Revenue Management Problem Policy Search

Dawson Ren

December 27th, 2022

1 Setup

Given m nurses and n shifts, we have:

- $P \in \mathbb{R}^{m \times n}$, the probability that nurse i will select shift j. Note that the probabilities are all independent.
- $Q \in \mathbb{R}^{m \times n}$, the probability that nurse i will show up to shift j.
- $R \in \mathbb{R}^{m \times n}$, the revenue that is gained when nurse i covers shift j.
- $Y \in \{0,1\}^{m \times n}$, whether or not we will show nurse i shift j.

2 Finding the Optimal Policy

2.1 Speeding up Brute Force Search

We can brute force search for the optimal policy. We first consider possible speedups to this approach.

1. Do not consider infeasible policies. A feasible policy has at least one nurse willing to take any shift, or if $m \geq n$,

$$\forall j, 1 \leq j \leq n, \sum_{i=1}^{m} Y_{ij} \geq 1$$

   We can be more specific. The last nurse should always be available to be scheduled for every shift in an optimal policy, assuming that all revenues are positive. This is because there is no possible downside to scheduling this nurse it doesn't decrease the probability of any future nurses scheduling the shift, since it is the last nurse.

2.2 Revenue Function Decomposition

The expected revenue is:

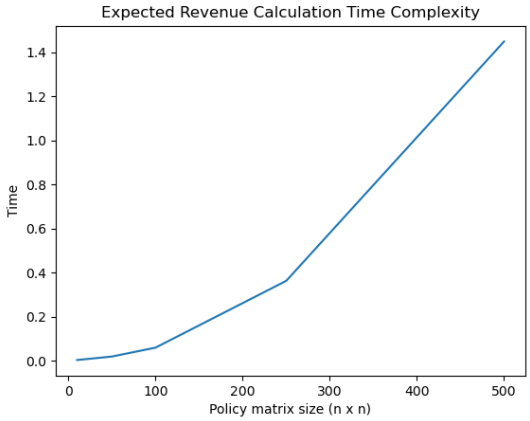
$$\mathbb{E}[R] = \sum_{i=1}^{m} \sum_{j=1}^{n} R_{ij} P_{ij} Q_{ij} Y_{ij} \prod_{k=1}^{i-1} (1 - Y_{kj} P_{kj})$$

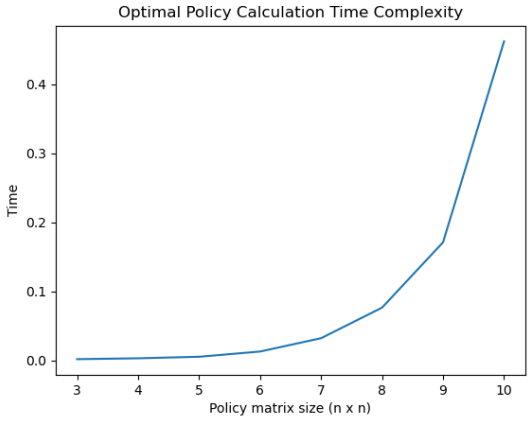
The expected revenue is then in $O(m^2 n)$ time, as calculating the product is in $O(m)$ time and there are $O(mn)$ entries. Note that we can swap the order of the sums and consider the expected revenue across assignments for columns. In the code, this means we calculate the expected revenue of each shift and sum them together for the expected value of the total policy.

$$\mathbb{E}[R] = \sum_{j=1}^{n} \sum_{i=1}^{m} R_{ij} P_{ij} Q_{ij} Y_{ij} \prod_{k=1}^{i-1} (1 - Y_{kj} P_{kj})$$

Using caching, we can amortize the cost of calculating the product factor. That is, a tuple value of (shift, policy assignments to that shift) maps to the expected revenue of that shift. That way, calculating the expected value of many policies may have significant performance gains. Consider the case where we perform a brute force search over all policies, of which there are 2^{mn}. Without caching, we would calculate the expected revenue for all of them, which takes $O(m^2 n)$ time, so the total time complexity is $O(m^2 n 2^{mn})$. With caching, our caching table contains at most $n2^m$ entries. The number of entries in the table grows exponentially less quickly than the total policies we have to search ($n < 2^n$). Therefore, a majority of policies will take $O(1)$ time to calculate the expected revenue, reducing the time complexity to $O(2^{mn})$ if we "warm up" by filling in the cache.

However, we can be even more clever. Why not consider each shift and find the optimal policy for just that shift? The optimal policy is then the concatenation of all those shifts. This reduces our time complexity to $O(n2^m)$. By a simple testing script, caching still results in around a 5x speedup.





3 Results

We first gut-check our asymptotic analysis with a rudimentary timer setup that can be found in tests/time_complexity.py.

This aligns with our predictions. The expected revenue looks like some polynomial on n which is between 2 or 3. The optimal policy calculation is clearly exponential, but the rate of increase is much slower, as solving for a $5x5$ policy with the naive method resulted in a very long (greater than 1 minute) runtime.

In order to analyze the performance of our implementation, we use the Scalene profiler.

See the main.py script for the driver code. We simply create an 12x12 problem for the algorithm to solve. The main qualitative observations are that calculating the availability of all nurses for a given policy is our main bottleneck (78% of time spent). Right now, about half of the time is spent using Python versus native code (courtesy of Numpy), and there is more experimentation that can be done with C/C++ code/Cython to speed up this part of the process.