Exercise
Reinforcement Learning for Job Shop Scheduling with PPO

1. Objective

The goal of this exercise is to implement a reinforcement learning agent for the Job Shop Scheduling Problem (JSSP) using Proximal Policy Optimization (PPO) [2].

Students will design a compact vector state representation and train an MLP actor-critic policy. The agent will learn to construct feasible schedules by repeatedly selecting the next job or operation to dispatch. The trained policy will be evaluated on synthetic JSSP instances and Taillard benchmark instances [1] up to size 20 x 20.

2. Problem Description

A Job Shop Scheduling Problem consists of:

- n jobs
- m machines
- each job contains m ordered operations
- each operation must be processed on a specific machine
- each machine can process at most one operation at a time
- operations cannot be interrupted once started
- the objective is to minimize the makespan.

The makespan is defined as: $C_{max} = \max C_j$

where C_j is the completion time of job j. The optimization objective is: $min\ C_{max}$

3. Provided Data

Students will be given:

- synthetic JSSP instances up to 20 x 20
- Taillard benchmark instances [1] up to 20 x 20
- a parser or loading template for the data files
- optional starter code for basic instance representation.

Each instance should be represented using two arrays:

$$Processing\ times[j,k]$$

$$machines[j,k]$$

where j is the job index and k is the operation index within the job.

4. Required Environment

Implement a JSSP environment with the following least methods:

```
reset()
step(action)
get_available_actions()
get_state()
```

```
is_done()
compute_makespan()
```

The environment must construct schedules incrementally. At each step, the agent selects one valid job j. The environment then schedules the next unscheduled operation of that job at its earliest feasible start time.

For job j, let the next operation be k. The operation requires machine: $M\{j,k\}$

and processing time: $p\{j,k\}$

The start time is: $start\{j,k\} = max(job\ available[j], machine\ available[M\{j,k\}])$

The completion time is: $end\{j,k\} = start\{j,k\} + p\{j,k\}$

After scheduling this operation, the environment updates:

$$job\ available[j] <- end\{j,k\}$$

$$machine\ available[M\{j,k\}] <- end\{j,k\}$$

$$operation\ index[j] <- operation\ index[j] + 1$$

The episode ends when all operations of all jobs have been scheduled.

5. Action Space

The action is the choice of the next job to dispatch: $a_t \in \{0,1,\ldots,n-1\}$

Only jobs with remaining unscheduled operations are valid: $A\ valid(s_t) = \{j : operation\_index[j] < m\}$

Invalid actions must be masked. The policy must assign zero probability to invalid actions:

$$\pi_\theta(a \mid s_t) = 0, if\ a\ notin\ A\_valid(s_t)$$

In practice, this can be implemented by setting invalid logits to a very negative value before applying softmax.

```
logits[invalid_actions] = -1e9
probs = torch.softmax(logits, dim=-1)
```

6. State Representation

A fixed size vector representation must be sesigned. For instances up to 20 x 20, smaller instances may be padded to size 20.

6.1 Per-job features

For each job j, useful features may include:

- current operation index
- remaining number of operations
- remaining processing time
- job availability time
- processing time of the next operation
- machine required by the next operation
- whether the job is still active.

$$x_j = [op\ idx_j, remaining\ ops_j, remaining\ work\ j,\ job\ available_j, next\ processing\ time_j, next\ machine_j, active_j]$$

6.2 Machine features

For each machine q, useful features may include:

- machine availability time
- current load

- remaining assigned work
- utilization estimate.

$$z_q = [machine\ available_q, machine\ load_q, remaining\ machine\ work_q]$$

6.3 Global features

- current partial makespan
- number of scheduled operations
- fraction of completed operations
- average machine availability
- maximum machine availability.

The final state vector may be written as: $s_t = [x_1, \dots, x_{20}, z_1, \dots, z_{20}, global\ features]$

All features should be normalized. Recommended examples include:

$$p_{tilde}\{j,k\} = p\{j,k\} / max(p)$$

$$remaining\ work_{t,ilde}j = remaining\ work\ j / sum\{j,k\}\, p\{j,k\}$$

$$machine\ available_{tilde}\ q = machine\ available\ q / estimated\ upper\ bound$$

7. Reward Design

Students must implement at least one sparse reward and one shaped reward.

7.1 Sparse reward

The simplest reward is given only at the end of the episode:

$$r_t = \begin{cases} 0, if\ the\ episode\ is\ not\ finished \\ -C_{max}, if\ the\ episode\ is\ finished \end{cases}$$

This reward is aligned with the final objective, but it may be difficult to learn because feedback arrives only at the end of the schedule.

