A 3-player game theoretic model of a choice between two queueing systems with strategic managerial decision making

Michalis Panayides

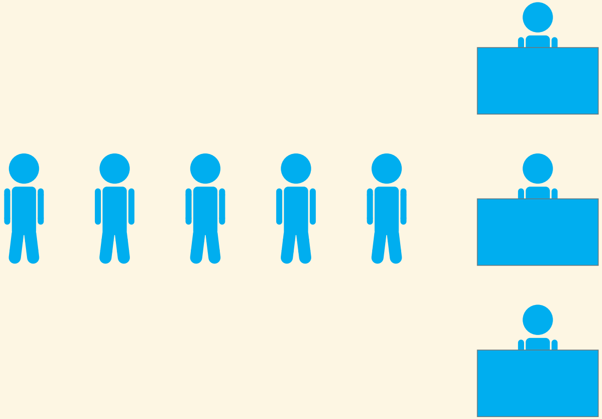
EURO 2021 Athens

About me

About me

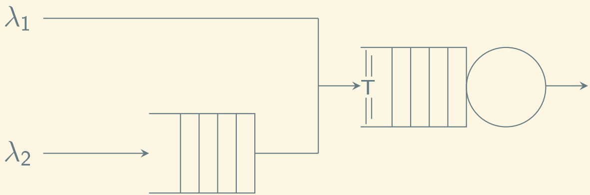
Queues - Examples

Queues - Examples

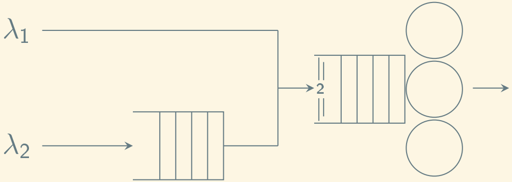


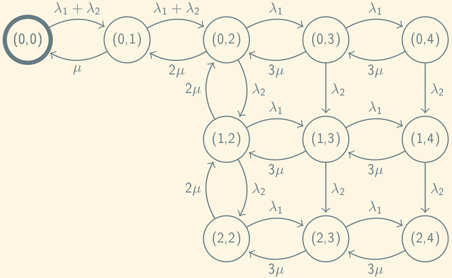
Queues - Examples

Queueing network structure

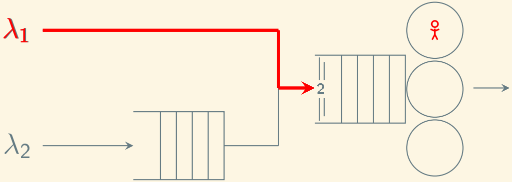


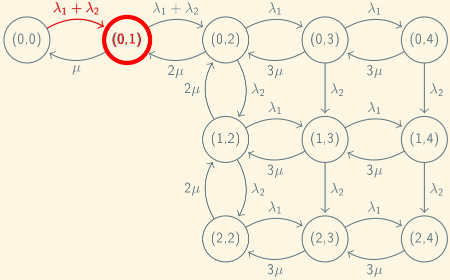
Markov Chain - Custom network



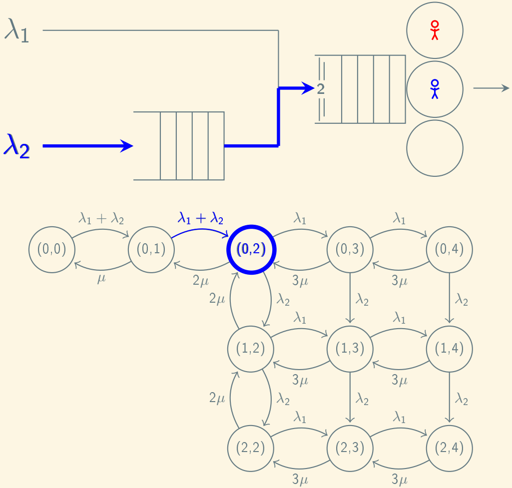


Markov Chain - Custom network

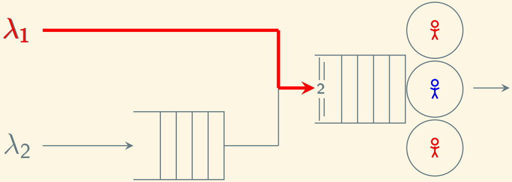


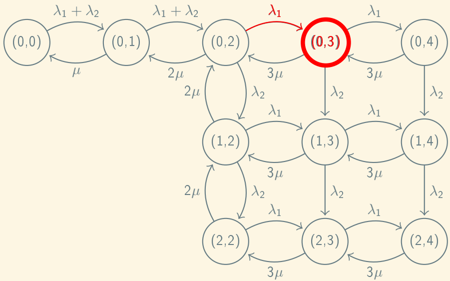


Markov Chain - Custom network

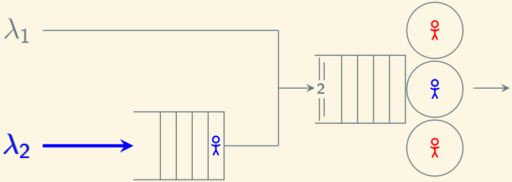


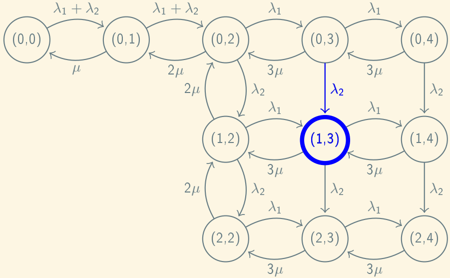
Markov Chain - Custom network



