

Implementing The Mean Value Algorithm for the  
Solution of Queueing Network Models

Herb Schwetman

Department of Computer Sciences

Purdue University

West Lafayette, IN 47907

CSD-TR-355

(revised February 15, 1982)

# Implementing The Mean Value Algorithm for the Solution of Queueing Network Models

*Herb Schwetman*

Department of Computer Sciences  
Purdue University  
West Lafayette, IN 47907

## ABSTRACT

The Mean Value Analysis (MVA) algorithm of Reiser and Lavenberg is a very simple technique for the solution of fairly complex queueing network models. This paper describes a computer program which uses the MVA algorithm to solve almost all known versions of closed queueing networks which have 'product form' solutions.

## Introduction

Queueing network models of computer systems are emerging as both a tool useful in analyzing and predicting system performance and as a topic of research [BCMP75]. Until 1978, the most popular technique for obtaining (exact) solutions of these networks was the convolution technique [ReKo75, BaBS77, BrBa80]. In 1978, Reiser and Lavenberg [ReLa78] introduced a new technique, based on a Mean Value Analysis, which could be used to solve these models. This MVA technique is simpler to describe and implement than the convolution technique. Also, the convolution technique usually required a step to obtain performance parameters which resulted in a numerically unstable algorithm. The MVA technique does not have such a step and is thought to be more stable [ChSa80].

The algorithm as presented by Reiser and Lavenberg had one major drawback, namely the storage required to store all of the intermediate values could become quite large. Many researchers have examined the MVA algorithm and have suggested [Zaho80] modifications which could drastically reduce the storage required. Also, Reiser and Lavenberg hinted at extensions to their algorithm which would enlarge the set of solveable models to include features handled by the convolution programs. Balbo [Balb79] and Bruell and Balbo [BrBa80] have published some of these extensions.

This paper first describes the original MVA algorithm, using the notation of Denning and Buzen [DeBu78], instead of the notation used by Reiser and Lavenberg. This description covers both single and multiple class models, with service centers of the four admissable types (first come/first served - FCFS, processor sharing - PS, infinite servers - IS, and last come/last served-preempt - LCFS). Next, a storage management technique is presented. Extensions to the original MVA algorithm permit solution of models with multi-server facilities, state dependent service rates, and jobs which switch between classes. The paper is complete in the sense that all equations and steps are presented, so that interested readers could implement these algorithms in their own programs. The paper concludes with three complete examples, so that other programs can be checked for accuracy.

### **The MVA Algorithm (Single Class)**

An example of the type of model we are working with appears in Figure 1. In this model, there are  $K$  service facilities (also called stations), and  $N$  jobs circulating through the network of facilities. When a job arrives at the  $i$ -th facility, it joins the queue of other jobs which are awaiting service. Depending on the service discipline of the facility, the waiting tasks receive enough use of the facility to satisfy each demand. These demands are characterized in one of the

following ways, depending on the service discipline of the facility:

1. If the service discipline is FCFS, then the distribution of service intervals is negative exponential, with mean  $S_i$ .
2. If the service discipline is PS, IS or LCFS, then the distribution can be any distribution with a rational Laplace transform with mean  $S_i$ .

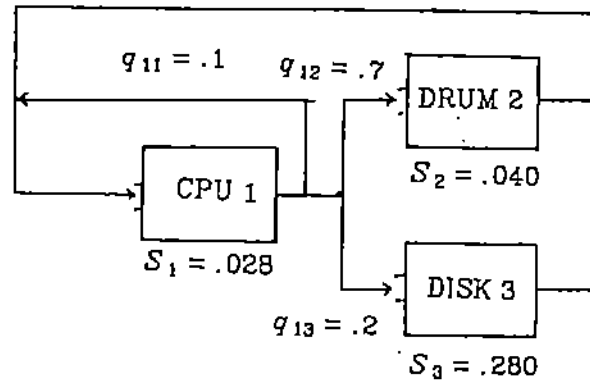


Figure 1

Example of Queueing Network Model [Buze73]

When a job leaves a facility, it travels to another facility. The probability of a job leaving station  $i$  and going to station  $j$  is  $q_{ij}$ . In subsequent sections, we will deal with models with several distinct classes of jobs, with facilities with multiple servers and with service demands which can depend on the number of jobs at the facility. A set of numbers,  $V_i$ , for  $1 \leq i \leq K$ , are also needed. These are any set of solution values of the set of equations

$$V_j = \sum_{i=1}^K V_i q_{ij} \quad (1 \leq j \leq K).$$

These  $V_i$ 's, when suitably normalized, can be interpreted as the number of visits to each facility per job, hence the use of the term visit ratios. We should note

that while a central server model [Buze73] is shown in Figure 1, nothing in the MVA algorithm limits the arrangement of the facilities of the model. In particular, in our formulation, different routings for jobs traveling between facilities are handled by different transition probabilities ( $q_{ij}$ ) or, equivalently, by different visit ratios ( $V_i$ ).

When a model is solved, we obtain several measures of the performance of the system; these performance variables include, for station  $i$ ,  $U_i$ , the facility utilization,  $X_i$ , the facility throughput rate,  $\bar{n}_i$ , the mean queue length, and  $W_i$ , the mean job response time. We let the index 0 denote the "outside world" with respect to the system. Thus,  $X_0$  is the system job throughput rate. We sometimes affix an argument  $N$  to a performance measure (e.g.  $U_i(N)$ ), to emphasize that it is computed for a particular level of multiprogramming. All of this notation is summarized in Table 1.

