VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

MATHEMATICAL MODELING (CO2011)

Assignment(Semester: 241, Duration: 04 weeks)

"Overview on Policies"

Instructor(s): Lê Hồng Trang

Students: Nguyễn Lê Nguyên - 2312365 *(Group L06 - Team 82, **Leader**)*
Hồ Minh Nhiên - 2312517 *(Group L06 - Team 82)*
Nguyễn Đăng Khoa - 2311614 *(Group L06 - Team 82)*
Trịnh Vũ Đức Hải - 2310886 *(Group L06 - Team 82)*
Lê Quang Khải - 2311548 *(Group L06 - Team 82)*

HO CHI MINH CITY, NOVEMBER 2024

Contents

List of Symbols

$\mathbb{N}$ Set of natural numbers

$\mathbb{R}$ Set of real numbers

$\mathbb{R}^+$ Set of positive real numbers

List of Acronyms

ODE (First-Order) Ordinary Differential Equation

IVP Initial-Value Problem

LTE Local Truncation Error

DS Dynamical System

Fig. Figure

Tab. Table

Sys. System of Equations

Eq. Equation

e.g. For Example

i.e. That Is

List of Figures

List of Tables

Member list & Workload

No.	Fullname	Student ID	Problems	% done
1	Nguyễn Lê Nguyên	2312365	-Tìm hiểu giải thuật Genetic - Đưa tập dữ liệu kiểm thử vào môi trường cutting_stock	100%
2	Nguyễn Đăng Khoa	2311614	- Tìm hiểu Stock filled_rate optimize heuristic	100%
3	Lê Quang Khải	2311548	- Tìm hiểu Case study - LaTeX	100%
4	Hồ Minh Nhiên	2312517	- Cải tiến thuật toán,sửa lỗi - LaTeX	100%
5	Trịnh Vũ Đức Hải	2310886	- Tìm hiểu Case Study - LaTeX	100%

Table 1: Member list & workload

1 Giới thiệu thuật toán

1.1 Stock filled-rate optimization heuristic

Thuật toán Fill Rate Optimization là một phương pháp phổ biến trong bài toán cắt nguyên liệu (cutting stock). Thuật toán này hoạt động dựa trên việc sắp xếp các đối tượng cần cắt (hoặc đóng gói) theo diện tích giảm dần, sau đó tìm vị trí đặt chúng sao cho tối ưu tỷ lệ lấp đầy của vùng trống. Trong bài toán cắt 2D, thuật toán này hướng đến:

- **Tối ưu hóa tỷ lệ lấp đầy không gian trống.**
- **Giảm thiểu lãng phí không gian.**

Mô tả thuật toán

1. **Sắp xếp sản phẩm**: Các sản phẩm được sắp xếp theo thứ tự diện tích giảm dần. Mục tiêu là xử lý các sản phẩm lớn trước để tận dụng không gian hiệu quả.
2. **Phân tích không gian trống**:
   - Tìm tất cả các vùng trống khả dụng trong kho bằng cách quét toàn bộ kho và xác định các hình chữ nhật rỗng.
   - Xác định kích thước và vị trí của mỗi vùng trống.
3. **Tìm vị trí đặt phù hợp nhất**:
   - Duyệt qua từng sản phẩm và tìm vị trí trong các vùng trống với tỷ lệ lấp đầy cao nhất, được tính bằng:

$$\text{Tỷ lệ lấp đầy} = \frac{\text{Diện tích sản phẩm}}{\text{Diện tích vùng trống}}$$

   - Xem xét cả hai chiều quay (ngang và dọc) của sản phẩm để tăng khả năng đặt.
4. **Cập nhật kho**: Sau khi đặt một sản phẩm, trạng thái kho được cập nhật để loại bỏ vùng trống đã sử dụng. Tiếp tục với sản phẩm kế tiếp.

Mã giả của thuật toán Fill Rate Optimization

```
1. Sort the products in decreasing order of area (width * height):
sorted_products = sort(products by width * height in descending order)

2. For each product in sorted_products:
   a. If product.quantity > 0:
      i. Set prod_size = product.size
      ii. Initialize:
          best_action = None
          max_fill_rate = -1

      iii. For each stock in stocks:
           - Calculate stock_size = (stock_width, stock_height)
           - Find empty_areas = get_empty_areas(stock)

           - For each area in empty_areas:
               - For both orientations (width, height) of prod_size:
```

```
                    - If the product fits in the area:
                        fill_rate = (prod_area) / (area_area)
                        If fill_rate > max_fill_rate:
                            Update:
                                best_action = (stock_idx, position, size)
                                max_fill_rate = fill_rate

      iv. If best_action is not None:
           - Place the product into the stock at best_action.position
           - Reduce product.quantity by 1

3. Return the list of actions
```

Ưu điểm của thuật toán Fill Rate Optimization:

1. **Tối ưu hóa không gian hiệu quả**: Tỷ lệ lấp đầy cao giúp giảm thiểu lãng phí không gian trống.
2. **Thích hợp với nhiều bài toán thực tế**: Ứng dụng rộng rãi trong quản lý kho, sản xuất, và cắt vật liệu.
3. **Linh hoạt trong xử lý**: Hỗ trợ cả hai chiều quay của sản phẩm và khả năng phân tích vùng trống.

