for_crossover[i + 1])
                offspring.extend([child1, child2])

        offspring = [mutate(ind) for ind in offspring]

        # Cắt tỉa quần thể để đảm bảo kích thước không vượt quá POPULATION_SIZE
        population.extend(offspring)
        population = sorted(population, key=lambda ind: fitness(ind), reverse=True)
        population = population[:POPULATION_SIZE]

        current_best = max(population, key=lambda ind: fitness(ind))
        fitness_history.append(fitness(current_best))

        if fitness(current_best) > fitness(best_individual):
            best_individual = current_best

    return best_individual, fitness(best_individual)
```

Hình 2.2.9: Hàm quy trình thực hiện giải thuật di truyền.

CHƯƠNG 3. CÁC KẾT QUẢ THỰC NGHIỆM

3.1. Các Tình Huống

Trong giải thuật Genetic Algorithm, các thông số quan trọng và kỹ thuật tối ưu hóa đóng vai trò then chốt trong việc tìm kiếm giải pháp tốt nhất. Để đạt được kết quả tối ưu, chúng ta cần thử nghiệm 6 trường hợp các thông số với kỹ thuật để lựa chọn ra giải pháp phù hợp. Quá trình thử nghiệm này đòi hỏi sự tỉ mỉ và khả năng quan sát, phân tích kết quả để chọn ra sự kết hợp tối ưu của các thông số và kỹ thuật phù hợp nhất với đặc điểm bài toán.

3.1.1. Thông số thực nghiệm

Trong giải thuật genetic thì có 4 thông số quan trọng với hiệu suất của thuật toán. Mỗi thông số được thử nghiệm với các giá trị khác nhau để đánh giá tác động của chúng đến các chỉ số hiệu suất, từ đó so sánh hiệu quả giữa các sự thay đổi của các thông số này.

Thông số	Giá trị			
Kích thước quần thể	50	100	200	500
Số thế hệ	50	100	200	500

Thông số	Giá trị		
Tỉ lệ lai ghép	0.7	0.8	0.9
Tỉ lệ đột biến	0.02	0.03	0.04

3.1.2. Phương pháp sử dụng

Trong quá trình thực hiện thuật toán di truyền, các phương pháp như *chọn lọc, lai ghép và đột biến* đóng vai trò quan trọng trong việc tối ưu hóa hiệu suất. Việc thử nghiệm các phương pháp này nhằm đánh giá sự ảnh hưởng của từng kỹ thuật đến hiệu quả tổng thể của thuật toán.

Toán tử di truyền	Phương pháp
Selection	Elitism Selection Tournament Selection Roulette Wheel Selection Rank Selection
Crossover	One-point crossover Muti-point crossover Uniform crossover Arithmetic Crossover

Mutation	Bit-flip mutation Swap Mutation Scramble Mutation Gaussian Mutation

3.1.3. Kịch bản thực nghiệm

Cấu hình tiêu chuẩn:

Thông số:

- Kích thước quần thể: 200
- Số thế hệ: 200
- Tỉ lệ lai ghép: 0.8
- Tỉ lệ đột biến: 0.04

Toán tử di truyền:

- Phương pháp chọn lọc: Elitism Selection
- Phương pháp lai ghép: One-Point Crossover
- Phương pháp đột biến: Bit-flip Mutation

Phương pháp tiếp cận:

- Đánh giá chi tiết vai trò của từng thông số và kỹ thuật trong việc tối ưu hóa.
- Cách tiếp cận này đảm bảo rằng hiểu rõ tác động của từng yếu tố trong khi các yếu tố khác được giữ cố định.
- Xác định được các thông số và kỹ thuật có khả năng mang lại kết quả tối ưu sau khi kết hợp các thống số và kỹ thuật đã được tối ưu riêng lẻ.

Các kịch bản kiểm thử với dữ liệu đầu vào:

Testcase 1:

- Weights: [4, 2, 5, 9, 10, 14, 16, 18, 13, 19, 4, 1, 3, 7]
- Values: [2, 4, 6, 9, 7, 11, 12, 4, 8, 10, 2, 8, 12, 9]
- Max capacity: 40

Testcase 2:

- Weights: [4, 2, 5, 9, 10, 14, 16, 18, 13, 19, 4, 1, 3, 7, 8]
- Values: [2, 4, 6, 9, 7, 11, 12, 4, 8, 10, 2, 8, 12, 9, 14]
- Max capacity: 45

Testcase 3:

- Weights: [45, 78, 123, 237, 56, 111, 89, 232, 344, 190, 70, 250, 135, 376, 92, 187, 315, 220, 180, 260]
- Values: [98, 210, 155, 345, 67, 122, 134, 89, 450, 170, 77, 190, 280, 300, 113, 265, 190, 158, 99, 340]
- Max Capacity: 2393

Testcase 4:

- Weights: [125, 305, 78, 432, 217, 150, 590, 238, 660, 495, 312, 276, 132, 476, 189, 640, 72, 810, 385, 159, 250, 190, 360, 540, 290, 458, 634, 710, 430, 525]
- Values: [210, 305, 135, 445, 275, 320, 150, 220, 490, 399, 318, 525, 410, 340, 230, 465, 110, 525, 285, 210, 190, 175, 345, 250, 330, 510, 470, 455, 290, 375]
- Max Capacity: 7418

Testcase 5:

- Weights: [879, 3946, 3736, 2324, 2104, 1971, 3252, 3712, 2493, 2780, 1350, 1463, 3868, 2466, 191, 1960, 2347, 343, 2348, 2902, 2840, 3047, 77, 2390, 1454, 153, 3741, 1872, 370, 3658, 2525, 1488, 261, 1755, 899, 3782, 1147, 455, 3858, 2640, 2493]
- Values: [3083, 836, 3615, 3391, 1409, 3999, 1829, 1216, 3220, 278, 774, 3602, 158, 761, 3771, 288, 797, 1953, 3947, 2882, 3993, 3179, 1353, 654, 3481, 3914, 543, 709, 2035, 3998, 2837, 2678, 456, 691, 2941, 800, 1804, 2817, 3910, 2594, 953]
- Max Capacity: 50000

Testcase 6:

- Weights: [330, 176, 2431, 2808, 1038, 3996, 3193, 805, 627, 2257, 1649, 1210, 2642, 508, 1192, 3446, 3626, 1706, 3068, 2977, 1688, 709, 3522, 85, 3350, 451, 2430, 151, 2696, 922, 3930, 323, 1817, 184, 1043, 2839, 3439, 484, 2432, 1783, 622]
- Values: [2870, 308, 570, 137, 445, 771, 108, 469, 3678, 2760, 842, 1237, 1829, 967, 313, 1322, 651, 679, 2254, 525, 1571, 697, 2262, 3065, 3059, 2753, 2264, 1820, 47, 2845, 339, 2294, 729, 2603, 1720, 3385, 1854, 2539, 762, 2005, 201]
- Max Capacity: 35000

Cấu hình cơ sở cấu tiêu chuẩn làm cơ sở so sánh. Trong mỗi kịch bản trường hợp kiểm thử, một thông số sẽ được thay đổi, trong khi các thông số còn lại được giữ nguyên theo cấu hình tiêu chuẩn.

