Mean Value Analysis for Closed, Separable, Multi Class Queueing Networks with Single Server & Delay Queues

Chandrakanth Chereddi
Department of Electrical & Computer Engineering, and
Coordinated Science Laboratory,
University of Illinois at Urbana-Champaign
cchered2@uiuc.edu

ABSTRACT

This project is aimed at studying and implementing different techniques to compute mean values of interest for closed, product-form (separable), multi-class queueing networks with single-server and delay queues.

In the first few sections theorems which form the basis for MVA algorithms are presented. Following this is a discussion on the exact MVA algorithm. In the second part, two classes of approximate MVA algorithms are presented. The main class being those which use iterative methods. Various iterative techniques vary mainly in the approximation they use for the most time and space consuming step of the exact MVA algorithm. Two non iterative algorithms are also discussed, however their accuracy is not very satisfactory and should be used only when a rough estimate will suffice. We also discuss the asymptotic complexities of every algorithm presented.

A software implementation of these algorithms has been developed as part of this project. I present the design of this software, explaining it's salient and useful features. The software is so designed that new algorithms and extensions can be added with minimum effort. Finally, I present some analysis and results from what has been observed in experimentation. Directions for future work and extensions are also mentioned, in some algorithm descriptions, and in the conclusion.

General Terms

Queueing Theory, Performance Analysis.

Keywords

Mean Value Analysis, Approximate Algorithms, Queueing Theory Analysis.

1. INTRODUCTION

Mean Value Analysis (MVA) is a popular tool used in the performance analysis arena. Out of this one particular class of networks, ie. those with multiple classes of customers, separable queues, with single servers and delay queues, are amenable to computation. This ease of computation was possible because of the *Arrival Instant Theorem* proven by Reiser and Lavenberg [9] and Sevcik and Mitrani [3] separately.

NB: In this report the terms "multiple customer classes" and "routing chains" are used interchangeably. During discussion both these terms are to be understood as one and the same. Similarly, when a reference is made to the word "servers", it does not mean multiple servers at a particular service center, but is referring to the service centers themselves (each with only one server, ie. fixed rate), to avoid the long recital.

Based on the arrival instant theorem, Reiser and Lavenberg designed an algorithm to exactly compute the mean values and this is the algorithm which is most widely used. This algorithm has a very intuitive physical interpretation (which stems from the arrival instant theorem), which made it possible to develop many heuristic extensions. These extensions form the basis for the approximate versions of MVA.

The exact MVA algorithm was a great reduction in both space and time complexity of computation from what was previously possible. Yet, was intractably large for networks with many customer classes, each with a large population. However, it was still possible to come up with good approximations. These approximate algorithms start with an initial uniform distribution of mean queue lengths at each service center, and then predict what the queue length would be with one less customer in each class. This prediction can be combined with Little's law and the arrival instant theorem to re-compute the mean queue length with the initial population configuration. This iteration continues until when there is little or no variation between the initial and final values of the mean queue lengths (at the initial population) of an iteration.

Some approximate algorithms try to avoid this iteration until convergence, as many of the iterative methods do not have known convergence properties [6] and/or converge too slowly. However, these techniques have a high error ratio and are best used only as rough estimates.

2. BACKGROUND

In this section I will briefly discuss the background required to understand the material to be presented later.

Arrival Instant Theorem: The arrival instant theorem states that the in an N customer closed queueing network, the average number of customers found upon arrival by a customer at the jth queue is $Q_j(N-1)$, the average number of customers seen by a random observer in the $(N-1)$ customer closed network (ie. with one less customer.

A couple of important observations.

- The mean queue length of a customer class at any service center is a monotonically increasing function with

the population (when there is no change in population in any other customer class).

- Mean queue length when there are no customers in a particular class is, obviously, zero.

System parameters, which are given as inputs are as follows.

- Two Integers: C Customer Classes and K Service Centers (load independant or single-server).
- Two vectors: Customer Population, denoted with $\vec{\mathbf{N}} = (N_1, N_2, \ldots, N_C)$, where N_c denotes customers in class c. Think time, denoted by $\vec{\mathbf{Z}} = (Z_1, Z_2, \ldots, Z_C)$, where Z_c denotes the average think time of a customer in class c.
- Matrix: Service demand, denoted with $D_{c,k}$ to mean the service demand of a customer of class c at center k.

In the current software these parameters are passed using the following data structure.

```
typedef struct queueing_formalism {
  /* Queueing formalism Identifier */
    unsigned int id;
  /* Population vector of size 'dim' 1D */
    unsigned int *p;
  /* Number of customer classes */
    unsigned int dim;
  /* No of servers/service centers */
    unsigned int servers;
  /* Service demand C x K matrix */
    float **sd;
  /* Think time of size 'dim' 1D */
    float *think_time;
} qf;
```

Performance measures of interest:

- $R_{c,k}(\vec{\mathbf{N}})$ = Average residence time of class c customer at center k.
- $R_c(\vec{\mathbf{N}})$ = Average response time of a class c customer in the network.
- $X_c(\vec{\mathbf{N}})$ = Throughput of the class c.
- $Q_{c,k}(\vec{\mathbf{N}})$ = The mean queue length of class c at center k.
- $Q_k(\vec{\mathbf{N}})$ = The mean total queue length at service center k.

Please note that the values Q_k and R_c can be computed by summing the corresponding rows or columns in $Q_{c,k}$ and $R_{c,k}$ respectively. Hence, the output of all the MVA algorithms in this project are going to be one vector and two matrices, to respectively represent the throughput per class and one matrix each for response time and the mean queue length. The outputs of a mean value analysis algorithm are represented as a data structure shown below.

```
typedef struct mean_values {
  /* Queueing Formalism Identifier */
    unsigned int id;
  /* ID of the MVA algorithm used */
    unsigned int algo_id;
  /* Mean residence time C x K matrix */
    float **res_time;
  /* Mean queue length C x K matrix */
    float **mq_len;
  /* Throughput - 1D Vector */
    float *tput;
} mvals;
```

NB: In the many equations $\vec{\mathbf{1}}_c$ is used, to mean a population vector, where the population of class c is 1, and zero for every other class, ie. $N_i = 1$ when $c = i$ and zero otherwise.

