KU LEUVEN kulak

Bachelor in Mathematics
Algorithms and Data Structures
Patrick De Causmaecker, San Pham

MDP for Hospital Admissions implementation

Academic year 2019-2020

Nika Pountouchachvili

Checklist for finished* products

finished*: Either finished with me being sure it is a working product, or me needing some guidance in how to fix the code.

List all States	Done (?)
List all Actions for each state	Done (?)
Calculate the transition probabilities	Done (even for arbitrary size!)
Calculate the costs	Done (?)
Implement the VIA	Not Done

State Space and Action Space

I have fixed the state space by working solely with the resource utilization cap. If there is a state that overutilizes some resources, we cannot work with it, and it is omitted from the state space. Same thing works for the action space: taking each state from the state space I evaluate the possible distributions of the action onto the state to get the next state. If the next state is in the state space, I keep the action, if it is not I will have to omit the action for that specific state.

However, I do have some issues with this method. It seems like the right method to use, but it has nasty consequences (for thecost function in particular, but I will handle that in the section of the cost function):

- One possible problem is that we have too few states (or so it seems)
- My solution: I added a sort of "acceptability factor" to the maximum usage. E.g. say the maximum usage for a resource L_1 is 5 units and $E_1 = 2$ with average usage of L_1 by E_1 is 2.6 units. Then $2.6 * 2 = 5.2$, and that is greater than 5. What I did to fix it is setting a new "more flexible" max: $1.1 * 5 = 5.5$ so that the above example is an acceptable state. However, I do not know if this is the right way of handling it. I can tinker around with it by changing the global variable "bandwidth" in my code. Right now it is set to 1.

Probability Calculation

Although the state space and action space were not optimally created, the probability of going from one state to another is. My approach was not entirely the same as the sum in the paper, as it was terribly inefficient to implement. Instead I came up with a method to recreate every single possibility of going from one state to another, which is a guaranteed foolproof method. As an example you can look at figure 1. I started out with partitioning the integer n in a list of d elements. E.g.

$$n = 5, d = 2 : [5, 0], [4, 1], [3, 2], [2, 3], [1, 4], [0, 5].$$

After having all these partitions I added a 0 for the discharge pattern:

$$[5, 0, 0], [4, 1, 0], [3, 2, 0], [2, 3, 0], [1, 4, 0], [0, 5, 0].$$

Then I kept only the ones that, when subtracting from state x_{t+1} I do not have any negative components (I will denote the action distribution that we subtract as a_t). This lets us know that, whatever the transitioning is, we can only transition to the states that the given action distributions gives us. One of these states you can find in figure 1 this is $(0, 1, 2)$ for example. Here we subtracted $a_t = [2, 1, 0]$ from

$x_{t+1} = [2, 2, 2]$, which indeed yields $[0, 1, 2]$ (denote x'_t). After doing that we need to find all different transitions from state $x_t = [1, 2, 5]$ to state $x'_t = [0, 1, 2]$. Luckily we do not need to worry about E_3, as it is a discharge pattern, so x_t might as well be $[1, 2, 0]$. This is not hard either. We will go by this one treatment pattern at a time: E_1 in x_t is 1, while it is 0 in x'_t. This means that all patients need to transition elsewhere, which gives us the possible transitions: $[0, 3, 0]$ or $[0, 2, 1]$. Since the second treatment pattern becomes a one in x'_t we can only transition every patient we initially had there from $[0, 3, 0] \rightarrow [0, 1, 2]$, or in the second case: $[0, 2, 1] \rightarrow [0, 1, 2]$. And then we need to multiply and count up the probabilities and we are done. Obviously this example is oversimplified. My code however look at every possibility through cartesian products and throws away impossible transitions. This does mean that the complexity becomes much greater in very large cases, but it is terribly more simple to implement than the paper's sum.

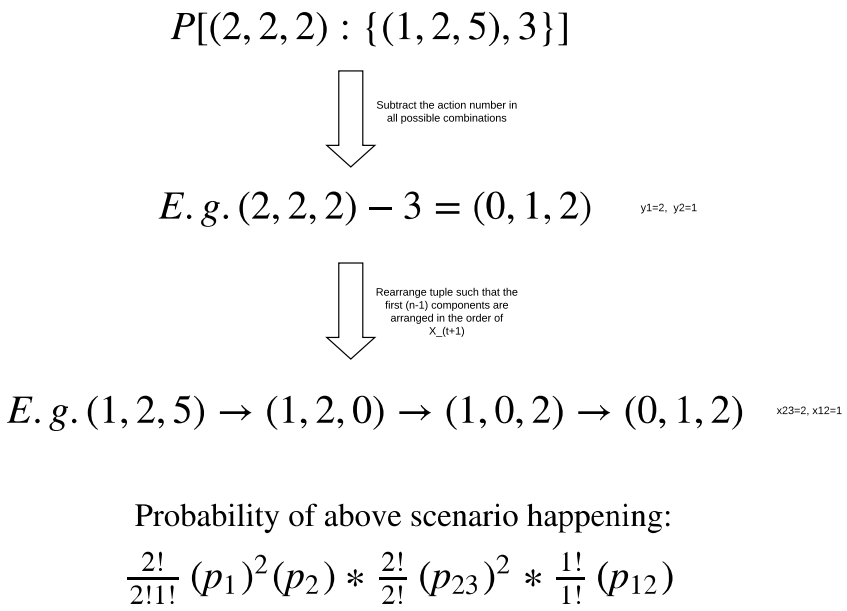

Figure 1: One possible way of reaching the desired state

1 Cost Function

For the cost function I based myself on the state and action space of the model. I have already said that the spaces look too limited to actually work. The cost function kind of confirms that hypothesis. I do find that if I maximize the cost function by a lot I get that the over cost does not affect my total cost by a lot... I think that probably has to do with some faulty code. I cannot work on the code however, if I do not know how to fix my state and action space.

My code works like this: given a state and an action, I calculate all states x_{t+1} that I can make with the action. I do not allow any transitions to happen, as I think that will make the code a lot more difficult to write, but if it is supposed to be written like that I can work on it right away. The actual sum I wrote with the formula.

State and action space

```
[0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0]
[0, 0, 0, 1, 0, 0]
[0, 0, 0, 1, 1, 0]
[0, 1, 0, 0, 0, 0]
[0, 1, 0, 1, 0, 0]
[1, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 1, 0]
[1, 1, 0, 0, 0, 0]
```

```
[[0, 0, 0, 0, 0, 0], ((0, 1), (0, 2), (1, 0), (1, 1), (2, 0), (0, 0))]
[[0, 0, 0, 0, 1, 0], ((0, 1), (1, 0), (0, 0))]
[[0, 0, 0, 1, 0, 0], ((0, 1), (1, 0), (0, 0))]
[[0, 0, 0, 1, 1, 0], ((0, 0),)]
[[0, 1, 0, 0, 0, 0], ((0, 1), (1, 0), (0, 0))]
[[0, 1, 0, 1, 0, 0], ((0, 0),)]
[[1, 0, 0, 0, 0, 0], ((0, 1), (1, 0), (0, 0))]
[[1, 0, 0, 0, 1, 0], ((0, 0),)]
[[1, 1, 0, 0, 0, 0], ((0, 0),)]
```