On the Power Series Algorithm

Ger Koole *
CWI
P.O. Box 94079
1090 GB Amsterdam, The Netherlands

The power series algorithm is a numerical procedure for solving general Markov processes. This paper gives a practical introduction to the algorithm, and presents some new results. We start by showing how the algorithm can be applied to a specific problem, the fork-join queue. Then we prove that the power series algorithm can indeed be used to solve general Markov processes. In the subsequent section we deal with the convergence properties of the algorithm. It behaves particularly well when applied to finite state processes, which is illustrated with the analysis of a bounded Petri net. We end with discussing the literature.

1 Introduction

Analytically obtaining performance measures of multi-dimensional queueing systems is often very difficult. Explicit solutions are only available for some very special models, like Jackson networks. Some specific two-dimensional models can also be solved analytically, for example by showing that solving the problem is equivalent to solving a well-studied complex analysis problem. See Boxma et al. [16] for an overview. The drawbacks of the analytical methods can be summarized as follows: the resulting problems are non-trivial to solve, we are confined to two dimensions, and small changes in the model usually lead to analytically intractable models.

On the other hand, simply numerically solving the steady state equations usually does not work well either. Often the state space is countable, giving need to truncate the state space at an appropriate level. These truncated state spaces are very big, especially if the dimension grows large and if the accuracy must be high, resulting in very long running times. Thus there is a need for efficient numerical methods to solve large Markov processes. The power series algorithm (psa) aims to be such a method. It was first developed by Hooghiemstra et al. [17] for a model in which several queues share the same servers, and later applied to several other queueing models in a series of papers by Blanc and co-authors (discussed in section 5). The examples in the next sections show that the psa often works very well, although there are some theoretical gaps.

*Supported by the European Grant BRA-QMIPS of CEC DG XIII

In section 2 we introduce the psa by applying the method to a fork-join queue, for which the algorithm works remarkably well. In section 3 it is shown that the psa can (formally) be applied to any Markov process with a single recurrent class. Formally, as there is no guarantee that the obtained power series converge. However, in section 4 we study the ϵ-algorithm which is capable of finding a limit for a divergent series based on its partial sums. It is shown that this works especially well for finite state processes. To illustrate this, the psa with the ϵ-algorithm is then applied to a bounded Petri net. The paper ends with a section discussing the literature.

2 THE FORK-JOIN QUEUE

In this section we give a practical introduction to the power series algorithm by applying it to a simple model with distributed processing, the *fork-join queue*. The fork-join queue consists of two or more parallel queues. Jobs arrive at the system according to a Poisson process with rate λ, and on arrival they place exactly one task in each queue (the fork primitive). This makes the arrival processes of the different queues dependent. The processing of the tasks in each queue however is independent of tasks in other queues, and is exponential with rate μ_i at queue i. Thus the marginal behaviour of each queue is that of a simple $M|M|1$ queue. A job leaves the system if all its tasks have finished service (the join primitive). As performance measure one usually takes the number of jobs in the system, which is equal to the maximum of the queue lengths (assuming that each queue services its tasks in FIFO order). This measure clearly depends on the simultaneous queue length distribution, giving need for methods to calculate it.

Many papers study the fork-join model, by using analytical methods, or by developing approximations or bounds. See section 3.1 of [16] for an overview. In this section we use the power series algorithm to compute the steady state distribution.

We use the following notation.

m - the number of queues
$x = (x_1, \ldots, x_m)$ - the state of the system
$|x| = x_1 + \ldots + x_m$
p_x - the stationary probability of state x
(A) - the indicator function of the event A
$e = (1, \ldots, 1)$
$e_i = (0, \ldots, 0, 1, 0, \ldots, 0)$ with the 1 in ith position

First take $\rho = \lambda^{1/m}$. (The choice of ρ will be motivated in the next section.) We try to write the stationary distribution as a power series of ρ. Thus we write p_x as

$$p_x = \sum_{k=0}^{\infty} a_{kx} \rho^k,$$

with a_{kx} the fixed coefficients of the power series. For the moment, assume that these power series converge. In the next section we show that $a_{kx} = 0$ if $k < |x|$. Therefore we prefer to write

$$p_x = \sum_{k=0}^{\infty} b_{kx} \rho^{|x|+k}. \tag{1}$$

The steady state equations are

$$\{\rho^m + \sum_{i=1}^{m} (x_i > 0)\mu_i\} p_x = (x > 0)\rho^m p_{x-e} + \sum_{i=1}^{m} \mu_i p_{x+e_i}. \tag{2}$$

Together with the normalizing equation

$$\sum_x p_x = 1, \tag{3}$$

they uniquely determine the steady state probabilities (assuming that the system is stable, i.e., that $\lambda = \rho^m < \min_i \mu_i$).

Now we insert (1) in (2) and (3). Equating the terms with $\rho^{|x|+k}$ in (2) gives

$$(k \geq m) b_{k-m,x} + \sum_{i=1}^{m} (x_i > 0)\mu_i b_{kx} = (x > 0) b_{k,x-e} +$$

$$(k \geq 1) \sum_{i=1}^{m} \mu_i b_{k-1,x+e_i}.$$

From (3) we get

$$b_{0,(0,0)} = 1; \; \sum_{x:|x| \leq k} b_{k-|x|,x} = 0, \; k > 0.$$

The important observation is that we can compute the b_{kx} recursively. Starting with $b_{0,(0,0)} = 1$, we can compute successively the b_{0x} for all x with $|x| = 1, 2, \ldots$. (In practice, we compute these numbers up to a certain level, say until $|x| \leq K$.) Subsequently, we compute the $b_{1,(0,0)}$ using the normalizing equation, and we can compute b_{1x} for x with $|x| = 1, 2, \ldots$, etc. This way all coefficients with $k + |x| \leq K$ are computed.

