FPT Education
FPT UNIVERSITY

Group 5 - AI17C

Reinforcement Learning
CUTTING STOCK

Nguyen An Khuong

Lets Go!

Members

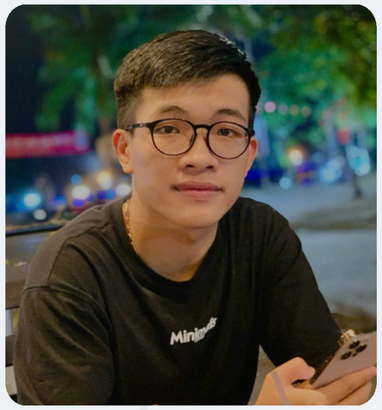
FPT Education
FPT UNIVERSITY

Bùi Đinh Thanh Danh

Nguyễn Minh Phát

Nguyễn Trọng Nghĩa

Phạm Gia Thịnh



Chế Minh Quang

Content

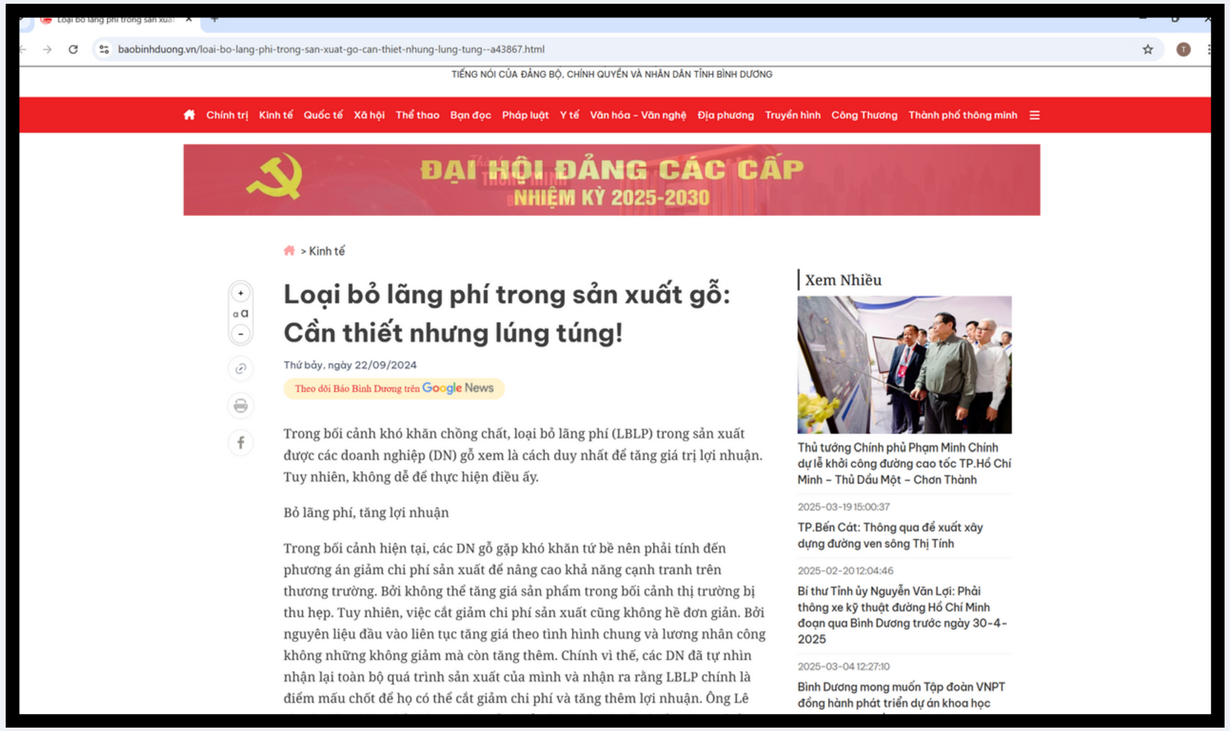
FPT Education
FPT UNIVERSITY

FPT Education
FPT UNIVERSITY

1. INTRODUCTION

CUTTING STOCK

1. Introduction



Source: https://baobinhduong.vn/loai-bo-lang-phi-trong-san-xuat-go-can-thiet-nhung-lung-tung--a43867.html

Source: https://laodong.vn/thi-truong/nganh-go-lao-dao-doanh-nghiep-gap-rut-tim-may-moc-tinh-gon-san-xuat-1207627.ldo

1. Introduction

FPT Education
FPT UNIVERSITY

- In the production of flooring, large wood blanks are often cut manually or according to fixed rules.
- The non-optimal arrangement of the cut pieces leads to significant material waste (the residual cannot be reused).

=>> This causes increased production costs, affecting price and sustainability.

FPT Education
FPT UNIVERSITY

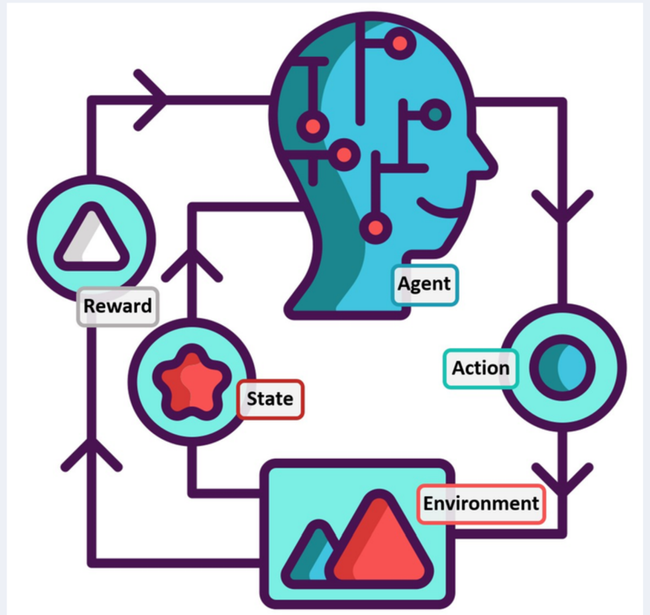
2. SOLUTION

CUTTING STOCK

2. Solution

- **Limitations of traditional methods:**
  - Cutting according to fixed rules
    → Not flexible, easy to create residuals that cannot be reused.
  - Manual sorting
    → Dependent on experience, inconsistent, difficult to expand.

- **Requirements**:
  - Need a solution that can learn from experience.
  - Need the ability to make decisions according to action sequences.
  - Need to adapt flexibly to input data.



2. Solution

Reinforcement Learning in the Cutting Stock problem of cutting wood flooring:

- Learning from experience:
  - Agent interacts with the wood cutting environment.
  - Learning through "trial - error - improvement".
  - Optimizing the excess wood area and the number of blanks used.

- Decision making according to the action chain:
  - RL optimizes the overall cutting action chain.
  - Considering long-term benefits, accepting short-term sacrifices.
  - Efficient when cutting large, multi-sized blanks.

2. Solution

FPT Education
FPT UNIVERSITY

Datasets

Data Structure:

- Number of columns: 4 columns (batch_id, type, width, height)
- Number of rows: 675 rows
- Data type: 2 types (stock, product)

Stocks

- Average size: 76.7 × 81.5 (width × height)
- Smallest size: 15 × 15
- Largest size: 150 × 150

Products

- Average size: 24.8 × 25.0 (width × height)
- Minimum size: 15 × 15
- Large size: 35 × 35

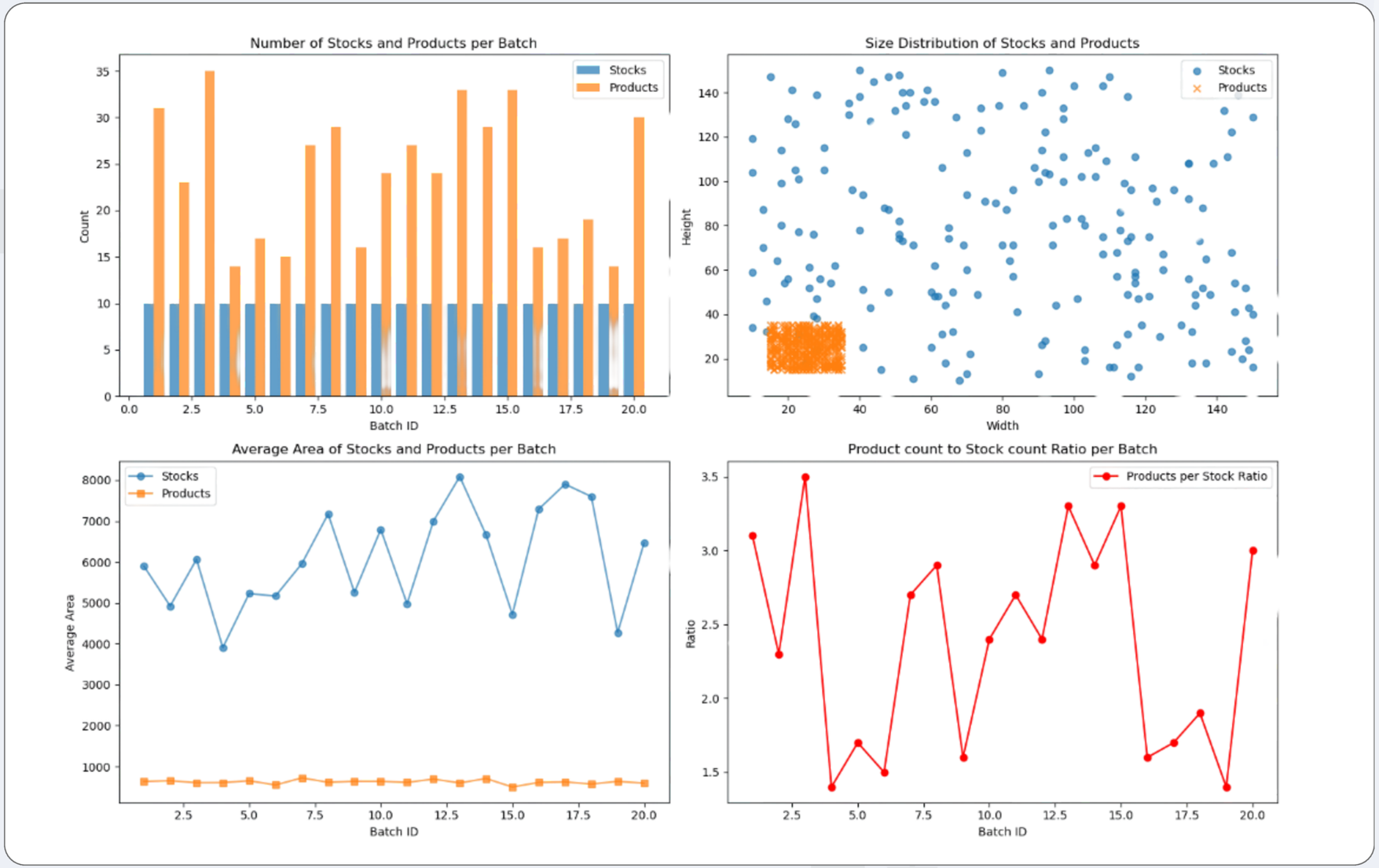
2. Solution

Datasets

- Total Stocks: 200 (10 stocks/batch × 20 batches)
- Total Products: 494
- Average Product/Stock Ratio: 2.47