Nhược điểm của thuật toán Fill Rate Optimization:

1. **Độ phức tạp tính toán cao**: Việc phân tích vùng trống và kiểm tra tỷ lệ lấp đầy cho tất cả các vị trí có thể tốn nhiều thời gian.
2. **Không đảm bảo tối ưu toàn cục**: Chỉ tối ưu ở từng bước hiện tại mà không xét đến toàn bộ bài toán.
3. **Phụ thuộc vào cách sắp xếp ban đầu**: Kết quả phụ thuộc vào thứ tự sắp xếp sản phẩm theo diện tích.

Kết luận: Thuật toán Fill Rate Optimization là một giải pháp trực quan và hiệu quả để giải quyết bài toán cắt nguyên liệu 2D. Mặc dù không đảm bảo tối ưu toàn cục, nó vẫn rất hữu ích trong các ứng dụng thực tế nhờ khả năng triển khai dễ dàng và tối ưu hóa tốt ở mức độ cục bộ.

1.2 Genetic Algorithm

Thuật toán di truyền (Genetic Algorithm - GA) là một phương pháp tìm kiếm metaheuristic dựa trên cơ chế tiến hóa sinh học. GA được thiết kế để giải quyết các bài toán tối ưu hóa bằng cách mô phỏng các quá trình tiến hóa như *chọn lọc tự nhiên* (Selection), *lai ghép* (Crossover), và *đột biến* (Mutation).

Trong bài toán cắt nguyên liệu 2D, thuật toán GA được áp dụng để:

- Giảm thiểu **diện tích lãng phí**: Các khoảng trống không thể sử dụng trên tấm nguyên liệu.
- Giảm **số lượng tấm nguyên liệu** cần sử dụng.

Các giai đoạn chính trong thuật toán GA

Khởi tạo quần thể (Initialize Population)

- **Ý nghĩa:** Tạo một tập hợp các giải pháp ban đầu (các cá thể), mỗi cá thể đại diện cho một cách sắp xếp các sản phẩm lên tấm nguyên liệu.
- **Thực hiện:**
  1. Tạo ngẫu nhiên các cá thể hoặc dựa trên các chiến lược tham lam (heuristic).
  2. Mỗi cá thể được biểu diễn dưới dạng danh sách các hành động (*actions*), trong đó mỗi hành động xác định:
     - Tấm nguyên liệu nào sẽ chứa sản phẩm.
     - Vị trí đặt sản phẩm trên tấm (tọa độ x, y).
     - Kích thước của sản phẩm.
  3. Các sản phẩm lớn thường được ưu tiên sắp xếp trước để giảm diện tích lãng phí.

Đánh giá độ thích nghi (Evaluate Fitness)

- **Ý nghĩa:** Xác định chất lượng của mỗi cá thể (giải pháp) trong quần thể dựa trên tiêu chí tối ưu hóa.
- **Hàm thích nghi (Fitness Function):** Hàm thích nghi được thiết kế để ưu tiên các giải pháp sử dụng ít tấm nguyên liệu nhất có thể, giảm thiểu số lượng tài nguyên cần thiết mà không quan tâm đến diện tích lãng phí.
  - Được tính dựa trên:
    - **Số lượng tấm nguyên liệu sử dụng:** Giải pháp càng sử dụng ít tấm nguyên liệu, độ thích nghi càng cao.
  - Điểm fitness càng thấp, chất lượng giải pháp càng tốt.

Tiến hóa quần thể (Evolve Population)

Quá trình tiến hóa được thực hiện qua nhiều thế hệ. Mỗi thế hệ gồm ba bước chính: **Chọn lọc (Selection)**, **Lai ghép (Crossover)**, và **Đột biến (Mutation)**.

a. Chọn lọc (Selection)

- **Ý nghĩa:** Lựa chọn các cá thể tốt từ thế hệ hiện tại để tham gia sinh sản.
- **Phương pháp chọn lọc:**
  - **Rank Selection:**
    - Sắp xếp quần thể theo điểm fitness.
    - Phân bổ xác suất chọn dựa trên xếp hạng, đảm bảo cá thể tốt hơn có xác suất cao hơn.

b. Lai ghép (Crossover)

- **Ý nghĩa:** Tạo thế hệ con bằng cách kết hợp thông tin từ hai cá thể cha mẹ.
- **Phương pháp: Two-Point Crossover.**
  - Chọn ngẫu nhiên hai điểm trên danh sách hành động của cha mẹ.
  - Ghép đoạn giữa hai điểm từ cha mẹ khác nhau để tạo thành con.
  - Ví dụ:

    Cha mẹ A: [A1, A2, A3, A4] Cha mẹ B: [B1, B2, B3, B4]

    Con: [A1, A2, B3, B4]
- **Xác suất lai ghép:** Thông thường từ 0.6 đến 0.9.

c. Đột biến (Mutation)

- **Ý nghĩa:** Thay đổi ngẫu nhiên một phần của cá thể để duy trì tính đa dạng và tránh rơi vào cực trị cục bộ.
- **Thực hiện:**
  - Chọn ngẫu nhiên một hành động trong cá thể.
  - Thay đổi hành động bằng cách chọn tấm nguyên liệu khác hoặc thay đổi vị trí đặt sản phẩm.
- **Xác suất đột biến:** Nhỏ, thường từ 0.01 đến 0.2.

