Assignment 1 — Markov Decision Theory

Please provide brief but careful motivations of your answers. Include all source code that you use for this assignment in your report and consider using the `listings` package for proper code formatting.

Problem 1 – Unreliable Machine

Suppose we have a machine for making computer chips. When the machine is in *excellent* condition, it produces two chips per day, but a failure might occur with probability 0.1. When this happens, all chips produced on that day are damaged. After failure, an *emergency repair* is necessary during the night to get the machine running again the next day, but such fix leaves the machine in *good* condition. Whenever the machine is not in excellent condition, it can only produce one chip per day. A machine in good condition fails with probability 0.3, which results in *poor* condition the next day. Machines in poor condition are very unreliable, as they break down with probability 0.6, but their condition cannot degrade further.

It is possible to put the machine back into excellent condition by executing a *full maintenance* procedure, which takes two full days. Your task is to determine when to deploy this procedure in order to maximize the total output of five days.

(a) Formalize the above problem as a Markov decision problem. Make sure to specify

- the state space $\mathcal{I}$,
- the action space $\mathcal{A}$,
- direct rewards $r^a(\cdot)$,
- final rewards $q(\cdot)$,
- transition probabilities $p^a(\cdot,\cdot)$.

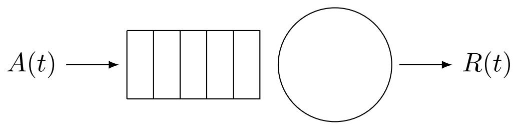
(b) Suppose our machine is currently in excellent condition. Formulate the optimal maintenance strategy for five days and calculate the corresponding maximum expected number of undamaged chips that are produced under this strategy.

Problem 2 – Capacity Scaling

Consider a queue that holds jobs to be processed on a machine. We assume that the system operates in discrete time slots. Let $A(t)$ denote the number of jobs arriving to the queue during time slot $t = 1, 2, \ldots$. Arrivals are distributed identically and independently in each time slot as

$$A(t) = \begin{cases} 0 & \text{w.p. } 0.5, \\ 2 & \text{w.p. } 0.4, \\ 5 & \text{w.p. } 0.1. \end{cases}$$

There is room for up to $Q_{\max} = 5$ jobs in the queue. When jobs arrive to a full queue, they are simply discarded. At the start of each time slot, we need to choose the *service capacity* $R(t) \in \{0, 1, 2, 3\}$ of the machine, which is the maximum number of jobs that can be handled during the time slot.



More precisely, the evolution of the number of jobs in the queue is governed by

$$Q(t+1) = \min\{\ \max\{Q(t) - R(t), 0\} + A(t),\ Q_{\max}\ \}.$$

There are two types of cost associated with the system. Maintaining a queue of length i during a time slot induces a *holding cost* of

$$C_h(i) = \begin{cases} i & \text{for } i \in \{0,1,2\}, \\ 5 & \text{for } i = 3, \\ 7 & \text{for } i = 4, \\ 10 & \text{for } i = 5. \end{cases}$$

Furthermore, the *service cost* associated with choosing service capacity $R(t) = r$ is given by

$$C_s(r) = \begin{cases} 0 & \text{for } r = 0, \\ 0.5 & \text{for } r = 1, \\ 2.5 & \text{for } r = 2, \\ 7.0 & \text{for } r = 3, \end{cases}$$

regardless of whether all this capacity was actually used. Our task is to determine a policy that minimizes the long-term average cost of operating this system.

(a) Formalize the above queueing system as a Markov decision process.

(b) Consider the decision rule $\mathbf{f_0}$ that picks $R(t) = 1$ in each time slot, no matter the state of the system. Compute the long-term average cost for this policy and provide relative rewards.

(c) Perform one step of policy improvement to obtain $\mathbf{f_1}$.

(d) Use successive approximation to obtain an optimal policy and give the corresponding minimum long-term average operational cost.

(e) Suppose that the condition of the system degrades heavily when the queue is at maximum capacity. Suppose we want to avoid having a full queue for more than 10% of the time. Explain how you could formalize this requirement and indicate how an optimal policy under this constraint can be found. You do not have to compute the solution.