FPT UNIVERSITY - QUY NHON CAMPUS
AI DEPARTMENT

REINFORCEMENT LEARNING (REL303M)

Assignment (Semester: SP25, Duration: 05 weeks)

"Cutting Stock Problem"

(Version 0.1)

Instructor: Nguyễn An Khương, khuongna2@fe.edu.vn

Quy Nhon, January 2025

Table of Contents

Abstract. The Cut Stock Problem (CSP) is a classic optimization problem in the field of Operations Research and Industrial Engineering. It involves determining the most efficient way to cut a set of smaller lengths (orders) from a given larger length (stock), such that waste is minimized. This problem arises in various industries such as paper, steel, and textile, where materials are produced in standard lengths and need to be cut into smaller sizes to meet customer demand. The complexity of the problem increases with the number of different order sizes and the requirement to fulfill all demands exactly. Various solution approaches have been proposed for the CSP, including dynamic programming, column generation, and heuristic methods.

Team Assignment Goals: Learn concepts, ideas and methods to

a) Literature Review: Conduct a comprehensive literature review on the Cut Stock Problem. This should include understanding the problem's history, its significance in various industries, and the different mathematical models used to represent it.

b) Algorithm Analysis: Analyze different algorithms and heuristics that have been proposed to solve the Cut Stock Problem. Understand their strengths, weaknesses, and the conditions under which they perform best.

c) Case Studies: Identify and study real-world cases where the Cut Stock Problem is relevant. Understand how these problems were addressed and what solutions were implemented.

d) Tool Development: Develop a tool or software that can solve the Cut Stock Problem. This could involve implementing one or more of the algorithms studied in goal 2.

e) Performance Evaluation: Evaluate the performance of the developed tool on both synthetic and real-world data. Compare its performance with existing tools.

f) Future Work: Identify gaps in the current state of research and propose future work. This could involve developing new algorithms, improving existing ones, or exploring new application areas for the Cut Stock Problem.

1 Introduction to Cutting-stock problem

In operations research, the cutting-stock problem is the problem of cutting standard-sized pieces of stock material, such as paper rolls or sheet metal, into pieces of specified sizes while minimizing material wasted. It is an optimization problem in mathematics that arises from applications in industry. Regarding computational complexity, the problem is an NP-hard problem reducible to the knapsack problem. The problem can be formulated as an integer linear programming problem.

Firstly, the simplification relationships should follow a straight line. Hence, the term linear. Secondly, all the values need to be subject to constraints, which may be in numerical or in terms or properties. And lastly, the solution has to optimize (i.e., maximize or minimize, depending on the problem) the quantity of a given variable.

Cutting-stock problems can be classified in several ways. One way is the dimensionality of the cutting: the industrial applications of 1D occur when cutting pipes, cables, and steel bars. Two-dimensional (2D) problems are encountered in furniture, clothing and glass production. When either the master item or the required parts are irregular-shaped (a situation often encountered in the leather, textile, and metals industries), this is referred to as the nesting problem.

Not many three-dimensional (3D) applications involving cutting are known; however, the closely related 3D packing problem has many industrial applications, such as packing objects into shipping containers (see, e.g. containerization: the related sphere packing problem has been studied since the 17th century (Kepler conjecture)).

2 One-dimensional cutting-stock problem

The one-dimensional cutting-stock problem is a classic problem in operations research and industrial engineering. It involves determining the most efficient way to cut a set of smaller lengths (orders) from a given larger length (stock) such that waste is minimized. This problem arises in various industries such as paper, steel, and textile, where materials are produced in standard lengths and need to be cut into smaller sizes to meet customer demand. The complexity of the problem increases with the number of different order sizes and the requirement to fulfil all demands exactly.