Batch ID	Số lượng Stock	Số lượng Product	Tỷ lệ Product/Stock
1	10	41	4.1
2	10	23	2.3
3	10	45	4.5
4	10	14	1.4
5	10	17	1.7
6	10	15	1.5
7	10	27	2.7
8	10	30	3.0
9	10	16	1.6
10	10	24	2.4
11	10	27	2.7
12	10	24	2.4
13	10	33	3.3
14	10	29	2.9
15	10	33	3.3
16	10	16	1.6
17	10	17	1.7
18	10	19	1.9
19	10	14	1.4
20	10	30	3.0

2. Solution


Number of Stocks and Products per Batch
Stocks
Products
Count
Batch ID
Size Distribution of Stocks and Products
Stocks
Products
Height
Width
Average Area of Stocks and Products per Batch
Stocks
Products
Average Area
Batch ID
Product count to Stock count Ratio per Batch
Products per Stock Ratio
Ratio
Batch ID

FPT Education
FPT UNIVERSITY

3. TASKLISTS

CUTTING STOCK

3. Tasklists

FPT Education
FPT UNIVERSITY

TASK ID	TASK TITLE	PIC	Support	START DATE	END DATE	DURATION IN DAYS	PCT OF TASK COMPLETE
0	General work			21/01/2025	31/01/2025		
0.1	Clear requirements	Nghĩa,Danh, Phát, Thịnh, Quang	Mr. Khương	21/01/2025	22/01/2025		100%
0.2	Define solutions	Nghĩa,Danh, Phát, Thịnh, Quang		23/01/2025	25/01/2025		100%
0.3	Task list and timeline for project	Nghĩa,Danh, Phát, Thịnh, Quang		26/01/2025	27/01/2025		100%
0.4	Plan deatails	Nghĩa,Danh, Phát, Thịnh, Quang		28//01/2025	30/01/2025		100%
0.5	Research	Nghĩa,Danh, Phát, Thịnh, Quang		31/01/2025			
1	Data			1/2/2025	8/2/2025		
1.1	Build plan to collect data	Nghĩa,Danh, Phát, Thịnh, Quang		1/2/2025	1/2/2025		100%
1.2	Research dataset	Nghĩa,Danh, Phát, Thịnh, Quang		2/2/2025	6/2/2025		100%
1.3	Generate data	Nghĩa,Danh, Phát, Thịnh, Quang		2/2/2025	6/2/2025		100%
1.4	Review & Report	Nghĩa,Danh,Phát,Quang	Thịnh	7/2/2025	8/2/2025		100%
2	Environemnts			9/2/2025	16/2/2025		
2.1	Define the components of Reinforcement Learning	Nghĩa,Danh, Phát, Thịnh,Quang		9/2/2025	11/2/2025		100%
2.2	Build plan & setup environments (install library, algorithms,..)	Nghĩa,Danh, Phát, Thịnh, Quang		12/2/2025	13/2/2025		100%
2.3	Building a simulation environment - CuttingStockEnv	Nghĩa,Danh, Phát, Thịnh, Quang		14/02/2025	16/02/2025		100%
3	Algorithms			17/2/2025	15/3/2025		
3.1	Research and Select Algorithms (Q-Learning, Actor-Critic, Heuristic-based approaches)	Nghĩa,Danh, Phát, Thịnh, Quang		17/2/25	18/2/2025		100%
3.2	Implement Heuristic Policies (First Fit, Best Fit, Combined Approach)	Nghĩa,Danh,Thịnh	Quang,Phát	19/2/25	28/2/25		100%
3.3	Implement Q-Learning Algorithm	Nghĩa,Phát,Quang	Thịnh,Danh	19/2/25	28/2/25		100%
3.4	Implement Actor-Critic Algorithm	Thịnh,Danh	Nghĩa,Phát	19/2/25	28/2/25		100%
3.5	Train and Fine-tune RL Agents	Nghĩa,Danh, Phát, Thịnh, Quang		1/3/25	12/3/25		100%
3.6	Compare Performance between Heuristic and RL-based Approaches	Nghĩa,Danh, Phát, Thịnh, Quang		13/3/25	15/3/25		100%
4	Final Review & Report			16/03/2025	23/03/2025		
4.1	Check and review the entire project, create a doc file to describe in detail the operating principles and the team's problem solving process.	Nghĩa,Danh, Phát, Thịnh, Quang		16/03/2025	19/03/2025		100%
4.2	Prepare ppt file to report results	Nghĩa,Danh, Phát, Thịnh, Quang		20/03/2025	23/03/2025		100%

FPT Education
FPT UNIVERSITY

4. ENVIRONMENT

CUTTING STOCK

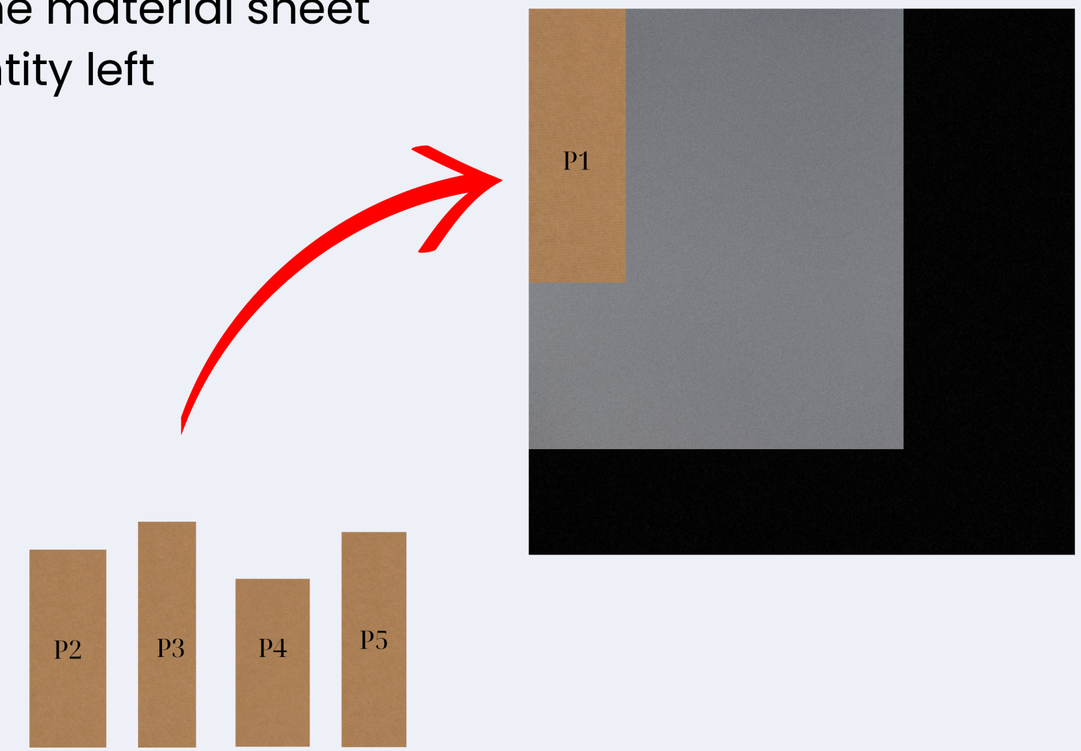
4. Environment

Constraint handling

- Each product must be placed in the available space only
- Product must be completely contained within the material sheet
- Only cut products that are listed and have quantity left

Sparse reward environment

- +1 only if all products have been cut
- 0 otherwise



4. Environment

FPT Education
FPT UNIVERSITY

Reset

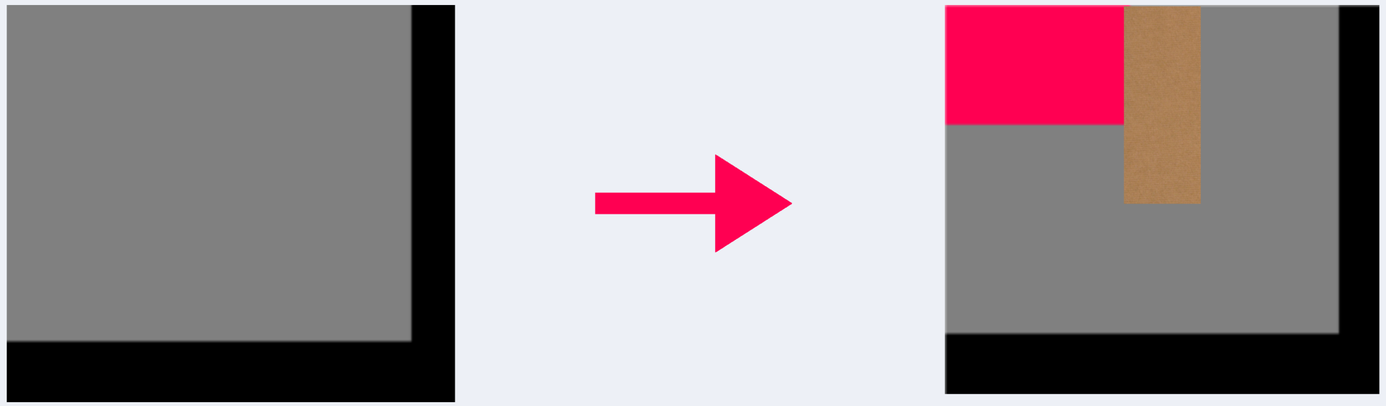
- Stocks: Large wooden boards with random sizes from min_w x min_h to max_w x max_h.
  - -2: Area outside the actual size of stocks
  - -1: Empty area, product can be placed
  - ≥0: Index of the product has been placed (will be updated in the step() method)
- Products: Each product is a dictionary with 2 information:
  - size: Size (width, height) of the product
  - quantity: Number of products to cut

