Detecting Deadlock in Discrete Event Simulations of Queueing Networks

Geraint Palmer

This section will define and 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. Open queueing networks are those networks that have at least one node to which customers arrive from the exterior, and and least one node from which customers leave to the exterior.

1 Deadlock

Definition 1. *When a simulation is in a situation where at least one service station, despite having arrivals, ceases to finish any more services due to recursive upstream blocking the system is said to be in deadlock.*

Deadlock can be experienced in any open queueing network that experiences blocking, with at least once cycle containing all service stations with resticted queueing capacity.

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 them. We say customer e 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. at the destination service station. 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.

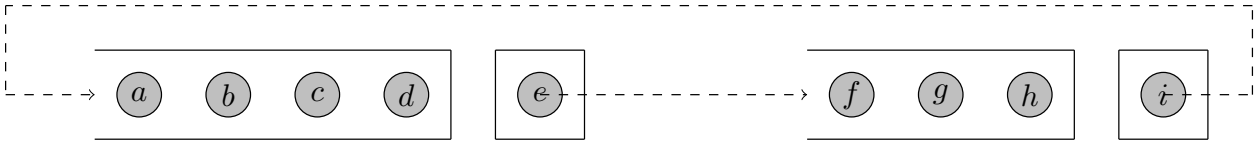
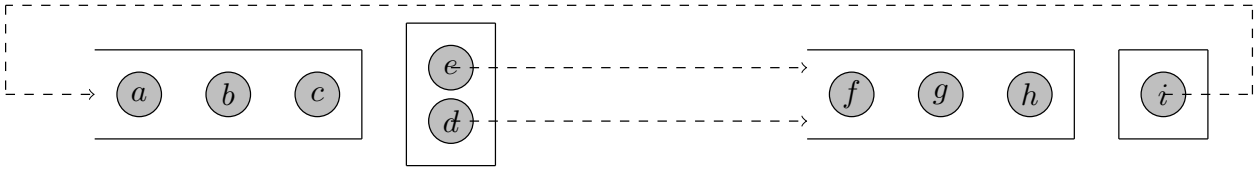
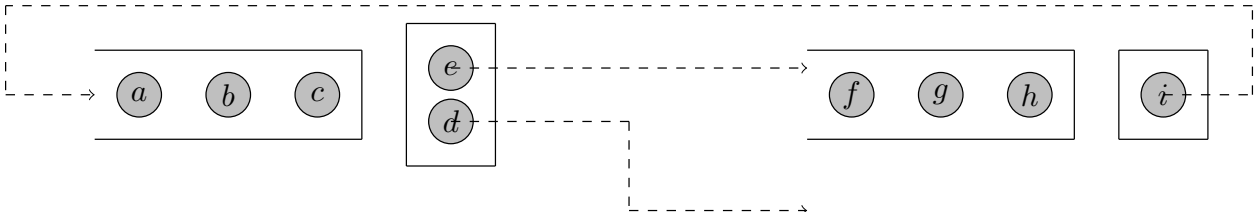


Figure 1: Two nodes in deadlock.



(a) Two nodes in deadlock.



(b) Two nodes not deadlocked

Figure 2: Two nodes: a) in deadlock and b) 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.

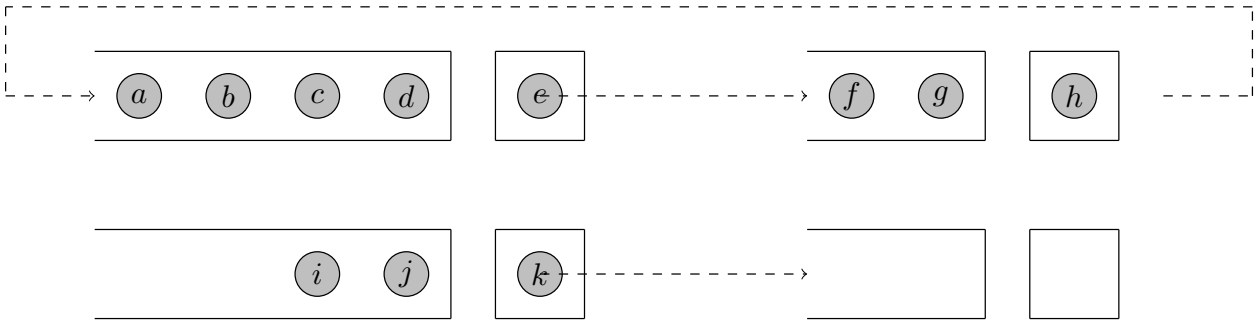


Figure 3: A deadlock situation where not all nodes are deadlocked.

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 as different types of deadlock. Each different type of deadlocked state can be denoted $(i_1, i_2, i_3...)$ where $i_k = -1$ if node k node is 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.

