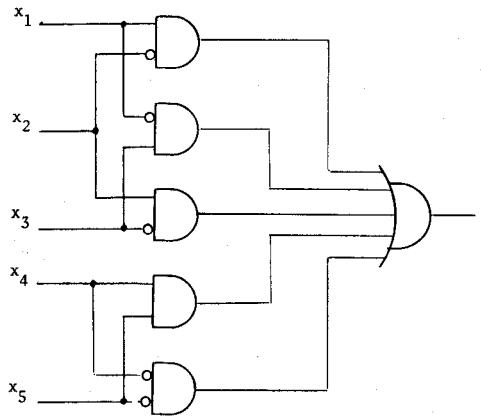

Fig. 5. $f(x_1, x_2, x_3, x_4, x_5) = x_1\bar{x}_2 + \bar{x}_1x_3 + x_2\bar{x}_3 + x_4x_5 + \bar{x}_4\bar{x}_5$.

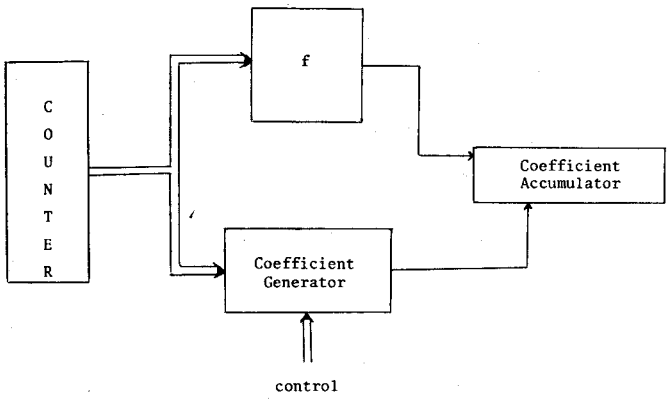

Fig. 6. Basic test scheme.

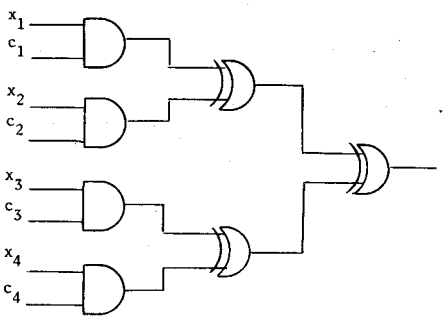

Fig. 7. A four-variable coefficient generator.

tested using very simple test equipment. The test circuitry could be included on board the chip since the overhead involved is comparatively small. Since the test procedure involves a high-speed counter cycling at maximum speed through all input combinations, the network under test is exercised at speed and a number of dynamic errors will be detected which would otherwise have been missed by conventional test-set approaches.

ACKNOWLEDGMENT

The authors would like to thank L. Ochs and J. Ciardi, Tektronix Inc., who initiated the project that led to the work reported here.

REFERENCES

[1] I. Berger and Z. Kohavi, "Fault detection in fanout free combinational networks," *IEEE Trans. Comput.*, vol. C-22, pp. 908–914, 1973.
[2] C. R. Edwards, "Characterization of threshold functions under the Walsh transform and linear translation," *Electron. Lett.*, vol. 11, pp. 563–565, 1975.
[3] Y. M. El-ziq and S. Y. H. Su, "Fault diagnosis of MOS combinational networks," *IEEE Trans. Comput.*, vol. C-31, pp. 129–139, 1982.
[4] S. L. Hurst, *Logical Processing of Digital Signals.* New York: Crane-Russak, 1978.
[5] M. G. Karpovsky, *Finite Orthogonal Series in the Design of Digital Devices.* New York: Wiley, 1976.
[6] I. Kohavi and Z. Kohavi, "Detecting of multiple faults in combinational logic networks," *IEEE Trans. Comput.*, vol. C-21, pp. 556–558, 1972.
[7] J. C. Muzio and S. L. Hurst, "The computation of complete and reduced sets of orthogonal spectral coefficients for logic design and pattern recognition purposes," *Comput. Elec. Eng.*, vol. 5, pp. 231–249, 1978.
[8] J. Savir, "Syndrome testable design of combinational circuits," *IEEE Trans. Comput.*, vol. C-29, pp. 442–451, 1980.
[9] ——, "Syndrome testing of 'syndrome untestable' combinational circuits," *IEEE Trans. Comput.*, vol. C-30, pp. 606–608, 1981.
[10] A. K. Susskind, "Testing by verifying Walsh coefficients," in *Proc. 11th Int. Symp. FTC*, 1981, pp. 206–208.
[11] A. Tzidon, I. Berger, and Y. M. Yoeli, "A practical approach to fault detection in combinational circuits," *IEEE Trans. Comput.*, vol. C-27, pp. 968–971, 1978.

