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

MATHEMATICAL MODELING (CO2011)

Assignment

"Cutting stock problem"

Instructor(s): Nguyễn Văn Minh Mẫn, *Mahidol University*
Nguyễn An Khương, *CSE-HCMUT*
Lê Hồng Trang, *CSE-HCMUT*
Mai Xuân Toàn, *CSE-HCMUT*
Trần Hồng Tài, *CSE-HCMUT*
Nguyễn Quang Đức, *CSE-HCMUT*

Student(s): Lê Thị Kim Ngân - 2115391 (Group L06, Team 45).
Dương Khả Cơ - 2112952 (Group L06, Team 45).
Trương Mạnh Huy - 2311259 (Group L06, Team 45).
Hoàng Ngô Thiên Phúc - 2212612 (Group L05, Team 45).
Đặng Minh Nhật - 2212387 (Group L06, Team 45).

HO CHI MINH CITY, DECEMBER 2022

Contents

Member list & Workload

No.	Fullname	Student ID	Problems	% done
1	Lê Thị Kim Ngân	2115391	Question 4	%
2	Dương Khả Cơ	2112952	Question 1	%
3	Trương Mạnh Huy	2311259	Question 3	%
4	Hoàng Ngô Thiên Phúc	2212612	Question 2	%
5	Đặng Minh Nhật	2212387	Question 5	%

Table 1: Member list & workload

1 Mô hình hóa bài toán

Với n phần tử hình chữ nhật nhỏ 2DCSP sẽ cắt tất cả các phần tử trong các vật liệu hình chữ nhật lớn với mục tiêu là giảm thiểu số lượng vật liệu được sử dụng. Ký hiệu x và y lần lượt là chiều rộng và chiều dài của hình chữ nhật bao bọc. Dựa trên phương pháp của Chen et al có thể thiết lập một mô hình toán học với mục tiêu là tối thiểu hóa diện tích sử dụng (tức là Min xy) được đề cập trong mục 1.1. Trong 2DCSP, số lượng vật liệu tối thiểu có thể giảm chi phí sản xuất. Do đó, ở mục 1.2 sẽ giải thích cách xây dựng lại hai chương trình 2DCSP mới dựa trên mô hình ban đầu trong phần 1.1, các thuật ngữ bao gồm biến quyết định và tham số, được giới thiệu trong bảng 2 và bảng 3.

1.1 Bài toán cắt trong một vật liệu

Bài toán cắt trong một vật liệu, hay còn gọi là *bài toán phân bổ (assortment problem)*, yêu cầu cắt một tập hợp n hình chữ nhật nhỏ từ một tấm vật liệu hình chữ nhật sao cho diện tích sử dụng là tối thiểu. Điều kiện chính của bài toán là:

- Đảm bảo các hình không chồng lấn nhau
- Tất cả các hình chữ nhật phải nằm hoàn toàn trong giới hạn của tấm vật liệu.

Mô hình toán học có thể được thiết lập với **hàm mục tiêu**: Mô hình toán học có thể được thiết lập với **hàm mục tiêu**:

$$\text{Min } xy$$

Trong đó:

- x: Chiều rộng của tấm vật liệu hình chữ nhật bao bọc toàn bộ các hình chữ nhật
- y: Chiều dài của tấm vật liệu

Tham số và biến quyết định

Tham số	**Ý nghĩa**
n	Số lượng các hình chữ nhật cần cắt từ vật liệu
m	Số lượng vật liệu hình chữ nhật có kích thước giống nhau
(p_i, q_i)	Chiều rộng và chiều dài của vật thể hình chữ nhật i, với $i = 1, \ldots, n$
$(\overline{x}, \overline{y})$	$\overline{x}$ và $\overline{y}$ lượt là giới hạn trên của x và y. Biểu diễn chiều rộng và chiều dài tối đa của vật liệu hình chữ nhật.

Table 2: Tham số trong 2DCSP.

Biến	Ý nghĩa
(x_i, y_i)	Tọa độ góc dưới bên trái của vật thể hình chữ nhật i
(x, y)	Tọa độ góc trên bên phải của vật liệu hình chữ nhật
$(a_{i,j}, b_{i,j}, c_{i,j}, d_{i,j})$	Các biến nhị phân để mô tả điều kiện không chồng lấn giữa hai hình chữ nhật i và j
$(u_{i,j}, v_{i,j})$	Các biến nhị phân mô tả điều kiện không chồng lấn được xác định theo cách khác (xem Mô hình b).
s_i	Biến nhị phân chỉ định hướng của vật thể hình chữ nhật i. $s_i = 1$ nếu chiều rộng song song với trục x
Y	Tổng chiều dài tích lũy của các vật liệu được sử dụng

Table 3: Biến quyết định trong 2DCSP.

Ràng buộc

1. **Không chồng lấn giữa các hình chữ nhật:**

$$x_i + p_i s_i + q_i(1 - s_i) \le x_j + x(1 - a_{ij}), \quad \forall i, j \tag{1}$$
$$x_j + p_j s_j + q_j(1 - s_j) \le x_i + x(1 - b_{ij}), \quad \forall i, j \tag{2}$$
$$y_i + q_i s_i + p_i(1 - s_i) \le y_j + y(1 - c_{ij}), \quad \forall i, j \tag{3}$$
$$y_j + q_j s_j + p_j(1 - s_j) \le y_i + y(1 - d_{ij}), \quad \forall i, j \tag{4}$$

Đồng thời:

$$a_{ij} + b_{ij} + c_{ij} + d_{ij} = 1, \quad \forall i, j$$

2. **Ràng buộc biên:**

$$x_i + p_i s_i + q_i(1 - s_i) \le x, \quad \forall i \tag{5}$$
$$y_i + q_i s_i + p_i(1 - s_i) \le y, \quad \forall i \tag{6}$$

3. **Hướng của hình chữ nhật**

$$s_i = \begin{bmatrix} 1 & \text{nếu chiều rộng song song trục } x, \\ 0 & \text{nếu chiều rộng song song trục } y. \end{bmatrix}$$

1.2 Mô hình tổng quát cho bài toán cắt hai chiều

Trong bài toán cắt tổng quát, mục tiêu là tối thiểu hóa số lượng vật liệu cần sử dụng để cắt toàn bộ n hình chữ nhật. Các vật liệu có thể được sử dụng nhiều lần, với mục tiêu tối ưu hóa tổng chiều dài của chúng. **Hàm mục tiêu**

$$\text{Min } Y$$

Trong đó:

- Y: Tổng chiều dài các vật liệu.

Ràng buộc

1. **Không chồng lấn:**
$$a_{ij} + b_{ij} + c_{ij} + d_{ij} = 1, \quad \forall i, j$$
Các ràng buộc không chồng lấn tương tự như bài toán cắt một vật liệu.

2. **Phân bổ vật liệu:**
$$\sum_{k=1}^{m} Q_{i,k} = 1, \quad \forall i$$
Trong đó:
   - $Q_{i,k} = 1$: Nếu hình i thuộc về vật liệu k.
   - $Q_{i,k} = 0$: Nếu hình i không thuộc vật liệu k.