$K$	number of facilities
$S_i$	mean service interval, facility $i$
$q_{ij}$	probability of $i$ to $j$ transition
$V_i$	visit ratios
$N$	number of active jobs
$U_i$	utilization
$X_i$	throughput rate
$\bar{n}_i$	mean queue length
$W_i$	mean response time
$X_0$	system job throughput rate

Table 1

Notation

The MVA algorithm uses the visit ratios ( $V_i$ ) plus the mean service intervals ( $S_i$ ) to produce values of the performance variables for any number of active jobs ( $N$ ). The steps of the algorithm are as follows:

$$\bar{n}_i(0) = 0 \quad (1 \leq i \leq K) \tag{0}$$

then for each facility  $i$

$$W_i(N) = \begin{cases} S_i [1 + \bar{n}_i(N - 1)] & \text{if FCFS, PS or LCFS} \\ S_i & \text{if IS} \end{cases} \quad (1)$$

$$X_0(N) = \frac{N}{\sum_{i=1}^K V_i W_i(N)} \quad (2)$$

$$X_i(N) = X_0(N) V_i \quad (3)$$

$$\bar{n}_i(N) = X_i W_i(N) \quad (4)$$

By repeating steps 1-4 for values of  $N$  ranging from 1 up to the desired level of multiprogramming, the solution is obtained.

The derivations of these formulae are available in several places ([ReLa78], [Balb79], [BuDe80], and [BrBa80], to name a few). Step (1) can be derived, but it also can be justified on the intuitive grounds that for FCFS queues, the response time experienced by an arriving job at facility  $i$  is composed of the service times for all of the jobs which arrived earlier plus the service time of the new job ( $S_i \bar{n}_i(N-1) + S_i$ ). For PS queues, the response time is simply the service time degraded by the number of jobs at the queue. ( $S_i [1 + \bar{n}_i(N-1)]$ ). Equation (2) is the application of Little's Law for the system as a whole. Equation (3) is one form of the definition of  $V_i$ . Equation (4) is Little's Law applied to each individual queue.

The key to the MVA algorithm is the use of equation (1) to obtain the facility response time with  $N$  jobs active from the mean queue length with  $N-1$  jobs active. This follows from a relationship which has been known (e.g. [DeBu78] had the equivalent expression  $\bar{n}_i(N) = U_i(N) [1 + \bar{n}_i(N-1)]$  on page 253). This relationship was proved as a theorem by Sevcik and Mitriani in [SeMi79]. This theorem states that for networks with product form solutions, the queue length distribution at a facility when a job arrives is the same as the usual queue length distribution with one fewer jobs active in the system. The operational analysis

version of this theorem is proved in [BuDe80].

An complete example of this algorithm as applied to a simple model appears in the section containing the examples.

### The MVA Algorithm (Multiple Classes)

Queueing network models can also be solved for cases with several classes of jobs. In this version, a class is a set of jobs with the same branching probabilities (and consequently the same visit ratios) and the same mean service intervals. We use  $R$  to denote the number of job classes and  $S_{i\tau}$  and  $V_{i\tau}$  to denote the mean service interval and visit ratio at facility  $i$  for class  $\tau$  jobs respectively. Jobs in different classes can have different branching probabilities and, for PS, IS and LCFS facilities, different mean service intervals; for FCFS facilities, the mean service intervals for all classes must be equal (i.e.  $S_{i1}=S_{i2}=\dots=S_{iR}$ , if facility  $i$  is FCFS). All of the performance variables have both class and global values which will be denoted, for example,  $U_{i\tau}$  (utilization of facility  $i$  for class  $\tau$  jobs) and  $U_i$  (utilization at facility  $i$  for all jobs) respectively.

The branching probabilities for multiple class models can be extended to denote the switching from one class to another as a job travels from one facility to another. The notation  $q_{i\tau,js}$  signifies a job of class  $\tau$  which leaves station  $i$  and joins class  $s$  at station  $j$  with probability  $q_{i\tau,js}$ . In the initial version of the multiple class MVA algorithm, we omit this class switching feature; this feature is added in a later section.

For the multiple class model, we have to extend to concept of number of jobs in the system to the number of jobs in each class. In order to do this, we adopt a vector notation  $\mathcal{N} = (N_1, N_2, \dots, N_R)$ , where  $N_r$  denotes the number of jobs in class  $R$ .  $\mathcal{N}$  will be called a job configuration or mix. In the multiple class version of the MVA algorithm, we need to denote a "class  $\tau$  predecessor

configuration"; this will be denoted  $(\mathcal{N} - 1_r)$  and is defined to be  $(N_1, \dots, N_r - 1, \dots, N_R)$ .

The  $V_{ir}$  parameters are now obtained as the solutions to the set of equations

$$V_{js} = \sum_{i,r} V_{ir} q_{ir,js} \quad (1 \leq i, j \leq K, 1 \leq r, s \leq R)$$

Thus, given a model, we can use the multiple class MVA algorithm to obtain solutions for any configuration of jobs  $(\mathcal{N})$  as follows:

$$\bar{n}_i(0, \dots, 0) = 0 \quad (1 \leq i \leq K) \tag{5}$$

then for each class  $r$  and each facility  $i$

$$W_{ir}(\mathcal{N}) = \begin{cases} S_{ir} [1 + \bar{n}_i(\mathcal{N} - 1_r)] & \text{if FCFS, PS or LCFS} \\ S_{ir} & \text{if IS} \end{cases} \tag{6}$$

$$X_{0r}(\mathcal{N}) = \frac{N_r}{\sum_{i=1}^K V_{ir} W_{ir}(\mathcal{N})} \tag{7}$$

$$X_{ir}(\mathcal{N}) = X_{0r}(\mathcal{N}) V_{ir} \tag{8}$$

$$\bar{n}_i(\mathcal{N}) = \sum_{r=1}^R X_{ir}(\mathcal{N}) W_{ir}(\mathcal{N}) \tag{9}$$

In the multiple class case, a simple enumeration of the job configurations is no longer sufficient, in contrast to the single class algorithm. The problem now is that while we are solving a configuration  $\mathcal{N} = (N_1, \dots, N_R)$ , we need the global mean queue lengths at each station for the  $R$  predecessor configurations  $\mathcal{N} - 1_r$ , for  $1 \leq r \leq R$ . Thus, in order to have an efficient algorithm, we need to be certain that the queue lengths for each predecessor configuration are already computed and available as configuration  $\mathcal{N}$  is being solved. Since the number of configurations leading to a final configuration can be quite large, efficient enumeration and storage management schemes are essential. As soon as a good scheme is developed, we can then present an example of a multiple class queue-

ing network model.