4. Lặp lại qua các thế hệ

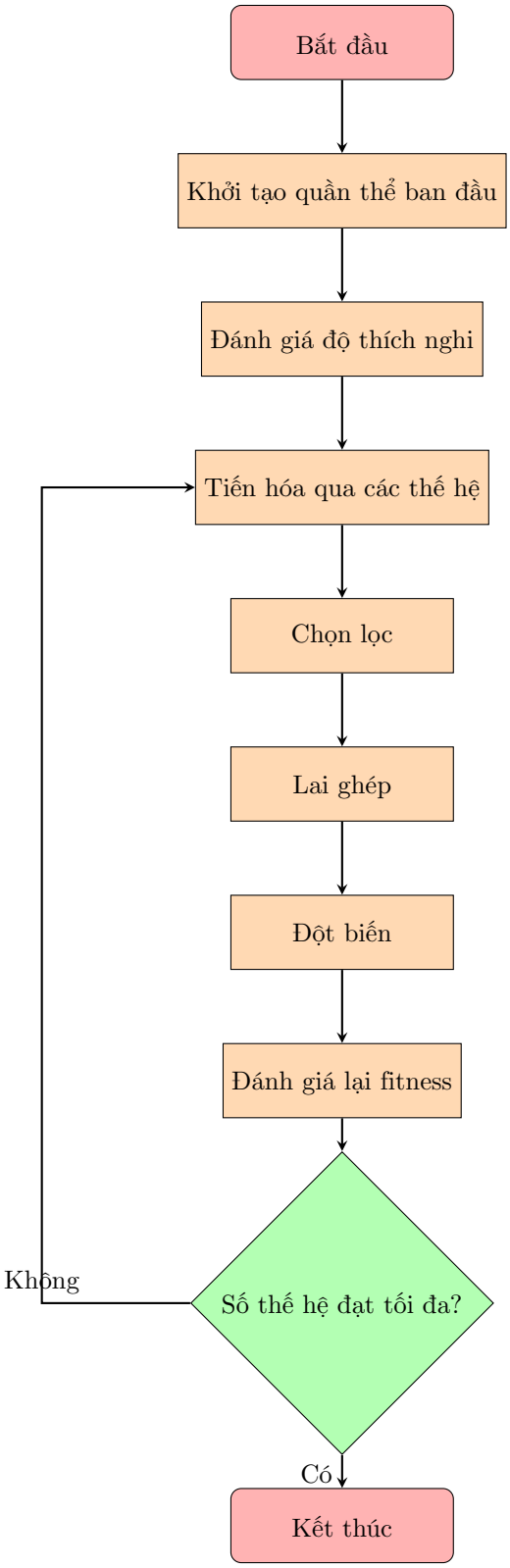
- Các bước *Chọn lọc*, *Lai ghép*, và *Đột biến* được thực hiện lặp lại qua nhiều thế hệ.
- Mỗi thế hệ, các cá thể tốt dần thay thế cá thể kém, cải thiện chất lượng quần thể.

5. Kết thúc

- Dừng thuật toán khi đạt một trong các điều kiện sau:
  - Số thế hệ tối đa.
  - Không có cải thiện đáng kể trong fitness qua nhiều thế hệ.
- Trả về cá thể tốt nhất trong quần thể làm lời giải.

Mô tả thuật toán

Flowchart của thuật toán Genetic Thuật toán Genetic (GA) có thể tóm tắt được mô tả qua các bước sau:


Bắt đầu
Khởi tạo quần thể ban đầu
Đánh giá độ thích nghi
Tiến hóa qua các thế hệ
Chọn lọc
Lai ghép
Đột biến
Đánh giá lại fitness
Số thế hệ đạt tối đa?
Không
Có
Kết thúc

Ưu điểm và nhược điểm của thuật toán GA

Ưu điểm:

1. Tính thích nghi cao, có thể áp dụng cho nhiều loại bài toán.
2. Đảm bảo tính đa dạng của quần thể, giảm khả năng bị kẹt trong cực trị cục bộ.

Nhược điểm:

1. Đòi hỏi thời gian tính toán lớn.
2. Không đảm bảo tối ưu toàn cục.

2 Đánh giá hiệu suất chương trình

2.1 Chỉ Số Đánh Giá Hiệu Suất Thuật Toán `trim_loss`

Chỉ số **trim_loss** được tính trong hàm **get_info** như sau:

```
    trim_loss = []

        for sid, stock in enumerate(self._stocks):
            if self.cutted_stocks[sid] == 0:
                continue
            tl = (stock == -1).sum() / (stock != -2).sum()
            trim_loss.append(tl)
```

Từ đoạn code trên, ta có được công thức sau:

$$\text{trim\_loss} = \frac{\sum \frac{\text{Diện tích đã dùng trên stock}}{\text{Diện tích stock}}}{\text{Số lượng các stock đã dùng}}$$

Như vậy, chỉ số trên đánh giá độ lãng phí không gian trung bình trên các stock mà ta dùng để cắt. Do đó mục tiêu của chương trình là giảm chỉ số này. Ngoài ra, chương trình cũng cung cấp một chỉ số gọi là **filled_ratio**, tuy nhiên chỉ số này chỉ phản số lượng stock đã sử dụng (dưới dạng tỉ số) nên không phải là mục tiêu của chương trình.

