A game theoretic model of the behavioural gaming that takes place at the EMS - ED interface

Michalis Panayides

Contents

1 Introduction

2 Overview of game theoretic model

The problem studied is a 3-player normal form game. The players are:

- the decision makers of two queueing systems;
- a service that distributes individuals to these two queueing systems.

This is a standard normal form game [15], in that each player in this game has their own objectives which they aim to optimise. More specifically, the queueing systems' objective is captured by an upper bound of the time that a fixed proportion of individuals spend in the system, while the distributor aims to minimise the time that its individuals are blocked.

The queueing systems are designed in such a way where they can accept two types of individuals. These are the individuals that the distributor allocates to them and other individuals from other sources. Each queueing system may then choose to block the individuals that arrive from the distributor when the system reaches a certain capacity. The strategy sets for each queueing system is the set $\{T \in \mathbb{N} \mid 1 \leq T \leq N\}$ where $N \in \{N_A, N_B\}$ are the total capacities of the two queueing systems. We denote the chosen actions from the strategy set as T_A, T_B and call these *threshold*s.

Both queueing systems follow a queueing model that has two waiting spaces for individuals. The first waiting zone is where the individuals queue right before receiving their service and has a capacity of $N-C$, where N is the total capacity of the waiting space and C is the number of servers. The second waiting zone is where the individuals, that are sent from the distributor, stay until they are allowed to enter the first waiting zone. The second waiting zone has a capacity of M and no servers.

This is shown diagrammatically in Figure 1.

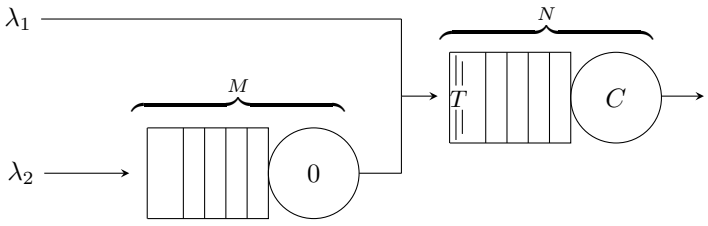

Figure 1: A diagrammatic representation of the queueing model. The threshold T only applies to arrivals from the first buffer. If the second buffer is at that threshold only individuals of the first type are accepted (at a rate λ_1) and individuals of the second type (arriving at a rate λ_2) are held blocked in the first buffer.

Note here that both types of individuals can become lost to the system. Individual allocated from the distributor become lost to the system whenever an arrival occurs and the second waiting zone is at full capacity (M individuals already waiting). Similarly, other individuals get lost whenever they arrive at the first waiting zone and it is at full capacity ($N - C$ individuals already waiting).

Following this queuing model, the two queueing systems' choice of strategy will then rely solely on satisfying their own objective, which is to make sure that the waiting time in the first waiting zone of a proportion of individuals will be below the predefined target time.

$$P(W < t) \geq \hat{P} \tag{1}$$

where W is the mean waiting time of all individuals, t is the time target and $\hat{P}$ is the percentage of individuals need to be within that target. There are numerous objective functions that can be used to capture this behaviour. For example one approach is to use the threshold that maximises the probability that the mean waiting is more than the target time, and completely ignore the percentage goal.

$$\arg\max_{T_i} \quad P(W_i < t) \tag{2}$$

A more sophisticated objective function would be to get the proportion of individuals as close to the percentage aim. In other words, to find the threshold that minimises the difference between the probability and the percentage goal (or maximise its negation).

$$\arg\max_{T_i} \quad -\Big(\hat{P} - P(W_i < t)\Big)^2 \tag{3}$$

The third player, the distributor has their own choices to make and their own goals to satisfy. The strategy set of the third player is the proportion $0 \leq p_A \leq 1$ of individuals it sends to the first queueing system (the proportion $p_B = 1 - p_A$ is sent to the second queueing system). In addition, the distributor aims to minimise any potential blockages that may occur, given the pair of thresholds chosen by the two queueing systems. Thus, its objective is to minimise the blocked time of the individuals (B_i) that they send to the two queueing systems. Apart from the time being blocked, an additional aspect that may affect the decision of the distributor is the proportion of lost individuals (L_i). Equation 2 can be used to capture a mixture between the two objectives L_i and B_i where $i \in \{A, B\}$.

$$(p_A, p_B) \quad s.t. \quad \alpha L_A(p_A) + (1-\alpha)B_A(p_A) = \alpha L_B(p_B) + (1-\alpha)B_B(p_B)$$

Here, α represents the "importance" of each objective, where high α indicates a higher weight on the proportion of lost individuals and smaller α a higher weight on the time blocked.

Using equations (3) and (2) gives an imperfect information extensive form game. An imperfect information game is defined as an extensive form game where some of the information about the game state is hidden for at least one of the players [4]. In this study the state of the problem that is hidden is the threshold that each of the first two players chooses to play. In other words, each queueing system chooses to play a strategy without knowing the other system's strategy. The distributor then, fully aware of the chosen threshold strategies, distributes individuals among the two systems in order to minimise the time that its individuals will be blocked. Figure 2 illustrates this.

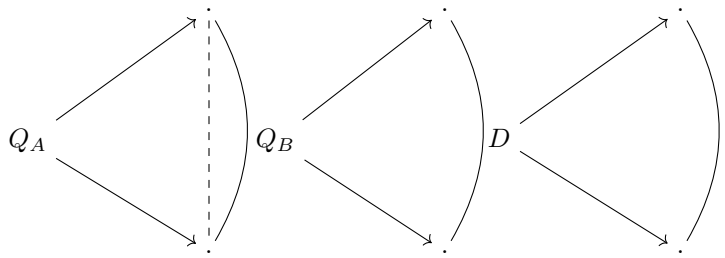

Figure 2: Imperfect information Extensive Form Game between the distributor and the 2 queueing systems

The first queueing system Q_A decides on a threshold, then the second system Q_B chooses its own threshold, without knowing the strategy of Q_A, and finally the distributor makes its choice. Note here that the dotted line represents the fact that Q_B is unaware of the state of the game when making its own decisions.

In order to define the normal form game the two payoff matrices of the players are required. From equation 3 the utilities of the players can be formulated as:

$$U^i_{T_1,T_2} = -\left(\hat{P} - P(W_i < t)\right)^2 \quad \textbf{where } i \in \{A, B\} \tag{4}$$

Consequently, the payoff matrices of the game can be populated by these utilities:

$$A = \begin{pmatrix} U^A_{1,1} & U^A_{1,2} & \dots & U^A_{1,N_B} \\ U^A_{2,1} & U^A_{2,2} & \dots & U^A_{2,N_B} \\ \vdots & \vdots & \ddots & \vdots \\ U^A_{N_A,1} & U^A_{N_A,2} & \dots & U^A_{N_A,N_B} \end{pmatrix}, B = \begin{pmatrix} U^B_{1,1} & U^B_{1,2} & \dots & U^B_{1,N_B} \\ U^B_{2,1} & U^B_{2,2} & \dots & U^B_{2,N_B} \\ \vdots & \vdots & \ddots & \vdots \\ U^B_{N_A,1} & U^B_{N_A,2} & \dots & U^B_{N_A,N_B} \end{pmatrix} \tag{5}$$

Based on the choice of strategy of these two players the distributor will then make their own choice of the proportion of individuals to send to each system.

3 A queueing model with 2 consecutive buffer centres

In this section, a more in-depth explanation of the queueing model shown in figure 1 will be given. This is a queuing model that consists of two waiting spaces, one for each type of individual.

The model consists of two types of individuals; type 1 and type 2. Type 1 individuals arrive instantly at waiting zone 1 and proceed to wait to receive their service. Type 2 individuals arrive at waiting zone 2 and wait there until they are allowed to move to waiting zone 1. They are allowed to proceed only when the number of individuals in waiting zone 1 **and** in service is less than a pre-determined threshold T. When the number of individuals is equal to or exceeds this threshold, all second type individuals that arrive will stay *"blocked"* in waiting zone 1 until the number of people in the system is reduced below T. This is shown diagrammatically in figure 1. The parameters of the described queueing model are:

- λ_i: The arrival rate of individuals of type $i \in \{1, 2\}$
- μ: The service rate for individuals receiving service
- C: The number of servers
- T: The threshold at which individuals of the second type are blocked

Under the assumption that all rates (arrival and service) are Markovian the queuing system corresponds to a Markov chain [10]. The states of the Markov chain are denoted by (u, v) where:

- u is the number of individuals blocked
- v is the number of individuals either in waiting zone 1 or in the service centre

We denote the state space of the Markov chain as $S = S(T)$ which can be written as the disjoint union (6).

$$\begin{aligned} S(T) =& S_1(T) \cup S_2(T) \text{ where:} \\ S_1(T) =& \left\{(0, v) \in \mathbb{N}_0^2 \mid v < T\right\} \\ S_2(T) =& \{(u, v) \in \mathbb{N}_0^2 \mid v \geq T\} \end{aligned} \tag{6}$$

The transition matrix Q of the Markov chain consists of the transition rates between the numerous states of the model. Every entry $Q_{ij} = Q_{(u_i,v_i),(u_j,v_j)}$ represents the transition rate from state $i = (u_i, v_i)$ to state $j = (u_j, v_j)$ for all $(u_i, v_i), (u_j, v_j) \in S$. The entries of Q can be calculated using the state-mapping function described in (7):

$$Q_{ij} = \begin{cases} \Lambda, & \textbf{if } (u_i, v_i) - (u_j, v_j) = (0, -1) \textbf{ and } v_i < \mathrm{t} \\ \lambda_1, & \textbf{if } (u_i, v_i) - (u_j, v_j) = (0, -1) \textbf{ and } v_i \geq \mathrm{t} \\ \lambda_2, & \textbf{if } (u_i, v_i) - (u_j, v_j) = (-1, 0) \\ v_i\mu, & \textbf{if } (u_i, v_i) - (u_j, v_j) = (0, 1) \textbf{ and } v_i \leq C \textbf{ or} \\ & \quad (u_i, v_i) - (u_j, v_j) = (1, 0) \textbf{ and } v_i = T \leq C \\ C\mu, & \textbf{if } (u_i, v_i) - (u_j, v_j) = (0, 1) \textbf{ and } v_i > C \textbf{ or} \\ & \quad (u_i, v_i) - (u_j, v_j) = (1, 0) \textbf{ and } v_i = T > C \\ -\sum_{j=1}^{|Q|} Q_{ij} & \textbf{if } i = j \\ 0, & \textbf{otherwise} \end{cases} \tag{7}$$

Note that Λ here denotes the overall arrival rate in the model by both types of individuals (i.e. $\Lambda = \lambda_1 + \lambda_2$). A visualisation of how the transition rates relate to the states of the model can be seen in the general Markov chain model shown in figure 3.

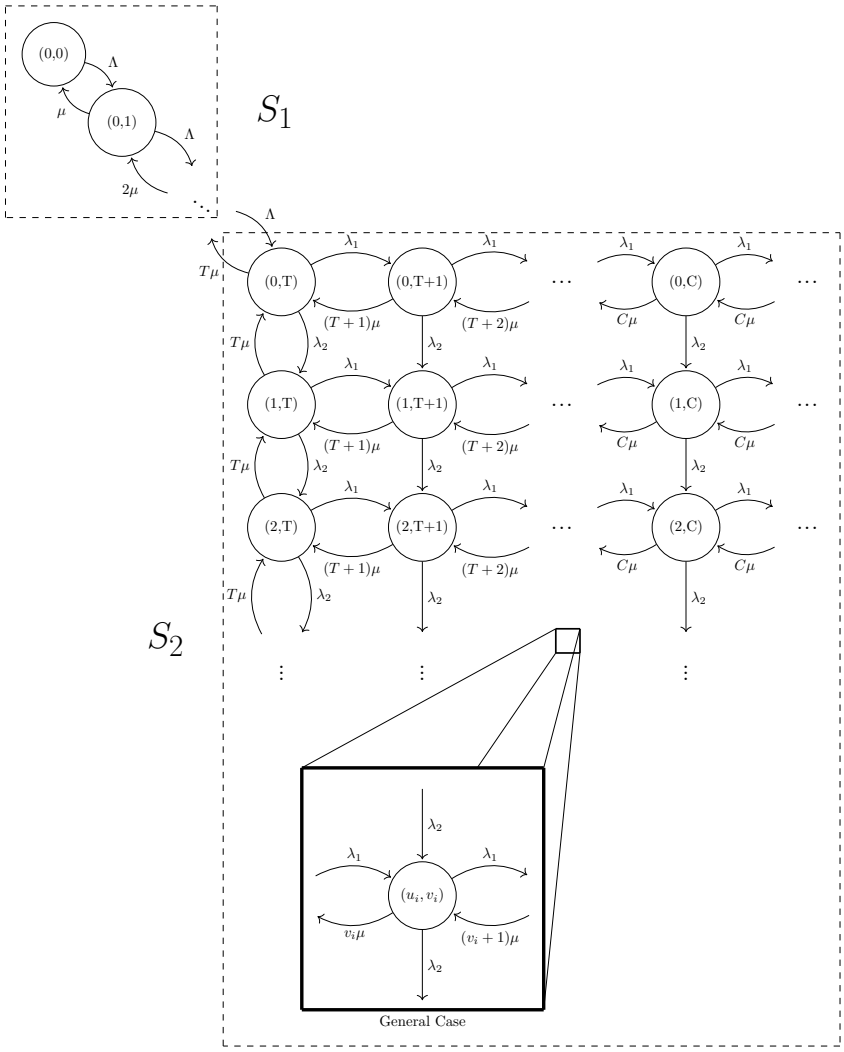

Figure 3: General case of the Markov chain model

In order to consider this model numerically an adjustment needs to be made. The problem defined above assumes no upper boundary to the number of individuals that can wait for service or for the ones that are blocked in the buffer centre. Therefore, a different state space $\tilde{S}$ is constructed where $\tilde{S} \subseteq S$ and there is a maximum allowed number of individuals N that can be in the system and a maximum allowed number of individuals M that can be blocked in the buffer centre:

$$\tilde{S} = \{(u, v) \in S \,|u \leq M, v \leq N\} \tag{8}$$

The transition matrix Q defined in (7) can be used to get the probability vector π. The vector π is commonly used to study stochastic systems and it's main purpose is to keep track of the probability of being at any given state of the system. π_i is the steady state probability of being in state $(u_i, v_i) \in \tilde{S}$ which is the i^{th} state of $\tilde{S}$ for some ordering of $\tilde{S}$. The term *steady state* refers to the instance of the vector π where the probabilities of being at any state become stable over time. Thus, by considering the steady state vector π the relationship between it and Q is given by:

$$\frac{d\pi}{dt} = \pi Q = 0$$

Using vector π there are numerous performance measures of the model that can be calculated. The following equations utilise π to get performance measures on the average number of people at certain sets of state.