Now the approximation is calculated by taking all other coefficients 0, which is equivalent with omitting all terms of order higher than K. Numerical results for this model with $m = 2$, $\mu_1 = 1$, $\mu_2 = 2$ and varying λ can be found in table 1. For each combination of K and λ the approximation of p_{00} and of $L = \sum_x \max\{x_1, x_2\} p_x$, the mean number of jobs in the system, is given. The exact values were calculated by truncating the state space at a sufficiently high level, and then solving the steady state equations iteratively. Note how well the psa performs, even for very small values of K, over the whole stability region of the model (the stability condition is here $\lambda < 1$). Only for λ close to 1, we have to take K large to get a reasonable estimate of L. On the other hand, even for $\lambda = 1$, for which the system is unstable, the approximation of p_{00} converges fast to 0. (Note that some computations were omitted in this case, indicated with "-".)

λ	actual values	$K=5$	$K=20$	$K=50$	$K=100$
.1	.8849, .1275	.8849, .1265	.8849, .1275	.8849, .1275	.8849, .1275
.5	.4578, 1.0688	.4568, .8279	.4578, 1.0678	.4578, 1.0688	.4578, 1.0688
.75	.2184, 3.0706	.2153, 1.4253	.2184, 2.9017	.2184, 3.0683	.2184, 3.0706
.9	.0849, 9.0419	.0800, 1.8425	.0849, 5.9041	.0849, 8.3959	.0849, 8.9956
1	0, ∞	−.0062, -	.0000, -	.0000, -	.0000, -

Table 1. Approximations of p_{00} and L for the fork-join queue with $\mu_1 = 1$, $\mu_2 = 2$

3 General Markov processes

In the previous section we applied the psa to the fork-join queue, showing that the coefficients of the power series of the stationary probabilities can be computed recursively, and using these to compute performance measures. In this section we show that the coefficients of these power series can be computed recursively for any Markov process, if the variable ρ of the power series is incorporated in the model in a suitable way.

Thus we start with an arbitrary Markov process with state space X (possibly countable) to which we want to apply the power series algorithm. We denote the transition rate from x to y by q_{xy} (for ease of notation we assume throughout that $q_{xx} = 0$). To use the psa, we consider an additional Markov process with the same state space X, but with the transitions replaced by $\rho^{f(x,y)}q_{xy}$. The problem is how to choose $f(x,y)$ such that the algorithm works for the additional process. By inserting $\rho = 1$ afterwards, we get the results for the original process. Of course, we can choose other values for ρ (as long as it amounts to a useful model), like we did in the previous section for the fork-join queue.

First we associate with every state x a *level* $l(x)$ ($l(x) \in \{0,1,\ldots\}$). The idea is to write the stationary probability p_x as

$$p_x = \sum_k b_{kx}\rho^{l(x)+k}, \tag{4}$$

assuming that these series converge.

We show that the right choice for f is $f(x,y) = (l(y) - l(x))^+$, i.e., all transitions are of the form $\rho^{(l(y)-l(x))^+}q_{xy}$. Thus transitions to lower level states are not changed, but transitions to higher levels get a factor ρ for each level the next state is higher. To make the psa work we have to assume the following.

Assumption 3.1 *The states can be classified in levels* $0,1,2,\ldots$ *such that:*
(i) There is a single level 0 *state (denoted by* 0*).*
(ii) The states within each level can be ordered such that there are no transitions to higher ordered states within that level.
(iii) $\sum_{y:l(y)\leq l(x)} q_{xy} > 0$ *for all* $x \in X$*, i.e., transitions to lower level states are possible in each state.*

Because state 0 can be reached from every other state in a finite number of steps it follows directly that there can only be a single recurrent class. This is the only restriction implicated by assumption 3.1: at the end of the section we will see that we can order the states of any Markov process with a single recurrent class such that the assumption is satisfied. The choice of the ordering however can have important implications for the speed of the algorithm.

Note that the assumption implies a partial ordering of the states: $x \prec y$ if $l(x) < l(y)$ or if $l(x) = l(y)$ and there is no transition from x to y. We assume that the states are numbered $0, 1, 2, \ldots$ such that if $x \prec y$, then $x < y$.

As an illustration, consider the fork-join queue of the previous section. There we took $l(x) = |x|$, and indeed, an arrival (which increases the level by m) has a factor ρ^m. Moreover, assumption 3.1 is satisfied: the empty state is the single level 0 state, there are no transitions within each level, and in each state (except 0) there is at least one non-empty queue, making transitions to lower level states possible.

Another possible choice for l in the fork-join model is $l(x) = \max_i x_i$. Then every arrival just has a factor ρ. Now there are transitions possible within each level, but it is easily seen that they can be ordered as required in assumption 3.1(ii). Note that if the maximum is attained by two or more queues there are no transitions to strictly lower levels, but only within the same (and to higher) levels. However, that is all that is required by assumption 3.1(iii). It is easily seen that both choices of $l(x)$ basically lead to the same approximations.

The next theorem states that we can indeed write p_x in the form (4), that is, that $p_x = O(\rho^{l(x)})$.

