## Part 1 Define the Environment:

**Introduction - Modified Water Supply Environment**

We've implemented a modified environment simulating a market player in the water market.

The agent is tasked with managing water supply for a household, where he is required to meet the household demands and for this end, he is allowed to choose how much water to purchase from each of two different sources (at the same hour), on an hourly basis, each with distinct pricing, while minimizing total expenses.

The environment is composed of cycles, each cycle can be thought as 1 week which is composed of hours: the steps in the environment.

**The demand:**

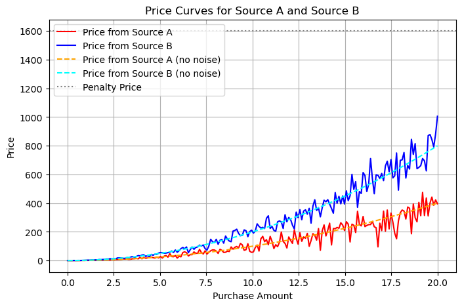
Each hour (step) in the week (cycle) is a normal random variable. Each have different mean but the same variance.

The daily demand is composed of 2 peaks.

For part 3 Env, The weekly demand for Sunday-Friday follows the same hourly daily expectations, apart from Saturday, where the demand is especially low (this is in accordance with real world data).

**The pricing:**

Each Source (Source\_A, Source\_B) has a different base price. While Source\_B is pricier.

The price of each purchase is non-linear. It is determined by the volume of the purchase in a single hour. The more volume, the steeper the price. We've speculated this setting will drive the agent to find the right combination to make its purchases from source\_A and the "initially expensive" Source\_B.

In Part 3, The base Price of the two sources is also dependent on the hour of the day, where in certain hours the price is higher.

Environment Simplification and experimentation (Applicable to part 3):

Our environment implementation support customization that allow to simplify the environment or to make it compatible with various RL algorithms.

Resolution - When creating the environment, there is ability to choose with what time resolution will the environment run. i.e., one such setting is, will it have 168 hours in a week or any other number of hours, e.g. 14. This allow to run a simplified version of the environment on RL algorithms that are resource heavy for complex environments, e.g., q-learning, which for a large discrete environment can be very memory and time expensive.

Discretization – The environment allow to declare that it's Observation Space and/or Actions Space will be discrete. As some RL algorithm, require this.

Normalization - for the non-discrete Obs’, and/or Acts’ spaces, there are settings to normalize them. i.e., the observation features vector will have values in [0,1] and the same for the action space.

**State Variables:**

The state at each timestep includes five variables, capturing both the internal state of the agent (water level) and the external environment conditions (timestep, demand, prices):

MAX\_WATER\_LEVEL = E.G. 300

Water\_level: [0, MAX\_WATER\_LEVEL]

Price\_A E.g.: [1,2] (The base price of source A, For part 3)

Price\_b E.g.: [1.5,3] (The base price of source B, For part 3)

Demand [0,300] (in our environment, having a full reservoir is always enough for an hourly demand). (for part 2 the range is smaller: [0,20])

Current\_time E.g: [0,167] The current hour in the cycle (the week) (for part 2 the range is smaller (24)

**Action Space:**

Buy\_from\_A E.g.: [0,300] Volume of purchase from source\_A in the current timestep.

Buy\_from\_B E.g.:[0,300] Volume of purchase from source\_B in the current timestep.

**Dynamics**

**The price function:**

Takes a base price for a water source, and insert to a non-linear function. For part 2:

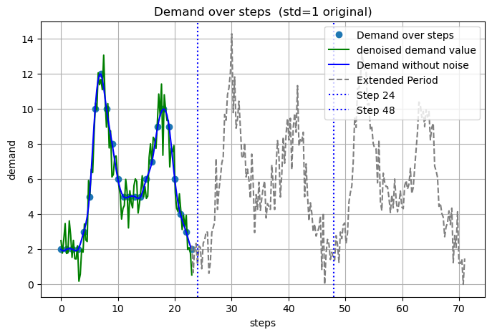
Capped at a maximum parice dictated buy a large penalty price for not meeting a demand.

**Reward function:**

The reward function is:

(In part 2 we normalized this reward by dividing by Penalty price, to have nicer values)

This means the reward function is negative and the agents will the try to maximize it by minimizing it's expenses.



Weekly(Cycle) demand:

As stated earlier, each hourly demand is a normal random variable with a different mean and the same variance.