### Enumerating Job Configurations

The problem of enumerating job configurations and storing intermediate results is illustrated in Figure 2. In Figure 2, we show the intermediate configurations required to solve a two class model for a final configuration of (3,2).

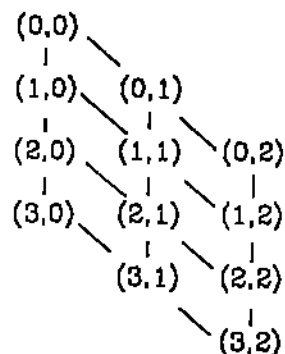


Figure 2

#### Configurations Required to Solve (3,2) Model

We can see, for example, that in order to solve the model for configuration (2,1), we need to use the mean queue lengths for configurations (2,0) (required for class 2) and (1,1) (required for class 1).

Examination of Figure 2 leads to the conclusion that there could be more than one way of enumerating the configurations so that all predecessor configurations are already computed for every configuration [Brum81]. The most obvious technique is to process each column proceeding from left to right. Another, equivalent scheme is to proceed from top to bottom along the diagonals. Still, another approach is to process each horizontal row, proceeding from top to bottom. In this approach, we never have to save more than the one preceding row. Brumfield [Brum81] has found that none of these three schemes

CJ  
11

is optimal for all cases.

In the horizontal-row scheme, we notice that each row corresponds to a level of multiprogramming and consists of all legal combinations of  $R$  job classes which add up to that level of multiprogramming. Notice that at some point, the number of configurations to be solved begins to decrease; this occurs as the maximum number of jobs for some class is reached.

With this scheme, we need first to generate every legal configuration at each successive level of multiprogramming. As the algorithm calculates facility queue lengths at one level, these must be saved for use in calculating response times at the next level. Furthermore, we need to save only queue lengths for one level back; i.e. at configuration  $N$ , we need only  $(N-1)$ , for some values of  $\tau$ . In the scheme being discussed, we calculate only queue lengths which will be required at the next level. The algorithm for generating successive configurations is shown in Figure 3.

$R$	Number of classes
$\text{max} = (N_1, \dots, N_R)$	Final configuration
$M = \sum_{t=0}^R N_t$	Final level of multiprogramming
$N = (0, \dots, 0)$	Current configuration
$\text{const} = (0, \dots, 0)$	Current constraints

```
.....  
for k = 1 step 1 until M do begin  
    for i = 1 step 1 until R do begin  
        N[i] = 0;  
        const[i] = min(k, max[i]);  
    end;  
    mp = 0;  
    while N[R] ≤ const[R] do begin
```

```
    fillconfiguration;  
    solve;  
    if N[R] = max[R] then stats;  
    advanceconfiguration;  
end;  
- end;
```

Figure 3a

```
procedure fillconfiguration;  
begin  
    x = k - mp;  
    i = 1;  
    while x > 0 and i ≤ R do begin  
        N[i] = min(x, const[i]);  
        x = x - N[i];  
        mp = mp + N[i];  
        i = i + 1;  
    end;  
end;
```

Figure 3b

```
procedure advanceconfiguration;  
begin  
    i = 1;  
    sw = true;  
    while sw do  
        if (mp = k or N[i] = const[i]) and i < R then begin
```

```
        mp = mp - N[i];
        N[i] = 0;
        i = i + 1;
    end
else begin
    N[i] = N[i] + 1;
    mp = mp + 1;
    sw = false;
end;
end;
```

Figure 3c

#### Algorithm to Generate All Configurations

The algorithm in Figure 3 generates only the required configurations for successive levels of multiprogramming. The remaining problem is to store the calculated queue lengths so that they can be easily retrieved as needed. The technique we use is to consider a configuration of jobs as an index into an array of queue lengths. These queue lengths are stored at the locations given by the index as they are calculated. At the beginning of the next pass or level of multiprogramming, these are copied into another array. Then as predecessor queue lengths are needed, they are accessed using the same technique.

As an example, assume that we have three job classes ( $R = 3$ ) and the final configuration is (3,2,1). Since the configuration at each level of multiprogramming,  $k$ , must sum to  $k$ , we need to use only two subscripts to uniquely locate each configuration. Figure 4 illustrates the scheme used. In Figure 4, we arbitrarily selected the right-most two subscripts (job class populations) as the

index or locator for that configuration.

	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$
0	100	200	300			
1	010	110	210	310		
2		020	120	220	320	
3	001	101	201	301		
4		011	111	211	311	
5			021	121	221	321

Figure 4

Successive Configurations for (3,2,1)

In the program, these were stored in a linear array, and the addressing calculation is done only once for each class for each level of multiprogramming. Also, if the final configuration is arranged so that the class with the largest number of jobs is the first class, there will be a minimal amount of wasted space in this array. In Figure 4, it can be seen that for  $k = 4$  and  $N = (2,1,1)$ , the three required predecessor configurations are  $(N - 1_1) = (1,1,1)$ ,  $(N - 1_2) = (2,0,1)$ , and  $(N - 1_3) = (2,1,0)$ . Furthermore, each of these can be located in the  $k = 3$  column using as locators (1,1), (0,1), and (1,0) respectively.