3. EXACT MEAN VALUE ANALYSIS

The exact MVA algorithm is a combination of the Arrival Instant Distribution and Little's law, first proposed in [1]. It depends on the following class of equations.

$$A_{c,k}(\vec{\mathbf{N}}) = Q_k(\vec{\mathbf{N}} - \vec{\mathbf{1}}_c) \quad (1)$$

$$R_{c,k}(\vec{\mathbf{N}}) = D_{c,k} \cdot (1 + A_{c,k}(\vec{\mathbf{N}})) \quad (2)$$

$$R_c(\vec{\mathbf{N}}) = \sum_{k=1}^{K} R_{c,k}(\vec{\mathbf{N}}) \quad (3)$$

$$X_c(\vec{\mathbf{N}}) = \frac{N_c}{Z_c + R_c(\vec{\mathbf{N}})} \quad (4)$$

$$Q_{c,k}(\vec{\mathbf{N}}) = R_{c,k}(\vec{\mathbf{N}}) \cdot X_c\vec{\mathbf{N}} \quad (5)$$

$$Q_k(\vec{\mathbf{N}}) = \sum_{c=1}^{C} Q_{c,k}(\vec{\mathbf{N}}) \quad (6)$$

Intuitively, if we know what the mean total queue length at a particular service center is, with one less customer in that customer class, we can compute the mean values for the current population. This is exactly how recursive algorithms work, ie. build on a broken sub-problem. Here the sub problem being the mean total queue length with one less customer in each class. It is well known wisdom that such algorithms can usually be rewritten as dynamic programming solutions, usually at the expense of some extra space complexity. The MVA algorithm hence has been modified from it's initial recursive solution to a more tractable dynamic programming solution.

In figure 1, one can see how the mean queue length is built using the mean queue length with one less customer in each class.

The MVA algorithm as was implemented in this project, is explained below in algorithm 1. This implementation draws it's basic form from [5].

However, as you will note, an important step in the algorithm is to produce all populations which sum to a particular value, and for each such population store the value of $Q_k \forall k$. This immediately makes the storage requirement of the order $\mathbf{\Theta}(K \prod_{c=1}^{C}(N_c + 1))$. The time complexity would be an order of C greater at $\mathbf{\Theta}(KC \prod_{c=1}^{C}(N_c + 1))$.

A diagrammatic representation of how the various steps in the algorithm would work is shown in figure 1. If we proceed in the top-down fashion, we would end up with the

Algorithm 1 Dynamic Programming Algorithm for Mean Value Analysis

for $k \leftarrow 1$ to K do $Q_k(\vec{0}) \leftarrow 0$
for $n \leftarrow 1$ to $\sum_{c=1}^{C} N_c$
for each feasible population $\vec{n} \equiv (n_1, n_2, \ldots, n_c)$ with n total customers **do**
  for $c \leftarrow 1$ to C **do**
    for $k \leftarrow 1$ to K **do**
      $R_{c,k} \leftarrow D_{c,k} \cdot [1 + Q_k(\vec{n} - \vec{1}_c)]$
    end for
    for $c \leftarrow 1$ to C do $X_c \leftarrow \dfrac{n_c}{Z_c + \sum_{k=1}^{K} R_{c,k}}$
    for $k \leftarrow 1$ to K do $Q_k(\vec{n}) \leftarrow \sum_{c=1}^{C} X_c \cdot R_{c.k}$
  end for
end for

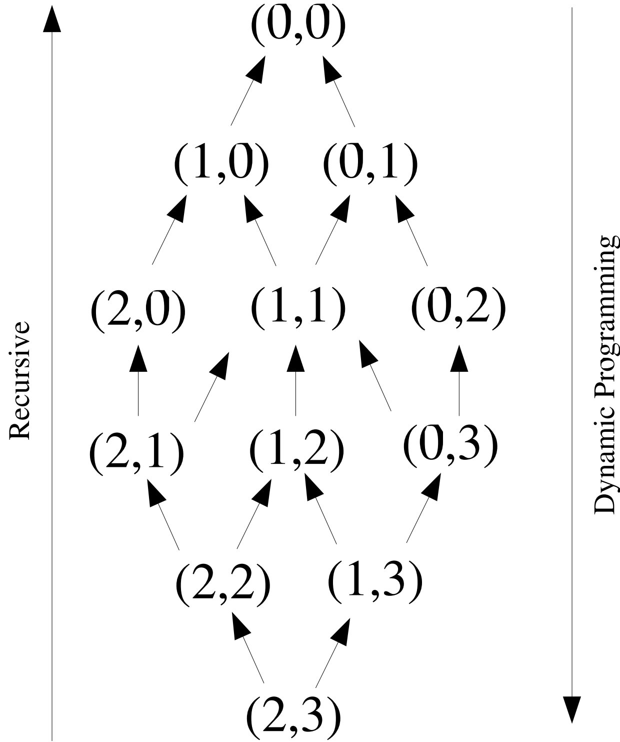

Figure 1: Queue lengths at each sub population are calculated. Populations sum to the same value at each level. We start with the base case where queue length for zero population is zero at all centers.

algorithm as above. We can also proceed in the bottom up fashion, however, we would repeatedly solve for some of the sub populations (which is easily avoided in the dynamic programming version).

