Singapore Management University

School of Computing and Information Systems

2024/2025 Semester 2

CS606: AI Planning and Decision Making

Assignment 2 Optional **Challenge Question** (Due 16 March 2025 1159PM)

Maximum bonus marks = 5 marks

**Submission Instructions**

In the same code folder, you are to revise and include the following:

(1) the revised trained model (under the folder src/dr\_alns/trained\_models)

Note: We will not run python dr\_alns\_trainer.py, but we will examine your code

(2) python codes (namely, psp\_AlnsEnv.py and dr\_alns\_main.py)

**We will run your code in command line. Please make sure your code can be executed as** python dr\_alns\_main.py <instance> <random\_seed\_list>

**Problem Statement**

In the recent years, machine learning methods have been applied to solve mathematical programming problems. Reinforcement Learning (RL) is a branch of machine learning where an agent optimizes its actions in an environment to maximize cumulative rewards through trial and error. Due to the complex and resource intensive task of selecting proper configuration of selection and acceptance parameters, Reijnen *et al* 2024 propsed a Deep Reinforcement Learning (DRL) framework to learn, based on the state of the search, to configure ALNS.

**Question**

Implement a DRL approach to configure ALNS in python using the code template provided for the same problem (see **Annex** below for details). Similar to the Main Question, you should test your code against the set of problem instances given in the folder psp\_instances.

You will use the [gymnasium library](https://www.gymlibrary.dev/index.html), which provides standardized environments for training RL across a variety of tasks employing an [Actor-Critic RL Algorithm](https://www.gymlibrary.dev/index.html) optimized with [Proximal Policy Optimization](https://jonathan-hui.medium.com/rl-proximal-policy-optimization-ppo-explained-77f014ec3f12) using the [stable\_baselines3](https://stable-baselines3.readthedocs.io/en/master/) package. The **Actor** and **Critic Network (**which are both neural networks) will be handled by the stable\_baselines3 package, so you will not be required to code it. The **Actor** in this case will be responsible for selecting **good destroy and repair operators**, while the **Critic** will **value** the **Actor’s Actions** based on the current **state information.**

**Reference:**

Reijnen, Robbert; Zhang, Yingqian; Lau, Hoong Chuin; Bukhsh, Zaharah. Online Control of Adaptive Large Neighborhood Search Using Deep Reinforcement Learning. Proceedings of the *34th International Conference on Automated Planning and Scheduling,* 2024.

**Marking Criteria**

Your work will only be graded if you receive at least 10 marks for part (b) of the Main Question. That is, if your vanilla ALNS code can produce feasible solutions *and* the objective value of the final solution for the given dataset S2.json is strictly better than our baseline.

The bonus 5 marks will be awarded based on a curve that compares your results with the corresponding results obtained by your classmates.

We will be deriving your results based on both the given dataset S2.json and a hidden dataset, which we will generate by running the following:

python dr\_alns\_main.py <instance> <random\_seed\_list>

For each instance, we will take the median of 10 objective values obtained with 10 different random seeds given in the <random\_seed\_list>.

**Annex**: **Explanation of Deep RL ALNS Code Template**

For this Challenge Question*,* you are provided additionally with 1 skeleton file (psp\_AlnsEnv\_skeleton.py) and 2 main files (dr\_alns\_trainer.py, to train RL model and dr\_alns\_main.py, to run alns iterations after model training). Along with this, the src/dr\_alns directory contains other files that will be used for the DRL implementation. Refer detailed description in the comments in the python files. The places that will need to be modified will include a // within the comment (e.g. # // Modify with your name).

**1. src/dr\_alns/dr\_configs/pspAlnsEnv.yml** – This **file** contains training configurations for model training. Refer to the comments within the file and stable\_baselines3 documentation for these parameters which are used as inputs.

**Some configurations you might want to explore with:**

* **Environment Parameters**
  + environment.iterations: number of ALNS iterations for model training
  + instances: instances to train on: [<start\_instance\_number>, <end\_instance\_number>]
* **PPO Parameters**
  + n\_step: increasing may lead to slower training, but may increase training stability (less fluctuations
* batch\_size: smaller mini\_batches leads to noisier gradients but can potentially help escape local minima
* gamma: higher gamma places more weights on short-term rewards
* clip\_range: Decreasing clip\_range can prevent drastic policy updates, promoting more stable training.
* learning\_rate: Decreasing learning\_rate may lead to more stable but slower training.
* max\_grad\_norm: Decreasing max\_grad\_norm limits the gradient's norm, which helps in preventing large, destabilizing model weight updates.

**You do not need to modify anything in this file if you wish to stick with the default values.**

**2. src/dr\_alns/trained\_models** – This **directory** is where the RL trained model will be saved in. The model has been configured to be saved in folders containing information about the training. Hence when you run dr\_alns\_trainer.py, do not be alarmed when there are sub folders created.

**The folder will typically look like this:**

|- trained\_models

| |- pspAlnsEnv

| | |- 1\_PPO\_ActorCriticPolicy\_2000000\_10\_02-03\_16-10

| | | |- intermediate\_models

| | | | |- **… some intermediate models**

| | | |- tensorboard\_logging\_1

| | | | |- **… some logs**

| | | |- model.zip

The latest model is stored in model.zip and the path to this file will need to be set within dr\_alns\_main.py when running the ALNS iterations with the RL trained model.

Note that 1\_PPO\_ActorCriticPolicy\_2000000\_10\_02-03\_16-10 is named in the following manner so that you do not mix up different models being trained.