In Figure 3, the routine "solve" was not described. This is a routine which implements the MVA algorithm. The inputs include K, the number of facilities, R, the number of classes,  $N$ , the current configuration of jobs,  $\{V_{ir}\}$  and  $\{S_{ir}\}$ , the visit ratios and mean service intervals respectively, and  $\{\bar{n}_i(N - 1_r)\}$ , the queue lengths for the predecessor configurations. Figure 5 illustrates the kind of routine which is required. Figure 6 shows the use of the output from "solve" to produce some performance statistics.

procedure solve;

begin

for r := 1 step 1 until R do begin

```
sum := 0;
for i := 1 step 1 until K do begin
    if type[i] = IS then  $W_{ir} := S_{ir}$ 
    else  $W_{ir} := S_{ir} * (1 + \bar{n}_i (N - 1_r))$ ;
    sum = sum +  $V_{ir} * W_{ir}$ ;
end;
 $W_{0r} = sum$ ;
 $X_{0r} := N_r / W_{0r}$ ;
end;
for i := 1 step 1 until K do begin
    sum := 0;
    for r := 1 step 1 until R do
        sum := sum +  $X_{0r} * V_{ir} * W_{ir}$ ;
     $\bar{n}_i := sum$ ;
end;
end;
```

Figure 5

### Description of MVA Solve Routine

```
procedure stats;
begin
    for r := 1 step 1 until R do
        for i := 1 until K do begin
             $W_{ir} :=$  from solve;       $W_{ir}$  from solve
             $X_{ir} := X_{0r} * V_{ir}$ ;       $X_{0r}$  from solve
             $U_{ir} := S_{ir} * X_{ir}$ ;
```

```
       $\bar{n}_{tr} := X_{tr} * W_{tr};$   
end;  
  
end;
```

Figure 6

### Calculating Performance Statistics

#### Class Switching

In queueing network models of a more general form, jobs are able to switch between job classes. This class switching feature is required if certain types of systems are to be accurately modeled. The system in Figure 7 is an example of this [BCMP75]. In this model, it is necessary to define two job classes, with jobs switching between these classes. Referring to Figure 7, notice that jobs arriving from the terminals at the drum must depart to the CPU, while jobs arriving at the drum from the CPU must depart to the terminals. This can be accomplished with the branching probabilities  $q_{11,21} = 1$ ,  $q_{21,32} = 1$ ,  $q_{32,22} = 1$ , and  $q_{22,11} = 1$ .

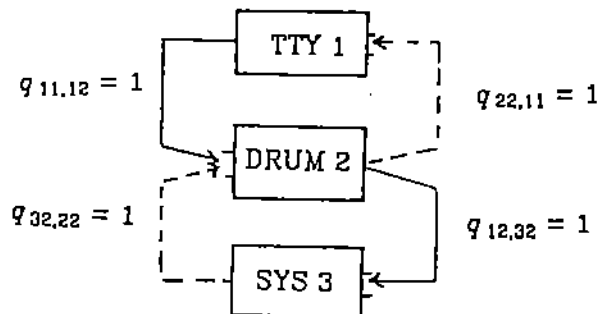


Figure 7

#### Model with Class Switching

In the versions of queueing network models discussed so far, we were able to assume that the number of jobs in each class was known and constant. Now,

with class switching, the number of jobs in a class is not known and constant. The method used for such models is to form "superclasses" from the classes; all classes in a superclass have jobs switching only between classes in the superclass [BrBa80]. These superclasses are formed so that the number of jobs in a superclass does remain constant and is known. Our solution technique for models with class switching consists of four steps:

1. Form superclasses (made up of classes among which jobs switch),
2. Parameterize these superclasses (i.e. determine visit ratios and service intervals for elements of each superclass),
3. Use the MVA algorithm to obtain solutions for these superclasses, and
4. Obtain solutions for the jobs in the original classes.

Assuming that we are given service intervals and visit ratios for the original classes,  $\{S_{ir}\}$  and  $\{V_{ir}\}$ , and the descriptions of the  $Q$  superclasses,  $\{SC_q$ , for  $1 \leq q \leq Q\}$  where  $SC_q$  consists of one or more jobs classes, we can parameterize the model based on superclasses as follows [Balb79], [BrBa80]:

1. The visit ratios,  $V_{iq}^*$  are given by

$$V_{iq}^* = \sum_{r \in SC_q} V_{ir}$$

2. Some auxiliary proportionality constants are:

$$\alpha_{ir} = \frac{V_{ir}}{V_{iq}^*} \text{ for } r \in SC_q$$

3. The service intervals are

$$S_{iq}^* = \sum_{r \in SC_q} \alpha_{ir} S_{ir}$$

4. The populations of the superclasses are

$$N_q^* = \sum_{r \in SC_q} N_r$$

Given these superclass parameters, we can then apply the MVA algorithm of

the preceding sections to obtain solutions for each superclass, namely  $\bar{n}_i^*(N - 1_q)$ , and  $X_0^*(N)$ . With these solutions, we can then calculate the class solution values as follows: for  $\tau \in SC_q$

$$W_{\tau}(N) = \begin{cases} S_{\tau} [1 + \bar{n}_i^*(N - 1_q)] & \text{if FCFS, PS, or LCFS} \\ S_{\tau} & \text{if IS} \end{cases} \quad (10)$$

$$X_{\tau}(N) = \alpha_{\tau} X_0^*(N) V_{iq}^* \quad (11)$$

$$\bar{n}_{\tau}(N) = X_{\tau}(N) W_{\tau}(N) \quad (12)$$

$$U_{\tau}(N) = S_{\tau} X_{\tau}(N) \quad (13)$$

We should point out that this approach requires all of the mean queue lengths for the next-to-the-last level of multiprogramming, to obtain  $W_{\tau}(N)$ , (equation 10). An alternative approach is to calculate  $W_{\tau}(N)$  at the same time that  $W_{iq}^*(N)$  for each superclass is calculated. The choice of approaches would depend on the procedure being used to enumerate all of the job classes and levels of multiprogramming. More specifically, if the queue lengths for  $(N - 1_q)$  are being stored anyway, then the approach using equation (10) should be used; otherwise, this alternative approach is probably preferred.