Exact MVA algorithm is available as a C function call with the following prototype in our implementation.

```
void
exact_mva(IN qf *fm, OUT mvals *mv)
```

4. APPROXIMATE MEAN VALUE ANALYSIS: ITERATIVE ALGORITHMS

Approximate mean value analysis algorithms all aim at giving an approximation to $A_{c,k} = Q_k\vec{\mathbf{N}} - \vec{\mathbf{1}}_c$ (this equation is the main reason for the recursion). Hence, they can directly compute $Q_{c,k}(\vec{\mathbf{N}})$, without having to recurse to levels lower than that. This procedure iterates until the value of $Q_{c,k}(\vec{\mathbf{N}})$ converges. The choice of when the iteration will cease can be made more complex, and is the basis for improvements to a few approximate MVA algorithms.

The basic skeleton for any iterative approximation algorithm is as follows.

Algorithm 2 Skeleton of Iterative Approximate MVA Algorithms

Initialize $Q_{c,k} \leftarrow \dfrac{N_c}{K}$ ie. uniform load distribution
while Termination conditions not met **do**
  for $c \leftarrow 1$ to C **do**
    for $k \leftarrow 1$ to K **do**
      $A_{c,k} \leftarrow$ Approximate using system params and $Q(\vec{\mathbf{N}})$
    end for
  end for
  for $c \leftarrow 1$ to C **do**
    for $k \leftarrow 1$ to K **do**
      $R_{c,k} \leftarrow D_{c,k} \cdot [1 + Q_k(\vec{\mathbf{N}} - \vec{\mathbf{1}}_c)]$
    end for
    for $c \leftarrow 1$ to C do $X_c \leftarrow \dfrac{N_c}{Z_c + \sum_{k=1}^{K} R_{c,k}}$
    for $k \leftarrow 1$ to K do $Q_k(\vec{\mathbf{N}}) \leftarrow \sum_{c=1}^{C} X_c R_{c.k}$
  end for
  *S*et termination based on some condition (usually convergence of $Q(\vec{\mathbf{N}})$)
end while

However, based on the amount of computation and the level of approximation used, the algorithms vary. Please keep in mind that the statement '*'Approximate using system parameters and $Q(\vec{\mathbf{N}})$"*, could it self be a sub routine, which could have higher complexity than $\mathbf{O(1)}$. Usually this is a $\mathbf{O}(KC)$. Also, most popular approximate MVA algorithms have a time and space complexity of $\mathbf{O}(KC)$, when considering the number of iterations to be constant or a relatively small value. If the number of iterations is a factor based on the system parameters (which is possible), the asymptotic complexities would be quite different.

Important Note: Unless otherwise mentioned, after an approximation has been applied for $A_{c,k}$, the remaining parts

of the exact MVA algorithm remain unchanged, ie. iterations to calculate $R_{c,k}(\vec{\mathbf{N}}), Q_{c,k}(\vec{\mathbf{N}})$ and $X_c(\vec{\mathbf{N}})$. Also note that the space and time complexity of the exact MVA, even when $A_{c,k}$ is exactly known is $\Theta(KC)$, and hence is a **lower bound**, on what can be achieved using the arrival instant theorem as the basis.

4.1 Large Customer Population Algorithm

Bard [11] proposed an approximate MVA algorithm which is useful when the number of customers in a network is large. This approximation is rather naive, and is based on the observation that removing one customer from a large population will not effect the mean total queue length at any service center.

$$Q_{c,k}(\vec{\mathbf{N}} - \vec{\mathbf{1}}_c) \approx Q_{c,k}(\vec{\mathbf{N}})$$

This implies the approximation used is

$$A_{c,k} = Q_k(\vec{\mathbf{N}} - \vec{\mathbf{1}}_c) = \sum_{i=1}^{C} Q_{i,k}(\vec{\mathbf{N}} - \vec{\mathbf{1}}_c) \approx Q_k(\vec{\mathbf{N}})$$

Since the above expression at worst has a time and space complexity of $O(KC)$, the overall approximate MVA algorithm itself would have a time complexity of $O(KC)$.

4.2 Proportional Estimation Algorithm

As the name suggests, Proportional Estimation (PE) [10] algorithm calculates the value of $A_{c,k}(\vec{\mathbf{N}})$ based on proportional estimation from the initial estimate for $Q_{c,k}(\vec{\mathbf{N}})$. The PE algorithm is based on the following observations.

1. Mean total queue length is a monotonically increasing function with the population.
2. Mean queue length for the zero population case is, obviously, zero.
3. The above two observations imply that on a curve, which can be approximated as a straight line, we know the values at two points, viz. $(0, 0)$ and $(N_c, Q_{c,k}(\vec{\mathbf{N}}))$.
4. Removing one customer from a customer class does not greatly effect, the mean queue lengths of other classes.

From the above observations it would intuitive to guess that what the approximation would be.

$$Q_{i,k}(\vec{\mathbf{N}} - \vec{\mathbf{1}}_c) \approx \frac{N_c - 1}{N_c} \cdot Q_{c,k}(\vec{\mathbf{N}}) \; \forall c = i$$

$$Q_{i,k}(\vec{\mathbf{N}} - \vec{\mathbf{1}}_c) \approx Q_{i,k}(\vec{\mathbf{N}}) \; \forall c \neq i$$

This leads to an approximate computation of $A_{c,k}$ as follows.

$$A_{c,k} \approx Q_k(\vec{\mathbf{N}} - \vec{\mathbf{1}}_c) \approx \sum_{i=1}^{C} Q_{i,k}(\vec{\mathbf{N}} - \vec{\mathbf{1}}_c)$$

$$\approx Q_k(\vec{\mathbf{N}}) - \frac{1}{N_c} Q_{c,k}(\vec{\mathbf{N}})$$

Proportional estimation is the most popular and widely used approximate MVA algorithm. It has a rather intuitive understanding to it, which made it possible to develop other heuristic extensions. This became the basis for two new algorithms (discussed later) proposed by Wang and Sevcik, known as the Queue Line (QL) and the Fraction Line (FL) algorithm.