## Part 2 Training and evaluation results with Gymnasium models:

We have used the environment to train models each for 200,000 steps (in the tensorboard graphs we can start to see convergence pattern at about this amount of steps) with the following models: DQN, A2C, SAC, PPO.

A black background with white text

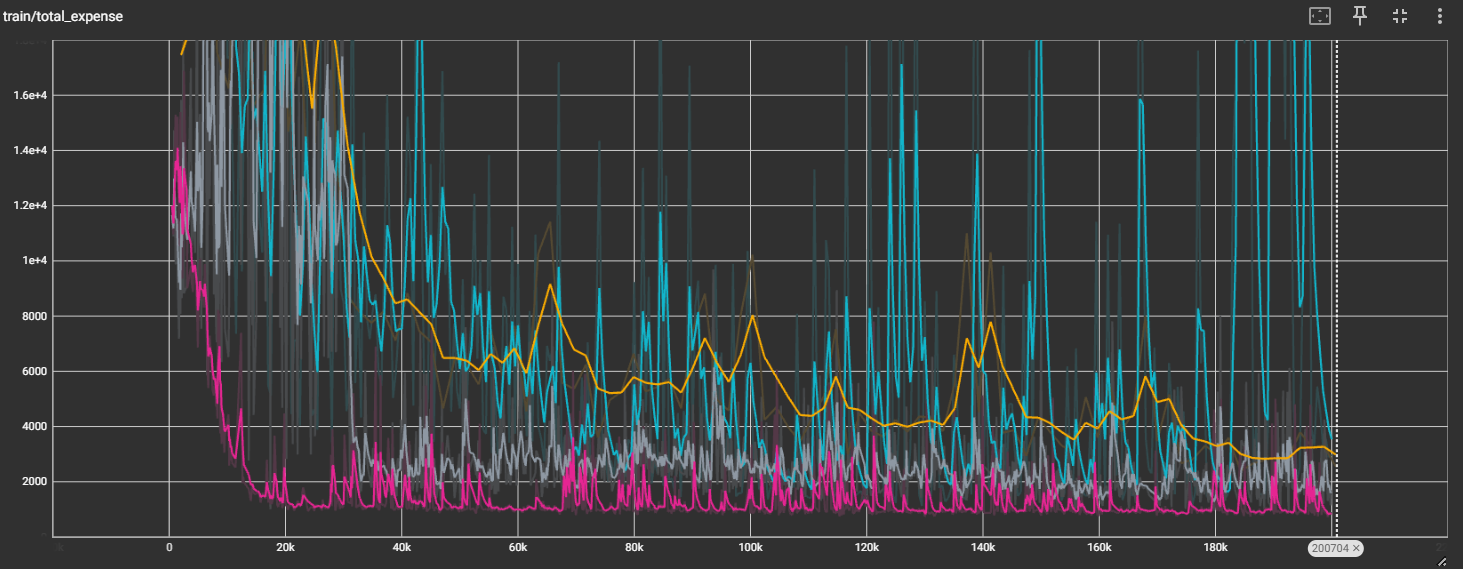
AI-generated content may be incorrect.Reward over time: **(1)**

A graph with colorful lines

AI-generated content may be incorrect.