3. **Ràng buộc biên trên từng vật liệu:**
$$x_i + p_i s_i + q_i(1 - s_i) \le x, \quad \forall i, k \tag{7}$$
$$y_i + q_i s_i + p_i(1 - s_i) \le y, \quad \forall i, k \tag{8}$$

4. **Hướng và vị trí:**
$$y_i + q_i s_i + p_i(1 - s_i) \le k \cdot y + (1 - Q_{i,k}) \cdot M, \quad \forall i, k \tag{9}$$
$$y_i \ge (k - 1) \cdot y \cdot Q_{i,k}, \quad \forall i, k \tag{10}$$

2 Thuật toán

2.1 Thuật toán Divisible Optimization Policy

2.1.1 Giới thiệu

Thuật toán *Divisible Optimization Policy* là một heuristic được phát triển nhằm tối ưu hóa việc sử dụng không gian lưu trữ trong các kho chứa (stocks). Heuristic này tập trung vào việc kiểm tra tính "chia hết" giữa chiều cao kho và chiều cao của sản phẩm, từ đó đảm bảo việc điền đầy không gian một cách hiệu quả. Ý tưởng chính là sử dụng khả năng chia hết làm tiêu chí quyết định, kết hợp với việc ưu tiên các sản phẩm có diện tích lớn nhất để tăng tính hiệu quả trong quá trình sắp xếp.

2.1.2 Thuật toán

Thuật toán hoạt động dựa trên các bước sau:

1. **Lọc và sắp xếp sản phẩm:** Lọc các sản phẩm còn số lượng khả dụng, sau đó sắp xếp theo diện tích giảm dần để ưu tiên các sản phẩm lớn nhất.
2. **Duyệt qua các kho chứa:**
   - Xác định kích thước hiện tại của kho chứa (stock).
   - Kiểm tra từng sản phẩm để xác định liệu sản phẩm đó có thể đặt vào một cột (column) trong kho mà chiều cao của cột chia hết (hoặc gần chia hết theo một mức sai số `tolerance`) cho chiều cao sản phẩm.
   - Đối với mỗi sản phẩm, thử cả hai trạng thái: kích thước gốc và kích thước xoay (nếu có thể xoay).
3. **Điền không gian:** Khi tìm được sản phẩm phù hợp, thuật toán sẽ điền đầy không gian theo chiều dọc của cột bằng sản phẩm đó.
4. **Thuật toán dự phòng:** Nếu không tìm được sản phẩm phù hợp trong kho hiện tại, thuật toán chuyển sang sử dụng phương pháp dự phòng (ví dụ: Greedy heuristic).

Thuật toán sử dụng tham số `tolerance` để cho phép một mức độ sai số nhỏ trong việc kiểm tra chia hết, nhằm tăng tính linh hoạt khi xử lý các trường hợp thực tế.

2.1.3 Ưu điểm và nhược điểm

Ưu điểm:

- Tận dụng tốt không gian lưu trữ, giảm thiểu lãng phí không gian nhờ việc tối ưu hóa dựa trên tiêu chí chia hết.
- Ưu tiên các sản phẩm có diện tích lớn, giúp tối ưu hóa việc điền đầy không gian.
- Tính linh hoạt cao nhờ tham số `tolerance`, phù hợp với nhiều loại kịch bản sắp xếp thực tế.

Nhược điểm:

- Là một heuristic, thuật toán không đảm bảo tìm được lời giải tối ưu toàn cục trong mọi trường hợp.

- Việc kiểm tra tính chia hết và thử nghiệm các trạng thái xoay có thể tốn thời gian khi số lượng sản phẩm và kích thước kho lớn.
- Hiệu quả của thuật toán phụ thuộc nhiều vào giá trị `tolerance`; lựa chọn không phù hợp có thể làm giảm hiệu suất.

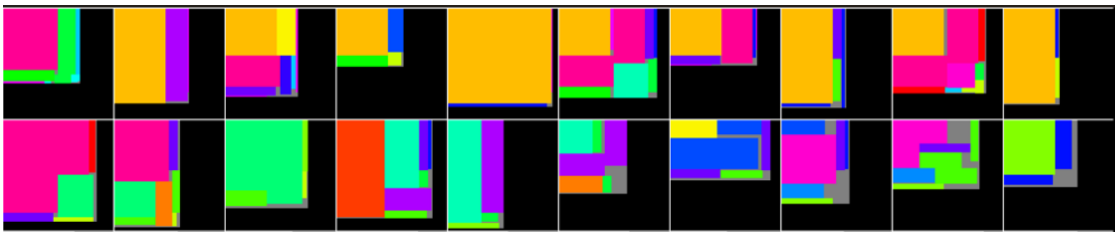

Figure 1: Mô phỏng Divisible Optimization

2.2 Thuật toán Greedy Rotateable Decreasing

2.2.1 Giới thiệu

Thuật toán *Greedy Rotateable Decreasing* là một heuristic dựa trên phương pháp *best fit*, được mở rộng thêm khả năng xoay sản phẩm và ưu tiên sắp xếp các sản phẩm theo diện tích giảm dần. Ý tưởng chính của thuật toán là giảm thiểu không gian trống bằng cách thử nghiệm nhiều hướng sắp xếp cho sản phẩm và chọn cách phù hợp nhất trong từng kho chứa (stock).

2.2.2 Thuật toán

Thuật toán được thực hiện theo các bước sau:

1. **Lọc và sắp xếp sản phẩm:** Chỉ giữ lại các sản phẩm còn số lượng khả dụng và sắp xếp chúng theo diện tích giảm dần.
2. **Duyệt qua các kho chứa:**
   - Với mỗi kho, xác định kích thước hiện tại.
   - Thử lần lượt từng sản phẩm (theo thứ tự diện tích giảm dần):
     - Kiểm tra khả năng đặt sản phẩm ở trạng thái kích thước gốc.
     - Nếu không thể, thử xoay sản phẩm và kiểm tra lại.
   - Nếu tìm được vị trí phù hợp, đặt sản phẩm tại vị trí đó và cập nhật trạng thái kho chứa.
3. **Xử lý trường hợp không phù hợp:** Nếu không thể đặt bất kỳ sản phẩm nào vào kho hiện tại, chuyển sang kho tiếp theo.
4. **Trả về kết quả:** Nếu không tìm được vị trí phù hợp trong tất cả các kho, trả về trạng thái không khả dụng.

2.2.3 Ưu điểm và nhược điểm

Ưu điểm:

- Đơn giản và dễ triển khai, với hiệu suất tốt trong các trường hợp thực tế.
- Khả năng xoay sản phẩm tăng tính linh hoạt trong việc sắp xếp.
- Ưu tiên các sản phẩm lớn giúp giảm lãng phí không gian ở các bước đầu.

Nhược điểm:

- Là heuristic, không đảm bảo lời giải tối ưu toàn cục.
- Hiệu suất phụ thuộc vào thứ tự sắp xếp ban đầu của sản phẩm.
- Có thể không hiệu quả nếu các sản phẩm có kích thước đa dạng và không đồng đều.

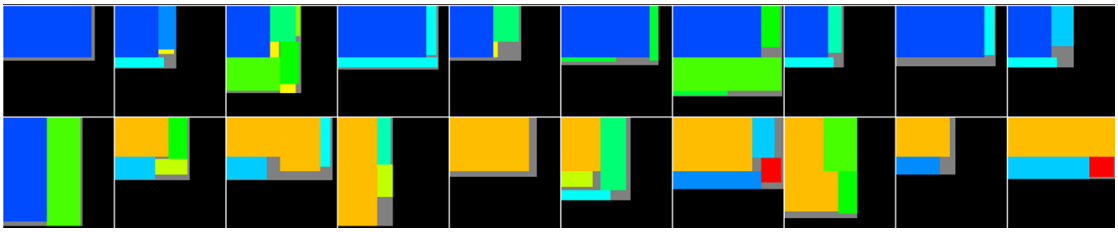

Figure 2: Mô phỏng Greedy Rotateable Decreasing

3 Bài toán cắt vật liệu hai chiều trong thực tế

3.1 Mô tả bài toán

Một công ty sản xuất nội thất chuyên sản xuất đồ nội từ các tấm gỗ nguyên liệu hình chữ nhật. Khách hàng đặt hàng các sản phẩm như bàn, ghế, tủ, và mỗi sản phẩm yêu cầu các tấm gỗ nhỏ hơn với kích thước cụ thể. Công ty muốn tối ưu hóa việc cắt các tấm nguyên liệu này để sản xuất đủ số lượng yêu cầu của khách hàng, đồng thời giảm thiểu lượng vật liệu bị lãng phí.