### Load Dependent Service Rates

The class of queueing networks being considered can include stations with service rates which depend on the number of jobs at the station. While it is possible to have these rates depend on the number of jobs in each class [BCMP75], we will restrict our implementation to rates which depend only on the total number of jobs (i.e. all classes) at the station [ReLa78]. More specifically, we assume that there is a fundamental service rate,  $\mu_{\tau} = \frac{1}{S_{\tau}}$ , and a set of multipliers (called the capacity function) for station  $i$ ,  $C_i(n_i)$ ,  $n_i = 1, 2, \dots, |N|$ , where

$\mu_{ir}(n_i) = \frac{1}{S_{ir} C_i(n_i)}$  is the service rate for class  $r$  jobs at station  $i$  when  $n_i$  jobs are present. Notice that if  $\mu_{ir}(2) = 2 \mu_{ir}$ , i.e. if the service rate doubles when two jobs are present, then  $C_i(2) = \frac{1}{2}$ .

In order to calculate waiting times for this type of station, we need to define  $P_i(n_i | N)$ , the probability of having  $n_i$  jobs at station  $i$  when the job mix is  $N$ . Given this marginal probability, we can then modify the calculation of  $W_{ir}(N)$  in the MVA algorithm as follows:

$$W_{ir}(N) = \sum_{j=1}^{|N|} j S_{ir} C_i(j) P_i(j-1 | N-1_r). \quad (14)$$

In order to obtain  $P_i(j | N)$ , we must perform some additional calculations:

$$P_i(j | N) = \sum_{r=1}^R X_{ir}(N) S_{ir} C_i(j) P_i(j-1 | N-1_r) \quad \text{for } j = 1, \dots, |N|. \quad (15)$$

$$P_i(0 | N) = 1 - \sum_{j=1}^{|N|} P_i(j | N) \quad (16)$$

These additional computations have two effects on the algorithm: (1) they can cause a significant increase in the amount of storage required (for the  $P_i(j | N)$ 's), and (2) equation (16) can introduce numerically erroneous results into all of the succeeding computations [ChSa80]. An obvious symptom of trouble is the appearance of negative values for  $P_i(0 | N)$ , but this may not always happen. An obvious correction for this symptom is to force  $P_i(0 | N)$  to be 0, if this occurs. The effects of this "correction" have not been analyzed, so this "cure" can only be classed as a heuristic at this time [BrBa80][BuDe80].

It should also be noted that calculation of device utilizations for servers with load dependent service rates can not be done as in the earlier sections. These can be computed from the state probabilities: e.g.,

$$U_i(N) = 1 - P_i(0 | N).$$

### Multiple Server Stations

A station with multiple servers can be realized in the framework of queuing network models as a single server station with a load dependent service rate, as follows:

let  $d_i$  be the number of servers at station  $i$ ; then

$$\mu_{ir}(n_i) = \begin{cases} n_i \mu_{ir} & \text{if } n_i \leq d_i \\ d_i \mu_{ir} & \text{if } n_i > d_i \end{cases}$$

The multipliers of the above section then become

$$G_i(n_i) = \begin{cases} \frac{1}{n_i} & \text{if } n_i \leq d_i \\ \frac{1}{d_i} & \text{if } n_i > d_i \end{cases}$$

While the equations of the previous section could be used to solve multiple server stations, the special form of these rate functions can be exploited to develop an algorithm which requires less storage [ReLa78]. The resulting formulae are:

$$W_{ir}(N) = \frac{S_{ir}}{d_i} \left[ 1 + \bar{n}_i(N - 1_r) + \sum_{j=0}^{d_i-2} (d_i - j) P_i(j | N - 1_r) \right], \quad (17)$$

and

$$P_i(j | N) = \frac{1}{j} \sum_{r=1}^R X_{ir}(N) S_{ir} P_i(j-1 | N - 1_r), \quad \text{for } j=1, \dots, d_i-1 \quad (18)$$

$$U_i(N) = \sum_{r=1}^R X_{ir}(N) S_{ir} \quad (19)$$

$$P_i(0 | N) = 1 - \frac{1}{d_i} \left[ U_i(N) + \sum_{j=1}^{d_i-1} (d_i - j) P_i(j | N) \right]. \quad (20)$$

The amount of additional storage required for stations with multiple servers is fairly modest, if the number of such stations and the number of servers per stations are modest. Unfortunately, the numerical problem

mentioned in the section on load dependent service rates is still present (in equation 20). For this type of server, the device utilizations are computed as part of the calculation.

### Open and Mixed Networks

Open networks are networks in which all jobs arrive at the system from an external source with a specified mean arrival rate,  $X_{0r}$ , for class  $r$  jobs. Since equilibrium must be maintained, all such jobs must eventually depart to an external sink, and the system throughput rate must also be  $X_{0r}$  for these classes of jobs. Networks with both open and closed classes of jobs are said to be mixed networks [BCMP75]. Since each of the stations in an open network is an M/M/1 queue (because of the assumptions stated earlier), standard techniques can be used to solve these networks. More specifically, given  $\{S_{ir}\}$  and  $\{V_{ir}\}$  as before, and  $\{X_{0r}\}$ , the arrival rate for class  $r$  jobs, where  $r$  denotes an open class, we can solve open networks with single server, load independent stations as follows:

$$X_{ir} = X_{0r} V_{ir}, \quad (21)$$

$$\rho_{ir} = X_{ir} S_{ir} \quad (22)$$

$$\rho_i = \sum_{r=1}^R \rho_{ir} \quad (23)$$

$$W_{ir} = \begin{cases} \frac{S_{ir}}{1 - \rho_i} & \text{if FCFS, PS, or LCFS} \\ S_{ir} & \text{if IS} \end{cases} \quad (24)$$

$$\bar{n}_{ir} = X_{ir} W_{ir} \quad (25)$$

$$U_{ir} = \rho_{ir} \quad (26)$$

Notice that for the network to be in equilibrium,  $\rho_i$  for each FCFS, PS, or LCFS station must be less than 1.