- Average number of people in the system:

$$L = \sum_{i=1}^{|\pi|} \pi_i (u_i + v_i)$$

- Average number of people in the service centre:

$$L_H = \sum_{i=1}^{|\pi|} \pi_i v_i$$

- Average number of people in waiting zone 2:

$$L_A = \sum_{i=1}^{|\pi|} \pi_i u_i$$

Consequently, there are some additional performance measures of interest that are not as straightforward to calculate. Such performance measures are the mean waiting time in the system (for both type 1 and type 2 individuals), the mean time blocked in waiting zone 2 (only valid for type 2 individuals) and the proportion of individuals that wait in waiting zone 1 within a predefined time target.

3.1 Waiting time

Waiting time is the amount of time that individuals from either type wait in waiting zone 1 so that they can receive their service. For a given set of parameters there are three different performance measures around the mean waiting time that can be calculated; the mean waiting time of type 1 individuals:

$$W^{(1)} = \frac{\sum_{\substack{(u,v) \in S_A^{(1)} \\ v \geq C}} \frac{1}{C\mu} \times (v - C + 1) \times \pi(u,v)}{\sum_{(u,v) \in S_A^{(1)}} \pi(u,v)} \tag{9}$$

The mean waiting time of type 2 individuals:

$$W^{(2)} = \frac{\sum_{\substack{(u,v) \in S_A^{(2)} \\ min(v,T) \geq C}} \frac{1}{C\mu} \times (\min(v+1,T) - C) \times \pi(u,v)}{\sum_{(u,v) \in S_A^{(2)}} \pi(u,v)} \tag{10}$$

The overall mean waiting time:

$$W = \frac{\lambda_1 P_{L_1'}}{\lambda_2 P_{L_2'} + \lambda_1 P_{L_1'}} W^{(1)} + \frac{\lambda_2 P_{L_2'}}{\lambda_2 P_{L_2'} + \lambda_1 P_{L_1'}} W^{(2)} \tag{11}$$

Here $S_A^{(1)}$ and $S_A^{(2)}$ are the set of accepting states for type 1 and type 2 individuals. These are the set of states that the model is able to accept a specific type of individuals.

$$S_A^{(1)} = \{(u,v) \in S \mid v < N\} \tag{12}$$

$$S_A^{(2)} = \begin{cases} \{(u,v) \in S \mid u < M\}, & \textbf{if } T \leq N \\ \{(u,v) \in S \mid v < N\}, & \textbf{otherwise} \end{cases} \tag{13}$$

Equation 11 makes use of the proportion of type 1 and type 2 individuals that are not lost to the system. These probabilities are given by $P_{L_1'}$ and $P_{L_2'}$ where:

$$P_{L_1'} = \sum_{(u,v) \in S_A^{(1)}} \pi(u,v) \qquad\qquad P_{L_2'} = \sum_{(u,v) \in S_A^{(2)}} \pi(u,v) \tag{14}$$

Appendix B gives more details on the recursive formula that equations (9), (10) and (11) originate from.

Figures 4, 5 and 6 show a comparison between the calculated mean waiting time using Markov chains and the simulated waiting time using discrete event simulation over a range of values of λ_2 (details of the discrete event simulation model are given in appendix A). The simulation was ran 100 times and the recorded mean waiting time at each iteration is used to populate the violin plots.

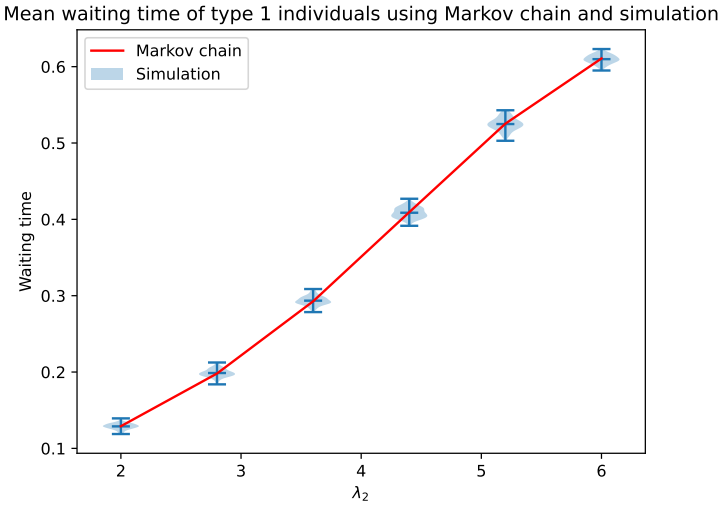

Figure 4: Comparison of mean waiting time for type 1 individuals between values obtained from the Markov chain formulas and values obtained from simulation.

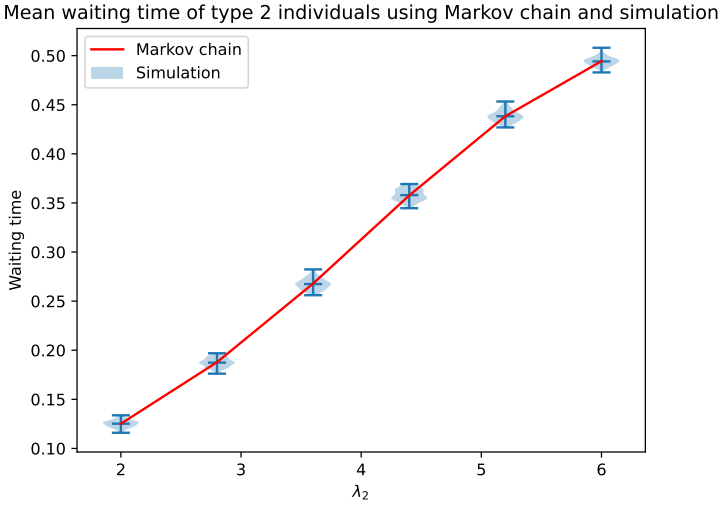

Figure 5: Comparison of mean waiting time for type 2 individuals between values obtained from the Markov chain formulas and values obtained from simulation.

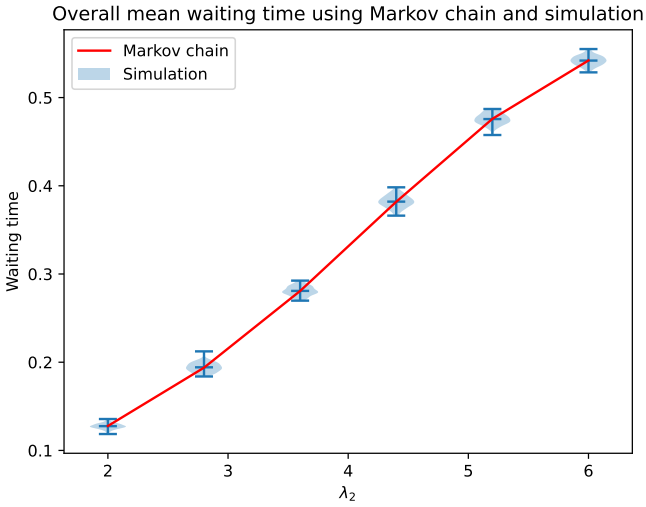

Figure 6: Comparison of mean waiting time for both types of individuals between values obtained from the Markov chain formulas and values obtained from simulation.

3.2 Blocking time

Unlike the waiting time, the blocking time is only calculated for individuals of the second type. That is because individuals of the first type cannot be blocked. Thus, one only needs to consider the pathway of type 2 individuals to get the mean blocking time of the system. The mean blocking time can by calculated using:

$$B = \frac{\sum_{(u,v)\in S_A^{(2)}} \pi_{(u,v)}\ b(u,v)}{\sum_{(u,v)\in S_A^{(2)}} \pi_{(u,v)}} \tag{15}$$

Here $S_A^{(2)}$ is the set of accepting states of type 2 individuals (defined in (13)) and $b(u,v)$ is the mean time that an individual will be blocked when the system is at state (u,v). For all the states of the system $b(u,v)$ is given by:

$$b(u,v) = \begin{cases} 0, & \textbf{if } (u,v) \notin S_b \\ c(u,v) + b(u-1,v), & \textbf{if } v = N = T \\ c(u,v) + b(u,v-1), & \textbf{if } v = N \neq T \\ c(u,v) + p_s(u,v)b(u-1,v) + p_a(u,v)b(u,v+1), & \textbf{if } u > 0 \textbf{ and } v = T \\ c(u,v) + p_s(u,v)b(u,v-1) + p_a(u,v)b(u,v+1), & \textbf{otherwise} \end{cases} \tag{16}$$

Note that S_b is defined as the set of states where individuals can be blocked and is given by:

$$S_b = \{(u, v) \in S \mid u > 0\} \tag{17}$$

Additionally, $c(u, v)$ is the mean sojourn time for each state and p_s and p_a are the probabilities that the next event to occur will be a service completion or an arrival of a type 1 individual:

$$c(u,v) = \begin{cases} \frac{1}{\min(v,C)\mu}, & \text{if } v = N \\ \frac{1}{\lambda_1 + \min(v,C)\mu}, & \text{otherwise} \end{cases} \tag{18}$$

$$p_s(u,v) = \frac{\min(v,C)\mu}{\lambda_1 + \min(v,C)\mu}, \qquad p_a(u,v) = \frac{\lambda_1}{\lambda_1 + \min(v,C)\mu} \tag{19}$$

The system of equations produced by (16) can be solved by considering the linear system $Zx = y$. Assuming i and j represent states $(u_i, v_i), (u_j, v_j) \in S_b$ then Z_{ij} is given by:

$$Z_{ij} = \begin{cases} p_a, & \textbf{if } j = i+1 \textbf{ and } v_i \neq N \\ p_s, & \textbf{if } j = i-1 \textbf{ and } v_i \neq N, v_i \neq T \\ & \textbf{or } j = i - N + T \textbf{ and } u_i \geq 2,\ v_i = T \\ 1, & \textbf{if } j = i-1 \textbf{ and } v_i = N \\ -1, & \textbf{if } i = j \\ 0, & \textbf{otherwise} \end{cases} \tag{20}$$

Equation (21) shows this.

$$Z = \begin{pmatrix} -1 & p_a & 0 & \dots & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ p_s & -1 & p_a & \dots & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & p_s & -1 & \dots & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & -1 & 0 & 0 & 0 & \dots & 0 & 0 \\ p_s & 0 & 0 & \dots & 0 & 0 & -1 & p_a & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & p_s & -1 & p_a & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & \dots & 1 & -1 \end{pmatrix}, x = \begin{pmatrix} b(1,T) \\ b(1,T+1) \\ b(1,T+2) \\ \vdots \\ b(1,N) \\ b(2,T) \\ b(2,T+1) \\ \vdots \\ b(M,N) \end{pmatrix}, y = \begin{pmatrix} -c(1,T) \\ -c(1,T+1) \\ -c(1,T+2) \\ \vdots \\ -c(1,N) \\ -c(2,T) \\ -c(2,T+1) \\ \vdots \\ -c(M,N) \end{pmatrix} \tag{21}$$

Additional details on the blocking time formula (15) can be found in appendix C.

Figure 7 illustrates a comparison between the formulas that arise from the Markov chain model and the equivalent values of the blocking time extracted from discrete event simulation (appendix A). The blocking time is calculated using both methods for a range of values of λ_2. The simulation was ran 100 times and the recorded mean blocking time at each iteration is used to populate the violin plots.

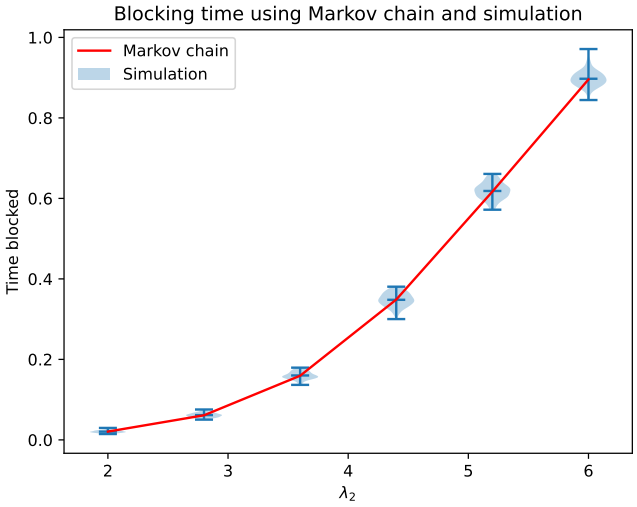

Figure 7: Comparison of mean blocking time between values obtained from the Markov chain formulas and values obtained from simulation.

3.3 Proportion of individuals within target

Another performance measure that is taken into consideration is the proportion of individuals whose waiting and service times are within a specified time target t. Similar to section 3.1 three formulas are needed for this performance measure.