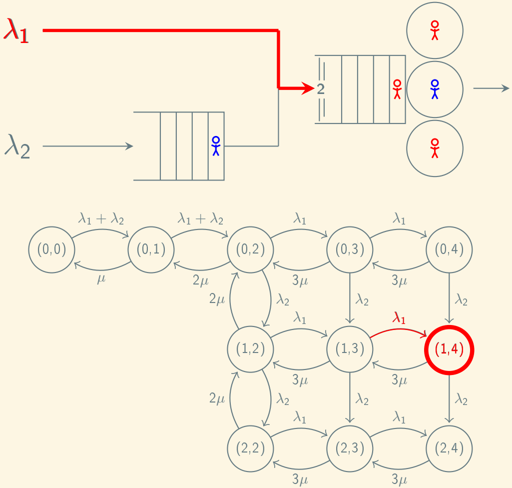


Markov Chain - Custom network

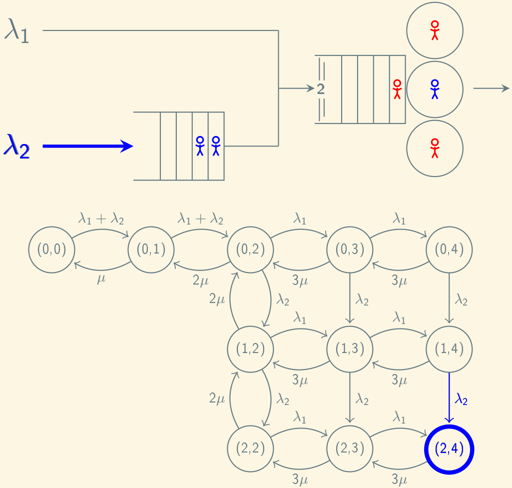




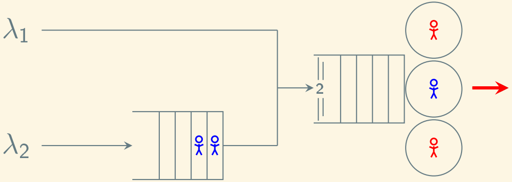
Markov Chain - Custom network

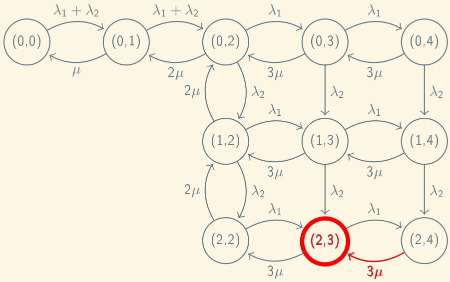


Markov Chain - Custom network

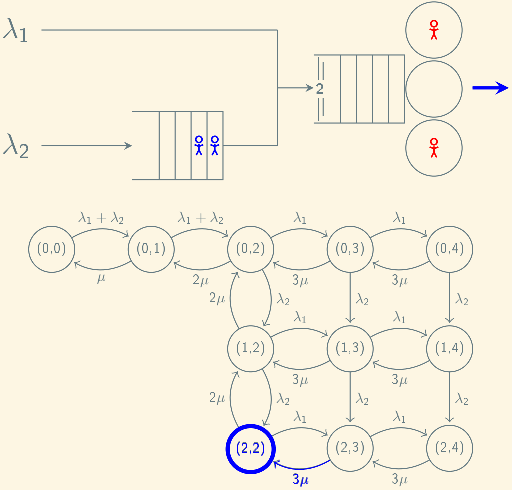


Markov Chain - Custom network

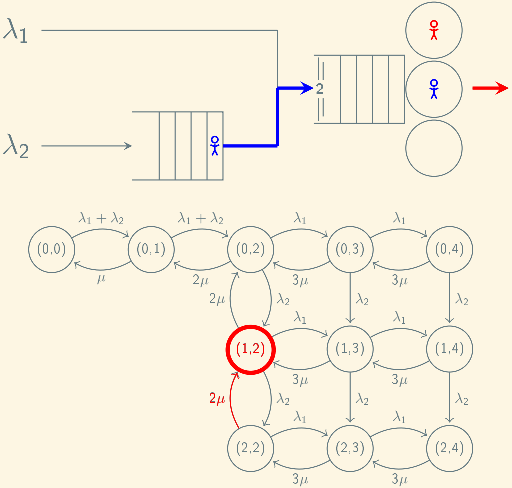




Markov Chain - Custom network

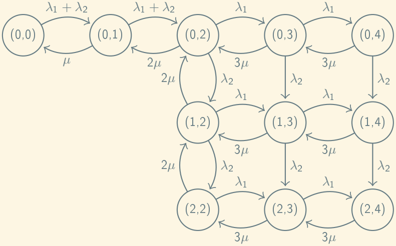


Markov Chain - Custom network



Steady state probabilities - Custom network

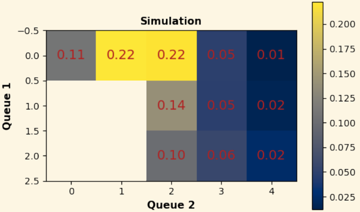
$$
Q = \begin{array}{c}
\begin{array}{cccccc}
(0,0) & (0,1) & (0,2) & & (2,3) & (2,4)
\end{array} \\
\left(\begin{array}{cccccc}
-\lambda_1 - \lambda_2 & \lambda_1 + \lambda_2 & 0 & \dots & 0 & 0 \\
\mu & -\mu - \lambda_1 - \lambda_2 & \lambda_1 + \lambda_2 & \dots & 0 & 0 \\
