Introduction to Queueing Theory

Prof. N. Selvaraju
Department of Mathematics
Indian Institute of Technology Guwahati, India

Lecture - 33
Closed Jackson Networks, Convolution Algorithm

Hi and hello, everyone. What we have been seeing so far is the Closed Jackson Network or Gordon-Newell Network. We have seen the basic theory and how it is different or how one can analyze that based upon our analysis of the open Jackson network that we have done earlier. And we have seen an example.

Example.

- *The steady state joint probability distribution is given by*

$$p_{M-m,m} = \frac{1}{G(M)}\rho_1^{M-m}\rho_2^m, \qquad m = 0, 1, 2, \ldots, M.$$

We must find ρ_i *from* $\mu_i\rho_i = \sum_{j=1}^{k} \mu_j r_{ji}\rho_j$. *Here, the routing matrix* R *is* $R = \begin{pmatrix} q & 1-q \\ 1-p & p \end{pmatrix}$ *and hence the traffic equations becomes*

$$\mu_1\rho_1 = \mu_1 q\rho_1 + \mu_2(1-p)\rho_2$$
$$\mu_2\rho_2 = \mu_1(1-q)\rho_1 + \mu_2 p\rho_2$$

Since these equations are linearly dependent, as we know already, we can set one of the ρ_i*'s to* 1 *and solve for the other.*

Set $\rho_1 = 1$. *Then, from the second equation, we get* $\rho_2 = \dfrac{1-q}{1-p}\,\dfrac{\mu_1}{\mu_2}$.

In that example, one thing I just want to point out here is this particular point. If you look here, the traffic equation is given by $\mu_i\rho_i = \sum_{j=1}^{k} \mu_j r_{ji}\rho_j$, in terms of $rhoi$. But now, it may be a little awkward to use the same ρ_i because we have been using this ρ_i for denoting the utilization factors. In this form, if you directly look at everything properly like it will, it may be of that, but here it is not exactly equal to ρ_i of the actual utilization of the thing, but it is relative to that. Because we are setting, for example, in this particular case $\rho_1 = 1$, we are setting.

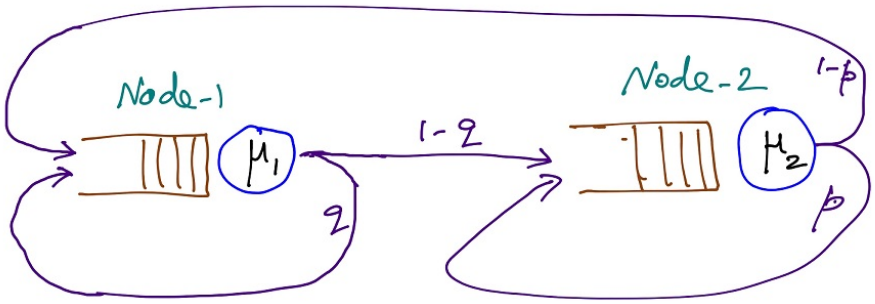
We know that may not be the case here in general for any network. But the ρ_2; then we are obtaining in terms of that. Suppose, if you set $rho_2 = 1$, then ρ_1 would be some other value. So, it is relative. So, this relative thing will be taken care of when actually you are computing the $G(M)$, when you are actually computing $p_{M-m,m}$ like it will not

make a difference.

But, if you want to interpret this ρ as the actual utilization, that is not correct. So, it can be less than 1 or more than 1; yeah, as you can see from $\rho_2 = \frac{1-q}{1-p}\frac{\mu_1}{\mu_2}$, for example, for certain values of p, q, μ_1, μ_2, like this may even be more than 1 or less than 1. But what we are saying is we are it is just relative to this 1, we are which we are fixing it. That is what we have to keep remembering.

So, this is not really the actual throughput or actual utilization. I mean throughput in the sense λ_i's corresponding ones, but this is the relative utilization is what this ρ_i. So, never interpret that as such is actual. So, that is why some books like they use a different notation for this ρ_i rather than using ρ_i as we are using it here. But from the context of a closed queuing network, you just have to keep in mind that this is not actually the actual utilization, but it is a relative utilization that we have here.