The proportion of type 1 individuals within a time target:

$$P(X^{(1)} < t) = \frac{\sum_{(u,v)\in S_A^{(1)}} P(X^{(1)}_{(u,v)} < t)\pi_{u,v}}{\sum_{(u,v)\in S_A^{(1)}} \pi_{u,v}} \tag{22}$$

The proportion of type 2 individuals within a time target:

$$P(X^{(2)} < t) = \frac{\sum_{(u,v)\in S_A^{(2)}} P(X^{(2)}_{(u,v)} < t)\pi_{u,v}}{\sum_{(u,v)\in S_A^{(2)}} \pi_{u,v}} \tag{23}$$

The overall proportion individuals within a time target (where $P_{L_1'}$ and $P_{L_1'}$ are defined in (14)):

$$P(X < t) = \frac{\lambda_1 P_{L_1'}}{\lambda_2 P_{L_2'} + \lambda_1 P_{L_1'}} P(X^{(1)} < t) + \frac{\lambda_2 P_{L_2'}}{\lambda_2 P_{L_2'} + \lambda_1 P_{L_1'}} P(X^{(2)} < t) \tag{24}$$

Here $P(X^{(1)}_{(u,v)})$ and $P(X^{(2)}_{(u,v)})$ are defined as the proportion of individuals within the time target t when starting from state (u, v). These expression can be calculated by:

$$P(X^{(1)}_{(u,v)} < t) = \begin{cases} 1 - \sum_{i=0}^{v-1} \frac{1}{i!} e^{-\mu t}(\mu t)^i, & \textbf{if } C = 1 \\ & \textbf{and } v > 1 \\ 1 - (\mu C)^{v-C} \mu \sum_{k=1}^{|\vec{r}|} \sum_{l=1}^{r_k} \frac{\Psi_{k,l}(-\lambda_k) t^{r_k - l} e^{-\lambda_k t}}{(r_k - l)!(l-1)!}, & \textbf{if } C > 1 \\ & \textbf{and } v > C \\ 1 - e^{-\mu t}, & \textbf{if } v \leq C \end{cases} \tag{25}$$

where $\vec{r} = (v - C, 1)$, $\vec{\lambda} = (C\mu, \mu)$ and $\lambda_0 = 0, r_0 = 1$.

$$P(X^{(2)}_{(u,v)} < t) = \begin{cases} 1 - \sum_{i=0}^{\min(v,T)-1} \frac{1}{i!} e^{-\mu t}(\mu t)^i, & \textbf{if } C = 1 \\ & \textbf{and } v, T > 1 \\ 1 - (\mu C)^{\min(v,T)-C} \mu \sum_{k=1}^{|\vec{r}|} & \textbf{if } C > 1 \\ \quad \times \sum_{l=1}^{r_k} \frac{\Psi_{k,l}(-\lambda_k) t^{r_k - l} e^{-\lambda_k t}}{(r_k - l)!(l-1)!}, & \textbf{and } v, T > C \\ 1 - e^{-\mu t}, & \textbf{if } v \leq C \\ & \textbf{or } T \leq C \end{cases} \tag{26}$$

where $\vec{r} = (\min(v, T) - C, 1)$, $\vec{\lambda} = (C\mu, \mu)$ and $\lambda_0 = 0, r_0 = 1$.

The function $\Psi_{k,l}$ used in equations (25) and (26) is defined as:

$$\Psi_{k,l}(t) = \begin{cases} \frac{(-1)^l (l-1)!}{\lambda_2} \left[\frac{1}{t^l} - \frac{1}{(t+\lambda_2)^l} \right], & k = 1 \\ -\frac{1}{t(t+\lambda_1)^{r_1}}, & k = 2 \end{cases}$$

Refer to appendix D for a more in-depth explanation of the origins of equations (22) - (26).

Figures 8, 9 and 10 show a comparison of the mean proportion of individuals within target when using Markov chains and discrete event simulation (appendix A). The simulation was ran 100 times and the recorded proportions at each iteration is used to populate the violin plots.

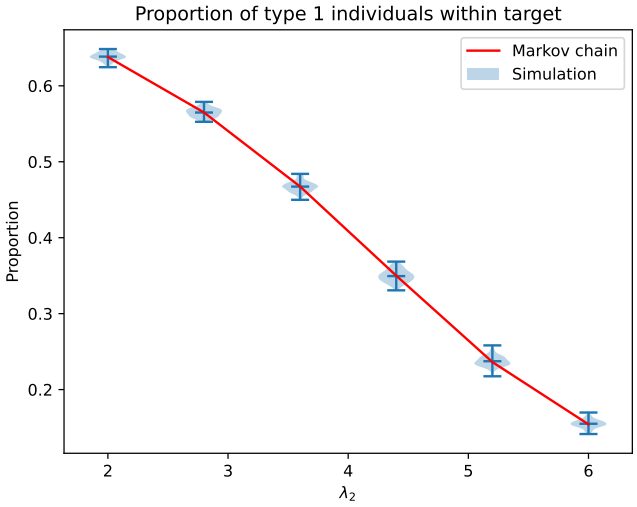

Figure 8: Comparison of proportion within target time for type 1 individuals between values obtained from the Markov chain formulas and values obtained from simulation.

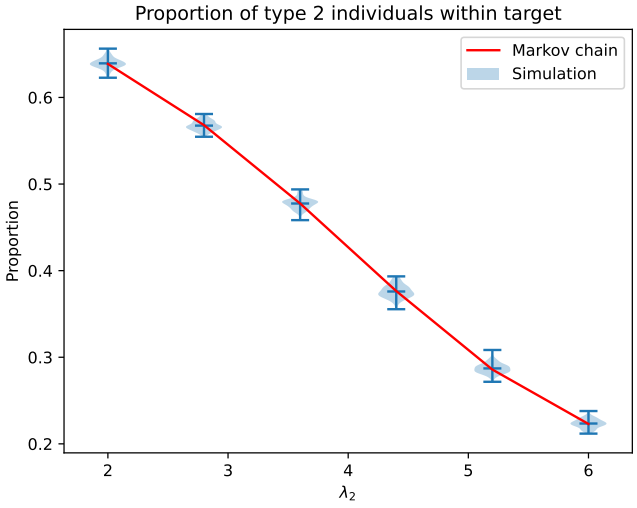

Figure 9: Comparison of proportion within target time for type 2 individuals between values obtained from the Markov chain formulas and values obtained from simulation.

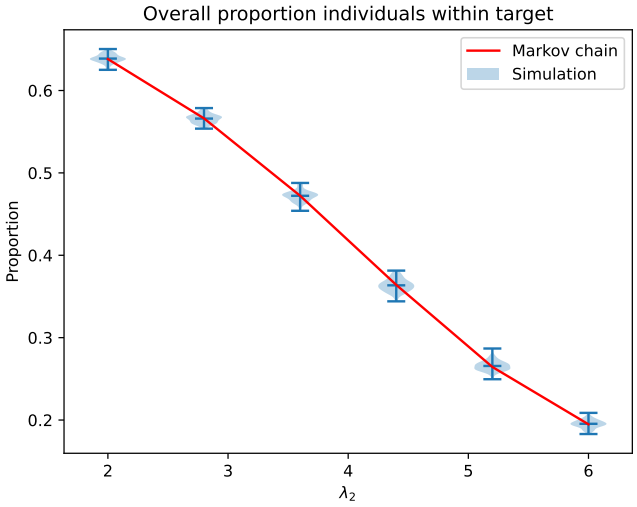

Figure 10: Comparison of proportion within target time for both type of individuals between values obtained from the Markov chain formulas and values obtained from simulation.

4 Methodology

Th problem defined in section 2 describes a normal-form game between the decision makers of two queueing systems and a third player that decides how to distribute individuals to the systems. The strategy space of the two queueing systems is defined as the possible values that the threshold parameter can take $T_i \in (1, N_i)$. Then, the distributor has to decide on the proportion of individuals to send to each queueing system p_A and p_B, where $p_A, p_B \in [0, 1]$ and $p_A + p_B = 1$. Figure 2 form section 2 shows a diagrammatic representation of the game to be played and the decisions to be made. As described in section 2 queueing system A decides on a strategy, then queueing system B chooses its own threshold, unaware of the first queueing system's choice. Finally, the distributor makes its choice based on the strategies that the queueing systems chose to play.

The problem can be formulated by 3 matrices; the two payoff matrices of the normal form game and the routing matrix. The payoff matrices and their utilities are defined by equations (4) and (5) in section 2.

The routing matrix holds the values (p_A, p_B) which are the proportion of class 2 individuals to send to queueing system A and B. Each pair (p_A, p_B) can be calculated using equation (2). Equation 2 is defined and explained in section 2. Thus, using equation 2 for all possible sets of thresholds we can get the full routing matrix that consists of the proportions to send to queueing system A (p_A) and to queueing system B (p_B).

$$R = \begin{pmatrix} (p^A_{1,1}, p^B_{1,1}) & (p^A_{1,2}, p^B_{1,2}) & \cdots & (p^A_{1,N_B}, p^B_{1,N_B}) \\ (p^A_{2,1}, p^B_{2,1}) & (p^A_{2,2}, p^B_{2,2}) & \cdots & (p^A_{2,N_B}, p^B_{2,N_B}) \\ \vdots & \vdots & \ddots & \vdots \\ (p^A_{N_A,1}, p^B_{N_A,1}) & (p^A_{N_A,2}, p^B_{N_A,2}) & \cdots & (p^A_{N_A,N_B}, p^B_{N_A,N_B}) \end{pmatrix} \tag{27}$$

Note that since $p^A_{i,j} + p^B_{i,j} = 1$ the routing matrix needs only to store one of the two values; either $p^A_{i,j}$ or $p^B_{i,j}$. Thus, the routing matrix R can be simplified to:

$$R = \begin{pmatrix} p^A_{1,1} & p^A_{1,2} & \cdots & p^A_{1,N_B} \\ p^A_{2,1} & p^A_{2,2} & \cdots & p^A_{2,N_B} \\ \vdots & \vdots & \ddots & \vdots \\ p^A_{N_A,1} & p^A_{N_A,2} & \cdots & p^A_{N_A,N_B} \end{pmatrix} \tag{28}$$

The game can thus be partitioned into a normal form game between the two queueing systems and then finding the distributor's best strategy.

4.1 Backwards Induction

In order to populate the payoff matrices and the routing matrix the method of backwards induction is used. Consider figure 2 and the flow of the game that was described (i.e. $Q_A, Q_B \rightarrow D$). Due to the fact that the payoff matrices A and B depend on the routing matrix R the entries of the matrices are calculated in a backwards way ($D \rightarrow Q_A, Q_B$). Thus for every pair of strategies T_A, T_B, the proportion of individuals that are to be transported to each queueing system is calculated first.

Consider a game where the capacities of the two systems are $N_A = 4$ and $N_B = 3$. The strategy space of players Q_A and Q_B would then be $T_A = \{1, 2, 3, 4\}$ and $T_B = \{1, 2, 3\}$ respectively. Now, starting from an arbitrary starting point of $T_A = 1$ and $T_B = 1$, the corresponding entry of the routing matrix the routing matrix can be calculated. Using equation (2) for $T_A = 1$ and $T_B = 1$:

$$\begin{aligned} &\alpha L_A(p_A; T_A = 1; T_B = 1) + (1 - \alpha) B_A(p_A; T_A = 1; T_B = 1) = \\ &\alpha L_B(p_B; T_A = 1; T_B = 1) + (1 - \alpha) B_B(p_B; T_A = 1; T_B = 1) \end{aligned} \tag{29}$$

The values of p_A and p_B that satisfy equation (29) can be found by using Brent's bisection algorithm [5]. The calculated value of p_A corresponds to the entry on the first row and first column of the routing matrix:

$$R = \begin{pmatrix} p^A_{1,1} & - & - \\ - & - & - \\ - & - & - \\ - & - & - \end{pmatrix} \tag{30}$$

In fact all remaining values of the routing matrix can be calculated by solving $N \times M$ similar equations using Brent's bisection algorithm. Therefore, having calculated the entire routing matrix the payoff matrices can be calculated. Consider once again the case of $T_A = 1, T_B = 1$.

$$U^A_{1,1} = 1 - \left(\hat{P} - P(W_A(T_A = 1, T_B = 1) < t)\right)^2$$
$$U^B_{1,1} = 1 - \left(\hat{P} - P(W_B(T_A = 1, T_B = 1) < t)\right)^2 \tag{31}$$

These are the utilities of players A and B when both players choose a strategy of $T = 1$. Here R is the predefined waiting time target to be met by a percentage of individuals $\hat{P}$ and $P(W_i < t)$ is properly defined in section 3.3.

$$A = \begin{pmatrix} U^A_{1,1} & - & - \\ - & - & - \\ - & - & - \\ - & - & - \end{pmatrix}, \quad B = \begin{pmatrix} U^B_{1,1} & - & - \\ - & - & - \\ - & - & - \\ - & - & - \end{pmatrix} \tag{32}$$

Similar to the routing matrix, the above procedure can be repeated for all possible values of T_A and T_B to populate all entries of the payoff matrices.

4.2 Nash Equilibrium

Having calculated the payoff matrices A and B, several algorithms can be used to help us find the Nash equilibrium of the game. The Nash equilibrium is the point of the game were both players have no incentive to deviate from their played strategies [12]. In other words their chosen strategies are a best response to each other.

The Lemke-Howson algorithm is used to get all Nash equilibria of the game [14]. Lemke-Howson uses best response polytopes to get one of the Nash equilibrium of the game. By iterating over all possible dropped labels of the best response polytope all Nash equilibria can be acquired. All game theoretic calculations were done in Python using the Nashpy library [23].

4.3 Learning Algorithms

In order to analyse the strategies played by the two EDs, the learning algorithm asymmetric replicator dynamics is used [1]. The two players are treated as two separate populations where each individual in the population is assigned a strategy. As the game progresses the proportion of each player playing each strategy changes based on their previous interactions. The fitness of each strategy is defined as:

$$f_x = Ay, \quad f_y = x^T B \tag{33}$$

Here, $x \in \mathbb{R}^{m\times 1}$ and $y \in \mathbb{R}^{n\times 1}$ correspond to the proportion of individuals that play each strategy on the two populations. Similarly, the average fitness of each strategy is defined as:

$$\phi_x = f_x x^T, \quad \phi_y = f_y y \tag{34}$$

The rate of change of strategy i of both types of individuals is captured by:

$$\frac{dx}{dt}_i = x_i((f_x)_i - \phi_x), \quad \text{for all } i \tag{35}$$

$$\frac{dy}{dt}_i = y_i((f_y)_i - \phi_y), \quad \text{for all } i \tag{36}$$

In addition to asymmetric replicator dynamics, the learning algorithms fictitious play and stochastic fictitious play [9] were used.

