SOME TRANSFORM APPROACHES TO THE COMPUTATION OF THE NORMALIZATION CONSTANT IN QUEUEING NETWORKS

B. EKLUNDH, B. STAVENOW
Lund Institute of Technology, Lund, Sweden

ABSTRACT

Some basic and well-known results for queueing networks are given. The fact that computation of the normalization constant can be viewed as a discrete case convolution is observed and the use of a transform approach to this is suggested. Some transform methods for computation of digital convolution are investigated and applied to the problem of computing G(N), the normalization constant. Through examples, the different methods are compared and put in relation to a direct computation without transforms. It is seen that in many cases the transform approach requires fewer operations and consequently reduces the processing load.

INTRODUCTION

One of the most promising new methods, for analysis of systems, is the theory of queueing networks. It is, with this theory, possible to get, among other things, the state probabilities for the number of customers in the system. The state probabilities indicate how probable it is to find a certain number of customers in a node of the network, where each node represents a server and a queue. The probabilities are calculated from a system of equations, but these equations are, as is usually the case, linearly dependent, and the state probabilities are therefore given within a multiplicative constant. This constant is, in a usual manner, obtained by summing the state probabilities over the entire state space.

The number of states is, however, dependent on the number of nodes and the number of customers in the network, and grows rapidly with these quantities. A straightforward summation of states therefore yields a bulk of computation that quite fast becomes unacceptable. It is, however, possible to use a recursive technique, which substantially reduces the computation load. Each step in the recursion is described as a digital convolution, and is performed in M steps, the number of nodes in the queueing network. It is a well-known fact that convolutions can easily be computed with transform methods, and that convolution corresponds to multiplication of transforms. We will in this paper develop some methods for computation of the aforementioned constant. We will thereby make use of a transform approach and point out some interesting properties of different transforms.

THE PROBLEM, IN ESSENCE

Let us, primarily, suppose that we have a closed queueing network with M service centers and N customers. The states of this network are M-tuples \((n_1, n_2, \ldots, n_M)\), where \(n_i\) is the number of customers at service center \(i\). A feasible state has the property

\[
\sum_{i=1}^{M} n_i = N
\]

and we denote the set of all feasible states by

\[
S(N, M) = \{(n_1, n_2, \ldots, n_M) | 0 \leq n_i \leq N \text{ for } 1 \leq i \leq M \text{ and } \sum_{i=1}^{M} n_i = N\}
\]

A service center is further characterized by its service discipline, and the following types are permitted:

1. Single server, FCFS
2. Single server, processor sharing
3. Infinite server
4. Single server, LCFS-PR

Now, let \(\mu_i(k)\) denote the mean service rate of service center \(i\), when there are \(k\) customers at that center. If the service centers are of types 1-4, it can be shown that the probability that the system is in state \((n_1, n_2, \ldots, n_M)\), which we denote by \(p(n_1, n_2, \ldots, n_M)\), have the product form, i.e.

\[
p(n_1, n_2, \ldots, n_M) = \frac{1}{G(N)} \prod_{i=1}^{M} f_i(n_i)
\]

where

\[
f_i(n_i) = \begin{cases}
\frac{n_i}{\mu_i(j)} & \text{if service center type 1, 2 or 4} \\
\frac{1}{\mu_i} & \text{service center 3}
\end{cases}
\]

The \(e_i\)'s are solutions to the following equations

\[
e_j = \sum_{i=1}^{M} e_{ij} P_{ij}, \quad j = 1, \ldots, M
\]

where \(P_{ij}\) is the transition probability between node \(i\) and \(j\).

\(G(N)\) is the normalization constant which we want to compute.