Thông tin bài toán thực tế

- **Nguồn nguyên liệu đầu vào**: Công ty nhập các tấm gỗ tiêu chuẩn có kích thước cố định là $240\,\text{cm} \times 120\,\text{cm}$. Đây là kích thước chuẩn phổ biến của tấm gỗ MDF dùng để làm đồ nội thất.
- **Đơn hàng**: Công ty nhận được đơn hàng với yêu cầu các mảnh gỗ có kích thước và số lượng được biểu diễn trong bảng dưới đây:

Loại sản phẩm	Số lượng cần cắt	Chiều dài (cm)	Chiều rộng (cm)
Mặt bàn	12	80	70
Cánh tủ	30	120	90
Kệ tủ	20	100	80

Table 4: Bảng thông tin về các loại sản phẩm và yêu cầu cắt

3.2 Bàn luận hướng giải quyết

Tổng quan về bài toán

Bài toán cắt vật liệu hai chiều trong sản xuất nội thất là một bài toán vô cùng cơ bản nhưng rất quan trọng trong ngành gỗ. Việc tối ưu tài nguyên sản xuất để tăng lợi nhuận và giảm chi phí là yếu tố thiết yếu trong bất kỳ doanh nghiệp sản xuất nào. Đặc biệt, trong ngành sản xuất đồ nội thất, nơi mà vật liệu như gỗ và các tấm MDF có giá trị cao, việc tận dụng tối đa diện tích của các tấm nguyên liệu không chỉ giúp tiết kiệm chi phí mà còn giảm thiểu lượng vật liệu thừa, từ đó nâng cao hiệu quả sản xuất.

Giải quyết bài toán

Các thuật toán để giải quyết bài toán về vật liệu 2 chiều đã được đề cập đến trong phần 2, vậy vấn đề này hoàn toàn có thể dựa trên thuật toán Greedy Rotateable Decreasing đã đề cập trước đó, lúc này ta xem tấm gỗ MDS chính là một kho chứa còn mặt bàn, cánh tủ , kệ tủ là những sản phẩm .

Lọc và sắp xếp sản phẩm

- Cánh tủ : diện tích $= 10800\,\text{cm}^2$, số lượng yêu cầu là 30.
- Kệ tủ : diện tích $= 8000\,\text{cm}^2$, số lượng yêu cầu là 20.
- Mặt bàn : diện tích $= 5600\,\text{cm}^2$, số lượng yêu cầu là 12.

Duyệt qua từng kho chứa

- Do chưa có kho chứa nên ta tạo kho chứa với kích thước ban đầu là 240 cm × 120 cm (diện tích 28800 cm^2).
- Đầu tiên, ta cho cánh tủ vào kho chứa: diện tích còn lại của kho chứa là 28800 – 10800 = 18000 cm^2, số lượng yêu cầu của cánh tủ giảm đi 1 (còn 29 cánh tủ).
- Tiếp tục, ta thêm kệ tủ vào kho chứa: diện tích còn lại là 18000 – 8000 = 10000 cm^2, số lượng yêu cầu của kệ tủ giảm đi 1 (còn 19 kệ tủ).
- ...
- Các sản phẩm tiếp theo được tiếp tục thêm vào kho chứa cho đến khi không còn đủ diện tích để chứa sản phẩm hoặc đạt yêu cầu của tất cả các sản phẩm.

Ta thực hiện liên tục các quá trình thêm, tối ưu bằng cách xoay và tạo kho chứa mới nếu kho chứa cũ không còn có thể chứa sản phẩm nào nữa, cho đến khi đã đạt yêu cầu của tất cả sản phẩm.

4 Đánh giá hiệu suất

Chương trình đánh giá kết quả của hai giải thuật **(Divisible Optimization Policy, Greedy Rotateable Decreasing)** được nhóm triển khai ở phần 2 sẽ thu thập các kết quả khi chạy 10 testcase random cho mỗi giải thuật đồng thời so sánh kết quả cắt của 2 giải thuật với việc so sánh trực tiếp cùng seed.

- Số liệu, hình ảnh của 2 giải thuật được thu thập bằng ngôn ngữ lập trình Python.

Nhóm tiến hành chạy thử ngẫu nhiên 10 testcase và thu thập các thông số để tiến hành đánh giá. Với mỗi bài toán, các thông số mà nhóm đánh giá gồm có:

- **filled_ratio(Tỉ lệ lấp đầy):** Đây là tỷ lệ phần diện tích đã được cắt thành các vật phẩm yêu cầu so với diện tích tổng thể của vật liệu ban đầu. Nó có thể được tính bằng cách chia diện tích đã sử dụng (diện tích các vật phẩm được cắt từ tấm vật liệu ban đầu) cho diện tích của vật liệu ban đầu.
- **total_filled_area(Tổng diện tích đã sử dụng):** Là tổng diện tích của tất cả các vật phẩm đã được cắt từ vật liệu ban đầu.
- **total_available_area(Tổng diện tích có sẵn):** Là tổng diện tích của vật liệu dùng để cắt thành các vật phẩm. Diện tích đáp ứng cho việc cắt các vật liệu.
- **num_stocks_used(Số lượng vật liệu sử dụng):** Là số lượng các vật liệu đã được sử dụng trong quá trình cắt. Vật liệu có diện tích, kích thước cố định, nhóm cần bao nhiêu vật liệu để đáp ứng số lượng vật phẩm cần cắt.
- **average_filled_ratio(Tỷ lệ lấp đầy trung bình):** Đây là tỷ lệ trung bình của diện tích đã được sử dụng trên tổng diện tích có sẵn của tất cả các vật liệu đã được sử dụng.
- **time(Thời gian):** Thời gian để giải quyết bài toán cắt.

Dưới đây là bảng kết quả thu được khi chạy các testcase bằng 2 thuật toán **Divisible Optimization Polic** và **Greedy Rotateable Decreasing**.

filled_ratio	total_filled_area	total_available_area	num_stocks_used
0.919498284640 43	126237	137289	27
0.7762219959266803	42686	54992	9
0.8815643412176586	72268	81977	16
0.9470198675496688	20878	22046	4
0.9513398068795758	105124	110501	20
0.8105481457384516	39866	49184	9
0.7802623091808213	18145	23255	3
0.9348419000038235	146701	156926	27
0.9288712225952992	150492	162016	28
0.8846781845060788	80189	90642	15

Table 5: Kết quả khi chạy random 10 testcase với thuật toán Divisible Optimization Policy

Thời gian chạy hết 10 testcase trên là Time taken: 2781.720254421234 seconds.

filled_ratio	total_filled_area	total_available_area	num_stocks_used
0.862315892261372	84776	98312	17
0.8758176315532559	53826	61458	11
0.9366230565036026	19759	21096	3
0.871816134846731	151390	173649	34
0.8889738999884513	123162	138544	26
0.8592321755027422	34780	40478	6
0.8681107305631102	162473	187157	33
0.9458535565155307	131153	138661	25
0.8492618038298494	58098	68410	14
0.8908756621530995	118229	132711	24

Table 6: Kết quả khi chạy random 10 testcase với thuật toán Greedy Rotateable Decreasing

Thời gian chạy hết 10 testcase trên là Time taken: 1387.7623329162598 seconds.