♦ **EXAMPLE 1.** *A company produces metal rods that are 100 units long. They receive orders for rods of lengths 45, 36, 30, and 15 units. The respective demands for these lengths are 7, 5, 8, and 9.*

Goal: *The goal is to determine how to cut the 100-unit rods to fulfil these orders while minimizing waste.*

One possible solution:

- Cut 7 rods into lengths of 45 and 55 units. Use the 55-unit pieces to fulfill the demand for 30-unit and 25-unit rods.
- Cut 5 rods into lengths of 36 and 64 units. Use the 64-unit pieces to fulfill the demand for 30-unit and 34-unit rods.
- Cut 8 rods into lengths of 30 and 70 units. Use the 70-unit pieces to fulfill the demand for 15-unit rods.
- Cut 9 rods into lengths of 15 and 85 units. Use the 85-unit pieces to fulfill the remaining demand for 15-unit rods.

This solution minimizes the waste while fulfilling all the orders. However, it's important to note that this is just one possible solution. The optimal solution would depend on the specific demands and the cutting costs.

3 Two-dimensional cutting-stock problem

The two-dimensional cutting stock problem (2D CSP) is a type of optimization problem where the goal is to cut smaller rectangular items from larger rectangular sheets of material in a way that minimizes waste or the number of sheets used. This problem is common in industries like manufacturing, textiles, and packaging.

♦ **EXAMPLE 2.** *A company needs to cut smaller rectangular pieces from large stock sheets of metal. The dimensions of the stock sheets can be seen in Table 1 and the number of demand for products in Table 2.*

Table 1: The sizes of stocks

The i^{th} stocks	Length (inches)	Width(inches)
1	24	14
2	24	13
3	18	10
4	13	10

Goal: *The goal is to determine how to cut the stock sheet to fulfill the demand while minimizing material wasted.*

Table 2: The demand sizes

The i^{th} product	Length (inches)	Width(inches)	Number of demand
1	2	1	5
2	4	2	5
3	5	3	2
4	7	4	3
5	8	5	2

Based on the counting, some optimal cutting patterns are shown in Figures 1 to 4.

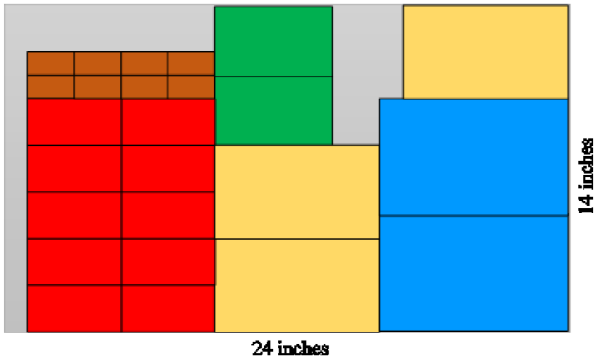


Fig. 1: *The combination of cutting patterns on the first stock.*

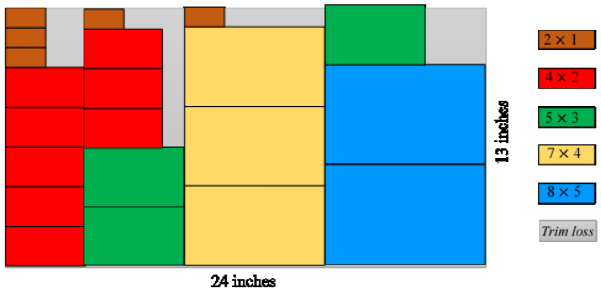
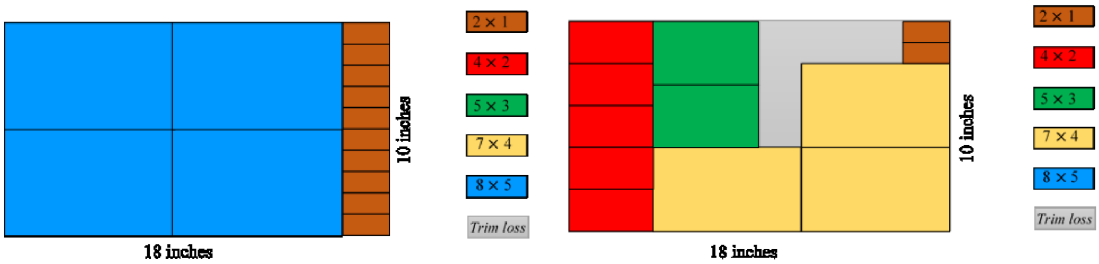