4. Environment

FPT Education
FPT UNIVERSITY

Step

- Extract information from actions (stock_idx, size, position).
- Find the right product from the product inventory
- Processing placing the product on the material sheet (Make sure the product does not exceed the plate size.)
- Check end status and calculate rewards : Terminated = True, Rewards = 1



4. Environment

get_info

- Provide metrics to evaluate the effectiveness of the solution
- Measures the ratio of material panels used to total panels

  Filled_ratio = Used stock / Total stock

- Measure the percentage of empty (wasted) area on used material panels

  Trim_loss = Number of Empty Cells / Total Valid Cells

FPT Education
FPT UNIVERSITY

5. Algorithms

CUTTING STOCK

Heuristic Algorithm

Heuristic is an approximate algorithm, used to solve complex problems quickly without finding the absolute optimal solution.

✅ **Goal:**

- Find a “good enough” solution instead of “best”.
- Save computation time, especially such as Cutting Stock, Packing, Scheduling...

Heuristic in 2D Cutting-Stock Problem

Cut rectangular products (products) from material sheets (stock sheets) so that:

- Meet enough product demand
- Reduce material waste
- Optimize the amount of stock used

Heuristic Algorithm

First-Fit Algorithm

Browse each product (In descending order of size – from largest to smallest)

🔁 **How it works:**

For each product:

1. Browse the existing stocks, in order of priority
2. Within each stock:
   - Browse valid empty positions in order from (0,0) → right → down
   - As soon as you find a position with enough space, place the product there
   - Stop browsing → move to the next product
3. If there is no stock with enough space, open a new stock and place the product at position (0, 0)

Heuristic Algorithm

First-Fit Algorithm

Function First_fit_2DCSP:
  push one of each stock_type into the stock_list.
  for $stock_i$ **from** the largest **to** smallest area in stock_list:
    for $prod_j$ **from** the largest **to** smallest area in product_list:
      if $prod_j$ **can fit** into $stock_i$:
        place $prod_j$ **into** $stock_i$
      else:
        let "stockType" **be** the first stock_type that **fits** $prod_j$
        create a new stock of type "stockType"
        place $prod_j$ **into** the new stock
      if the textbfdemand for $prod_j$ is textbfmet:
        remove $prod_j$ **from** the item list

Source: Link

Heuristic Algorithm

Best-Fit Algorithm

Place the product in the stock that has the least amount of excess after placing it.

🔄 **How it works:**

1. Browse each product.
2. For each stock: Find all possible locations.
3. Estimate the excess area if placed.
4. Choose the best location (with the least excess) to place the product.
5. If there is no suitable stock, create a new stock.

⚡ **Features:**

- The solution quality is better than First-Fit.
- Takes more time to calculate.
- Suitable when material optimization is needed.

Heuristic Algorithm

Best-Fit Algorithm

```
Function Best_fit_2DCSP:
    push one of each stock_type into the stock_list.
    sort product_list from largest to smallest area

    for each prodⱼ in product_list:
        if quantity of prodⱼ ≤ 0:
            continue

        let min_waste be infinity
        let best_stock be NULL
        let best_position be NULL

        for each stockᵢ in stock_list:
            if stockᵢ size < prodⱼ size:
                continue

            for (x, y) in all positions can cut prodⱼ from stockᵢ:
                if region (x,y) to (x+prod_w, y+prod_h) in stockᵢ is empty:
                    let waste be remaining empty area after placing prodⱼ
                    if waste < min_waste:
                        min_waste = waste
                        best_stock = stockᵢ
                        best_position = (x, y)

        if best_stock is not NULL:
            place prodⱼ into best_stock at best_position
            update quantity of prodⱼ

        else:
            create a new stock
            place prodⱼ into the new stock at position (0,0)
            update quantity of prodⱼ
```

Heuristic Algorithm

FPT Education
FPT UNIVERSITY

Comparison : First-Fit vs Best-Fit

Criteria	🟥 First-Fit	🟩 Best-Fit
Core Strategy	Place the item into the **first stock** it fits	Place the item into the **stock with the least remaining space** after placement
Placement Scan Order	Scan from **(0, 0)** – left to right, top to bottom	Scan **all valid positions** in all available stocks
When to Stop Searching?	**Stop immediately** after finding the first valid position	**Evaluate all options**, then select the best one
Trim Loss Optimization	❌ Poor – may leave many unused areas	✅ Good – selects the position that leaves the least waste
Time Efficiency	✅ Very fast – minimal computation	❌ Slower – needs full layout evaluation
Stock Usage Strategy	Open new stock immediately if no fit	Try to **fully utilize** current stocks
Consolidation of Items in Few Stocks	❌ Low – often spreads items across many stocks	⚠️ Medium – better, but may still scatter items
Practical Application	Best when **speed** is more important than material usage	Best when **material efficiency** is a priority
Implementation Complexity	✅ Very simple to implement	⚠️ More complex – requires tracking remaining areas
Placement in New Stock	Always places at position `(0, 0)`	Also places at `(0, 0)` if no better option is found

Heuristic Algorithm

Combination Algorithm

Combine First-Fit + Best-Fit to achieve both efficiency and speed.

🔄 **How it works:**

- Phase 1: First-Fit Placement – quickly place all products.
- Phase 2: Best-Fit Refinement – optimize placement to reduce waste.
- Phase 3: Stock Merging (if applicable) – merge smaller stocks into larger stocks to reduce total stock.

🔥 **Goal:**

- Keep the same good time performance as First-Fit
- But still achieve the same material efficiency as Best-Fit
- Optimize both quality and speed → suitable for practical industrial applications.