In a mixed network, we let  $\mathcal{N}$  denote the mix of jobs in the closed classes. Then  $\mathcal{N} = (\Omega)$  denotes the network with only jobs from the open classes present; the starting values for the MVA algorithm for mixed networks are those given above; i.e.

$$\bar{n}_i(\Omega) = \sum_{\tau \text{ open}} \bar{n}_{i\tau}. \quad (27)$$

For  $\mathcal{N}$  varying from  $(\Omega)$  up to the final configuration of jobs from the closed classes, we have, for each closed class  $s$  and each open class  $\tau$ ,

$$W_{is}(\mathcal{N}) = S_{is} [1 + \bar{n}_i(\mathcal{N} - 1_s)] \quad (28)$$

$$X_{Os}(\mathcal{N}) = \frac{N_s}{\sum_{i=1}^K V_{is} W_{is}(\mathcal{N})} \quad (29)$$

$$X_{is}(\mathcal{N}) = X_{Os}(\mathcal{N}) V_{is} \quad (30)$$

$$\bar{n}_{iCL}(\mathcal{N}) = \sum_{s \text{ closed}} X_{is}(\mathcal{N}) W_{is}(\mathcal{N}). \quad (31)$$

$$W_{i\tau}(\mathcal{N}) = \frac{S_{i\tau}}{1 - \rho_i} [1 + \bar{n}_{iCL}(\mathcal{N})] \quad (32)$$

$$\bar{n}_{iOP}(\mathcal{N}) = \sum_{\tau \text{ open}} X_{i\tau} W_{i\tau}(\mathcal{N}) \quad (33)$$

$$\bar{n}_i(\mathcal{N}) = \bar{n}_{iCL}(\mathcal{N}) + \bar{n}_{iOP}(\mathcal{N}). \quad (34)$$

These equations depend on the fact that the  $X_{i\tau}$ 's (the throughputs of the open classes) must remain constant for all  $\mathcal{N}$ , in effect a forced job flow rate at each station for jobs in open classes. Equation (32) can be interpreted as the mean waiting time for a job in an open class plus the delay caused by jobs in the closed classes. These equations can be derived from similar equations given in

[ChSa80].

### **Examples**

To illustrate the operation of the MVA algorithm, we include three examples. The first consists of a system with three stations and two job classes. The second illustrates class switching, and the third, mixed (open and closed) network models.

#### **Example 1 - Multiple Classes**

The first example is taken from an article by Reiser [Reis76]. It represents a system with two classes of users (labeled APL and IMS respectively). The APL users (class 1) have a mean think time at their terminals of 1.0 second and a mean service time at the system of 0.025 sec. The IMS users have a mean think time of 15.0 sec and a service time of 0.500 sec. The two groups of terminals are represented by two devices with IS scheduling (no delay) and the system is a single server device with PS scheduling. Figure 8 is a diagram of this model; Table 2 lists the values of the input parameters; Table 3 show the solution steps required to solve the model for 2 APL customers and 1 IMS customer; and Table 4 show the results (the values of the usual performance variables). Table 5 shows the results for 15 APL customers and 5 IMS customers and can be compared with the results given by Reiser.

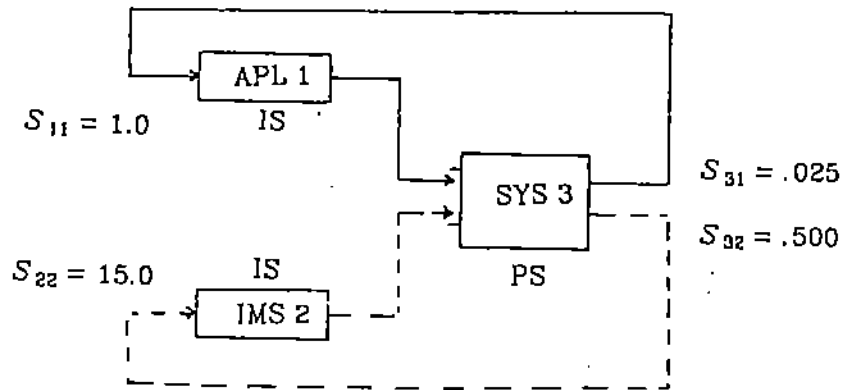


Figure 8

Model with Two Job Classes [Reis76]

$$[V_{\leftarrow}] = \begin{bmatrix} 1.0 & 0 \\ 0 & 1.0 \\ 1.0 & 1.0 \end{bmatrix}$$

$$[S_{\leftarrow}] = \begin{bmatrix} 1.0 & 0 \\ 0 & 15.0 \\ .025 & .500 \end{bmatrix}$$

Table 2

Input Parameters - Example 1

$N$	$W_{11}$	$W_{22}$	$W_{31}$	$W_{32}$	$X_{01}$	$X_{02}$	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$
(0,0)						0	0	0	
(1,0)	1.0		.025		.976		.976		.024
(0,1)		15.0		.500		.065		.968	.032
(2,0)	1.0		.026		1.950		1.950		.050
(1,1)	1.0	15.0	.026	.512	.975	.064	.975	.967	.058
(2,1)	1.0	15.0	.027	.525	1.949	.064	1.949	.966	.085