Definition 2. *A transient deadlock state is when there are still some changes of state whilst a subgraph of the queueing network is itself in deadlock.*

Definition 3. *The absorbing deadlock state is when all subgraphs of the queueing network are in deadlock.*

For a queueing network Q with N service stations, the absorbing deadlocked state corresponds to $(i_1 = -1, ..., i_N = -1)$, 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.
Descendants	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 out-degree of zero.
Knot	In a directed graph, a knot is a set of vertices with out-edges such that while traversing the directed edges of that directed graph, once a vertex in the knot is reached, you cannot rech any vertex that is not in the knot.

NEED CONSISTANT NOTATION, REFERENCES FOR DEFINITIONS.

5 State 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 $D(t) = (V(t), E(t))$ as the state graph of Q at time t.

The vertices at time t, $V(t)$ correspond to servers in the queueing system. Thus, $|V(D(t))| = \sum_{i=1}^{N} c_i$ for all $t \geq 0$.

The edges at time t, $E(t)$ correspond to a blocking relationship. There is a directed edge at time t from vertex $X_a \in V(t)$ to vertex $X_b \in V(t)$ if and only if an individual occupying the server corresponding to vertex X_a is being blocked by an individual occupying the server corresponding to vertex X_b.

The state graph $D(t)$ can be partitioned into N service-station subgraphs, $D(t) = \bigcup_{i=1}^{N} d_i(t)$, 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 individual's 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 incident to that server are removed.

This general process of building up the state graph as the queueing network is simulated will now be shown. 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. As an example, a customer labelled $(A_2, 10, C)$ would have an i.d. number of 2, is occupying server A_2 and is currently waiting to join node C. If a customer isn't occupying a server the notation $\emptyset$ is used. Similarly for customers occupying a server and still in service, their next destination is yet undecided, so $\emptyset$ is used.

The simulation starts with full queues, and every server occupied by a customer in service. This is shown in Figure 4.

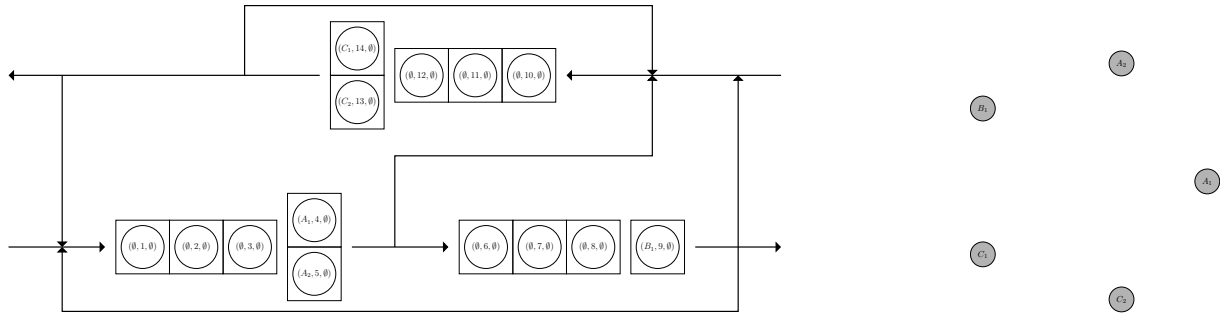


Figure 4

Customer 13 finishes service, and is blocked from entering node A. Figure 5.

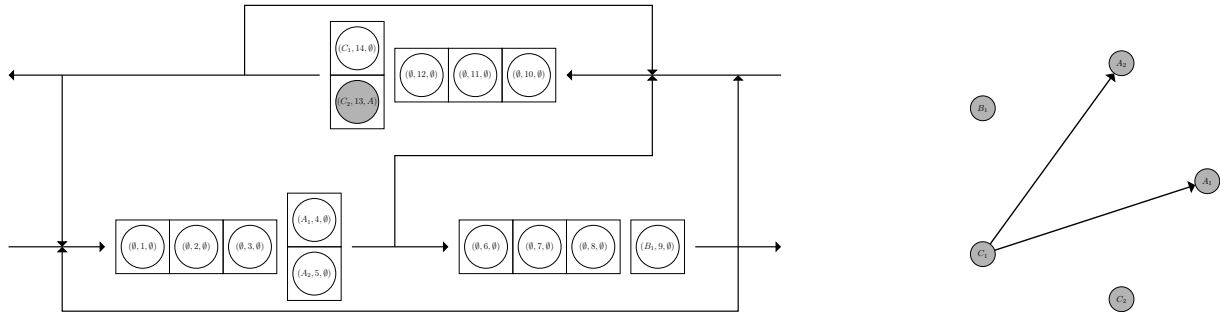


Figure 5

Then customer 4 finishes service and is blocked from entering node B. Figure 6.