THEOREM 3.2 *Under assumption 3.1, $p_x = O(\rho^{l(x)})$.*

Proof. We are going to use the idea of the equivalent proof given in [18] for the $BMAP|PH|1$ queue studied there. We use induction, first considering p_0. Because $p_0 = 1$ if $\rho = 0$ it is clear that $p_0 = O(1)$. Define $L_z = \{x | x \leq z\}$. Assume that $p_x = O(\rho^{l(x)})$ for all $x \in L_z$. We complete the induction step by looking at the balance equation between states in L_z and states in $X \backslash L_z$:

$$\sum_{x \in L_z} \sum_{y \notin L_z} \rho^{l(y)-l(x)} q_{xy} p_x = \sum_{x \notin L_z} \sum_{y \in L_z} q_{xy} p_x.$$

Now we show that $z' = z + 1$ is of the required order. Using the induction hypothesis, and the structure of the transitions, it is clear that the left hand side of the equation is of order $O(\rho^{l(z')})$. For z' we have that $\sum_{y \in L_z} q_{z'y} > 0$, and thus (using that all coefficients are non-negative) $p_{z'} = O(\rho^{l(z')})$. □

Next we derive the equations for which we can recursively compute the b_{kx}. The equilibrium equations are:

$$\sum_y \rho^{(l(y)-l(x))^+} q_{xy} p_x = \sum_y \rho^{(l(x)-l(y))^+} q_{yx} p_y.$$

Inserting $p_x = \rho^{l(x)} \sum_k \rho^k b_{kx}$ gives:

$$\sum_y \rho^{(l(y)-l(x))^+} q_{xy} \rho^{l(x)} \sum_k \rho^k b_{kx} = \sum_y \rho^{(l(x)-l(y))^+} q_{yx} \rho^{l(y)} \sum_k \rho^k b_{ky}.$$

Consider for fixed x the terms with $\rho^{l(x)+k}$:

$$\sum_{y:l(y)\leq l(x)} q_{xy} b_{kx} + \sum_{y:l(y)>l(x)} q_{xy} b_{k-l(y)+l(x),x} =$$
$$\sum_{y:l(y)\leq l(x)} q_{yx} b_{ky} + \sum_{y:l(y)>l(x)} q_{yx} b_{k-l(y)+l(x),y}. \tag{5}$$

From this equation we can derive b_{kx}, for $x \neq 0$, assuming we have already calculated b_{ky} for $y < x$ and b_{ly} for $l < k$ and sufficiently many y (depending on the model at hand). This can only be done if the coefficient of b_{kx} is positive (which is guaranteed by assumption 3.1(iii)) and if $q_{yx} = 0$ if $y > x$ and $l(y) = l(x)$ (guaranteed by assumption 3.1(ii)). This procedure can be repeated until all coefficients which are needed have been calculated.

The b_{k0} can be determined from $\sum_x p_x = 1$: it easily follows that $b_{00} = 1$ (if $\rho = 0$ this is the only recurrent state, by assumption 3.1(i)) and that for $k > 0$

$$\sum_{x:l(x)\leq k} b_{k-l(x),x} = 0.$$

If there is more than one level 0 state (which is the case with multiple recurrent classes) each recurrent class should be handled separately.

An important aspect of the method is the choice of l. We already saw that for specific models l could be chosen in some smart way. However, to show that the psa is applicable to general Markov processes with a single recurrent class we have to specify a choice of l which always satisfies assumption 3.1. To do so, number the states 0,1,2,..., such that from state n there is a transition possible to one or more states in $\{0, 1, \dots, n-1\}$. This can be done easily, as long as state 0 is taken to be recurrent. Now take $l(x) = x$ for each state, and assumption 3.1 is satisfied.

So far, we have only talked about continuous time Markov processes, and not about discrete time Markov chains. They can be solved as well, simply by taking $\rho = 1$ in a model with $\sum_y q_{xy} = 1$ for each x. The only complication is that we cannot assume $q_{xx} = 0$. However, it is readily seen that the term $q_{xx} b_{kx}$ cancels on both sides of (5).

3.1 Examples

In this subsection we discuss the choice of l for several well known queueing models. Note however that for the applicability of the method the actual transition rate is not important, but just whether it is positive or not. This gives the possibility to change the models considerably without choosing another l.

Birth-death processes. Let us first consider a class of models which comprise the models discussed so far, the *m-dimensional birth-death processes*. Such a process can be seen as consisting of m queues, where arrival and departure rates depend on the state, and can occur in batches (in different queues simultaneously), but in which no arrivals and departures can occur simultaneously, avoiding transitions between queues. Thus the possible transitions out of x are of the form $x \to x+y$ or $x \to x-y$, with $y \geq 0$ (and $x-y \geq 0$). We assume that for each $x \neq 0$ there is a $y \neq 0$ such that $q_{x,x-y} > 0$. If we take $l(x) = |x|$, it is easily seen that these m-dimensional birth-death processes satisfy assumption 3.1. It is also a rich class. Not only the examples of section 2, the fork-join queue and the shortest queue model fall into it, but also the model of [5, 17] and numerous other models, like single server queues with batch arrivals, belong to it. Note that in some cases the levels can be chosen more economically, as we saw for the fork-join model.

Networks of queues. A tandem of queues is an example of a model which does not fall in the class of problems described above, but where we can take $l(x) = |x|$. Indeed, if customers enter queue 1, and join after service queue 2, ..., up to m, then the possible transitions within each level are all of the form $x \to x - e_i + e_{i+1}$, giving an ordering within level k: $(k, 0, \ldots, 0) \prec \ldots \prec (0, \ldots, 0, k)$.

For models with a more general routing structure, as in Jackson networks, this does not work any more; cycles within a level become possible. A solution is to take as state space $(x_1 + \ldots + x_m, x_2 + \ldots + x_m, \ldots, x_m)$, or, equivalently, to take $l(x) = x_1 + 2x_2 + \ldots + mx_m$. For this choice of l we can allow transitions from one queue to another, i.e., transitions of the form $x \to x - e_i + e_j$ (for x with $x_i > 0$), in addition to the batch arrivals and departures from the m-dimensional birth-death process.