Fig. 2: *The combination of cutting patterns on the second stock.*



(a) (b)

Fig. 3: *(a) The first combination of cutting patterns on the third stock. (b) The second combination of cutting patterns on the third stock.*

Figure 1 to 4 shows the combination of a cutting pattern for each stock. There is one combination of the first and second stock, two combinations of the third stock and three combinations of the

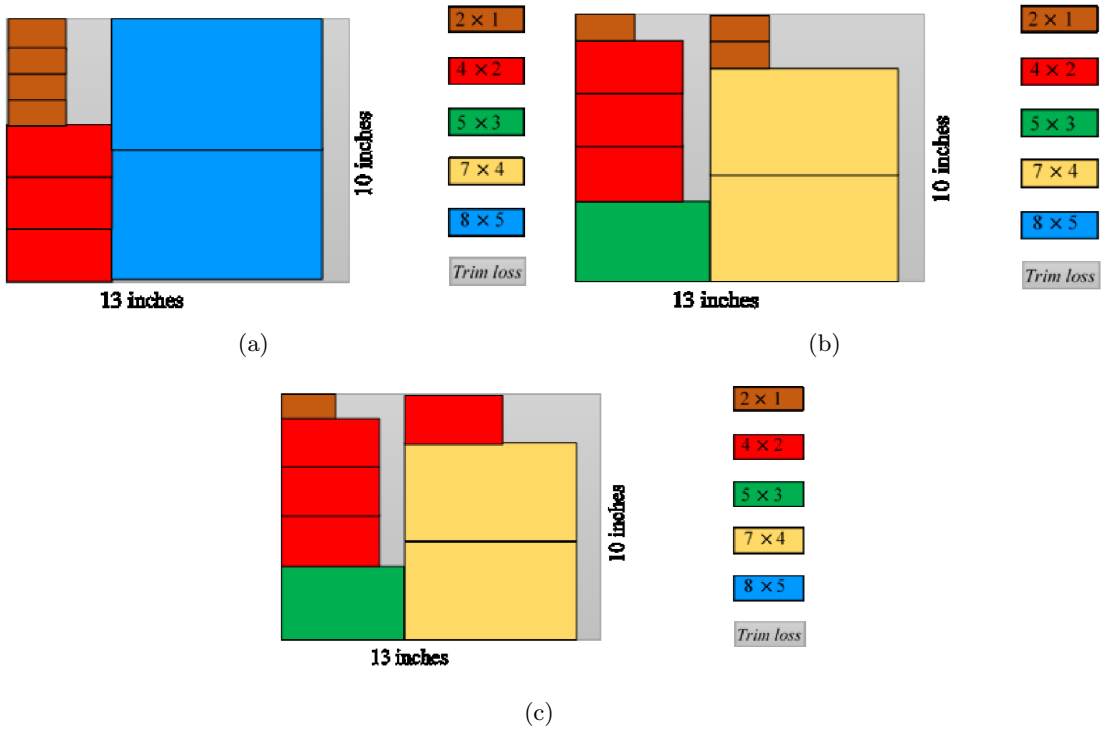


Fig. 4: *(a) The first combination of cutting patterns on the fourth stock. (b) The second combination of cutting patterns on the fourth stock. (c) The third combination of cutting patterns on the fourth stock*

fourth stock. The number of products that will be produced based on the combination of cutting patterns can be seen in Table 3.