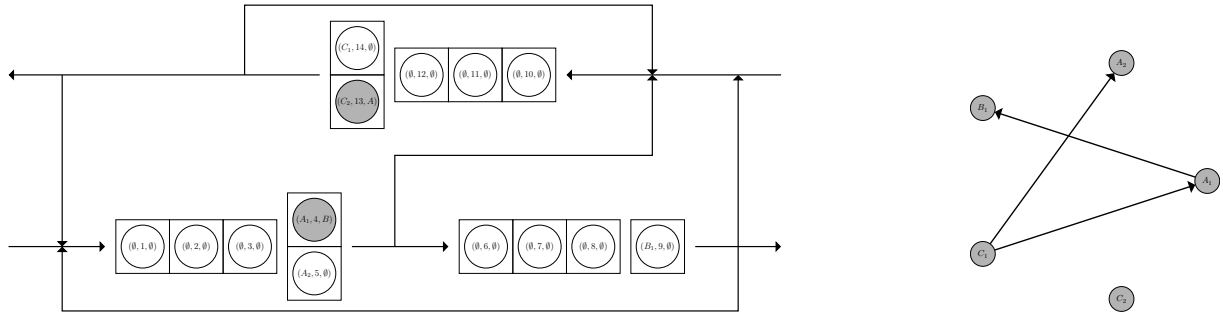


Figure 6

Then customer 9 finishes service and is blocked from entering node A. Figure 7.

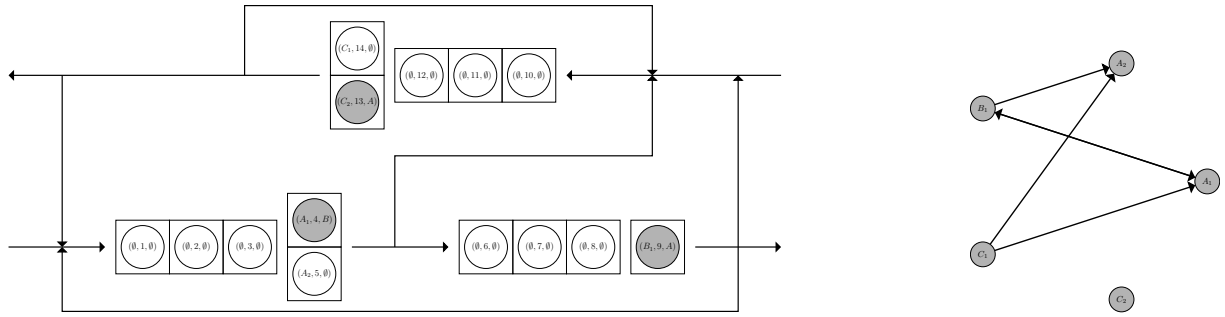


Figure 7

Finally, in Figure 8 customer 5 finishes service and wants to reenter the queue for node A but is blocked. A deadlock situation arises as customer 5 is waiting for customer 4 to move, who is waiting for customer 9 to move, who is waiting for either customer 4 or 5 to move.

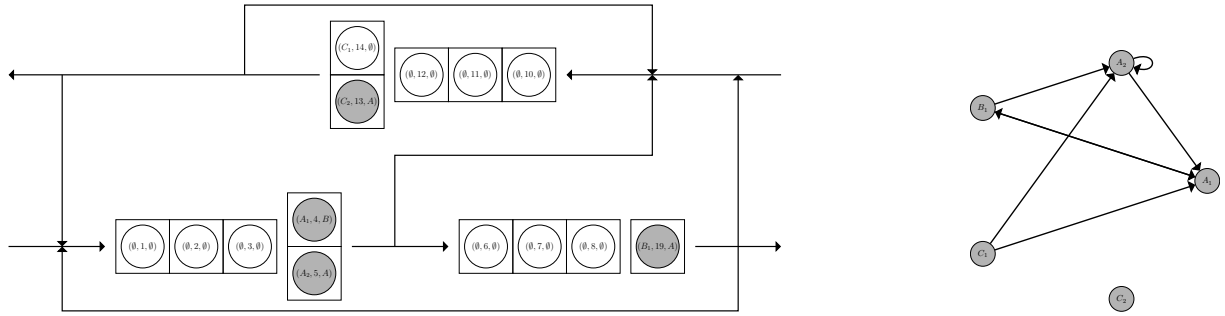


Figure 8

The rules on how edges are removed from the state graph will now be shown. For illustrative purposes the queueing network here is a different queueing network than discussed above.

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. This is shown in Figure 9.