2.2 Đánh giá hiệu quả các thuật toán dựa trên 2 tập dữ liệu tự tạo

2.2.1 Cách đưa tập dữ liệu vào môi trường cutting_stock

Dữ liệu về stocks và products được tạo ngẫu nhiên trong hàm reset như trong ảnh:

```python
def reset(self, seed=None, options=None):
    # We need the following line to seed self.np_random
    super().reset(seed=seed)
    self.cutted_stocks = np.full((self.num_stocks,), fill_value=0, dtype=int)
    self._stocks = []

    # Randomize stocks
    self._stocks = []
    for _ in range(self.num_stocks):
        width = np.random.randint(low=self.min_w, high=self.max_w + 1)
        height = np.random.randint(low=self.min_h, high=self.max_h + 1)
        stock = np.full(shape=(self.max_w, self.max_h), fill_value=-2, dtype=int)
        stock[:width, :height] = -1  # Empty cells are marked as -1
        self._stocks.append(stock)
    self._stocks = tuple(self._stocks)



    # Randomize products
    self._products = []
    num_type_products = np.random.randint(low=1, high=self.max_product_type)
    for _ in range(num_type_products):
        width = np.random.randint(low=1, high=self.min_w + 1)
        height = np.random.randint(low=1, high=self.min_h + 1)
        quantity = np.random.randint(low=1, high=self.max_product_per_type + 1)
        product = {"size": np.array([width, height]), "quantity": quantity}
        self._products.append(product)
    self._products = tuple(self._products)
```

Figure 1: Products và Stocks được tạo ngẫu nhiên trong hàm reset

Do đó ta chỉ cần xóa 2 phần tạo ngẫu nhiên trên và thay bằng mảng products và stocks mà ta tự tạo như sau:

```python
width2 = [51, 80, 91, 63, 59, 58, 78, 85, 96, 57, 75, 65, 74, 53, 93, 65, 69, 82, 79, 55, 67, 93, 84, 66, 74, 53, 62, 89, 58, 77
height2 = [88, 75, 91, 55, 83, 85, 95, 58, 54, 91, 99, 95, 73, 67, 92, 92, 76, 97, 69, 62, 57, 53, 77, 85, 90, 98, 80, 97, 96, 9

ite = len(width2)
for j in range(self.num_stocks):

    stock = np.full(shape=(self.max_w, self.max_h), fill_value=-2, dtype=int)
    stock[:width2[j], :height2[j]] = -1
    self._stocks.append(stock)


self._stocks = tuple(self._stocks)
```

Figure 2: Tạo 100 stocks trong hàm reset

Đối với product, do có hai tập dữ liệu, ứng với **i=0** và **i=1** nên ta có thể tận dụng tham số **option** của môi trường để lựa chọn tập dữ liệu khi chạy file main. Thông tin về hai tập dữ liệu product sẽ được cung cấp chi tiết ở phần dưới.

```
        width1 = []
        height1 = []
        quantity1 = []  #max là 20
        i = options # Cậu tự chọn
        if i==0:
            width1 = [28, 37, 1, 42, 13, 37, 26, 46, 32, 28, 43, 36, 14, 3, 19, 28, 6, 40, 6, 33, 2, 42, 23, 34, 8]
            height1= [18, 25, 43, 40, 17, 47, 34, 41, 31, 30, 31, 4, 38, 40, 15, 49, 14, 38, 24, 22, 13, 25, 9, 42, 49]
            quantity1 = [2, 2, 2, 5, 2, 4, 7, 10, 2, 7, 6, 3, 3, 10, 3, 6, 8, 9, 9, 5, 5, 4, 5, 10, 10]
        elif i == 1:
            width1 =  [7, 20, 47, 1, 45, 1, 48, 36, 11, 15, 11, 35, 23, 18, 2, 9, 38, 38, 27, 49, 42, 20, 15, 34, 25]
            height1 =  [43, 41, 49, 27, 21, 32, 29, 49, 50, 4, 6, 17, 36, 49, 23, 35, 37, 41, 33, 34, 25, 6, 16, 47, 11]
            quantity1 =  [1, 2, 3, 4, 5, 6, 1, 2, 3, 3, 2, 2, 2, 2, 2, 5, 6, 3, 4, 4, 3, 2, 2, 12, 10]


        self._products = []
        ite = len(width1)
        for j in range(25):

            width = width1[j]
            height = height1[j]
            quantity = quantity1[j]
            product = {"size": np.array([width, height]), "quantity": quantity}
            self._products.append(product)

        self._products = tuple(self._products)
```

Figure 3: Tạo product stocks trong hàm reset

Hàm main được nhóm chỉnh sửa như sau để chạy trên hai tập dữ liệu về product trên, trong đó **option** được truyền vào để lựa chọn tập dữ liệu ứng với giá trị 0 hoặc 1:

```
if __name__ == "__main__":

    observation, info = env.reset(seed=SEED,options=0)  # Seed cố định cho môi trường

    policy = GeneticPolicy()
    for _ in range(1000):
        action = policy.get_action(observation,info)
        observation,reward,terminated,truncated,info = env.step(action)
        if terminated or truncated:
            print(info)
            break


env.close()
```

Figure 4: Hàm main được nhóm chỉnh sửa

2.2.2 Dữ liệu về stocks

Việc kiểm thử thuật toán trên hai tập dữ liệu product sẽ dùng chung một tập dữ liệu về stock được cho trong bảng sau(do có 100 stock nên bảng chỉ liệt kê ra một số stock):