Let us look at one more example here, which is basically what we have in the text as well; the two-machine three-node closed queuing network because the example that we are going to see is going to come again and again. So, just pay attention to this because in the remaining complete portion, whether this algorithm that we are going to introduce, we are going to come back to this example only, and we are going to talk about its implementation aspects. So, that is why this particular example is important. The previous one we have given because, in a simple node situation, you may have more different; for example, this



network that we have, is a single server queue. So, for a single server, one can have much more quite neat results for various other quantities like what we have had here $\frac{G(M-1)}{G(M)}$, for example, these $\rho_2 \frac{G(M-1)}{G(M)}$ kinds of quantities that we have had. You can write more, but in general, it will be difficult to give that. But our concern is only obtaining the joint distribution. So, we will confine to that.

So, let us look at it here. This is what we call as a two-machine three-node Closed Jackson Network or Gordon-Newell Network.

Example. *[Two-Machine Three-Node Closed Network]*

- Two special-purpose machines are in operating condition and they need to be maintained in that position at all times. The machines break down according to an $Exp(\lambda)$ distribution.
  ▶ Call this operating node as node 1.

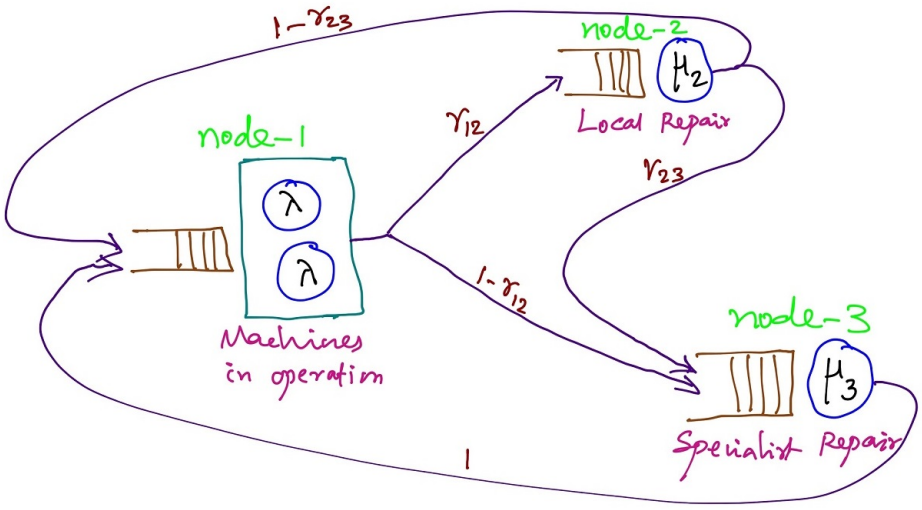
- *Upon failure, a machine*
  ▶ has a probability of r_{12} being repaired locally (node 2) by a single repairman with repair times following an $Exp(\mu_2)$ distribution, or
  ▶ must be repaired by the single specialist (node 3) with probability $1 - r_{12}$ who works according to an $Exp(\mu_3)$ distribution.

- After the service completion locally, the machine may require specialist attention with probability r_{23}, or return to operation with probability $1 - r_{23}$.

- After the special service (node 3), the unit always returns to operation $(r_{31} = 1)$.

- Here at node 1, the servers are machines, so $c_1 = 2$ with the mean service (or holding time) at node 1 is the mean time to failure of a machine. This means that $\mu_1 = \lambda$.

So, if you depict this diagrammatically



So you have really node 1, you have two machines which are in working condition which is with rate λ, it will fail. Now, once it fails with probability r_{12}, it goes to node 2, which will take an exponential amount of time for the local repair. Then after which, either it can move to node 3 for specialist repair, which will happen with probability μ_3, or it can come to the operating node with $1 - r_{23}$. Similarly, from here, once the machine fails, it can also go for specialist repair with this probability, but after repairing in the specialist repair facility, the machine will come to operating condition. So, this is with probability 1; you have this node. So, this is what we had here. Really, like there is no queuing happens here. Just for the completion sake, we have put this. But as you see, there are two servers or a maximum of 2; what you call 2 servers are there and two machines only, 2 customers only. So, both can be accommodated here. Whereas here, there could be a possibility that one is being repaired, the other is waiting for repair; one is being repaired other is waiting for the repair. But here it will be both. So, there is actually there is no queue. But we are just depicting it because we are allowing it, no problem. But there is no queue happens there actually. This is what the model is. Now, what do we have to do?