A Simple Derivation of the MVA and LBANC Algorithms from the Convolution Algorithm

SIMON S. LAM

***Abstract*—The convolution algorithm, the mean value analysis (MVA) algorithm, and the LBANC algorithm are major algorithms for the solution of closed product-form queueing networks. For fixed-rate service centers, the efficiency of each algorithm is greatly improved by a recursive solution. We show that the recursive relations in all three algorithms are closely related so that each one can be easily derived from any of the others.**

***Index Terms*—Convolution algorithm, local balance, mean value analysis, queueing networks, recursive solutions.**

I. INTRODUCTION

Three well-known computational algorithms for the numerical solution of closed multichain product-form queueing networks [1] are the convolution algorithm [2]–[5], the mean value analysis (MVA) algorithm [6], and the LBANC algorithm [7]. The efficiency of each algorithm is greatly improved by a recursive relation when dealing with fixed-rate service centers. (The applicable service disciplines at a fixed-rate service center are described in [1].) In the next section, we define our notation and introduce the recursive relations in the three algorithms. A simple derivation of the recursive relations in the MVA and LBANC algorithms from the convolution algorithm's recursive relation is shown in Section III.

II. PRELIMINARIES

Consider a network of M service centers and K closed routing chains. Let λ_{mk} be the mean number of visits of chain k customers to center m between successive visits to center m^*_k (chosen arbitrarily), let τ_{mk} be the mean service time of chain k customers at center m, and define the traffic intensities

$$\rho_{mk} = \lambda_{mk}\tau_{mk}$$

for $m = 1, 2, \cdots, M$ and $k = 1, 2, \cdots, K$. Let N_k be the population size of chain k for $k = 1, 2, \cdots, K$. $\boldsymbol{N} = (N_1, N_2, \cdots, N_K)$ is said to be the population vector of the network.

Manuscript received June 28, 1982; revised September 13, 1982. This work was supported by the National Science Foundation under Grant ECS 78-01803.

The author is with the Department of Computer Sciences, University of Texas at Austin, Austin, TX 78712.

Consider a closed multichain network with population vector $\boldsymbol{N}$. The equilibrium probability of the network state $(\boldsymbol{n}_1, \boldsymbol{n}_2, \cdots, \boldsymbol{n}_M)$ is given by the following product-form solution [1]:

$$\frac{p_1(\boldsymbol{n}_1)p_2(\boldsymbol{n}_2)\cdots p_M(\boldsymbol{n}_M)}{G(\boldsymbol{N})} \qquad \text{for } \mathbf{0} \le \boldsymbol{n}_m \le \boldsymbol{N},\ m = 1, 2, \cdots, M$$

where $\mathbf{0}$ is a K-long vector of zeros, $\boldsymbol{n}_m$ is a K-long vector of nonnegative integers, and $G(\boldsymbol{N})$ is the normalization constant. Each real-valued function p_m can be thought of as a K-dimensional array indexed between $\mathbf{0}$ and $\boldsymbol{N}$. The normalization constant $G(\boldsymbol{N})$ is simply an element of the following array (the element indexed by $\boldsymbol{N}$):

$$g_{\{1,2,\cdots,M\}} = p_1 \circledast p_2 \circledast \cdots \circledast p_M$$

where $\circledast$ denotes a convolution operation between two arrays [3], [4]. In general, if SUBNET $= \{m_1, m_2, \cdots, m_s\}$ is a set of integers chosen from 1 to M, we define

$$g_{\text{SUBNET}} = g_{m_1} \circledast g_{m_2} \circledast \cdots \circledast g_{m_s}.$$