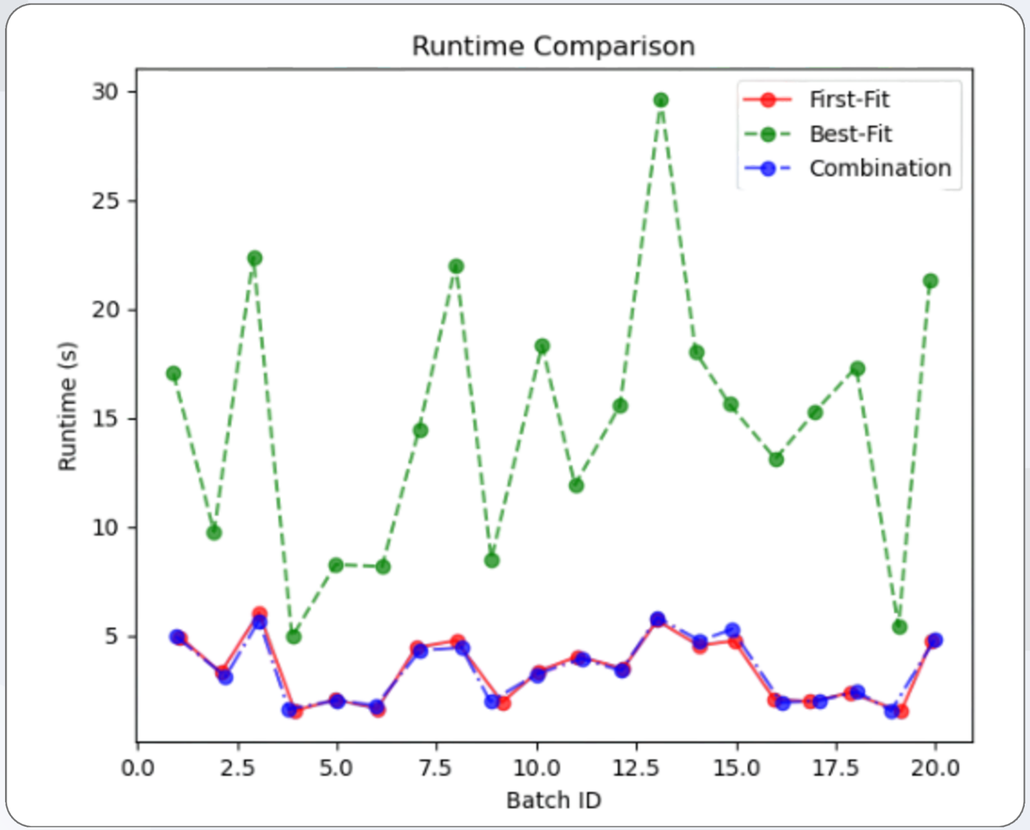
Heuristic Algorithm

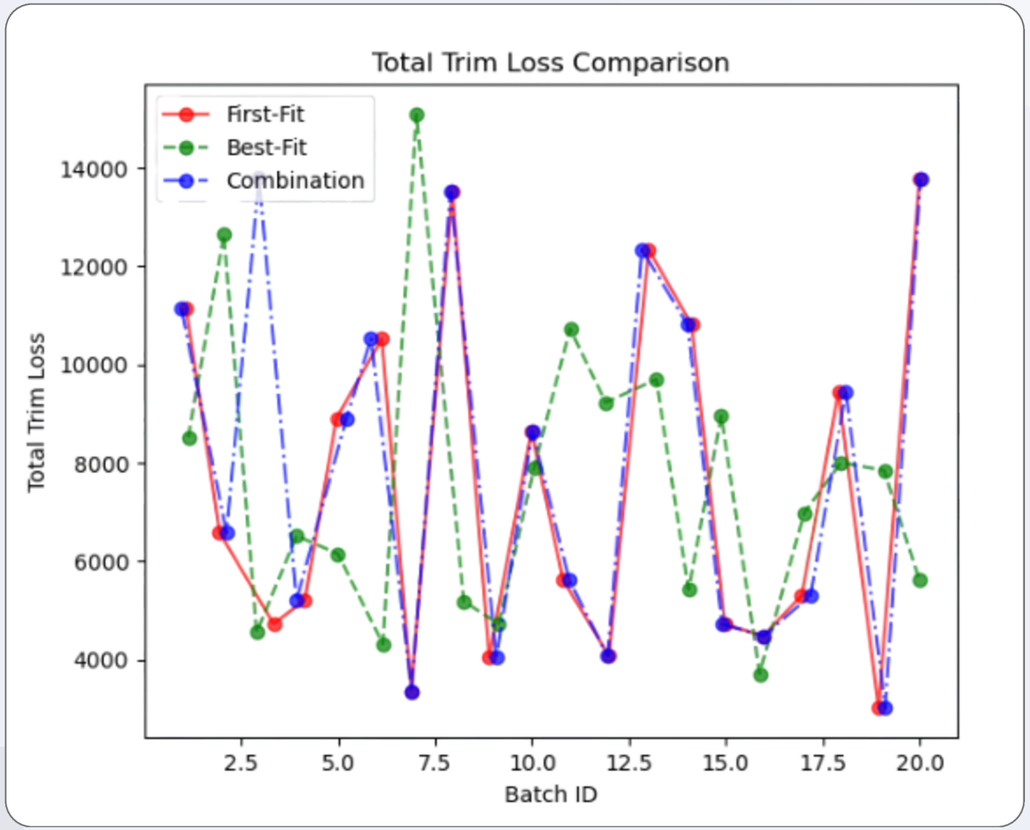
Combination Algorithm

Function Combination_2DCSP:
  for $stock_i$ **from** the largest **to** smallest area in stock list:
    for $prod_j$ **from** the largest **to** smallest area in product list:
      for (x, y) **in** all position can cut $prod_j$ **from** $stock_i$:
        if cut $prod_j$ at position (x, y):
          let S_{ij} be the area of the smallest rectangle contain cut area
      let (x_0, y_0) be the position **where** the value of S_{ij} is smallest
      cut the $prod_j$ **from** $stock_i$ **at** position (x_0, y_0)
  for $stock_i$ **from** the smallest **to** largest area in cut stock list:
    for $stock_j$ **from** the smallest **to** largest area in uncut stock list:
      if cut area of $stock_i$ is smaller than $stock_j$ **then**:
        move cut set of $stock_i$ **to** $stock_j$