Stock Index	Width	Height
1	51	88
2	80	75
3	91	91
4	63	55
5	59	83
6	58	85
	.....	
95	60	65
96	52	63
97	63	88
98	87	87
99	78	59
100	59	88

Table 2: Tập dữ liệu stocks

Hình ảnh các stock khi chạy chương trình như sau:

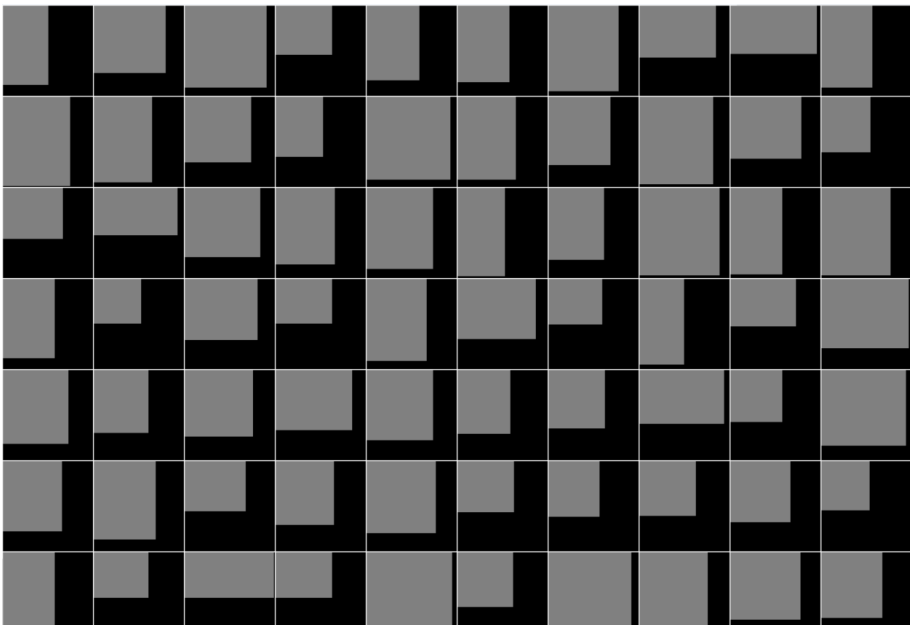


Figure 5: Các stocks trong tập dữ liệu trên

2.2.3 Tập dữ liệu product kiểm thử 1

Product Index	Width	Height	Quantity
1	7	43	1
2	20	41	2
3	47	49	3
4	1	27	4
5	45	21	5
6	1	32	6
7	48	29	1
8	36	49	2
9	11	50	3
10	15	4	3
11	11	6	2
12	35	17	2
13	23	36	2
14	18	49	2
15	2	23	2
16	9	35	5
17	38	37	6
18	38	41	3
19	27	33	4
20	49	34	4
21	42	25	3
22	20	6	2
23	15	16	2
24	34	47	12
25	25	11	10

Table 3: Tập dữ liệu product 1

2.2.3.a Kết quả thuật toán Genetic

Với tổng số cá thể trong một quần thể là 10 và tiến hóa qua 20 thế hệ, thuật toán Genetic tạo ra một cấu hình tương đối nhưng chưa thật sự tối ưu:

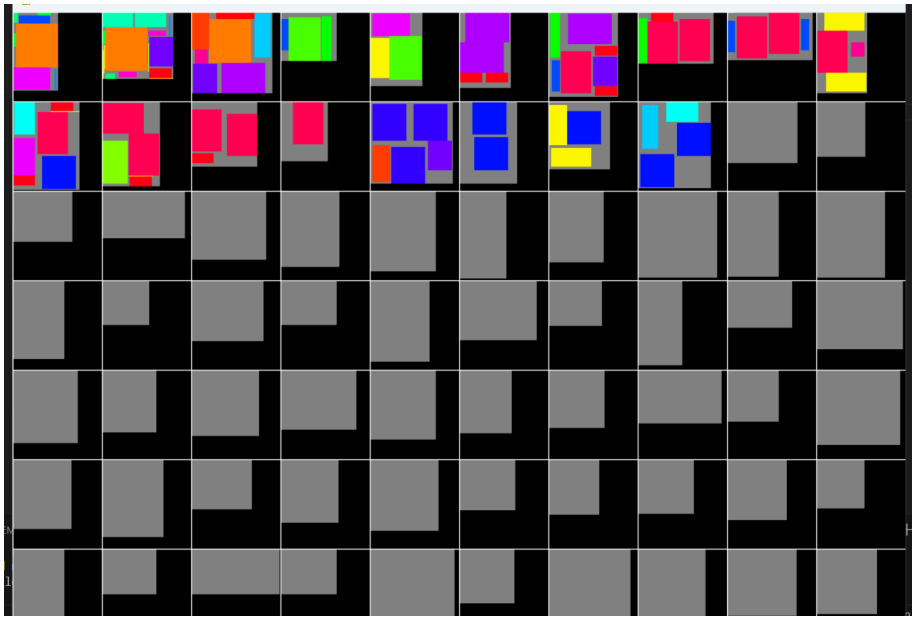


Figure 6: Kết quả thuật toán Genetic trên tập dữ liệu 1

Kết quả các chỉ số đánh giá thuật toán như sau:

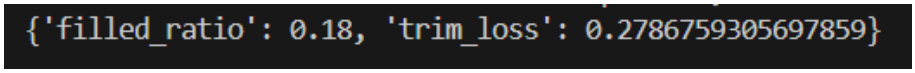


Figure 7: Kết quả hai chỉ số đánh giá của Genetic trên tập dữ liệu 1

Tuy nhiên, thông qua việc in thông tin mỗi 20 cá thể của 10 thế hệ vào một file text (thông qua thư việc logging), ta nhận thấy thuật toán đã tìm ra được cá thể tốt nhất (hay cấu hình tốt nhất) ngay từ giai đoạn khởi tạo quần thể, ta có thể quan sát điều đó thông qua hai ảnh dưới đây:

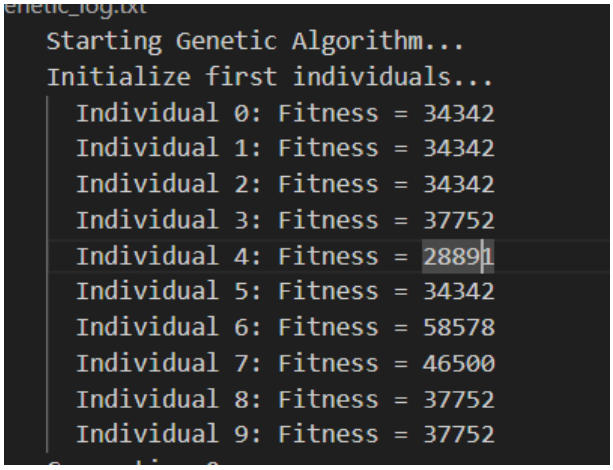


Figure 8: 10 cá thể ban đầu được khởi tạo (test 1)

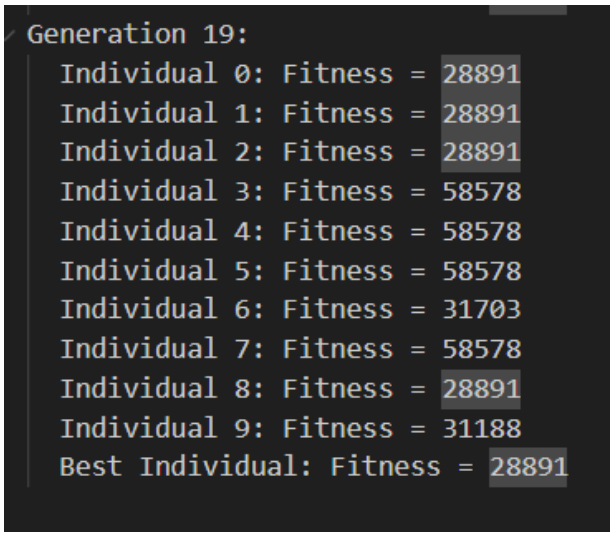


Figure 9: 10 cá thể trong thế hệ cuối cùng(test 2)

Việc lai ghép và đột biến đã không thể tạo ra cá thể tốt hơn. Do đó cá thể với điểm fitness thấp nhất (28891 đơn vị diện tích trống trên các stock đã dùng) đã được giữ lại qua 20 thế hệ thông qua cơ chế chọn lọc và giữ lại 2 cá thể ưu việt nhất của thế hệ hiện tại cho thế hệ sau.

2.2.3.b Kết quả Stock filled_rate optimize heuristic

Sau khi đặt tất cả các products và stocks thì kết quả như hình:

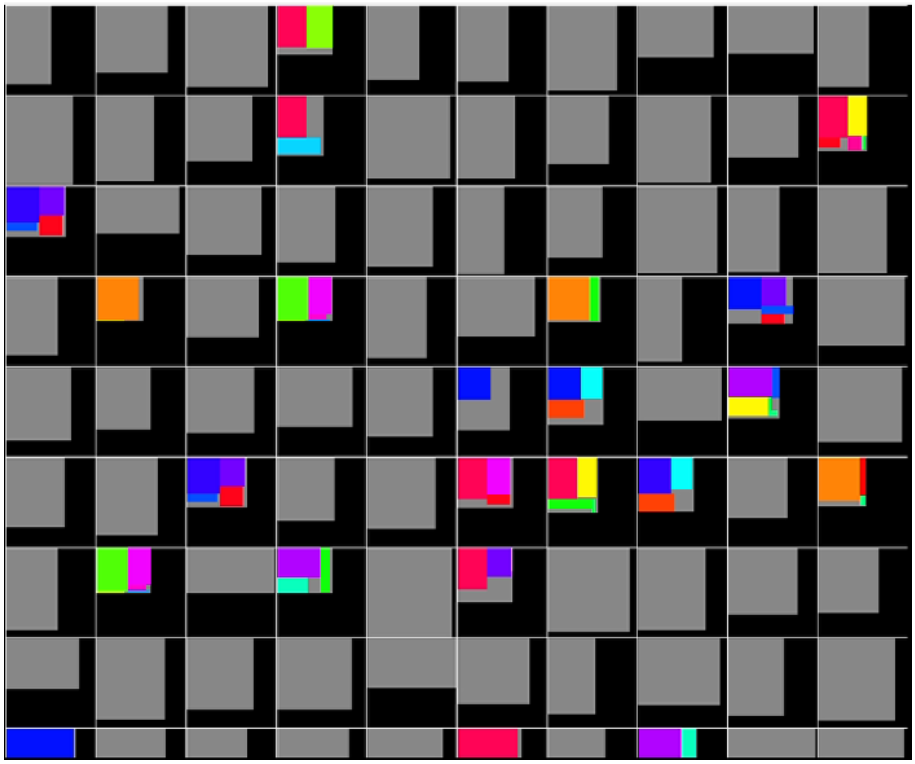


Figure 10: Kết quả sau khi đặt tất cả các products vào stocks (test 1)