5 EMS - ED application

5.1 Application

The system defined in section 2 can be applied to a healthcare scenario with two such queueing systems. The two queueing systems can be thought of as two Emergency Departments (ED) and the Emergency Medical Services (EMS) as a distributor that distributes patients to them.

There are numerous news articles that focus on the problems that arise when ambulances stay parked outside of the hospital for a long amount of time [2], [7]. Some news reports also comment on the long idle time of ambulances when they are not in use [24] and there are several reports of examples where this became an issue for new patients [16]. A particular article looked into this problem from the point of view of an ambulance paramedic:

> "He knows if a patient arrives more urgently in need of help, his will be pushed back" [6]
>
> "We used to bring poorly patients in, and we were out on the road again in 15 minutes." [6]

The patients that are distributed by the EMS arrive at the hospital via an ambulance and are then either unloaded at the ED or stay blocked outside in the ambulance. Whether or not the ambulance and the patient stay blocked is determined by the threshold that the given ED chooses to play. High threshold indicates that the ED accepts patients even if it is relatively full, while low threshold means that the ED blocks ambulances more frequently.

The parameters of the model correspond to the following parameters of the ED and the EMS:

- λ_2: The rate of non-emergency patients that the EMS receives

- λ_{1_i}: The arrival rate of other patients to ED $i \in \{1, 2\}$
- μ_i: The service rate of patients at ED $i \in \{1, 2\}$
- C_i: The number of available resources (beds) in ED $i \in \{1, 2\}$
- T_i: The action that ED $i \in \{1, 2\}$ chooses to play which corresponds to the threshold at which they do not accept EMS patients.
- N_i: The total patient capacity of ED $i \in \{1, 2\}$
- M_i: The total parking capacity of ED $i \in \{1, 2\}$
- R: The time target for both EDs
- $\alpha \in [0, 1]$: *Weighting* of blocking time and lost individuals (equation 2)

The strategies of the two EDs are the range of thresholds that they can choose from and their utilities is the proportion of individuals whose time in the system is within a predetermined target time. The EMS has to decide how to distribute its patients among the two EDs so that the weighted combination of the ambulance blocking time and the percentage of lost ambulances is minimised. As defined in 2, the interaction between the two EDs is a Normal form game that is then used to inform the decision of the EMS. Note that the formulated game here assumes that prior to making a choice the EMS knows the strategies that each ED is playing (figure 2). This corresponds to reacting to experienced delays.

5.2 Numerical Results

This subsection aims to analyse how the gaming framework can affect the performance measures of the two hospitals and how to avoid certain inefficient situations.

The most commonly used method for analysing normal form games is the Nash equilibrium described in section 4.2 Consider the following game:

- $\lambda_{1_1} = 1$
- $\lambda_{1_2} = 2$
- $\lambda_2 = 2$
- $\mu_1 = 2$
- $\mu_2 = 2.5$
- $C_1 = 2$
- $C_2 = 2$
- $R = 2$
- $N_1 = 10$
- $N_2 = 10$
- $M_1 = 6$
- $M_2 = 6$
- $\alpha = 0.5$

The set of possible actions to choose from for player 1 and player 2 is the set of thresholds that the EDs can choose from:

$$T_1 = \{0, \ldots, N_1\}, \quad T_2 = \{0, \ldots, N_2\} \tag{37}$$

For this example the only Nash equilibrium of the game is achieved when both players choose a threshold of $T_1 = 10, T_2 = 10$. This means that the two players' best response to each other is to only block ambulances when they are full.

Nash equilibria is a theoretical measure which can be inconsistent with intuitive notions about what should be the outcome of a game [17]. Therefore it might not be the best way to describe human behaviour. Since the work of Maynard Smith [20] evolutionary game theory gives the tools for the emergence of stable behaviour. One such model that allows for asymmetric payoffs, as is the case above, is replicator dynamics described in section (4.3). Stable outcomes of this algorithm will correspond to a subset of Nash equilibria but more importantly, will give a model of emergent behaviour.

The use of a learning algorithm allows to investigate, not only the outcome of the game, but also how that outcome is reached. Consider figure 11. By running asymmetric replicator dynamics on the system the behaviour that emerges can be observed. It can be seen that for this particular set of parameters the strategies of the two hospitals converge over time. Both hospital 1 (row player) and hospital 2 (column player) seem to be playing the same strategy s_{10} which indicates that thresholds $T_1 = 10$ and $T_2 = 10$ are played. What is more important in this example is how the two hospitals reached these decisions which also highlights the importance of using a learning algorithm. Hospital 2 is able to reach the decision in a short amount of time while hospital 1 takes longer and goes through numerous strategies to get there.

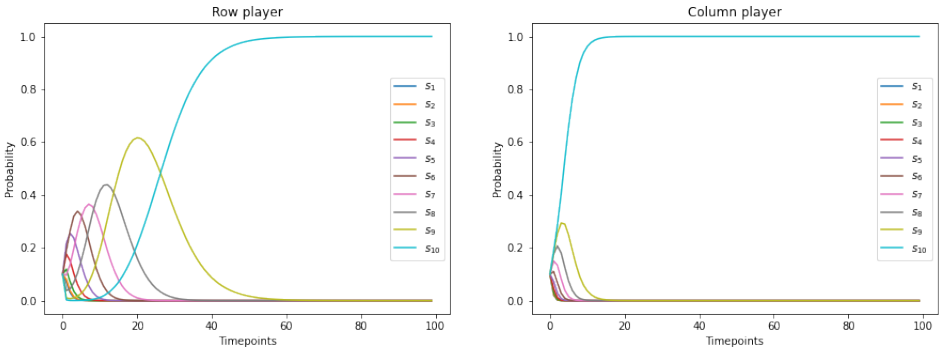

Figure 11: Asymmetric replicator dynamics run

In order to analyse how efficient the strategies played at every iteration are, the concept of the price of anarchy is used. Price of anarchy is a measure that is used to quantify the efficiency of a behaviour [19]. In other words the price of anarchy is the worst-case equilibria measure and it is defined as:

$$\text{PoA} = \frac{\max_{s \in E} F(s)}{\min_{s \in S} F(s)} \tag{38}$$

Here, S is the set of all sets of strategies (s_1, s_2), E is the set of all possible sets of equilibria and F is the cost function to measure the efficiency for.

To study the efficiency of the strategies being played a new concept is introduced that considers the ratio between each hospital's best achievable blocking time and the one that is being played. This new concept is defined as the compartmentalised price of anarchy of the players of the game and is defined as $PoA_i(\tilde{s})$, where $i \in \{1, 2\}$ to distinguish among the two players/hospitals where $\tilde{s}$ is the strategy that is being played by player i. The compartmentalised price of anarchy aims to measure inefficiencies in the model. The PoA for the blocking time of player i for strategy $\tilde{s}$ is given by:

$$PoA_i(\tilde{s}) = \frac{B_i(\tilde{s})}{\min_{s \in S_i} B_i(s)} \tag{39}$$

Consider a game with two smaller (lower N_i, M_i) and busier (higher λ_{1_i} and λ_2) hospitals with the following set of parameters:

- $\lambda_{1_1} = 4.5$
- $\mu_1 = 2$
- $C_1 = 3$
- $T_1 \in [1, N_1]$
- $N_1 = 6$
- $M_1 = 5$
- $\lambda_{1_2} = 6$
- $\mu_2 = 3$
- $C_2 = 2$
- $T_2 \in [1, N_2]$
- $N_2 = 7$
- $M_2 = 4$
- $\lambda_2 = 10.7$
- $R = 2$
- $\alpha = 0.9$

Initial scenario: Using equation (39) and asymmetric replicator dynamics the emergent behaviour can be measured and the compartmentalised price of anarchy at every iteration for both players. Figure 12 shows the strategies that are being played and the values of $PoA_1(s)$ and $PoA_2(s)$ for all iterations of the learning algorithm until it reaches an evolutionary stable pair of strategies.

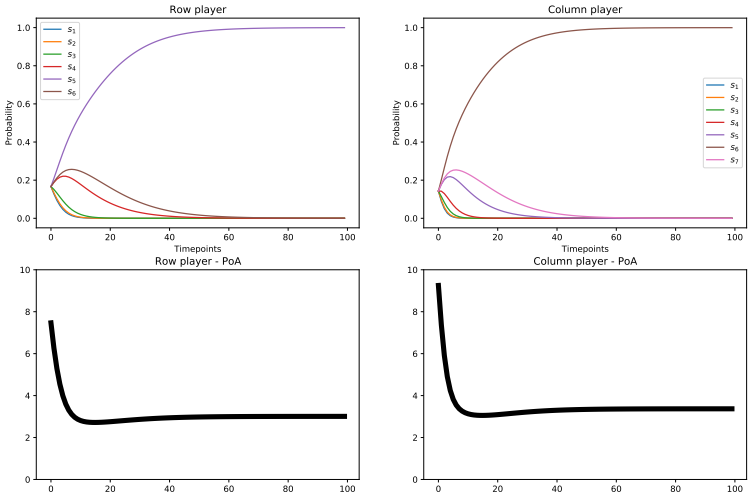

Figure 12: The strategies played when running asymmetric replicator dynamics along with the compartmentalised price of anarchy of the blocking time at each iteration of the learning algorithm

The learning algorithm reaches a stable pair of strategies where $T_1 = 5$ and $T_2 = 6$. After a number of iterations the price of anarchy for both players stabilises and barely increases until the final iteration.

Increasing λ_2: Figure 13 shows a similar run of the algorithm but when the strategies begin to stabilise an increase in the arrival rate of ambulances occurs (i.e $\lambda_2 = 24$).

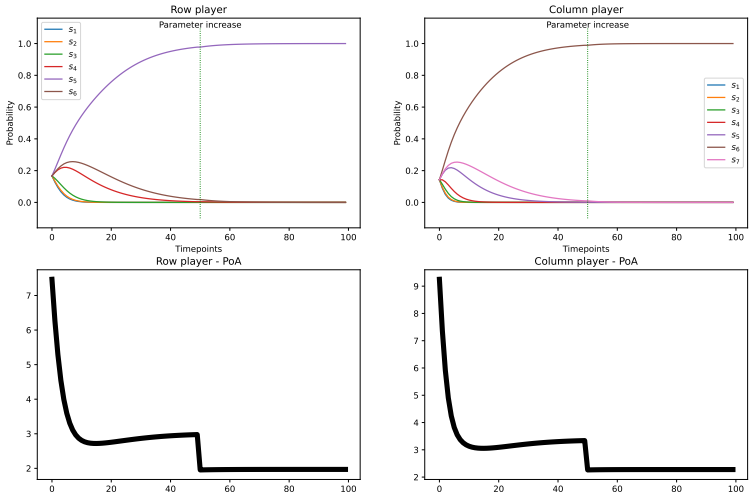

Figure 13: The strategies played when running asymmetric replicator dynamics along with the compartmentalised price of anarchy of the blocking time at each iteration of the learning algorithm. After a number of iterations the arrival rate of ambulance patients is significantly increased to flood the system completely $\lambda_2 = 24$.

By increasing λ_2 there is no change as to how players behave ($T_1 = 5, T_2 = 6$), but the efficiency of the system does change. There is a decline in the price of anarchy of the blocking time which at first glance indicates that upon flooding the system it becomes the loss in efficiency due to rational individual behaviour decreases. This is non-sensical though. What it really shows is that the steep increase in λ_2 leaves the system unable to cope regardless of the decisions made.

Increasing number of servers C_1 and C_2: Figure 14 shows a run of asymmetric replicator dynamics with a change in the number of servers of the hospitals. The number of servers are increased from $C_1 = 3, C_2 = 2$ to $C_1 = 4, C_2 = 3$.

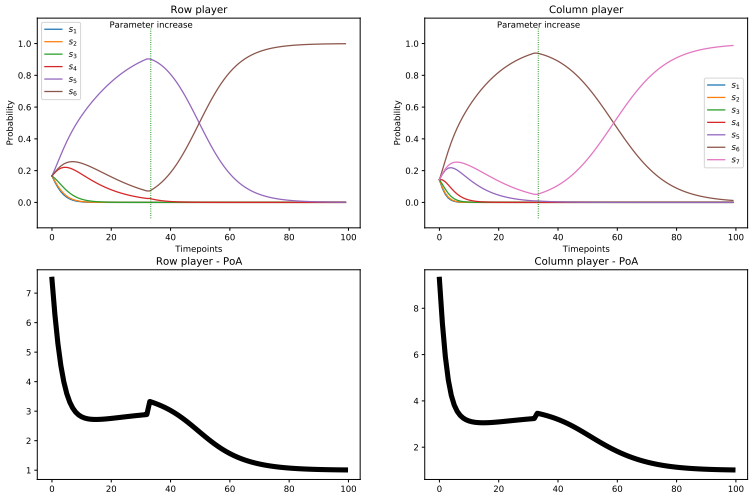

Figure 14: The strategies played when running asymmetric replicator dynamics along with the compartmentalised price of anarchy of the blocking time at each iteration of the learning algorithm. After a number of iterations the number of servers for both systems are increased by one.

In this case, both the behaviour as well as the price of anarchy change. The players change their strategies from $T_1 = 5, T_2 = 6$ to $T_1 = 6, T_2 = 7$ and the PoA of the blocking time goes down. By adding more resources to the models they are able to increase their efficiency. Although this is a good way to escape such inefficiencies, it might not always be cost efficient.