The time and space complexity of the PE algorithm is at $O(KC)$. However, the termination condition may worsen this, based on the error tolerance given. In some cases, it is possible that the convergence will take very long. Only for the single customer case, the convergence properties have been proven [6]. Generally an upper bound on number of iterations is chosen in case the algorithm does not converge. In my implementation I have chosen the error tolerance at 0.1% and maximum iterations to be, arbitrarily, at 100 iterations.

4.3 Queue Line Algorithm

The Queue Line algorithm was proposed by Wang and Sevcik in 1996 [2], based on the observations made in the PE algorithm. It stems from a very intuitive, yet insightful observation explained below.

Main Observation: In the PE algorithm the main idea was to interpolate between two points on a monotonic curve, one of the points being the origin. However, it is not very hard to get a better estimate, if we better knew the position of another point on the curve, between the origin and N_c, ie. closer to $(N_c - 1)$.

Hence, the main contribution of the Queue Line algorithm is to provide a close estimate, as to what the value of $Q_{c,k}(\vec{\mathbf{N}} - (N_c - 1)\vec{\mathbf{1}}_c)$, which is basically the value on the approximation curve at 1 instead of 0. Here, $(\vec{\mathbf{N}} - (N_c - 1)\vec{\mathbf{1}}_c)$, is the population configuration where all, except one, customers have been removed from class c.

Other approximation assumptions are similar to those of the PE algorithm. Hence the main equations for the QL algorithm would be as follows.

- *When $N_c = 1$ and $c = i$*

  $$Q_{i,k}(\vec{\mathbf{N}} - \vec{\mathbf{1}}_c) = 0$$

  When $N_c = 1$, the population configuration $(\vec{\mathbf{N}} - \vec{\mathbf{1}}_c)$ does not have any customers which belong to class c. So the the mean total queue length at that service center would be the same as removing the entire estimate for $Q_{c,k}$ from the mean total queue length at that service center with population configuration $(\vec{\mathbf{N}})$.

- *When $N_c > 1$ and $c = i$*

  $$\frac{Q_{i,k}(\vec{\mathbf{N}}) - Q_{i,k}(\vec{\mathbf{N}} - \vec{\mathbf{1}}_c)}{1} \approx$$

  $$\frac{Q_{i,k}(\vec{\mathbf{N}}) - Q_{i,k}(\vec{\mathbf{N}} - (N_c - 1)\vec{\mathbf{1}}_c)}{N_c - 1}$$

  That is, equate the slopes of the two lines.

- *When $N_c > 1$ and $c \neq i$*

  $$Q_{i,k}(\vec{\mathbf{N}} - \vec{\mathbf{1}}_c) \approx Q_{i,k}(\vec{\mathbf{N}})$$

  Removing one customer from a customer class does not greatly effect, the mean queue lengths of other classes at that service center.

This leads use to the following equations for $A_{c,k}$.

- *When* $N_c = 1$

$$A_{c,k}(\vec{\mathbf{N}}) \approx Q_k(\vec{\mathbf{N}}) - Q_{c,k}(\vec{\mathbf{N}})$$

- *When* $N_c > 1$

$$A_{c,k}(\vec{\mathbf{N}}) \approx Q_k(\vec{\mathbf{N}}) - \frac{1}{N_c - 1}\left\{Q_{c,k}(\vec{\mathbf{N}}) - \alpha\right\}$$

where

$$\alpha = \frac{D_{c,k}\left[1 + Q_k(\vec{\mathbf{N}}) - Q_{c,k}(\vec{\mathbf{N}})\right]}{Z_c + \sum_{j=1}^{K}\left\{D_{c,j}\left[1 + Q_j(\vec{\mathbf{N}}) - Q_{c,j}(\vec{\mathbf{N}})\right]\right\}}$$

α is algebraically equivalent to $Q_{c,k}(\vec{\mathbf{N}} - (N_c - 1)\vec{\mathbf{1}}_c)$

The space and time complexity of the QL algorithm is similar to that of the PE at $O(KC)$. However, the constants in the asymptotic are greater than what is expected in the PE, due to extra computation cost of α. But for most practical purposes, Wang and Sevcik believe that QL algorithm can replace the PE algorithm, since computational resources are not as constrained. Again, QL considers the number of iterations constant similar to PE, hence the observed complexity in practice may be slightly more than $O(KC)$.

4.4 Fraction Line Algorithm

The Fraction Line (FL) algorithm was also suggested by Wang and Sevcik [2]. It only varies slightly from the way QL works. The main observation here is that the interpolation for $Q_{i,k}(\vec{\mathbf{N}} - \vec{\mathbf{1}}_c)$ is between $(1, Q_{i,k}(\vec{\mathbf{N}} - (N_c - 1)\vec{\mathbf{1}}_c))$ and $(N_c, \frac{Q_{c,k}((\vec{\mathbf{N}})}{N_c})$. Intuitively, one can think about this as *scaled* interpolation. the math will be shown in the next few lines.

This leads us to the following approximations. **NB:** Only the second equation is different from that of QL.