Another approach, which does not fit into the framework of this section, is when we take again $l(x) = |x|$, but a transition of the form $x \to x - e_i + e_j$ with $i > j$ gets a factor ρ. Thus we have given a transition from queue i to queue j a factor ρ, although the states lie within the same level. For the other transitions the factors are taken normally, including the transitions from queue i to j if $i < j$. The psa works again in this case, and the ordering within a level is the same as for the tandem model.

Now we study models where the state of the system is not completely described by the queue lengths only: we consider polling models, where the position of the server belongs to the state, and models with an additional Markov process representing the environment (generalizing the arrival or service processes).

Markov arrival processes. First consider a single queue with arrivals according to a Markov arrival process (MAP). Assume that the states y of the MAP are numbered, such that the psa can be applied to the Markov process underlying the MAP, with levels $l(y) = y$ (which gives the restriction that there must be a single recurrent class). The states are of the form (x, y), with y the state of the MAP and x the number of customers in the queue. State (x, y) has level $x + y$. The only possible transitions within a level are of the form

$(x, y) \to (x+1, y-1)$, thus assumption 3.1(ii) is easily satisfied. The same holds for assumption 3.1(iii), assuring that the psa works for this model. Note that the rates at which arrivals occur do not necessarily have a factor ρ in it (as in the transition above), because the state of the MAP changes also. Only if the state of the MAP remains the same at arrival instants (the special case of a Markov modulated Poisson process), then each arrival has factor ρ. The term MAP is somewhat misleading, as it suggests that the transition rates within the Markov process governing the arrivals must be independent of x. As this is not true, it is perhaps better to speak of an auxiliary Markov process.

General service times. Such an auxiliary Markov process (AMP) can also be used to model (potential) departures from a queue with Poisson arrivals. However, when modeling departures, it is more natural to freeze the AMP (i.e., keep it in the same state until a customer arrives) when the queue is empty, instead of letting it make transitions without having customers in the queue to serve. But, as transitions to lower level states must be possible from each state except 0, it can only be frozen in state 0, which is therefore of the form $(0, 0)$. Thus the AMP can only be frozen if the transition in the AMP generating the departure is of the form $y \to 0$. If we want to be able to freeze the AMP in different states, a less obvious choice of levels has to be made.

Polling models. An interesting generalization of an auxiliary Markov process governing departures (and possibly also arrivals) is to multiple queues. As Blanc [10] shows, an important class of models which can then be modeled are the polling models. In its simplest form, the state of the AMP denotes the position of the server (i.e., at or between which queues the server is), but generalizations in different directions are possible, like the AMP denoting the service phase, or even the number of customers already served at the current queue, to be able to model for example the limited service discipline. When the server in a polling system finds an empty queue, the server usually moves to the next server; therefore the problem with freezing the departure process occurs only in the single queue case.

3.2 Memory management

At first sight it seems that what we gain by faster computations is lost again by the extra memory use, because we increased the dimension with 1. Although we need some more memory than simple iterative methods, this is usually not true. If we want to compute the stationary distribution for fixed ρ or if we want to compute a function of this distribution (like the moments of the stationary number of customers) for varying ρ, we need not keep all b_{kx} in memory.

We approximate the stationary distribution using all terms with coefficients ρ^k, with $k \leq K$. Let $N = \#\{x | l(x) \leq K\}$, the number of states with level equal to or smaller than K. Assume that the maximum number of levels a transition can go up is $\bar{k}$. It is best to compute the coefficients b_{kx} for constant value of $k + l(x)$ together. (This has the advantage that the partial sums, needed for the normalizing equation, can directly be computed and need not be kept in memory separately.) It is clear from (5) that to compute the b_{kx} with $k + l(x) = n$, only the coefficients $b_{k'x'}$ with $k' + l(x') = n - \overline{k}, \dots, n-1$ need to be kept in memory. As soon as a coefficient is computed, it is used to update the partial sums approximating p_x. For example, in the fork-join queue only two levels are needed, while in the shortest queue model even 1 level is sufficient. In total, $N(\overline{k}+2)$ numbers need to be kept in memory: $N\overline{k}$ for the already computed coefficients, N for the coefficients now to be computed, and N for the approximations of the stationary probabilities. (Note that if $\overline{k} > K$, all coefficients need to be kept in memory.)

Now consider the situation that we want to compute a certain performance measure $\mathbb{E}f = \sum_x f(x)p_x$, say for various values of ρ. Instead of having approximations for each p_x, we keep in memory the coefficients of the power series expansion of $\mathbb{E}f$. If a coefficient b_{kx} is computed, it is added to the $(k + l(x))$th coefficient of the expansion of $\mathbb{E}f$. Now, $N(\overline{k}+1) + K$ numbers are necessary. Note that the same performance measure can be computed for several ρ. Using this, the coefficients b_{kx} need only to be computed once to supply the results produced by the psa for the examples of section 2. If we are only interested in $\mathbb{E}f$ for a specific value of ρ, then $N(\overline{k}+1)+1$ numbers are sufficient.

4 Convergence

In section 2 we saw that the power series expansions of p_{00} and L in the fork-join queue converge for all values of ρ for which the system is stable. However, in general this need not be the case. This is not surprising: the psa develops each stationary probability as a power series around $\rho = 0$, and the radius of convergence of such a series is in general unknown. This section is devoted to the study of the convergence properties.