3.1.4. Tiêu chí đánh giá kết quả thí nghiệm

Để đánh giá hiệu suất của bài toán thì ta sử dụng các phương pháp thống kê với các bộ chỉ số về hiệu suất như:

$$\text{std} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(f_i - \bar{f})^2}$$

- f_i: fitness của cá thể thứ i.
- $\bar{f}$: trung bình fitness của quần thể.
- n: số cá thể trong quần thể.

1. Fitness cuối cùng

Mục đích: Đo lường hiệu quả của thuật toán khi tìm kiếm giá trị tối ưu.

Đánh giá:

- Giá trị cao: Thuật toán tìm được lời giải tốt.
- So sánh giữa các thí nghiệm:
  - Tìm trung bình fitness cuối cùng qua nhiều lần chạy.
  - Xác định thuật toán/quy mô quần thể/quy tắc vận hành mang lại fitness cao nhất.

2. Tốc độ hội tụ

Mục đích: Đo lường thời gian (hoặc số thế hệ) để đạt được giải pháp tối ưu.

Đánh giá:

- Tốc độ nhanh (ít thế hệ):
  - Quần thể có kích thước nhỏ có xu hướng hội tụ nhanh nhưng dễ mất đa dạng.
  - Tăng tốc độ tính toán.
- Tốc độ chậm (nhiều thế hệ):
  - Quần thể lớn duy trì sự đa dạng lâu hơn, nhưng mất nhiều thời gian để hội tụ.
  - Có thể cần tối ưu các thông số (chọn lọc, lai ghép, đột biến).

3. Độ lệch chuẩn (Standard Deviation):

- Để đánh giá độ ổn định của thuật toán. Sẽ là một phương pháp thống kê hiệu quả để đo lường mức độ phân tán cũng như biến động của một tập dữ liệu. Standard Deviation ở đây được dùng để đo lường sự phân tán của giá trị Fitness trong quần thể.
- **Độ lệch chuẩn thấp** (std lớn):
  - Fitness các cá thể trong quần thể có sự khác biệt lớn thì thuật toán vẫn đang khám phá.
  - Quá trình tìm kiếm có thể chưa hội tụ hoặc quần thể chưa đồng nhất.
- **Độ lệch chuẩn cao** (std nhỏ):

- Các cá thể trong quần thể có giá trị fitness gần nhau thì thuật toán đã hội tụ.
- Quá trình hội tụ có thể ổn định nhưng dễ bị mắc kẹt ở cực trị cục bộ nếu std quá nhỏ.

4. Fitness trung bình (Mean Fitness):

- Fitness trung bình để đánh giá quá trình tiến hóa của các cá thể qua các thế hệ như:
  - Đo lường sự tiến bộ của quần thể: Sự tăng lên của Mean Fitness cho thấy xu hướng phát triển của các quần thể đang ngày càng trở nên tốt hơn và các giải pháp kém dần bị loại bỏ
  - Phát hiện hội tụ: Khi Mean Fitness tiến gần đến Best Fitness và giá trị của Mean Fitness không tăng điều này cho thấy dấu hiệu hội tụ sớm của thuật toán tức là các quần thể bị mắc kẹt tại cực trị địa phương (local optinum)
  - Đánh giá sự đa dạng của quần thể: Nếu khoảng cách giữa Mean Fitness và Best Fitness lớn sẽ thể hiện quần thể tồn tại nhiều cá thể kém hơn và vẫn còn độ đa dạng. Nếu khoảng cách nhỏ thậm chí trùng nhau điều này thể hiện quần thể đang mất đi sự đa dạng và các cá thể đang trở nên đồng nhất.
  - Tinh chỉnh tham số: cũng có thể dựa vào khoảng cách giữa Mean Fitness và Best Fitness để tinh chỉnh tham số.

Việc sử dụng các phương pháp thống kê sẽ là một cách hiệu quả để đánh giá hiệu suất của thuật toán. Là nền tảng để chọn ra các toán tử cũng như điều chỉnh các tham số sao cho phù hợp với bài toán.

3.2. Phân Tích và Đánh Giá

Qua thử nghiệm với các thông số trên ta thấy được kết quả cho ra được fitness tối ưu nhất. Fitness của các thế hệ tăng dần do ảnh hưởng của Elitism và dần ổn định trở về sau khi đạt được kết quả tối ưu. Tuy nhiên vẫn còn những hạn chế tiềm ẩn khi áp dụng giải thuật với các bài toán khác sau này. Khi giải bài toán Knapsack 0/1 bằng giải thuật Genetic Algorithm tuy đạt được giải pháp tối ưu nhưng kết quả vẫn gặp vấn đề về hội tụ sớm cũng như thiếu đa dạng quần thể điều này sẽ làm giảm khả năng khám phá trong không gian giải pháp rộng lớn dẫn đến bài toán chỉ tìm được giải pháp tối ưu cục bộ thay vì tối ưu toàn cục.

3.2.1. Ảnh hưởng của các thông số

Với bộ thông số cấu hình tiêu chuẩn chỉ thay đổi mỗi thông số được thực nghiệm ta có được bộ kết quả sau số lần chạy nhất định để đánh giá được sự ảnh hưởng của thông số đó đến kết quả hiệu suất bài toán. Ngoài ra, để cho việc đánh giá được trực quan hơn thì sử dụng thêm biểu đồ với các chỉ số đánh giá để thấy rõ hơn sự thay đổi qua từng thế hệ.

1. Kích thước quần thể

Fitness cuối cùng:

– Quần thể có xu hướng tăng dần khi kích thước quần thể lớn hơn, đặc biệt rõ rệt trong các bài toán phức tạp như Testcase 5,6. Quần thể lớn giúp khám phá không gian tìm kiếm toàn diện hơn, giảm nguy cơ hội tụ vào các cực trị cục bộ.
– Tuy nhiên ở một số trường hợp, sự cải thiện không đáng kể khi kích thước quần thể vượt qua ngưỡng nhất định, cho thấy cần tối hoá kích thước.