- *When* $N_c = 1$ *and* $c = i$

$$Q_{i,k}(\vec{\mathbf{N}} - \vec{\mathbf{1}}_c) = 0$$

- *When* $N_c > 1$ *and* $c = i$

$$\frac{Q_{i,k}(\vec{\mathbf{N}})}{N_c} - \frac{Q_{i,k}(\vec{\mathbf{N}} - \vec{\mathbf{1}}_c)}{N_c - 1} \approx$$

$$\frac{1}{N_c - 1}\left[\frac{Q_{i,k}(\vec{\mathbf{N}})}{N_c} - \frac{Q_{i,k}(\vec{\mathbf{N}} - (N_c - 1)\vec{\mathbf{1}}_c)}{1}\right]$$

- *When* $N_c > 1$ *and* $c \neq i$

$$Q_{i,k}(\vec{\mathbf{N}} - \vec{\mathbf{1}}_c) \approx Q_{i,k}(\vec{\mathbf{N}})$$

It is easy to derive with some algebra, when $c = i$.

$$Q_{i,k}(\vec{\mathbf{N}} - \vec{\mathbf{1}}_c) \approx \frac{N_c - 2}{N_c}Q_{i,k}(\vec{\mathbf{N}}) + Q_{i,k}(\vec{\mathbf{N}} - (N_c - 1)\vec{\mathbf{1}}_c)$$

This can be contrasted with the QL approximation (some algebra required), when $c = i$.

$$Q_{i,k}(\vec{\mathbf{N}} - \vec{\mathbf{1}}_c) \approx \frac{N_c - 2}{N_c - 1}Q_{i,k}(\vec{\mathbf{N}}) + \frac{Q_{i,k}(\vec{\mathbf{N}} - (N_c - 1)\vec{\mathbf{1}}_c)}{N_c - 1}$$

Observe, that the scaling factor for $Q_{i,k}(\vec{\mathbf{N}})$ is different. It has been observed experimentally, that the FL algorithm does better when the population is small, or when the queue lengths at service centers are small (ie. not many bottlenecked nodes). If the population is large, it is clear that $\frac{N_c-2}{N_c-1} \geq \frac{N_c-2}{N_c}$ when $N_c \geq 2$. This also leads us to another observation that the PE, QL and FL algorithms will produce the same result when $N_c \leq 2$ (again with some algebra). However, as N_c increases, the difference between both the scaling factor decreases, which is quite opposite to what is observed (ie. as N_c increases, both the algorithms have a great deal of difference between them). This stems from the fact that as N_c increases, so does $Q_{i,k}(\vec{\mathbf{N}})$, and even a slight disturbance will lead to huge estimation errors by the FL algorithm. Also, the fact that in the FL algorithm, the value of $Q_{i,k}(\vec{\mathbf{N}} - (N_c - 1)\vec{\mathbf{1}}_c)$ is not scaled with N_c makes it more biased towards it. This bias also makes it particularly bad for large population cases, with large mean total queue lengths.

4.5 Aggregate Queue Length Algorithm

Zahorjan, Eager and Sweillam [4] proposed the Aggregate Queue Length algorithm, which is seen as a modification to the Linearizer algorithm previously proposed by Chandy and Neuse [7]. In principle, I think that though AQL is more accurate it use the same interpolation trick as the PE algorithm. However, it recurses one level deeper. That is, the computation of $Q_{i,k}(\vec{\mathbf{N}} - \vec{\mathbf{1}}_c)$ is approximated by computing $Q_{i,k}(\vec{\mathbf{N}} - \vec{\mathbf{1}}_c - \vec{\mathbf{1}}_j)$ (which again approximates the $Q_{c,k}$ curve as a straight line).

The following sets of equations are used.

- Rewrite $A_{c,k}$ as

$$A_{c,k} = (N - 1)\left(\frac{Q_{i,k}(\vec{\mathbf{N}})}{N} + \rho_{c,k}(\vec{\mathbf{N}})\right)$$

Where

$$\rho_{c,k}(\vec{\mathbf{N}}) = \frac{Q_k(\vec{\mathbf{N}} - \vec{\mathbf{1}}_c)}{N - 1} - \frac{Q_k(\vec{\mathbf{N}})}{N}$$

- However, for the second level recursion, the approximation to be used will be

$$\rho_{c,k}(\vec{\mathbf{N}} - \vec{\mathbf{1}}_i) \approx \rho_{c,k}(\vec{\mathbf{N}})$$

The algorithm for the skeletal approximate MVA would now be a sub routine call. The new algorithm is shown in algorithm 3.

It is easy to observe, that given the above algorithm, and the skeletal approximate MVA as a subroutine, we would have C more iterations. This leads to a complexity of $O(KC^2)$ in time, while the space complexity remains same (though the constant in the asymptotic may almost double).

4.6 Other Iterative Algorithms

There are various other iterative algorithms in literature [12]. Some of these recurse one more level deeper than the AQL algorithm to provide better estimates. But many of the

Algorithm 3 Aggregate Queue Length Algorithm

Initialize $\rho_{c,k}(\vec{\mathbf{N}}) \leftarrow 0 \ \forall c, k$
while Termination conditions not met **do**

Call skeletal iterative approximate MVA algorithm with above approximation, for **one iteration** only.

Recompute termination condition
if Termination conditions not met **then**
for $i \leftarrow 1, \ldots, C$ **do**

Call skeletal iterative approximate MVA algorithm with population $\vec{\mathbf{N}} - \vec{\mathbf{1}}_i$, and above approximations for **one iteration** only

end for
Recompute $\rho_{c,k}$ for all c, k
end if
end while

properties for these algorithms, such as convergence, remain unproved.

5. APPROXIMATE MEAN VALUE ANALYSIS: NON-ITERATIVE ALGORITHMS

The class of non-iterative approximate algorithms is rather small, as no algorithms with convincing accuracy have been devised. In fact the most popular of algorithm and it's variation appear in a single publication by Hsieh and Lam [8].

5.1 Proportional Approximation Method

Mean queue length when distributed proportionally in a single serve case, over all servers as per their service demands, will lead to upper-bounds. However, when we do this for the multi chain case (multiple customer classes ie.), it will lead to an approximation. This observation is the basis for the Proportional Approximation Method (PAM) algorithm. It depends on the following set of equations.