Since the state probabilities must sum to one, we have that

\[
1 = \sum_{n \in S(N, M)} p(n_1, n_2, \ldots, n_M)
\]

\[
= \frac{1}{G(N)} \sum_{n \in S(N, M)} \prod_{i=1}^{M} f_i(n_i)
\]

and

\[
EKLUNDH / STAVENOW-1
\]
G(N) = \sum_{N(M,N,M) \neq 1} f_{i_1}(n_1) \tag{3}

Buzen (4) introduced, in order to simplify the computation of G(N), a function

\[ G_m(n) = \sum_{m \in S(n,m)} f_{i_1}(n_1) \tag{4} \]

which has the property \( G_m(n) = G(N). \)

With this function the following relation holds

\[ G_m(n) = \sum_{j=0}^{n} f_{m}(j)G_{m-1}(n-j) \quad n=0,1,\ldots,N \tag{5} \]

and we also have that

\[ G_1(n) = f_1(n) \quad n=0,1,\ldots,N \tag{6} \]

which can be viewed as an initial condition.

The recursive relation (5) is similar to a discrete case convolution and if we introduce

\[ f_i = (f_{i_1}(0),f_{i_1}(1),\ldots,f_{i_1}(N)) \]

and \( M \) vectors

\[ G_1, G_2, \ldots, G_M \]

where \( G_1 = f_1 \), we can write

\[ G_m = \ast G_{m-1} \quad m = 2,3,\ldots,M \tag{7} \]

where \( \ast \) denotes the convolution operator.

The discussion above can easily be extended to closed queueing networks with multiple customer classes. In this case we have \( R \) customer classes, and we let \( N_r \) denote the number of customers in class \( r \), \( r = 1,2,\ldots,R \). The state description will in this case be somewhat more complex. The states of the system are given by the vector

\[ (n_1,n_2,\ldots,n_R) \]

where \( n_r \) is the number of customers in service center \( r \). We thus get a multidimensional state description and the set of all feasible states is denoted by

\[ S(N_1,N_2,\ldots,N_R,M) = \{ (n_1,n_2,\ldots,n_M) \} \]

and

\[ \sum_{n_1 \leq m \leq N_1} \sum_{n_2 \leq m \leq N_2} \ldots \sum_{n_R \leq m \leq N_R} \]

The normalization constant can be expressed as

\[ G(N_1,N_2,\ldots,N_R) = \sum_{N_2(N_1,N_2,\ldots,N_R,M)} \sum_{n_1 \leq n_2 \leq N_1} \ldots \sum_{n_R \leq N_R} \]

where \( f_1(\cdot) \) is defined in analogy with the single customer class case.

Define, as earlier, a function

\[ G_m(n_1,n_2,\ldots,n_M) = \sum_{n_1 \leq n_2 \leq N_1} \ldots \sum_{n_R \leq N_R} \]

and observe that

\[ G_m(n_1,n_2,\ldots,n_R) = G(N_1,N_2,\ldots,N_R). \]

The recursive relation will in this case be

\[ G_m(n_1,n_2,\ldots,n_R) = \sum_{m = 2}^{M} G_{m-1}(n_1-v_1,n_2-v_2,\ldots,n_R-v_R) \tag{10} \]

which, as before, is a discrete case convolution. To initiate the recursion we set

\[ G_0(n_1,n_2,\ldots,n_R) = \begin{cases} 1 & \text{if } n_1 = n_2 = \ldots n_R = 0 \\ 0 & \text{else} \end{cases} \tag{11} \]

And we get, with the notation adopted before

\[ G_m = \ast G_{m-1} \quad m = 1,2,\ldots,M \tag{12} \]

where now \( G \) and \( f \) are \( R \)-dimensional arrays.

METHODS FOR CONVOLUTION, ONE DIMENSION

A common way to compute discrete case convolution is to use Discrete Fourier Transform, DFT, and to compute it with FFT, Fast Fourier Transform. The FFT technique is however not better than a direct evaluation of cyclic convolution for sequences smaller than approximately 32.