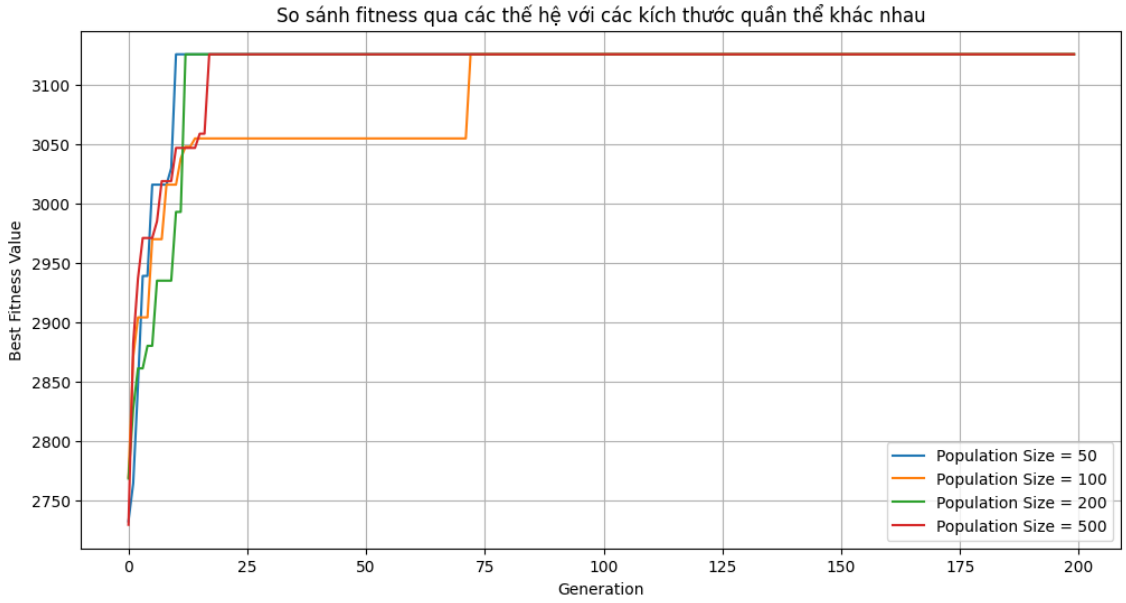
Tốc độ hội tụ:

– Quần thể nhỏ có xu hướng hội tụ nhanh hơn, với số thế hệ cần thiết ít hơn. Điều này phù hợp với các bài toán đơn giản hoặc yêu cầu kết quả nhanh chóng.
– Quần thể lớn thường ổn định, với độ lệch chuẩn giảm dần qua các thế hệ, do phải xử lí nhiều cá thể và khám phá không gian tìm kiếm rộng hơn. Tuy nhiên, tốc độ hội tụ chậm lại đi kèm việc đạt được giải pháp tốt hơn, ổn định hơn.

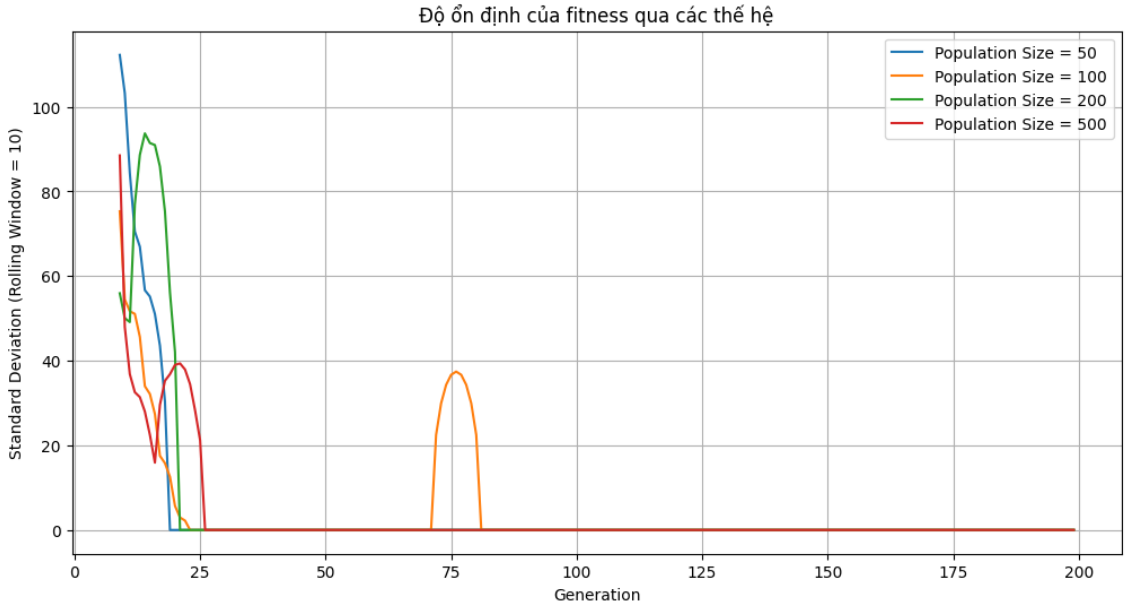
Độ lệch chuẩn:

– Quần thể nhỏ có độ ổn định thấp, dao động giữa các lần chạy, đặc biết ở các bài toán phức tạp. Một phần có thể do số lượng cá thể không đủ để duy trì sự đa dạng.
– Quần thể lớn thường ổn định, độ lệch chuẩn có xu hướng giảm dần qua các thế hệ, nhưng ở các bài toán phức tạp, dao động vẫn xảy ra.

Ngoài ra, để việc đánh giá trở nên trực quan hơn, các biểu đồ được sử dụng nhằm minh họa rõ ràng sự thay đổi của các chỉ số qua từng thế hệ.



Hình 3.2.1: Biểu đồ biểu diễn fitness qua các thế hệ với từng kích thước quần thể.



Hình 3.2.2: Biểu đồ biểu diễn sự thay đổi của độ ổn định (std) qua từng thế hệ.

Việc sử dụng các biểu đồ trong nghiên cứu giúp minh họa rõ ràng sự thay đổi của các chỉ số qua từng thế hệ, từ đó hỗ trợ đánh giá hiệu quả của từng cấu hình thử nghiệm. Các biểu đồ như **fitness qua các thế hệ** và **độ lệch chuẩn (std) qua các thế hệ** không chỉ giúp làm nổi bật sự cải thiện qua thời gian mà còn cung cấp cái nhìn trực quan về sự ổn định của quá trình hội tụ.

Qua thực nghiệm với thông số kích thước quần thể ta đánh giá được rằng, với các tham số được thí nghiệm trong ảnh hưởng của kích thước quần thể thì:

- **Quần thể nhỏ (50-100 cá thể)**: Thích hợp với các bài toán đơn giản hoặc yêu cầu thời gian xử lý ngắn, nhưng dễ bị hội tụ sớm và kém ổn định.
- **Quần thể lớn (200-500 cá thể)**: Phù hợp với các bài toán phức tạp, đạt được fitness cuối cùng tốt hơn và ổn định hơn, mặc dù tốc độ hội tụ chậm hơn.

Việc chọn kích thước quần thể cần dựa trên đặc thù của bài toán, khi cần cân nhắc giữa **tốc độ hội tụ** và **độ ổn định** để đạt được giải pháp tối ưu nhất. Quần thể nhỏ có thể cho kết quả nhanh nhưng không ổn định, trong khi quần thể lớn sẽ giúp cải thiện độ chính xác và ổn định nhưng với thời gian hội tụ dài hơn.