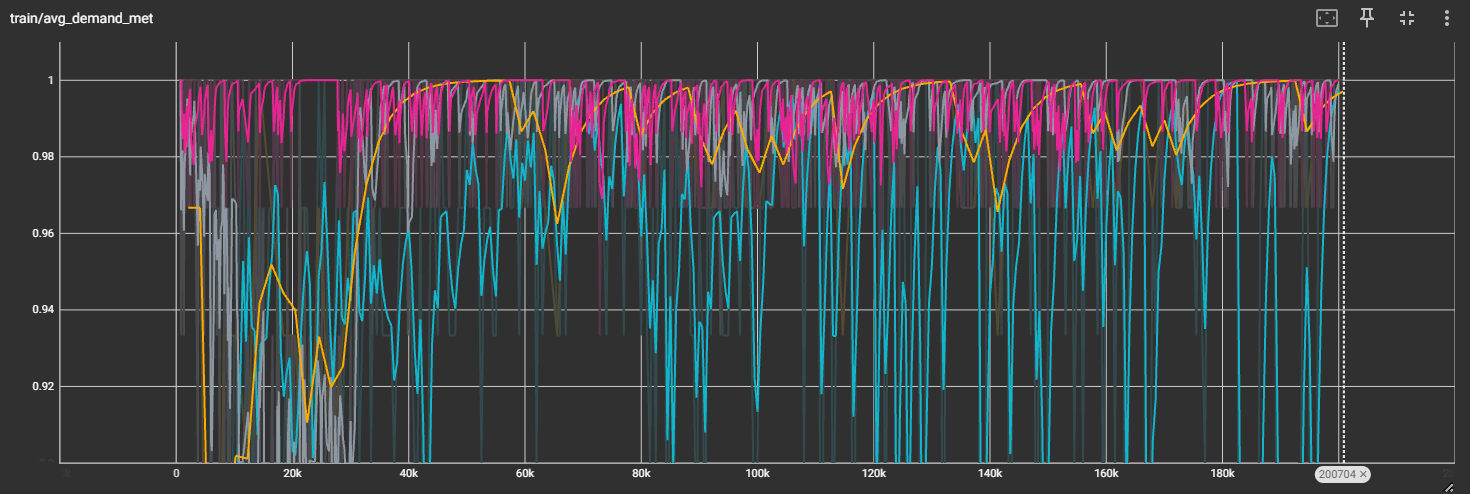
A black background with white text

AI-generated content may be incorrect.Actual Expense(un-normalized reward, see more fine details when meeting the demand):**(2)**



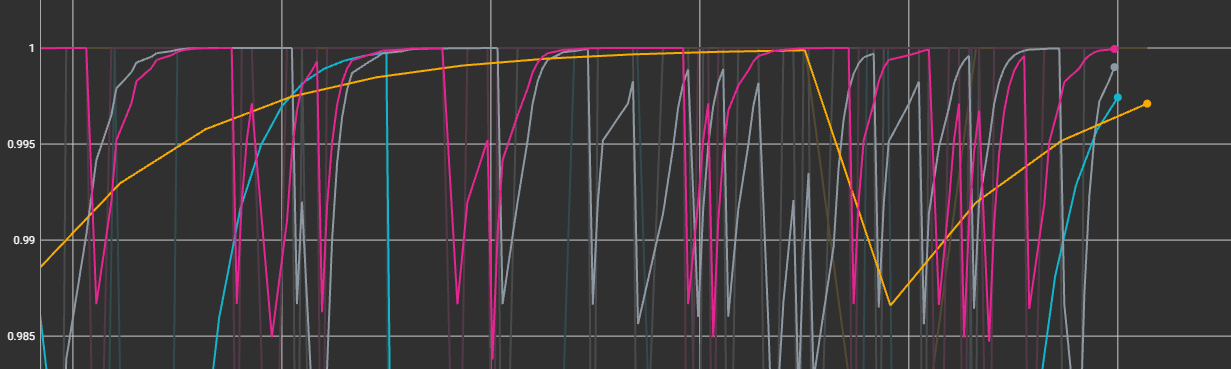
A black background with white text

AI-generated content may be incorrect.Percentage demand met: **(3)**



Closeup of the final section:

A black background with white text

AI-generated content may be incorrect.

Evaluating robustness - using **demand with noise std=3** (instead of std=1): **(4)**

(high expense values = Worse)

Models Evaluation:

A screenshot of a graph

AI-generated content may be incorrect.DQN: Mean Expense: 45.80 +/- 197.66, A/B Ratio: 1.76

A2C: Mean Expense: 112.36 +/- 238.49, A/B Ratio: **(inf)**

SAC: Mean Expense: 32.96 +/- 149.04, A/B Ratio: 1.66

PPO: Mean Expense: 85.31 +/- 242.02, A/B Ratio: **(inf)**

Robustness Models Evaluation:

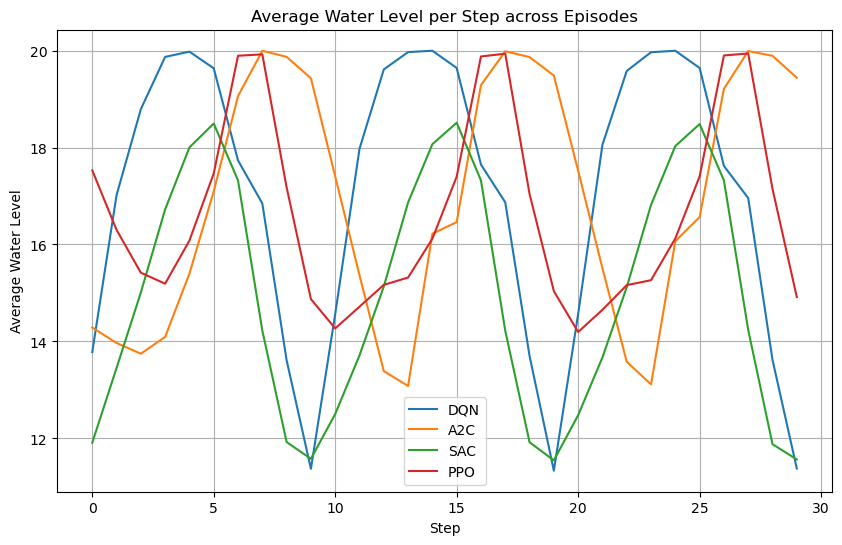
DQN: Mean Expense: 97.89 +/- 483.39, A/B Ratio: 1.62