7.2 Shaped reward

A shaped reward may provide feedback after each dispatching decision. One possible reward is:

$$r_t = -\Delta\, C_{max,t} - \lambda_{idle}\, \Delta\, idle_t$$

where: $\Delta\, C_{max,t} = C_{max,t} - C_{max,t-1}$

and Δ idle is the increase in machine idle time caused by the action.

8. PPO Agent

PPO must be created with an actor - critic neural network.

The actor outputs action logits: $logits = f_\theta(s_t)$

After action masking, the policy is obtained by applying softmax to the masked logits:

$$\pi_\theta(\alpha_t \mid s_t) = softmax(masked\ logits)$$

The critic estimates the value function: $V_\theta(s_t)$

Discounted returns should be computed as: $G_t = \sum_{\kappa=0}^{T-t} \gamma^\kappa r_{t+k}$

The advantage can then be estimated as: $A_t = G_t - V_\theta(s_t)$

The PPO ratio can computed by $r_t(\theta) = \pi_{\theta,new}(\alpha_t|s_t) \,/\, \pi_{\theta,old}(\alpha_t|s_t)$

The clipped actor loss function is calculated by $L_{clip}(\theta) = E_t[min(\, r_t(\theta)\, A_t, clip(r_t(\theta), 1 - \varepsilon, 1 + \varepsilon)A_t)]$

The critic loss function is calculated by $L_{V(\theta)} = (V_\theta(s_t) - G_t)^2$

The entropy bonus is calculated by $H(\pi_\theta) = \sum_\alpha \pi_\theta(\alpha_t|s_t) \log \pi_\theta(\alpha_t|s_t)$

The total loss is calculated by $L = -L_{clip}(\theta) + c\_v\, L_{V(\theta)} - c\_e\, H(pi\_theta)$

9. Experiments

9.1 Training

Train the PPO agent on synthetic JSSP instances. A suggested curriculum is:

- 6 x 6
- 10 x 10
- 15 x 15
- 20 x 20

Students may either train one model directly on mixed-size instances or train progressively from smaller to larger instances.

9.2 Evaluation

Evaluate the trained agent on:

- unseen synthetic instances
- Taillard benchmark instances [1] up to 20 x 20

For each evaluated instance, report the achieved makespan: $C_{max}\ PPO$

If best known solution values are provided, compute the optimality gap:

$$Gap(\%) = ((C_{max}\ PPO - C_{max}\ BKS) \,/\, C_{max}\ BKS) * 100$$

If best-known solution values are not provided, report the achieved makespan and clearly state that no reference value was available.

9.3 Analysis questions

Discuss the following questions:

- Does the PPO agent improve during training?
- Does the shaped reward make learning easier than the sparse reward?
- Does the trained policy generalize from synthetic instances to Taillard instances?
- What happens as the instance size increases?
- What are the main limitations of the implemented approach?

10. Implementation Requirements

Everyone must submit at the end of the exercise

- Python source code
- instructions to reproduce training and evaluation
- training logs or saved result files
- a short written explanation of the environment, PPO implementation [2], and experimental findings.

Recommended project structure:

```
assignment_jssp_ppo/
|
├── data/
|   ├── synthetic/
|   └── taillard/
|
├── env/
|   └── jssp_env.py
|
├── agents/
|   ├── actor_critic.py
|   └── ppo.py
|
├── train.py
├── evaluate.py
├── utils.py
|
└── README
```

11. Suggested Hyperparameters

The following values are suggested starting points. Students may tune them depending on the observed training behavior.

Hyperparameter	Name	Suggested value
Discount factor	γ	0.99
PPO clipping parameter	ε clip	0.2
Learning rate	learning_rate	3e-4
Entropy coefficient	entropy_coef	0.01
Value loss coefficient	value_coef	0.5
PPO update epochs	num_epochs	3
Batch size	batch_size	32 or smaller, depends PC
Hidden dimension	hidden_dim	128 or smaller
Hidden layers	hidden_layers	2

Suggested actor-critic network:

```
Actor MLP:
    input_dim = state_dim
    hidden_dim = 128
    hidden_layers = 2
    output_dim = max_jobs

Critic MLP:
    input_dim = state_dim
    hidden_dim = 128
    hidden_layers = 2
    output_dim = 1
```

The actor outputs one logit per job: $logits[j]$

The critic outputs one scalar value estimate: $V(s)$

References

[1] Taillard Instances https://optimizizer.com/TA.php

[2] Proximal Policy Optimization Algorithms https://arxiv.org/abs/1707.06347