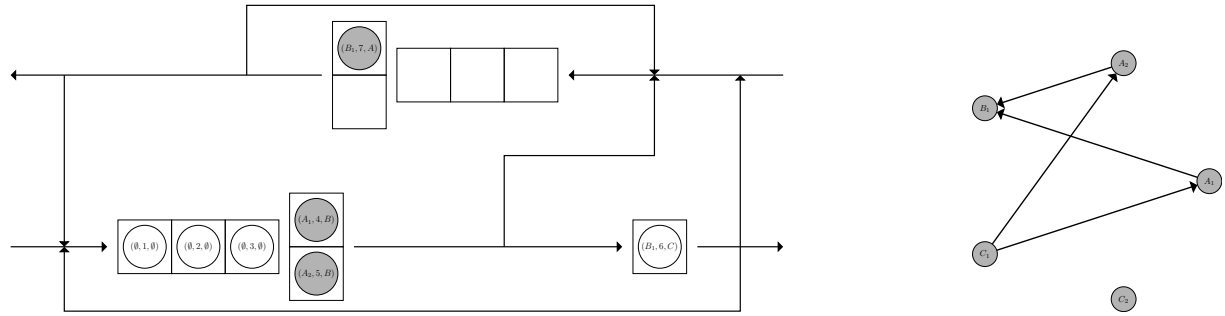


Figure 9

Customer 6 finishes service and immediately joins service at node C. Figure 10.

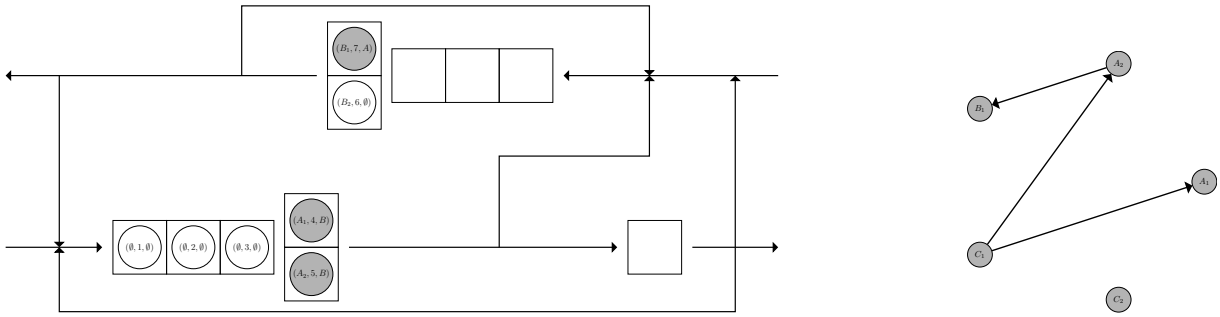


Figure 10

Now there is room for customer 4 to move into service at node B. Figure 11. Notice that the edge $A_2 \longrightarrow B_1$ remains in the state graph, as customer 5 is still blocked by that server.

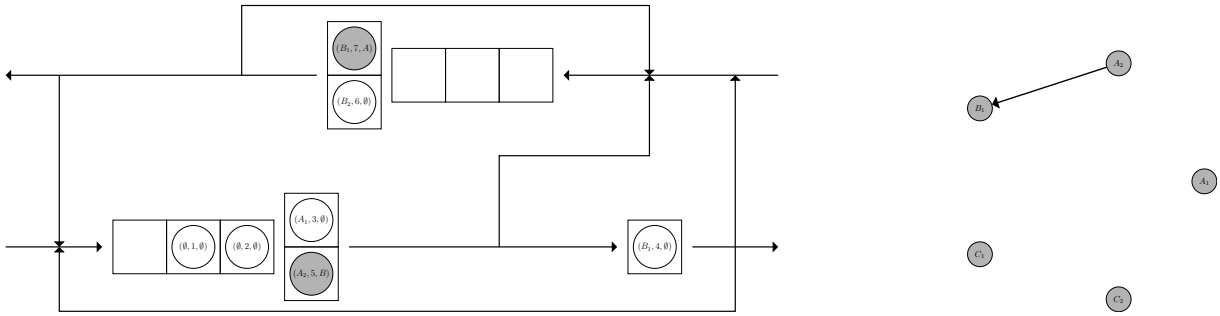


Figure 11

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 12.

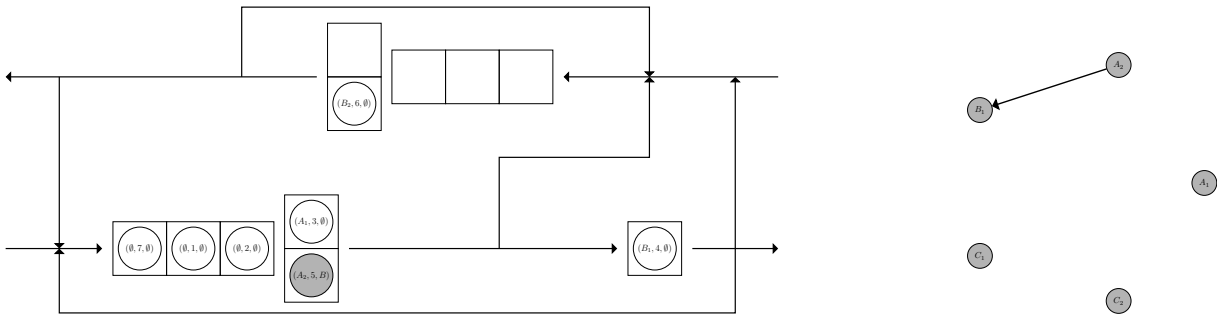


Figure 12

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 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.

