Building Game Theoretical Software in a Research Environment

With an application to healthcare modelling

James Campbell & Dr Vince(nt) Knight

School of Mathematics

Sage open-source mathematical software system

Sage: "Creating a viable free open source alternative to Magma, Maple, Mathematica and Matlab". Ref? Sage is Open Source, which means that anyone can contribute to it's upkeep and improvement, but only after a stringent review scheme. We settled on three projects to add to Sage, one of which has already been included in a recent release.

Our Contribution

- Normal Form Games;
- Matching Games;
- Cooperative Games.

Matching Games allow us to solve problems where players need to be paired with each other, but they have their own preferences. We normally look for stable matching where no player has any incentive to change their pairing.

$$\begin{array}{ll} c\colon (B,A,C) & C\colon (a,b,c) \\ b\colon (B,C,A) & B\colon (a,c,b) \\ a\colon (A,B,C) & A\colon (b,c,a) \end{array}$$

Figure 1: A stable matching

Co-operative Games are used in situations where players each contribute to a system and those separate contributions require their own payoff. Normal Form Games involve players choosing different strategies against each other and obtaining a payoff. Nash equilibria occur when no player has any incentive to change which strategy they play.

```
def _is_NE(self, a, b, p1_support, p2_support, M1, M2):
    # Check that supports are obeyed
    if not (all([a[i] > 0 for i in p1_support]) and
        all([b[j] > 0 for j in p2_support]) and
        all([a[i] == 0 for i in range(len(a)) if i not in p1_support]) and
        all([b[j] == 0 for j in range(len(b)) if j not in p2_support])):
        return False

    # Check that have pair of best responses
    p1_payoffs = [sum(v * row[i] for i, v in enumerate(b)) for row in M1.rows()]
    p2_payoffs = [sum(v * col[j] for j, v in enumerate(a)) for col in M2.columns()]

    if p1_payoffs.index(max(p1_payoffs)) not in p1_support:
        return False
    if p2_payoffs.index(max(p2_payoffs)) not in p2_support:
        return False

    return True
```

Figure 2: Example Code

Fig 2 is a function we wrote to check whether two support vectors are in fact Nash Equilibrium. It goes throught the formal definition of a Nash Equilibrium and ensures that every condition is met.

Stackelberg game

The issue of waiting times for ambulances at at two hospitals can be modelled as a 3 player Stackelberg game where each hospital has its own AE and Ward.

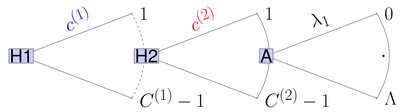


Figure 3: Underlying Stackelberg Game

To be able to solve the game, a queueing model of the stochastic process associated with the hospital is required. Patients arrive at the AE at rate λ and if there is space in the queue they join it. If there is no space in the queue that patient is lost. Each patient has an AE service rate, μ, which represents how long their treatment in AE will last. A proportion, p, of patients are then dismissed immediately. Those who are not dismissed are admitted to the ward if there is space, otherwise they will wait in AE, continuing to block a bed. Once admitted, they are treated in the ward with a service time $\hat{\mu}$ and then dismissed without delay.

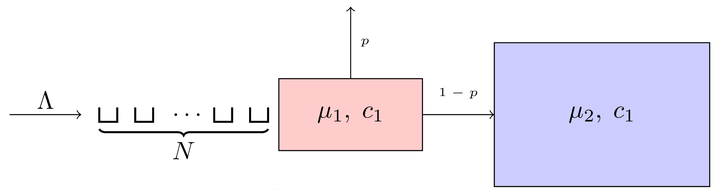


Figure 4: A single hospital

Markov Chain Formulation

A Markov Chain is ideal for this because the system is memoryless, ie. at any one time the probability of a patient arriving or finishing their service is independent of what has happened before.

Validation

Validation of the Markov Chain model was carried out by comparing various results from a Discrete Event simulation with analytical results from the Markov Chain.

In Fig 5 the blue is the analytical approach and the box plots show simulated data.
Fig 6 shows the probability of being in state (i, j) for varying λ. Again it compares analytical results to simulated ones.

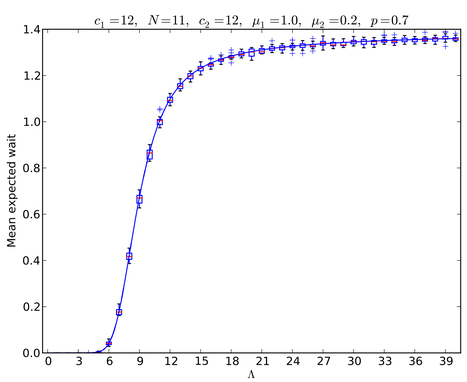
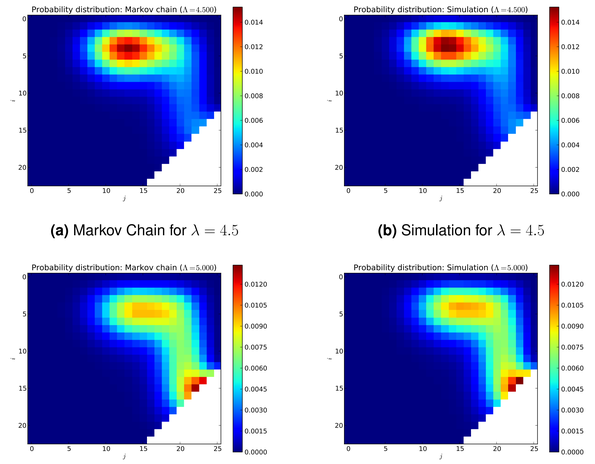


Figure 5: Theoretical and simulated expected wait times



(a) Markov Chain for $\lambda = 4.5$ **(b)** Simulation for $\lambda = 4.5$

(c) Markov Chain for $\lambda = 5$ **(d)** Simulation for $\lambda = 5$

Figure 6: Theoretical and simulated distribution of π for $\Lambda \in \{4.5, 5\}$

Limitations of the Markov Model

The main problem with the Markov Chain approach is that calculating the payoffs required for the associated Normal Form Game is computationally expensive. It therefore required use of the Cardiff University supercomputer, RAVEN. This approach also assumes that the ambulances do not observe the system.

Markov Decision Process

A Markov Decision Process is a model of decision making in a dynamic framework in which a decision maker makes decisions based on a particular system state. The process is that an agent starts off in a particular state, makes a decision, and then has a probabilistic transition depending on the previous decision.

Q-Learning

Q-learning is the process of assigning a state-action value or Q-value to the combination of being in a state, taking an action and observing a reward. The Q-value is then updated by assessing the maximum value of being in the new state. The higher the Q-value the more likely a player is to choose action a when in state s. The effect of using Q-Learning with the ambulances can be seen in Fig 7.

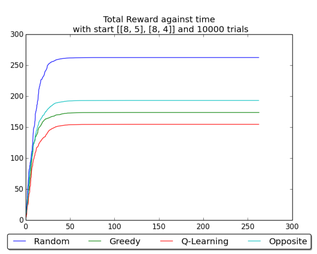


Figure 7: The effect of Q-learning