Detecting Deadlock in Discrete Event Simulations of Queueing Networks

Geraint Palmer

This section will discuss the properties and detection of deadlock in queueing networks. Throughout the section, when discussing queueing networks, it is assumed that the queueing network is open and connected.

1 Explanation of Deadlock

Deadlock can be experienced in any open queueing network with feedback loops, where at least one service station has limited queueing capacity, and where individuals can be blocked from joining the queue at the next destination. Deadlock occurs when a customer finishes service at node i and is blocked from transitioning to node j; however the individuals in node j are all blocked, directly or indirectly, by the blocked individual in node i. That is, deadlock occurs if every individual blocking individual X, directly or indirectly, are also blocked.

In figure 1 a simple two node queueing network is shown in a deadlocked state. Customer e has finished service at node 1, but remains there as there is not enough queueing space at node 2 to accept him. We say he is blocked by customer i, as he is waiting for customer i to be released. Similarly, customer i has finished service at node 2, but remains there as there is not enough queueing space at node 1, customer i is blocked by customer e.

When there are multiple servers, individuals become blocked by all customers in service or blocked at their destination. Figure 2a shows two nodes in deadlock, customer i is blocked by both d and e, who are both blocked by customer i. However in 2b, customer i is blocked by both d and e, and customer d isn't blocked, and so there is no deadlock.

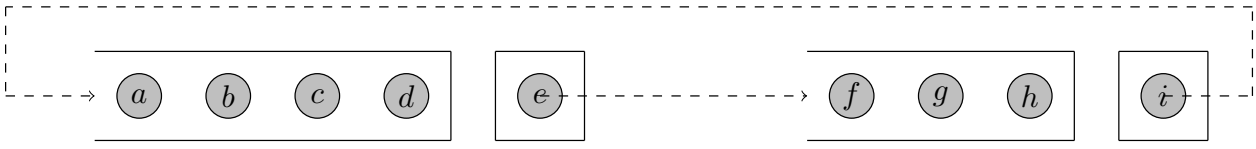
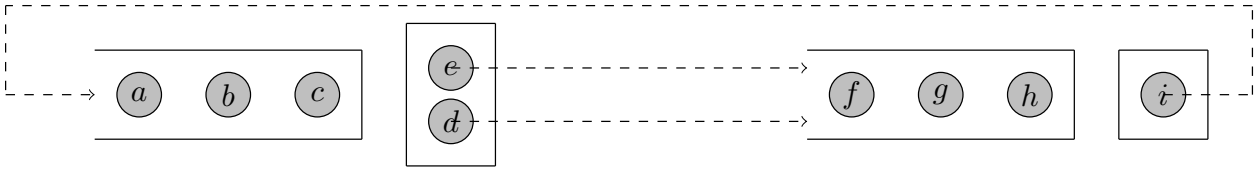
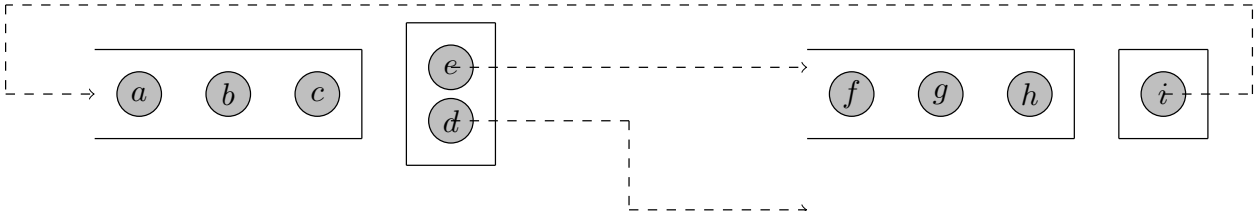


Figure 1: Two nodes in deadlock.



(a) Two nodes in deadlock.



(b) Two nodes not deadlocked

Figure 2: Two nodes in deadlock and not in deadlock.