Ngoài ra, để có được một cái nhìn tổng quan hơn về cả hai thuật toán, nhóm tiến hành kiểm thử cùng seed với cả hai thuật toá, dưới đây là các kết quả mà nhóm thu được:

Các yếu tố so sánh	Divisible Optimization Policy	Greedy Rotateable Decreasing
filled_ratio	0.8864717756470334	0.8864717756470334
total_filled_area	49185	49185
total_available_area	55484	55484
num_stocks_used	12	12

Table 7: So sánh kết quả thu được khi chạy 2 thuật toán với seed 104

Các yếu tố so sánh	Divisible Optimization Policy	Greedy Rotateable Decreasing
filled_ratio	0.7455966805334245	0.78319478768816
total_filled_area	69720	69720
total_available_area	93509	93509
num_stocks_used	17	16

Table 8: So sánh kết quả thu được khi chạy 2 thuật toán với seed 105

Các yếu tố so sánh	Divisible Optimization Policy	Greedy Rotateable Decreasing
filled_ratio	0.8953767373811266	0.8953767373811266
total_filled_area	122398	122398
total_available_area	136700	136700
num_stocks_used	24	24

Table 9: So sánh kết quả thu được khi chạy 2 thuật toán với seed 241

1. So sánh kết quả từ 2 thuật toán: Bảng 5 và Bảng 6 cho thấy các kết quả thu được từ 10 testcase của hai thuật toán Divisible Optimization Policy và Greedy Rotateable Decreasing. Dưới đây là các nhận xét:

   - Tỷ lệ lấp đầy (filled_ratio): Thuật toán Divisible Optimization Policy có tỷ lệ lấp đầy dao động từ 0.776 đến 0.951, với giá trị trung bình khá cao, đặc biệt là các thử nghiệm đạt tỷ lệ lấp đầy vượt trội như 0.951.

     Thuật toán Greedy Rotateable Decreasing có tỷ lệ lấp đầy dao động từ 0.849 đến 0.945, giá trị này khá ổn định và không có giá trị cực cao như thuật toán trước. Tuy nhiên, tỷ lệ lấp đầy trung bình của Thuật toán 2 vẫn nằm trong khoảng tương đối cao, dao động từ 0.859 đến 0.945.
   - Tổng diện tích đã sử dụng (total_filled_area) và tổng diện tích có sẵn (total_available_area): Cả hai thuật toán đều có diện tích đã sử dụng và diện tích có sẵn tương đối tương

đồng trong các testcase, với các giá trị diện tích có sẵn (total_available_area) dao động từ 22,046 đến 220,046 cho cả hai thuật toán.

- Số lượng vật liệu sử dụng (num_stocks_used): Thuật toán Divisible Optimization Policy có xu hướng sử dụng nhiều vật liệu hơn, với số kho hàng dao động từ 3 đến 28.

  Thuật toán Greedy Rotateable Decreasing có xu hướng sử dụng ít kho hàng hơn, từ 3 đến 34 kho. Điều này có thể chỉ ra rằng thuật toán này có khả năng phân bổ kho hàng tốt hơn và sử dụng tài nguyên một cách tối ưu hơn.
- Thời gian thực thi (Time taken): Thuật toán Divisible Optimization Policy mất 2781.72 giây để chạy 10 testcase, trong khi Greedy Rotateable Decreasing chỉ mất 1387.76 giây. Điều này cho thấy thuật toán Greedy Rotateable Decreasing có thời gian thực thi nhanh hơn rất nhiều so với Divisible Optimization Policy.

2. So sánh kết quả khi chạy cùng một seed: Bảng 7, Bảng 8 và Bảng 9 cho thấy kết quả khi hai thuật toán được chạy với các giá trị seed cố định. Các nhận xét chính từ ba bảng này là:

   - Tỷ lệ lấp đầy (filled_ratio): Với seed 104, hai thuật toán cho kết quả filled_ratio bằng nhau (0.886), cho thấy chúng đạt kết quả tương đương về việc tối ưu hóa diện tích.

     Với seed 105, thuật toán Greedy Rotateable Decreasing có tỷ lệ lấp đầy cao hơn (0.783 so với 0.746), cho thấy thuật toán này hiệu quả hơn một chút trong việc sử dụng diện tích.

     Với seed 241, hai thuật toán có tỷ lệ lấp đầy hoàn toàn giống nhau (0.895), cho thấy sự tương đồng về hiệu quả sử dụng diện tích.
   - Tổng diện tích đã sử dụng và tổng diện tích có sẵn: Các giá trị total_filled_area và total_available_area đều giống nhau giữa hai thuật toán cho mỗi seed, cho thấy cả hai thuật toán đều giải quyết cùng một bài toán với cùng một lượng vật liệu và diện tích.
   - Số lượng vật liệu sử dụng (num_stocks_used): Với seed 104, cả hai thuật toán sử dụng cùng 12 vật liệu.

     Với seed 105, thuật toán Greedy Rotateable Decreasing sử dụng ít kho hàng hơn (16 kho so với 17 kho).

     Với seed 241, cả hai thuật toán đều sử dụng 24 kho hàng, cho thấy sự tương đương trong việc phân bổ kho hàng.

3. Kết luận:

   - Về thời gian thực thi: Thuật toán Greedy Rotateable Decreasing có ưu thế vượt trội về thời gian, chạy nhanh hơn khoảng 2 lần so với Divisible Optimization Policy. Điều này có thể quan trọng khi cần tối ưu hóa tốc độ tính toán trong các ứng dụng yêu cầu thời gian xử lý ngắn.
   - Về tỷ lệ lấp đầy: Cả hai thuật toán cho thấy kết quả khá ổn định và hiệu quả về tỷ lệ lấp đầy, với Divisible Optimization Policy có thể đạt được tỷ lệ cao hơn trong một số trường hợp. Tuy nhiên, sự khác biệt về tỷ lệ lấp đầy không lớn, cho thấy cả hai thuật toán đều có thể tối ưu hóa diện tích sử dụng tốt.
   - Về số lượng vật liệu sử dụng: Thuật toán Greedy Rotateable Decreasing có xu hướng sử dụng ít kho hơn trong các thử nghiệm, điều này có thể chỉ ra khả năng phân bổ tài nguyên tốt hơn.

5 Áp dụng thuật toán học tăng cường (Reinforcement Learning) giải bài toán cắt tấm hai chiều

Học tăng cường (Reinforcement Learning - RL) là một lĩnh vực của trí tuệ nhân tạo, cho phép một tác nhân (người ra quyết định) hành động thông minh hơn thông qua quá trình học hỏi. Trong quá trình hoàn thiện, nhóm nghiên cứu đã chọn hai thuật toán:

5.1 Giải thuật học tăng cường Q-Learning

5.1.1 Giới thiệu

Q-Learning là một thuật toán trong học tăng cường (Reinforcement Learning - RL) và thuộc loại học không giám sát (unsupervised learning). Phương pháp này được sử dụng để giải các bài toán quyết định theo chuỗi, trong đó tác nhân (agent) tương tác trực tiếp với môi trường nhằm tối ưu hóa tổng phần thưởng (reward) thu được trong suốt quá trình hoạt động.