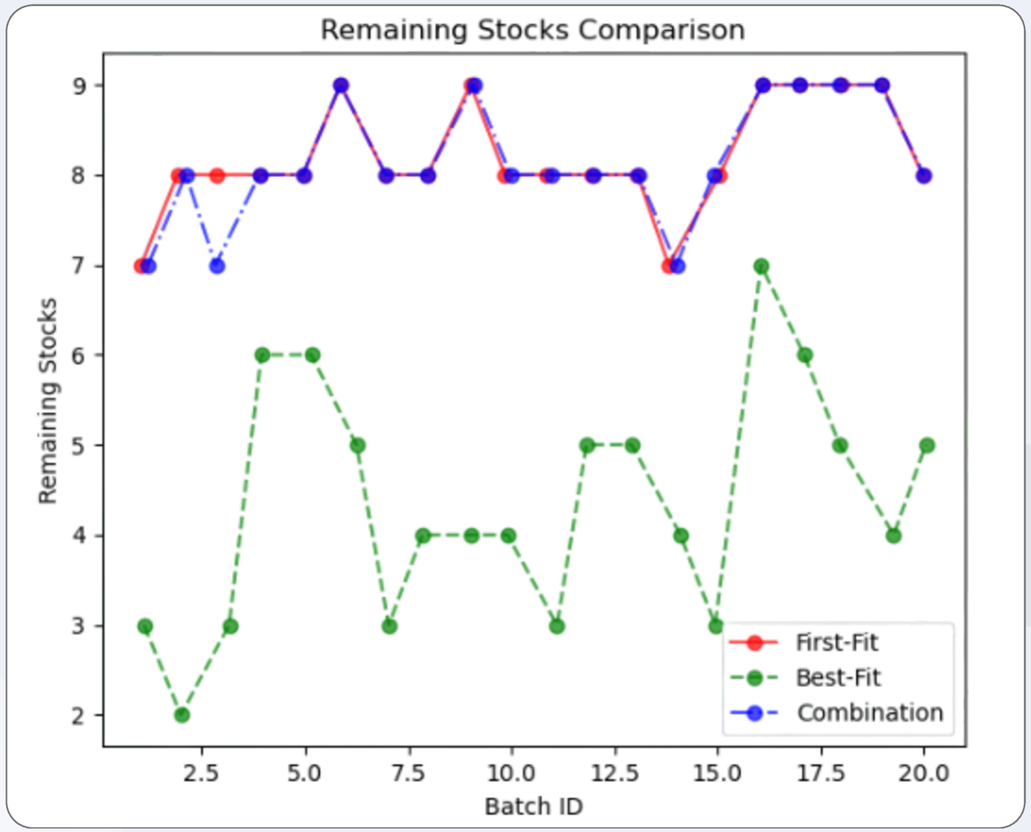
Source: Link

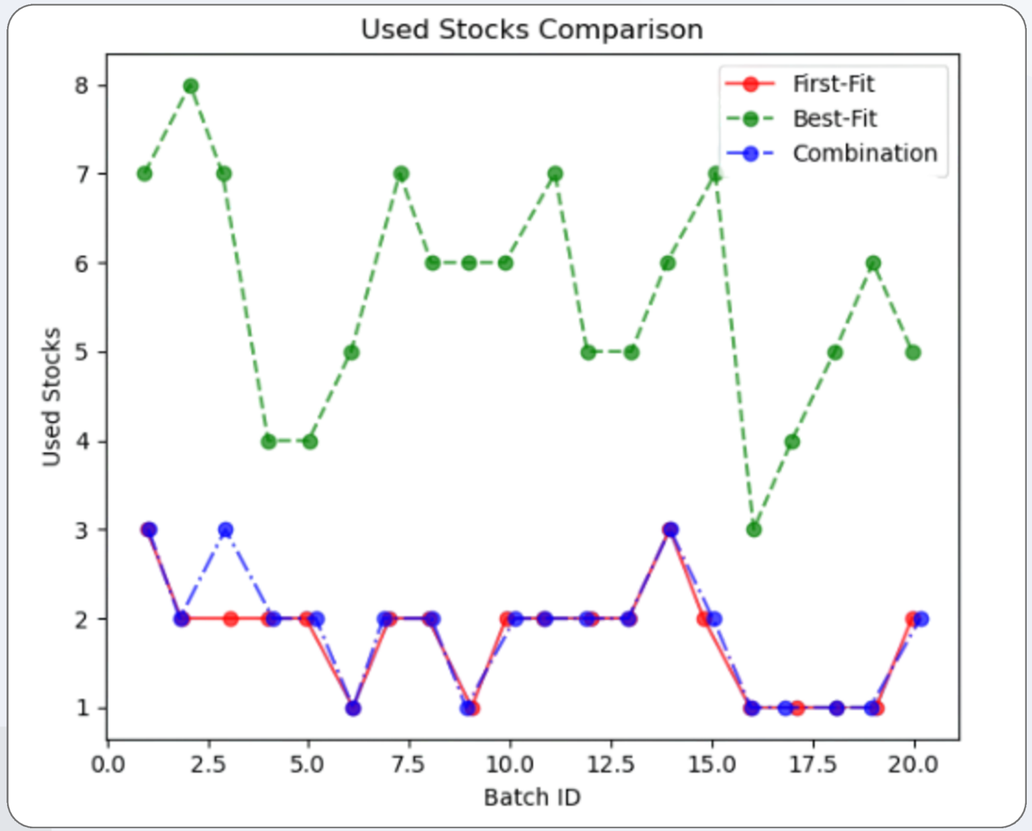
Heuristic Algorithm





Heuristic Algorithm



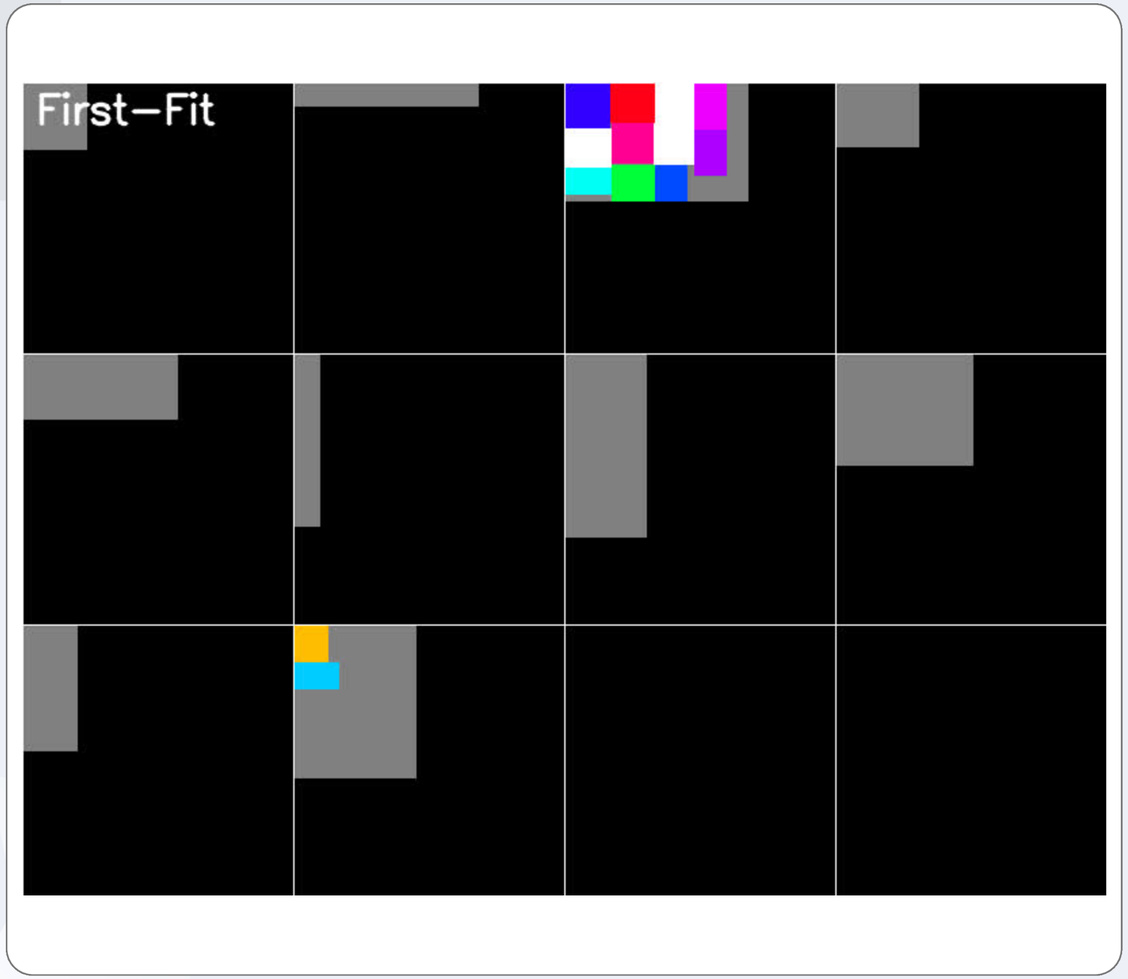


Heuristic Algorithm

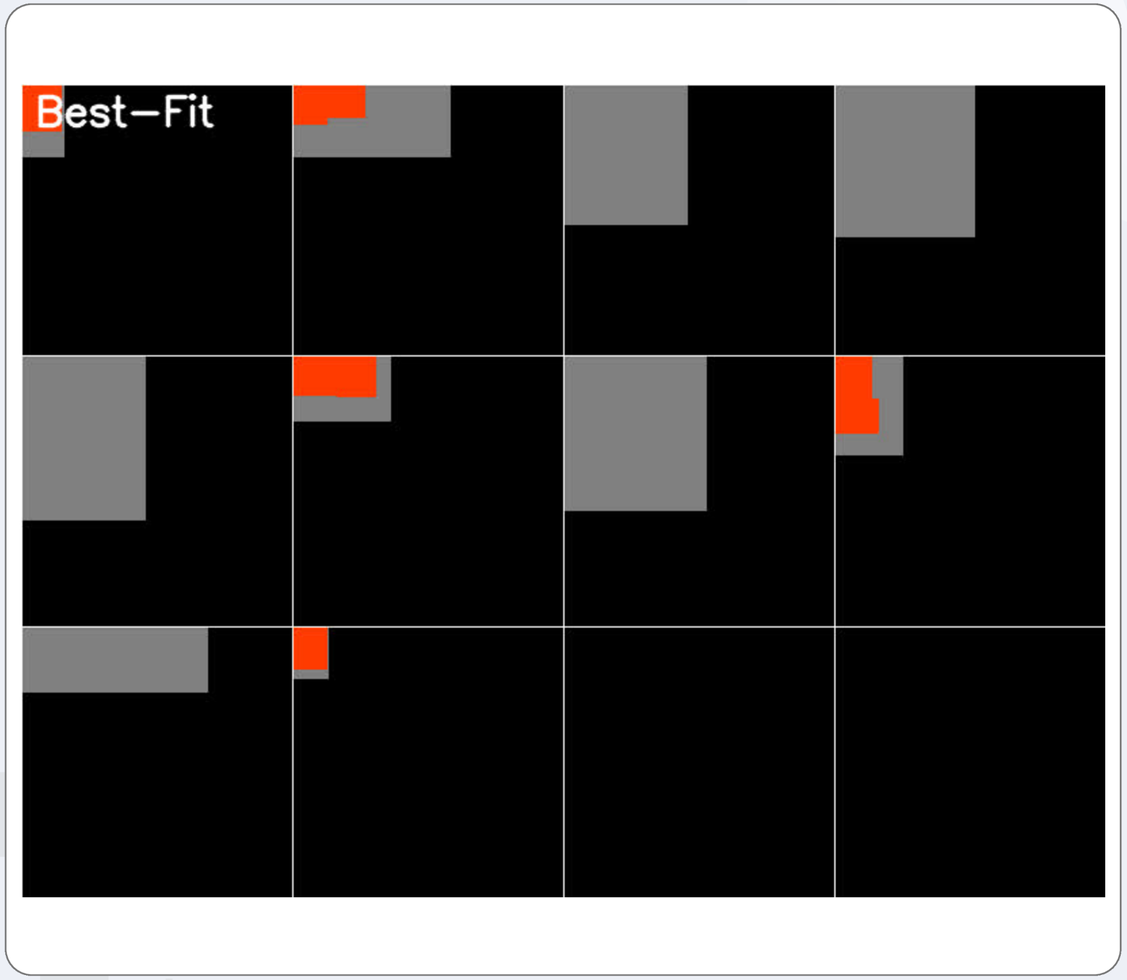
- Combination: Best in used stocks (Used Stocks: 2.5, Remaining Stocks: 5.5), fast run time (10 seconds), but not lowest Trim Loss (8000).
- Best-Fit: Best in Trim Loss (7000) and Average Used Stock Area (750), but slower (15 seconds).
- First-Fit: Fast (10 seconds), but worst in Trim Loss (9000), Used Stocks (4), and Average Used Stock Area (1000).

Chỉ số	First-Fit	Best-Fit	Combination
Runtime (giây)	10	15	10
Total Trim Loss	9000	7000	8000
Remaining Stocks	4.5	5	5.5
Used Stocks	4	3	2.5
Avg Used Stock Area	1000	750	900

Heuristic Algorithm



Batch 3



Batch 10

Q-Learning Algorithm

How Q learning works?

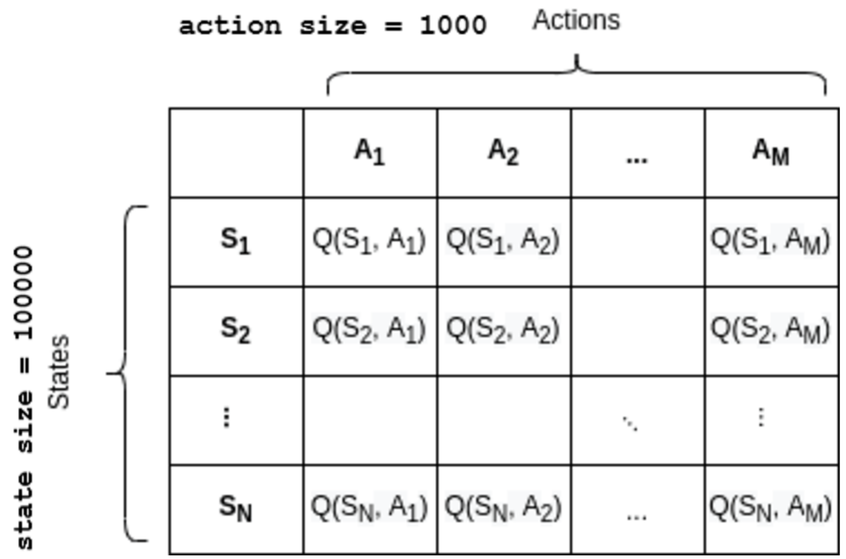
1. Initialize environment and Q-table

create environment

The environment is created using the CuttingStockEnv class with a list of stocks and products:

- Stocks: Raw material sheets with maximum size of (120,120)
- Products: Cut products with maximum size of (120,120)

Q-table



Q-Learning Algorithm

FPT Education
FPT UNIVERSITY

How Q learning works?

2. Initialize environment and Q-table

Convert the state from the environment into an integer to store in the Q-table. In this problem:
The state is calculated based on the empty_space in stocks and the number of remaining products (remaining_products):

```
state = (empty_space * 1000 + remaining_products) % state_size
```

Q-Learning Algorithm

How Q learning works?

3. Action Selection

Use epsilon-greedy strategy:

- If *random < ε: Choose random action (random.randint(0, action_size-1)).*
- *Otherwise: Choose action with highest Q value (np.argmax(Q_table[state])).*

Then convert action from Q-table into real action in environment, for example: which stock to choose, which product to cut, at which position.

=> Convert action to environment

=> Take action and get feedback

Q-Learning Algorithm

FPT Education
FPT UNIVERSITY

How Q learning works?

4. Reward Calculation

- Bonus calculation based on:
- filled_ratio: Material used.
- trim_loss: Material discarded.
- stock_bonus: Based on unused stock:

```
stock_bonus = lambda_bonus * (num_stocks_unused / total_stocks)
reward = (filled_ratio - trim_loss) + stock_bonus
```

Q-Learning Algorithm

How Q learning works?

5. Update Q-table

After each step, the Q value is updated using the Bellman formula:

*Q_table[state, action] = (1 - alpha) * Q_table[state, action] + alpha * (reward + gamma * np.max(Q_table[next_state]))*

This formula balances between maintaining previous experience and learning from new feedback, thereby gradually converging on the optimal policy for the agent during training.