6 Theorem

Theorem 1. *A deadlocked state arises at time t if and only if $D(t)$ contains a knot.*

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 $\deg^{\text{out}}(X) = 0$, and there is a path from every other non-sink vertix to X. This imples that X's occupant is not blocked and is a descendent of another vertrex. Therefore Q is not deadlocked as there does not exist a vertex whose descendents are all blocked.

Now assume that we have deadlock. For a vertex X who is deadlocked, all decendents of X are are occupied by individuals who are blocked, and so must have out-degrees greater than 0. And so there is no path from X to a vertex with out-degree of 0. □

Lemma 1. *For a queueing network with two nodes or less, a deadlocked state arises if and only if there exists a weakly connected component without a sink node.*

Proof. Consider a one node queueing network Q_1.

If there is deadlock, then all servers are occupied by blocked individuals, and so all servers have an out-edge.

Consider a two node queueing network Q_2.

If both nodes are involved in the deadlock, so there is a customer in node 1 blocked from entering node 2, and a customer from node 2 blocked from entering node 1, then all servers in node 1 and node 2 in $D(t)$

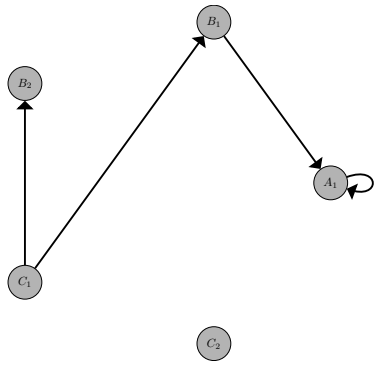


Figure 13: A Counter-Example State Digraph.

will have out edges as they are occupied by a blocked individual. The servers of node 1 and 2 consist of the entirety of $D(t)$, and so there is no sink nodes.

Now consider the case when only one node is involved in the deadlock. Without loss of generality, let's say that node 1 is in deadlock with itself, then the servers of node 1 have out-edges. For the servers of node 2 to be part of that weakly conneced component, there either needs to be an edge from a server in node 1 to a server in node 2, or and edge from a server in node 2 to a server in node 1. An edge from a server in node 1 to a server in node 2 implies that a customer from node 1 is blocked from entering node 2, and so node 1 is not in deadlock with itself. An edge from a server in node 2 to a server in node 1 implies that a customer in node 2 is blocked from entering node 1. In this case the server in node 2 has an out-edge, and so there is still no sink.

For the case of a queueing network with more than two nodes, the following counter-example proves the claim:

Begin with all servers occupied by customers in service. The customer at server B_1 is blocked from entering node A. Then the customer at server C_1 is blocked from entering node B. Then the customer at server A_1 is blocked from entering node A. The resulting state digraph in Figure 13 has a weakly connected component with a sink. □

Lemma 2. *An absorbing deadlocked state arises at time t if $D(t)$ doesn't contain a sink vertex.*

Proof. A vertex with out-degree greater than zero represents an occupied server whose occupant has finished service and is blocked. If all vertices have out-degree greater than zero, then all servers are occupied by blocked individuals. A release at vertex X_a can only be triggered by one of X_a's descendents finishing service. As all servers are occupied by blocked individuals, no server can finish service, and so no server can release their occupant, implying an absorbing deadlocked state. □

7 Finding Knots

Here is an initial draft of an algorithm for dynamically finding a knot in the state digraph of the queueing network.

Theorem 2. *The existance of a knot in the state digraph is equivalent to the existance of a vertex with an out-edge but without a path to a sink.*

Proof. Consider a vertex X that has an out-edge, but does not have a path to a sink. All of X's descendants also do not have paths to sinks.

As the number of vertices in $D(t)$ are finite ($|V(D(t))| = \sum_{i=1}^{N} c_i$ for all $t \geq 0$), then all the possible paths leading out from X must either be part of a cycle of lead to a cycle. And so all paths from X must terminate in a knot. □

Keeping track of the list of sinks in D will be useful. Let's call this list sink_list. Initially all vertices of $D(0)$ are sinks, and belong to sink_list. At the point when a vertex gains an out-edge, that vertex is removed from sink_list. At the point when a vertex has an out-edge removed, if $\deg^{\text{out}} = 0$ then that vertex is added to sink_list. Therefore, at any one point we have knowledge of all the sinks in $D(t)$.