Note that the whole queueing network need not be deadlocked, only a part of it. If one section of the network is in deadlock, then the system is deadlocked, even though customers may still be able to have services and transitions in other areas of the network. An example is shown in figure 3. Here nodes 1 and 2 are in deadlock, so individuals e and h cannot transition to the next node as they are blocking one another. Individual k on the other hand is free to move to its next destination. This idea is expanded on in the next section.

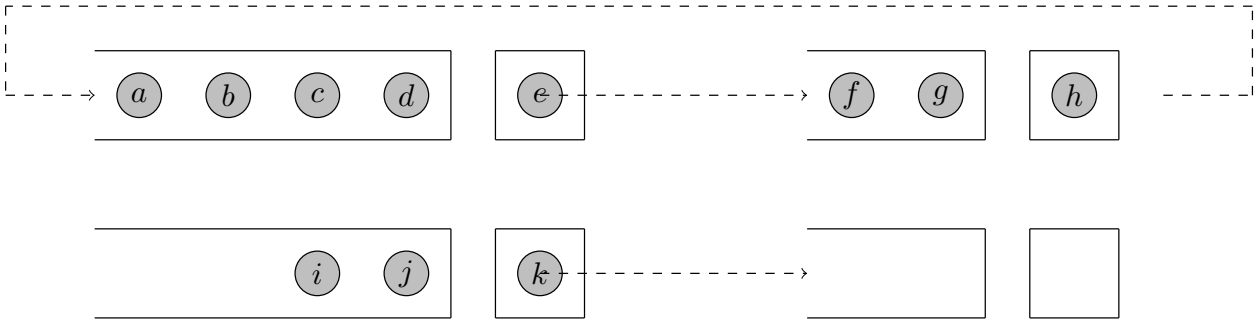


Figure 3: A deadlock situation where not all nodes are deadlocked.

2 Types of Deadlock

The previous section introduced the idea that parts of a queueing network can be in a deadlocked state, although other parts will continue to flow. The different configurations of which nodes experience deadlock can be thought of ad different types of deadlock. Each different type of deadlocked state can be denoted $\Delta_{i,j,k...}$ with the indices $i, j, k...$ denoting which nodes are participating in that deadlocked state. The amount of different types of deadlock that a queueing network can experience is equal to the number of directed cycles in in the queueing network's routing matrix.

For connected queueing networks, these deadlocks can be classified into transient deadlocked states and

the absorbing deadlocked state. For a queueing network Q with N service stations, the absorbing deadlocked state corresponds to $\Delta_{1,\ldots,N}$, the state where all service stations experience deadlock. It should be clear that if the queueing network is connected, then there is a non-zero probability that once one part of the network is in deadlock, the whole system will fall into a deadlocked state, simply by the individuals in the non-deadlocked nodes attempting to transition into a deadlocked node. That is, once Q falls into one of the transient deadlocked states, it will eventually transition, either directly or through other transient deadlocked states, into the absorbing deadlocked state.

If the routing matrix of Q is complete, that is there is a possible route from every service station to every other service station, then there are $\sum_{i=1}^{N} \binom{N}{i}$ possible deadlock types.

3 Literature Review

Most of the literature on blocking conveniently assumes the networks are deadlock-free. For closed networks of K customers with only one class of customer, [3] proves the following condition to ensures no deadlock: for each minimum cycle C, $K < \sum_{j \in C} B_j$, the total number of customers cannot exceed the total queueing capacity of each minimum subcycle of the network. The paper also presents algorithms for finding the minimum queueing space required to ensure deadlock never occurs, for closed cactus networks, where no two cycles have more than one node in common. This result is extended to multiple classes of customer in [4], with more restrictions such as single servers and each class having the same service time distribution. Here a integer linear program is formulated to find the minimum queueing space assignment that prevents deadlock. The literature does not discuss deadlock properties in open restricted queueing networks.

General deadlock situations that are not specific to queueing networks are discussed in [2]. Conditions for this type of deadlock, also referred to as deadly embraces, to potentially occur are given:

- Mutual exclusion: Tasks have exclusive control over resources.
- Wait for: Tasks do not release resources while waiting for other resources.
- No preemption: Resources cannot be removed until they have been used to completion.
- Circular wait: A circular chain of tasks exists, where each task requests a resource from another task in the chain.

