# AI 4102: Reinforcement Learning and Autonomous Systems

## Programming Assignment #2 (for Module 2)

Release date: 6th October, 2024, 5:00 pm (IST)

Due date: 29th October, 2024, 11:59 pm (IST)

Maximum Score: 100 marks (this assignment is graded)

Read the following before you move forward:

1. This programming assignment is worth **10% of the total marks** of this course.

2. **Late policy:** You can be late by a maximum of 3 days. Your assignment will not be accepted more than 3 days after the due date. The **actual score** and **received score** (received score is actual score minus penalty for late submission) are related as follows:

$$\text{received score} = \begin{cases} \text{actual score} & ;\text{late by } 0 \text{ days} \\ 0.9 \cdot \text{actual score} & ;\text{late by } 1 \text{ day} \\ 0.7 \cdot \text{actual score} & ;\text{late by } 2 \text{ days} \\ 0.5 \cdot \text{actual score} & ;\text{late by } 3 \text{ days} \\ 0 & ;\text{late by more than } 3 \text{ days} \end{cases}$$

3. Plagiarism, if detected, no matter how big or small, will lead to a score of ZERO for all the team member; no exceptions! It is ok to take help from AI tools or talk to other teams. But, you cannot copy from another team or any AI tools.
   a. It is down right stupid to copy-paste from AI tools and think that is acceptable.
   b. Also, it is a bad idea to keep open another teams work in front of you while you do the assignment. While you may think you are just taking help, your reports and your codes will get heavily influenced by the other team to the point that it will qualify as being plagiarized.

4. This is a team project. You have to do the project with the team that you selected in the excel sheet that was shared with you.
   a. **Only one submission per team.**
   b. Submission must be made using the google drive link that was shared with you.
   c. **Your submission in the google drive link must contain ONLY FOUR files:**
      i. **report.pdf**
      ii. **policy_evaluation.py**
      iii. **value_iteration.py**
      iv. **policy_iteration.py**

Read the deliverables carefully to understand what needs to be submitted in the report and in the Python codes. **DON'T submit any other files.** You MUST NOT zip/compress these four files. **You MUST NOT put these four files in a subfolder inside the google drive link.** Directly drop these four files in the google drive folder corresponding to your team.

# Server Scaling in Data Centers with Deferrable Computational Demand

## Section 1: Sharing workload

Unlike programming assignment 1, the tasks in this assignment are **not decoupled**. This leads to some challenges to divide task among yourself. The following are my two cents about sharing workload:

1. Every team member should read section 3 on their own. Reading section 3 is not a team work.
2. In section 4, there are three tasks with increasing level to difficulty. Hence, it is NOT advisable to do one before doing the other and also NOT advisable to divide these three tasks among yourself. All the team members should sit together and do section 4 together. By doing so, you are also **preparing for minor 2 and the end-sem because there is definitely going to one problem on MDP formulation**.
3. Section 5 is completely dependent on section 3. So, get done with section 3 first. In section 5, there are three tasks that are decoupled. You can easily divide these tasks among yourself and do it independently.
4. You can all share **task 1** of section 6. Each one of you can choose to do one run and then collate the results. **Tasks 2 to 5** of section 6 are decoupled. You can easily divide these four tasks among yourself and do it independently.

Remember, the above points are just my suggestion. You may choose to use your strategy.

## Section 2: Useful resources

The following are some useful resources for this assignment:

1. Check lecture 10 and lectures 14 to 16 for MDP-related concepts. These lecture notes may not be complete in which case you can just refer the book.
2. You can check lectures 11 to 13 and 17 to 18 to find examples of MDP formulation.
3. Check lectures 17 and 18 for examples on writing the Bellman optimality equation.
4. In lectures 17 and 18 folder, I have included Python code for policy iteration for one of the problems that we solved in class.

## Section 3: Introduction

A data center is a physical facility that provides access to computing resources. Among other things, data centers have servers (basically a computer). Users can remotely connect with these servers and use it to do their computational tasks. Whenever we are using cloud computing websites like Amazon AWS, Google Colab, ChatGPT etc., some server housed in some data center around the world is being used.

It takes a lot of energy to run a server. Hence, when a server is not in use, it **seems** better that we immediately switch it off to save some energy cost. But such an action can come at a cost. What if we need to use a server again (maybe immediately) after switching it off? Well, we have to turn on a server. But turning on a server incurs a switching cost. This cost reflects the excess energy used to boot a sever, wear-and-tear costs, the delay in migrating connections/data etc. So obviously there is a **trade-off** between minimizing energy cost and switching cost.

