Modeling a Nursing Home as a POMDP to Manage Covid-19

Leo Schirokauer

January 1, 2021

1 Problem Specification

In this model we formally model the whole nursing home as a POMDP. In order to accommodate the complexity of this model, a generative definition for the POMDP, as well as an online solver, were used. A generative model requires defining a gen(s, a) function that provides $(s, a) \mapsto (s', r, o)$ where s is the initial state, a is the action taken, s' is the next state, and r and o are the observation and reward associated with the transition respectively. We also define an observation distribution O, which returns $O(e, s) = \mathbb{P}(e|s)$ for evidence e and state s. To formally model the nursing home as a POMDP we define a state s as:

$$s = [S, I, M, D]$$

where S is the number of susceptible people, I is the number of infected people, M is the number of immune people (we assume a very low rate of reinfection), and D is the number of people with complications requiring removal to a hospital or long term isolation, and $S + I + M + D = N$ for fixed population size N. We define actions as:

$$a \in [1, 2]$$

where 1 means "wait" or "do-nothing", and 2 means entering lockdown. Each time interval is meant to represent around one day. We define a reward function $R(s, a)$ given by:

$$R(s, 1) = c \cdot (s[1] + s[3]) + k \cdot s[2] + d \cdot s[4]$$

for $a = 1$

$$R(s, 2) = -t$$

for $a = 2$, where c is a small positive reward for healthy (susceptible/immune) people, k is a small negative reward for infected people (reflecting the decreased quality of life of being sick), and d is a large negative reward representing the cost and quality of life decrease of more serious complications. t is a fixed cost for going into lockdown (residents are very unhappy in lockdown).

To complete the POMDP specification, we have observations which are the number of positive tests in a random sample of size m taken at each time point. We also define a transition generating function which generates $T(s, a) = s'$, which is essentially a simulation that generates the next state from a state-action pair. The details of the observation and transition generating function will be described in the Approach section.

2 Approach

To implement the model we used the generative interface in juliaPOMDP and the BasicPOMCP solver with a Particle Filter for belief state updates [1][2]. The reward function, states, and actions are described in the previous section. To implement the observations, we implemented a custom distribution $O(s)$ parametrized

in terms of a state s, which implements a PMF function returning $\mathbb{P}(P = z|s)$ for observing z positive cases given state s, and a random sample for generating observations. The PMF of O is given by:

$$O(z, s) = \mathbb{P}(P = z|s) = \sum_{i=0}^{\min(s[2],m)} f_1(i) \sum_{j=0}^{z} f_2(j) f_3(z - j)$$

where f_1 is a hypergeometric PMF with parameters $s[2]$ positive example, $s[1] + s[3]$ negative examples, and sample size m, and f_2 and f_3 are binomial PMF's with parameters bin(i,x) and bin($m - i$,$1 - y$) respectively, for x the sensitivity and y the specificity. Basically we condition on how many infected people we get in the sample, so conditionally the number of positive tests is the sum of two binomials. To generate observations we also implement a simple random sampler that randomly selects m people from states $s[1]$, $s[2]$, and $s[3]$ and "tests" each one using the sensitivity/specificity parameters. To implement the transition generating function we run a simple simulation at each step (shown below) where c is the average number

```
procedure SIMULATE(s)
    A ← zeros(s[1])
    for i ∈ [1, s[2]] do
        for j ∈ [1, c] do
            if U(0,1) < p then
                A(rand(1, len(A)) = 1
            end if
        end for
    end for
end procedure
```

of social contacts and p is the infection probability. Basically we make an array of the uninfected people, and then for each infected person we randomly choose c people for them to contact (random indices), and they infect each person with probability p. p can be estimated from available data for each specific case, and we can model p itself as a random variable $p \sim \text{Uniform}(a, b)$ to account for the fact that transition probabilities may vary based on the nature of the interaction.

The transitions probabilities for people in states $T(s[2])$ (infected) and $T(s[3])$ (immune) are modeled using a simple multinomial with fixed probabilities given by p_1, p_2, p_3, p_4 where p_i gives the probability of transitioning to state $s[i]$ at each time point. $s[4]$ (removed state) is considered terminal. We assume that $s[3]$ (immune) is almost terminal with a very low rate of reinfection. Under lockdown we reduce c to 0 so that there are no social contacts. This is meant to be a flexible framework and all of these parameters/probabilities can be changed depending on the exact situation or specific nursing home that is being modeled.

In order to solve the POMDP we used the Partially Observable Monte-Carlo Planning (POMCP) implementation in juliaPOMDP [1][2]. POMCP is an online POMDP solver that employs a Monte-Carlo tree search from the agent's belief state to estimate the value of taking each action at any time point. To update belief states the built in particle filter (ParticleFilters.jl) was used. The particle filter uses a particle collection to model the belief state distribution and updates particle weights using the observation distribution. There are two important parameters that we specified for the solver which were the UCB exploration constant which we set to 7000 (approximately twice the difference between estimated value of best and worst policy) and used a simple rollout policy where the model always takes action 1, to estimate the value of leaf nodes. The formula for the UCB exploration paramters was recommended by Dr. Sunberg who wrote juliaPOMDP.

In addition to the POMCP solver we also examine the performance of two other strategies. For benchmarking we consider the class of "simple" strategies where we do not take into account the belief state but

instead take the lockdown option at fixed time intervals. The idea is that without using belief states the best we can do is lockdown every h days for some fixed number h. We also consider a "cutoff-heuristic" strategy where we lockdown the nursing home any time the number of cases in the belief state exceeds a fixed cutoff. To evaluate each strategy we considers total discounted rewards over a 30 time step period (meant to be around 1 month).