In recent years, some new methods have, however, been introduced which are better than a direct evaluation for sequence lengths below 32 and which are better than FFT for sequences up to a length of several hundreds. Here we get the word "better" in the sense that the algorithms demand fewer operations, especially multiplications, since these are assumed to be significantly slower than additions. Among these techniques the one developed by Agarwal and Cooley (1) seems to be one of the most promising.

Agarwal and Cooley have developed special algorithms for convolutions, which they call rectangular transforms. These transforms do, however, provide efficient algorithms for very short sequences only, up to approximately 10. In this form they are consequently not very useful, but Agarwal and Cooley devised a general mapping technique due to Agarwal and Burrus (2), by which it is possible to transform an original single dimensional problem into a multidimensional, in which the short convolution algorithms, mentioned above, are used in each dimension. One has to demand that the dimensions in the multidimensional case are mutually prime for the mapping to be one-to-one.

Another way to perform digital convolution is to use the so-called NTTs, Number Theoretic Transforms which are very similar to FFT and which can be computed with a FFT-technique, without multiplications. The calculations are performed in the ring of integers modulo some integer. These transforms suffer, however, severely from restrictions on the length of the sequences that can be transformed and need sometimes to be implemented in hardware to be efficient. The FNT further requires a machine with not too short word length, due to overflow considerations. The restrictions on sequence length can yet be omitted by using the multidimensional technique due to Agarwal and Cooley, but even so it is not applicable in our case.
Table I. Number of Multiplications and Additions for Convolution Using Composite FFT Algorithms (n = sequence length, radices 2, 4, 8)

<table>
<thead>
<tr>
<th>n</th>
<th>Multiplications</th>
<th>Additions</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>8</td>
<td>42</td>
</tr>
<tr>
<td>8</td>
<td>22</td>
<td>114</td>
</tr>
<tr>
<td>16</td>
<td>86</td>
<td>297</td>
</tr>
<tr>
<td>32</td>
<td>214</td>
<td>711</td>
</tr>
<tr>
<td>64</td>
<td>518</td>
<td>1683</td>
</tr>
<tr>
<td>128</td>
<td>1478</td>
<td>3939</td>
</tr>
</tbody>
</table>

Table II. Number of Multiplications and Additions for Short Convolutions by methods proposed by Agarwal - Cooley (n = sequence length)

<table>
<thead>
<tr>
<th>Number of customers</th>
<th>Number of service centers</th>
<th>Direct calculation</th>
<th>FFT</th>
<th>Agarwal - Cooley</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>mult. add. dim.</td>
<td>mult. add.</td>
<td>factors</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>44</td>
<td>65</td>
<td>16</td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td>189</td>
<td>230</td>
<td>32</td>
</tr>
<tr>
<td>20</td>
<td>6</td>
<td>779</td>
<td>860</td>
<td>64</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>1769</td>
<td>1890</td>
<td>64</td>
</tr>
<tr>
<td>40</td>
<td>6</td>
<td>3159</td>
<td>3320</td>
<td>128</td>
</tr>
<tr>
<td>50</td>
<td>6</td>
<td>4949</td>
<td>5150</td>
<td>128</td>
</tr>
</tbody>
</table>

Table III. Comparison between various methods to compute G(N)

Now, to compute G(N) we suggest that each step in the recursion is computed with some of the aforementioned methods. When convolutions are computed by transform methods it can be seen that the sequences have to exceed a certain value for the transform approach to be superior to direct calculation. In a subsequent example we will show that since to receive G(N) one does not need all values given by the transform technique, and consequently the limit at which a transform approach becomes better than direct calculation is raised somewhat. In spite of this the transform methods can be seen to be better than direct calculation when the number of customers in the network is sufficiently large.

EXAMPLE

We will in this example study a closed queueing network with one class of customers, and compare different methods to compute the discrete case convolutions defined in equation (7). The various methods we will investigate, are direct calculation, FFT, and the method proposed by Agarwal - Cooley. We will thereby assume that the vectors \( \mathbf{f} \) in the recursive formula (7) are precomputed, an assumption that seems reasonable, since these vectors are needed also to compute the state probabilities. We further assume, that there are M service centers and N customers in the networks.