Kết quả các chỉ số đánh giá thuật toán như sau:

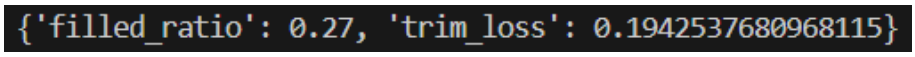


Figure 11: Kết quả hai chỉ số đánh giá của giải thuật trên tập dữ liệu 1

Nhìn chung giải thuật cho hiệu quả tốt hơn Genetic tuy nhiên nó không thể tối ưu cực bộ, bằng chứng là chỉ có 1 procducts được đặt vào stock_idx thứ 45

2.2.4 Tập dữ liệu product kiểm thử 2

Product Index	Width	Height	Quantity
1	28	18	2
2	37	25	2
3	1	43	2
4	42	40	5
5	13	17	2
6	37	47	4
7	26	34	7
8	46	41	10
9	32	31	2
10	28	30	7
11	43	31	6
12	36	4	3
13	14	38	3
14	3	40	10
15	19	15	3
16	28	49	6
17	6	14	8
18	40	38	9
19	6	24	9
20	33	22	5
21	2	13	5
22	42	25	4
23	23	9	5
24	34	42	10
25	8	49	10

Table 4: Tập dữ liệu product 2

2.2.4.a Kết quả thuật toán Genetic

Tương tự như trên, kết quả chưa thật sự tối ưu dù thời gian chạy là rất lớn (10-15 phút).

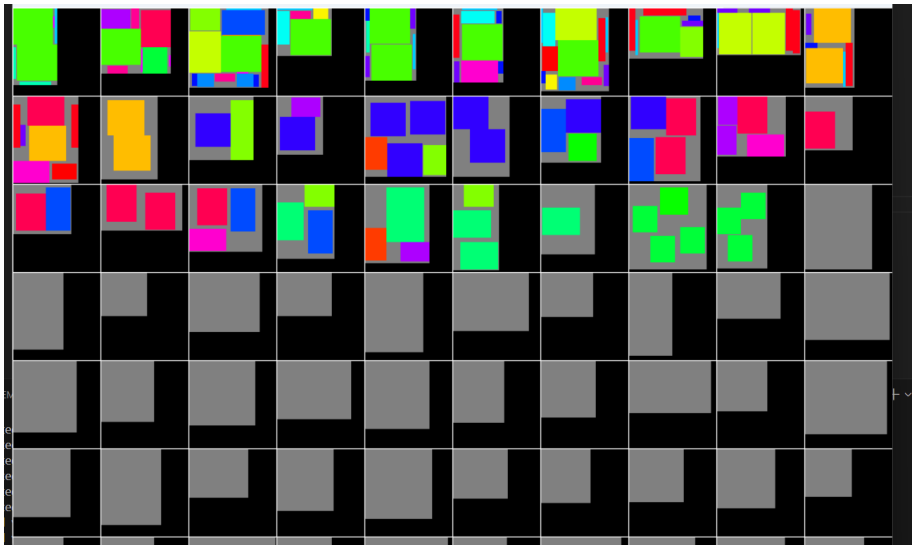


Figure 12: Kết quả thuật toán Genetic trên tập dữ liệu 2

Kết quả các chỉ số đánh giá thuật toán như sau:

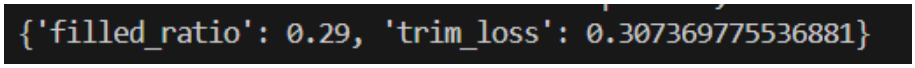


Figure 13: Kết quả hai chỉ số đánh giá của Genetic trên tập dữ liệu 2

Tuy nhiên, thông qua việc in thông tin mỗi 20 cá thể của 10 thế hệ vào một file text (thông qua thư việc logging), ta nhận thấy thuật toán đã tìm ra được cá thể tốt nhất (hay cấu hình tốt nhất) ngay từ giai đoạn khởi tạo quần thể, ta có thể quan sát điều đó thông qua hai ảnh dưới đây:

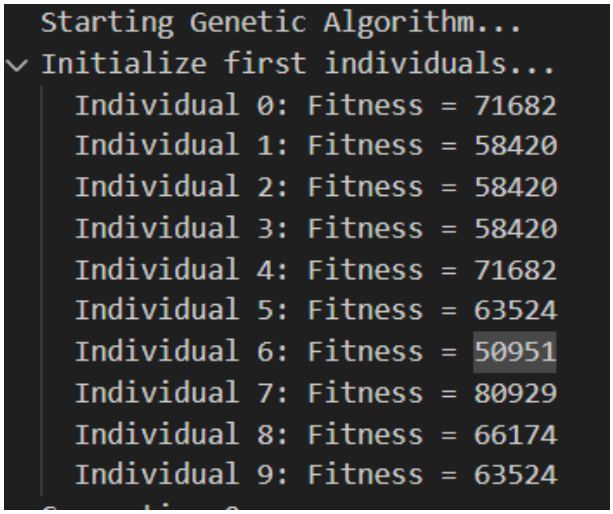


Figure 14: 10 cá thể ban đầu được khởi tạo (test 2)