Table 3

Solution for  $N = (2,1)$  - Example 1

device	class	$U_{ir}$	$X_{ir}$	$\bar{n}_{ir}$	$W_{ir}$
1	1	1.949	1.949	1.949	1.000
	2	0	0	0	0
	tot	1.949	1.949	1.949	1.000
2	1	0	0	0	0
	2	.966	.084	.966	15.000
	tot	.966	.084	.966	15.000
3	1	.049	1.949	.052	.027
	2	.032	.084	.034	.525
	tot	.081	2.031	.086	.043

Table 4

Results for  $M = (2,1)$  - Example 1

device	class	$U_{ir}$	$X_{ir}$	$\bar{n}_{ir}$	$W_{ir}$
1	1	14.3	14.3	14.3	1.0
	2	0	0	0	0
	tot	14.3	14.3	14.3	1.0
2	1	0	0	0	0
	2	4.707	.314	4.707	15.0
	tot	4.707	.314	4.707	15.0
3	1	.358	14.3	.677	.047
	2	.157	.314	.293	.934
	tot	.515	14.6	.970	.086

Table 5

Results for  $M = (15,5)$  - Example 1 [Reis76]

### Example 2 - Class Switching

The second example illustrates class switching. The system being modeled consists of three devices, a CPU and two I/O devices. Jobs in class 1 leave the CPU and join class 2 with probability .1; jobs of class 2 leave the CPU and join class 1 with probability .2. Figure 9 is a diagram of this model. Tables 6, 7, 8 and 9 show the parameters, solution steps and results for this example. It can be noted that the solution (superclass 1) illustrates solution of a single class QNM.

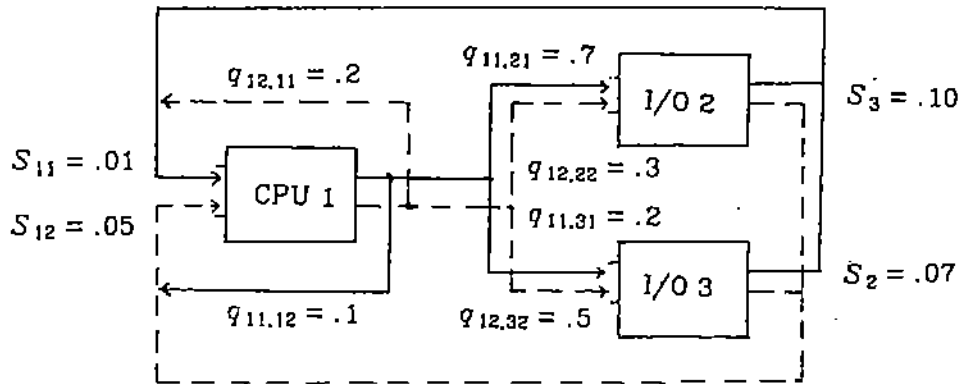


Figure 9  
Class Switching Example

$$[S_{ir}] = \begin{bmatrix} .01 & .05 \\ .07 & .07 \\ .10 & .10 \end{bmatrix}$$

$(ir, js)$	$q_{ir, js}$	$ir$	$V_{ir}$
11,21	.7	11	10
11,31	.2	21	7
11,12	.1	31	2
12,22	.3	12	5
12,32	.5	22	1.5
12,11	.2	32	2.5

Table 6  
Input Parameters - Example 2

$$[V_{iq}^*] = \begin{bmatrix} 15 \\ 8.5 \\ 4.5 \end{bmatrix}$$

$$[a_{qr}] = \begin{bmatrix} .667 & .333 \\ .824 & .177 \\ .444 & .556 \end{bmatrix}$$

$$[S_{iq}^*] = \begin{bmatrix} .023 \\ .070 \\ .100 \end{bmatrix}$$

$N_1 = 3, N_2 = 0$ , implies  $N_1^* = 3$

Table 7

Auxiliary Constants - Example 2

$n$	$W_1^*$	$W_2^*$	$W_3^*$	$X_n^*$	$\bar{n}_1^*$	$\bar{n}_2^*$	$\bar{n}_3^*$
0					0	0	0
1	.023	.070	.100	.717	.251	.427	.322
2	.029	.100	.132	1.062	.465	.902	.633
3	.034	.133	.163	1.261	.647	1.427	.926

Table 8

Solution for Superclass - Example 2

	11	21	31	12	22	32
$W_{qr}$	.015	.133	.163	.073	.133	.163
$X_{qr}$	12.609	8.826	2.522	6.304	1.891	3.152
$\bar{n}_{qr}$	.185	1.175	.412	.462	.252	.515
$U_{qr}$	.126	.618	.252	.315	.132	.315

Table 9

Class Results for  $\mathcal{N} = (3,0)$  - Example 2

### Example 3 - Mixed (Open and Closed Classes) Model

The third example illustrates the operation of the MVA algorithm, modified to solve mixed models. In a mixed model, there are both open job classes (classes with external arrivals) and closed job classes (classes with a fixed and

constant number of jobs). In this example, class 1 is the open class and class 2 is the closed class.

Figure 10 is a diagram of this model. Tables 10, 11 and 12 give the parameter values and solution steps for an external arrival rate of 1 class job per second and 3 class 2 jobs active.

