A NOTE ON THE COMPUTATIONAL COST OF THE LINEARIZER ALGORITHM FOR QUEUEING NETWORKS

E.DE SOUZA E SILVA
R.R.MUNTZ *

NCE-04/88

Abril/88

Universidade Federal do Rio de Janeiro
Núcleo de Computação Eletrônica
Caixa Postal 2324
20001 - Rio de Janeiro - RJ
BRASIL

* UCLA Computer Science Department

Esta pesquisa contou com o apoio financeiro da NSF-USA INT-8514377 e CNPq.
Este trabalho foi publicado como relatório técnico do Departamento de Ciência da Computação da UCLA CSD-870025, Junho 1987.

Resumo

Linearizer é um dos mais conhecidos algorítmos de aproximação para se obter soluções numéricas para redes de filas com solução em forma de produto. Na explanação original de Linearizer, o custo computacional foi mostrado ser de $O(MK^3)$ para um modelo com M filas e K classes de *jobs*. Nesta nota mostramos que, com algumas manipulações algébricas simples, Linearizer pode ser modificado de tal maneira que o custo computacional seja reduzido para $O(MK^2)$.

Abstract

Linearizer is one of the best known approximation algorithms for obtaining numeric solutions for product form queueing networks. In the original exposition of Linearizer, the computational cost was stated to be $0(MK^3)$ for a model with M queues and K job classes. We show in this note that with some straight forward algebraic manipulation Linearizer can be modified to require only $0(MK^2)$ computational cost.

1 Introduction.

The Linearizer algorithm is one of the best known approximation techniques for closed product form queueing network models. The algorithm was proposed by Chandy and Neuse [CHAN82] in 1982. In that paper the authors indicate that the computational cost of the algorithm is $O(MK^3)$ where M is the number of centers in the network and K is the number of closed chains. However, the computational cost of Linearizer can be reduced to $O(MK^2)$. This reduction cost was mentioned in a footnote of [DeSO86], and was obtained after a small modification to the original algorithm. Due to the importance of Linearizer we feel that these results needed fuller exposition and dissemination.

In the interest of brevity we will assume familiarity with the Linearizer algorithm. We rely on [CHAN82] to provide motivation and requisite background. We feel this is appropriate since we are presenting a small modification to the original algorithm and wish to avoid a lengthy presentation. However, to make this note self contained, we will briefly describe the algorithm and its equations which will be referred throughout the note. In section 2 we summarize the Linearizer algorithm. In section 3 we emphasize why its cost was $O(MK^3)$ and show how the reduction in cost can be achieved. We present our conclusions in section 4.

2 The Linearizer Algorithm.

The discussion in this section parallels the one presented in [CHAN82] and is introduced here for completeness. The following notation will be used throughout the rest of the note and is summarized below. (This notation is the same one used in [CHAN82]).

M	=	number of service centers.
K	=	total number of chains in the network.
N_k	=	number of customers in chain k.
$\vec{N}$	=	population vector = $(N_1, \ldots, N_K)$.
$m(k)$	=	a specified service center visited by chain k.
ν_{mk}	=	visit ratio of a chain k customer to center m, scaled so that $\nu_{m(k)k} = 1$.
s_{mk}	=	mean service time of a chain k customer at center m.
Y_{mk}	=	$\nu_{mk} Y_k$ = mean throughput of chain k customer at center m, where $Y_k = Y_{m(k)k}$.
L_{mk}	=	mean number of customers of chain k at center m.
L_m	=	$\sum_{k=1}^{K} L_{mk}$.

W_{mk}	=	mean waiting time (queueing time + service time) of a chain k customer at center m.
$\vec{e}_k$	=	K-dimensional vector whose k-th element is one and whose other elements are zero.