Example.

- The steady state joint probability distribution is given by

$$p_{n_1,n_2,n_3} = \frac{1}{G(2)} \frac{\rho_1^{n_1}}{a_1(n_1)} \rho_2^{n_2} \rho_3^{n_3}, \qquad n_i = 0, 1, 2; \quad i = 1, 2, 3,$$

where $a_1(n_1) = 1$ for $n_1 = 0, 1$ and $a_1(2) = 2$. We must find ρ_i from $\mu_i \rho_i = \sum_{j=1}^{k} \mu_j r_{ji} \rho_j$.

Here, the routing matrix R is

$$R = \begin{pmatrix} 0 & r_{12} & 1 - r_{12} \\ 1 - r_{23} & 0 & r_{23} \\ 1 & 0 & 0 \end{pmatrix}$$

and hence the traffic equations becomes

$$\begin{aligned} \lambda \rho_1 &= \mu_2 (1 - r_{23}) \rho_2 + \mu_3 \rho_3 \\ \mu_2 \rho_2 &= \lambda r_{12} \rho_1 \\ \mu_3 \rho_3 &= \lambda (1 - r_{12}) \rho_1 + \mu_2 r_{23} \rho_2 \end{aligned}$$

Example. Since these equations are linearly dependent, as we know already, we can set one of the ρ_i's to 1 and solve for the rest.

Set $\rho_2 = 1$. Then, from the second equation, we get $\rho_1 = \dfrac{\mu_2}{r_{12}\lambda}$. Substituting these two into the third equation, we get

$$\rho_3 = \frac{\lambda(1 - r_{12})}{\mu_3} \frac{\mu_2}{\lambda r_{12}} + \frac{\mu_2}{\mu_3} r_{23} = \frac{\mu_2 (1 - r_{12} + r_{12} r_{23})}{r_{12} \mu_3}.$$

We thus have the steady state solution for the closed network as

$$p_{n_1,n_2,n_3} = \frac{1}{G(N)} \left(\frac{\mu_2}{r_{12}\lambda} \right)^{n_1} \frac{1}{a_1(n_1)} \left(\frac{\mu_2 (1 - r_{12} + r_{12} r_{23})}{r_{12} \mu_3} \right)^{n_3}, \quad n_1, n_3 = 0, 1, 2.$$

The normalizing constant $G(N)$ can be obtained by summing p_{n_1,n_2,n_3} over all cases for which $n_1 + n_2 + n_3 = 2$.
♦ There are six cases in total: $(2, 0, 0), (0, 2, 0), (0, 0, 2), (1, 1, 0), (1, 0, 1), (0, 1, 1)$.

Now, like if I evaluate these quantities for each of these cases and I sum it up to express what is going to be my $G(N)$ here. So I can obtain the normalization condition appropriately. So, that is how you handle this. So, you have for this network. Now, for illustration purposes, for easy understanding, and then going forward as well, let us take some specific values for the parameters.

Example. *[Illustration]*

- *Assume* $\lambda = 2,\ \mu_2 = 1,\ \mu_3 = 3,\ r_{12} = \dfrac{3}{4},\ r_{23} = \dfrac{1}{3}$. Then, the joint distribution is

$$p_{n_1,n_2,n_3} = \frac{1}{G(N)} \left(\frac{2}{3}\right)^{n_1} \frac{1}{a_1(n_1)} \left(\frac{2}{9}\right)^{n_3},$$

  where $G(2)$ is computed to be

$$G(2) = \left(\frac{2}{3}\right)^2 \frac{1}{2} + 1 + \left(\frac{2}{9}\right)^2 + \frac{2}{3} + \frac{2}{3}\frac{2}{9} + \frac{2}{9} = \frac{187}{81} = 2.3086.$$

$\bar{n}$	(2,0,0)	(0,2,0)	(0,0,2)	(1,1,0)	(1,0,1)	(0,1,1)
$p_{\bar{n}}$	0.0962	0.4332	0.0214	0.2888	0.0642	0.0962

  So, once we obtain this distribution, then marginals, the mean number in the system, everything can be obtained from this. But there are certain facts; I mean, if these were the parameter values, then there are certain facts that you need to look at with respect to the values as you always look at.

