Reinforcement Learning

July 22, 2020

1 Simplest Model: Processes 1 Scheduling

As an experiment, we will first try to implement an environment to schedule patients for processes one, ignoring the second processes. The objective in this simple case is first, to minimize the number of overlaps, that is booking a slot that is already booked, and second, to minimize the number of patients who are booked on two different weeks

1.1 The Environment Specification

- State Variables
    1. **Schedule**: Consists of a $Days \times SLOTS$ matrix of of integers. An entry is 0 when there is no patient is scheduled in that slot, and equal to the patient number scheduled there otherwise
    2. **current day**: The day we are currently on. This variable is used to either stop the processes, and to also prevent scheduling on days before the current day
    3. **current patient number**: The number of patients in the system up to this point.
    4. **remaining slots**: Number of slots available for scheduling
- Exogenous Variables
    1. **Patient**: This is the number of collection days required for the current patient **Remark**: We can later change each patient from being a number to a random number generator, so that we can schedule her without knowing how many collection days are required
    2. **End day**: Binary variable ,0 or 1, telling us if we should move to next day
- Decision Variables
    1. **Day**: The day to schedule the patient on
    2. **Week**: The week to schedule the patient on

- Transition Function

    1. $Schedule_{t+1}$ is either equal to $Schedule_t$ when no scheduling has occurred (a full day have been chosen) or there are avialble slots and they are filled
    2. $current\_day_{t+1} = current\_day_t + end\_day_t$
    3. **current patient number** is increased by 1 when a scheduling happens, is the same otherwise
    4. **remaining slots** is decreased by the number of required collection days when scheduling happens, and if a new day occurs, subtract number of slots available. Otherwise, the same
    5. **patient** is a randomly generated patient (with appropriate distribution), when distribution occurs, otherwise the same.
    6. **End of day** is randomly generated when scheduling occurs, otherwise

- Objective (For this simple model)

1.2 Environment Implementation

First, we have a few **parameters** outside the environment that let us control it:

1. **DAYS** The number of days the model spans (Not countin weekends)
2. **SLOTS**: The number of available slots per day
3. **COLLECTIONS**: The maximum number of collection days a patient can require
4. **NEXT_DAY_P**: The probability of transitioning to the next day after an update of the state of the system . Notice that, the smaller this probability is, the more patients we expect to see coming into the system per day

We are using the Open AI GYM to implement our environment, the internal states of the environment that are used for observations are the following

1. **schedule**: This is a Numpy $DAYS \times SLOTS$ matrix of integers, corresponding the schedule variable
2. **current_day**: This is a an integer that records the day we are on.
3. **current_patient_number**: An integer that record the number of the current patient in the system
4. **remaining_slots**: An integer that records the number of open slots in the schedule
5. **patient**: An integer that records the the number of collection days required for the current patient

6. **end_day**: An integer, with possible values 0 or 1 that indicates if we should move to the next day or not

in addition to these state, we also keep track of variables that are not used for observations, but which can be used as an evaluation metric for a policy choice:

1. **overlap_count**: An integer that records how many times the system picked an already filled slot in the schedule

2. **total_reward**: a number that records the total rewards accumulated by the system (given a reward system)

3. **two_weeks_violations**: An integer that records the number of patients that are scheduled for collection on two different weeks (there is a weekend in the middle).

Open AI's GYM package requires to define the following to implement a GYM environment:

- **Actions and Observation Spaces**: We have to specifty the type and shape of each observation and action in advance, becasue we want to interface with Open AI's Stable Baseline package later on, both observations and actions should consist of a one dimensional Numpy Array:

    1. **observation_space**: we will represent each observation by a $1 + 1 + 1 + 1 + 1 + DAY \times SLOTS$ Numpy array, which is divided as follows:
        - **index 0**: Contains current_day
        - **index 1** : Contains current_patient_number
        - **index 2**: Contains remaining_slots
        - **index 3**: Contains patient
        - **index 4**: Contains end_of_day
        - **index 5 - end**: Contains flattened schedule
    2. **action_Space**: Consists of one number, which is the day we want to schedule the patient on.