2. Số thế hệ

Fitness cuối cùng:

- Giá trị fitness cuối cùng trong hầu hết các trường hợp không thay đổi nhiều khi tăng số thế hệ (ngoại trừ một số ngoại lệ như Testcase 5 và Testcase 6). Điều

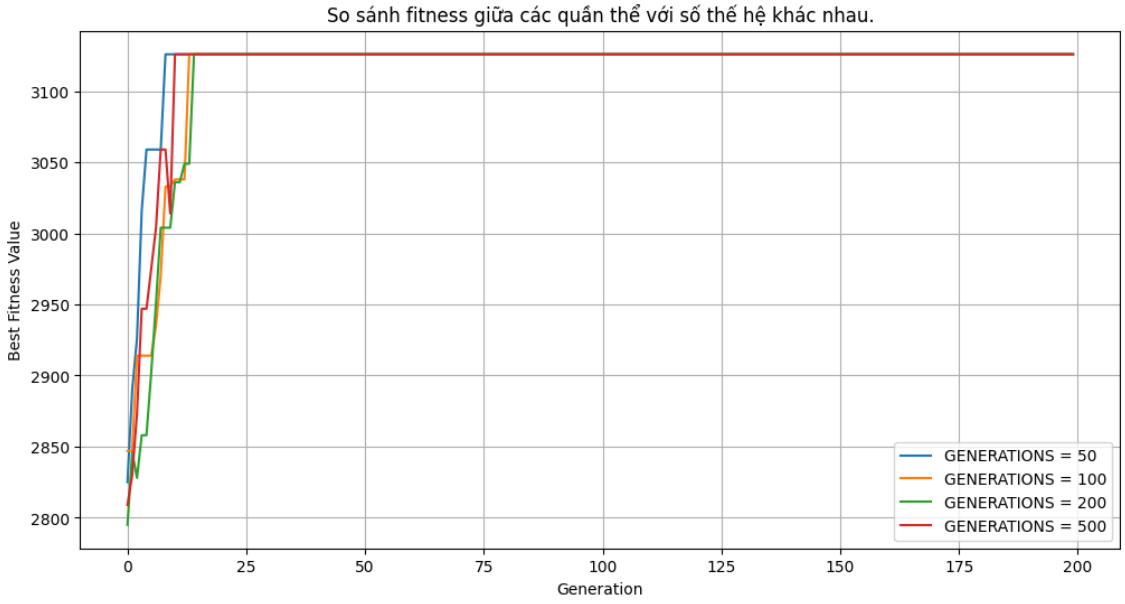
này chỉ ra rằng, việc tăng số thế hệ sau một ngưỡng nhất định (ví dụ, 500 thế hệ) không mang lại cải thiện đáng kể về kết quả tối ưu.
- Testcase 3 và Testcase 4 cho thấy sự khác biệt đáng kể về giá trị fitness cuối cùng khi tăng số thế hệ lên 200, có thể là do không gian tìm kiếm lớn hơn hoặc các điều kiện cụ thể của bài toán.

Tốc độ hội tụ:

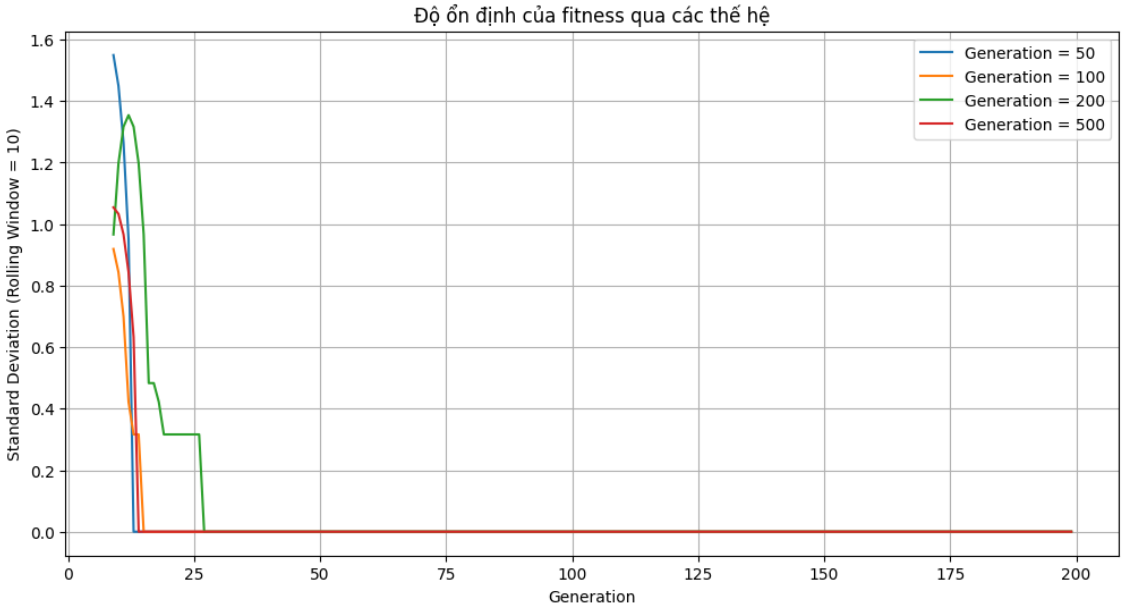
- Tốc độ hội tụ thường nhanh hơn khi số thế hệ tăng lên, đặc biệt là trong các testcase nhỏ (Testcase 1, 2, 3). Với Testcase lớn hơn (Testcase 5, 6), tốc độ hội tụ chậm dần, cho thấy vấn đề càng phức tạp thì cần nhiều thế hệ hơn để đạt đến trạng thái tối ưu cục bộ hoặc toàn cục.
- Trong các testcase lớn (Testcase 5, 6), tốc độ hội tụ không cải thiện rõ rệt dù số thế hệ tăng, có thể do ảnh hưởng của không gian tìm kiếm phức tạp.

Độ ổn định (Standard Deviation - STD):

- Độ ổn định không cải thiện đáng kể khi tăng số thế hệ, thậm chí có xu hướng dao động cao hơn ở các trường hợp phức tạp (Testcase 5, 6). Điều này có thể phản ánh:
- Sự thay đổi fitness giữa các cá thể trong quần thể.
- Ảnh hưởng từ các tham số khác như kích thước quần thể hoặc chiến lược lai ghép/đột biến.



Hình 3.2.3: Thể hiện so sánh fitness giữa các số thế hệ



Hình 3.2.4: Thể hiện độ ổn định của fitness qua các thế hệ

Kết luận:

Với các bài toán đơn giản, sử dụng khoảng **100-200 thế hệ** là đủ để đạt hiệu quả cao mà không lãng phí tài nguyên. Với các bài toán lớn hơn, cần kết hợp tối ưu các tham số khác như kích thước quần thể, tỷ lệ đột biến để tăng hiệu suất.

3. Tỉ lệ lai ghép

Tỷ lệ lai ghép trong bài toán Knapsack có ảnh hưởng rõ rệt đến hiệu suất và chất lượng giải pháp.

Tỷ lệ lai ghép thấp (0.7) thường giúp duy trì tính đa dạng trong quần thể và hội tụ nhanh, nhưng lại có độ ổn định thấp và dễ mắc kẹt ở cực trị cục bộ, khiến kết quả không nhất quán trong các bài toán phức tạp.

Tỷ lệ lai ghép trung bình (0.8) thể hiện sự cân bằng tốt giữa tốc độ hội tụ, khả năng tối ưu hóa, và độ ổn định, giúp quần thể đạt được fitness cao mà vẫn duy trì sự đồng đều giữa các thế hệ, phù hợp với đa số bài toán.