To illustrate the problems, we consider the *shortest queue model* in section 4.1, and show that the power series involved do not converge for certain values of ρ. To improve the convergence properties we make use of an algorithm applicable to arbitrary power series, the *ϵ-algorithm*, which was first used by Blanc in conjunction with the psa. The ϵ-algorithm is the subject of section 4.2.

In section 4.3 we take a closer look at finite state Markov processes. The ϵ-algorithm is particularly well suited to deal with this type of processes. Finally in section 4.4 we use the results to analyze a bounded Petri net.

4.1 The shortest queue model

In the shortest queue model, we have a Poisson(λ) stream of jobs arriving at m parallel queues. Each job consists of a single task which is routed to the shortest queue. In case of a tie, each queue is selected with equal probability. The server at queue i again has service rate μ_i. Simply choosing $\rho = \lambda$ makes the psa work here. We will not go into all details of the balance equations, but we just give the equation from which the b_{kx} are derived (for $m = 2$):

$$(k>0)b_{k-1,x} + \sum_{i=1,2}(x_i>0)\mu_i b_{kx} =$$

$$\Big((x_1>0)(x_1\le x_2)+\tfrac{1}{2}(x_1=x_2+1)\Big)b_{k,x-e_1}+$$

$$\Big((x_2>0)(x_2\le x_1)+\tfrac{1}{2}(x_2=x_1+1)\Big)b_{k,x-e_2}+$$

$$(k>0)\sum_{i=1,2}\mu_i b_{k-1,x+e_i}.$$

The numerical results for $\mu_1 = 1$, $\mu_2 = 2$ and $K = 50$ can be found in table 2. For different values of λ, p_{00} and $L = \sum_x (x_1 + x_2)p_x$, the mean number of jobs in the system, are computed. Clearly, the power series expansions for $\lambda = 1.5$ do not converge; in this case the power series expansion of the p_x converges only if $\lambda \le 1$. Note that the stability condition here is $\lambda < \mu_1 + \mu_2$, thus the psa does not always give satisfactory answers in the whole stability region. Elaborate numerical experiments with this model can be found in [4, 11].

λ	actual values	$K = 50$
.1	.9279, .0746	.9279, .0746
.5	.6859, .3824	.6859, .3825
1	.4551, .9523	.4550, .8526
1.5	.2806, 1.5401	−29.5875, 30.8450

Table 2. Approximations of p_{00} and L for the shortest queue model with $\mu_1 = 1$, $\mu_2 = 2$

4.2 The ϵ-algorithm

The ϵ-algorithm was introduced by Wynn (see e.g. [20]) to accelerate the convergence of power series. Given the partial sums $S_m = \sum_{k=0}^{m} c_k \rho^k$, a two-dimensional array with entries $\epsilon_r^{(m)}$ is computed, using the formula

$$\epsilon_{r+1}^{(m)} = \epsilon_{r-1}^{(m+1)} + (\epsilon_r^{(m+1)} - \epsilon_r^{(m)})^{-1},$$

with initial conditions

$$\epsilon_{-1}^{(m)} = 0,\ m = 1, 2, \ldots,$$

and

$$\epsilon_0^{(m)} = S_m,\ m = 0, 1, \ldots.$$

Now $\epsilon_r^{(m)}$ with r even is used instead of S_m to approximate the limit S_∞. The numbers $\epsilon_r^{(m)}$ with r odd are only used as intermediate results.

The idea behind the ϵ-algorithm is that $\epsilon_{2r}^{(m)}$ approximates S_∞ by a quotient of polynomials, the numerator of degree $m + r$, the denominator of degree r, which are completely determined by the first $2r + m$ coefficients of the power series to be approximated. In the cases considered in this paper, the zeros of the denominator apparently converge to the singularities of S_∞, thereby extending the region of convergence.

Although the ϵ-algorithm involves repeated subtraction and division, Wynn [20] states that it is often remarkably stable. This is in compliance with our findings.

We applied the psa with the ϵ-algorithm to the shortest queue model of section 2. The results can be found in table 3. We see that the psa together with the ϵ-algorithm gives the correct answers for all values of λ.

λ	actual values	$K = 50$
.1	.9279, .0746	.9279, .0746
.5	.6859, .3824	.6859, .3825
1	.4551, .9523	.4550, .8526
1.5	.2806, 1.5401	.2805, 1.5415
2	.1509, 2.7628	.1506, 2.7751
3	0, ∞	.0000, 2.28×10^6

Table 3. Approximations of p_{00} and L for the shortest queue model with $\mu_1 = 1$, $\mu_2 = 2$, using the ϵ-algorithm

For the models we studied the ϵ-algorithm works very well. However, if $\epsilon_r^{(m+1)} = \epsilon_r^{(m)}$ for some values of r and m, problems may arise, because $\epsilon_{r+1}^{(m)}$ cannot be computed any more. A simple example is $(1-\rho^3)^{-1}$, for which the power series expansion has coefficients $1, 0, 0, 1, 0, 0, \ldots$. Application of the ϵ-algorithm easily shows what goes wrong. On the other hand, sometimes we find $\epsilon_{2r}^{(m+1)} = \epsilon_{2r}^{(m)}$ for all $m \geq m'$. Then $\epsilon_{2r}^{(m')}$ is exactly $\sum_{k=0}^{\infty} c_k \rho^k$. Some special cases for which this occurs are identified in the next subsection.