Dynamic state-graphs are defined, with resources as vertices and requests as edges. For scenarios where there is only one type of each resource, deadlock arises if and only if the state-graph contains a cylce.

In [1] the vertices and edges of the state graph are given labels in relation to a reference node. Using these labels *simple bounded circuits* are defined whose existance within the state graph is sufficient to detect deadlock.

4 Definitions

$\|V(D)\|$	The order of the directed graph D is its number of vertices.
Weakly connected component	A weakly connected component of a digraph containing X is the set of all nodes that can be reached from X if we ignore the direction of the edges.
Direct successor	If a directed graph contains an edge from X_i to X_j , then we say that X_j is a direct successor of X_i.
Ancestors	If a directed graph contains a path from X_i to X_j , then we say that X_i is an ancestor of X_j.
Decendents	If a directed graph contains a path from X_i to X_j , then we say that X_j is a descendant of X_i.
$\deg^{\text{out}}(X)$	The out-degree of X is the number of outgoing edges emanating from that vertex.
Subgraph	A subgraph H of a graph G is a graph whose vertices are a subset of the vertex set of G, and whose edges are a subset of the edge set of G.
Sink vertex	A sink vertex is a vertex in a directed graph that has no out-degree of zero.

NEED CONSISTANT NOTATION, REFERENCES FOR DEFINITIONS.

5 Explaining Digraph

Presented is a method of detecting when deadlock occurs in an open queueing network Q with N nodes, using a dynamic directed graph, the state graph. Let the number of servers in node i be denoted by c_i.

Define $D(t)$ as the state graph of Q at time t. Each vertex of $D(t)$ is an occupied server, occupied by an individual who is either in service or blocked, and so $|V(D(t))| \leq \sum_{i=1}^{N} c_i, \quad \forall t \geq 0$. The state graph shows the blockage relationships between servers: a directed edge from one X_a to X_b represents that the customer occupying X_a has finished service and is waiting for the X_b to release its occupant before he can be realsed himself.

The state graph $D(t)$ can be partitioned into N service-station subgraphs, $d_1(t), d_2(t), ..., d_N(t)$, where the vertices of $d_i(t)$ represent the servers of node i. The vertex set of each subgraph is static over time, however their edge sets may change.

The state graph is dynamically built up as follows. When an individual finishes service at node i, and this individuals next destination is node j, but there is not enough queueing capacity for j to accept that individual, then that individual remains at node i and becomes blocked. At this point c_j directed edges between this individual's server and the vertices of $d_j(t)$ are created in $D(t)$.

When an individual is released and another customer who wasn't blocked occupies their server, that

servers out-edges are removed. When an individual is released and another customer who was previously blocked occupies their server, that server's out-edges are removed along with the in-edge from the server who that previously blocked customer occupied. When an individual is released and there isn't another customer to occupy that server, then all edges incedent to that server are removed.

This general process of building up the state graph as the queueing network is simulated is illustrated in figure 4. Customers are labelled (i, j, k) where i denotes the server that customer is occupying, j denotes that individuals i.d. number, and k denotes the service station that customer is waiting to enter. The simulation starts with full queues, and every server occupied by a customer in service. Customer 14 finishes service, and is blocked from entering node A. Then customer 5 finishes service and is blocked from entering node B. Then customer 10 finishes service and is blocked from entering node A. Finally customer 6 finishes service and wants to reenter the queue for node A but is blocked. A deadlock situation arises as customer 6 is waiting for customer 5 to move, who is waiting for customer 10 to move, who is waiting for either customer 5 or 6 to move. The state graph is shown at each event.