Incentivising players: From figures 13 and 14 it can be seen that we can change some parameters of the model to make it more efficient. The approach used on figure 15 is slightly different than the previous cases. Once the played strategies in asymmetric replicator dynamics strategies converge the payoff matrices of the two are scaled in such a way so that the utilities of the selected strategy are penalised. This corresponds to a precise policy change where more societally beneficial behaviours are incentivised.

Matrices A and B represent the original payoff matrices while matrices $\tilde{A}$ and $\tilde{B}$ represent the incentivised payoff matrices. It can be observed that matrix $\tilde{A}$ is a scaled version of matrix A only on the row that is most frequently played and similarly matrix $\tilde{B}$ of matrix B only on the column that is most frequently played (matrix A: row 5, matrix B: column 6, see figure 15). Note that the values of the payoff matrices only differ on the fourth digit which is the raw utility of the strategies played.

$$A = \begin{bmatrix} 0.9995052 & 0.9995052 & 0.9995052 & 0.9995052 & 0.9995052 & 0.9995052 & 0.9995052 \\ 0.99954989 & 0.99954978 & 0.99954961 & 0.99954924 & 0.99954845 & 0.9995466 & 0.999539 \\ 0.99968233 & 0.99968194 & 0.99968151 & 0.99968067 & 0.99967874 & 0.9996734 & 0.999649 \\ 0.99990299 & 0.99990245 & 0.99990189 & 0.99990081 & 0.99989832 & 0.9998909 & 0.9998517 \\ 0.99999996 & 0.99999994 & 0.99999992 & 0.99999988 & 0.99999973 & 0.9999989 & 0.9999859 \\ 0.9998773 & 0.99987995 & 0.99988236 & 0.99988643 & 0.99989417 & 0.9999126 & 0.9999712 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.99917127 & 0.99925823 & 0.99946188 & 0.99968499 & 0.9998942 & 1.0 & 0.99982128 \\ 0.99917127 & 0.99925479 & 0.99945638 & 0.99968051 & 0.99989153 & 0.99999997 & 0.9998333 \\ 0.99917127 & 0.99924532 & 0.99943793 & 0.99966451 & 0.99988286 & 0.99999966 & 0.99985269 \\ 0.99917127 & 0.99924146 & 0.99942878 & 0.99965484 & 0.99987667 & 0.99999921 & 0.99986701 \\ 0.99917127 & 0.9992342 & 0.99941013 & 0.99963286 & 0.99986077 & 0.9999972 & 0.9998958 \\ 0.99917127 & 0.99921279 & 0.99934961 & 0.99954933 & 0.99978407 & 0.99997106 & 0.99997276 \end{bmatrix}$$

$$\tilde{A} = \begin{bmatrix} 0.99950518 & 0.99950518 & 0.99950518 & 0.99950518 & 0.99950518 & 0.99950518 & 0.99950518 \\ 0.99954989 & 0.99954978 & 0.99954961 & 0.99954924 & 0.99954845 & 0.99954656 & 0.99953879 \\ 0.99968233 & 0.99968194 & 0.99968151 & 0.99968067 & 0.99967874 & 0.99967339 & 0.9996492 \\ 0.99990299 & 0.99990245 & 0.99990189 & 0.99990081 & 0.99989832 & 0.9998909 & 0.99985171 \\ 0.99969996 & 0.99969994 & 0.99969992 & 0.99969988 & 0.99969973 & 0.99969895 & 0.99968586 \\ 0.9998773 & 0.99987995 & 0.99988236 & 0.99988643 & 0.99989417 & 0.99991264 & 0.99997122 \end{bmatrix}$$

$$\tilde{B} = \begin{bmatrix} 0.99917127 & 0.99925823 & 0.99946188 & 0.99968499 & 0.9998942 & 0.9997 & 0.99982128 \\ 0.99917127 & 0.99925479 & 0.99945638 & 0.99968051 & 0.99989153 & 0.99969997 & 0.9998333 \\ 0.99917127 & 0.99924532 & 0.99943793 & 0.99966451 & 0.99988286 & 0.99969966 & 0.99985269 \\ 0.99917127 & 0.99924146 & 0.99942878 & 0.99965484 & 0.99987667 & 0.99969921 & 0.99986701 \\ 0.99917127 & 0.9992342 & 0.99941013 & 0.99963286 & 0.99986077 & 0.9996972 & 0.9998958 \\ 0.99917127 & 0.99921279 & 0.99934961 & 0.99954933 & 0.99978407 & 0.99967106 & 0.99997276 \end{bmatrix}$$

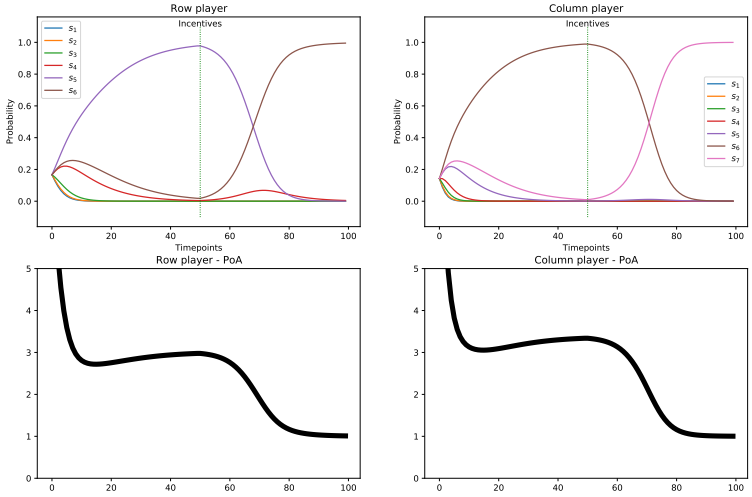

Figure 15: The strategies played when running asymmetric replicator dynamics along with the compartmentalised price of anarchy of the blocking time at each iteration of the learning algorithm. After a number of iterations the most dominant strategy is being penalised.

Figure 15 shows that players start playing strategies $T_1 = 5$ and $T_2 = 6$ and mid-run of the learning algorithm a penalty is applied to these strategies on the payoff matrix. By incentivising the players in such a way the players change their strategies to $T_1 = 6$ and $T_2 = 7$, and thus ambulance patients are accepted in the ED more often. Hence, the PoA for both EDs is decreased, meaning that the whole system is more efficient in terms of the blocking time.

6 Conclusion

References

[1] Elvio Accinelli and Edgar J Sánchez Carrera. Evolutionarily stable strategies and replicator dynamics in asymmetric two-population games. In *Dynamics, Games and Science I*, pages 25–35. Springer, 2011.

[2] Vivienne Aitken. The red cross drafted into scotland's biggest hospital as ambulances queue up. *Daily Record*, 2021.

[3] Mohamed Akkouchi. On the convolution of exponential distributions. *J. Chungcheong Math. Soc*, 21(4):501–510, 2008.

[4] Dietmar Berwanger and Laurent Doyen. On the power of imperfect information. *Leibniz International Proceedings in Informatics, LIPIcs*, 2:73–82, 2008.

[5] Richard P Brent. Algorithms for minimization without derivatives, chap. 4, 1973.

[6] Owain Clarke. A&e queues mean wales ambulances can't take 999 calls. *BBC Wales*, 2021.

[7] Julie Crouch. Thousands of hours lost as ambulance idle outside hospitals in derbyshire. *Staffordshire Live*, 2021.

[8] Stefano Favaro and Stephen G Walker. On the distribution of sums of independent exponential random variables via wilks' integral representation. *Acta applicandae mathematicae*, 109(3):1035–1042, 2010.

[9] Drew Fudenberg, Fudenberg Drew, David K Levine, and David K Levine. *The theory of learning in games*, volume 2. MIT press, 1998.

[10] John G Kemeny and J Laurie Snell. *Markov chains*, volume 6. Springer-Verlag, New York, 1976.

[11] Smaili Khaled, Therrar Kadri, and Seifedine Kadry. Hypoexponential distribution with different parameters. *Journal of Applied Mathematics*, 4:624–631, 04 2013.

[12] David M Kreps. Nash equilibrium. In *Game Theory*, pages 167–177. Springer, 1989.

[13] Benjamin Legros and Oualid Jouini. A linear algebraic approach for the computation of sums of erlang random variables. *Applied Mathematical Modelling*, 39(16):4971–4977, 2015.

[14] C. E. Lemke and J. T. Howson, Jr. Equilibrium points of bimatrix games. *Journal of the Society for Industrial and Applied Mathematics*, 12(2):413–423, 1964.

[15] Michael Maschler, Eilon Solan, and Eilon Zamir. *Game Theory*. Cambridge University Press, 2013.

[16] Ben McAdams. Caldicot girl waited nine hours for ambulance after fall. *South Wales Argus*, 2021.

[17] Roger B Myerson. Refinements of the nash equilibrium concept. *International journal of game theory*, 7(2):73–80, 1978.

[18] Michalis Panayides. 11michalis11/ambulancedecisiongame: v0.0.2 , September 2021.

[19] Tim Roughgarden. *Selfish routing and the price of anarchy*. MIT press, 2005.

[20] John Maynard Smith. Evolutionary game theory. *Physica D: Nonlinear Phenomena*, 22(1-3):43–49, 1986.

[21] William J. Stewart. *Probability, Markov Chains, Queues, and Simulation*. Princeton University Press, mar 2019.

[22] The Ciw library developers. Ciwpython/ciw: v2.1.3, oct 2020.

[23] The Nashpy project developers. Nashpy: v0.0.25, August 2021.

[24] James Thomas. Ambulances queue at hereford hospital as nhs pressure mounts. *Hereford Times*, 2021.

Appendices

A Discrete Event Simulation

For the purposes of this study, a discrete event simulation (DES) model was constructed to support the Markov chain version described in section 3. The queueing model was built in python using the Ciw library [22].

The constructed model simulates a queueing system with two waiting spaces and two types of individuals. The expected behaviour of the nodes in Ciw have been modified such that individuals moving from waiting zone 2 into waiting zone 1 get blocked if there are more than T individuals in waiting zone 1.

The same performance measures described in sections 3.1, 3.2 and 3.3 can also be calculated using the DES model. The simulation can be ran a number of times to eliminate stochasticity and the outcomes of the two methods can be directly comparable.

A.1 Tutorial: building the DES model

The DES model is constructed in a generic way that so that it can be used for any queueing system with two waiting spaces and two types of individuals. For instance, consider a queueing system with the following parameters, as described in section 3:

- $\lambda_1 = 2$
- $\lambda_2 = 3$
- $\mu = 1$
- $C = 6$
- $T = 10$
- $N = 20$
- $M = 10$

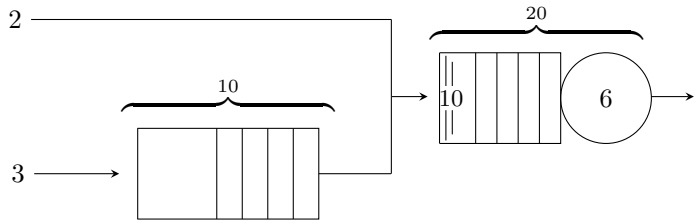

Figure 16: A diagrammatic representation of the queueing model example

This model will be studied by using `ambulance_game` [18]. Install the created library in your python environment, by running the following command in the command line:

```
$ python -m pip install ambulance_game
```

Having installed the package, the following code can be used to simulate the queueing system defined earlier and get all the data records for a single run.

```
>>> import ambulance_game as abg
>>> Simulation = abg.simulation.simulate_model(
...     lambda_1=3,
...     lambda_2=2,
...     mu=1,
...     num_of_servers=6,
...     threshold=10,
...     system_capacity=20,
...     buffer_capacity=10,
...     seed_num=0,
... )
>>> all_records = Simulation.get_all_records()
>>> all_records[3]
Record(id_number=1, customer_class=0, node=2, arrival_date=0
.4728763843239206, waiting_time=0.0, service_start_date=0
.4728763843239206, service_time=0.5457131455415929,
service_end_date=1.0185895298655134, time_blocked=0.0,
exit_date=1.0185895298655134, destination=-1,
queue_size_at_arrival=0, queue_size_at_departure=4)
```

The above block code outputs the fourth individual record from the simulation object. The simulation object can be used to view every event that occurred

in the simulation The data records can then be used to get overall performance measures about the constructed queueing model. The overall waiting time that individuals wait in waiting zone 1 can be acquired by running:

```
>>> import numpy as np
>>> mean_wait = np.mean(
...     [record.waiting_time for record in all_records if
record.node == 2]
... )
>>> mean_wait
0.3608431242229529
```

This value is the average waiting time of all the customers in the system for a single run. By nature, discrete event simulation can output different results for different runs of the same set of parameters. This stochasticity can be reduced by running the simulation multiple times and then getting the mean waiting time from all the runs.

```
>>> all_simulations = abg.simulation.get_multiple_runs_results(
...     lambda_1=3,
...     lambda_2=2,
...     mu=1,
...     num_of_servers=6,
...     threshold=10,
...     system_capacity=20,
...     buffer_capacity=10,
...     seed_num=0,
...     num_of_trials=10,
... )
>>> mean_wait = np.mean(
...     [np.mean(w.waiting_times) for w in all_simulations]
... )
>>> mean_wait
0.3566390561071839
```

A.2 How-to guide

A.2.1 How to install:

The package can be installed by either running:

```
$ python -m pip install ambulance_game
```

in the command line or via the instructions provided in the GitHub repository.

A.2.2 How to simulate the model:

The required arguments that need to be passed to the `simulate_model()` function are the following:

- lambda_1 (λ_1): The arrival rate of class 1 individuals.
- lambda_2 (λ_2): The arrival rate of class 2 individuals.

- mu (μ): The service rate of the servers.
- num_of_servers (C): The number of servers in the system.
- threshold (T): The threshold that indicates when to start blocking class 2 individuals.