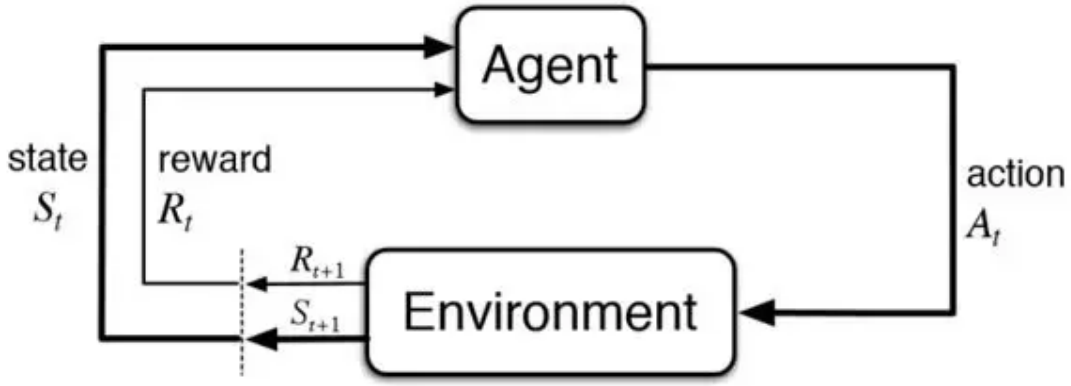

Figure 3: Q-Learning

5.1.2 Thuật ngữ cơ bản trong Q-Learning

- **Agent (Tác nhân):** Chủ thể thực hiện các quyết định, chẳng hạn như một chương trình máy tính hoặc một robot.
- **Environment (Môi trường):** Không gian nơi tác nhân hoạt động và tương tác, có thể là môi trường thực tế, một mô phỏng, hoặc một trò chơi.
- **State (Trạng thái):** Thể hiện tình huống hiện tại của môi trường mà tác nhân có thể nhận thức. Ví dụ, trạng thái của một robot có thể là tọa độ hiện tại của nó.
- **Action (Hành động):** Các lựa chọn mà tác nhân có thể thực hiện ở mỗi trạng thái, chẳng hạn như tiến lên, rẽ trái, hoặc dừng lại.
- **Reward (Phần thưởng):** Giá trị nhận được khi tác nhân thực hiện một hành động trong một trạng thái cụ thể, phản ánh mức độ hiệu quả của hành động đó.

- **Policy (Chính sách):** Chiến lược mà tác nhân áp dụng để quyết định hành động tối ưu trong từng trạng thái.

5.1.3 Ý tưởng của Q-Learning

Q-Learning sử dụng một cấu trúc dữ liệu gọi là **Q-Table**.

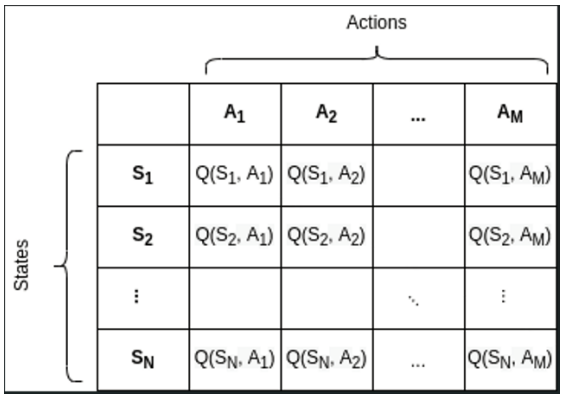

Figure 4: Q-Table

Trong đó mỗi phần tử đại diện cho một giá trị Q (Q-Value). Giá trị Q tương ứng với mỗi cặp trạng thái và hành động, được ký hiệu là $Q(s,a)$. Nó ước lượng mức độ hiệu quả của việc thực hiện hành động a tại trạng thái s. Công thức cập nhật giá trị Q được biểu diễn như sau:

$$Q[s,a] \leftarrow Q[s,a] + \alpha \cdot \left(r + \gamma \cdot \max_{a'} Q[s',a'] - Q[s,a]\right)$$

Trong đó:

- $Q[s,a]$: Giá trị Q hiện tại của trạng thái s và hành động a, đại diện cho kỳ vọng tích lũy phần thưởng khi bắt đầu tại trạng thái s, thực hiện hành động a và sau đó tiếp tục tuân theo chính sách tối ưu.
- α (*learning rate*): Tốc độ học, một tham số nằm trong khoảng $[0,1]$, điều chỉnh mức độ thay đổi của giá trị Q khi cập nhật.
- r: Phần thưởng nhận được sau khi thực hiện hành động a tại trạng thái s.
- γ (*discount factor*): Hệ số chiết khấu, nằm trong khoảng $[0,1]$, xác định tầm quan trọng của phần thưởng trong tương lai. Khi γ lớn, tác nhân ưu tiên phần thưởng dài hạn hơn phần thưởng trước mắt.
- $\max_{a'} Q[s',a']$: Giá trị Q tối đa có thể đạt được tại trạng thái kế tiếp s' khi thực hiện bất kỳ hành động a' nào.
- $r + \gamma \cdot \max_{a'} Q[s',a'] - Q[s,a]$: Sai số *Temporal Difference (TD error)*, đo lường sự khác biệt giữa giá trị Q hiện tại và giá trị Q được điều chỉnh dựa trên kinh nghiệm mới.
- $Q[s,a] \leftarrow Q[s,a] + \alpha \cdot$ TD error: Phương trình này cập nhật giá trị Q, giúp nó tiến gần hơn đến giá trị chính xác.

Bằng cách sử dụng biểu thức trên, giá trị $Q[s,a]$ liên tục được điều chỉnh để tiệm cận giá trị thực, giúp tác nhân học được chính sách tối ưu qua thời gian.

5.1.4 Giải thuật Q-Learning

5.1.4.a Chính sách tham lam ϵ (ϵ-greedy policy)

Chính sách ϵ-greedy được sử dụng để cân bằng giữa hai mục tiêu chính trong học tăng cường:

- **Khai thác (Exploitation):** Lựa chọn hành động tốt nhất tại thời điểm hiện tại dựa trên giá trị $Q[s,a]$, nhằm tối ưu hóa phần thưởng.
- **Khám phá (Exploration):** Thử các hành động mới một cách ngẫu nhiên để thu thập thêm thông tin về môi trường, từ đó cải thiện chiến lược học.

Cách chọn hành động:

- Với xác suất $1-\epsilon$: Chọn hành động có giá trị $Q[s,a]$ cao nhất:
$$a = \arg\max_{a'} Q[s,a']$$
Trong đó, $\arg\max$ trả về hành động a tương ứng với giá trị $Q[s,a]$ lớn nhất.
- Với xác suất ϵ: Chọn một hành động ngẫu nhiên, không phụ thuộc vào giá trị $Q[s,a]$.

5.1.4.b Quy trình hoạt động của Q-Learning

Khởi tạo

- Tạo bảng giá trị $Q[s,a]$, với mọi giá trị ban đầu bằng 0 hoặc giá trị nhỏ ngẫu nhiên.
- Xác định các tham số: α (tốc độ học), γ (hệ số chiết khấu), và ϵ (tham số chính sách tham lam).

Lặp qua các episode

(a) Bắt đầu mỗi episode bằng cách khởi tạo trạng thái s.

(b) Lặp lại đến khi trạng thái mục tiêu hoặc trạng thái kết thúc được đạt:

- Chọn hành động a bằng chính sách ϵ-greedy:
  - Với xác suất ϵ: Chọn một hành động ngẫu nhiên.
  - Với xác suất $1-\epsilon$: Chọn hành động tối ưu dựa trên $Q[s,a]$.
- Thực hiện hành động a, quan sát phần thưởng r và trạng thái tiếp theo s'.
- Cập nhật giá trị $Q[s,a]$ theo công thức:
$$Q[s,a] \leftarrow Q[s,a] + \alpha \cdot \Big(r + \gamma \cdot \max_{a'} Q[s',a'] - Q[s,a]\Big)$$
- Chuyển trạng thái: $s \leftarrow s'$.

Kết thúc Lặp lại quy trình trên đến khi bảng Q hội tụ (giá trị thay đổi không đáng kể) hoặc đạt đến số lượng episode tối đa.