Direct calculation

Direct calculation of a noncyclic convolution of length (N+1) corresponds to \( \frac{(N+1)(N+2)}{2} \) multiplications and \( \frac{N(N+1)}{2} \) additions. In our problem, the first component of the vectors \( \mathbf{f} \) is always equal to unity. This reduces the number of multiplications to \( \frac{(N-1)N}{2} \), while the number of additions are unaffected.

To receive G(N), all components of the vector \( G_M \) do not have to be computed, since the component \( G_M(N) = G(N) \), depends only on the vectors \( \mathbf{f} \) and \( \mathbf{G}_{M-1} \). Thus we finally establish that the number of operations needed to compute G(N) by direct calculation is

\[
M-2, \frac{(N-1)N}{2} + N-1 \text{ multiplications}
\]

and

\[
M-2, \frac{N(N+1)}{2} + N \text{ additions}
\]

Fast Fourier Transform (FFT)

Computation of convolutions by means of a FFT-algorithm brings, in addition to some computational advantages, some disadvantages. It introduces round off errors, since it requires processing of trigonometric functions and the vectors \( \mathbf{f} \) and \( \mathbf{G} \) must further be extended to vectors of length at least 2N+1, with the last N components all zero, since a transform method usually performs a cyclic convolution. The FFT-algorithm also requires that the vectorlength is a power of two, a fact that in the same way as the cyclic property tends to increase the length of the vectors. The number of operations needed for convolutions using a composite FFT-algorithm are summarized in table I.
methods were developed by Nussbaumer et al (7). The technique makes use of polynomial transforms, and can be viewed as a NTT computed modulo a polynomial.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|c|c|}
\hline
M & R & N_1 & N_2 & N_3 & Direct calculation & Polynomial transforms \\
& & & & & mult. & dim. \\
& & & & & add. & mult. & add. \\
\hline
3 & 2 & 5 & 10 & - & 1319 & 1385 & 21x21 & 1637 & 18796 \\
3 & 2 & 10 & 10 & - & 4234 & 4355 & 21x21 & 1692 & 18851 \\
6 & 2 & 15 & 10 & - & 3674 & 35375 & 35x35 & 26794 & 77274 \\
6 & 2 & 15 & 15 & - & 72194 & 73215 & 35x35 & 26874 & 77354 \\
6 & 2 & 30 & 30 & - & 977335 & 981180 & 63x63 & 94371 & 1120892 \\
3 & 3 & 5 & 5 & 10 & 28709 & 29165 & 21x21x21 & 34634 & 624030 \\
3 & 3 & 10 & 10 & 10 & 286164 & 287495 & 21x21x21 & 35569 & 624965 \\
6 & 3 & 5 & 15 & 15 & 746594 & 750815 & 35x35x35 & 959774 & -2\times10^7 \\
6 & 3 & 15 & 15 & 15 & -10^7 & -10^7 & 35x35x35 & -10^6 & -2\times10^7 \\
\hline
\end{tabular}
\caption{Comparison between direct calculation and polynomial transform method to calculate \(G(N_1,N_2,\ldots,N_R)\).}
\end{table}

Agarwal-Cooley Approach

The method introduced by Agarwal-Cooley performs, as well as FFT, cyclic convolution, and therefore the vectors must be extended with at least \(N\) zeros. In this method, the original vectors, in our case of length greater than or equal to \(2N+1\), are mapped into multidimensional vectors of dimension \(N_1\times N_2\times N_3\times\ldots\), where the numbers \(N_1, N_2, \ldots\) are relatively prime. The number of multiplications and additions required to convolve these vectors can be derived from the number of operations needed in each dimension.