- Only 9.62% of the time, both machines are operating.
- At least one machine available for 44.92% of the time.
- Performance not up to the mark. Decide how to improve!

So, this is with respect to these particular values, but what we are looking at is how we are getting this and how we are implementing this. So, that is what then you are obtaining here with respect to this particular example that we have here. So, this example, we will keep coming back and remember this particular slide where we have the distribution, as well as the probabilities, given. So, this part is what you will come back to again and again. In the previous example, because it was a two-node, the computation of the normalization constant, which is basically $G(N)$, was relatively easier. In this case, because there are two customers only and even though there are three nodes, things were not that difficult; a naive computation can give easily. But in practice, this is not that easy to compute. The main computational difficulty when you are trying to obtain the joint distribution of the number in the system in the whole network is connected with the evaluation of this $G(N)$. Up to this; the process is the same. Now, $G(N)$ is this way of evaluating by enumerating all such possibilities. Here it so happened that $n_1 + n_2 + n_3 = 2$ gave rise to only 6 states. So, you can evaluate all the 6 states what was going to be, and then you can compute this $G(2)$ directly. But it may not be that simple in general.

- In Gordon-Newell networks, the joint probability distribution is determined in terms of the normalization constant $G(N)$.
- In the examples considered so far, the 'naive computation' approach of calculating $G(N)$ was easy. But, this is not the case in general.

- For large N and k (i.e., for large networks), there are many possible ways to allocate N customers among the k nodes (it is actually $\binom{N+k-1}{k-1}$ ways which is of order N^{k-1}).
  So, you can imagine if N is large or k is large, what would be the number of states that might come.

  ▶ Calculations also become prone to numerical errors.

- Efficient algorithms are needed to make the computation of $G(N)$ easier (and less prone to numerical errors).

- Buzen (1973) developed an efficient algorithm to compute $G(N)$ recursively, using a total of Nk multiplications and Nk additions (single server case - which is a significant improvement).
  ▶ Very useful for larger networks.

  Again, we trust that it is very useful for larger values. In smaller networks, with smaller values of N and k, we have seen that it can be done very easily, it is not very complex, but for large networks, this is. So, what is this algorithm is, that is what we will see, how we can compute this $G(N)$.

- We will now describe Buzen's Algorithm or Convolution Algorithm.

  So, let us call this factor which is the product factor in the product form solution; whatever is the product term, let that factor be called as $f_i(n_i)$. So, here we are writing it for, in general, for a multi-server case. If you specialize this algorithm to a single-server case, then you can get some more expression, some more neat results as well for certain performance quantities of interest as well. But since that was not our interest mainly. So, we restrict ourselves to, I mean, or we take it in the generality of multi-server case. So, we retain this factor. By the way, this algorithm is also called as the convolution algorithm because the factor coming out is basically convolution form it will come into this case.

- Let $f_i(n_i) = \dfrac{\rho_i^{n_i}}{a_i(n_i)}$ where $a_i(n_i) = \begin{cases} n_i! & n_i < c_i \\ c_i! c_i^{n_i - c_i} & n_i \geq c_i \end{cases}$.
  Then, the normalization constant $G(N)$ can be written as

$$G(N) = \sum_{n_1+n_2+\cdots+n_k=N} \prod_{i=1}^{k} f_i(n_i).$$

- Now, define an auxiliary function

$$g_m(n) = \sum_{n_1+n_2+\cdots+n_m=n} \prod_{i=1}^{m} f_i(n_i). \qquad \text{(i.e., with } m \text{ nodes } \& \ n \text{ customers)}.$$

- Observe that we have $G(N) = g_k(N)$. We now set up a recursive scheme to calculate $G(N)$.

Now, what we will do is that the algorithm is basically aimed at the computation of $g_m(n)$ in a recursive way. So, that is a scheme that we have to figure out what is the recursive scheme that we can have.