For a network with population vector $\boldsymbol{N}$, define

$L_{mk}(\boldsymbol{N})$ = mean number of chain k customers at center m

$T_{mk}(\boldsymbol{N})$ = throughput of chain k customers at center m

and

$D_{mk}(\boldsymbol{N})$ = mean delay of chain k customers at center m.

The Convolution Algorithm

The convolution algorithm first computes a set of normalization constants and then computes performance measures in terms of the normalization constants. Thus, throughputs are given by

$$T_{mk}(\boldsymbol{N}) = \lambda_{mk}\frac{G(\boldsymbol{N} - \mathbf{1}_k)}{G(\boldsymbol{N})} \tag{1}$$

where $\mathbf{1}_k$ is a unit vector with its kth component equal to one and all others equal to zero, and $G(\boldsymbol{N} - \mathbf{1}_k)$ is the normalization constant of a network with population vector $\boldsymbol{N} - \mathbf{1}_k$. The mean queue lengths in a fixed-rate service center are given by [4]

$$L_{mk}(\boldsymbol{N}) = \rho_{mk}\frac{G_{m+}(\boldsymbol{N} - \mathbf{1}_k)}{G(\boldsymbol{N})} \tag{2}$$

where $G_{m+}(\boldsymbol{N} - \mathbf{1}_k)$ is the normalization constant of a network with population vector $\boldsymbol{N} - \mathbf{1}_k$ and $M + 1$ centers where the extra center has the same set of traffic intensities that center m has. By Little's law [8], we have

$$D_{mk}(\boldsymbol{N}) = L_{mk}(\boldsymbol{N})/T_{mk}(\boldsymbol{N}).$$

The sequential convolution algorithm obtains the array $g_{\{1,2,\cdots,M\}}$ by the following procedure:

$$\begin{aligned} &g_{\{1\}} = p_1 \\ &g_{\{1,2,\cdots,m\}} = g_{\{1,2,\cdots,m-1\}} \circledast p_m \qquad m = 2, 3, \cdots, M. \end{aligned} \tag{3}$$

If center m is a fixed-rate center, then the array $g_{\{1,2,\cdots,m\}}$ can be computed efficiently using the following recursive relation [2], [4]:

$$g_{\{1,2,\cdots,m\}}(\boldsymbol{i}) = g_{\{1,2,\cdots,m-1\}}(\boldsymbol{i}) + \sum_{k=1}^{K} \rho_{mk} g_{\{1,2,\cdots,m\}}(\boldsymbol{i} - \mathbf{1}_k) \qquad \text{for } \mathbf{0} \le \boldsymbol{i} \le \boldsymbol{N}. \tag{4}$$

(Note that throughout this paper, we adopt the convention that any quantity, such as $g_{\{1,2,\cdots,m\}}(\boldsymbol{i} - \mathbf{1}_k)$ above, whose argument has one or more negative components is equal to zero.)