=> Repeat and decrease epsilon

Q-Learning Algorithm

FPT Education
FPT UNIVERSITY

Q learning Algorithm Pseudocode

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
    $S \leftarrow S'$
  until S is terminal

Q-Learning Algorithm

Datasets

- **s**tocks

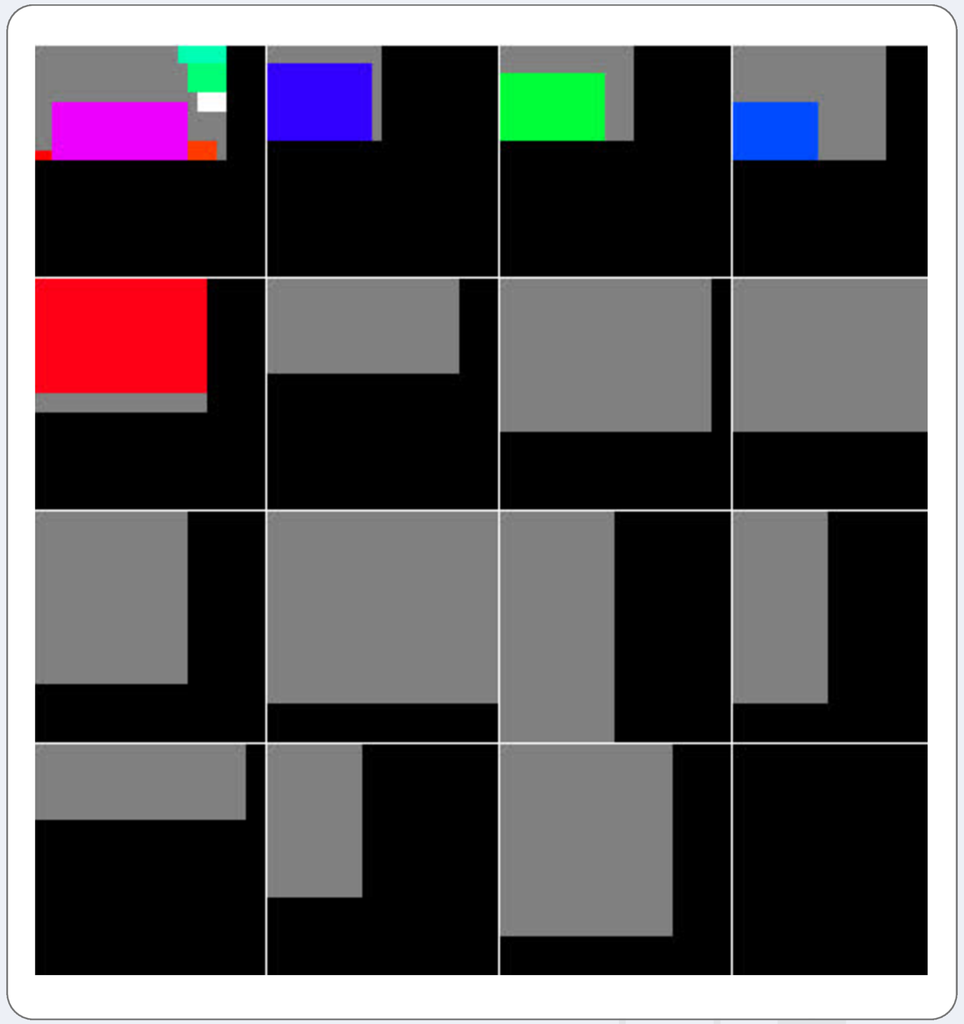
(50, 50), (60, 40), (70, 50), (80, 60), (90, 70),
(100, 50), (110, 60), (120, 80), (130, 90), (140, 100),
(150, 120), (160, 130), (170, 140), (180, 150), (200, 200)

- products

(10, 5), (15, 10), (20, 10), (25, 15), (30, 20),
(35, 20), (40, 30), (45, 25), (50, 30), (55, 35),
(20, 15), (25, 10), (30, 15), (35, 20), (40, 25),
(45, 30), (50, 35), (55, 40), (60, 45), (65, 50),
(70, 30), (75, 40), (80, 50), (85, 55), (90, 60),
(15, 10), (20, 15), (25, 20), (30, 25), (35, 30)

Q-Learning Algorithm

Evaluation



Actor-Critic Algorithms

a) What is Actor-Critic?

Actor-Critic is a hybrid reinforcement learning (RL) approach that combines:

- Policy-based RL (Actor): Learns the optimal policy directly.
- Value-based RL (Critic): Evaluates actions for more stable learning.
  - Actor ($\pi_\theta(a|s)$):
    - Represents the policy model, selecting actions.
    - Updated using the policy gradient method.
  - Critic ($V_w(s)$):
    - Estimates the value function of a state.
    - Helps stabilize learning by providing feedback to Actor.

Actor-Critic Algorithms

b) Architecture of Actor-Critic

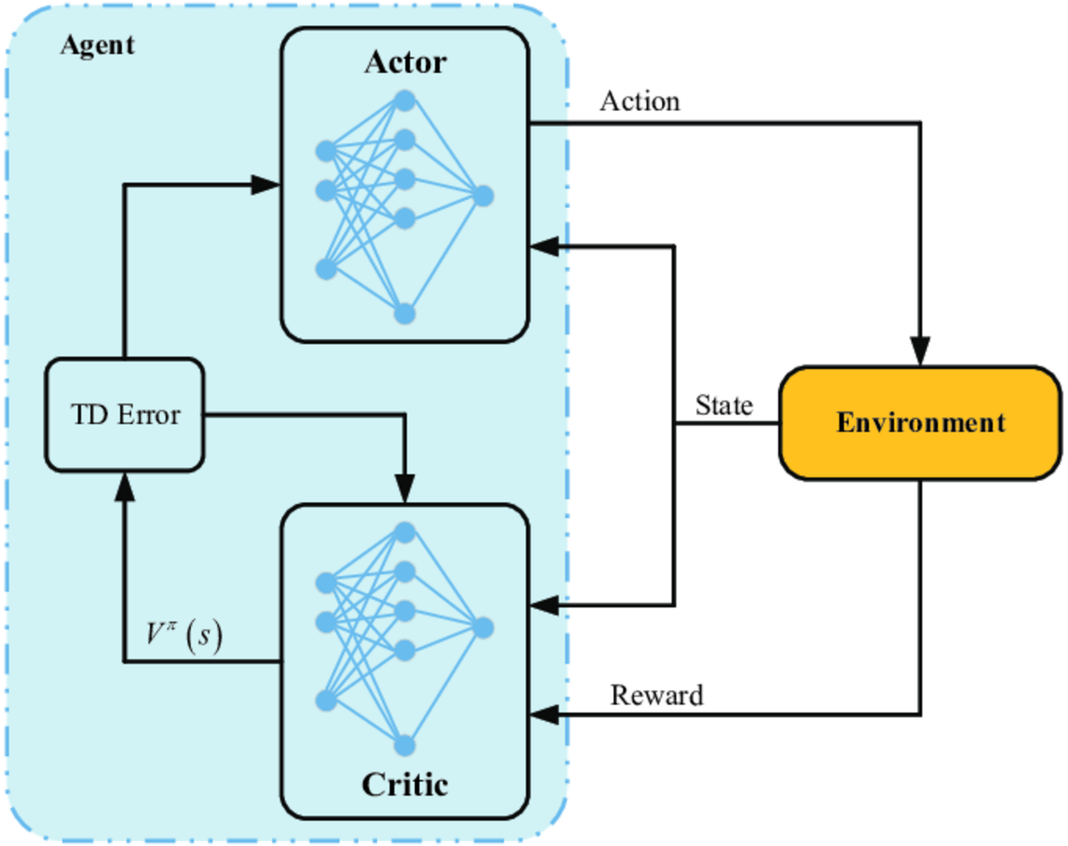


Figure 5 - available via license: Hybrid NOMA/OMA-Based Dynamic Power Allocation Scheme Using Deep Reinforcement Learning in 5G Networks
(https://www.mdpi.com/2076-3417/10/12/4236)

Actor-Critic Algorithms

c) Hyperparameters and Workflow

- **Hyperparameters**
  - GAMMA = 0.99
  - LR = 0.001
  - MAX_EPISODES = 1000
  - MAX_STEPS = 200
  - MAX_W = 100
  - MAX_H = 100
  - MAX_PRODUCTS = 100
  - EPSILON_START = 0.5
  - EPSILON_END = 0.01
  - EPSILON_DECAY = 0.995

Actor-Critic Algorithms

c) Hyperparameters and Workflow

Loop (for each episode, up to $\text{MAX\_EPISODES} = 1000$):

1. **Initialize:**
   - $S \leftarrow \text{env.reset}()$ (processed via process_state).
   - $\epsilon \leftarrow 0.5$ (initially).
2. **Loop while S is not done and $\text{step} < 200$ (for each time step):**
   - $A \sim \pi(\cdot|S, \theta)$ (via select_action, with ϵ-greedy).
   - Take action A, observe S', R (via env.step, reward adjusted).
   - Compute return R_t (cumulative discounted reward, normalized).
   - $\delta \leftarrow R_t - \hat{v}(S, w)$ (advantage).
   - $w \leftarrow w - \alpha \nabla_w \text{smooth\_l1\_loss}(\hat{v}(S, w), R_t)$ (via optimizer).
   - $\theta \leftarrow \theta + \alpha \delta \nabla_\theta \ln \pi(A|S, \theta)$ (via optimizer).
   - $\epsilon \leftarrow \max(0.01, \epsilon \cdot 0.995)$.
   - $S \leftarrow S'$.