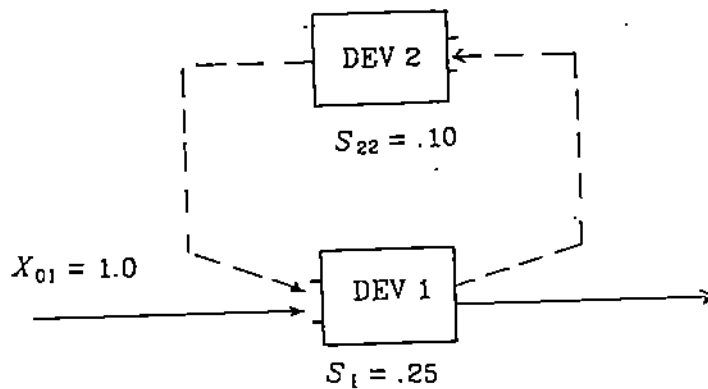


Figure 10

Mixed QNM - Example 3

$$[V_{\#}] = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

$$[S_{\#}] = \begin{bmatrix} .25 & .25 \\ 0 & .10 \end{bmatrix}$$

$$X_{01} = 1.0$$

Table 10

Input Parameters - Example 3

$X_{11}$	1.0
$\rho_{11}$	.25
$\rho_1$	.25
$W_{11}$	.33
$\bar{n}_{11}$	.33
$\bar{n}_1$	.33

Table 11

Solution of Open Class - Example 3

$N_2$	$W_{12}$	$W_{22}$	$X_{02}$	$\bar{n}_{12}$	$\bar{n}_{22}$	$W_{11}$	$\bar{n}_{11}$	$\bar{n}_1$
0								.333
1	.333	.100	2.308	.769	.231	.590	.590	1.359
2	.590	.123	2.806	1.855	.345	.885	.885	2.540
3	.885	.135	2.943	2.604	.396	1.201	1.201	3.805

Table 12

Solution - Example 3

The model in Example 3 was simulated, in order to verify the correctness of the solution. In the simulation model, a fixed number of class 2 jobs (the closed class) were initiated and the model allowed to continue for 2500 arrivals of class 1 jobs (the open class). Each experiment was repeated three times. The experiment was run for  $N_2$ , the number of jobs in the closed class, varying from 1 to 5. A comparison of the output of the MVA algorithm and the these 15 simulation runs is given in Table 13.

CPU Utilization

$N_2$	MVA	Run 1	Run 2	Run 3	Avg.
1	.827	.828	.818	.823	.822
2	.951	.954	.950	.951	.952
3	.986	.987	.987	.985	.986
4	.996	.995	.994	.997	.995
5	.999	.999	.998	.999	.999

CPU Response Time - Open Class

1	.590	.833	.590	.547	.590
2	.885	.913	.856	.905	.891
3	1.201	1.217	1.240	1.137	1.198
4	1.528	1.485	1.442	1.513	1.480
5	1.859	1.874	1.835	1.861	1.857

CPU Response Time - Closed Class

1	.333	.335	.328	.320	.328
2	.590	.595	.577	.582	.585
3	.885	.902	.898	.864	.888
4	1.201	1.193	1.160	1.190	1.181
5	1.528	1.550	1.505	1.523	1.526

Table 13

Comparison with Simulation Results - Example 3

**Discussion**

This report has given a fairly complete description of the implementation of the MVA algorithm, with several extensions. A computer program was written in the C Programming Language following the steps given in this paper. The results given in the section with examples were obtained from this program. Every attempt has been made to verify that the program is giving correct results.

The major extension not included in this note (and in the program) deals with devices with either multiple servers or load dependent servers and which are visited by jobs from open classes. These extensions, along with a few others, have been recently presented by Sauer [Sauer81].

The purpose of this paper is describe the MVA algorithm with extensions in

such a way that the reader could implement a program to solve QNM's. The examples are provided, to allow the results to be verified.

### References

- [BaBS77] Balbo, G. and S. Bruell, and H. Schwetman, "Customer classes and closed network models - a solution technique", *Proceedings IFIPS 77*, North Holland Pub., 1977, p. 559.
- [Balb79] Balbo, G. "Aproximate solutions of queueing network models of computer systems", Ph.D. Thesis, Dept. Computer Sciences, Purdue Univ., 1979.
- [BCMP75] Baskett, F., Chandy, K., Muntz, R. and F. Palacios, "Open, closed, and mixed networks of queues with different classes of customers", *Jour. ACM*, (22,2), April, 1975, p. 248.
- [BrBa80] Bruell, S. and G. Balbo, *Computational Algorithms for Closed Queueing Networks*, North Holland Pub., 1980.
- [Brum81] Brumfield, J., "Storage Management Schemes for Implementing Mean Value Analysis", CSD-TR, Dept. of Computer Sciences, Purdue University, February, 1981.
- [Buze73] Buzen, J., "Computational algorithms for closed queueing networks with exponential servers", *Comm. ACM*, (16,9), Sept. 1973, p. 527.
- [BuDe80] Buzen, J. and P. Denning, "Operational treatment of queue distributions and mean value analysis", *Computer Performance*, (1,1), June 1980, p. 6.
- [ChSa80] Chandy, K. and C. Sauer, "Computational algorithms for product form queueing networks", *Comm. ACM*, (23,10), Oct. 1980, p.

- [DeBu78] Denning, P. and J. Buzen, "The operational analysis of queueing network models", *Comp. Surveys*, (10,3), Sept. 1978, p. 225.
- [Reis76] Reiser, M., "Interactive models of computer systems", *IBM Systems Jour.* (15,4), 1976, p. 309.
- [ReLa78] Resier, M. and S. Lavenberg, "Mean value analysis of closed multichain queueing networks", IBM Research Report, RC 7023, March, 1978, also appeared in *Jour. ACM.* (27,2), April 1980, p. 313.
- [Saue81] Sauer, C., "Computational Algorithms for State-Dependent Queueing Networks", IBM Report RC 8698 (2/13/81), IBM T.J. Watson Research Center, Yorktown Heights, NY.
- [SeMi79] Sevcik, K. and Mitrani, "The distribution of queueing network states at input and output instants", *Proc. 4th International Symposium on Modelling and Performance Evaluation of Computer Systems*, IFIPS W.G. 7.3, Vienna, Austria, February, 1979.
- [Zaho80] Zaborjan, J., "The approximate solution of large queueing network models", Ph.D. Thesis, Dept. of Computer Science, Univ. of Toronto, 1980.