The MVA Algorithm

The MVA algorithm skips the normalization constants and solves for the performance measures directly using the following recursive relation given that center m is a fixed-rate center [6]:

$$D_{mk}(\boldsymbol{N}) = \tau_{mk}\left[1 + \sum_{h=1}^{K} L_{mh}(\boldsymbol{N} - \mathbf{1}_k)\right] \tag{5}$$

$$T_k(\boldsymbol{N}) = \frac{N_k}{\sum_{m=1}^{M} D_{mk}(\boldsymbol{N})\lambda_{mk}} \tag{6}$$

and

$$L_{mk}(\boldsymbol{N}) = \lambda_{mk}T_k(\boldsymbol{N})D_{mk}(\boldsymbol{N}) \tag{7}$$

where $T_k(\boldsymbol{N})$ is the throughput of chain k customers at center m^*_k. Both equations (6) and (7) are based upon Little's law. Equation (5) is the key relation in the MVA algorithm and is related to the arrival theorem [9], [10]. The initial condition for the MVA recursion is

$$L_{mk}(\mathbf{0}) = 0 \qquad \text{for } m = 1, 2, \cdots, M \text{ and } k = 1, 2, \cdots, K.$$

The LBANC Algorithm

Define the unnormalized mean queue lengths

$$q_{mk}(\boldsymbol{N}) = G(\boldsymbol{N})L_{mk}(\boldsymbol{N}) \qquad \text{for all } m \text{ and } k. \tag{8}$$

For center m being a fixed-rate center, the LBANC algorithm uses the following recursion to obtain the unnormalized mean queue lengths and normalization constants [7]:

$$q_{mk}(\boldsymbol{N}) = \rho_{mk}\left[G(\boldsymbol{N} - \mathbf{1}_k) + \sum_{h=1}^{K} q_{mh}(\boldsymbol{N} - \mathbf{1}_k)\right] \tag{9}$$

and

$$G(\boldsymbol{N}) = \frac{\sum_{m=1}^{M} q_{mk}(\boldsymbol{N})}{N_k}. \tag{10}$$

This last equation is a consequence of the observation that

$$\sum_{m=1}^{M} L_{mk}(\boldsymbol{N}) = N_k \qquad \text{for any } k.$$

The initial condition for the LBANC recursion is

$$G(\mathbf{0}) = 1 \text{ and } q_{mk}(\mathbf{0}) = 0 \qquad \text{for all } m \text{ and } k.$$

For the sake of completeness, we note that the recursive solutions described above for the MVA and LBANC algorithms are still applicable if the network consists of both fixed-rate centers and infinite-servers (IS) centers [6], [7]. If center m is an IS center, then (5) in the MVA algorithm is simply replaced by

$$D_{mk}(\boldsymbol{N}) = \tau_{mk} \qquad \text{for all } k$$

and (9) in the LBANC algorithm is simply replaced by

$$q_{mk}(\boldsymbol{N}) = \rho_{mk}G(\boldsymbol{N} - \mathbf{1}_k) \qquad \text{for all } k.$$

II. The Derivation

We next proceed to derive (5) for the MVA recursion starting with the convolution algorithm's recursive relation in (4). We note that

$$G_{m+}(\boldsymbol{N} - \mathbf{1}_k) = g_{\{1,2,\cdots,M\}\cup\{m\}}(\boldsymbol{N} - \mathbf{1}_k).$$

By (4), we have

$$G_{m+}(\boldsymbol{N} - \mathbf{1}_k) = G(\boldsymbol{N} - \mathbf{1}_k) + \sum_{h=1}^{K} \rho_{mh}G_{m+}(\boldsymbol{N} - \mathbf{1}_k - \mathbf{1}_h).$$

Divide both sides by $G(\boldsymbol{N})$, multiply by ρ_{mk}, and applying (2), we get

$$\begin{aligned} L_{mk}(\boldsymbol{N}) &= \rho_{mk}\left[\frac{G(\boldsymbol{N} - \mathbf{1}_k)}{G(\boldsymbol{N})} + \sum_{h=1}^{K} \rho_{mh}\frac{G_{m+}(\boldsymbol{N} - \mathbf{1}_k - \mathbf{1}_h)}{G(\boldsymbol{N})}\right] \\ &= \tau_{mk}\lambda_{mk}\frac{G(\boldsymbol{N} - \mathbf{1}_k)}{G(\boldsymbol{N})}\left[1 + \sum_{h=1}^{K} \rho_{mh}\frac{G_{m+}(\boldsymbol{N} - \mathbf{1}_k - \mathbf{1}_h)}{G(\boldsymbol{N} - \mathbf{1}_k)}\right] \\ &= \tau_{mk}T_{mk}(\boldsymbol{N})\left[1 + \sum_{h=1}^{K} L_{mh}(\boldsymbol{N} - \mathbf{1}_k)\right] \end{aligned}$$