Table 3: The number of products that will be produced

No.	The size of demand	Number of products
1.	2 inches × 1 inch	33
2.	4 inches × 2 inches	33
3.	5 inches × 3 inches	9
4.	7 inches × 4 inches	13
5.	8 inches × 5 inches	11

Based on the results that can be seen in Table 3, there are five sizes of demand with each number of products. The most products are for 2 inches 1 inch and 4 inches 2 inches, 33 plates respectively.

4 QUESTIONS for ASSIGNMENT SEMESTER SP25

Students prepare a report consisting of the theoretic answer and computational solution following requests.

1. **Problem Formulation:** Formulate the two-dimensional cutting-stock problem as an integer linear programming problem. Define the decision variables, objective function, and constraints.

2. **Case Study:** Identify a real-world scenario where the two-dimensional cutting-stock problem is relevant. Describe the scenario and discuss how the problem could be solved in this context.

3. **RL Approach:** Apply at least one Reinforcement Learning algorithm to solve the 2D cutting-stock problem. See Appendix B.

4. **Performance Evaluation:** Discuss how you would evaluate the performance of the algorithm used. What metrics would you use? How would you gather the necessary data?

5 Instructions and requirements

Students have to follow the instructions and comply with the requirements below. *Lecturers do not solve the cases arising because students do not follow the instructions or do not comply with the requirements.*

5.1 Instructions

Students must work closely with the other members of their team of 3-5 members.

The initial code for this assignment is available at this Github repository. This code is written in Python using `gymnasium` package. Details about the code can be found in Appendix A. If you have any questions about the assignment during work, **please post that question on the class's Zalo group.**

Regarding the background knowledge related to the topic, students should refer to all the references. However, it would be best to put all of them in the reference section of your report.

5.2 Requirements

- Deadline for submission: **March 23, 2025**. Students have to answer each question clearly and coherently.
- Each team when submitting its report **need to submit also a log file (diary)** in which clearly state: **weekly work progress for all week(s)**, tasks, content of opinions exchanged of the members, ...
- Programming languages: Python
- **Reporting requirements:** Your report should include the following sections:
  - **Introduction:** Introduce the problem and goals of the major exercise.
  - **Acknowledge**
  - **Algorithm**
  - **Modeling:** Describe how you modeled the problem as a linear programming problem. This should include defining variables, objective functions, and constraints.
  - **Analyze results:** Analyze and interpret model results, including explaining the optimal solution and its consequences for the company's operations.
  - **Conclusion:** Provide final comments and extensibility of the model and algorithms.
  - **References**

5.3 Submission

- Students must submit their team report via the social constructive learning system; compress all necessary files into a single file named *"Assignment-REL301m-Team_ name.zip"* and submit it to the assignment submission site (see a sample submission at https://tinyurl.com/REL301mSP25SampleSubmission).
- All of the meeting minutes of your group must be combined into one .txt file and consist of the % of each member's effort on the whole (total effort is 100%) on the first line.
- Solution code must be submitted as a GitHub repository.
- Noting that for each team, **only the leader will submit the report of the team.**

6 Evaluation and cheating treatment

6.1 Evaluation

Each assignment will be evaluated as in Table 4.

Table 4: Evaluation

Criteria	**Score (%)**
- Analyze, answer coherently, systematically, focus on the goals of the questions and requests	30%
- The programs are neatly written and executable	30%
- Correct, clear, and intuitive graphs & diagrams	20%
- Background section is well written, correct, and appropriate	15%
- Well written report and correct	5%

6.2 Cheating treatment

The assignment must be done by each group separately. Students in a group will be considered as cheating if:

- There is an unusual similarity among the reports (especially in the background section). In this case, ALL similar submissions are considered cheating. Therefore, the students of a group must protect their group's work.
- They do not understand the works written by themselves. You can consult from any source, but make sure that you know the meaning of everything you have written.

Students will be judged according to the university's regulations if the article is cheating.

7 COMPLEMENTS:

Few well-known 2D Cut Stock Problem and analysis are

1. **Extra Material: Cutting Stock**,

   get from link https://ampl.com/mo-book/notebooks/05/cutting-stock.html.