The following equations are solved for the PAM algorithm.

$$\gamma_{c,k} = \frac{D_{c,k}}{\sum_{i=1}^{K} D_{c,i}}$$

$$Q_{c,k}(\vec{\mathbf{N}}) = \gamma_{c,k} \cdot N_c$$

When $c \neq i$

$$Q_{i,k}(\vec{\mathbf{N}} - \vec{\mathbf{1}}_c) = Q_{i,k}(\vec{\mathbf{N}})$$

else

$$Q_{i,k}(\vec{\mathbf{N}} - \vec{\mathbf{1}}_c) = Q_{i,k}(\vec{\mathbf{N}}) - \gamma_{i,k}$$

$$R_{c,k}(\vec{\mathbf{N}}) = D_{c,k} \cdot (1 + Q_{c,k}(\vec{\mathbf{N}} - \vec{\mathbf{1}}_c))$$

$$R_c(\vec{\mathbf{N}}) = \sum_{k=1}^{K} R_{c,k}(\vec{\mathbf{N}})$$

$$X_c(\vec{\mathbf{N}}) = \frac{N_c}{Z_c + R_c(\vec{\mathbf{N}})}$$

As one can see, this is a rather straight forward approximation, and has a time and space complexity of $\Theta(KC)$.

5.2 Proportional Approximation Method Improved

The PAM algorithm in it's basic form may have utilization greater than one at some service centers. This is due to approximation errors. The PAM Improved (PAMI) version solves this with a sanity check.

The PAMI algorithm is as follows.

1. Execute the equations in PAM.
2. Calculate server utilization from the following formula

$$U_k(\vec{\mathbf{N}}) = \sum_{c=1}^{C} D_{c,k} \cdot X_c(\vec{\mathbf{N}})$$

3. Find largest utilization for chain, c as S_c, for all chains.

$$S_c = \mathbf{max}(U_k(\vec{\mathbf{N}})) \forall c, k$$

4. If $S_c > 1$, normalize throughput of that chain

$$X_c(\vec{\mathbf{N}}) = \frac{X_c(\vec{\mathbf{N}})}{S_c}$$

5. Recalculate utilization and throughput for all chains.

Since, the only additional step is the scaling of utilization, the space and time complexity remains at $\Theta(KC)$.

6. SOFTWARE DESIGN

I will list a few basic implementation details which may be of interest to anyone who wishes to use this package or develop extensions. First, I would like to make this software available for download as open source. I was surprised that algorithms this important do not have open implementations. This posed a great problem for me especially since benchmarking and correctness measures become very tough. For a measure of correctness, one can always compare the approximate and exact MVA algorithms, as they are expected to be in close range (usually less than 1% error).

6.1 Queueing Formalism Parser

For efficient experimentation, a file parser was required, which can easily read in and populate the "queueing formalism" data structure (explained previously) from a textual human readable representation. The following example format shows a queueing formalism in text.

```
# Formalism id - 11
11
# Chains, Servers
3 2
# Population
2 2 1
# Think time
72.447 14.302 61.304
# Service demands
0.441 5.176
2.968 16.907
3.375 8.480
# **Done**
```

The rules are as follows

- Any line which starts with a "#" or is blank (spaces/tabs etc.) is ignored.
- The first line contains a integer number which will be used to identify the formalism (non zero positive).
- The next line contains two space separated integers, first being number of chains and the next being number of servers (service centers).
- Followed by a line with 'chains' number of integers (non zero) - population vector.
- Think time per customer class, space separated float values (zero or positive).
- 'Server' number of lines, each with 'chains' number of floating point values - service demand per customer class per service center (zero or positive).

The parser function is available with the following prototype.

```
int parse_qfile(IN FILE *q_file,
    OUT qf ***parsed_fms,
    OUT unsigned int *nof_qfs)
```

It reads a file until end of file is reached or until the characters "END" are read on a new line. All memory allocation is done internal to the parser. On error, the returned value is set to be negative.

6.2 General Interface to MVA Algorithms

MVA algorithms generally read in a queueing formalism, and return a populated data structure with mean values (data structure explained previously). Function prototypes are of the following format usually.

```
[algorithm_name](IN qf *fm, OUT mvals *mv)
```

such as

```
exact_mva(IN qf *fm, OUT mvals *mv)
approx_mva_pe(IN qf *fm, OUT mvals *mv)
...
```

To add a new algorithm, one needs to provide a similar interface, and provide a new ID and a one line description for the algorithm in the mva_defs.h file. Requisite header files many be defined as per the algorithm.

As a matter of principle, memory allocated internal to the mva algorithms is to be deallocated before exiting the function. This is to avoid memory leaks, and unnecessary interactions. All memory required to return the mean values is allocated by the calling function. It may be a good feature to implement a void pointer to a private data structure (for any custom interactions with special algorithms), but this is a trivial extension.

6.3 Algorithms Implemented

The following MVA algorithms have been implemented in the package.

```
exact_mva(IN qf *fm, OUT mvals *mv)       /* Exact */
approx_mva_lcp(IN qf *fm, OUT mvals *mv)  /* LCP   */
approx_mva_pe(IN qf *fm, OUT mvals *mv)   /* PE    */
approx_mva_ql(IN qf *fm, OUT mvals *mv)   /* QL    */
approx_mva_fl(IN qf *fm, OUT mvals *mv)   /* FL    */
approx_mva_aql(IN qf *fm, OUT mvals *mv)  /* AQL   */
approx_mva_pamb(IN qf *fm, OUT mvals *mv) /* PAM   */
approx_mva_pami(IN qf *fm, OUT mvals *mv) /* PAMI  */
```

In addition there are a few experimental patches and modification which I tried. Some seemed promising but some were disastrous. However, due to time and space constraints I cannot go into the details of what I have tried in this report. I will try to put it up someplace soon, cause what has not worked is as good an advice to give as to what has worked.

6.4 Other Details of Interest

Many helper functions which will ease development and debugging have been provided in the common_funcs.c and common_funcs.h files. Especially those required to compute errors for comparison, and easier memory allocation and deallocation.