Trích xuất chính sách tối ưu Chính sách tối ưu được xác định bằng cách chọn hành động có giá trị $Q[s,a]$ cao nhất tại mỗi trạng thái s:

$$\pi(s) = \arg\max_{a} Q[s,a]$$

5.1.4.c Mã giả Q-Learning

Input: Không gian trạng thái và hành động, các tham số α, γ, ϵ
Output: Bảng giá trị $Q[s, a]$
Khởi tạo $Q[s, a] = 0$ với mọi s, a;
for *mỗi episode* **do**
  Khởi tạo trạng thái s;
  while *trạng thái s chưa kết thúc* **do**
    Chọn hành động a theo chính sách ϵ-greedy;
    Thực hiện hành động a, nhận phần thưởng r và trạng thái kế tiếp s';
    Cập nhật $Q[s, a]$:

$$Q[s, a] \leftarrow Q[s, a] + \alpha \cdot \left(r + \gamma \cdot \max_{a'} Q[s', a'] - Q[s, a] \right)$$

    Cập nhật trạng thái: $s \leftarrow s'$;
  end
end
return *Bảng giá trị $Q[s, a]$*

Algorithm 1: Thuật toán Q-Learning

5.1.4.d Ưu điểm và nhược điểm của Q-Learning

Ưu điểm:

- **Tính linh hoạt:** Q-Learning không yêu cầu thông tin về mô hình môi trường (transition model) mà chỉ cần tương tác với môi trường để học.
- **Đảm bảo hội tụ:** Với điều kiện tham số học (α, γ, ϵ) được thiết lập hợp lý và môi trường tuân theo các quy tắc Markov (MDP), thuật toán Q-Learning đảm bảo hội tụ đến chính sách tối ưu.
- **Khả năng tổng quát:** Có thể mở rộng để áp dụng trong nhiều bài toán khác nhau, bao gồm quản lý tài nguyên, robot học, và tối ưu hóa logistics.

Nhược điểm:

- **Khả năng mở rộng kém:** Trong môi trường có không gian trạng thái lớn hoặc liên tục, việc lưu trữ và cập nhật Q-Table trở nên không khả thi. Điều này đòi hỏi sử dụng các phương pháp gần đúng như Deep Q-Learning (DQN).
- **Tốc độ hội tụ chậm:** Nếu tham số không được điều chỉnh cẩn thận, thuật toán có thể mất rất nhiều thời gian để hội tụ.
- **Không tối ưu với exploration:** Chính sách ϵ-greedy có thể lãng phí thời gian vào việc chọn các hành động ngẫu nhiên không hiệu quả, đặc biệt khi ϵ cao.

5.2 Giải thuật SARSA (State-Action-Reward-State-Action)

5.2.1 Giới thiệu

Giải thuật SARSA (State-Action-Reward-State-Action) là một thuật toán học tăng cường (Reinforcement Learning - RL) trực tiếp học từ kinh nghiệm và tương tác với môi trường, với mục tiêu tìm kiếm một chính sách tối ưu trong việc lựa chọn hành động để đạt được phần thưởng tối đa. SARSA là một phương pháp học có giám sát (supervised learning) trong học tăng cường, trong đó mỗi hành động được thực hiện trong môi trường được theo dõi và cập nhật để cải thiện chiến lược quyết định của agent.

Khác với thuật toán Q-Learning, SARSA không chỉ học từ các trạng thái và hành động đã chọn mà còn cập nhật giá trị Q theo hành động tiếp theo được thực hiện. Điều này làm cho SARSA phản ứng nhanh hơn với môi trường có sự thay đổi, đặc biệt trong các tình huống cần sự thận trọng trong lựa chọn hành động, vì nó cân nhắc đến cả hành động thực tế thay vì chỉ đơn giản là dựa vào hành động tốt nhất.

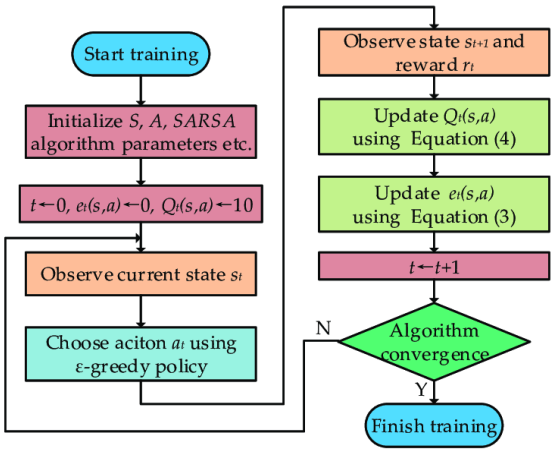

Figure 5: SARSA Training Process

5.2.2 Thuật ngữ cơ bản trong Q-Learning

- **Agent (Tác nhân):** Chủ thể thực hiện các quyết định, chẳng hạn như một chương trình máy tính hoặc một robot.
- **Environment (Môi trường):** Không gian nơi tác nhân hoạt động và tương tác, có thể là môi trường thực tế, một mô phỏng, hoặc một trò chơi.
- **State (Trạng thái):** Thể hiện tình huống hiện tại của môi trường mà tác nhân có thể nhận thức. Ví dụ, trạng thái của một robot có thể là tọa độ hiện tại của nó.
- **Action (Hành động):** Các lựa chọn mà tác nhân có thể thực hiện ở mỗi trạng thái, chẳng hạn như tiến lên, rẽ trái, hoặc dừng lại.
- **Reward (Phần thưởng):** Giá trị nhận được khi tác nhân thực hiện một hành động trong một trạng thái cụ thể, phản ánh mức độ hiệu quả của hành động đó.
- **Policy (Chính sách):** Chiến lược mà tác nhân áp dụng để quyết định hành động tối ưu trong từng trạng thái.

5.2.3 Ý tưởng của SARSA

SARSA là một thuật toán học tăng cường theo chính sách (on-policy), nghĩa là thuật toán học từ chính các hành động mà tác nhân thực hiện trong quá trình khám phá môi trường. Khác với Q-Learning, nơi giá trị Q được cập nhật dựa trên hành động tối ưu có thể thực hiện sau khi di chuyển đến trạng thái kế tiếp, trong SARSA, hành động tiếp theo a' được chọn thực tế bởi tác nhân và dùng để cập nhật giá trị Q.

Công thức cập nhật giá trị $Q(s,a)$ trong SARSA được biểu diễn như sau:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot \Big(r + \gamma \cdot Q(s',a') - Q(s,a)\Big)$$

Trong đó:

- $Q(s,a)$: Giá trị Q hiện tại của cặp trạng thái s và hành động a, phản ánh mức độ tốt khi thực hiện hành động a tại trạng thái s.
- α (*learning rate*): Tốc độ học, điều chỉnh mức độ ảnh hưởng của sai số trong việc cập nhật giá trị Q.
- r: Phần thưởng nhận được khi thực hiện hành động a tại trạng thái s.
- γ (*discount factor*): Hệ số chiết khấu, xác định mức độ ưu tiên cho các phần thưởng tương lai so với phần thưởng ngay lập tức.
- $Q(s',a')$: Giá trị Q tại trạng thái s' và hành động thực tế a' mà tác nhân chọn sau khi chuyển đến trạng thái kế tiếp s'.
- $r + \gamma \cdot Q(s',a') - Q(s,a)$: Sai số *Temporal Difference (TD error)*, đo lường sự khác biệt giữa giá trị kỳ vọng và giá trị thực tế khi tác nhân thực hiện hành động tại trạng thái s.
- $Q(s,a) \leftarrow Q(s,a) + \alpha \cdot$ TD error: Cập nhật giá trị Q, điều chỉnh giá trị của $Q(s,a)$ để tiến gần hơn đến giá trị thực tế dựa trên hành động tiếp theo thực tế.