A2C: Mean Expense: 164.95 +/- 539.13, A/B Ratio: **(inf)**

SAC: Mean Expense: 100.74 +/- 587.07, A/B Ratio: 1.61

PPO: Mean Expense: 125.20 +/- 431.36, A/B Ratio: **(inf)**

Average water level at each step by model (interaction with demand requirements): **(5)**



Discussion:

**Fig (1):** all the models converged, some to better rewards. We can see, that A2C and PPO look very similar, as the algorithms are similar. PPO has restrication on it’s policy updates, that makes it smoother.

Both of these are On-Policy algorithms (doesn’t use old trajectories). We can see that A2C has large variance toward the final episodes, where our Off-Policy algorithms (DQN, SAC) look very stable.

SAC converged to the best value, while DQN came close 2nd.

**Fig (2):** in this figure we see similar results as now the reward isn’t normalized

**Fig (3):** here we see that aside from A2C all the models manage to meet the demand on a very high percentage (>99%), which is expected as failing to do so will result in a high reward.

We can see the models try to explore new actions but are punished harshly when those changes lead to unmet demand and then return/find policies that allow the demand to be met. This behavior causes fluctuations.

**Fig (4): (i)** All the models performed worse (as expected) when the demand become more erratic. SAC which has been the best algorithm, now perform like the DQN on average but with higher variance. We can see the variance in the expense increase drastically across all the models.

**(ii)** Only DQN and SAC (Off-Policy) manage to find a balance between buying from Source\_A and Soure\_B. the balance which is about 1.6-1.8 ration toward Source\_A. The other On-Policy, chose only to buy from source\_A, The initially cheaper one.

Interestingly In the robustness test they perform very alike in that regard (1.6), which might explain the similar results in the expenses.

SAC ratios haven’t changed much in the robustness test.

**Fig (5):** By looking at the average water level per step, we can see how the model interacts with the demand requirements. SAC menages to keep a lower water level and get great results. Buying small amounts is cost effective, and although this graph doesn’t show it, we can deduce SAC keep its purchases small.

Looks like SAC found a good balance between A and B and also The right amount of water to buy each step.

A2C and PPO are similar in the phase and in the behaviour of trying to maximalize the water level (“playing safe”).

DQN also "plays it safe” buy buying water to the maximum, but its phase is more similar to the SAC model.

## Part 3: Research a New Training Paradigm

In this section, we introduce a novel hybrid reinforcement learning paradigm designed to work on a relatively large discrete action space.

Direct application of tabular q-learning and DQN methods poses challenges in terms of memory requirements and convergence speed, as large action spaces require a larger Q-table in the tabular case, or a more complex architecture for the neural network in the DQN case. (Because the network needs to converge on more outputs)

To overcome this issue, we propose a two-layer model that combines the fast convergence and stability of discrete Q-learning on smaller action spaces with the fine-grained optimization capability of Model Based approaches (In our case, MCTS).

Our approach involves a two-step decision process:

1. **Discretization and Range Selection:**  
   We first discretize each dimension of the action space. By partitioning each action into intervals —each represented by its lower bound—we reduce the number of actions to a more manageable size. For example, an action originally represented as a tuple (12, 33) might be discretized into ranges such as (10–20, 30–40). A tabular Q-learning agent, employing an epsilon-greedy strategy, then selects one of these discretized buckets. This reduction not only minimizes the memory footprint by using a smaller Q-table but also enhances convergence, as the learning process is performed on a reduced and simplified action space.
2. **Refinement via MCTS:**  
   Once a discretized range is chosen, we refine the action within that range using MCTS. The MCTS search operates within a smaller horizon – and the selection is done via UCT selection with advantage, where the advantage baseline is the average reward of the parent node. The final action is obtained by adding the refinement (i.e., the difference from the bucket’s lower bound) to the discretized selection, producing an action from the original action space. For example, If the action space is 100 X 100 and we discretize the action space to 10 equally sized ranges. The q-learning layer will choose a single range for each item in the tuple. Let’s say (30-40, 50-60) and the MCTS layer will select how much to above the lower bound of each range the final action will be. So MCTS selects (2, 3) the final action will be (32, 43).