Tỷ lệ lai ghép cao (0.9) thường mang lại fitness cuối cùng cao nhất, đặc biệt trong các bài toán phức tạp, nhưng có nhược điểm là dễ hội tụ sớm và mất nhiều thế hệ hơn để đạt tối ưu.

Nhìn chung, tỷ lệ lai ghép **0.8** được xem là lựa chọn tối ưu cho phần lớn bài toán, nhưng việc điều chỉnh tỷ lệ này theo yêu cầu cụ thể có thể cải thiện đáng kể hiệu suất của thuật toán.

4. Tỉ lệ đột biến

Tỷ lệ đột biến trong giải thuật Genetic Algorithm là một tham số sẽ ảnh hưởng trực tiếp đến hiệu quả và khả năng khám phá không gian của thuật toán. Qua thí nghiệm ta thấy được:

Tỷ lệ đột biến thấp (0.02) tỷ lệ đột biến quá thấp sẽ ảnh hưởng đến kết quả của bài toán vì nó giảm đi khả năng khám phá các vùng chưa được tìm kiếm dẫn đến không tìm được kết quả tối ưu.

Tỷ lệ đột biến trung bình (0.03) tỷ lệ đột biến này cải thiện được khả năng ra kết quả tốt hơn 0.02.

Tỷ lệ đột biến cao (0.04) tỷ lệ đột biến này cải thiện được khả năng ra kết quả tối ưu hơn tỉ lệ đột biến 0.03 và cũng là tỷ lệ đột biến lí tưởng bởi càng tăng tỉ lệ đột biến thì khả năng ra kết quả đúng càng lớn nhưng cũng đồng thời phá vỡ các thế hệ tốt khiếm quá trình hội tụ trở nên chậm chạp hơn nên tỷ lệ 0.04 là một tỷ lệ lý tưởng cho bài toán.

Vậy việc lựa chọn tỷ lệ đột biến (**0.04**) sẽ là lựa chọn phù hợp để đảm bảo vừa đạt được kết quả tối ưu nhất, vừa không làm ảnh hưởng đến quá trình hội tụ.

3.2.2. So sánh các toán tử

1. Phương pháp chọn lọc

a) Roulette Wheel Selection

Dựa trên bảng số liệu của từng testcase, sau đây là đánh giá về hai phương pháp **Elitism Selection** và **Roulette-wheel Selection** trên ba tiêu chí chính:

- **Fitness cuối cùng:** Roulette-wheel Selection thường đạt giá trị fitness cuối cùng cao hơn so với Elitism Selection, đặc biệt trong các bài toán có không gian tìm kiếm phức tạp hoặc kích thước lớn.
- **Tốc độ hội tụ:** Elitism Selection cho thấy độ ổn định cao hơn trong các bài toán đơn giản, với kết quả ít dao động hơn giữa các lần chạy.
- **Độ ổn định (STD):** Roulette-wheel Selection thường có STD thấp hơn Elitism Selection trong các bài toán phức tạp, phản ánh sự ổn định và khả năng khai thác hiệu quả các khu vực tiềm năng trong không gian tìm kiếm.

Kết luận : Elitism Selection phù hợp với các bài toán cần hội tụ nhanh, trong khi Roulette-wheel Selection tăng cường tính đa dạng.

b) Rank Selection

Sau khi tiến hành thử nghiệm và phân tích kết quả dựa trên các tiêu chí đánh giá đã thiết lập, chúng tôi đưa ra một số nhận định và kết luận quan trọng như sau:

- **Fitness cuối cùng**: Elitism Selection consistently đạt giá trị fitness cao hơn Rank Selection ở tất cả các testcase, thể hiện ưu thế vượt trội trong việc bảo toàn các cá thể tốt nhất.

- **Tốc độ hội tụ**: Elitism Selection thường hội tụ nhanh hơn hoặc bằng so với Rank Selection, nhờ khả năng giữ lại những cá thể mạnh qua mỗi thế hệ.
- **Độ ổn định**: Elitism Selection có độ lệch chuẩn thấp hơn, cho thấy khả năng ổn định tốt hơn và giảm thiểu rủi ro hội tụ sai.

Kết luận: Với ưu thế về fitness, tốc độ hội tụ và độ ổn định, *Elitism Selection* là kỹ thuật lựa chọn vượt trội hơn, phù hợp với các bài toán cần tối ưu hóa nhanh và ổn định.

c) Tournament Selection

Ảnh hưởng tới đồ thị: Best Fitness có xu hướng tăng chậm hơn so với sử dụng Elitism điều này thể hiện Tournament Selection sẽ tốn thời gian nhiều hơn để tìm cá thể mạnh nhất trong quần thể. So với Elitism Selection, Tournament Selection sẽ giúp duy trì đa dạng quần thể hơn so với Elitism Selection bởi nó chọn ra các nhóm ngẫu nhiên thay vì xếp hạng toàn cục. Điều này tạo cơ hội cho các cá thể mặc dù đã bị loại khỏi giải đấu nhưng vẫn có thể tham gia và chiến thắng ở giải đấu khác và cũng thể hiện rằng các cá thể yếu cũng có thể có mặt trong quần thể mà tham gia vào quá trình sinh sản.

Kết luận: Vậy nên Tournament Selection sẽ phù hợp hơn với bài toán yêu cầu sự đa dạng.

2. Phương pháp lai ghép

a) One-point Crossover

Dựa trên bảng số liệu của từng testcase, sau đây là đánh giá về hai phương pháp **One-point Crossover** và **Uniform Crossover** trên ba tiêu chí chính:

- **Fitness cuối cùng**: Uniform Crossover thường mang lại giá trị fitness cuối cùng cao hơn so với One-point Crossover, nhờ khả năng kết hợp tốt hơn các gen từ bố mẹ.
- **Tốc độ hội tụ**: One-point Crossover cho thấy độ ổn định tốt hơn trong các bài toán nhỏ hoặc có cấu trúc đơn giản, vì nó ít tạo ra sự đột biến lớn trong quần thể.
- **Độ ổn định**: Uniform Crossover thường có STD cao hơn một chút so với One-point Crossover trong các bài toán nhỏ, nhưng điều này phản ánh sự đa dạng và khả năng tìm kiếm mạnh mẽ hơn trong không gian giải pháp.

Kết luận : Uniform Crossover mang lại hiệu suất cao ở bài toán phức tạp, còn One-point Crossover phù hợp với bài toán đơn giản.

b) Arithmetic Crossover

Sau khi tiến hành thử nghiệm và phân tích kết quả dựa trên các tiêu chí đánh giá đã thiết lập, chúng tôi đưa ra một số nhận định và kết luận quan trọng như sau:

- **Fitness cuối cùng**: One-point Crossover thường đạt fitness cuối cùng cao hơn hoặc tương đương Arithmetic Crossover, nhờ cách lai ghép tập trung và trực quan hơn.
- **Tốc độ hội tụ**: One-point Crossover hội tụ nhanh hơn trong hầu hết các trường hợp, giúp giảm số thế hệ cần thiết để đạt được giải pháp tốt.

- **Độ ổn định**: Arithmetic Crossover có độ ổn định tốt hơn một chút, nhưng chênh lệch không đáng kể trong bài toán này.