Ở mỗi bước, tác nhân không chỉ dựa vào hành động tối ưu có thể thực hiện tại trạng thái tiếp theo mà còn thực hiện hành động thực tế đã chọn. Điều này giúp thuật toán SARSA học được chính sách tốt hơn trong môi trường không hoàn hảo và thay đổi dần dần theo kinh nghiệm thực tế của tác nhân.

5.2.4 Giải thuật SARSA

5.2.4.a Chính sách tham lam ϵ (ϵ-greedy policy)

Tương tự như Q-Leaning, chính sách ϵ-greedy trong SARSA được sử dụng để cân bằng giữa hai mục tiêu chính trong học tăng cường:

- **Khai thác (Exploitation):** Lựa chọn hành động tốt nhất tại thời điểm hiện tại dựa trên giá trị $Q[s,a]$, nhằm tối ưu hóa phần thưởng.
- **Khám phá (Exploration):** Thử các hành động mới một cách ngẫu nhiên để thu thập thêm thông tin về môi trường, từ đó cải thiện chiến lược học.

Cách chọn hành động:

- Với xác suất $1-\epsilon$: Chọn hành động có giá trị $Q[s,a]$ cao nhất:

$$a = \arg\max_{a'} Q[s,a']$$

  Trong đó, $\arg\max$ trả về hành động a tương ứng với giá trị $Q[s,a]$ lớn nhất.
- Với xác suất ϵ: Chọn một hành động ngẫu nhiên, không phụ thuộc vào giá trị $Q[s,a]$.

5.2.4.b Quy trình hoạt động của SARSA

Khởi tạo

- Tạo bảng giá trị $Q[s,a]$, với mọi giá trị ban đầu bằng 0 hoặc giá trị nhỏ ngẫu nhiên.
- Xác định các tham số: α (tốc độ học), γ (hệ số chiết khấu), và ϵ (tham số chính sách tham lam).

Lặp qua các episode

(a) Bắt đầu mỗi episode bằng cách khởi tạo trạng thái s.

(b) Lặp lại đến khi trạng thái mục tiêu hoặc trạng thái kết thúc được đạt:

- Chọn hành động a theo chính sách ϵ-greedy:
  - Với xác suất ϵ: Chọn một hành động ngẫu nhiên.
  - Với xác suất $1-\epsilon$: Chọn hành động tối ưu dựa trên $Q[s,a]$.
- Thực hiện hành động a, quan sát phần thưởng r và trạng thái tiếp theo s'.
- Chọn hành động tiếp theo a' tại trạng thái s' theo chính sách ϵ-greedy.
- Cập nhật giá trị $Q[s,a]$ theo công thức:

$$Q[s,a] \leftarrow Q[s,a] + \alpha \cdot \Big(r + \gamma \cdot Q[s',a'] - Q[s,a]\Big)$$

- Chuyển trạng thái: $s \leftarrow s'$ và hành động: $a \leftarrow a'$.

Kết thúc Lặp lại quy trình trên đến khi bảng Q hội tụ (giá trị thay đổi không đáng kể) hoặc đạt đến số lượng episode tối đa.

Trích xuất chính sách tối ưu Chính sách tối ưu được xác định bằng cách chọn hành động có giá trị $Q[s,a]$ cao nhất tại mỗi trạng thái s:

$$\pi(s) = \arg\max_{a} Q[s,a]$$

5.2.4.c Mã giả SARSA

Input: Không gian trạng thái và hành động, các tham số α, γ, ϵ
Output: Bảng giá trị $Q[s, a]$
Khởi tạo $Q[s, a] = 0$ với mọi s, a;
for *mỗi episode* **do**
  Khởi tạo trạng thái s;
  Chọn hành động a theo chính sách ϵ-greedy;
  while *trạng thái s chưa kết thúc* **do**
    Thực hiện hành động a, nhận phần thưởng r và trạng thái kế tiếp s';
    Chọn hành động tiếp theo a' theo chính sách ϵ-greedy;
    Cập nhật $Q[s, a]$:

$$Q[s, a] \leftarrow Q[s, a] + \alpha \cdot \Big(r + \gamma \cdot Q[s', a'] - Q[s, a]\Big)$$

    Cập nhật trạng thái: $s \leftarrow s'$ và hành động: $a \leftarrow a'$;
  end
end
return *Bảng giá trị* $Q[s, a]$

Algorithm 2: Thuật toán SARSA

5.2.4.d Ưu điểm và nhược điểm của SARSA

Ưu điểm:

- **Tính linh hoạt:** SARSA là thuật toán học tăng cường theo chính sách, vì vậy nó có thể học từ chính các hành động mà tác nhân thực hiện trong quá trình khám phá.
- **Phản ánh chính sách thực tế:** Vì SARSA cập nhật giá trị Q dựa trên hành động thực tế mà tác nhân chọn, nên nó học chính sách tối ưu trong môi trường thực tế.
- **Học chính sách an toàn hơn:** Bằng việc duy trì chính sách theo đúng hành động mà tác nhân thực hiện, SARSA có thể học chính sách an toàn hơn, đặc biệt trong các môi trường không ổn định.

Nhược điểm:

- **Khả năng hội tụ chậm:** Vì SARSA học theo chính sách thực tế thay vì hành động tối ưu, tốc độ hội tụ có thể chậm hơn so với Q-Learning trong một số môi trường.
- **Không tối ưu với exploration:** Chính sách ϵ-greedy có thể lãng phí thời gian vào việc chọn các hành động ngẫu nhiên không hiệu quả, đặc biệt khi ϵ cao.
- **Tính ổn định trong môi trường động:** SARSA có thể không học nhanh như Q-Learning trong các môi trường có sự thay đổi lớn, vì nó luôn học dựa trên các hành động thực tế thay vì hành động tối ưu có thể xảy ra.

5.3 So sánh giữa Q-Learning và SARSA

Cả hai thuật toán Q-Learning và SARSA đều là các thuật toán học tăng cường được sử dụng để tối ưu hóa chính sách hành động. Tuy nhiên, chúng khác biệt về cách tiếp cận

trong việc cập nhật giá trị Q và ảnh hưởng đến việc học và hội tụ:

Thuật toán	Ưu điểm	Nhược điểm
Q-Learning	- Học chính sách tối ưu mà không cần biết hành động thực tế trong bước tiếp theo. - Đảm bảo hội tụ đến chính sách tối ưu trong môi trường MDP. - Có thể sử dụng trong các bài toán có không gian trạng thái lớn.	- Cần nhiều thời gian để hội tụ khi không gian trạng thái lớn. - Chính sách học được có thể không ổn định trong môi trường động. - Cần nhiều dữ liệu huấn luyện và không luôn đảm bảo tối ưu trong thực tế.
SARSA	- Học chính sách dựa trên hành động thực tế được chọn trong quá trình học. - Thích hợp hơn trong môi trường động, vì nó điều chỉnh chính sách dựa trên hành động thực tế.	- Khó hội tụ đến chính sách tối ưu nếu lựa chọn hành động không hiệu quả. - Cần phải cập nhật chính sách liên tục khi môi trường thay đổi. - Có thể dẫn đến việc tối ưu hóa kém trong các môi trường có độ biến động cao.