With:

- Calculate return

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-t} r_T$$

- Normalize reward value

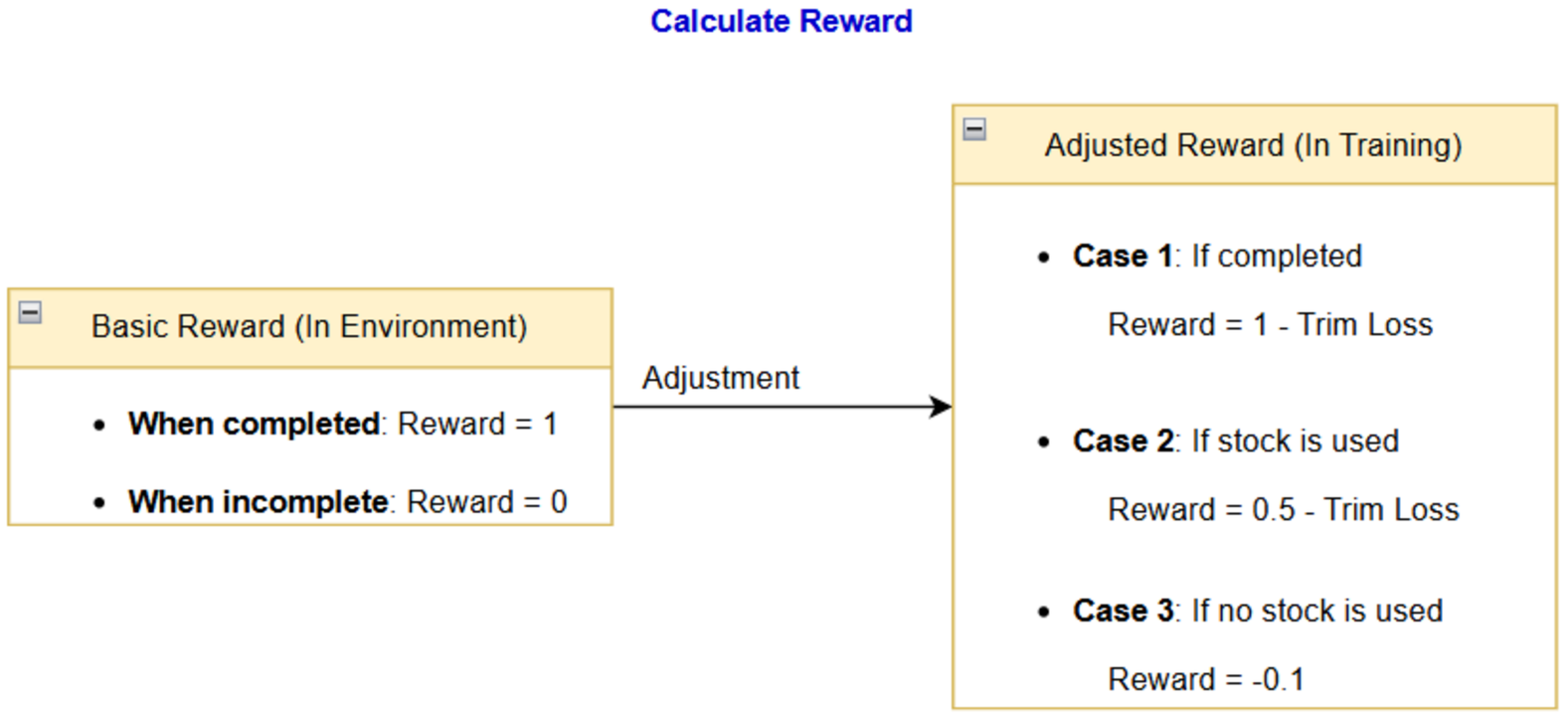
$$R_t \leftarrow \frac{R - \text{mean}(R)}{\text{std}(R) + 10^{-5}}$$

- Smooth L1 Loss

$$\text{smooth\_l1\_loss}(\hat{v}(S, w), R_t) = \begin{cases} 0.5(\hat{v}(S, w) - R_t)^2, & \text{if } |\hat{v}(S, w) - R_t| < 1 \\ |\hat{v}(S, w) - R_t| - 0.5, & \text{if } |\hat{v}(S, w) - R_t| \geq 1 \end{cases}$$