Off course, this tradeoff can't be completely avoided but it can be mitigated when the computational demands are deferrable. **Deferrable demands are those that we don't need to do immediately when it arrives; we can do it within a time period of its arrival**. Deferrable demands helps in mitigating the tradeoff mentioned in the previous paragraph because even if we switch off a server that is not being used and a lot of computational demand arrives immediately after that, we don't have to worry a lot about switching the server on again and incurring a switching cost because we can wait for some time (because the demands are deferrable) before switching on that server.

In this assignment, **your task is to use Markov decision process (MDP) to design optimal policy for managing servers in order to minimize all the involved costs**.

## Section 4: Theory Component (40 marks)

The first component of the assignment is to formulate the problem as an MDP and get the Bellman optimality equation for this problem. However, you will do this in steps; in **increasing order of difficulty**.

Deliverables: For problems 1 to 3 described below, document the following in the report:

1. Formulate the problem as an MDP (states and state space, actions and action space, and average reward for each state-action pair).
2. Write the Bellman optimality equation.

**Problem 1** (10 marks): For this problem, we don't consider that the demands as deferrable. Let $d_t$ denote the number of servers required to process the computational demand at time slot $t$. Rather than calling $d_t$ as the number of servers required to process the computational demands, we will instead call $d_t$ as the computational demand of time slot $t$ (as in the demand of servers). We have, $d_t \in \{0,1,\cdots,D\}, \forall t$. $d_t$ is an iid process with the probability of $d_t = d$ being $\phi_d$.

In every time slot, the data center has to decide $u_t$, the number of servers the data center switches on/switches off at time $t$. $u_t$ is positive when data center switches on more servers and $u_t$ is negative when the data center switches off existing servers that are on. Due to operational constraints, the number of servers that are on can exceed $\boldsymbol{S_{max}}$ at any time. The following points are crucial:

1. All computational demands in any time slot must be served.
2. If we switch on more servers at time $t$, we get to use these servers (along with the servers that were already on) to serve the computational demand $d_t$ at time $t$ itself; we don't have to wait till the next time slot to use those servers that were immediately switched on. Similarly, if we switch off servers at time $t$, we can't use those servers to serve the computational demand $d_t$ at time $t$.

The objective is to minimize the $\beta$-discounted cost where the cost in a time slot is the sum of:

1. **Switching cost:** If the data center switches on $x$ servers in a time slot, it will cost $\theta_1 x$, where $\theta_1 > 0$ is a system parameter. There is no switching cost associated with switching off servers.
2. **Energy cost:** Energy cost has two components. The first is the costs to keep a server on. The second is the costs to use a server. So, if $x$ servers are on out of which only $y \leq x$ are used to server computational demand then the energy cost in that time slot is $\theta_2 x + \theta_3 y^2$, where $\theta_2, \theta_3 > 0$ are system parameters. It is to be noted that the servers that were switched on/off in a time slot has to be accounted for while calculating the energy cost of that time slot.

Problem 2 (15 marks): This problem is same as Problem 1 but here we consider that the computational demands are deferrable. We can defer serving a computational demand by at most $\tau \geq 0$ time slots. To elaborate, the "effective" computational demand at time $t$ consist of the current demand $d_t$ and also $l_{t-1}, l_{t-2}, \cdots, l_{t-\tau}$ where $l_{t-k}$ is the number of computational demands that originated in time $t-k$ that has not been served till time $t$. I have made a small YouTube video (click here) to help you understand the setup. You can choose to ignores this video if this writeup provides good enough explanation.

The data center has to decide $u_t$ as it did in problem 1. Along with that the data center also has to decide how much of demand $d_t$ it wants to serve, how much of demand $l_{t-1}$ it wants to serve, how much of demand $l_{t-2}$ it wants to serve and so on till how much of demand $l_{t-\tau}$ it wants to serve. While making these decisions the data center must ensure that while it can defer serving a demand by at most $\boldsymbol{\tau}$ time slots, all the demands must finally be served (just like in problem 1).

As in problem 1, the objective is to minimize the $\beta$-discounted cost where the cost in a time slot is the sum of switching cost, energy cost, and penalty cost. Switching cost and energy cost is same as problem 1. Penalty cost at time $t$ is

$$\sum_{k=1}^{\tau} \alpha_k l_{t-k}$$

where $\alpha_k > 0, \forall k$ and $\boldsymbol{\alpha_k}$ is monotonic increasing in $\boldsymbol{k}$. Penalty cost ensures that even though deferring is allowed, it is avoided as much as possible. $\alpha_k$ being monotonic increasing in $k$ ensures that the more we defer the more we are penalized.