Table 10: So sánh ưu và nhược điểm giữa Q-Learning và SARSA

Lựa chọn thuật toán: Nếu bài toán yêu cầu học và tối ưu hóa chính sách hành động mà không có sự đảm bảo về hành động tốt nhất trong tương lai, Q-Learning sẽ là lựa chọn hợp lý. Q-Learning có thể học chính sách tối ưu mà không cần phải dựa vào hành động thực tế trong quá trình học. Tuy nhiên, nếu bài toán có tính chất động và yêu cầu điều chỉnh chính sách dựa trên hành động thực tế trong mỗi bước, SARSA có thể là một giải pháp hữu ích, vì nó học chính sách dựa vào hành động thực tế được chọn trong quá trình tương tác với môi trường.

5.4 Hiện thực bằng ngôn ngữ Python

Link Github: *https://github.com/MinzNhat/Reinforcement_Learning*

Hướng dẫn cài đặt:

(a) Clone dự án về máy:

```
git clone https://github.com/MinzNhat/Reinforcement_Learning
```

(b) Di chuyển vào thư mục dự án:

```
cd Reinforcement_Learning
```

(c) Cài đặt các package cần thiết:

```
pip install -r requirements.txt
```

(d) Tạo file .env tại thư mục gốc của dự án để chọn lựa policy_id và một số config khác:

```
POLICY_ID=1
NUM_EPISODES=100
LEARNING_RATE=0.1
DISCOUNT_FACTOR=0.9
EXPLORATION_RATE=0.2
```

(e) Tiến hành train model:

```
python main.py
```

Lưu ý:

- **POLICY_ID=1:** Model sẽ chạy mô hình SARSA.
- **POLICY_ID=2:** Model sẽ chạy mô hình Q-Learning.

5.5 Kết quả

1. Kết quả mô hình SARSA với 100 EPISODES:

```
--- Training Summary ---
Total rewards for each episode: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Filled ratios for each episode: [0.25, 0.27, 0.07, 0.28, 0.04, 0.11, 0.22, 0.09, 0.41, 0.28, 0.09, 0.33, 0.09, 0.03, 0.17, 0.18, 0.2, 0.37, 0.11, 0.15, 0.04, 0.01, 0.0
6, 0.03, 0.28, 0.34, 0.01, 0.37, 0.06, 0.17, 0.19, 0.27, 0.28, 0.2, 0.14, 0.19, 0.4, 0.08, 0.07, 0.18, 0.26, 0.05, 0.06, 0.26, 0.08, 0.27, 0.26, 0.45, 0.09, 0.15, 0.38
, 0.17, 0.19, 0.03, 0.34, 0.2, 0.08, 0.15, 0.08, 0.3, 0.35, 0.38, 0.01, 0.15, 0.09, 0.5, 0.14, 0.08, 0.07, 0.21, 0.05, 0.41, 0.19, 0.37, 0.05, 0.14, 0.01, 0.28, 0.01,
0.09, 0.32, 0.22, 0.26, 0.04, 0.19, 0.41, 0.48, 0.02, 0.19, 0.01, 0.09, 0.23, 0.39, 0.22, 0.11, 0.5, 0.3, 0.09, 0.26, 0.33]
Highest filled ratio: 0.50
```

Figure 6: Kết quả mô hình SARSA

2. Kết quả mô hình Q-Learning với 100 EPISODES:

```
--- Training Summary ---
Total rewards for each episode: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Filled ratios for each episode: [0.11, 0.18, 0.05, 0.16, 0.24, 0.24, 0.03, 0.23, 0.41, 0.05, 0.33, 0.18, 0.32, 0.11, 0.22, 0.12, 0.04, 0.17, 0.31, 0.36
, 0.17, 0.33, 0.24, 0.23, 0.28, 0.11, 0.2, 0.16, 0.25, 0.23, 0.2, 0.36, 0.18, 0.14, 0.4, 0.32, 0.31, 0.03, 0.36, 0.12, 0.1, 0.47, 0.2, 0.11, 0.2, 0.04,
 0.18, 0.16, 0.11, 0.42, 0.11, 0.03, 0.28, 0.03, 0.33, 0.38, 0.34, 0.1, 0.6, 0.16, 0.23, 0.4, 0.18, 0.56, 0.14, 0.04, 0.09, 0.27, 0.13, 0.1, 0.22, 0.11
, 0.19, 0.38, 0.42, 0.28, 0.35, 0.09, 0.29, 0.18, 0.08, 0.19, 0.39, 0.09, 0.07, 0.1, 0.18, 0.17, 0.34, 0.06, 0.04, 0.04, 0.45, 0.26, 0.26, 0.15, 0.08,
0.15, 0.18, 0.12]
Highest filled ratio: 0.60
```

Figure 7: Kết quả mô hình Q-Learning

3. Nhận xét:

- **SARSA:**
  - Mô hình cung cấp một giải pháp ổn định với tỷ lệ lấp đầy (filled ratio) duy trì ở mức trung bình.
  - Tỷ lệ lấp đầy cao nhất đạt **0.50**, thấp hơn so với Q-Learning, cho thấy SARSA có xu hướng ổn định hơn nhưng không đạt hiệu quả tối ưu cao nhất.

  - SARSA thực hiện cập nhật chính sách trên mỗi bước (state-action pair), do đó có khả năng thích ứng tốt trong các môi trường biến đổi liên tục.
  - Vì sử dụng hành động kế tiếp thực tế trong quá trình cập nhật giá trị, SARSA thường an toàn hơn trong các môi trường ngẫu nhiên, nhưng có thể dẫn đến tốc độ hội tụ chậm hơn so với Q-Learning.
- **Q-Learning:**
  - Mô hình đạt hiệu quả cao hơn trong việc tối đa hóa tỷ lệ lấp đầy, với tỷ lệ cao nhất đạt **0.60**.
  - Tuy nhiên, tỷ lệ lấp đầy dao động lớn (từ **0.05** đến **0.60**), cho thấy mô hình cần tối ưu thêm các tham số như *epsilon* hoặc *learning rate* để giảm thiểu sự bất ổn định.
  - Q-Learning sử dụng giá trị hành động tối ưu giả định cho trạng thái kế tiếp (max-Q), giúp tối đa hóa hiệu quả trong môi trường xác định.
  - Nhờ tập trung vào giá trị tối ưu, Q-Learning thường hội tụ nhanh hơn nhưng có nguy cơ bất ổn định trong môi trường nhiều ngẫu nhiên nếu không tinh chỉnh các tham số.
- **So sánh:**
  - *SARSA:* Phù hợp với các bài toán yêu cầu tính ổn định và tránh sự dao động lớn trong kết quả, nhưng không đạt hiệu quả tối ưu cao.
  - *Q-Learning:* Phù hợp với các bài toán phức tạp cần tìm giải pháp tối ưu, nhưng cần thời gian để hội tụ và điều chỉnh thêm các tham số để giảm biến động trong kết quả.
  - So với Q-Learning, SARSA mang tính thận trọng hơn vì nó xem xét rủi ro từ các hành động tiếp theo thực tế, thay vì giả định giá trị tối ưu.

5.6 Kết luận

Cả hai thuật toán SARSA và Q-Learning đều có thể ứng dụng hiệu quả trong việc giải quyết bài toán cắt tấm hai chiều. Việc lựa chọn thuật toán nào phụ thuộc vào yêu cầu cụ thể của bài toán:

- **SARSA:** Thích hợp cho các tình huống cần sự ổn định và tránh các dao động lớn trong kết quả. Ngoài ra, SARSA có thể hiệu quả hơn trong môi trường không chắc chắn.
- **Q-Learning:** Hữu ích khi cần tối ưu hóa tỷ lệ lấp đầy cao nhất, nhưng yêu cầu sự tinh chỉnh tham số để đạt kết quả ổn định hơn. Q-Learning thường phù hợp với các môi trường xác định hơn, nơi mà giá trị tối ưu có thể được dự đoán chính xác.