0 & 2\mu & -2\mu - \lambda_1 - \lambda_2 & \dots & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & \dots & -\lambda_1 - 3\mu & \lambda_1 \\
0 & 0 & 0 & \dots & 3\mu & -3\mu
\end{array}\right)
\end{array}
\begin{array}{c}
(0,0) \\ (0,1) \\ (0,2) \\ \\ (2,3) \\ (2,4)
\end{array}
$$

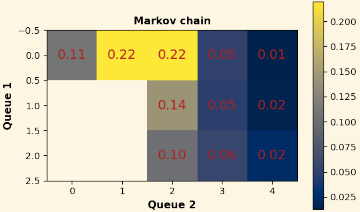


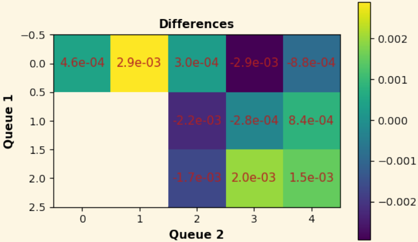
$$
\frac{d\pi}{dt} = \pi Q = 0, \qquad \sum \pi_{(u,v)} = 1
$$

$$
\pi = \begin{bmatrix} \pi_{(0,0)} \\ \pi_{(0,1)} \\ \pi_{(0,2)} \\ \vdots \\ \pi_{(2,3)} \\ \pi_{(2,4)} \end{bmatrix}
$$

Steady state probabilities - Comparison







Performance Measures - Number of individuals

$$(u_i, v_i) \longrightarrow \pi_{(u_i, v_i)}$$

Performance Measures - Number of individuals

$$(u_i, v_i) \longrightarrow \pi_{(u_i, v_i)}$$

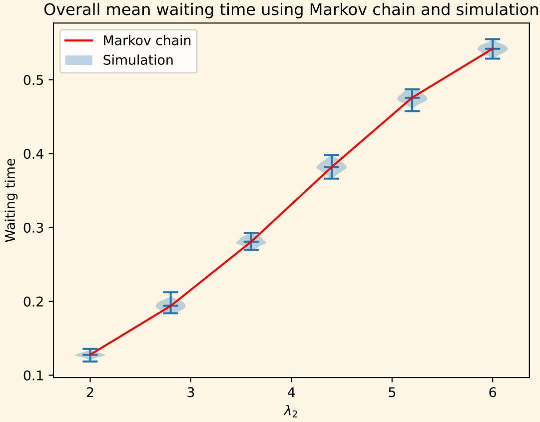
$$L = \sum_{i=1}^{|\pi|} \pi_i (u_i + v_i)$$