Remark. Apart from the ϵ-algorithm, a method using conformal mappings can be applied to improve the convergence properties. It is based on putting $\rho = \theta/(1 + G - G\theta)$ (with G a positive constant), and then calculating p_x as a power series in θ. Using the notation

$$p_x = \sum_{k=0}^{\infty} u_{kx} \theta^{l(x)+k},$$

it can be shown that the u_{kx} can be computed recursively if the b_{kx} can be computed recursively. Moreover, all singularities of the expansion of p_x inside the unit disk (causing the power series not to converge for $\rho = 1$) but outside the disk with center $\rho = 1/2$ and radius $1/2$, can be removed with such a conformal mapping. Thus, the psa can again be applied, and the convergence properties are considerably improved. For more details, see for example [5].

4.3 Finite State Processes

In this section we study Markov processes which have a finite state space, with N elements. This will allow us to write the stationary probabilities as quotients of polynomials in ρ. From this we conclude that the ϵ-algorithm, if applicable, produces exact results.

Let G be the infinitesimal generator of the Markov process, i.e., $g_{ij} = q_{ij}$ if $i \neq j$, and $g_{ii} = -\sum_j q_{ij}$. Construct G' from G by replacing the last column by e. Then the steady state vector is the unique solution of the equation $pG' = e_N$, if we assume that the process consists of a single recurrent class. Note that all elements of G' are polynomials of ρ.

To compute the stationary probabilities p_x we can apply Cramer's rule, that is,

$$p_x = \frac{|G'_x|}{|G'|},$$

where G'_x is obtained from G' by replacing the xth row by e_N, and where we denote by $|\cdot|$ the determinant of a matrix. As all entries of both matrices are polynomials in ρ, we conclude that p_x is a quotient of polynomials in ρ, i.e., it is a rational function.

Again, assume that the maximum number of levels a transition can go up is $\overline{k}$. Then all entries are of order $\leq \overline{k}$, and as the last column consists of 1's, both determinants are of order $\leq (N-1)\overline{k}$. As it is useless to have more levels than states, and thus $\overline{k} \leq N-1$, we can assume in general that each determinant is of order $\leq (N-1)^2$.

From [20] we know that $\epsilon_{2r}^{(0)}$ approximates S_∞ with a uniquely determined rational function where both the numerator and the denominator are of order r. Thus to compute the stationary probabilities exactly it is sufficient to compute $\epsilon^{(0)}_{2(N-1)\overline{k}}$. If $\overline{k}$ is small (in most examples we had $\overline{k} = 1$), this can often be done, even for reasonably sized models.

Another interesting implication of p_x being a rational function is that p_x is analytic in $\rho = 0$. Indeed, p_x has a finite number of poles, each of which is unequal to 0, as $p_x = (x = 0)$ for $\rho = 0$. Thus, for ρ small enough, the power series converge, without applying the ϵ-algorithm. If ρ is the traffic intensity (as it was in most queueing examples), this leads to a light traffic result.

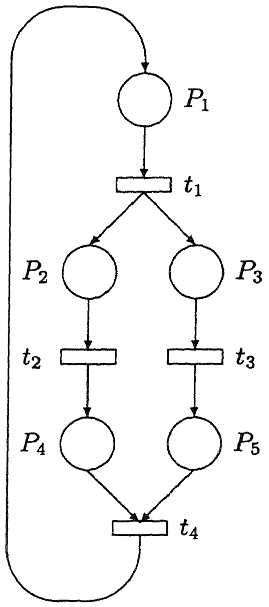

Figure 1. A stochastic Petri net

4.4 Petri Net Example

To illustrate the ideas from the previous subsection, we analyze the simple stochastic Petri net depicted in figure 1. We denote its markings with $(x_1, \ldots, x_5)$, where x_i is the number of tokens at place P_i. As initial markings we take $(n, 0, 0, 0, 0)$, for various n. This marked graph is live and bounded, and to represent its reachability set we can restrict ourselves to (x_1, x_2, x_3), as $x_4 = n - x_1 - x_2$ and $x_5 = n - x_1 - x_3$. Transition t_i has an exponential firing time with rate λ_i.

Note that this Petri net is strongly related to the fork-join queue. Indeed, transition t_1 corresponds to the fork primitive. Transitions t_2 and t_3 correspond to the distributed processing of the tasks, and transition t_4 is only enabled if there is both a token at P_4 and at P_5, that is, if a job has finished service in the fork-join queue. Thus, in queueing terms, the Petri net exists of a closed cycle of three centers, one of which is a fork-join queue, and two of them are simple single server queues.

To apply the psa, we have to partition the state space into levels. We took as levels $l(x_1, x_2, x_3) = n - x_1$. Consequently, transition t_1 gets a term ρ, thus λ_1 is replaced by $\rho\lambda_1$. Equation (5) becomes:

$$b_{kx}\{\lambda_2(x_2 > 0) + \lambda_3(x_3 > 0) + \lambda_4(x_1 + x_2 < n, x_1 + x_3 < n)\} +$$

$$b_{k-1,x}\lambda_1(x_1 > 0, k > 0) =$$

$$b_{k,(x_1+1,x_2-1,x_3-1)}\lambda_1(x_2 > 0, x_3 > 0) +$$

$$b_{k,(x_1,x_2+1,x_3)}\lambda_2(x_1 + x_2 < n) + b_{k,(x_1,x_2,x_3+1)}\lambda_3(x_1 + x_3 < n) +$$

$b_{k-1,(x_1-1,x_2,x_3)}\lambda_4(x_1 > 0, k > 0)$.

We are interested in the throughput of the system, i.e., the average number of firings of t_1 per unit of time. This is equivalent to computing the stationary probability of having 0 tokens in P_1 (denoted by p), as the throughput is equal to $(1-p)\lambda_1$.