Kết luận: One-point Crossover là kỹ thuật lai ghép phù hợp hơn, nhờ khả năng hội tụ nhanh và đạt được fitness cao, giúp quá trình tối ưu hóa hiệu quả hơn.

c) Multi – point Crossover

Ảnh hưởng tới đồ thị: Khiến cho Best Fitness tiến triển chậm hơn so với Uniform Crossover vì nó tạo ra sự kết hợp gene phức tạp hơn khi trao đổi nhiều đoạn trên chuỗi gene. Multi – point Crossover tạo ra các cá thể con phức tạp điều này sẽ khiến không đạt được tối ưu cục bộ so với Uniform Crossover.

Kết luận: Multi - point Crossover sẽ phù hợp hơn với các bài toán cần có sự đa dạng.

3. Phương pháp đột biến

a) Scramble Mutation

Dựa trên bảng số liệu của từng testcase, sau đây là đánh giá về hai phương pháp Scramble Mutation và Bit-flip Mutation trên ba tiêu chí chính:

- **Fitness cuối cùng**: Scramble Mutation thường đạt giá trị fitness cuối cùng cao hơn so với Bit-flip Mutation trong hầu hết các bài toán do Scramble Mutation có khả năng hoán đổi và sắp xếp lại các gen trong một đoạn nhất định
- **Tốc độ hội tụ**: Bit-flip Mutation cho thấy độ ổn định tốt hơn trong các bài toán đơn giản, vì nó chỉ thay đổi một số ít gen và ít tạo ra sự xáo trộn lớn trong quần thể.
- **Độ ổn định**: Scramble Mutation thường có STD cao hơn Bit-flip Mutation, đặc biệt trong các bài toán lớn, phản ánh sự đa dạng mạnh mẽ trong quần thể mà nó tạo ra. Tuy nhiên, sự khác biệt này có thể giảm dần khi số thế hệ tăng, vì quần thể có xu hướng hội tụ về vùng tối ưu chung.

Kết luận : Scramble Mutation cải thiện khám phá không gian tốt hơn, trong khi Bit-flip Mutation hỗ trợ tối ưu hóa cục bộ.

b) Swap Mutation

Sau khi tiến hành thử nghiệm và phân tích kết quả dựa trên các tiêu chí đánh giá đã thiết lập, chúng tôi đưa ra một số nhận định và kết luận quan trọng như sau:

- **Fitness cuối cùng**: Bit-flip Mutation thường đạt fitness cuối cùng cao hơn Swap Mutation, cho thấy khả năng tạo ra giải pháp tốt hơn.
- **Tốc độ hội tụ**: Cả hai kỹ thuật có tốc độ hội tụ tương đương, nhưng Swap Mutation đôi khi hội tụ nhanh hơn trong các bài toán phức tạp.
- **Độ ổn định**: Swap Mutation thường ổn định hơn, đặc biệt trong các testcase lớn như Testcase 5 và 6, nhờ cách biến đổi ít phá vỡ cấu trúc của các cá thể.

Kết luận: Mặc dù Swap Mutation có độ ổn định cao hơn, *Bit-flip Mutation* là lựa chọn tối ưu nhờ khả năng đạt fitness cuối cùng tốt nhất, đáp ứng mục tiêu chính của bài toán tối ưu hóa.

c) Gaussian Mutation

Ảnh hưởng trên đồ thị: Khiến cho Best Fitness tăng trưởng chậm hơn do Gaussian Mutation tạo ra những biến đổi nhỏ và dần dần trong quần thể một cách liên tục mà không thúc đẩy sự thay đổi lớn một cách nhanh chóng.

Kết luận: Gaussian Mutation là một phương pháp phức tạp vì giá trị của gene phân phối theo xác xuất chuẩn và chỉ phù hợp với các bài toán yêu cầu sự đa dạng.

CHƯƠNG 4. KẾT LUẬN

4.1. Các Kết Quả Đạt Được

Sau khi thực hiện các thí nghiệm và so sánh hiệu năng của các thuật toán trong giải quyết bài toán Knapsack, đây là phương án tối ưu cho bài toán này:

- 200 cá thể trong quần thể, 200 thế hệ, tỷ lệ lai ghép 0.8, và tỷ lệ đột biến 0.04.
- Elitism Selection, Uniform Crossover, Bit-flip Mutation .

4.1.1. Kết quả của các thông số ảnh hưởng đến thuật toán

Quá trình thực nghiệm này đã nghiên cứu ảnh hưởng của các số thông số quan trọng trong GA bao gồm **kích thước quần thể**, **tỷ lệ lai ghép**, **số thế hệ**, và **tỷ lệ đột biến**. Dựa trên các kết quả thu được, ta có thể đánh giá sự tác động của từng thông số đến hiệu quả tối ưu hóa thông qua các chỉ số như fitness cuối cùng, tốc độ hội tụ (thế hệ), và độ ổn định (std). Mỗi thông số 50, 100, 200, 500 sẽ được thực hiện với nhiều lần test. Thu được các kết quả như sau:

Kích thước quần thể (Population Size) là yếu tố quan trọng ảnh hưởng đến khả năng khám phá không gian tìm kiếm và duy trì tính đa dạng của quần thể.

- **Fitness cuối cùng:** Kích thước quần thể lớn hơn có xu hướng đạt được giá trị fitness cao hơn hoặc tương đương với các kích thước nhỏ. Tuy nhiên, sự khác biệt thường không đáng kể.
- **Tốc độ hội tụ:** Các quần thể có kích thước vừa (100-200) thường hội tụ nhanh hơn so với các kích thước lớn (500) hoặc nhỏ (50). Điều này có thể giải thích rằng quần thể nhỏ thường thiếu tính đa dạng, dẫn đến hội tụ sớm vào các cực trị cục bộ, trong khi quần thể lớn cần nhiều thời gian hơn để xử lý các cá thể.
- **Độ ổn định:** Kích thước quần thể lớn hơn mang lại độ ổn định cao hơn, đặc biệt ở các trường hợp phức tạp. Điều này chỉ ra rằng các cá thể trong quần thể lớn ít bị dao động mạnh khi tiến hóa qua các thế hệ.

Tỷ lệ lai ghép (Crossover Rate) ảnh hưởng đến cách thức trao đổi thông tin giữa các cá thể. Qua kết quả thử nghiệm:

- **Fitness cuối cùng:** Các giá trị fitness cuối cùng tương đối ổn định khi tỷ lệ lai ghép dao động từ 0.7 đến 0.9.
- **Tốc độ hội tụ:** Tỷ lệ lai ghép thấp (0.7) có xu hướng hội tụ chậm hơn so với các giá trị cao hơn (0.8, 0.9). Điều này cho thấy việc trao đổi gen giữa các cá thể thường xuyên hơn giúp đẩy nhanh quá trình tiến hóa.
- **Độ ổn định:** Tỷ lệ lai ghép thấp thường mang lại độ ổn định cao hơn. Ví dụ, ở Testcase 5, độ ổn định tại tỷ lệ 0.7 là 2491.07, thấp hơn so với các tỷ lệ 0.8 và 0.9.