The main equation of the mean value analysis algorithm [REIS80], equation (1) below,

$$W_{mk}(\vec{N}) = s_{mk}(1 + L_m(\vec{N} - \vec{e}_k)) \tag{1}$$

requires a solution for all populations from $(0, \ldots, 0)$ up to $\vec{N}$. Statistics for a network with population $\vec{N}$ are calculated recursively from statistics for the same network with population $\vec{N} - \vec{e}_k$, for all values of k, $1 \leq k \leq K$. Linearizer "breaks" the recursion by estimating, heuristically, statistics for population $\vec{N} - \vec{e}_k$ from statistics for population $\vec{N}$, in the following way:

Define $F_{mk}(\vec{N})$ as the fraction of chain k jobs at service center m when the population is $\vec{N}$,

$$F_{mk}(\vec{N}) = \frac{L_{mk}(\vec{N})}{N_k} \tag{2}$$

Define $D_{mkj}(\vec{N})$ as the difference in the fraction of chain k jobs at service center m, when we have full population and the same fraction when we have full population in all the chains except chain j where there is one less job.

$$D_{mkj}(\vec{N}) = F_{mk}(\vec{N} - \vec{e}_j) - F_{mk}(\vec{N}) \tag{3}$$

From (2) and (3) it is easy to see that the following identity is true:

$$L_{mk}(\vec{N} - \vec{e}_j) = (\vec{N} - \vec{e}_j)_k(F_{mk}(\vec{N}) + D_{mkj}(\vec{N})) \tag{4}$$

where $(\vec{N} - \vec{e}_j)_k$ is the population of chain k when one chain j job is removed from the network.

In order to compute $L_{mk}(\vec{N})$ we use equations (4), (1) and Little's result. However, to solve (4) we need the values of $D_{mkj}(\vec{N})$. As we will indicate later, Linearizer estimates the values of $D_{mkj}(\vec{N})$, by invoking successively the following Core algorithm:

Core Algorithm

Step 1 Initialization: get estimate values for $D_{mkj}(\vec{N})$ and $L_{mk}(\vec{N})$ $\forall\, m,\ k,\ j$.

Step 2 From equations (2) and (4) compute new estimates $L_{mk}(\vec{N} - \vec{e}_j)$, $\forall\, m,\ k,\ j$.

Step 3 Use the values of $L_{mk}(\vec{N} - \vec{e}_j)$ computed above to compute $W_{mk}(\vec{N})$ from (1). New estimates of $L_{mk}(\vec{N})$ can be easily obtained using Little's result (i.e., by applying the other two MVA equations).

Step 4 If the biggest difference between the new and old estimates of $L_{mk}(\vec{N})$ is less than a specified tolerance, then stop. Otherwise go to Step 2.

The Core algorithm above assumes the values of $D_{mkj}(\vec{N})$ are known. Linearizer estimates these values by invoking the Core algorithm for population $\vec{N}$ and all populations $\vec{N} - \vec{e}_j\ \forall j$, and assuming that $D_{mkj}(\vec{N} - \vec{e}_l) = D_{mkj}(\vec{N})$. In summary we have:

Linearizer Algorithm:

Step 1. Initialization: assume initial values for $L_{mk}(\vec{N})$, $L_{mk}(\vec{N} - \vec{e}_j)$ and assume $D_{mkj}(\vec{N}) = 0$. Set $I = 1$.

Step 2. Apply the Core algorithm for population $\vec{N}$. For that, use the most recent values of $L_{mk}(\vec{N})$ and $D_{mkj}(\vec{N})$.

Step 3. If $I = 3$, then stop. Otherwise continue.

Step 4. Call the Core algorithm for all populations $\vec{N} - \vec{e}_j$, $\forall\, j$. Use the most recent values of $L_{mk}(\vec{N} - \vec{e}_j)$ and $D_{mkj}(\vec{N})$.

Step 5. From equations (2) and (3) compute new estimates of $F_{mk}(\vec{N})$, $F_{mk}(\vec{N} - \vec{e}_j)$ and $D_{mkj}(\vec{N})$.

Step 6. $I = I + 1$. Go to Step 2.