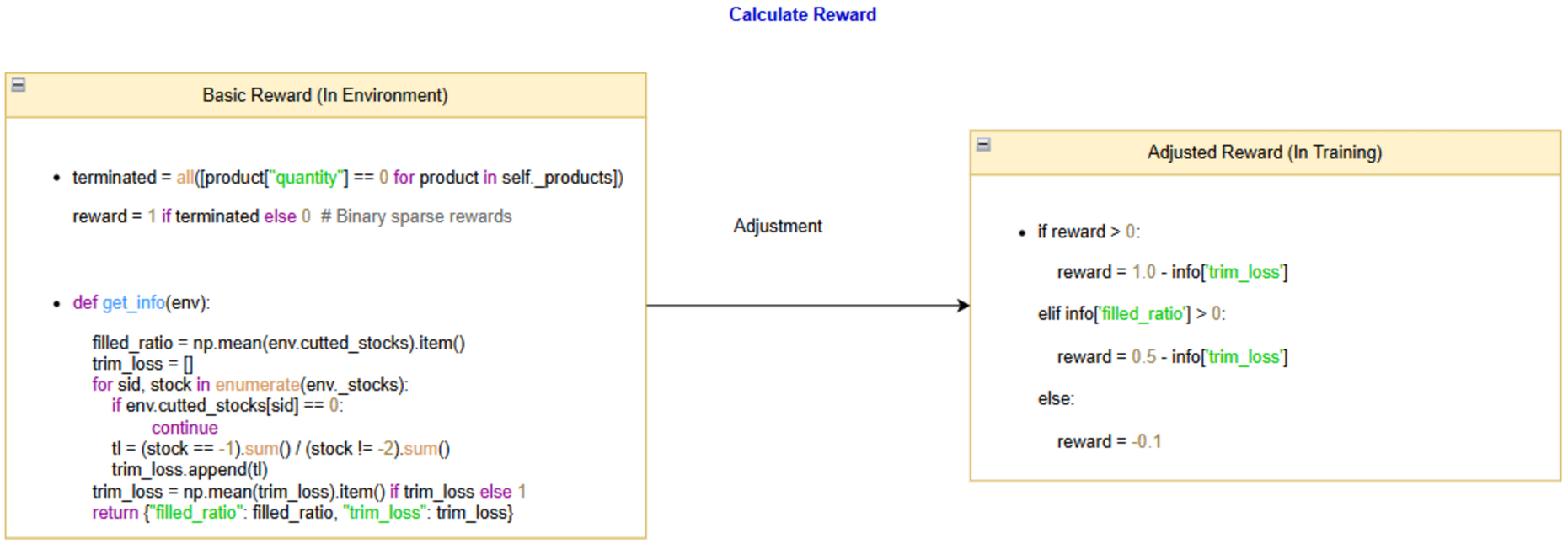
Actor-Critic Algorithms

c) Hyperparameters and Workflow



Actor-Critic Algorithms

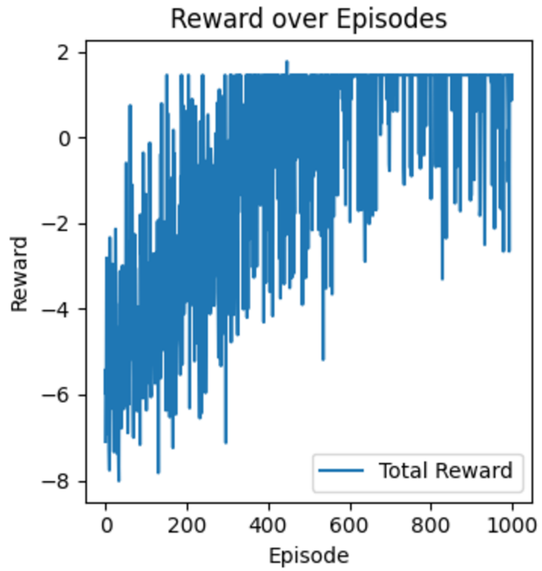
c) Hyperparameters and Workflow

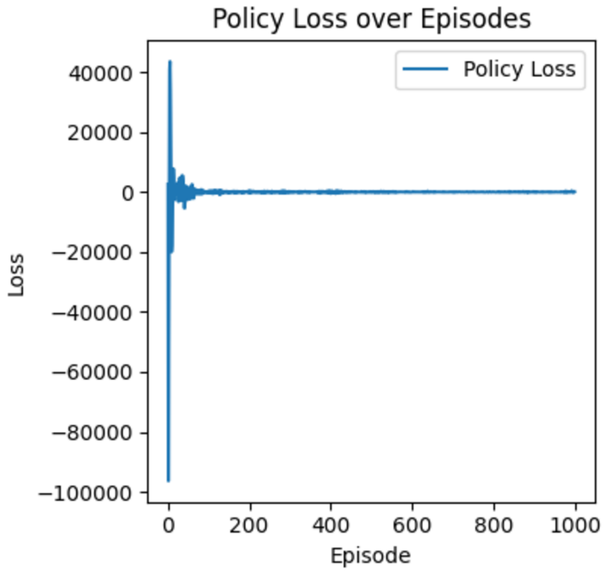


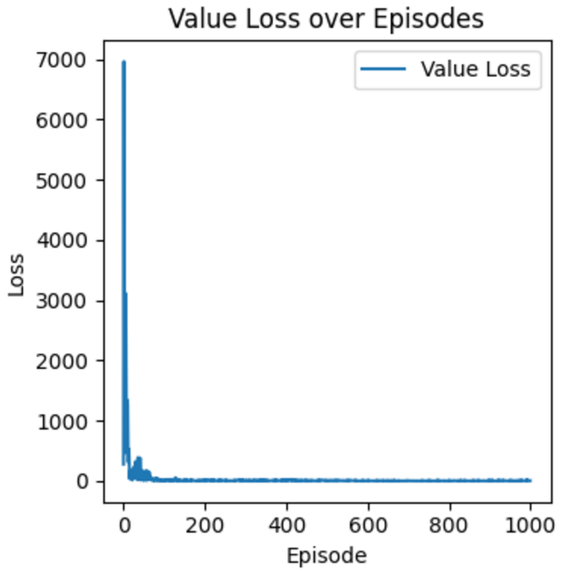
Actor-Critic Algorithms

d)Evaluate

- **Evaluate:** Train with 10 stocks and 31 products



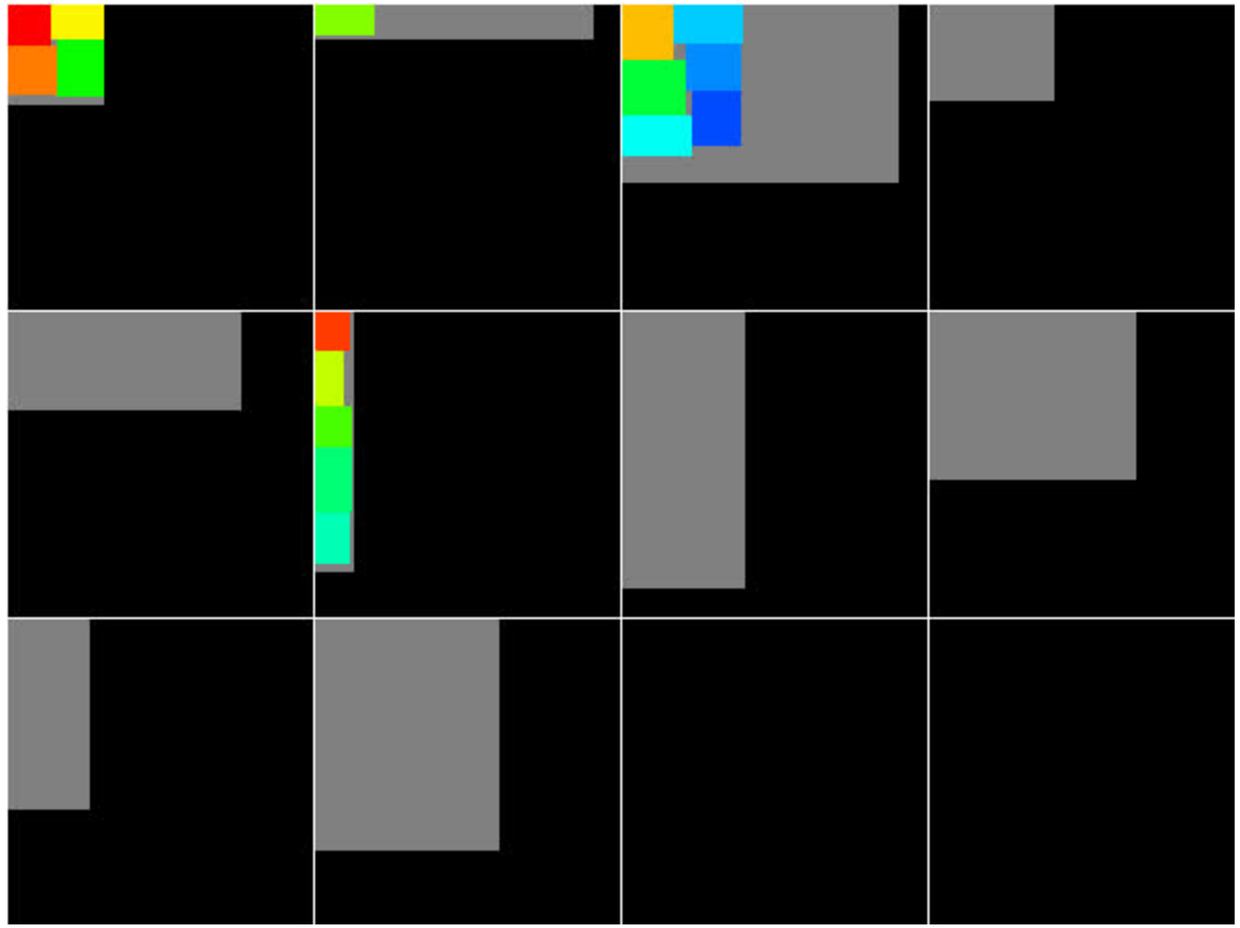




Actor-Critic Algorithms

d) Evaluate

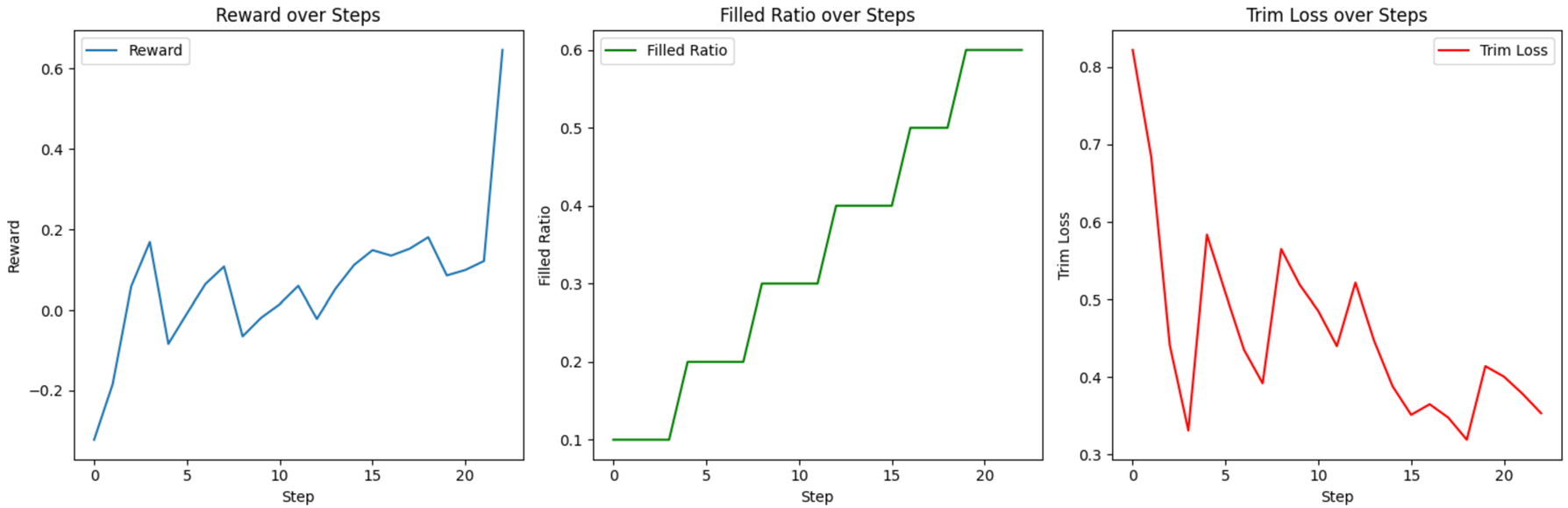
- **Evaluate:** Train with 10 stocks and 31 products



Actor-Critic Algorithms

d) Evaluate

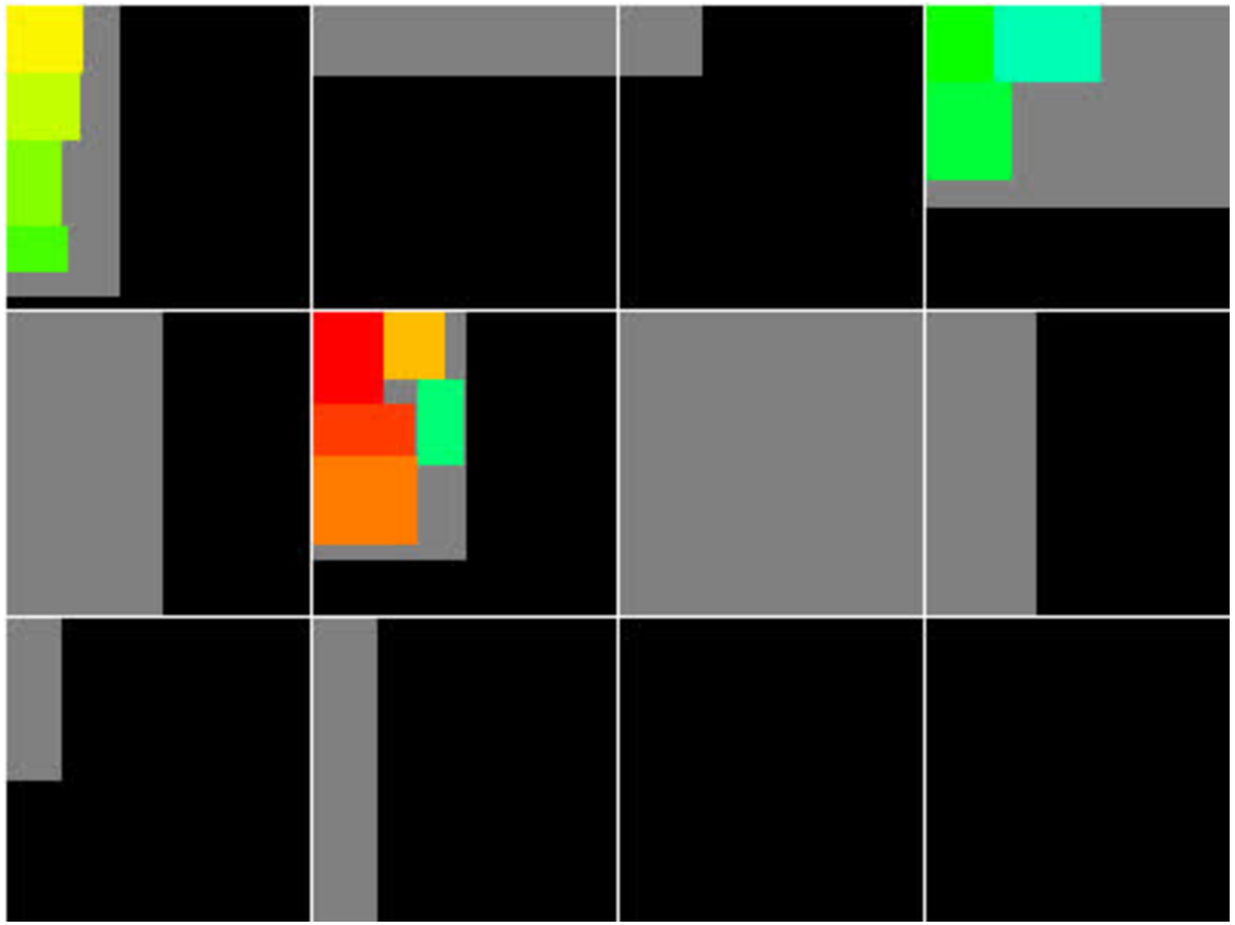
- **Evaluate:** Test with 10 stocks and 23 products



Actor-Critic Algorithms

d) Evaluate

- **Evaluate:** Test with 10 stocks and 23 products



FPT Education

FPT UNIVERSITY

DEMO

CUTTING STOCK

FPT Education

FPT UNIVERSITY

Thank You