Điều này có thể do tỷ lệ lai ghép thấp giữ lại nhiều gen của thế hệ trước, làm giảm sự biến động trong quần thể.

Số thế hệ (Number of Generations) xác định thời gian mà thuật toán có thể tiến hóa. Các kết quả thử nghiệm chỉ ra rằng:

- **Fitness cuối cùng:** Giá trị fitness thường tăng khi số thế hệ tăng, nhưng có giới hạn. Tuy nhiên, ở các bài toán đơn giản hơn, số thế hệ lớn không mang lại cải thiện đáng kể.
- **Tốc độ hội tụ:** Các testcase cho thấy tốc độ hội tụ không phải lúc nào cũng tỷ lệ thuận với số thế hệ. Trong bài Test, tốc độ hội tụ cao nhất (1 thế hệ) đạt được ở 100 thế hệ, trong khi ở 200 thế hệ, tốc độ hội tụ chậm hơn (4 thế hệ).
- **Độ ổn định:** Số thế hệ lớn hơn thường làm tăng độ dao động (std), đặc biệt ở các testcase phức tạp. Điều này có thể do quá trình tiến hóa kéo dài gây ra sự phân tán trong quần thể.

Tỷ lệ đột biến (Mutation Rate) được đánh giá qua chuỗi kết quả từ nhiều thế hệ. Qua thí nghiệm ta thu được:

- **Fitness cuối cùng:** Giá trị fitness cuối cùng tăng dần khi tỷ lệ đột biến cao hơn, đạt đỉnh ở các thế hệ cuối (thế hệ 195-200).
- **Tốc độ hội tụ:** Tỷ lệ đột biến cao giúp duy trì tính đa dạng trong quần thể, tuy nhiên không có sự khác biệt lớn về tốc độ hội tụ giữa các thế hệ.
- **Độ ổn định:** Khi tỷ lệ đột biến tăng, độ ổn định (std) có xu hướng giảm, cho thấy thuật toán tập trung hơn vào các vùng tìm kiếm tiềm năng.

Các thử nghiệm trên chỉ ra rằng sự kết hợp giữa các thông số có vai trò quan trọng trong việc đạt được hiệu suất tối ưu . **Thu ra được:**

- **Kích thước quần thể** vừa phải (100-200) là lựa chọn tốt cho hầu hết các bài toán vì cân bằng được giữa tốc độ hội tụ, giá trị fitness và độ ổn định.
- **Tỷ lệ lai ghép** cao (0.9) thường hiệu quả hơn trong các bài toán phức tạp, do khả năng khám phá không gian tìm kiếm tốt hơn.
- **Số thế hệ** tầm trung (200-500) phù hợp khi cần đạt giá trị fitness cao nhất, nhưng nên xem xét thời gian tính toán để đảm bảo hiệu quả.
- **Tỷ lệ đột biến** vừa phải giúp tăng tính đa dạng và tránh hội tụ sớm vào cực trị cục bộ.

Qua đây kết luận ra được tùy thuộc vào bài toán cụ thể, cần lựa chọn và điều chỉnh các thông số này một cách linh hoạt để đạt được hiệu quả tối ưu. Những bài toán phức tạp có thể yêu cầu thông số lớn . Ngược lại, những bài toán đơn giản sẽ dùng thông số nhỏ điều này có thể giúp tối ưu bài toán .

4.1.2. Kết quả so sánh kỹ thuật GA

Cụ thể là Elitism Selection, **Uniform Crossover**, và **Bit-flip Mutation**, so sánh chúng qua các tiêu chí: **Fitness cuối cùng**, **tốc độ hội tụ**, và **độ ổn định**.

Fitness cuối cùng đại diện cho chất lượng của giải pháp được tìm thấy.

- **Elitism Selection**: Kỹ thuật này thường đạt Fitness tốt, ổn định qua các TestCase, nhưng không phải lúc nào cũng đạt giá trị cao nhất. Fitness cuối cùng từ Elitism Selection cho thấy ưu điểm trong việc giữ lại các cá thể tốt nhất qua mỗi thế hệ, đảm bảo rằng giải pháp không bị thoái hóa.
- **Uniform Crossover**: Kỹ thuật này thường đạt giá trị Fitness cuối cùng cao nhất trong nhiều TestCase, đặc biệt là trong các bài toán phức tạp. Tuy nhiên, điều này đi kèm với việc mất nhiều thế hệ hơn để hội tụ.
- **Bit-flip Mutation**: Đây là kỹ thuật có Fitness thấp hơn so với hai phương pháp còn lại trong phần lớn các trường hợp. Điều này cho thấy, mặc dù đột biến mang tính đa dạng hóa giải pháp, nhưng khi không được kết hợp tốt với các phương pháp khác, nó có thể làm giảm chất lượng giải pháp cuối cùng.

Tốc độ hội tụ là số thế hệ cần thiết để đạt được Fitness cao nhất hoặc trạng thái ổn định.

- **Elitism Selection**: Kỹ thuật này vượt trội về tốc độ hội tụ. Nó cho thấy khả năng nhanh chóng đạt đến giải pháp tốt nhờ cơ chế ưu tiên các cá thể mạnh nhất.
- **Uniform Crossover**: Tốc độ hội tụ của kỹ thuật này chậm hơn Elitism Selection trong hầu hết các trường hợp. Nguyên nhân là do Uniform Crossover tạo ra các cá thể mới từ hai cha mẹ với tính ngẫu nhiên cao, dẫn đến sự khám phá nhiều hơn, nhưng cũng làm chậm quá trình hội tụ.
- **Bit-flip Mutation**: Kỹ thuật này có tốc độ hội tụ tương đương hoặc chậm hơn Elitism Selection và Uniform Crossover, phụ thuộc vào bài toán. Đột biến giúp khám phá các giải pháp mới nhưng không hiệu quả trong việc hội tụ nhanh chóng.

Độ ổn định được đánh giá dựa trên giá trị độ lệch chuẩn (std) của các lần chạy.

- **Elitism Selection**: Độ ổn định của kỹ thuật này thường cao, thể hiện qua giá trị std thấp trong nhiều TestCase. Điều này chứng tỏ Elitism Selection đáng tin cậy khi cần giải pháp ổn định.
- **Uniform Crossover**: Kỹ thuật này có độ ổn định khá tốt, đặc biệt khi kết hợp với các thành phần GA khác. Mặc dù đôi khi độ ổn định có thể thấp hơn một chút so với Elitism Selection, sự khác biệt không đáng kể.
- **Bit-flip Mutation**: Độ ổn định của kỹ thuật này thường kém hơn so với hai kỹ thuật còn lại, thể hiện qua giá trị std cao hơn. Điều này là do tính ngẫu nhiên cao của đột biến, gây ra sự dao động trong kết quả.

4.2. Những Hạn Chế và Hướng Phát Triển

4.2.1. Những Hạn Chế

Việc áp dụng giải thuật di truyền (GA) cho bài toán Knapsack gặp phải một số thách thức và hạn chế như sau:

Chất lượng giải pháp: giải thuật di truyền (GA) không đảm bảo tìm được kết quả tối ưu toàn cục mà chỉ cung cấp kết quả gần đúng. Điều này đặc biệt rõ ràng khi không gian tìm kiếm của bài toán rất lớn hoặc phức tạp.

