CLD771 Minor Project Report

Evaluation of B.Tech. Minor Project

Name: Samarth Bhatia Entry Number: 2019CH10124

Project Title: Using Deep Reinforcement Learning for scheduling gasoline blending and distribution (SGBD)

Supervisor: Prof. Hariprasad Kodamana

Report 2: 15 October, 2021

Summary of work done:

- I have started to implement an environment similar to the one used in Paper 1, I have implemented the `observation_space`, the `action_space`, the reward function, the step function (subject to changes) and the reset function of the environment.
- The environment is compatible with `gym` and will be available as a python package when finished (with less detail) and is regularly being updated on Plutonium-239/deeprl-in-sgbd .

  The environment can be described as follows:

1. `observation_space`: the space of all possible values that can be observed. In our case these are inventory levels in all the different units (component tanks, blenders and product tanks), the amount of changeover in the blenders and the product tanks, tardiness in orders, flow rates and amounts flowed in between different units.

I have modeled all these as continuous variables using the `gym.spaces.Box` class

```
gym.spaces.Box(low= 0, high= 1, shape= (2,3))
```

This makes a continuous space ranging from `0` to `1` in the form of a `5x3` matrix. An example value of this can be:

$$\begin{bmatrix} 0.23 & 0.47 & 0.88 \\ 0.05 & 0.19 & 0.61 \\ 0.45 & 0.27 & 0.37 \\ 0.01 & 0.10 & 0.97 \\ 0.00 & 0.29 & 0.11 \end{bmatrix}_{5\times3}$$

2. `action_space`: the space of all possible actions that can be taken. Currently they are modelled as binary variables representing whether material is transferred between
3. component tanks `{i}`and blenders `{n}` : matrix of shape `i*n`
4. blenders `{n}` and product tanks `{j}` : matrix of shape `n*j`
5. product tanks `{j}` and order tanks `{o}` : matrix of shape `j*o`
6. and if product `{p}` is assigned to blender `{n}` : matrix of shape `p*n`

This is done efficiently using the `gym.spaces.MultiBinary` class, which allows us to create a one-hot/sparse matrix as shown below:

```
gym.spaces.MultiBinary(shape= (4,2))
```

This can be sampled at random using the `.sample()` function which will give us:

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}_{4\times2}$$

3. The `reward` function: For our environment and in Paper 1, the objective to minimize has been taken as the sum of the component costs, changeover costs and tardiness costs.
   So we can set the reward to be `-ve` of this sum, as in Reinforcement Learning, we try to maximize the reward.

$$\text{reward} = -(\text{component costs} + \text{changeover costs}[\text{blenders}] + \text{changeover costs}[\text{product tanks}] + \text{tardiness costs})$$

$$\text{maximize reward}$$

4. The `step` function: This is the main function in the environment; which takes an action (any possible action as defined in `action_space`) and propagates it through the whole setup for one timestep. This includes updating all inventory levels, changing value of flow rates, checking what product has been formed and packaged for being sent to the order, and all other ***dynamic*** aspects of **SGBD**.

   In our environment, the `step` function takes an `action` in the one-hot/sparse matrix form and converts it into meaningful *changes* in the system like material being sent between different units, different products being mixed in blenders at each timestep, the changing of the components in blenders etc. This has been implemented almost completely.
   The current implementation is not final, and can change as per requirements based on modelling improvements or if required by some agent

---

Future Plan:

- I am implementing the order generating system; the code would make orders for the given scheduling horizon. It will be pretty simple and I will test and figure out an order which is stable and has well distributed orders throughout the horizon.
- I have implemented about 90% of the `step` function. This means I will be able to complete the environment in a week, after which I will be able to implement and test out a lot of agents and RL algorithms and also establish baselines.
- Baseline results will be established considering the **Graphical Genetic Algorithms (GGA)** in Paper 1, and also using industry-standard packages like `stable-baselines` , `catalyst` (specialized for DRL using PyTorch for the NN part), etc.
- I will also look into ds4dm/**ecole**, a `gym`-like package focused more on **Mixed Integer Linear Programming (MILP)** algorithms.

  > An advantage of using the `gym` package from the start is that I can almost directly import my environment without many changes in `ecole` instead of having to rewrite all the code for a new API, as it is built to mimic `gym`'s API.

- After establishing baseline scores I will try different algorithms such as **A2C (Advantage Actor-Critic)**, **A3C (Asynchronous Advantage Actor-Critic)**, **PPO (Proximal Policy Optimization)** (which is a **Policy-Gradient** method) and others accordingly. I don't see much scoe for Deep Q-Networks for SGBD since most of the variables and actions are in the continuous space.