$$L_1 = \sum_{i=1}^{|\pi|} \pi_i u_i$$

$$L_2 = \sum_{i=1}^{|\pi|} \pi_i v_i$$

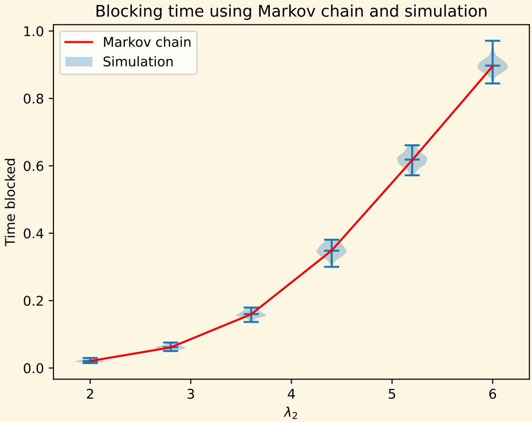
Performance Measures - Waiting time

$$W = \frac{\lambda_1 P_{L'_1}}{\lambda_2 P_{L'_2} + \lambda_1 P_{L'_1}} W^{(1)} + \frac{\lambda_2 P_{L'_2}}{\lambda_2 P_{L'_2} + \lambda_1 P_{L'_1}} W^{(2)} \tag{1}$$



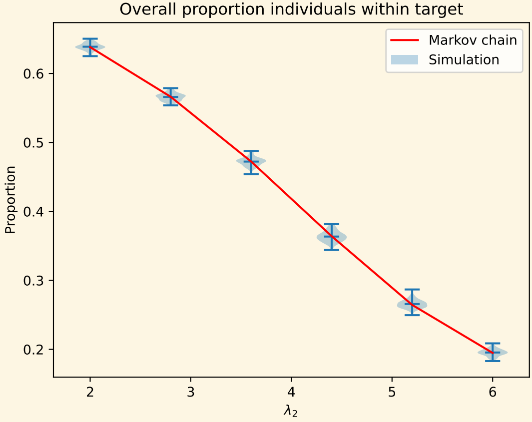
Performance Measures - Blocking time

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



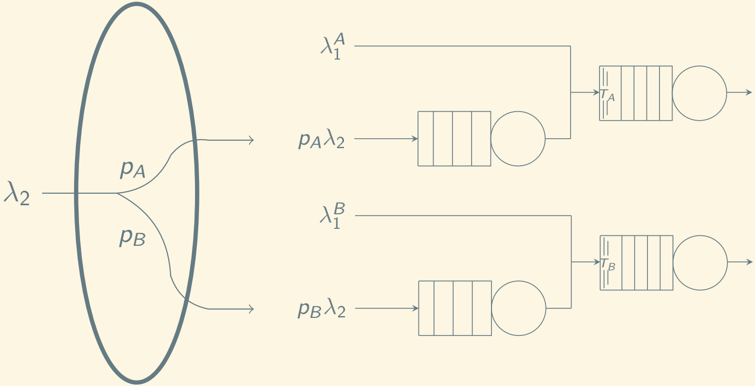
Performance Measures - Proportion within time

$$P(X < t) = \frac{\lambda_1 P_{L'_1}}{\lambda_2 P_{L'_2} + \lambda_1 P_{L'_1}} P(X^{(1)} < t) + \frac{\lambda_2 P_{L'_2}}{\lambda_2 P_{L'_2} + \lambda_1 P_{L'_1}} P(X^{(2)} < t) \tag{3}$$