- Take $g_m(n)$ and fix $n_m = i$. Then, we have

$$\begin{aligned} g_m(n) &= \sum_{i=0}^{n} \left(\sum_{n_1+\cdots+n_{m-1}+i=n} \prod_{j=1}^{m} f_j(n_j) \right) \\ &= \sum_{i=0}^{n} f_m(i) \left(\sum_{n_1+\cdots+n_{m-1}=n-i} \prod_{j=1}^{m-1} f_j(n_j) \right) \\ &= \sum_{i=0}^{n} f_m(i) g_{m-1}(n-i), \qquad n = 0, 1, \ldots, N. \end{aligned}$$

- Note from the above that $g_1(n) = f_1(n)$, for $n = 0, 1, 2, \ldots, N$, and $g_m(0) = 1$, for $m = 1, 2, \ldots, k$. These form the starting conditions.

- The above relationship of the auxiliary function can then be recursively used to calculate

$$G(N) = g_k(N).$$

- The algorithm is efficient for large networks.

We will see simple examples only where you may not see the power or utility of this algorithm, but in practice, this is quite useful for large networks. And there is nothing like 1 node, 1 customer, 2 customer network, which is much easier to even observe. You do not really need a lot of whole stuff of this kind of queuing analysis to be done, and only when you have a large number of systems that you have here.

- These functions also helps us in calculating the marginal distributions as well.

- Suppose that we want $p_i(n) = P\{N_i = n\}$.

  Let $S_i = n_1 + n_2 + \cdots + n_{i-1} + n_{i+1} + \cdots + n_k$. Then

$$\begin{aligned} p_i(n) &= \sum_{S_i=N-n} p_{n_1,n_2,\ldots,n_k} = \sum_{S_i=N-n} \frac{1}{G(N)} \prod_{i=1}^{k} f_i(n_i) \\ &= \frac{f_i(n)}{G(N)} \sum_{S_i=N-n} \prod_{j=1, j\neq i}^{k} f_j(n_j), \qquad n = 0, 1, 2, \ldots, N \end{aligned}$$

  In general, this may be cumbersome to compute. But for node k, the expression simplifies to

$$p_k(n) = \frac{f_k(n)}{G(N)} \sum_{S_k=N-n} \prod_{j=1}^{k-1} f_j(n_j) = \frac{f_k(n) g_{k-1}(N-n)}{G(N)}, \qquad n = 0, 1, 2, \ldots, N$$

  To find other marginals, permute the node of interest with k (requires resolving some of the functions $g_m(n)$).

If there is only one marginal distribution, then you can set that as the last node, and then you can do the computation. But if I want more than that, then this way of doing it would be difficult; there are efficient ways, but we are not getting into that. So, there are efficient ways, even with little modification to this process, that one can get the marginals. But what is the crux here, or what is the thing that you are taking away from here is that these quantities, that auxiliary functions that we defined, in the convolution algorithms or Buzen's algorithm, also help us to compute the marginal distribution. That is what you have to take. Now, let us see how this algorithm can be implemented for the example that we just did the two-machine three-node model.

Example. *[Two-Machine Three-Node Closed Network - Illustration - Revisited]*

- First, the factors $f_i(n_i)$ are

$$f_1(0) = 1, f_1(1) = \frac{2}{3}, f_1(2) = \frac{2}{9}, \;\; f_2(0) = f_2(1) = f_2(2) = 1, \;\; f_3(0) = 1, f_3(1) = \frac{2}{9}, f_3(2) = \frac{4}{81}.$$

So, once you obtain this, then your algorithms can start.

- *The $g_m(n)$'s are given by*

$$G(2) = g_3(2) = \sum_{i=0}^{2} f_3(i) g_2(2-i) = f_3(0)g_2(2) + f_3(1)g_2(1) + f_3(2)g_2(0)$$

and

$$g_2(2) = f_2(0)g_1(2) + f_2(1)g_1(1) + f_2(2)g_1(0)$$
$$g_2(1) = f_2(0)g_1(1) + f_2(1)g_1(0)$$

with the starting conditions

$$g_1(0) = f_1(0) = 1, g_1(1) = f_1(1) = \frac{2}{3}, g_1(2) = f_1(2) = \frac{2}{9}, \;\; g_m(0) = 1, m = 1, 2, 3.$$

Example.