2. A linear programming approach to the cutting-stock problem (1).
3. Two-Dimensional Cutting Problem (https://pure.iiasa.ac.at/id/eprint/3571).
4. A Global Method for a Two-Dimensional Cutting Stock Problems in the Manufacturing Industry (2)
5. A Heuristic Approach for Two-Dimensional Rectangular Cutting Stock Problem Considering Balance for Material Utilization and Cutting Complexity (3).
6. Exact Solution Techniques for Two-dimensional Cutting and Packing (4).
7. Cutting stock problems and solution procedures (5).
8. A near-optimal solution to a two-dimensional cutting stock problem (6).

References

[1] P. C. Gilmore and R. E. Gomory, "A linear programming approach to the cutting-stock problem," *Operations research*, vol. 9, no. 6, pp. 849–859, 1961.

[2] Y.-H. Huang, H.-C. Lu, Y.-C. Wang, Y.-F. Chang, and C.-K. Gao, "A global method for a two-dimensional cutting stock problem in the manufacturing industry," *Application of Decision Science in Business and Management*, p. 167, 2020.

[3] D. Wu and G. Yang, "A heuristic approach for two-dimensional rectangular cutting stock problem considering balance for material utilization and cutting complexity," *Advances in Materials Science and Engineering*, vol. 2021, no. 1, p. 3732720, 2021.

[4] M. Iori, V. L. De Lima, S. Martello, F. K. Miyazawa, and M. Monaci, "Exact solution techniques for two-dimensional cutting and packing," *European Journal of Operational Research*, vol. 289, no. 2, pp. 399–415, 2021.

[5] R. W. Haessler and P. E. Sweeney, "Cutting stock problems and solution procedures," *European Journal of Operational Research*, vol. 54, no. 2, pp. 141–150, 1991.

[6] C. Kenyon and E. Rémila, "A near-optimal solution to a two-dimensional cutting stock problem," *Mathematics of Operations Research*, vol. 25, no. 4, pp. 645–656, 2000.

[7] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[9] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems* (S. Solla, T. Leen, and K. Müller, eds.), vol. 12, MIT Press, 1999.

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[11] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of The 33rd International Conference on Machine Learning* (M. F. Balcan and K. Q. Weinberger, eds.), vol. 48 of *Proceedings of Machine Learning Research*, (New York, New York, USA), pp. 1928–1937, PMLR, 20–22 Jun 2016.

[12] A. R. Pitombeira-Neto and A. H. Murta, "A reinforcement learning approach to the stochastic cutting stock problem," *EURO Journal on Computational Optimization*, vol. 10, p. 100027, 2022.

[13] J.-Y. Su, C.-H. Liu, C.-S. Syu, J.-L. Kang, and S.-S. Jang, "A reinforcement learning development for the exact guillotine with flexibility on cutting stock problem," in *33rd European Symposium on Computer Aided Process Engineering* (A. C. Kokossis, M. C. Georgiadis, and E. Pistikopoulos, eds.), vol. 52 of *Computer Aided Chemical Engineering*, pp. 451–456, Elsevier, 2023.

A Code

This section will describe the initial code used for this assignment.

A.1 Initial code explanation

The initial code provided for this assignment is a simulation of the cutting-stock problem. The code contains the following files:

- `main.py`: The main file that runs the simulation.
- `policy.py`: The policy class that contains the policy information.
- `requirements.txt`: The requirements file that contains the required libraries.

To begin with this code, you need to have Python >=3.8 and install the required libraries by running the following command:

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

You can run the simulation by running the following command:

```
python main.py
```

The simulation will run for 100 episodes and print the results of each episode (ratio of filled stocks). This code implements a random and greedy policy. The random policy will randomly select a stock and cut it to fulfill the demand. The greedy policy will select the first acceptable stock to fulfill the demand.