where (1) has been applied. Dividing both sides by $T_{mk}(\boldsymbol{N})$ and applying Little's law, (5) is obtained. Conversely, it should be obvious that starting from (5) in the MVA algorithm, the recursive relation (4) in the convolution algorithm can be derived.

To derive (9), which is the key of the LBANC recursion, we again consider

$$G_{m+}(\boldsymbol{N} - \mathbf{1}_k) = G(\boldsymbol{N} - \mathbf{1}_k) + \sum_{h=1}^{K} \rho_{mh}G_{m+}(\boldsymbol{N} - \mathbf{1}_k - \mathbf{1}_h)$$

based upon the convolution algorithm's recursion. Multiply both sides of the above equation by ρ_{mk}; we get

$$\rho_{mk}G_{m+}(\mathbf{N}-\mathbf{1}_k) = \rho_{mk}\left[G(\mathbf{N}-\mathbf{1}_k) + \sum_{h=1}^{K}\rho_{mh}G_{m+}(\mathbf{N}-\mathbf{1}_k-\mathbf{1}_h)\right]$$

which becomes (9) in the LBANC recursion by observing from (2) and (8) that

$$q_{mk}(\mathbf{N}) = \rho_{mk}G_{m+}(\mathbf{N}-\mathbf{1}_k)$$

for all k and center m being a fixed-rate center.

It is interesting to note that the LBANC recursion is an intermediate step in the sequence of steps that transform the convolution recursion into the MVA recursion. The LBANC recursion involves both normalization constants and mean values, while the convolution recursion involves only normalization constants and the MVA recursion involves only mean values.

IV. Conclusions

The convolution, MVA, and LBANC algorithms provide recursive solutions for product-queueing networks consisting of fixed-rate and infinite-servers centers. We have shown that the recursive relations in all three algorithms are closely related so that each one can be easily derived from any of the others.

References

[1] F. Baskett, K. M. Chandy, R. R. Muntz, and F. Palacios, "Open, closed, and mixed networks of queues with different classes of customers," *J. Ass. Comput. Mach.*, vol. 22, pp. 248–260, Apr. 1975.

[2] J. P. Buzen, "Computational algorithms for closed queueing networks with exponential servers," *Commun. Ass. Comput. Mach.*, vol. 16, pp. 527–531, Sept. 1973.

[3] K. M. Chandy, U. Herzog, and L. S. Woo, "Parametric analysis of queueing networks," *IBM J. Res. Develop.*, vol. 19, pp. 50–57, Jan. 1975.

[4] M. Reiser and H. Kobayashi, "Queueing networks with multiple closed chains: Theory and computational algorithms," *IBM J. Res. Develop.*, vol. 19, pp. 283–294, May 1975.

[5] S. S. Lam and Y. L. Lien, "A tree convolution algorithm for the solution of queueing networks," Dep. Comput. Sci., Univ. Texas, Austin, Tech. Rep. TR-165, Jan. 1981; also, *Commun. Ass. Comput. Mach.*, vol. 26, pp. 203–215, Mar. 1983.

[6] M. Reiser and S. S. Lavenberg, "Mean value analysis of closed multichain queueing networks," *J. Ass. Comput. Mach.*, vol. 27, pp. 313–322, Apr. 1980.

[7] K. M. Chandy and C. H. Sauer, "Computational algorithms for product form queueing networks," *Commun. Ass. Comput. Mach.*, vol. 23, pp. 573–583, Oct. 1980.