This hybrid approach offers 3 main advantages:

* **Enhanced Sample Efficiency:** By narrowing down the action space through discretization, the Q-learning component converges more rapidly.
* **Improved Performance and Stability:** MCTS provides a more detailed search within each bucket, reducing error and adapting exploration to more refined aspects of the environment.
* **Memory Efficiency In Tabular Q-Learning:** Using this method we reduce the size of the Q-Table to (size\_of\_observation\_space) X (number\_of\_ranges) rather than (size\_of\_observation\_space) X (size of action space) this could be significant if the action space is really large.

However this approach introduces 2 main challenges:

* **High Computational Costs:** Running MCTS rollouts during each step is an order of magnitude more expensive than the baseline tabular approach.
* **Short Horizon for Refined Actions:** As we run MCTS search after each step in the refinement process during training having a deep MCTS tree would require many rollouts to converge to a good selection, which in turn require more computational resources. Therefore this model will struggle in environments where small deviations between actions may have significant long term impact.
* **Requires A simulator:** As we are using MCTS for refinement we are leveraging our knowledge of the environment that we built to simulate rollouts. This may not always be possible, so a different simulator will need to be provided, and the quality of the model will be (at least partially) dependent on the quality of the simulator.

Training the model:

Our initial implementation of the model uses a Q-table for the coarse policy prediction. In future iteration we could use a deep neural network in the same vain as DQN in order to generalize the approach to continuous observation spaces (or just more complex discrete ones)

The training method is divided into two phases:

* **Coarse Policy Training:**  
  The agent learns a discrete high-level policy using tabular Q-learning on a discretized action space. This phase builds a stable Q-table that selects coarse action buckets.
* **Refinement with MCTS:**  
  In subsequent episodes, when a bucket is chosen, MCTS is used to search within that bucket over a short horizon. The MCTS output refines the action by providing a precise offset, which is added to the bucket’s lower bound to form the final action. The Q-table is updated based on the reward received from executing this refined action.

Training the coarse policy before refining with MCTS reduces the computational costs of training as we have less MCTS rollouts. Since our MCTS uses the current Q-table during rollouts, MCTS predictions are not as useful for model convergence before the coarse policy has sufficiently converged.

We evaluated this new model against the tabular Q-learning method learned in class as baseline.

Training Graphs:  
  
A graph with a line graph and numbers

AI-generated content may be incorrect. A graph of a graph showing the results of a episode

AI-generated content may be incorrect.

As we can see from the graphs, The new model starts converges much faster than the tabular baseline, as converging on the discretized action space is easier than converging on the original space. After episode 200 the new model starts the refinement stage, improving the results even further, while hurting the model’s stability (we can see that the confidence interval for the loss is much larger)

Behavior in the environment:

A graph of a graph showing a number of purchases

AI-generated content may be incorrect.A graph of water level

AI-generated content may be incorrect.

In the graphs to the right we can see that the new model is buying an order of magnitude less water than the Tabular Q-learning model, while also maintaining much higher water levels throughout the episode. This is very efficient as the way we set up the environment it is more expensive to buy a lot of water at the same time than to buy small amounts each timestep.

Maintaining high water levels allows the model to avoid punishments for not supplying demand.

A graph of blue and orange bars

AI-generated content may be incorrect.A graph of blue and orange bars

AI-generated content may be incorrect.We can also see that the choice of water source in the finetuned model is better than the baseline model as it more often buys more water from the cheaper source A than the more expensive source B.

Potential improvements:

* **Improved MCTS implementation**: While limiting the refinement step to only a small portion of the learned episodes, these episodes are significantly more expensive. A more robust implementation of the MCTS calculation that includes parallelizing the MCTS rollouts as well as introducing dynamic programming may alleviate this cost.
* **Deep Neural Network**:In the same vain as we discussed the transition between tabular Q-learning to DQN we can introduce a neural network to approximate the value of a state instead of the Q-table. This would allow this model to be viable on more refined (or even continuous!) observation space.
* **Alternatives for refined policy model**:As discussed above, the main challenges in this approach are associated with the calculation cost of MCTS, as well as the requirement for a simulator. Experimenting with different model based approaches such as PETS, MPC and dyna-like synthetic rollouts could prove to be very effective.