Starting from `main.py`, we started to create a cutting stock environment by

```
    env = gym.make(
        "gym_cutting_stock/CuttingStock-v0",
        render_mode="human",  # Comment this line to disable rendering
    )
```

The environment is created by `gym.make` function with the environment name `CuttingStock-v0`. The environment is a subclass of `gym.Env` class. To begin using it, you need to reset the environment by calling the `reset` method.

```
    observation, info = env.reset()
```

The `reset` method will return the initial observation and information about the environment. The observation is a dictionary that contains the following information:

- `stocks`: A list of stocks that can be cut to fulfill the demands. Each stock is a numpy array of `max_width`, `max_height`. Elements in the array represent the index of the product. If the element equals -2, this position is not available. If element is equal -1, it means the stock is empty. If the element is equal to or greater than 0, it means the stock is filled with the product index.
- `products`: A list of products that need to be cut from the stocks. Each product is a dictionary that contains the `size` and `quantity` of the product.

After resetting the environment, you can take an action by calling the `step` method with the action as input.

```
    action = policy.get_action(observation, info)
    observation, reward, terminated, _, info = env.step(action)
```

The `step` method will return the next observation, reward, terminated, and information about the environment. The reward is 1 if you fulfill all demands. The terminated is a boolean that indicates if the episode is terminated. The info is a dictionary that contains the filled ratio. An action is a dictionary that contains `stock_idx`, `size`, and `position` of the stock to cut. The `stock_idx` is the index of the stock in the `stocks` list. The `size` is the size of the product to cut. The `position` is the position to cut the stock. Please note that all indexes of the stock and product are 0-based.

A.2 How to implement your own policy

To implement your own policy, you need to create a new class that inherits from the Policy class and implement the `get_action` method. The `get_action` method should take a list of demands and a list of stock as input and return a dictionary that contains action information. The action information should include the size of demand, stock index, and position to cut the stock. You should also implement the `__init__` method to initialize the policy with any required parameters. Please refer to the `RandomPolicy` class in the `policy.py` file for an example implementation.

You can start by creating a new file in the `student_submissions` directory and implementing your policy in `s22110xxx` folder. Your code should be named `policy2210xxx.py` where `2210xxx` is your student ID. The policy class should be named `Policy2210xxx` inherit from the `Policy` class. You can have some support files in `s22110xxx` folder. If you are in an honor class, you must implement a reinforcement learning policy. Once you have implemented your policy, you can uncomment the line in the `main.py` file that imports your policy and run the project to test your policy. You can only use basic Python libraries such as numpy, pandas, torch, TensorFlow, sci-kit-learn, and scipy to implement your policy. Please put the new library in the `requirements.txt` file.

After you complete your policy, you need to submit your code as a pull request to the main repository. The pull request should include the following information:

- Title: Your student ID
- A brief description of your policy
- The implementation of your policy showing the commits of team members
- The summarized results of your policy
- Any additional information you would like to include

Any solutions that are not submitted as a pull request or do not follow the above guidelines will not be accepted. If you have any questions or need help implementing your policy, please post a message in the discussion forum.

B Reinforcement Learning for Cutting Stock Problem

B.1 What is Reinforcement Learning?

Reinforcement learning (RL) (7) is a field of artificial intelligence that enables an agent (decision-maker) to act more intelligently through a learning process. In RL, key concepts include:

- **Environment:** The environment represents the “ground truth." For example, in a 2D cutting stock problem, the environment receives the size and position of the stock to be cut and returns whether the cut was successful.

- **State:** A state represents the current situation or configuration of the environment that the agent observes. For instance, in the cutting stock problem, the state could include the remaining uncut stock and the positions available for cutting.
- **Agent (or Actor):** The agent is responsible for making decisions (or actions), such as selecting the size and position to cut.
- **Policy:** A policy governs how decisions are made. It takes a state and provides probabilities of selecting each possible action. For instance, a random policy might select stock sizes and positions randomly.
- **Reward Model (or Surrogate Model):** Depending on the RL algorithm, a reward model can be utilized to score each decision. For example, a lower reward could be given if we cut a 3x1 pattern in the middle of a 3x3 stock instead of on the side.