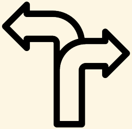


Game - Definition

Game - Players

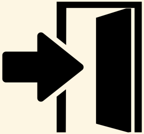


Game - Strategies

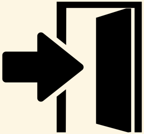


$p_A, p_B \in [0, 1]$

$p_A + p_B = 1$



$T_A \in [1, N_A]$



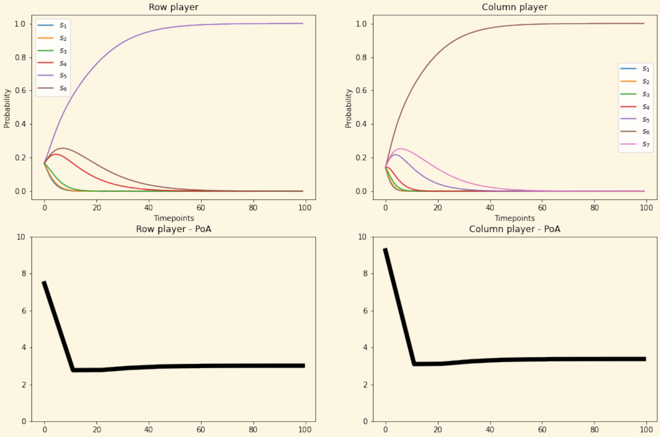
$T_B \in [1, N_B]$

Game - Formulation

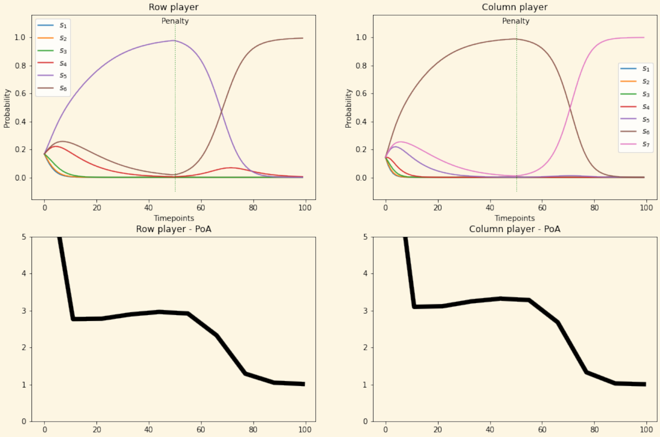
$$A = \begin{pmatrix} U^A_{1,1} & U^A_{1,2} & \cdots & U^A_{1,N_B} \\ U^A_{2,1} & U^A_{2,2} & \cdots & U^A_{2,N_B} \\ \vdots & \vdots & \ddots & \vdots \\ U^A_{N_A,1} & U^A_{N_A,2} & \cdots & U^A_{N_A,N_B} \end{pmatrix}, \quad B = \begin{pmatrix} U^B_{1,1} & U^B_{1,2} & \cdots & U^B_{1,N_B} \\ U^B_{2,1}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

$$R = \begin{pmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,N_B} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,N_B} \\ \vdots & \vdots & \ddots & \vdots \\ p_{N_A,1} & p_{N_A,2} & \cdots & p_{N_A,N_B} \end{pmatrix}$$

Learning algorithms - Asymmetric replicator dynamics



Learning algorithms - Asymmetric replicator dynamics



Code / Test

```python
def get_probability_of_accepting( ···
):
    """
    Generates the probability of acceptance for both class types of a given
    Markov model.

    Parameters
    ----------
    all_states : list
    pi : numpy.array
    threshold : int
    system_capacity : int
    buffer_capacity : int

    Returns
    -------
    list
        The probability of accepting an individual upon its arrival for class 0
        and class 1
    """
    prob_accept = [
        np.sum(
            [
                pi[state]
                for state in all_states
                if is_accepting_state(
                    state=state,
                    class_type=class_type,
                    threshold=threshold,
                    system_capacity=system_capacity,
                    buffer_capacity=buffer_capacity,
                )
            ]
        )
        for class_type in range(2)
    ]
    return prob_accept
```

```python
def test_get_probability_of_accepting_example():
    """
    Test that the probability of accepting an individual is as expected
    """
    all_states = [
        (0, 0),
        (0, 1),
        (0, 2),
        (0, 3),
        (1, 3),
        (2, 3),
        (0, 4),
        (1, 4),
        (2, 4),
        (0, 5),
        (1, 5),
        (2, 5),
    ]
    pi = np.array(
        [
            [0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
            [np.nan, np.nan, np.nan, 0.1, 0.1, 0.05],
            [np.nan, np.nan, np.nan, 0.05, 0.05, 0.05],
        ]
    )

    assert np.allclose(
        get_probability_of_accepting(
            all_states=all_states,
            pi=pi,
            threshold=2,
            system_capacity=5,
            buffer_capacity=2,      Michalis Panayides, 3 weeks ago via PR
        ),
        [0.8, 0.85],
    )
```

GitHub repository