First we computed the coefficients of the power series of p. As $p = \sum_{x_2+x_3 \leq n} p_{(0,x_2,x_3)}$, this is the sum of the stationary probabilities of all level n states. Thus, the first n coefficients of this power series are 0. There are no transitions 2 or more levels up, and therefore only 2 arrays the size of the state space (which is $N = 1^2 + 2^2 + \ldots + (n+1)^2 = (n+1)(n+2)(2n+3)/6$) and 1 array with the coefficients of p need to be kept in memory, according to section 3.2. After computing all coefficients of p up to a certain K, we applied the ϵ-algorithm, after omitting the trailing zeros. To apply this algorithm, 3 arrays of size K have to be stored.

Typical output for $\lambda_i = 1$ and $n = 3$ (30 states) can be found in table 4, where $\epsilon_r^{(m)}$ can be found for the series without trailing zeros. For reasons of space we left out the $\epsilon_r^{(m)}$ with $m > 7$.

	$m=0$	1	2	3	4	5	6	7
$r=0$	6.125000	-8.593750	1.455729	0.643993	51.176851	-128.409830	99.173984	-14.747156
1	-0.067941	0.099508	-1.231927	0.019789	-0.005568	0.004394	-0.008778	0.002312
2	-2.621754	0.704660	1.442896	11.740703	-28.031678	23.255298	75.427968	63.409885
3	0.400132	0.122653	0.116897	-0.030711	0.023892	0.010389	-0.080896	-0.003497
4	-2.899217	-172.300284	4.966029	-9.717844	-50.802362	64.473330	76.329890	-84.021912
5	0.116750	0.122538	-0.098813	-0.000448	0.019064	0.003445	-0.009733	0.002105
6	0.448322	0.448332	0.448334	0.448328	0.448333	0.448330	0.448332	
7	1.03×10^5	6.14×10^5	-1.73×10^5	1.78×10^5	-2.56×10^5	4.30×10^5		
8	0.448334	0.448332	0.448331	0.448331	0.448331			
9	-1.14×10^4	-7.77×10^5	2.38×10^6	-6.91×10^6				
10	0.448331	0.448331	0.448331					
11	-2.95×10^7	3.65×10^7						
12	0.448331							

Table 4. Approximations $\epsilon_r^{(m)}$ of p for the Petri net example with $\lambda_i = 1$ and $n = 3$

The table shows some interesting phenomena. As r gets large, for r even, the approximation gets better. Note that as $\epsilon_r^{(m)}$ gets close to $\epsilon_r^{(m+1)}$ for r even, then $\epsilon_{r+1}^{(m)}$ (which is only used as intermediate step, as $r+1$ is odd) gets very large. This does not lead to numerical instabilities (at least not in this case), as can be seen from the table. Even if we take r very large, we find the correct answer: e.g., $\epsilon_{100}^{(0)} = 0.448331$.

Note that, as $\bar{k} = 1$ and the number of states is 30, $\epsilon_{58}^{(0)}$ should give the correct answer (and it does: 0.448331).

In the following table results are given for various values of n. The first column gives n, the second the total number of states N, the third the computed value of p, and the fourth the lowest value of r for which $\epsilon_r^{(0)}$ approximates p with a precision of 5 digits. Note that this value of r is considerably lower than $2(N-1)$, the value for which $\epsilon_r^{(0)} = S_\infty$. This is done again for $\lambda_i = 1$. A good indication that the approximation is close are the values of $\epsilon_r^{(m)}$ for r odd; if they are big, $\epsilon_{r+1}^{(0)}$ is close. To compute $\epsilon_r^{(0)}$ for $n = 100$ and for r up to 500 took $\approx$ 15 minutes on a fast workstation.

n	N	p	r
1	5	.714286	2
2	14	.551546	6
3	30	.448331	8
5	91	.325768	16
10	506	.193286	30
25	6201	.087018	82
50	45526	.045405	198
100	348551	.023207	382

Table 5. Approximations of p for the Petri net example with $\lambda_i = 1$

For $n = 1$ the computation of the b_{kx} can easily be done by hand, and we find (for general firing rates) that $p = \alpha - \alpha^2\rho + \alpha^3\rho^2 - \ldots$, with $\alpha = \lambda_1(\lambda_2 + \lambda_3 + 3\lambda_4)/((\lambda_2 + \lambda_3)\lambda_4)$. If we apply the ϵ-algorithm once, i.e., if we compute $\epsilon_2^{(m)}$, we find that $\epsilon_2^{(m)} = \alpha(1+\alpha\rho)^{-1}$ for all m. Thus indeed, if we take $\lambda_i = \rho = 1$, we get $p = \frac{5}{7} \approx 0.714286$, coinciding with our numerical results.

5 Literature

The psa was introduced in Hooghiemstra et al. [17], where it is applied to a coupled processor model, which is a special case of the m-dimensional birth-death process discussed in section 3.1. (They assume single arrivals and departures, and they have some restrictions on the summed transition rates.) For this model it is shown that the psa can be applied and that p_x is analytic in $\rho = 0$, giving a light traffic result (as ρ was taken equal to the arrival rate).

Next Blanc started to work on the psa, resulting in a series of papers with various co-authors, [2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 18]. The convergence properties being the main hurdle for the application of the algorithm to various models, [5] proposes the use of conformal mappings (based on a private communication with Keane, Hooghiemstra & Van de Ree), and applies it successfully to a few models. More extensive computations on one of these models, the shortest queue model, can be found in [4]. The coupled processor model is studied again in [6].