- **The Basic Functions** : We will need to implement the following four basic functions

    1. **__init__**: Used to initialize an object of the class, inside it we must define our **observation_space** and **action_space**.
    2. **step**: This function takes an action as an input, and return the next observations , reward of the systems and a bunary variable that indicate if the processes is to be stopped or not
    3. **reset**: This function takes no arguments, and reset the environment to its initial state

4. **render**: This function takes no arguments and is used to render a graphical representation of the environment

We also implement the following helper functions:

- **Formating Logic**: We use these functions to format the observations of the system so that they can be used
    1. **makeObs**: This function takes all the internal states of the system and put them in a forma suitable for Open AI's Stable Baseline package
    2. **done**: This function returns a boolean indicating if we have reached the end of our processes
- **Transition Logic**: These functions are used in the function **step** and they implement the updating rules of state given an action
    1. **Transition**: This function takes an action. It performs the updating of the state. Return a Boolean indicating if the schedule was updated or not (in case the agent chooses an already filled slot)
    2. **updateSchedule**: This function takes an action and then updates the schedule. It return a boolean indicating if the schedule was updated or not
    3. **updateDay**: This function takes a boolean and updates the curent day depending on the boolean
    4. **updatePatientNumber**: This function takes a boolean and updates the curent patient number depending on the boolean
    5. **updateRemainingSlots**: This function takes a boolean and updates the remaining slots depending on the boolean
    6. **updatePatient**: This function takes a boolean and updates the patient depending on the boolean
    7. **updateEndOfDat**: This function takes a boolean and updates the end of day variable depending on the boolean
- **Reward Logic**: These functions implement our reward system, which is used for Reinforcement Learning **Remark**: This is the most incomplete part
    1. **Reward**: This function takes an action and a Bolean, and return the total reward
    2. **slotReward**: This function takes an an action and a Boolean, and return the reward we get from scheduling corrects slots (empty, early, ... etc)
    3. **twoWeeksReward**: This function takes an action and a boolean, and returns a reward we get from abiding or breaking the two weeks rule (Don't schedule same patient over two different weeks)

2 Simple Environment 2: Two Processes

2.1 assumptions

In our first model, we make the following assumptions:

- New patients are queued to be scheduled only at the beginning of every new day.
- The only constraint to the number of patients that can be scheduled in a time slot is the number of available nurses
- When a patient is scheduled, we do not schedule her for a specific doctor.
- The number of available slots per day for processes one = number of available nurses
- number of available slots for processes two = number of available nurses * hours of work.

2.2 Environment specification

- State Variables
    1. **Schedule 1**: Consists of a $Days \times SLOTS1$ matrix of of integers. An entry is 0 when there is no patient is scheduled in that slot, and equal to the patient number scheduled there otherwise
    2. **Schedule 2**: Consists of a $Days \times SLOTS2$ matrix of of integers. An entry is 0 when there is no patient is scheduled in that slot, and equal to the patient number scheduled there otherwise
    3. **current day**: The day we are currently on. This variable is used to either stop the processes, and to also prevent scheduling on days before the current day
    4. **current patient number**: The number of patients in the system up to this point.
    5. **remaining slots1**: Number of slots available for scheduling for processes 1
    6. **remaining slots1**: Number of slots available for scheduling for processes 2
- Exogenous Variables
    1. **Patient**: This is the number of collection days required for the current patient **Remark**: We can later change each patient from being a number to a random number generator, so that we can schedule her without knowing how many collection days are required
    2. **End day**: Binary variable ,0 or 1, telling us if we should move to next day

- Given an action and a state, we update the state as follows:
  1. If the slots picked are available, fill those slots with the patients info
  2. calculate the reward (See below) for this pair of action state
  3. check if the patients queue is empty, if it is, update time(day $\rightarrow$ day + 1, if day == 5, day =1 and week $\rightarrow$ week + 1).
  4. Stop the scheduling when either:
     - We reach the last day of the last week
     - one of the schedules is full
- The reward system consists of:
  1. Reward for picking empty slot
  2. More reward is giving the earlier the empty slot is
  3. Penalty for picking a slot that's already full
  4. Penalty for scheduling a patient for collection on two different weeks
  5. Penalty for scheduling a patient for processes 1 and 2 in two different weeks (? should we consider the situation that we have to arrange them in two weeks? for example, it takes 5 days in process 1)
  6. Reward or Penalty depending on how big the ratio $R = \frac{Q_1}{D_2 - D_1}$, the closer to one the better
  7. ? penalty for $D1 - D2 <$ collection numbers in process 1

2.3 Implementation of the Environment as a GYM Environment