From a modeling perspective, RL algorithms fall into two main categories: model-free and model-based. Since the transitions between states in the cutting stock problem are deterministic, we focus on model-free approaches. From an optimization perspective, there are also two types: on-policy and off-policy optimization. On-policy optimization requires the agent to learn from its own experience (which might currently not be optimal solutions). In contrast, off-policy optimization allows the agent to gain knowledge from other, similar problems with near-optimal or optimal solutions.

In this exercise, students are required to implement ***on-policy, model-free RL algorithm(s)*** to solve the $>= 2D$ cutting stock problem. Currently, the assignment repository only implements the 2D cutting stock problem. If you are interested in implementing a higher dimension of this problem, please open a pull request. The first team implementing this feature will get a *bonus*.

B.2 Model-free Reinforcement Learning

Model-free reinforcement learning has been widely used to solve various problems. Classical Q-learning (8) is one of the most famous RL algorithms. While popular in reinforcement learning, it has several limitations. It struggles with the exploration vs. exploitation trade-off, often failing to find an optimal balance between exploring new actions and exploiting known ones. As the action spaces (number of possible actions) grow larger, the algorithm faces the "curse of dimensionality," making it infeasible to store and update Q-values for all pairs. Additionally, Q-learning can suffer from slow convergence, especially in complex or sparse reward environments. When using function approximators like neural networks, the algorithm can become unstable and prone to issues such as overestimation bias. Q-learning also assumes a stationary environment, which limits its effectiveness in dynamic or changing environments. These limitations have led to the development of more advanced algorithms like policy gradients (9; 10) and actor-critic (11) methods.

The policy gradient algorithm is a class of RL methods that directly optimize the policy, which is the strategy used by the agent to select actions. Unlike value-based methods like Q-learning, which focus on learning a value function (e.g., how good an action is in a given state), policy gradient methods aim to learn a parameterized policy that maps states to actions. Below is a concise explanation of how it works.

1. **Policy Representation:** The policy is typically represented as a probability distribution over actions, often parameterized by a neural network with weights θ. This is called a *stochastic policy*, denoted as $\pi_\theta(a|s)$, meaning the probability of taking action a given state s.

2. Objective: The goal is to find the policy parameters θ that maximize the expected cumulative reward. The objective function $J(\theta)$ is defined as the expected return (sum of rewards) starting from the initial state and following the policy π_θ:

$$J(\theta) = \mathbb{E}\left[\sum_t R_t\right]$$

where R_t is the reward at time step t.

3. **Policy Gradient Theorem:** To optimize $J(\theta)$, the policy gradient theorem provides a way to compute the gradient of the expected return with respect to the policy parameters θ. The gradient is given by:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a)\right]$$

This means that the gradient is proportional to the gradient of the log-probability of actions weighted by the action-value $Q^\pi(s,a)$ under the policy.

4. **Updating the Policy:** Using the computed gradient, the parameters θ are updated using gradient ascent (or descent, depending on formulation), which incrementally improves the policy:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

where α is the learning rate.

Several popular algorithms build on the policy gradient method:

- **REINFORCE (9):** A basic policy gradient algorithm that uses the total return after each action as an estimate for the action-value.
- **Actor-Critic (11):** This method introduces a critic that estimates the value function $V^\pi(s)$, helping to reduce the variance of the policy gradient by using the advantage function $A^\pi(s,a)$, which measures how much better an action is compared to the average action.

In summary, policy gradient algorithms directly optimize the policy by adjusting its parameters to maximize expected rewards, making them versatile for tasks involving complex or continuous action spaces. You can find other RL algorithms in this textbook (7).

B.3 Modelling Cutting-Stock problem with Reinforcement Learning

You can refer to these papers to model the cutting stock problems (12; 13).