Now, the only action that can lead to a vertex transitioning from having a path to a sink and not having a path to a sink is adding an edge. Removing an edge of $D(t)$ will not cause deadlock, as removing an edge implies that a customer has moved, and a customer moving cannot cause deadlock. Therefore, we only need to check the vertex's paths to sinks when an edge is added.

8 Markov Chain Model

8.1 One Node Network

Consider the one node network with feedback loop shown in Figure 14. There is room for n customers to queue at any one time, customers arrive at a rate of Λ and served at a rate μ. Once a customer has finished service he rejoins the queue with probability r_{11}, and so exits the system with probability $1 - r_{11}$.

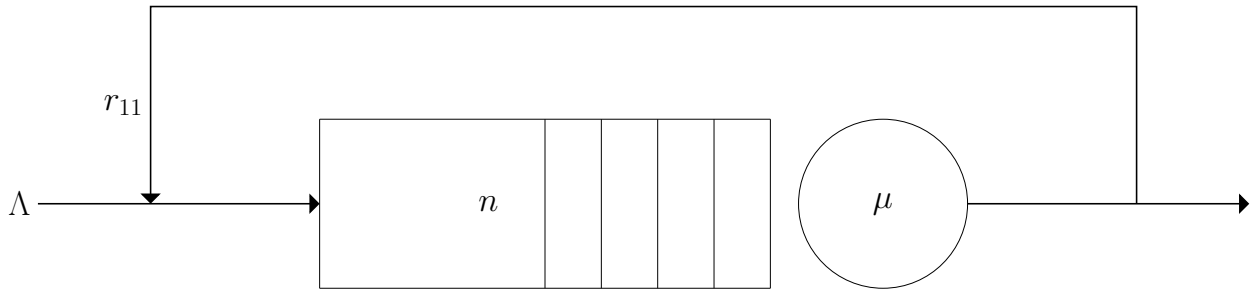


Figure 14: A one node queueing network.

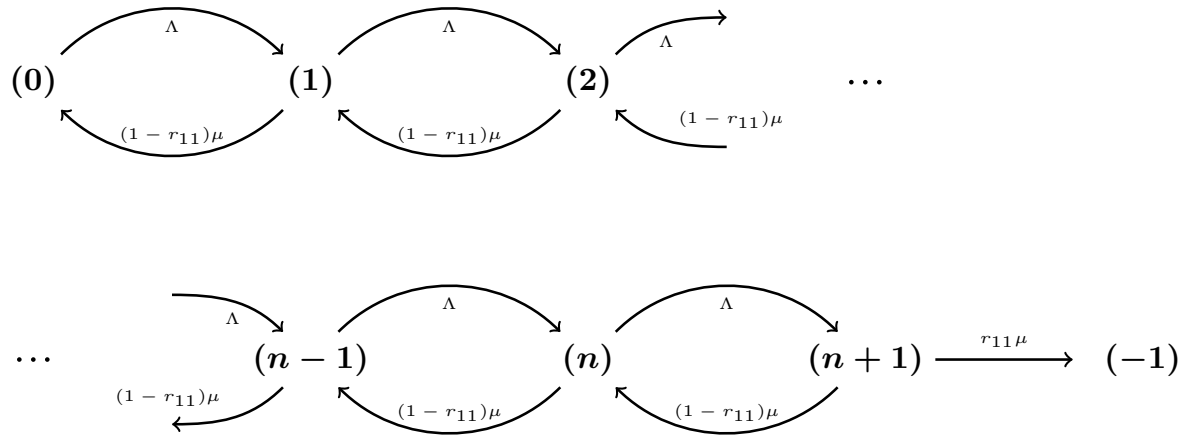


Figure 15: Markov chain of the one node system.

State space:

$$S = \{i \in \mathbb{N} | 0 \leq i \leq n+1\} \cup \{-1\}$$

Where i denotes number of individuals in service or waiting.

If we define $\delta = i_2 - i_1$ for all $i_k \in S | i_k \geq 0$, then the transitions are given by:

$$q_{i_1,i_2} = \begin{cases} \left.\begin{matrix} 0 & \text{if } i = n+1 \\ \Lambda & \text{otherwise} \end{matrix}\right\} & \text{if } \delta = 1 \\ (1 - r_{11})\mu & \text{if } \delta = -1 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$q_{i,-1} = \begin{cases} r_{11}\mu & \text{if } i = n+1) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

and

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

The Markov chain is shown in Figure 15.

8.2 Two Node Network without Self Loops

Consider the queueing network shown in Figure 16. 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}.