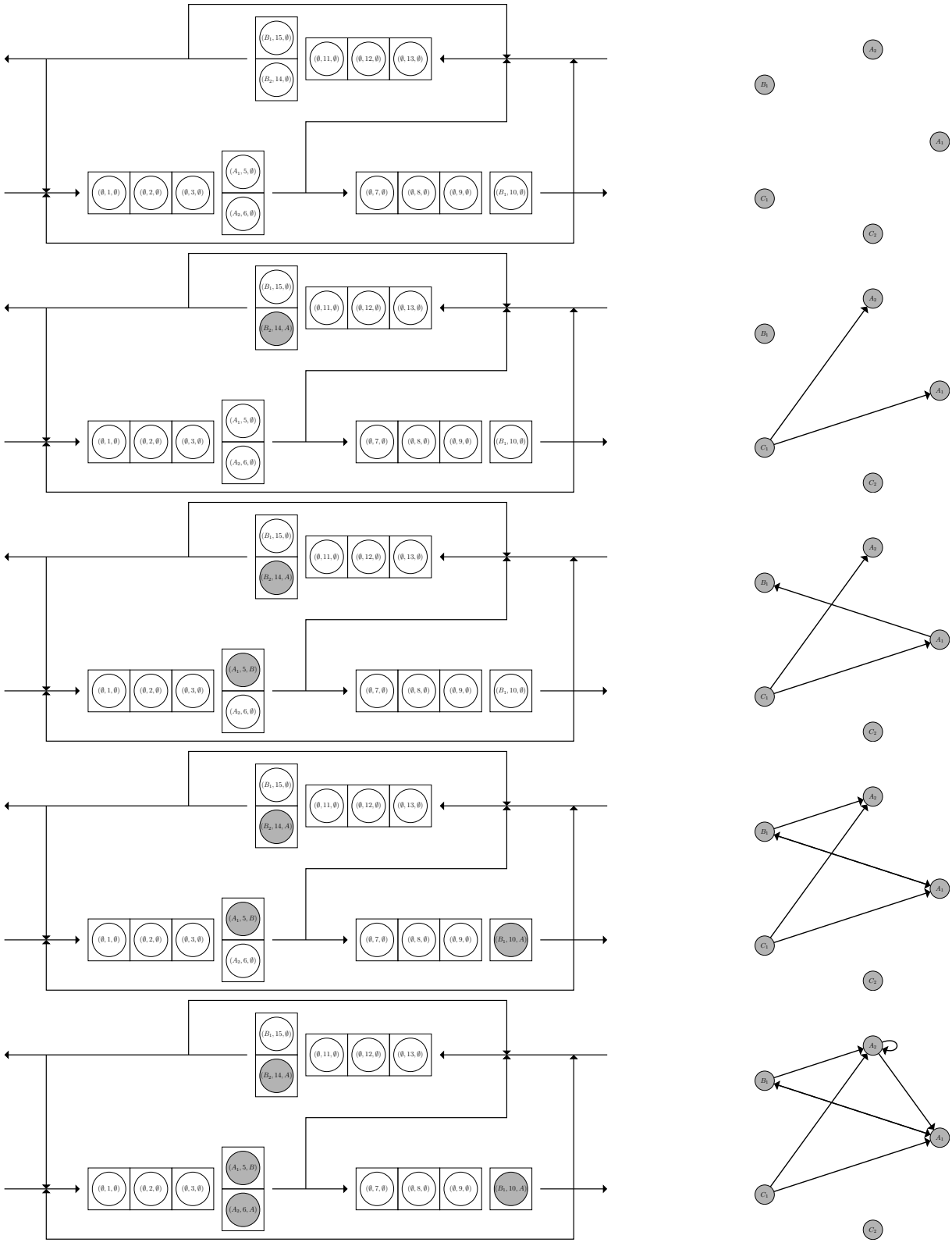


Figure 4: Illustrating the general build up of the state graph until deadlock.

The rules on how edges are removed from the state graph are shown in figure 5. Here the simulation begins with four customers occupying servers; those at node A blocked to node B, the customer at node C blocked to node A, and the customer at node B still in service. Customer 6 finishes service and immediately joins service at node C. Now there is room for customer 4 to move into service at node B. Notice that the edge $A_2 - -B_1$ remains in the state graph, as customer 5 is still blocked by that server. The customers queueing at node A move along the queue, with customer 3 begining service. This leaves enough room for customer 7 to join the back of the queue at A.