Problem 3 (15 marks): You must have realized by now that there are a lot of actions in problem 2 and consequently the action space is huge. So, while we can definitely write Bellman optimality equation for problem 2, solving the Bellman optimality equation and hence finding the optimal policy may be computationally challenging. In this problem, our idea is to address this computational challenge by reducing the number of actions and hence reducing the size of the action space. Everything else remains same as problem 1. The actions for this problem are:

1. $u_t$ as we did in problems 1 and 2.
2. $\delta_t$ which is the number of servers that the data centers want to use to serve the "effective" demand at time $t$. $\delta_t$ can also be interpreted as the net demand served in time slot $t$. Obviously, $\delta_t$ must be less than or equal to the number of servers that are on in time slot $t$ (while accounting for $u_t$). After $\delta_t$ is decided, the following rule is used to allocate $\delta_t$ among $d_t$, $l_{t-1}, l_{t-2}, \cdots, l_{t-\tau}$:

- Stage 1: The 1st priority goes to $l_{t-\tau}$. All of $\delta_t$ must be used to serve $l_{t-\tau}$. Only if $\delta_t > l_{t-\tau}$, then the remaining $\delta_t - l_{t-\tau}$ server can be used for the next stage. Let $\delta_t^1 = \delta_t - l_{t-\tau}$. We go to the next stage only if $\delta_t^1 > 0$.
- Stage 2: The 2nd priority goes to $l_{t-\tau-1}$. All of $\delta_t^1$ must be used to serve $l_{t-\tau-1}$. Let $\delta_t^2 = \delta_t^1 - l_{t-\tau-1}$. We go to the next stage only if $\delta_t^2 > 0$.
  - 
  - 
  - 
- Stage $\tau + 1$: The last priority goes to $d_t$. The remaining $\delta_t^\tau$ is used to serve $d_t$.

The intuition behind the above rule is to prioritize serving those demands that has been deferred more than the other and, in the process, incur less penalty cost. The difference between problems 2 and 3 is that in problem 2, we are explicitly deciding how much of each $d_t, l_{t-1}, l_{t-2}, \cdots, l_{t-\tau}$ must be served while in problem 3, we are just deciding the net demand $\delta_t$ to serve in time slot $t$ and then using a hardcoded rule to decide how $\delta_t$ can be allocated among $d_t, l_{t-1}, l_{t-2}, \cdots, l_{t-\tau}$. Since we are using a hardcoded rule, the policy obtained in problem 3 may be sub-optimal[1] compared to problem 2 but definitely more computationally efficient than problem 2. NOTE: While solving problem 3, choose action space wisely. Multiple action spaces may be correct but some may be computationally less demanding than the other.

## Section 5: Programming Component (40 marks)

The system parameters are $\phi_d$, $D$, $S_{max}$, $\tau$, $\theta_1$, $\theta_2$, $\theta_3$, and $\alpha_k$. Some of these parameters capture the probability distributions governing state transition and reward. You will assume that these parameters are known for all the deliverables below.

Deliverables:

1. (10 marks) Implement iterative policy evaluation to find the value function of a greedy policy for problem 1. Greedy policy is the one that minimizes the immediate average reward. You MUST implement this code in the Python script **policy_evaluation.py** that is provided to you. The function MUST return the value function for the greedy policy. The only thing that you should include in **report.pdf** for this part is an equation that describes the greedy policy.

2. (15 marks) Implement value iteration to solve the Bellman optimality equation for problem 3. You MUST implement this code in the Python script **value_iteration.py** that is provided to you. The function MUST return the optimal value function and the optimal policy. You MUST NOT include anything in **report.pdf** for this part. It is crucial that your code can run for any value of $\boldsymbol{\tau}$, even $\boldsymbol{\tau = 0}$ ($\tau = 0$ means non-deferrable demands like in problem 1).

[1] I don't think that the hardcoded rule is sub-optimal. You will get 5 buffer marks (out of 100 marks for this entire course) if you manage to RIGOROUSLY prove it. I have not taught you these kind of proofs, you have to find it out yourself. Alternately, you can also disprove my claim and get these 5 buffer marks.

3. **(15 marks)** Implement policy iteration to solve the Bellman optimality equation for problem 3. You MUST implement this code in the Python script **policy_iteration.py** that is provided to you. The function MUST return the optimal value function and the optimal policy. You MUST NOT include anything in **report.pdf** for this part. It is crucial that your code can run for any value of $\boldsymbol{\tau}$, even $\boldsymbol{\tau = 0}$ ($\tau = 0$ means non-deferrable demands like in problem 1).