A function which can output the mean values to a file and later parse it, is yet to be developed, and would be a good extension.

7. RESULTS AND ANALYSIS

For analysis, a few characteristic error values are calculated as suggested by by Wang and Sevcik [2]. Exact values are marked with an asterisk sign in the equations.

- Throughput - Mean absolute relative error,

$$\Gamma_X = \frac{1}{C}\sum_{c=1}^{C}\left|\frac{X_c - X_c^*}{X_c^*}\right|$$

- Response time - Mean response time error,

$$\Gamma_R = \frac{1}{C}\sum_{c=1}^{C}\left|\frac{R_c - R_c^*}{R_c^*}\right|$$

- Average queue length,

$$\Gamma_Q = \frac{1}{KC}\sum_{c=1}^{C}\sum_{k=1}^{K}\left|\frac{Q_{c,k} - Q_{c,k}^*}{Q_{c,k}^*}\right|$$

- Maximum absolute relative error in $Q_{c,k}$.

$$\Lambda_Q = max_{c,k}\left|\frac{Q_{c,k} - Q_{c,k}^*}{Q_{c,k}^*}\right|$$

To compute these error values, the following function can be used

```
int
compute_error_vals(IN qf *fm,/* The actual QF*/
    IN mvals *mv_exact,      /* Exact MVA values   */
    IN mvals *mv_approx,     /* Approximate values */
    OUT errs *er)
```

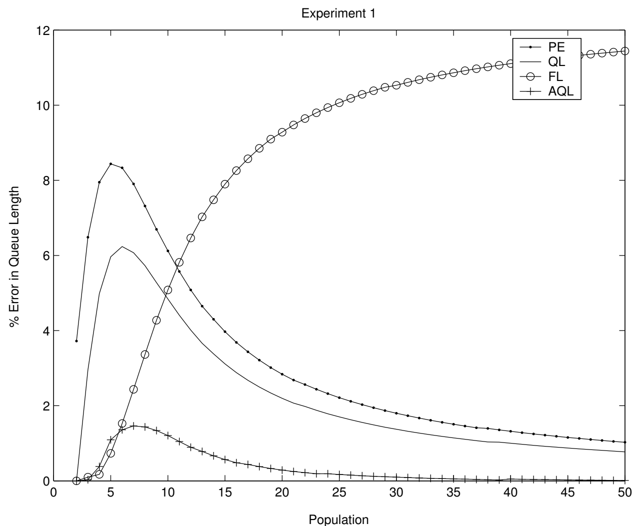

Figure 2: Shows error in mean queue length at different populations. As population and queue length increases the accuracy of the FL algorithm goes down. AQL is generally more accurate given it's extra computational complexity.

For the sake of understanding I have used hand crafted queueing formalisms which have been suggested in literature [2]. Out of which I will graphically show the performance on two formalisms. There are far too many experiments which have been run, many of which do not provide any necessary insight other than what is provided by the math. I will refrain from presenting those.

Main point of interest is a single customer case, where in one case, there are no bottle necked servers and the mean total queue length is not too large, while in another case it is. The experiment varies total population between 2 and 50.

Experiment 1: I use a queueing formalism with two service centers, single customer class, with think time of 10.0 and a service demand of 10.0 and 20.0 at each service center. The error percentages as compared to the exact MVA are shown in figure 2. On X-axis you can see that the population is varied from 2 to 50.

From the outputs I tried to develop a method to predict, which of the algorithms, ie. QL and FL, is more accurate (since QL is more accurate than PE usually, the competition was between the other two $O(KC)$ algorithms). As an initial try in that direction, I chose a prediction threshold for $\frac{Q_{c,k}}{N_c} > 0.85$ (remember that this is bounded by $[0, 1]$), to mean that when any location is bottle necked, then it is most likely that FL approximation has high error. This has proven to be a decent estimate. Please notice the line 'USE' in the figure 3. Note that there is no degradation in asymptotic complexity, as both QL and FL have same complexity, and hence both can be computed.

Experiment 2: In the second experiment a three centers, single customer class, with think time of 50.0 and a service demand of 10.0, 20.0 and 30.0 at each service center case is used. The error percentages in mean queue length as compared to the exact MVA are shown in figure 4. On X-axis you can see that the population is varied from 2 to 50.

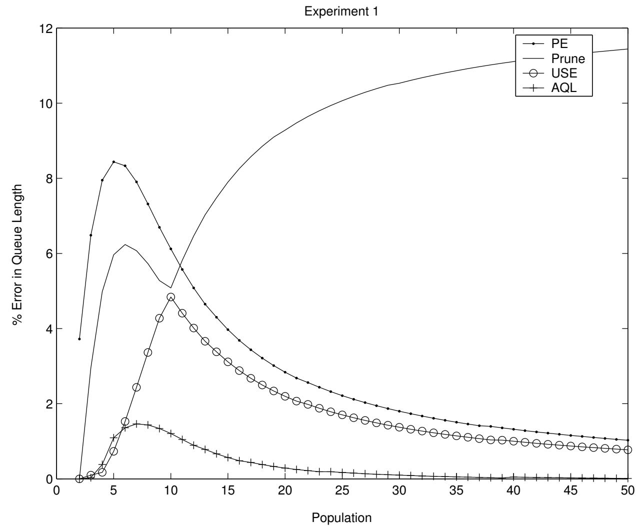

Figure 3: Based on outputs choose between QL or FL as the more accurate one.