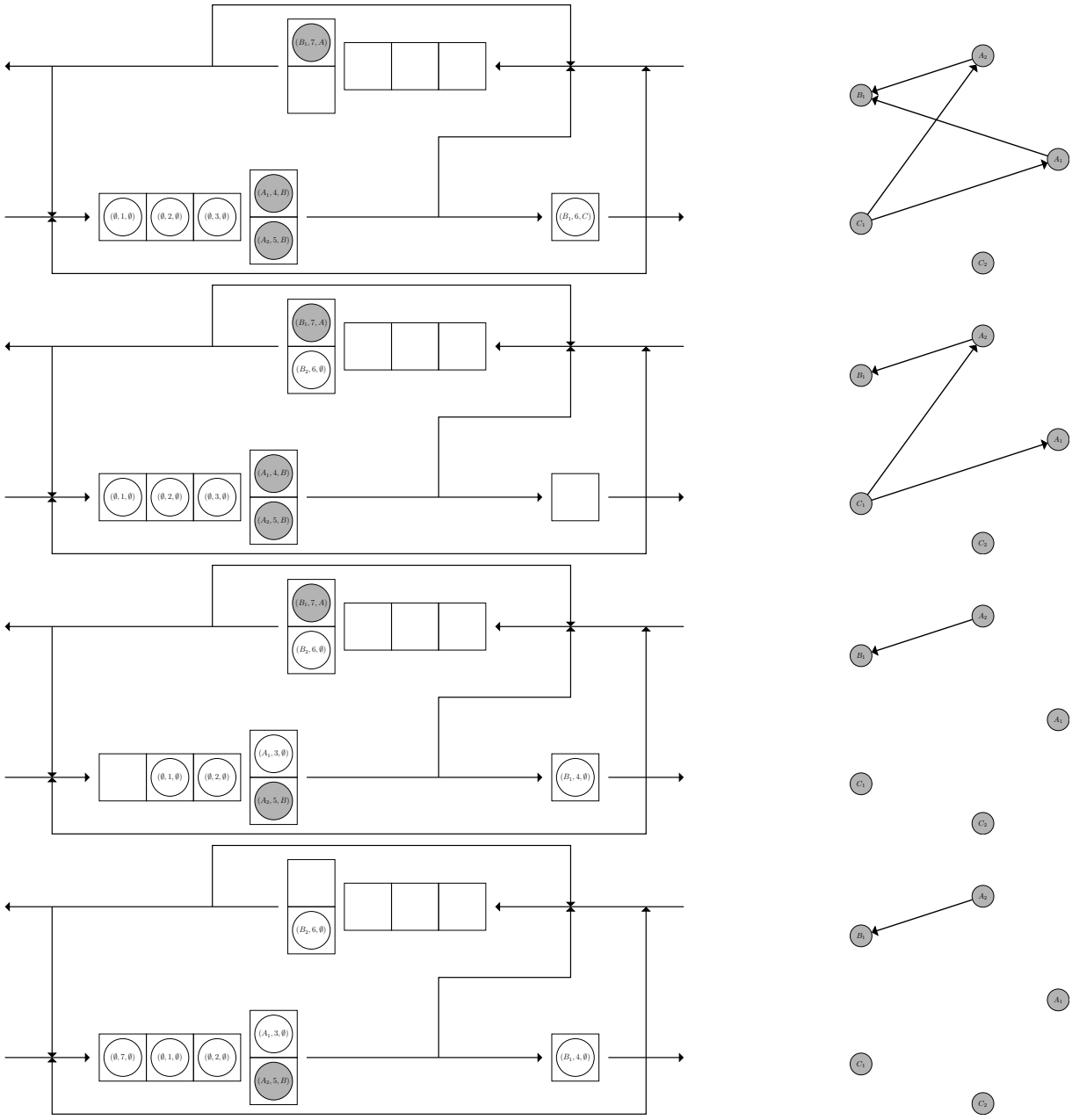


Figure 5: Illustrating how the state graph evolves when blocked customer start moving.