- Calculations give us

$$g_2(1) = 1 \cdot \frac{2}{3} + 1 \cdot 1 = \frac{5}{3}$$
$$g_2(2) = 1 \cdot \frac{2}{9} + 1 \cdot \frac{2}{3} + 1 \cdot 1 = \frac{17}{9}$$
$$g_3(2) = 1 \cdot \frac{17}{9} + \frac{2}{9} \cdot \frac{5}{3} + \frac{4}{81} \cdot 1 = \frac{187}{81} = 2.3086.$$
$$[\textit{Also, } g_3(1) = f_3(0)g_2(1) + f_3(1)g_2(0) = 1 \cdot \frac{5}{3} + \frac{2}{9} \cdot 1 = \frac{17}{9}.]$$

$g_m(n)$	$m=1$	$m=2$	$m=3$
$n=0$	1	1	1
$n=1$	$\frac{2}{3}$	$\frac{5}{3}$	$\frac{17}{9}$
$n=2$	$\frac{2}{9}$	$\frac{17}{9}$	$\frac{187}{81}$

- This example (where the algorithm is not much simpler) is to only illustrate the algorithm. The efficiency will be evident when you consider larger networks.

Now, the tabular form we have kept here, just for easy reference and understanding of how this algorithm works. So, the first column is what is given by $g_1(n)$; this is the one, $g_1(n)$ is what is given by the first column. This is given by the initial condition, which is equal simply $f_1(n)$. Once you compute that, then you will get this one. Similarly, this one $g_m(0)$ will give you this first row which is also coming from the initial condition, then the rest of the table you have to start filling. Now, you know that this $5/3$ if I look at here, it is here it involved this $2/3$ and 1 which are basically these two quantities. So, these two quantities will determine this and for determining this, all these 3 quantities determined this.

If I have more, then these 4 would determine this $4th$ quantity, $5th$ quantity I have this all 5 quantities will determine the $5th$ quantity here if I have something more. But this first column, the first row, will be given by the starting condition, then you have to start filling these elements one by one by computing with this. So, you have to multiply this appropriately. So, this is as you see here, this 1 is multiplied by this 1, and this $2/3$ is multiplied with this 1; that is what you know that is coming from the other ones, f function, f_2 function. So, that is what you get here. So, basically, how will you fill it up? You will have this; you will have this to start with, then you start with this involving these two and the corresponding f. These three and the f, like this, you will start filling up this table. Once you get this, then you will get this column and this column. Remember, here; we need not compute this. If our final objective is to compute the last element, which is the $G(N)$, at least for the last column, you need not compute the intermediate values but the last one directly; you can use all the previous column values, and then you can compute this. That is what pretty much we have done. But this column also, this entry also we have computed in this particular case then we have plug it here.

The reason that it is good that you have the complete table is that the performance measures or if, for example, these are the single server network, I mean you can write down a nice expression for the utilization and other performance measures throughput and other performance measures everything in terms of these, m's. Recall, in one of the previous examples, we wrote certain probability as $G(m-1)/G(m)$ sort of thing. So, this is sort of your, $G(n)$, and this is sort of your $G(n-1)$. So, that is what, in such situations, these entries may be of some help, but if your interest is only the final entry from the previous column, you can directly get here, and then you can stop as we have computed here. So, this is what the algorithm is and how one can work it out.

Now, once I compute; so, this Convolution algorithm or the Buzen's algorithm is only for computation of this normalization constant which will be plugged now into your usual expression for the joint probability distribution, and once we have the priority distribution, then you can get the other performers measures of interest.

So, go back, like if you have seen, $G(N)$ is what we are computing it through that algorithm rather than computing it in $G(2) = \left(\frac{2}{3}\right)^2 \frac{1}{2} + 1 + \left(\frac{2}{9}\right)^2 + \frac{2}{3} + \frac{2}{3}\frac{2}{9} + \frac{2}{9} = \frac{187}{81} = 2.3086$. this manner. Now, once you plug it here, then this $p_{n_1,n_2,n_3} = \frac{1}{G(N)} \left(\frac{2}{3}\right)^{n_1} \frac{1}{a_1(n_1)} \left(\frac{2}{9}\right)^{n_3}$, will give you these probabilities, and then further things will go further.

So, that is what you know you have here. Of course, a little bit more can be said if it is everything is a single server or just specific cases. But our objective was to introduce this algorithm in computations so that you can; if you want to understand more, you can explore yourself from here, from this point onwards. We will stop here, and then we will come back with another algorithm in the next lecture.
Thank you. Bye.