3 Experiments and Results

We evaluate each strategy using total discounted rewards over a 30 time step period (around 1 month). The full parameter settings for these results are given at the end of the section. First we consider the class of simple strategies, where each row gives the reward for lockdown every certain number of days (time interval). So for example row 1 gives the reward for lockdown every day (which is just $(30)(-100) = -3000$).

Average Reward for Simple Strategy Policies

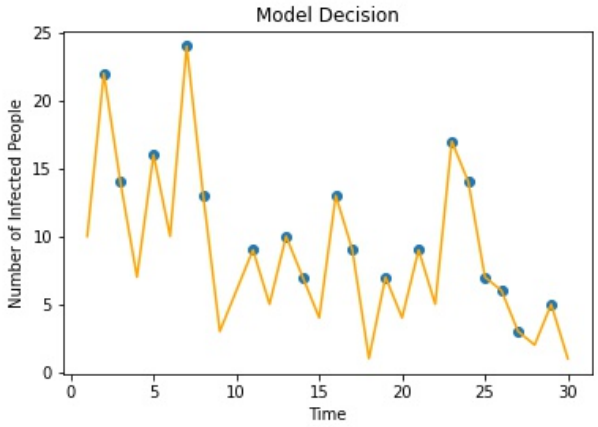
Time Interval	Reward
1	-3000
2	694.5
3	-263
4	-1964
5	-4092
6	-2272

We observe that the best we can do using a simple strategy (one that does not consider the belief state) is lockdown every other day. Any other option performs extremely poorly. These results are mainly for benchmarking, as they represent a reasonable solution given that we are not accounting for the belief state. Next we have the result over 10 simulations for the POMCP solver:

Reward for POMCP Solver

1	3510
2	1645
3	2175
4	1355
5	1495
6	2445
7	3530
8	3010
9	4525
10	1400
Average	2509.0

The POMCP solver significantly outperforms any simple strategy. To better understand the POMCP solver's method the graph below gives the number of infected people and the solver's decision at each time point for the first simulation. There is a dot on the line anytime the POMCP solver takes the lockdown action.



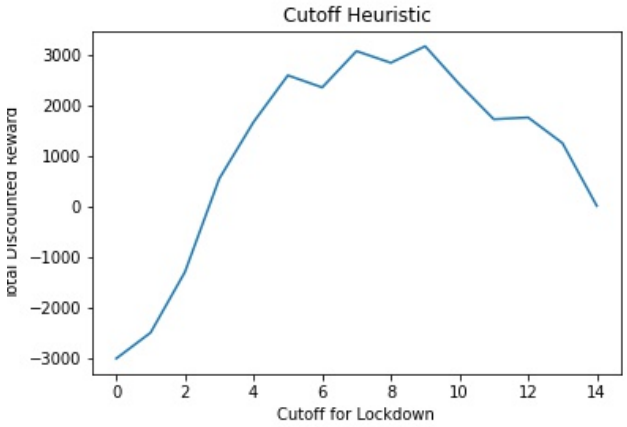
It's clear that anytime cases spike the model chooses lockdown, however there is not an obvious cutoff above which the solver always does lockdown. Whenever cases are very high (above around 10) the solver always does lockdown, however the cutoff for lockdown seems to decrease as time progresses; even when overall cases are low it still responds to spikes. I interpreted this to mean that the model is also considering the proportion of infected people to susceptible people. When there are also fewer susceptible people left the solver seems to have a lower cutoff for treating because the ratio of infected to susceptible is higher.

However we also wondered if we could do even better through custom heuristic policies. We designed a heuristic policy where we lockdown the nursing home anytime we believe there are more than some fixed number of cases.

Average Reward for Heuristic Policies

Cutoff	Reward
0	-3000
1	-2490.5
2	-1294
3	546.83
4	1666
5	2587.17
6	2848.5
7	3065.67
8	2834.83
9	3161.83
10	2409.16
11	1720.67
12	1756
13	1250.33
14	11.33

We see that the optimal heuristic-cutoff policy is to lockdown the nursing home whenever there are greater than 9 infected peolpe in the belief. The optimal cutoff-heuristic outperforms POMCP. We also can observe that the graph of reward vs. cutoff has an shape with a clear optimum:



Under all parameter settings I tried, the graph for cutoff vs. reward maintains this shape, suggesting that under most circumstances there does exist an optimal cutoff for going into lockdown. All results given have the following parameters:

$$R(s,2) = -100$$

$$R(s,1) = 10 \cdot (s[1] + s[3]) + -10 \cdot s[2] + -50 \cdot s[4]$$

$$p \sim \text{Uniform}(0.3, 0.6)$$

$$x = y = 0.98$$

$$m = s[1] + s[2] + s[3]$$

$$c = 3$$

$$T(s[2]) \sim \text{multinomial}([0.2, 0.5, 0.25, 0.05])$$

$$T(s[3]) \sim \text{multinomial}([0.0, 0.02, 0.98, 0.0])$$

$$N = 100$$

Initial state: $[90, 10, 0, 0]$

Discount Factor: 0.95

References

[1] Maxim Egorov, Zachary N. Sunberg, Edward Balaban, Tim A. Wheeler, Jayesh K. Gupta, and Mykel J. Kochenderfer. Pomdps.jl: A framework for sequential decision making under uncertainty. *Journal of Machine Learning Research*, 18(26):1–5, 2017.

[2] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23, pages 2164–2172. Curran Associates, Inc., 2010.