The ϵ-algorithm is first used in [7], for a type of multi-dimensional queueing model in which the server cyclically visits all queues, serving a geometric number of customers (if available) at each queue. This is called a Bernoulli schedule. There are no switching times, resulting in a single level 0 state. Also the shortest queue model is studied again ([11]), now with the ϵ-algorithm.

In Blanc [10] a multi-dimensional queueing model is studied, where the arrivals and departures (which occur one by one) are governed by an additional state component. The transition rates are allowed to depend on the entire state of the system, and arrivals and departures occur simultaneously with state transitions of the extra component. Many polling models fall into this framework. Denote, like we did for the polling model of section 3.1, the state with (x, y). Blanc takes as levels $l(x, y) = |x|$. Because of the transitions in the AMP, the $b_{k,(x,y)}$ cannot be solved recursively, but for each level there remains a set of equations, as many as there are states in the AMP. By solving this set of equations for each level, a solution is found. (Note the difference with the method proposed for these types of models in section 3.1.) The method proposed in [10] is applied to various polling models in Blanc [8, 9, 10], Blanc & Van der Mei [13, 14, 15], Altman et al. [2] and Altman [1]. A model with a single queue and batch arrivals is solved in a similar way in Van den Hout & Blanc [18]. In [13] it is shown that the psa can also be used to compute derivatives in addition to steady state probabilities. This is used to compute optimal values for certain system parameters.

A recent and fairly complete overview of Blanc's work on the psa is [12].

Bavinck et al. [3] study the coupled processor model with 2 queues, equal arrival rates in both queues, and a server that serves both queues with rate $\mu/2$ if $x > 0$; if one queue is empty it serves the other queue with rate μ. For this model the coefficients on the diagonal, i.e., the numbers b_{kx} with $x_1 = x_2$, are computed explicitly, and error bounds are derived.

In a way closely related to the psa Levine & Finkel [19] study the optimal routing for a queueing model. They write the discounted costs under each policy as a power series of the arrival rate to the system, and derive the first few terms using the optimality equation for the system. This leads to optimal policies under low (and in a similar way, high) traffic.

Acknowledgement. I like to thank Prof. J.W. Cohen for many interesting discussions on this subject.

References

1. E. Altman. Analysing timed-token ring protocols using the power series algorithm. In *Proceedings of the 14th International Teletraffic Conference*, 1994.
2. E. Altman, J.P.C. Blanc, A. Khamisy, and U. Yechiali. Polling systems with walking and switch-in times. To appear in *Stochastic Models*, 10, 1994.
3. H. Bavinck, G. Hooghiemstra, and E. de Waard. An application of Gegenbauer polynomials in queueing theory. *Journal of Computational and Applied Mathematics*, 49:1–10, 1993.
4. J.P.C. Blanc. A note on waiting times in systems with queues in parallel. *Journal of Applied Probability*, 24:540–546, 1987.
5. J.P.C. Blanc. On a numerical method for calculating state probabilities for queueing systems with more than one waiting line. *Journal of Computational and Applied Mathematics*, 20:119–125, 1987.

6. J.P.C. Blanc. A numerical study of a coupled processor model. In G. Iazeolla, P.J. Courtois, and O.J. Boxma, editors, *Computer Performance and Reliability*, pages 289–303. North-Holland, 1988.
7. J.P.C. Blanc. A numerical approach to cyclic-service queueing models. *Queueing Systems*, 6:173–188, 1990.
8. J.P.C. Blanc. The power-series algorithm applied to cyclic polling systems. *Stochastic Models*, 7:527–545, 1991.
9. J.P.C. Blanc. An algorithmic solution of polling models with limited service disciplines. *IEEE Transactions on Communications*, 40:1152–1155, 1992.
10. J.P.C. Blanc. Performance evaluation of polling systems by means of the power-series algorithm. *Annals of Operations Research*, 35:155–186, 1992.
11. J.P.C. Blanc. The power-series algorithm applied to the shortest-queue model. *Operations Research*, 40:157–167, 1992.
12. J.P.C. Blanc. Performance analysis and optimization with the power-series algorithm. In L. Donatiello and R. Nelson, editors, *Performance Evaluation of Computer and Communication Systems*, pages 53–80. Springer-Verlag, 1993. Lecture Notes in Computer Science 729.
13. J.P.C. Blanc and R.D. van der Mei. Optimization of polling systems by means of gradient methods and the power series algorithm. Technical Report FEW 575, Tilburg University, 1992.
14. J.P.C. Blanc and R.D. van der Mei. Optimization of polling systems with Bernoulli schedules. Technical Report FEW 563, Tilburg University, 1992.
15. J.P.C. Blanc and R.D. van der Mei. The power series algorithm applied to polling systems with a dormant server. In J. Labetoulle and J.W. Roberts, editors, *The Fundamental Role of Teletraffic in the Evolution of Telecommunications Networks*, pages 865–874. Elsevier, 1994.
16. O.J. Boxma, G.M. Koole, and Z. Liu. Queueing-theoretic solution methods for models of parallel and distributed systems. In: These proceedings, 1994.
17. G. Hooghiemstra, M. Keane, and S. van de Ree. Power series for stationary distributions of coupled processor models. *SIAM Journal on Applied Mathematics*, 48:1159–1166, 1988.
18. W.B. van den Hout and J.P.C. Blanc. The power-series algorithm extended to the $BMAP|PH|1$ queue. Submitted for publication, 1993.
19. A. Levine and D. Finkel. Load balancing in a multi-server queueing system. *Computers and Operations Research*, 17:17–25, 1990.
20. P. Wynn. On the convergence and stability of the epsilon algorithm. *SIAM Journal on Numerical Analysis*, 3:91–122, 1966.