To get the simulation object with all the data records, the following code can be used:

```
>>> import ambulance_game as abg
>>> Simulation = abg.simulation.simulate_model(
...     lambda_1=3,
...     lambda_2=2,
...     mu=1,
...     num_of_servers=6,
...     threshold=10,
...     seed_num=0,
... )
>>> Simulation.get_all_records()[4]
Record(id_number=2, customer_class=0, node=2, arrival_date=0
.5727571550618586, waiting_time=0.0, service_start_date=0
.5727571550618586, service_time=0.7159547497671506,
service_end_date=1.2887119048290092, time_blocked=0.0,
exit_date=1.2887119048290092, destination=-1,
queue_size_at_arrival=1, queue_size_at_departure=3)
```

Additional arguments that can be passed to the function are:

- *system_capacity* (N): The maximum number of individuals in waiting zone 2.
- *buffer_capacity* M: The maximum number of individuals in waiting zone 1.
- *seed_num*: The seed number for the random number generator.
- *runtime*: How long to run the simulation for.

A.2.3 How to get the performance measures for a single run:

From a single run of the simulation the data records can be used to get the average for certain performance measures. The following code can be used to get the mean waiting time, blocking time, service time and the proportion of individuals within target.

```
>>> records = Simulation.get_all_records()
>>> mean_wait = np.mean(
...     [w.waiting_time for w in records]
... )
>>> mean_wait
0.23845862661827116

>>> mean_block = np.mean(
...     [b.time_blocked for b in records]
```

```
... )
>>> mean_block
0.08501727452006658

>>> mean_service = np.mean(
...     [s.service_time for s in records]
... )
>>> mean_service
0.7102610863960119

>>> target = 1
>>> proportion_within_target = np.mean(
...     [w.waiting_time <= target for w in records]
... )
>>> proportion_within_target
0.9387470071827614
```

A.2.4 How to get the average performance measures:

To reduce the effects of stochasticity in the simulation, the simulation can be run numerous times and get the average performance measures out of all the runs.

```
>>> import numpy as np
>>> import ambulance_game as abg
>>> all_simulations = abg.simulation.get_multiple_runs_results(
...     lambda_1=3,
...     lambda_2=2,
...     mu=1,
...     num_of_servers=6,
...     threshold=10,
...     system_capacity=20,
...     buffer_capacity=10,
...     seed_num=0,
...     runtime=2000,
...     num_of_trials=10,
...     target=1,
... )
>>> mean_wait = np.mean([
...     np.mean(w.waiting_times) for w in all_simulations
... ])
>>> mean_wait
0.35585979549204577

>>> mean_service = np.mean([
...     np.mean(s.service_times) for s in all_simulations
... ])
>>> mean_service
1.002184850213415

>>> mean_block = np.mean([
...     np.mean(b.blocking_times) for b in all_simulations
... ])
>>> mean_block
0.3976966024549059
```

```
>>> np.mean([p.proportion_within_target for p in
all_simulations])
0.45785790578122043
```

A.2.5 How to get the steady state probabilities vector π:

To get the steady state probabilities of the model based on the simulation the following code can be used:

```
>>> import numpy as np
>>> import ambulance_game as abg
>>> simulation_object = abg.simulation.simulate_model(
...     lambda_1=1,
...     lambda_2=2,
...     mu=2,
...     num_of_servers=2,
...     threshold=3,
...     system_capacity=4,
...     buffer_capacity=2,
...     seed_num=0,
...     runtime=2000,
... )
>>> probs = abg.simulation.get_simulated_state_probabilities(
...    simulation_object=simulation_object,
... )
>>> np.round(probs, decimals=3)
array([[0.166, 0.266, 0.192, 0.147, 0.025],
       [  nan,   nan,   nan, 0.094, 0.024],
       [  nan,   nan,   nan, 0.058, 0.027]])

>>> total = np.nansum(probs)
>>> np.round(total, decimals=5)
1.0
```

Similarly to get the average steady state probabilities over multiple runs, one can use:

```
>>> import numpy as np
>>> import ambulance_game as abg
>>> probs =
abg.simulation.get_average_simulated_state_probabilities(
...     lambda_1=1,
...     lambda_2=2,
...     mu=2,
...     num_of_servers=2,
...     threshold=3,
...     system_capacity=4,
...     buffer_capacity=2,
...     seed_num=0,
...     runtime=2000,
...     num_of_trials=10,
... )
>>> np.round(probs, decimals=3)
array([[0.18 , 0.267, 0.197, 0.144, 0.024],
       [  nan,   nan,   nan, 0.085, 0.022],
       [  nan,   nan,   nan, 0.054, 0.026]])
```

```
>>> total = np.nansum(probs)
>>> np.round(total, decimals=5)
1.0
```

A.2.6 How to get the optimal distribution of class 2 individuals among 2 queuing models:

In the scenario where there are two queueing models and a service that distributes individuals to the models, (i.e. the scenario described in this paper) the simulation can be used to decide what proportion of individuals to send to each the model. Note that the output of the function shows the value of p_1, the proportion of class 2 individuals to be sent to queueing model 1.

```
>>> import ambulance_game as abg
>>> abg.simulation.calculate_class_2_individuals_best_response(
...     lambda_2=4,
...     lambda_1_1=1,
...     lambda_1_2=1,
...     mu_1=4,
...     mu_2=3,
...     num_of_servers_1=2,
...     num_of_servers_2=2,
...     threshold_1=3,
...     threshold_2=3,
...     system_capacity_1=3,
...     system_capacity_2=3,
...     buffer_capacity_1=2,
...     buffer_capacity_2=2,
...     seed_num_1=0,
...     seed_num_2=0,
...     num_of_trials=3,
...     warm_up_time=100,
...     runtime=1000,
... )
0.6343260586929469
```

A.3 Reference

The primary tool that was used in the construction of the discrete event simulation model was the python library `ciw`. See Ciw's documentation for a more detailed explanation of how it works and what are its capabilities [22].

Find below a detailed list of the functions that were created for the simulation model:

- `build_model`: *Builds a ciw object that represents a model of a queuing network with two waiting spaces.*
- `build_custom_node`: *Build a custom node to replace the default ciw.Node. Inherits from the original ciw.Node class and replaces two methods.*
- `simulate_model`: *Simulate the model by using the custom node and returning the simulation object*

- `extract_times_from_records`: *Get the required times (waiting, service, blocking) out of ciw's records where all individuals are treated the same way*
- `extract_times_from_individuals`: *Extract waiting times and service times for all individuals and proceed to extract blocking times for only class 2 individuals*
- `get_list_of_results`: *Modify the outputs so that they are returned in a list format where it is sometimes easier to be used by other functions.*
- `get_multiple_runs_results`: *Get the waiting times, service times and blocking times for multiple runs of the simulation*
- `extract_total_individuals_and_the_ones_within_target_for_both_classes`: *Extract the total number of individuals that pass through the model and the total number of individuals that exit the model within the given target.*
- `get_mean_proportion_of_individuals_within_target_for_multiple_runs`: *Get the average proportion of individuals within target by running the simulation multiple times*
- `get_simulated_state_probabilities`: *Calculates the vector π in a dictionary format or an array format*
- `get_average_simulated_state_probabilities`: *This function runs get_simulated_state_probabilities for multiple iterations to eliminate any stochasticity from the results*
- `get_mean_blocking_difference_between_two_systems`: *Given a predefined proportion of class's 2 arrival rate calculate the mean difference between blocking times of two systems with a given set of parameters*
- `calculate_class_2_individuals_best_response` *Obtains the optimal distribution of class 2 individuals such that the blocking times in the two systems are identical and thus minimised*

A.4 Explanation

Based on Ciw's functionality the simulation model stores all data records in a Record object. For every event that takes place a record is created with all the relevant information. For this specific library, the records that are stored, along with the range of values that they can take are as follows:

- **id_number** $\in R$.
- **customer_class** $= 0$
- **node** $= \{0, 1, 2, -1\}$
- **arrival_date** $\in R^+$
- **waiting_time** $\in R^+$
- **service_start_date** $\in R^+$
- **service_time** $\in R^+$
- **service_end_time** $\in R^+$

- **time_blocked** $\in R^+$
- **exit_date** $\in R^+$
- **destination** $= \{1, 2, -1\}$
- **queue_size_at_arrival** $\in N$
- **queue_size_at_departure** $\in N$

B Mean waiting time

The recursive formula described here is the origin of the closed-form formula described in section 3.1.

To calculate the mean waiting time one must first identify the set of states (u, v) where a wait will occur. For this particular Markov chain, this points to all states that satisfy $v > C$ i.e. all states where the number of individuals in the service centre exceed the number of servers. The set of such states is defined as *waiting states* and can be denoted as a subset of all the states, where:

$$S_w = \{(u, v) \in S \mid v > C\} \tag{40}$$

Additionally, there are certain states in the model where arrivals cannot occur. A type 1 individual cannot arrive whenever the model is at any state (u, N) for all u where N is the system capacity. Equivalently, a type 2 individual cannot arrive in the model when the model is at any state (M, v) for all $v \geq T$. Therefore the set of all such states that an arrival may occur are defined as *accepting states* and are denoted as:

$$S_A^{(1)} = \{(u, v) \in S \mid v < N\} \tag{12 revisited}$$

$$S_A^{(2)} = \begin{cases} \{(u, v) \in S \mid u < M\} & \textbf{if } T \leq N \\ \{(u, v) \in S \mid v < N\} & \textbf{otherwise} \end{cases} \tag{13 revisited}$$

Moreover, another element that needs to be considered is the expected waiting time In order to do so a variation of the Markov model has to be considered where when the individual arrives at any of the states of the model no more arrivals can occur after that.

Thus, one may acquire the desired time by calculating the inverse of the sum of the out-flow rate of that state. Since arrivals are ignored though the only way to exit the state will only be via a service. In essence this notion can be expressed as:

$$c^{(1)}(u, v) = \begin{cases} 0, & \textbf{if } u > 0 \textbf{ and } v = T \\ \frac{1}{\min(v, C)\mu}, & \textbf{otherwise} \end{cases} \tag{41}$$

Now, like in the type 1 individuals case, the sojourn time is needed. For type 2 individuals whenever individuals are at any row apart from the first one they automatically get a wait time of 0 since they are essentially blocked.

$$c^{(2)}(u,v) = \begin{cases} 0, & \textbf{if } u > 0 \\ \frac{1}{\min(v,C)\mu}, & \textbf{otherwise} \end{cases} \tag{42}$$

Note that whenever any type 1 individual is at a state (u, v) where $u > 0$ and $v = T$ (i.e. all states $(1, T), (2, T) \ldots, (M, T)$) the sojourn time is set to 0. This is done to capture the trip thorough the Markov chain from the perspective of individuals. Meaning that they will visit all states of the threshold column but only the one in the first row will return a non-zero sojourn time. Thus, the expected waiting time of type 1 and type 2 individuals when upon arriving at state (u, v) can be given by the following recursive formulas:

$$w^{(1)}(u,v) = \begin{cases} 0, & \textbf{if } (u,v) \notin S_w \\ c^{(1)}(u,v) + w^{(1)}(u-1,v), & \textbf{if } u > 0 \textbf{ and } v = T \\ c^{(1)}(u,v) + w^{(1)}(u,v-1), & \textbf{otherwise} \end{cases} \tag{43}$$

$$w^{(2)}(u,v) = \begin{cases} 0, & \textbf{if } (u,v) \notin S_w \\ c^{(2)}(u,v) + w^{(2)}(u-1,v), & \textbf{if } u > 0 \textbf{ and } v = T \\ c^{(2)}(u,v) + w^{(2)}(u,v-1), & \textbf{otherwise} \end{cases} \tag{44}$$

Finally, the mean waiting time can be calculated by summing over all expected waiting times of accepting states multiplied by the probability of being at that state and dividing by the probability of being in any accepting state.

$$W^{(1)} = \frac{\sum_{(u,v)\in S_A^{(1)}} w^{(1)}(u,v)\pi_{(u,v)}}{\sum_{(u,v)\in S_A^{(1)}} \pi_{(u,v)}} \tag{45}$$

$$W^{(2)} = \frac{\sum_{(u,v)\in S_A^{(2)}} w^{(2)}(u,v)\pi_{(u,v)}}{\sum_{(u,v)\in S_A^{(2)}} \pi_{(u,v)}} \tag{46}$$

C Mean blocking time

The set of states where individuals can be blocked is defined as:

$$S_b = \{(u,v) \in S \mid u > 0\} \tag{17 revisited}$$

The mean sojourn time for each state is given by the inverse of the out-flow of that state [21]. However, whenever a type 2 individual arrives at the system, no subsequent arrival of another type 2 individual can affect its pathway or total time in the system. Therefore, looking at the mean time in the system from the perspective of an individual of the second type, all such type 2 arrivals need to be ignored. Note here that this is not the case for individuals of the first type.

Whenever a type 2 individual is blocked and a type 1 individual arrives the type 2 individuals will stay blocked for some additional amount of time. Thus, the mean time that a type 2 individual spends at each state is given by:

$$c(u,v) = \begin{cases} \frac{1}{\min(v,C)\mu}, & \text{if } v = N \\ \frac{1}{\lambda_1 + \min(v,C)\mu}, & \text{otherwise} \end{cases} \qquad \text{(18 revisited)}$$

In equation (18), both service completions and type 1 arrivals are considered. Thus, from a blocked individual's perspective whenever the system moves from one state (u, v) to another state it can either:

- be because of a service being completed: we will denote the probability of this happening by $p_s(u, v)$.
- be because of an arrival of an individual of type 1: denoting such probability by $p_a(u, v)$.