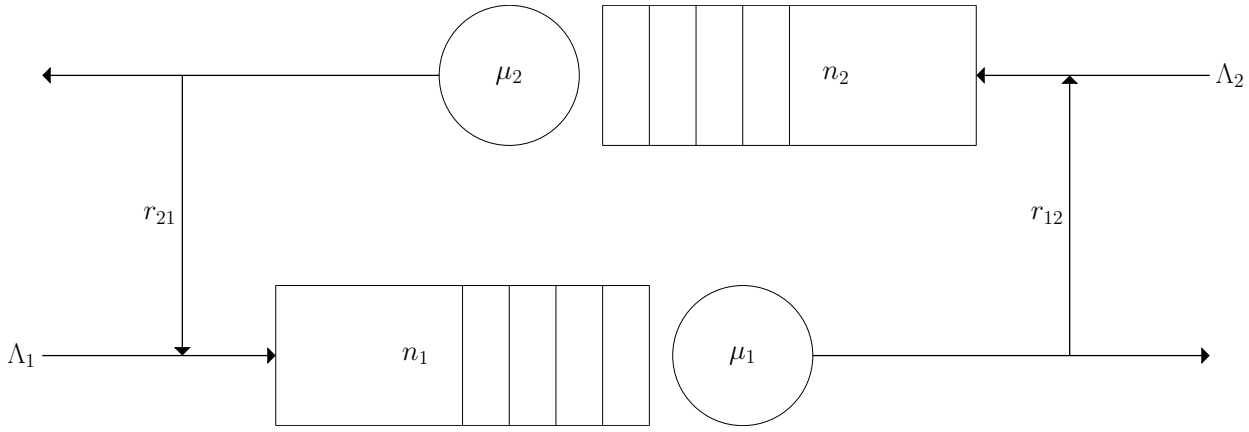


Figure 16: A two node queueing network.

- State space:

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

  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 $(-1,-1)$ denotes the deadlocked state.

If we define $\delta = (i_2, j_2) - (i_1, j_1)$ for all $(i_k, j_k), s \in S | i_k, j_k \geq 0$, then the transitions are given by:

$$q_{(i_1,j_1),(i_2,j_2)} = \begin{cases} \left.\begin{array}{rl} \Lambda_1 & \text{if } i_1 \leq n_1 \\ 0 & \text{otherwise} \end{array}\right\} & \text{if } \delta = (1,0) \\ \left.\begin{array}{rl} \Lambda_2 & \text{if } j_1 \leq n_2 \\ 0 & \text{otherwise} \end{array}\right\} & \text{if } \delta = (0,1) \\ \left.\begin{array}{rl} 0 & \text{if } j_1 = n_1 + 2 \\ (1-r_{12})\mu_1 & \text{otherwise} \end{array}\right\} & \text{if } \delta = (-1,0) \\ \left.\begin{array}{rl} 0 & \text{if } i_1 = n_1 + 2 \\ (1-r_{21})\mu_2 & \text{otherwise} \end{array}\right\} & \text{if } \delta = (0,-1) \\ \left.\begin{array}{rl} 0 & \text{if } j_1 = n_2 + 2 \\ r_{12}\mu_1 & \text{otherwise} \end{array}\right\} & \text{if } \delta = (-1,1) \\ \left.\begin{array}{rl} 0 & \text{if } i_1 = n_1 + 2 \\ r_{21}\mu_2 & \text{otherwise} \end{array}\right\} & \text{if } \delta = (1,-1) \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