3 Reduction of Computational Cost.

The Linearizer algorithm as described in [CHAN82], requires $K + 1$ calls to the Core Algorithm for each iteration through the top level steps (steps 2 through 6 above). The computational cost of $0(MK^3)$ comes from an assumed cost of $0(MK^2)$ for each call to the Core Algorithm. (In fact $O(MK^2)$ is the cost of a single Core iteration but tests have shown that the number of iterations is approximately a constant independent of M and K. More details concerning the number of iterations are presented in [CHAN82].) It is easy to see that this indeed is the cost of each call to the Core algorithm if it is implemented as described. We show below that the cost of each call to the Core algorithm (more precisely, the cost of each Core iteration) can be reduced to $0(MK)$ by some algebraic manipulations and simple restructuring of the algorithm. We emphasize that these manipulations will not change the algorithm in any material sense and the final outputs of the new algorithm will be identical to the original Linearizer algorithm. Thus all of the empirical evidence on the accuracy of Linearizer and comparisons with other approximations still hold with the new version of the algorithm.

The new version of the Core algorithm which we propose replaces Step 2 of the original algorithm (which requires computing $L_{mk}(\vec{M}-\vec{e}_j)$ $(\forall m,\ k,\ j,\ \vec{M}=\vec{N}$ or $\vec{M}=\vec{N}-\vec{e}_c)$ with computing $L_m(\vec{M}-\vec{e}_j)$ $(\forall m,\ j)$ directly. Note that the $L_{mk}(\vec{M}-\vec{e}_j)$ are not actually required in the Core algorithm. Only the $L_m(\vec{M}-\vec{e}_j)$ are actually used in Step 3 of the Core algorithm, to compute $W_{mk}(\vec{M})$.

We proceed then to develop an expression for $L_m(\vec{M}-\vec{e}_j)$. First assume $\vec{M}=\vec{N}$. From (2) and (4),

$$L_{mk}(\vec{N}-\vec{e}_j) = (\vec{N}-\vec{e}_j)_k \left[\frac{L_{mk}(\vec{N})}{N_k} + D_{mkj}(\vec{N})\right] \tag{5}$$

But then,

$$\begin{aligned} L_m(\vec{N}-\vec{e}_j) &= \sum_{k=1}^{K} L_{mk}(\vec{N}-\vec{e}_j) \\ &= \sum_{k=1}^{K} (\vec{N}-\vec{e}_j)_k \left[\frac{L_{mk}(\vec{N})}{N_k} + D_{mkj}(\vec{N})\right] \end{aligned} \tag{6}$$

To simplify the sum in the above equation we separate it into two parts: $k \neq j$ and $k=j$. We also note that,

$$(\vec{N}-\vec{e}_j)_k = N_k \qquad k \neq j$$

and

$$(\vec{N}-\vec{e}_j)_k = N_k - 1 \quad k = j$$

We then obtain,

$$L_m(\vec{N}-\vec{e}_j) = \sum_{\substack{k=1\\k\neq j}}^{K} N_k \left[\frac{L_{mk}(\vec{N})}{N_k} + D_{mkj}(\vec{N})\right] + (N_j-1)\left[\frac{L_{mj}(\vec{N})}{N_j} + D_{mjj}(\vec{N})\right] \tag{7}$$

Simple algebraic manipulation results in the following form of the above equation.

$$L_m(\vec{N}-\vec{e}_j) \;=\; L_m(\vec{N}) - \frac{L_{mj}(\vec{N})}{N_j} + D'_{mj}(\vec{N}) - D_{mjj}(\vec{N}) \qquad (8)$$

where $D'_{mj}(\vec{N}) \;=\; \sum_{k=1}^{K} N_k D_{mkj}(\vec{N})$.