The probabilities are given by:

$$p_s(u,v) = \frac{\min(v,C)\mu}{\lambda_1 + \min(v,C)\mu}, \qquad p_a(u,v) = \frac{\lambda_1}{\lambda_1 + \min(v,C)\mu} \qquad \text{(19 revisited)}$$

Having defined $c(u, v)$ and S_b a formula for the blocking time that is expected to occur at each state can be given by:

$$b(u,v) = \begin{cases} 0, & \textbf{if } (u,v) \notin S_b \\ c(u,v) + b(u-1,v), & \textbf{if } v = N = T \\ c(u,v) + b(u,v-1), & \textbf{if } v = N \neq T \\ c(u,v) + p_s(u,v)b(u-1,v) + p_a(u,v)b(u,v+1), & \textbf{if } u > 0 \textbf{ and } v = T \\ c(u,v) + p_s(u,v)b(u,v-1) + p_a(u,v)b(u,v+1), & \textbf{otherwise} \end{cases} \qquad \text{(16 revisited)}$$

A direct approach will be used to solve this equation here. By enumerating all equations of (16) for all states (u, v) that belong in S_b a system of linear equations arises where the unknown variables are all the $b(u, v)$ terms. Note here that these equations correspond to all blocking states as defined in (17). Equations that correspond to non-blocking states have a value of 0 as defined in (16) The general form of the equation in terms of C, T, N and M is given by:

$$
\begin{aligned}
b(1,T) &= c(1,T) + p_a b(1,T+1) && (47)\\
b(1,T+1) &= c(1,T+1) + p_s b(1,T) + p_a b(1,T+1) && (48)\\
b(1,T+2) &= c(1,T+2) + p_s b(1,T+1) + p_a b(1,T+3) && (49)\\
&\vdots\\
b(1,N) &= c(1,N) + b(1,N-1) && (50)\\
b(2,T) &= c(2,T) + p_s b(1,T) + p_a b(2,T+1) && (51)\\
b(2,T+1) &= c(2,T+1) + p_s b(2,T) + p_a b(2,T+2) && (52)\\
&\vdots\\
b(M-1,N) &= c(M,N-1) + b(M,N-1) && (53)\\
b(M,T) &= c(T,N) + p_s b(T-1,N) + p_a b(T,N+1) && (54)\\
&\vdots\\
b(M,N) &= c(M,N) + b(M,N-1) && (55)
\end{aligned}
$$

The equivalent matrix notation of the linear system of equations (47) - (55) is given by $Zx = y$, where:

$$
Z = \begin{pmatrix}
-1 & p_a & 0 & \dots & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0\\
p_s & -1 & p_a & \dots & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0\\
0 & p_s & -1 & \dots & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0\\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots\\
0 & 0 & 0 & \dots & 1 & -1 & 0 & 0 & 0 & \dots & 0 & 0\\
p_s & 0 & 0 & \dots & 0 & 0 & -1 & p_a & 0 & \dots & 0 & 0\\
0 & 0 & 0 & \dots & 0 & 0 & p_s & -1 & p_a & \dots & 0 & 0\\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots\\
0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & \dots & 1 & -1
\end{pmatrix}, x = \begin{pmatrix}
b(1,T)\\ b(1,T+1)\\ b(1,T+2)\\ \vdots\\ b(1,N)\\ b(2,T)\\ b(2,T+1)\\ \vdots\\ b(M,N)
\end{pmatrix}, y = \begin{pmatrix}
-c(1,T)\\ -c(1,T+1)\\ -c(1,T+2)\\ \vdots\\ -c(1,N)\\ -c(2,T)\\ -c(2,T+1)\\ \vdots\\ -c(M,N)
\end{pmatrix}
\qquad \text{(21 revisited)}
$$

The elements of the matrix Z can be acquired using Z_{ij} defined in equation (20) where i and j are states $(u_i, v_i), (u_j, v_j) \in S_b$ (17).

$$
Z_{ij} = \begin{cases}
p_a, & \textbf{if } j = i+1 \textbf{ and } v_i \neq N\\
p_s, & \textbf{if } j = i-1 \textbf{ and } v_i \neq N, v_i \neq T\\
 & \textbf{or } j = i - N + T \textbf{ and } u_i \geq 2,\ v_i = T\\
1, & \textbf{if } j = i-1 \textbf{ and } v_i = N\\
-1, & \textbf{if } i = j\\
0, & \textbf{otherwise}
\end{cases}
\qquad \text{(20 revisited)}
$$

Thus, having calculated the mean blocking time for all blocking states $b(u, v)$, they can be combined together in a formula. The resultant formula for the mean blocking time is given by:

$$B = \frac{\sum_{(u,v)\in S_A} \pi_{(u,v)}\ b(u,v)}{\sum_{(u,v)\in S_A} \pi_{(u,v)}} \qquad \text{(15 revisited)}$$

To illustrate how the described formula works consider a Markov model where $C = 2, T = 2, N = 4, M = 2$ (figure 17). The equations that correspond to such a model are shown in (56)-(61) and their equivalent matrix notation form is shown in (62).

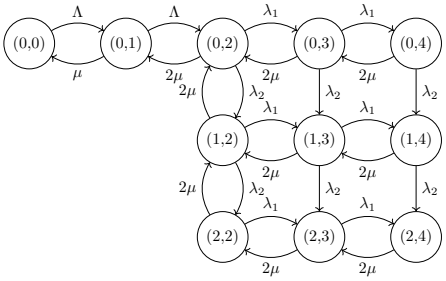

Figure 17: Example of Markov chain with $C = 2, T = 2, N = 4, M = 2$

$$b(1,2) = c(1,2) + p_a b(1,3) \quad (56)$$
$$b(1,3) = c(1,3) + p_s b(1,2) + p_a b(1,4) \quad (57)$$
$$b(1,4) = c(1,4) + b(1,3) \quad (58)$$
$$b(2,2) = c(2,2) + p_s b(1,2) + p_a b(2,3) \quad (59)$$
$$b(2,3) = c(2,3) + p_s b(2,2) + p_a b(1,4) \quad (60)$$
$$b(2,4) = c(2,4) + b(2,3) \quad (61)$$

$$Z = \begin{pmatrix} -1 & p_a & 0 & 0 & 0 & 0 \\ p_s & -1 & p_a & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ p_s & 0 & 0 & -1 & p_a & 0 \\ 0 & 0 & 0 & p_s & -1 & p_a \\ 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix}, x = \begin{pmatrix} b(1,2) \\ b(1,3) \\ b(1,4) \\ b(2,2) \\ b(2,3) \\ b(2,4) \end{pmatrix}, y = \begin{pmatrix} -c(1,2) \\ -c(1,3) \\ -c(1,4) \\ -c(2,2) \\ -c(2,3) \\ -c(2,4) \end{pmatrix} \quad (62)$$

D Mean blocking time

In order to consider such measure though one would need to obtain the distribution of time in the system for all individuals. The complexity of such task is caused by the fact that different individuals arrive at different states of the Markov model. Consider the case when an arrival occurs when the model is at a specific state.

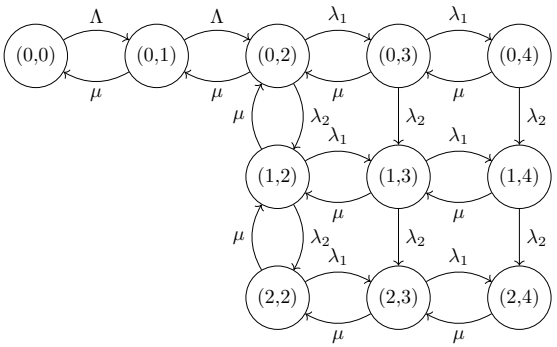

Figure 18: Example Markov model $C = 1, T = 2, N = 4, M = 2$

Time distribution at specific state (1 server): Consider the Markov model of figure 18 with one server and a threshold of two individuals. Assume that an individual of the first type arrives when the model is at state $(0, 3)$, thus forcing the model to move to state $(0, 4)$. The distribution of the time needed for the specified individual to exit the system from state $(0, 4)$ is given by the sum of exponentially distributed random variables with the same parameter μ. The sum of such random variables forms an Erlang distribution which is defined by the number of random variables that are added and their exponential parameter. Note here that these random variables represent the individual's pathway from the perspective of the individual. Thus, X_i represents the time that it takes to move from the i^{th} position of the queue to the $(i-1)^{\text{th}}$ position (i.e. for someone in front of them to finish their service) and X_0 is the time it takes to move from having a service to exiting the system.

$$\begin{aligned}
(0,4) \Rightarrow \quad & X_3 \sim Exp(\mu) \\
(0,3) \Rightarrow \quad & X_2 \sim Exp(\mu) \\
(0,2) \Rightarrow \quad & X_1 \sim Exp(\mu) \\
(0,1) \Rightarrow \quad & X_0 \sim Exp(\mu) \\
S = X_3 + X_2 + X_1 + X_0 &= Erlang(4, \mu)
\end{aligned} \tag{63}$$

Thus, the waiting and service time of an individual in the model of figure 18 can be captured by an erlang distributed random variable. The general CDF of the erlang distribution $Erlang(k, \mu)$ is given by:

$$P(S < t) = 1 - \sum_{i=0}^{k-1} \frac{1}{i!} e^{-\mu t} (\mu t)^i \tag{64}$$

Unfortunately, the erlang distribution can only be used for the sum of identically distributed random variables from the exponential distribution. Therefore, this approach cannot be used when one of the random variables has a different parameter than the others. In fact the only case where it can be used is only

when the number of servers are $C = 1$, or when an individual arrives and goes straight to service (i.e. when there is no other individual waiting and there is an empty server).

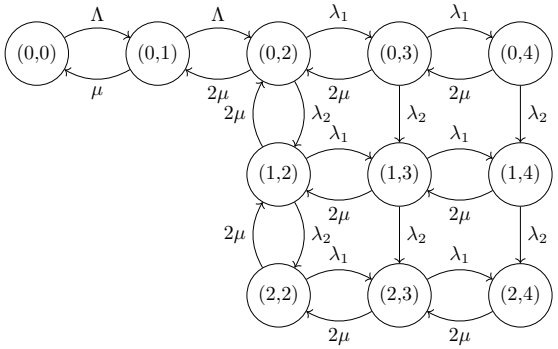

Figure 19: Example Markov model $C = 2, T = 2, N = 4, M = 2$

Time distribution at a state (multiple servers): Figure 19 represents the same Markov model as figure 18 with the only exception that there are 2 servers here. By applying the same logic, assuming that an individual arrives at state $(0, 4)$, the sum of the following random variables arises.

$$\begin{aligned} (0,4) \Rightarrow &\quad X_2 \sim Exp(2\mu) \\ (0,3) \Rightarrow &\quad X_1 \sim Exp(2\mu) \\ (0,2) \Rightarrow &\quad X_0 \sim Exp(\mu) \end{aligned} \tag{65}$$

Since these exponentially distributed random variables do not share the same parameter, an erlang distribution cannot be used. In fact, the problem can now be viewed either as the sum of exponentially distributed random variables with different parameters or as the sum of erlang distributed random variables. The sum of erlang distributed random variables is said to follow the hypoexponential distribution. The hypoexponential distribution is defined with two vectors of size equal to the number of Erlang random variables [3], [11]. The vector $\vec{r}$ contains all the k-values of the erlang distributions and $\vec{\lambda}$ is a vector of the distinct parameters as illustrated in equation (66).

$$\left.\begin{array}{l} Erlang(k_1, \lambda_1) \\ Erlang(k_2, \lambda_2) \\ \quad\vdots \\ Erlang(k_n, \lambda_n) \end{array}\right\} Hypo(\underbrace{(k_1, k_2, \ldots k_n)}_{\vec{k}}, \underbrace{(\lambda_1, \lambda_2, \ldots \lambda_n)}_{\vec{\lambda}}) \tag{66}$$

Equivalently, for this particular example:

$$\left.\begin{array}{l} X_2 \sim Exp(2\mu) \\ X_1 \sim Exp(2\mu) \end{array}\right\} X_1+X_2=S_1 \sim Erlang(2,2\mu) \left.\begin{array}{l} \\ \\ X_0 \sim Exp(\mu) \Rightarrow \end{array}\right\} S_1+S_2=H \sim Hypo((2,1),(2\mu,\mu)) \qquad X_0=S_2 \sim Erlang(1,\mu) \tag{67}$$

Therefore, the CDF of this distribution can be used to get the probability of the time in spent in the system being less than a given target. The general CDF of the hypoexponential distribution $Hypo(\vec{r}, \vec{\lambda})$, is given by the following expression [8]:

$$\begin{aligned} P(H < t) &= 1 - \left(\prod_{j=1}^{|\vec{r}|} \lambda_j^{r_j} \right) \sum_{k=1}^{|\vec{r}|} \sum_{l=1}^{r_k} \frac{\Psi_{k,l}(-\lambda_k) t^{r_k - l} e^{-\lambda_k t}}{(r_k - l)!(l-1)!} \\ \textbf{where} \qquad \Psi_{k,l}(t) &= -\frac{\partial^{l-1}}{\partial t^{l-1}} \left(\prod_{j=0, j \neq k}^{|\vec{r}|} (\lambda_j + t)^{-r_j} \right) \\ \textbf{and} \qquad & \lambda_0 = 0, r_0 = 1 \end{aligned} \tag{68}$$

The computation of the derivative makes equation (68) computationally expensive. In [13] an alternative linear version of that CDF is explored via matrix analysis, and is given by the following formula:

$$\begin{aligned} F(x) =& 1 - \sum_{k=1}^{n} \sum_{l=0}^{k-1} (-1)^{k-1} \binom{n}{k} \binom{k-1}{l} \sum_{j=1}^{n} \sum_{s=1}^{j-1} e^{-x\lambda_s} \prod_{l=1}^{s-1} \left(\frac{\lambda_l}{\lambda_l - \lambda_s} \right)^{k_s} \\ & \times \sum_{s < a_1 < \cdots < a_{l-1} < j} \left(\frac{\lambda_s}{\lambda_s - \lambda_{a_1}} \right)^{k_s} \prod_{m=s+1}^{a_1 - 1} \left(\frac{\lambda_m}{\lambda_m - \lambda_{a_1}} \right)^{k_m} \\ & \times \prod_{n=a_1}^{a_2 - 1} \left(\frac{\lambda_n}{\lambda_n - \lambda_{a_2}} \right)^{k_n} \cdots \prod_{r=a_l - 1}^{j-1} \left(\frac{\lambda_r}{\lambda_r - \lambda_{a_j}} \right)^{k_r} \sum_{q=0}^{k_s - 1} \frac{((\lambda_s - \lambda_{a_1})x)^q}{q!}, \\ & \text{for } x \geq 0 \end{aligned} \tag{69}$$