6 Theorem

Theorem

A deadlock situation arises at time t if and only if there exists a weakly connected component of $D(t)$ that doesn't contain a sink vertex.

Observations

Consider one weakly connected component $G(t)$ of $D(t)$. Consider the node $X_a \in G(t)$. If X_a is unoccupied, then X_a has no incident edges. Consider the case when X_a is occupied by individual a, whose next destination is node j. Then X_a's direct successors are the servers occupied by individuals who are blocked or in service at node j. We can interpret all X_a's decendents as the servers whose occupants are directly or indirectly blocking a, and we can interpret all X_a's ancestors as those servers whose individuals who are being blocked directly or indirectly by a.

Note that the only possibilities for $\deg^{\text{out}}(X_a)$ are being 0 or c_j. If $\deg^{\text{out}}(X_a) = c_j$ then a is blocked by all its direct successors. The only other situation is that a is not blocked, and $X_a \in G(t)$ because a is in service at X_a and blocking other individuals, in which case $\deg^{\text{out}}(X_a) = 0$.

It is clear that if any of X_a's decendents are occupied by individuals who are not blocked, then we do not have deadlock at time t; and if all of X_a's descendents are occupied by blocked individuals, then the system is deadlocked at time t. We also know that by definition all of X_a's ancestors are occupied by blocked individuals.

Also note that if a service-station subgraph $d_i(t)$ contains edges, then there is an individual in $X_a \in d_i(t)$ that is being blocked by himself. This does not necassarily mean there is deadlock.

Proof

Consider one weakly connected component $G(t)$ of $D(t)$ at time t. All vertices of $G(t)$ are either decendents of another vertex and so are occupied by an individual who is blocking someone; or are ancestors of another vertex, and so are occupied by someone who is blocked.

Assume that $G(t)$ contains a vertex X such that $\deg^{\text{out}}(X) = 0$. This imples that X's occupant is not blocked, so X must be a descendent of another vertrex. Therefore Q is not deadlocked as there exists a vertex whose descendents are not all blocked.

Now assume that we have deadlock. For a given vertex X, all decendents of X are are occupied by individuals who are blocked, and so must have out-degrees greater than 0. However, as our choice of X was arbitrary, this must be true for all members of $G(t)$, and so no members of $G(t)$ have out-degree of 0.

7 Markov Chain Model

Consider the queueing network shown in figure 6. This shows two $M/M/1$ queues, with n_i queueing capacity at at each service station and service rates μ_i. Λ_i is the external arrival rates to each service station. All routing possibilities except self loops are possible, where the routing probability from node i to node j is denoted by r_{ij}.