[8] J. C. Little, "A proof of the queueing formula: $L = \lambda W$," *Oper. Res.*, vol. 9, pp. 383–387, 1961.

[9] S. S. Lavenberg and M. Reiser, "Stationary state probabilities of arrival instants for closed queueing networks with multiple types of customers," *J. Appl. Prob.*, vol. 17, pp. 1048–1061, 1980.

[10] K. C. Sevcik and I. Mitrani, "The distribution of queueing network states at input and output instants," *J. Ass. Comput. Mach.*, vol. 28, pp. 358–371, Apr. 1981.

Efficient Iterated Rotation of an Object

W. RANDOLPH FRANKLIN

***Abstract*—This paper presents a more efficient method for iterated rotation in three dimensions where multiple points are being rotated by multiple angles about the same axis, as would be done in robotic simulation or computer graphic animation. General axes that do not necessarily pass through the origin and multiple composed rotations are also handled. The algorithm is numerically well conditioned for all axis directions.**

***Index Terms*—Animation, Cayley–Klein parameters, computer-aided design, computer graphics, Euler angle, quaternion, robotics, rotation, transformation.**

Introduction

A frequent operation in computer-aided design (CAD), computer graphics, and computer-assisted animation is the rotation of an object or surface in three dimensions. The usual procedure is to rotate certain fixed points such as vertices or knots such that the remainder of the object follows automatically. We frequently wish to rotate the same object repeatedly about the same axis by successively larger angles. One important use of animation is the verification of programs to control robot manipulator arms. Foley and Van Dam [4, pp. 254–255] present some other efficiency considerations for real-time rotation. If the object is following a more complicated path, each movement can be decomposed into a rotation and a translation or alternatively into two rotations. Optimizing the calculation of the composition of several iterated rotations is important in robot manipulator languages, but at present, calculating this in real time "is beyond the capabilities of most computers" [9, p. 263].

After defining its assumptions and notation, this paper will survey the various general rotation formulas. Some authors present only special cases, such as Chasen [1] who describes how to transform a curve in an oblique plane into another coordinate system. The general methods include: 1) rotation about the three coordinate axes in turn where the axes are either fixed in space or move with the object, 2) rotation using Euler angles or Cayley–Klein parameters, 3) rotation of the coordinate system about the X and Y axes to make the axis of rotation coincident with the Z axis, 4) a vector formulation, and 5) a quaternion formulation. Finally, new formulas optimized for the robot manipulator and animation cases will be presented.

Assumptions and Notation

We rotate a point, represented as a horizontal three-tuple or four-tuple, relative to a fixed set of axes. If the transformation is represented as a matrix, then it postmultiplies the vector representing the point.

The measures of execution time will be τ_+ and $\tau_\times$, respectively, the number of additions and subtractions, and the number of multiplications of floating-point numbers. The time for bookkeeping, integer arithmetic, and so on will be ignored in accordance with conventional procedures. This isolates the essential differences and suppresses variables that depend more on the compiler's efficiency than on the algorithm. We will sometimes count τ_t, the number of trignometric evaluations needed in calculating coefficients of an equation, before it is applied to the points. Other aspects of this precalculation time will generally be ignored.

Rotation About the X, Y, and Z Axes

This formulation appears in Giloi [5], Foley and van Dam [4, pp. 255–259], Paul [9, pp. 25–27], and Rogers and Adams [11]. They compute a rotation matrix as the composition of the following three matrices:

$$R_z(\alpha) = \begin{vmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

$$R_y(\beta) = \begin{vmatrix} \cos\beta & 0 & -\sin\beta \\ 0 & 1 & 0 \\ \sin\beta & 0 & \cos\beta \end{vmatrix}$$

Manuscript received October 30, 1980; revised April 18, 1983. This material is based upon work supported by the National Science Foundation under Grants ENG 79-08139 and ECS 80-21504.

The author is with the Department of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY 12181.

0018-9340/83/1100-1064$01.00