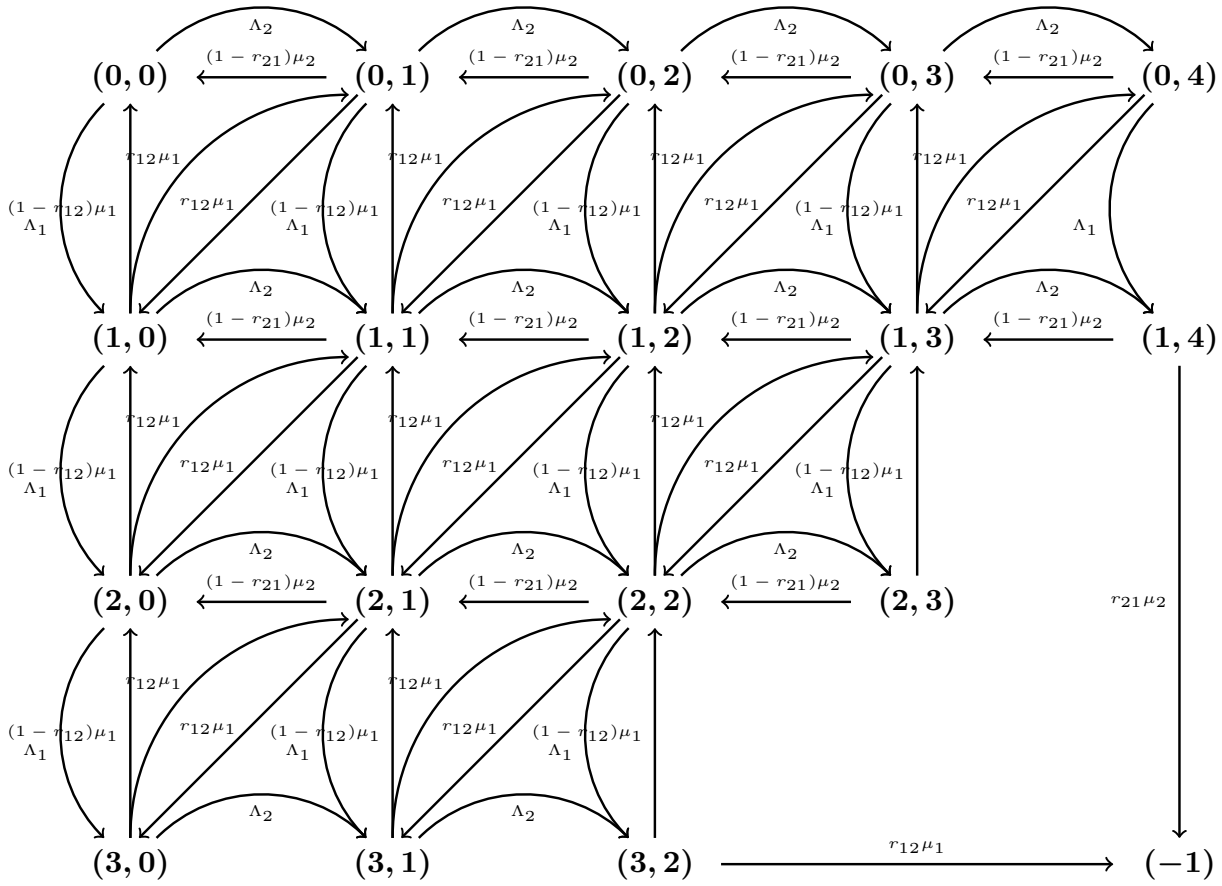


Figure 17: Markov chain of the two node system with $n_1 = 1$ and $n_2 = 2$.

$$q_{(i_1,j_1),(-1,-1)} = \begin{cases} r_{21}\mu_2 & \text{if } (i,j) = (n_1, n_2+2) \\ r_{12}\mu_1 & \text{if } (i,j) = (n_1+2, n_2) \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

and

$$q_{-1,s} = 0 \tag{6}$$

For $n_1 = 1$ and $n_2 = 2$, the resulting Markov chain is shown in Figure 17.

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

$$(I - T)^{-1} e \tag{7}$$

where e is a vector of 1s.

9 Resolving Deadlock

Once the system falls into a deadlocked state for the first time, that is the first transient deadlocked state since the last resolution, the simulated needs to automatically resolve the deadlock and allow services to resume again. This is not necassarily as simple as moving a blocked customer to his next node, as we need to conserve the numbers of customers at each service station. Closer inspection of the state digraph is required in order to find a way to resolve deadlock whilst conserving this property.

At deadlock, the service stations can be classified into the following three mutually exclusive categories:

- Nodes that are not deadlocked: These are nodes that do not contain any blocked individuals.
- Causation nodes, nodes causing deadlock: These are nodes where every server is occupied by a blocked individual, and there is at least one blocked individual waiting to enter that node who is directly or indirectly being blocked by an individual in this node.
- Affected nodes, nodes affected by deadlock: These are nodes containing at least one blocked individual who is directly or indirectly being blocked by an individual at a node that is causing deadlock, but is not classified as causing deadlock itself.

At the first instance of deadlock, there will only be one knot in $D(t)$. Let us denote the knot as K. The vertices of K correspond to servers. As there is no sink node, all vertices of K have an out-edge, and so all vertices in K contain a blocked individual. Therefore, there are no vertices in K belonging to nodes that are not deadlocked. All vertices of K correspond to servers of causation nodes, and a causation node has all its servers belonging to K. An affected node has servers belonging to the same weakly connected component as K, but does ot have servers in K.

When choosing which customer to move in order to resolve deadlock, we must be careful to conserve the number of customers at each service station. Causation nodes have full queues, and a customer may only be moved into a full queue if this causes another customer to simultaneously move from this node. Another complication arises due to the blocking mechanism used, in which those customers who have been blocked longer must be moved first. This property may have to be broken in order to ensure the conservation property is not. Assume that we have weighted the edges of the digraph with the time that they were created.

The following algorithm is proposed in order to resolve deadlock:

Find the knot K in $D(t)$
Find the cycle $C \in K$ whose average edge weight is minimum
Start at V_0
for V_i *in* C **do**
  Move the individual who is waiting to get to V_{i+1}
end
Redraw $D(t)$

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.