**It is named with some of the parameters defined in the** pspAlnsEnv.yml:<IDX>\_**<MODEL>**\_**<POLICY>**\_**<N\_STEPS>**\_**<N\_WORKERS>**\_**<DATE>\_<TIME>**

**DO NOT modify anything for this directory**

**3. src/dr\_alns/Trainer.py** – This **file** contains the class Trainer used to create, load and train the RL model using stable\_baselines3. Training is performed on the training datasets in psp\_instances/train.

**Functions**

* create\_env: Registers and initializes a gym environment for multiple workers. The environment created can be parallelized (SubprocVecEnv) or single-threaded (DummyVecEnv)
* get\_parameters: Loads and returns a yaml configuration based on the configuration file path provided.

**Trainer Class Methods**

* \_\_init\_\_: Initializes the Trainer class with environment and configuration settings. It sets up paths for saving trained models
* create\_model: Sets up the training environment and initializes an RL model using configurations from stable\_baselines3. It creates a directory for the model based on the training parameters and the current date and time.
* \_create\_model\_dir: Creates a unique directory for saving the model within a specified path, managing directory names based on the configuration and ensuring directories exist.
* \_save: Saves the trained model and configuration file to the disk, creating necessary directories and printing save locations.
* train: Manages the training process for the model, including setting up callbacks for saving intermediate models and handling manual interruption. It adjusts the save frequency based on the number of workers and starts the training, ensuring final model saving and logging training steps.

**DO NOT modify this .py file**

**4. psp\_AlnsEnv\_skeleton.py** – The **file** is where the custom RL environment is defined**.** Change the name of this file to psp\_AlnsEnv.py.

Every environment in RL requires a few standard functions such as reset() and step(), as such, we use the gym package’s base environment gym.Env to create your own custom environment. Each step that is taken in the environment would return state features which our model (Actor) will ingest and select actions to take.

The environment plays a crucial part in the RL training as our model will learn through interaction with our environment on what optimal actions to take. We will break down the pspAlnsEnv class methods.

**For this code to work, you need to import your implemented operators from the Main Question.**

**You do not need to modify the environment if you wish to stick to the default.**

**Class Methods**

* init – Initializes the gym environment by settings up key parameters.
  + initial\_solution, best\_solution, current\_solution: Placeholders for tracking the initial, best, and current solutions within the optimization process.
  + improvement,current\_improved: Metrics to track the progress and efficacy of the optimization process
  + reward: Total accumulated reward for the current episode.
  + done: A boolean indicating whether the current episode has finished.
  + episode: Counter for the number of episodes processed.
  + iteration: Counter for the number of generations (iterations) within the current episode.
  + max\_iterations: Maximum allowed generations per episode, derived from the configuration.
  + action\_space: Defined as a multi-discrete space where each dimension can take on a limited set of discrete values, influencing the number and type of actions that an agent can take at each step.
  + observation\_space: Configured as a continuous space with defined shape and bounds, representing different measurable features of the environment state that the agent observes to make decisions.

Within the \_\_init\_\_ function, you are to add any required attributes that will assist in defining your **additional state observations** within the \_step() and \_reset() function. We have included a simple observation\_space that we expect you to modify to include **new states** that may help the model to train better. **This is the function for you to implement.**

* make\_observation – Generates and returns the current state of the environment: Similar to \_\_init\_\_, you will need to modify to include your **additional state observations** here. **This is the function for you to implement.**
* reset – Resets the environment to a random initial state using Parser and using the same random initialization PSP.random\_initialize(SEED) as that used in the Vanilla ALNS implementation. Make sure to **import** and **add** the destroy and repair operators **implemented** in operators.py here after importing them. On top of this, you need to also reset your **additional state observations** if any here as well. **This is the function for you to implement.**
* step – Advances the environment by one step using the specified action, applying destroy and repair operations, and updates the environment state and solutions based on the action's outcome. In this function, self.reward\_and\_update()function is called, updating the rewards earned (if any) for the particular state and action pair. If you have defined any **additional state observations**, this is the function where you determine how the observations would change per step in the environment. **This is the function for you to implement**
* reward\_and\_update – Updates the environment's state based on the outcome of candidate solutions, adjusting rewards and tracking improvements or degradations. You are encouraged to modify the reward function which plays a crucial role RL training. Feel free to add additional helper functions or class attributes if required. **This is the function for you to implement**
* consider\_candidate – Evaluates a candidate solution against the current and best solutions, deciding whether to accept it based on its objective value. You are encouraged to try different acceptance criterias. You can refer to the **alns package** for their implementation in src/alns/criteria. Feel free to add additional helper functions or class attributes if required. **This is the function for you to implement**
* run – Executes a series of episodes using a trained model
* sample – Runs the environment with randomly sampled actions (used for testing)

**5. dr\_alns\_trainer.py** – The **file** contains the model training code, **DO NOT modify**

**6. dr\_alns\_main.py** – This is the main **file** that runs the Deep RL ALNS Iterations

**Lines to be modified:**

* change the path model\_path to where your model.zip is located (under src/dr\_alns/trained\_models). Refer to **2. src/dr\_alns/trained\_models** for more information
* save\_output('YourName\_DR\_ALNS', solution, 'solution')

– change 'YourName' to your full name

**Running the programs:**

**1) Run Deep RL model training in command line**

python dr\_alns.trainer.py

**2) Run Deep RL ALNS in command line**

python dr\_alns\_main.py <instance> <random\_seed\_list>

e.g. python dr\_alns\_main.py S2.json “606,707,808”