Hiệu suất tính toán: giải thuật di truyền (GA) yêu cầu nhiều thế hệ và kích thước quần thể lớn để đạt được kết quả tốt, dẫn đến chi phí tính toán cao. Điều chỉnh các tham số như tỷ lệ đột biến, lai ghép và kích thước quần thể là một nhiệm vụ phức tạp, đòi hỏi thời gian và kinh nghiệm.

Hội tụ chậm và cực trị cục bộ: giải thuật di truyền (GA) dễ bị mắc kẹt trong cực trị cục bộ, đặc biệt khi sự đa dạng trong quần thể giảm dần qua các thế hệ.

Xử lý ràng buộc: Đảm bảo nghiệm không vi phạm ràng buộc (ví dụ: trọng lượng tổng không vượt quá giới hạn) là một thách thức, đặc biệt với các bài toán có nhiều ràng buộc phức tạp.

Phụ thuộc vào thiết kế hàm thích nghi: Hàm thích nghi không phù hợp có thể dẫn đến việc thuật toán hội tụ vào các nghiệm không khả thi hoặc không tối ưu.

4.2.2. Hướng phát triển

Để khắc phục các hạn chế trên, một số hướng phát triển tiềm năng cho việc sử dụng GA trong bài toán Knapsack bao gồm:

Tăng cường biểu diễn và thiết kế hàm thích nghi: Áp dụng các cách biểu diễn cá thể hiệu quả hơn như mã nhị phân, mã thực hoặc mã lai. Thiết kế hàm thích nghi có cơ chế phạt (penalty function) để xử lý các nghiệm vi phạm ràng buộc, giúp thuật toán tập trung vào các nghiệm khả thi.

Lai ghép với các thuật toán khác: Kết hợp GA với các phương pháp như Quy hoạch động (Dynamic Programming) hoặc Tìm kiếm cục bộ (Local Search) để cải thiện chất lượng nghiệm. Sử dụng các chiến lược lai ghép để giảm thiểu hội tụ chậm hoặc cực trị cục bộ.

Tăng cường đa dạng di truyền: Duy trì sự đa dạng trong quần thể bằng cách sử dụng đột biến động (adaptive mutation) hoặc các chiến lược lai ghép đa dạng. Áp dụng cơ chế lai ghép giữa các quần thể khác nhau để mở rộng không gian tìm kiếm.

Tăng tốc độ hội tụ: Sử dụng chiến lược elitism để bảo vệ các nghiệm tốt nhất qua từng thế hệ. Áp dụng phương pháp chọn lọc tiên tiến như tournament selection hoặc rank selection để cải thiện chất lượng quần thể.

Sử dụng thuật toán song song: Chạy nhiều quần thể song song hoặc thực hiện các phép toán di truyền trên các luồng xử lý khác nhau để tăng tốc độ tính toán và cải thiện khả năng tìm nghiệm tối ưu.

Kết hợp học máy và học tăng cường: Áp dụng mô hình học máy để dự đoán hoặc tối ưu hóa các tham số của GA. Sử dụng học tăng cường để điều chỉnh chiến lược lai ghép, đột biến và chọn lọc trong quá trình tiến hóa.

TÀI LIỆU THAM KHẢO

1. Katoch, S., Chauhan, S. S., & Kumar, V. (2020). A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, *80*(5), 8091–8126. https://doi.org/10.1007/s11042-020-10139-6

2. Taskiran, G. K. (2010). *An improved genetic algorithm for knapsack problems (Master's thesis)*. Wright State University. Retrieved from https://corescholar.libraries.wright.edu/etd_all/328/

3. Tien, L. D. (2024, November 27). *Thuật toán di truyền - Ứng dụng giải một số bài toán kinh điển (phần 1)*. *Viblo*. https://viblo.asia/p/thuat-toan-di-truyen-ung-dung-giai-mot-so-bai-toan-kinh-dien-phan-1-RQqKLxJzK7z

PHỤ LỤC

Phụ lục A: Mã nguồn dự án

Truy cập trực tiếp qua Github tại liên kết sau: Bấm tại đây

Phụ lục B: Hướng dẫn cài đặt

Bước 1: Cài đặt Git

- Truy cập trang web chính thức của Git: Git Bash
- Tải về phiên bản Git phù hợp với hệ điều hành của bạn (Windows, macOS, Linux).
- Chạy file cài đặt và làm theo hướng dẫn trên màn hình để cài đặt Git.

Bước 2: Mở Terminal hoặc Command Prompt

- Windows: Mở `Command Prompt` hoặc `Git Bash`. Bạn có thể tìm thấy Git Bash trong menu Start sau khi cài đặt Git.
- macOS/Linux: Mở ứng dụng `Terminal` từ menu ứng dụng.

Bước 3: Clone Repository

- Trong terminal hoặc command prompt, gõ lệnh sau và nhấn Enter: git clone https://github.com/Avcuongy/Knapsack-Problem-Genetic-Algorithm.git
- Lệnh này sẽ tải dự án về máy tính của bạn. Sau khi quá trình clone hoàn tất. - Bạn sẽ thấy một thư mục mới có tên là Knapsack-Problem-Genetic-Algorithm chứa mã nguồn của dự án.

Bước 4:

- Mở Visual Studio Code.
- Trong VS Code, nhấn Ctrl + O (hoặc Cmd + O trên macOS) để mở thư mục.
- Chọn thư mục Knapsack-Problem-Genetic-Algorithm mà bạn vừa clone về.

Bước 5:

- Trong VS Code, mở thư mục Knapsack-Problem-Genetic-Algorithm, tìm thư mục Knapsack (GA).
- Mở thư mục Knapsack (GA) và tìm tệp GUI.py.
- Nhấp đúp vào GUI.py để mở tệp trong VS Code.
- Mở terminal trong VS Code và gõ lệnh sau để chạy tệp GUI.py

Phụ lục C: Bảng phân công

Thành viên	Nhiệm vụ

Đặng Xuân Cường	Thực nghiệm với các thông số ảnh hưởng: – Kích thước quần thể – Số thế hệ – Tỉ lệ lai ghép – Tỉ lệ đột biến Thiết lập cấu hình tiêu chuẩn: – Selection: Elitism Selection – Crossover: One-point crossover – Mutation: Bit-flip Mutation
Trần Thọ	Thực nghiệm và so sánh với cấu hình tiêu chuẩn: – Selection: Roulette Wheel Selection – Crossover: Uniform crossover – Mutation: Scramble Mutation
Dương Khải Nghiêm	Thực nghiệm và so sánh với cấu hình tiêu chuẩn: – Selection: Rank Selection – Crossover: Arithmetic Crossover – Mutation: Swap Mutation
Huỳnh Trần Bảo Việt	Thực nghiệm và so sánh với cấu hình tiêu chuẩn: – Selection: Tournament Selection – Crossover: Muti-point crossover – Mutation: Gaussian Mutation