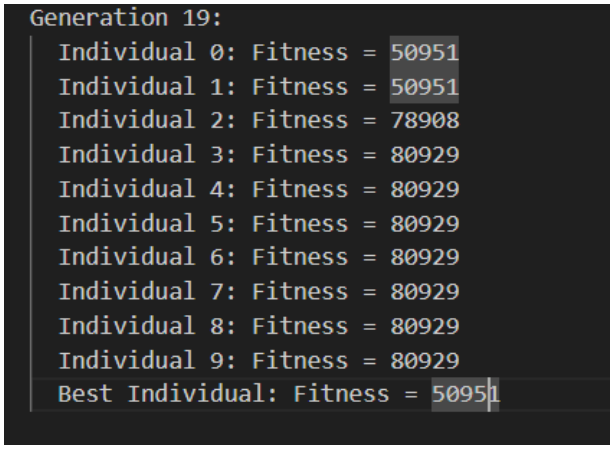


Figure 15: 10 cá thể trong thế hệ cuối cùng (test 2)

Việc lai ghép và đột biến đã không thể tạo ra cá thể tốt hơn. Do đó cá thể với điểm fitness thấp nhất (50951 đơn vị diện tích trống trên các stock đã dùng) đã được giữ lại qua 20 thế hệ thông qua cơ chế chọn lọc và giữ lại 2 cá thể ưu việt nhất của thế hệ hiện tại cho thế hệ sau.

2.2.4.b Kết quả Stock filled_rate optimize heuristic

Sau khi đặt tất cả các products và stocks thì kết quả như hình:

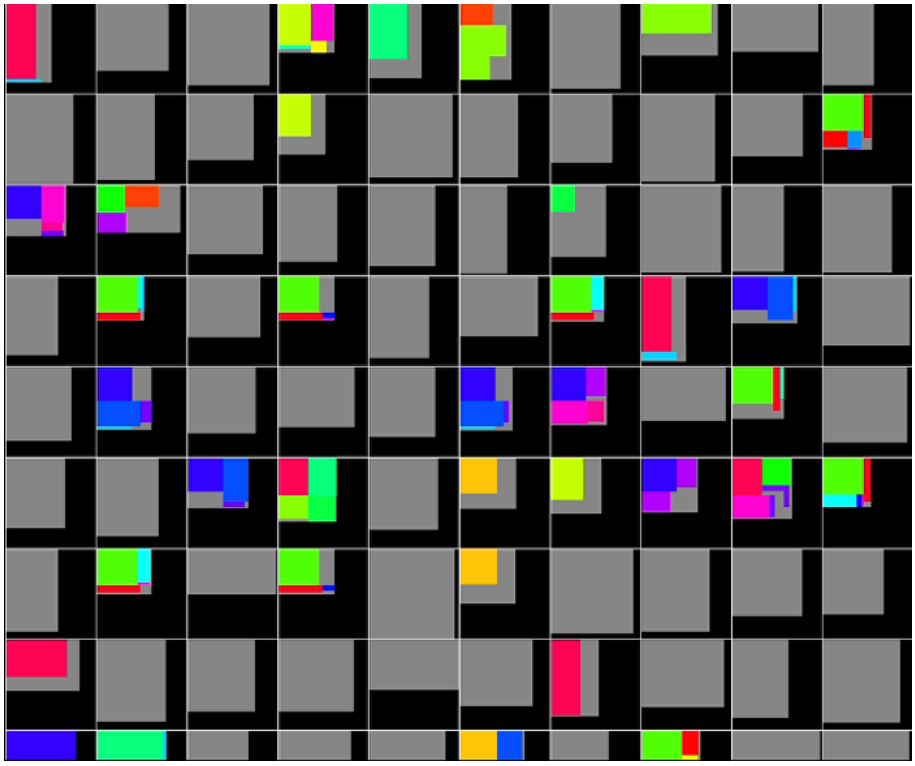


Figure 16: Kết quả sau khi đặt tất cả các products vào stocks (test 2)

Kết quả các chỉ số đánh giá thuật toán như sau:

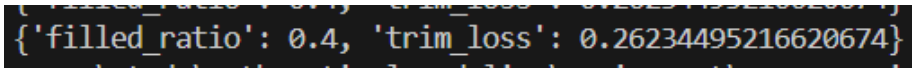


Figure 17: Kết quả hai chỉ số đánh giá của giải thuật trên tập dữ liệu 2

Có thể thấy qua 2 tập dữ liệu kiểm thử giải thuật trên đưa ra kết quả khá tối ưu cho bài toán Cutting Stock 2D, nó bỏ qua các stock có thể gây phí phạm để vào stock cần thiết để tối thiểu trim_loss nhất có thể.

2.3 Kết luận

Qua việc kiểm thử trên hai tập dữ liệu trên, thuật toán Genetic đã cho thấy sự thiếu hiệu quả trong việc tìm ra cấu hình tối ưu thông qua việc lai ghép và đột biến thông qua các thế hệ. Dù thời gian chạy là tương đối lớn. Trong khi đó Stock filled_rate optimize heuristic tỏ ra là một cách tiếp cận hiệu quả hơn khi cho ra kết quả tốt với thời gian chạy vừa phải.