Read these points before attempting the above deliverables:

1. Default system parameters: The following are the default values of the involved parameters: $D = 5$, $\tau = 4$, $S_{max} = 15$, $\theta_1 = 10$, $\theta_2 = 1$, $\theta_3 = 0.2$, $\alpha_k = 0.3k\ \forall k$. The default value of discount factor $\beta = 0.95$. In the pseudocode of iterative policy evaluation, value iteration, and policy iteration, there was a convergence threshold $\theta$ in the lecture slides (this $\theta$ is different from $\theta_1$, $\theta_2$, and $\theta_3$). Set this convergence threshold as $2$. An additional convergence parameter is $K_{min}$, the minimum number of iterations for iterative policy evaluation, value iteration, and policy evaluation of policy iteration. Set $K_{min} = 10$. These default value MUST be used for testing your code and doing analysis in the next section unless mentioned otherwise. NOTE: Your code must run for any values and not just the default ones.

2. You are provided a Python script **Assignment2Tools.py** that contains a function **prob_vector_generator()**. This function can be used to generate a probability distribution, $\phi_d$, that has a pre-specified mean and standard deviation. The instruction to use this function is there in **policy_evaluation.py**, **value_iteration.py**, and **policy_iteration.py**.

3. For a given set of system parameters, the optimal value function obtained using value and policy iteration should almost be the same. Otherwise, there is something wrong with your code.

4. While coding the Q-function, you may want to use the broadcasting operation of Numpy to speed your computation. The following are the run times for default parameters in my office laptop:
   - **policy_evaluation.py** produces results in like 5 seconds.
   - The run time of **value_iteration.py** and **policy_iteration.py** is less than $32$ minutes. One of them runs around three times faster than the other (not going to tell which one).

5. Must account for action space: You must take into consideration that **different states may have different action space**. This means a few things. *First,* while implementing value/policy iteration, the maxima/minima should be over the action space corresponding to the state. *Second,* for policy iteration, the policy can be initialized to any arbitrary value in the action space corresponding to the state.

## Section 6: Analysis Component (20 marks)

In this section, you have to use the codes that you wrote in section 4 to do some analysis. Answer the following questions. For all these questions, your answer MUST be in **report.pdf** ONLY. Any code that you write to answer these questions MUST NOT be there in **policy_evaluation.py**, **value_iteration.py**, **policy_iteration.py**, or **report.pdf** that you submit.

1. (5 marks) Compute average computation time of value iteration and policy iteration over 5 runs. Don't change the default parameters across these runs. For each run, you must use different values of $\phi_d$. Which approach is faster?
    - You may want to use the faster code for the remaining section.

2. (5 marks) Do necessary analysis to compare the performance of the greedy policy and the optimal policy for non-deferrable demands ($\tau = 0$). To do so you have to first decide how will you even compare the performance of two policies. Your final deliverables for this task is to generate two plots that compare the performance of greedy policy and the optimal policy as switching cost $\theta_1$ and standard deviation of $\phi_d$ varies. Give a logical explanation of the trends that you observe in these plots.

3. (5 marks) It seems intuitive that as switching cost $\theta_1$ increases, the optimal policy is less likely to switch off servers that are not being used. This should be true for both deferrable and non-deferable demands. But to reduce the computational challenge, you can just focus on non-deferrable demands here. Document necessary analysis (maybe plots) that either validates this intuition or negate it. Use your imagination to decide what analysis you want to do.

4. (5 marks) It seems intuitive that as the demands become more deferrable (as $\tau$ increases), the performance of the optimal policy is likely to be better. The reason for this is mentioned in the last paragraph of section 2. Document necessary analysis (maybe plots or tables) that either validates this intuition or negate it. Use your imagination to decide what analysis you want to do. Keep your value of $\tau$ between $0$ to $4$ (computation increases significantly as $\tau$ increases).

5. (5 marks) It seems intuitive that the need for deferrable loads increases as the computational demand becomes more fluctuating[2]. Document necessary analysis (maybe plots or tables) that either validates this intuition or negate it. Use your imagination to decide what analysis you want to do. Keep your value of $\tau$ between $0$ to $4$ (computation increases significantly as $\tau$ increases).

Remember: You will be graded based on the quality of your analysis in section 5.

[2] This is because higher fluctuation implies a higher probability of the demand either increasing or decreasing. When the demand decreases, we would want to switch off servers to save energy cost without the fear of incurring switching cost if the demand increases again immediately. This fear is mitigated as the demand becomes more deferrable. So it is obvious that as the fluctuation becomes more, the need of deferrable demand is more.