Specific CDF of hypoexponential distribution Equations (68) and (69) refers to the general CDF of the hypoexponential distribution where the size of the vector parameters can be of any size [8]. In the Markov chain models described in figures 18 and 19 the parameter vectors of the hypoexponential distribution are of size two, and in fact, for any possible version of the investigated Markov chain model the vectors can only be of size two. This is true since for any dimensions of this Markov chain model there will always be at most two distinct exponential parameters; the parameter for finishing a service (μ) and the parameter for moving forward in the queue ($C\mu$). For the case of $C = 1$ the

hypoexponential distribution will not be used as this is equivalent to an erlang distribution. Therefore, by fixing the sizes of $\vec{r}$ and $\vec{\lambda}$ to 2, the following specific expression for the CDF of the hypoexponential distribution arises, where the derivative is removed:

$$
\begin{aligned}
P(H < t) &= 1 - \left(\prod_{j=1}^{|\vec{r}|} \lambda_j^{r_j} \right) \sum_{k=1}^{|\vec{r}|} \sum_{l=1}^{r_k} \frac{\Psi_{k,l}(-\lambda_k) t^{r_k - l} e^{-\lambda_k t}}{(r_k - l)!(l-1)!} \\
\textbf{where} \qquad \Psi_{k,l}(t) &= \begin{cases} \frac{(-1)^l (l-1)!}{\lambda_2} \left[\frac{1}{t^l} - \frac{1}{(t+\lambda_2)^l} \right], & k = 1 \\ -\frac{1}{t(t+\lambda_1)^{r_1}}, & k = 2 \end{cases} \\
\textbf{and} \qquad \lambda_0 &= 0, r_0 = 1
\end{aligned} \tag{70}
$$

Note here that the only difference between equations (68) and (70) is the Ψ function. The next part proves that the expression for Ψ can be simplified for the cases of $k = 1, 2$. Equation (71) shows the expression to be proved.

$$
\Psi_{(k,l)}(t) = -\frac{\partial^{l-1}}{\partial t^{l-1}} \left(\prod_{j=0, j \neq k}^{|\vec{r}|} (\lambda_j + t)^{-r_j} \right) = \begin{cases} \frac{(-1)^l (l-1)!}{\lambda_2} \left[\frac{1}{t^l} - \frac{1}{(t+\lambda_2)^l} \right], & k = 1 \\ -\frac{1}{t(t+\lambda_1)^{r_1}}, & k = 2 \end{cases} \tag{71}
$$

Proof of equation (71) This section aims to show that there exists a simplified version of equation (68) that is specific to the proposed Markov model. Function Ψ is defined using the parameter t and the variables k and l. Given the Markov model, the range of values that k and l can take can be bounded. First, from the range of the double summation in equation (68), it can be seen that $k = 1, 2, \ldots, | \vec{r} |$. Now, $| \vec{r} |$ represents the size of the parameter vectors that, for the Markov model, will always be 2. That is because, for all the exponentially distributed random variables that are added together to form the new distribution, there only two distinct parameters, thus forming two erlang distributions. Therefore:

$$
k = 1, 2
$$

By observing equation (68) once more, the range of values that l takes are $l = 1, 2, \ldots, r_k$, where r_1 is subject to the individual's position in the queue and $r_2 = 1$. In essence, the hypoexponential distribution will be used with these bounds:

$$
\begin{aligned}
k = 1 \qquad &\Rightarrow \qquad l = 1, 2, \ldots, r_1 \\
k = 2 \qquad &\Rightarrow \qquad l = 1
\end{aligned} \tag{72}
$$

Thus the left hand side of equation (71) needs only to be defined for these bounds. The specific hypoexponential distribution investigated here is of the

form $Hypo((r_1, 1)(\lambda_1, \lambda_2))$. Note the initial conditions $\lambda_0 = 0, r_0 = 1$ defined in equation (68) also hold here. Thus the proof is split into two parts, for $k = 1$ and $k = 2$.

- $k = 2, l = 1$

$$\begin{aligned} LHS &= -\frac{\partial^{1-1}}{\partial t^{1-1}} \left(\prod_{j=0, j\neq 2}^{2} (\lambda_j + t)^{-r_j} \right) \\ &= -\left((\lambda_0 + t)^{-r_0} \times (\lambda_1 + t)^{-r_1} \right) \\ &= -\left(t^{-1} \times (\lambda_1 + t)^{-r_1} \right) \\ &= -\frac{1}{t(t + \lambda_1)^{r_1}} \end{aligned}$$

□

- $k = 1, l = 1, \ldots, r_1$

$$\begin{aligned} LHS &= -\frac{\partial^{l-1}}{\partial t^{l-1}} \left(\prod_{j=0, j\neq 1}^{2} (\lambda_j + t)^{-r_j} \right) \\ &= -\frac{\partial^{l-1}}{\partial t^{l-1}} \left((\lambda_o + t)^{-r_0} \times (\lambda_2 + t)^{-r_2} \right) \\ &= -\frac{\partial^{l-1}}{\partial t^{l-1}} \left(\frac{1}{t(t + \lambda_2)} \right) \end{aligned}$$

In essence, it is only needed to show that:

$$-\frac{\partial^{l-1}}{\partial t^{l-1}} \left(\frac{1}{t(t + \lambda_2)} \right) = \frac{(-1)^l (l-1)!}{\lambda_2} \left[\frac{1}{t^l} - \frac{1}{(t + \lambda_2)^l} \right]$$

Proof by Induction:

1. Base case ($l = 1$):

$$\begin{aligned} LHS &= -\frac{\partial^{1-1}}{\partial t^{1-1}} \left(\frac{1}{t(t + \lambda_2)} \right) = -\frac{1}{t(t + \lambda_2)} \\ RHS &= \frac{(-1)^1 (1-1)!}{\lambda_2} \left[\frac{1}{t^1} - \frac{1}{(t + \lambda_2)^1} \right] \\ &= -\frac{t + \lambda_2 - t}{\lambda_2 t(t + \lambda_2)} \\ &= -\frac{1}{t(t + \lambda_2)} \\ LHS &= RHS \end{aligned}$$

2. Assume true for $l = x$:

$$-\frac{\partial^{x-1}}{\partial t^{x-1}}\left(\frac{1}{t(t+\lambda_2)}\right) = \frac{(-1)^x(x-1)!}{\lambda_2}\left[\frac{1}{t^x} - \frac{1}{(t+\lambda_2)^x}\right]$$

3. Prove true for $l = x + 1$. Need to show that:

$$\frac{\partial^x}{\partial t^x}\left(\frac{-1}{t(t+\lambda_2)}\right) = \frac{(-1)^{x+1}(x)!}{\lambda_2}\left[\frac{1}{t^{x+1}} - \frac{1}{(t+\lambda_2)^{x+1}}\right]$$

$$\begin{aligned}
LHS &= \frac{\partial}{\partial t}\left[\frac{\partial^{x-1}}{\partial t^{x-1}}\left(\frac{-1}{t(t+\lambda_2)}\right)\right] \\
&= \frac{\partial}{\partial t}\left[\frac{(-1)^x(x-1)!}{\lambda_2}\left(\frac{1}{t^x} - \frac{1}{(t+\lambda_2)^x}\right)\right] \\
&= \frac{(-1)^x(x-1)!}{\lambda_2}\left(\frac{(-x)}{t^{x+1}} - \frac{(-x)}{(t+\lambda_2)^x}\right) \\
&= \frac{(-1)^x(x-1)!(-x)}{\lambda_2}\left(\frac{1}{t^{x+1}} - \frac{1}{(t+\lambda_2)^x}\right) \\
&= \frac{(-1)^{x+1}(x)!}{\lambda_2}\left(\frac{1}{t^{x+1}} - \frac{1}{(t+\lambda_2)^x}\right) \\
&= RHS
\end{aligned}$$

□

Proportion within target for both types of individuals Given the two CDFs of the Erlang and Hypoexponential distributions a new function has to be defined to decide which one to use among the two. Based on the state of the model, there can be three scenarios when an individual arrives.

1. There is a free server and the individual does not have to wait

$$X_{(u,v)} \sim Erlang(1, \mu)$$

2. The individual arrives at a queue at the n^{th} position and the model has $C > 1$ servers

$$X_{(u,v)} \sim Hypo((n, 1), (C\mu, \mu))$$

3. The individual arrives at a queue at the n^{th} position and the model has $C = 1$ servers

$$X_{(u,v)} \sim Erlang(n + 1, \mu)$$

Note here that for the first case $Erlang(1, \mu)$ is equivalent to $Exp(\mu)$. Consider $X^{(1)}_{(u,v)}$ to be the distribution of type 1 individuals and $X^{(2)}_{(u,v)}$ the distribution of type 2 individuals, when arriving at state (u, v) of the model.

$$X^{(1)}_{(u,v)} \sim \begin{cases} \textbf{Erlang}(v,\mu), & \textbf{if } C=1 \textbf{ and } v>1 \\ \textbf{Hypo}\left([v-C,1],[C\mu,\mu]\right), & \textbf{if } C>1 \textbf{ and } v>C \\ \textbf{Erlang}(1,\mu), & \textbf{if } v\leq C \end{cases} \tag{73}$$

$$X^{(2)}_{(u,v)} \sim \begin{cases} \textbf{Erlang}(\min(v,T),\mu), & \textbf{if } C=1 \textbf{ and } v,T>1 \\ \textbf{Hypo}\left([\min(v,T)-C,1],[C\mu,\mu]\right), & \textbf{if } C>1 \textbf{ and } v,T>C \\ \textbf{Erlang}(1,\mu), & \textbf{if } v\leq C \textbf{ or } T\leq C \end{cases} \tag{74}$$

Thus, the CDF of the random variables $X^{(1)}_{(u,v)}$ and $X^{(2)}_{(u,v)}$ can be calculated using equations (64) and (70):

$$P(X^{(1)}_{(u,v)}<t) = \begin{cases} 1-\sum_{i=0}^{v-1}\frac{1}{i!}e^{-\mu t}(\mu t)^i, & \textbf{if } C=1 \\ & \textbf{and } v>1 \\ 1-(\mu C)^{v-C}\mu\sum_{k=1}^{|\vec{r}|}\sum_{l=1}^{r_k}\frac{\Psi_{k,l}(-\lambda_k)t^{r_k-l}e^{-\lambda_k t}}{(r_k-l)!(l-1)!}, & \textbf{if } C>1 \\ \textbf{where } \vec{r}=(v-C,1) \textbf{ and } \vec{\lambda}=(C\mu,\mu) & \textbf{and } v>C \\ 1-e^{-\mu t}, & \textbf{if } v\leq C \end{cases} \tag{22 revisited}$$

$$P(X^{(2)}_{(u,v)}<t) = \begin{cases} 1-\sum_{i=0}^{\min(v,T)-1}\frac{1}{i!}e^{-\mu t}(\mu t)^i, & \textbf{if } C=1 \\ & \textbf{and } v,T>1 \\ 1-(\mu C)^{\min(v,T)-C}\mu & \textbf{if } C>1 \\ \quad\times\sum_{k=1}^{|\vec{r}|}\sum_{l=1}^{r_k}\frac{\Psi_{k,l}(-\lambda_k)t^{r_k-l}e^{-\lambda_k t}}{(r_k-l)!(l-1)!}, & \textbf{and } v,T>C \\ \textbf{where } \vec{r}=(\min(v,T)-C,1) & \\ \quad\vec{\lambda}=(C\mu,\mu) & \\ 1-e^{-\mu t}, & \textbf{if } v\leq C \\ & \textbf{or } T\leq C \end{cases} \tag{23 revisited}$$

In addition, the set of accepting states for type 1 $S^{(1)}_A$ and type 2 $S^{(2)}_A$ individuals defined in (12) and (13) are also needed here. Note here that, S denotes the set of all states of the Markov chain model.

$$S^{(1)}_A = \{(u,v)\in S \mid v<N\}$$
$$S^{(2)}_A = \begin{cases} \{(u,v)\in S \mid u<M\}, & \textbf{if } T\leq N \\ \{(u,v)\in S \mid v<N\}, & \textbf{otherwise} \end{cases}$$

The following formula uses the state probability vector π to get the weighted average of the probability below target of all states in the Markov model.

$$P(X^{(1)} < t) = \frac{\sum_{(u,v)\in S_A^{(1)}} P(X_{u,v}^{(1)} < t)\pi_{u,v}}{\sum_{(u,v)\in S_A^{(1)}} \pi_{u,v}} \tag{75}$$

$$P(X^{(2)} < t) = \frac{\sum_{(u,v)\in S_A^{(2)}} P(X_{u,v}^{(2)} < t)\pi_{u,v}}{\sum_{(u,v)\in S_A^{(2)}} \pi_{u,v}} \tag{76}$$

Overall proportion within target The overall proportion of individuals for both types of individuals is given by the equivalent formula of equation (11). The following formula uses the probability of lost individuals from both types to get the weighted sum of the two probabilities.

$$P_{L_1'} = \sum_{(u,v)\in S_A^{(1)}} \pi(u,v), \qquad P_{L_2'} = \sum_{(u,v)\in S_A^{(2)}} \pi(u,v)$$

$$\begin{aligned} P(X < t) = {} & \frac{\lambda_1 P_{L_1'}}{\lambda_2 P_{L_2'} + \lambda_1 P_{L_1'}} P(X^{(1)} < t) \\ & + \frac{\lambda_2 P_{L_2'}}{\lambda_2 P_{L_2'} + \lambda_1 P_{L_1'}} P(X^{(2)} < t) \end{aligned} \tag{24 revisited}$$