Using the same development as above when $\vec{M} = \vec{N} - \vec{e}_c$, we have:

$$L_m(\vec{M}-\vec{e}_j) \;=\; L_m(\vec{M}) - \frac{L_{mj}(\vec{M})}{(\vec{M})_j} + D'_{mj}(\vec{M}) - D_{mjj}(\vec{M}) - D_{mcj}(\vec{M}) \qquad (\vec{M})_j > 0 \;\; (9)$$

Assuming that the values for the $D'_{mj}(\vec{M})$ $\forall$ m, j , $\vec{M} = \vec{N}$ and $\vec{M} = \vec{N} - \vec{e}_c$ are available a priori, then the cost of the Core algorithm is easily seen to be $0(MK)$.

Now consider the computation of the $D'_{mj}(\vec{M})$ $\forall$ m, j in the context of the Linearizer algorithm. In each top level iteration of Linearizer, the Core algorithm is called once for population $\vec{M} = \vec{N}$ and for each population $\vec{M} = (\vec{N} - \vec{e}_c)$, $c = 1, 2, \ldots, K$. If, for each of these calls to the Core algorithm, it was required to recompute the D'_{mj}(population vector) then each Core algorithm call would indeed cost $0(MK^2)$. However, in Linearizer

$$D_{mkj}(\vec{N}) \;=\; D_{mkj}(\vec{N}-\vec{e}_c) \qquad \forall\, m,\, k,\, j,\, c \qquad (10)$$

and thus,

$$D'_{mj}(\vec{N}) \;=\; D'_{mj}(\vec{N}-\vec{e}_c) \qquad \forall\, m,\, k,\, j,\, c \qquad (11)$$

Therefore we can pre-compute $D'_{mj}(\vec{N})$ $\forall$ m, j at a cost of $0(MK^2)$ and use these values for each of the $K+1$ calls to the Core algorithm. It is simple to see that the cost of Linearizer is then $O(MK^2)$.

In summary, the following modifications are made to the original Linearizer algorithm.

1. In Steps 1 and 5 of the Linearizer algorithm, compute $D'_{mj}(\vec{N})$ $\forall$ m, j prior to all other computations and store for use in the calls to the Core algorithm during Step 2 and Step 3.

2. Step 2 of the Core algorithm is replaced by a computation of $L_m(\vec{M} - \vec{e}_j)$ $\forall$ m, j using (8) if $\vec{M} = \vec{N}$ or (9) if $\vec{M} = \vec{N} - \vec{e}_c$ with the precomputed values for $D'_{mj}(\vec{N})$ $(= D'_{mj}(\vec{N} - \vec{e}_l))$ and $D_{mkj}(\vec{N})$ $(= D_{mkj}(\vec{N} - \vec{e}_c))$.

4 Conclusion.

We have shown how Linearizer can be reorganized to reduce the computational cost to $0(MK^2)$. This is accomplished without altering the algorithm in any way that affects the results and thus preserves the empirical evidence of the accuracy of the method.

It is tempting to consider the reduction of the space requirements of the Linearizer to $0(MK)$ (from $0(MK^2)$) since we need only values for $D'_{mj}(\vec{N})$ and $D_{mjj}(\vec{N})$. However, each call to the Core algorithm for population $(\vec{N} - \vec{e}_l)$ requires the previous estimates for $L_{mj}(\vec{N} - \vec{e}_l)$. Thus it does not appear possible to reduce the order of magnitude space requirements for Linearizer without surgery that would materially alter the algorithm.

Acknowledgment

We thank Glenn Mackintosh for his careful reading and comments on an earlier version of this note.

References

[REIS80] M. Reiser & S. S. Lavenberg, "Mean-value Analysis of Closed Multichain Queueing Networks", *JACM*, Vol. 27, no. 2, pp. 313-322, April 1980.

[CHAN82] K.M. Chandy & D. Neuse, "Linearizer: A Heuristic Algorithm for Queueing Network Models of Computing Systems", *CACM*, Vol. 25, no. 2, pp. 126-134, February 1982.

[DeSO86] E. de Souza e Silva, S. S. Lavenberg & R. R. Muntz, "A Clustering Approximation Technique for Queueing Network Models with a Large Number of Chains", *IEEE Transactions on Computers*, Vol. C-35, no. 5, May 1986.