In table II the number of multiplications and additions for the transforms proposed by Agarwal-Cooley are given.

Finally, we compare in table III, the three methods with respect to the number of operations required. To compute \(G(N)\) only the last component of \(C_M\) is needed, and it is therefore sufficient to perform \((M-2)\) convolutions and to compute the component \(C_M(N)\) by direct calculation. We observe that Agarwal-Cooley's approach is superior to the FFT, in these examples, and that direct calculation is better than the Agarwal-Cooley approach, when the number of customers is small. If the number of customers increases, the ratio (time for addition)/(time for multiplication) must be considered to yield a fair comparison.

METHODS FOR CONVOLUTIONS, MORE THAN ONE DIMENSION

We saw, in the earlier description, that the recursion formula for multiple customer classes had the same basic form as in the single customer class case, but that we had to convolve \(R\)-dimensional arrays in each recursion step. The computation of multidimensional convolution is a most rapidly growing process, what the number of operations are concerned. It is therefore desirable to find a method by which the growth rate could be reduced. Efficient algorithms for multidimensional convolution have been developed by Nussbaumer et al (7). The technique makes use of polynomial transforms, and can be viewed as a NTT computed modulo a polynomial.

EXAMPLE

Let us now consider a queueing network with multiple customer classes. We will compare direct calculation of the normalization constant with a transform method, especially suitable for multidimensional convolution: the polynomial transform method. We assume in this example, that we have \(R\) customer classes, and that each class has a finite number \(N_r, r=1,2,\ldots,R\) of customers. The number of service centers is set to \(M\) and we do not allow customers to change class membership. The number of operations needed to compute a \(R\)-dimensional noncyclic convolution of dimension \((N_1+1)\times(N_2+1)\times\ldots\times(N_R+1)\) by direct calculation can easily be derived in a way similar to the case of one customer class.

Polynomial transforms

The polynomial transform method usually works on multidimensional arrays of dimension \(q^1\times q^2\times q^3\) with \(q\) prime, and performs a cyclic convolution. It is, however, possible to generalize the method, so that convolutions, such that all dimensions have a common prime factor, can be calculated. The method is very complex and we will, therefore, only present some comparisons, which point out some advantages with the method, while we refer to Nussbaumer and Quandalle (7) for a thorough presentation.

It can be seen in table IV that the number of multiplications needed to compute \(G(N_1,N_2,\ldots,N_R)\) with polynomial transforms is in most cases lower than with direct calculation.

An extension of the multiple customer class case is to allow switching between classes. The computation of \(G(\cdot)\) in this case can however with some restrictions be reduced to a form which is similar to the ordinary multiple class case with no class switching, and consequently the algorithms described in the previous case hold for class switching also. There is thus no reason to give a specific example in this case. How to perform the reduction from multiple classes with switching to multiple classes with no switching is described in, for example, (3).
CONCLUDING REMARKS

We have, in the previous paragraphs, outlined some methods for computation of the normalization constant in closed queueing networks. From examples it was seen that in many cases a reduction of the number of operations involved in the computation could be achieved.

The processing load to compute $G(N)$ tends to be great when the number of customers increases, and it is in such cases the transform approach is advantageous, which can be seen from the tables. When the number if customers is small the choice of method is not critical.

The theory of queueing networks could be used for prediction of behaviour in a number of systems, such as Computer Systems, Computer Networks etc. Many new systems are built with microcomputers which usually have no possibilities for multiplication in hardware. In such a case multiplication can be factor a 1000 more timeconsuming than addition, and therefore the algorithms outlined above will be extra advantageous in computers with no hardware multiplication possibilities, such as microcomputers.

The algorithms given in (1, 7) are presented in a general form. Once a specific problem is given, it is, however, possible to tailor an algorithm for that special case that will be even more efficient. This is especially the case with rectangular transforms. We suggest the reader to consult references above for any special algorithm.

REFERENCES