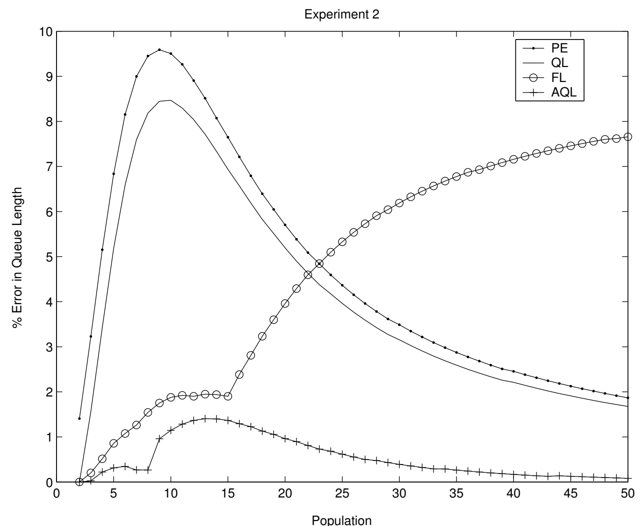

Figure 4: Shows error in mean queue length at different populations. As population and queue length increases the accuracy of the FL algorithm goes down.

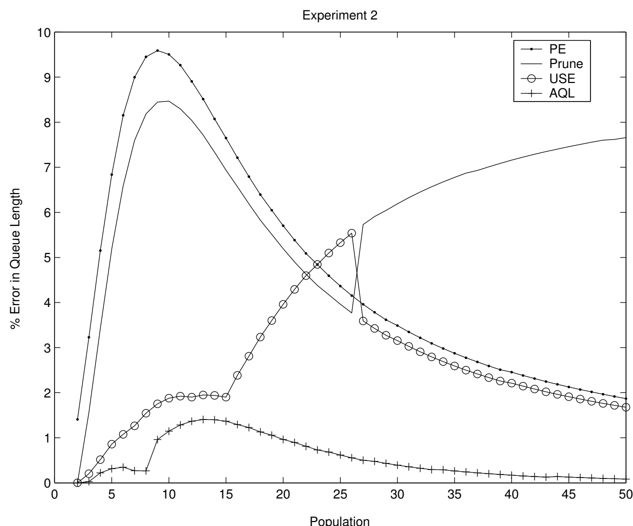

Figure 5: Based on output values and population QL or FL is chosen.

Again the same prediction threshold was used as in experiment 1, this yielded an output which is shown in figure5. As you can see there is some error in prediction, but not by a very large extent.

8. CONCLUSIONS AND FUTURE DIRECTIONS

In this project, I have tried to understand, implement, compare and analyze many mean value analysis techniques. In due course, I implemented an entire package which is capable of doing all this in the C programming language. A large amount of time was spent in carefully understanding and implementing the said algorithms. Such a process culminated in approximately 5000 lines of source code, out of which 50% is the main algorithm implementations, the remaining being very important driver and interface code. As one can guess, given the sensitivity of such data and computation intensive programming, many hours of sweat and blood which into this project.

Having said that, all through the text in this report, I have tried to give intuitive explanations to seemingly cryptic mathematical representations. The aim is that a reader will be able to understand these, and with some effort be able to start working on heuristic extensions. A few of which I myself am trying at this moment. The choice of making the source code freely available is also to facilitate the continuation of the study, should anyone be interested in doing so. Repeated work of implementation would sometimes be a waste in time, especially for the non algorithmic part. I hope that this can be avoided with the use of my driver functions and other interface parts of the package.

The software developed (in pre alpha) can be obtained from my personal web-site.
http://www.crhc.uiuc.edu/˜cchered2/

9. ACKNOWLEDGMENTS

I would like to thank Prof. William H. Sanders for many initial suggestions and guidance; and Vartika Bhandari for help in some murky parts of the mathematics.

10. REFERENCES

[1] M. Reiser and S. S. Lavenberg. Mean Value Analysis of closed multichain queueing networks. *Journal of the ACM*, 27(2):313–322, April 1980.

[2] H. Wang and K. C. Sevcik. Experiments with improved approximate mean value analysis algorithms. *Performance Evaluation*, 39(1-4): 189-206 (2000).

[3] K. C. Sevcik and I. Mitrani. The distribution of queueing network states at input and output instants. *Journal of ACM.*, 28(2): 358-371 April 1981.

[4] J.Zahorjan, D. L. Eager and H. M. Sweillam. Accuracy, speed and convergence of approximate mean value analysis. *Performance Evaluation*, 8(4):255-270, 1988.

[5] E. D. Lazowska, G. S. Graham, J. Zahorjan, and K. C. Sevcik. Quantitative System Performance: Computer System Analysis Using Queueing Networks Models. *Prentice-Hall*, Englewood Cliffs, NJ, 1984.

[6] K. R. Pattipati, M. M. Kostreva and J. L. Teele. Approximate mean value analysis algorithms for queueing networks, existence, uniqueness, and convergence results. *Journal of the ACM*, 37(3):646-673, July 1990.

[7] K. M. Chandy and D. Neuse. Linearizer: A heuristic algorithm for queueing network models of computing systems. *Communications of the ACM*, 25(2):126-134, February 1982.

[8] C. T. Hsieh and S. S. Lam. PAM - A non-iterative approximate solution method for closed multichain queueing networks. *ACM Transactions of Computer Systems* 4(2):178-185, May 1986.

[9] S. S. Lavenberg and M. Reiser. Stationary state probabilities of arrival instants for closed queueing networks with multiple types of customers. *Journal of Applied Probability*, 17(4):1048-1061, December 1980.

[10] P. J. Schweitzer. Approximate analysis of multiclass closed multichain queueing networks. *Proceedings of International Conference on Stochastic Control and Optimization*, 25-29, Amsterdam, Netherlands, 1979.

[11] Y. Bard. Some extensions to multiclass queueing network analysis. In: M. Arato, A. Butrimenko and E. Gelenbe, eds. *Performance of Computer Systems*, North-Holland, Amsterdam, Netherlands, 1979.

[12] H. Wang Approximate MVA Algorithms for Solving Queueing Network Models, *Masters' Thesis*, University of Toronto, June 1997.