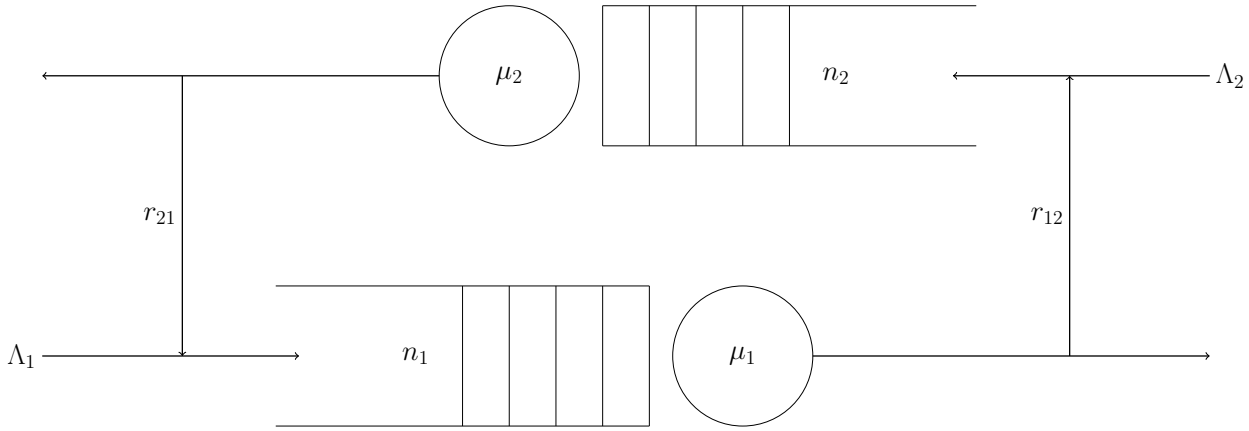


Figure 6: A two node queueing network.

- State space:

$$S = \{(i,j) \in \mathbb{N}^{(n_1+2\times n_2+2)} | 0 \leq n_1 + n_2 + 2 \geq i + j\} \cup \{\Delta_{1,2}\}$$

  Where i denotes number of individuals:

  - In service or waiting at the first node.
  - Occupying a server but having finished service at the second node waiting to join the first

  Where j denotes number of individuals:

  - In service or waiting at the second node.
  - Occupying a server but having finished service at the first node waiting to join the first

  and the state $\Delta_{1,2}$ denotes the deadlocked state.

If we define $\delta = (i_2, j_2) - (i_1, j_1)$, then for all $(i_1, j_1), (i_2, j_2), s \in S$ the transitions are given by:

$$q_{(i_1,j_1),(i_2,j_2)} = \begin{cases} \left.\begin{array}{rl} \Lambda_1 & \text{if } i_1 \leq n_1 \\ 0 & \text{otherwise} \end{array}\right\} & \text{if } \delta = (1,0) \\ \left.\begin{array}{rl} \Lambda_2 & \text{if } j_1 \leq n_2 \\ 0 & \text{otherwise} \end{array}\right\} & \text{if } \delta = (0,1) \\ (1-r_{12})\mu_1 \min(i_1,1) & \text{if } \delta = (-1,0) \\ (1-r_{21})\mu_2 \min(j_1,1) & \text{if } \delta = (0,-1) \\ \left.\begin{array}{rl} 0 & \text{if } j_1 = n_2+2 \\ r_{12}\mu_1 \min(i_1,1) & \text{otherwise} \end{array}\right\} & \text{if } \delta = (-1,1) \\ \left.\begin{array}{rl} 0 & \text{if } i_1 = n_1+2 \\ r_{21}\mu_2 \min(j_1,1) & \text{otherwise} \end{array}\right\} & \text{if } \delta = (1,-1) \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$q_{(i_1,j_1),\Delta_{1,2}} = \begin{cases} r_{21}\mu_2 & \text{if } (i,j) = (n_1, n_2+2) \\ r_{12}\mu_1 & \text{if } (i,j) = (n_1+2, n_2) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

and

$$q_{-1,s} = 0 \tag{3}$$

From this we can find the expected time until deadlock is reached. It is shown in [5] that for a discrete transition matrix of the form $P = \left(\begin{smallmatrix} T & U \\ 0 & V \end{smallmatrix}\right)$ then the expected number of time steps until absorbtion starting from state i is the ith element of the vector

$$(I - T)^{-1} e \tag{4}$$

where e is the ones vector.

References

[1] H. Cho, T. Kumaran, and R. Wysk. Graph-theoretic deadlock detection and resolution for flexible manufacturing systems. *IEEE transactions on robotics and automation*, 11(3):413–421, 1995.

[2] E. Coffman and M. Elphick. System deadlocks. *Computing surveys*, 3(2):67–78, 1971.

[3] S. Kundu and I. Akyildiz. Deadlock buffer allocation in closed queueing networks. *Queueing systems*, 4(1):47–56, 1989.

[4] J. Liebeherr and I. Akyildiz. Deadlock properties of queueing networks with finite capacities and multiple routing chains. *Queueing systems*, 20(3-4):409–431, 1995.

[5] W. Stewart. *Probability, markov chains, queues, and simulation.* Princeton university press, 2009.