Workload Minimization in Re-entrant Lines

Ger Koole & Auke Pot
Vrije Universiteit, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands

To appear in *European Journal of Operational Research*, 2005

Abstract

This paper is concerned with workload minimization in re-entrant lines with exponential service times and preemptive control policies. Using a numerical algorithm called the power series algorithm we obtain nearly optimal policies for systems with up to 8 queues. We also improve considerably the implementation of the power series algorithm.
Key words: Dynamic programming, re-entrant lines, power series algorithm, curse of dimensionality.

1 Introduction

Re-entrant lines are queueing networks that are characterized by the fact that jobs visit machines in a fixed order. Jobs can visit machines more than once, introducing the possibility to control the flow of jobs through the system by scheduling jobs that are at different production stages at the same machine. Every production stage is represented by a separate queue. It is well known that the standard stability condition $\rho < 1$ for each machine is not sufficient under certain scheduling rules for machines that are visited more than once (see, e.g., Kumar and Seidman [19] and Bramson [3]). Additional conditions are necessary to ensure stability of priority rules, as in Dumas [9]. In Kumar [17, 18] is shown that under certain deterministic policies the condition $\rho < 1$ is sufficient, namely: first buffer first served (FBFS) and last buffer first served (LBFS). This holds also for a simple policy that selects for each machine the next queue to serve in a random manner. It is intuitively clear that this policy does not minimize the total average expected workload; neither do FBFS and LBFS. Analytically finding optimal controls in re-entrant lines is extremely difficult, only in very simple cases the optimal policy can be derived explicitly: see, e.g., Koole and Righter [16]. Thus we have to rely on numerical methods.

The scheduling policy for which the system is stable is the basis of our search for scheduling policies that minimize the total average expected workload. For re-entrant lines with up to say 4 queues this can be done using standard methods from Markov decision theory. High-dimensional systems cannot be analyzed exactly due to the curse

of dimensionality: the number of possible configurations, or states in Markov decision terminology, grows exponentially in the number of queues. The main contribution of this paper is the introduction of a method to obtain good scheduling policies for re-entrant lines with a moderate number of queues, say up to 8. For this we use the power series algorithm, a numerical tool for solving Markov chains that can also be used for Markov decision chains. The power series algorithm is considerably faster than straightforward recursion. The usual implementation however was not adapted to the high dimensions that we are interested in; we improved it accordingly.

Two-station re-entrant lines are studied in Dai and Vande Vate [6], the fluid limit plays a central role and is the main subject of Dai and Vande Vate [7]. These papers supply stability conditions and stable policies. The results also hold for the original system of which the fluid limit was taken. A more general model with more than two stations is discussed in Dai [5].

2 Bernoulli policies

In this paper we consider the following class of models. There are P types of parts. The arrival rate of type p requests is λ_p, $p = 1, \ldots, P$. Type p requires N_p production stages, the nth production stage requiring an exponentially distributed processing time with rate μ_{pn}. There are M machines, and each machine processes a number of production stages of various parts. Let $\mathcal{A}_m$ denote the set of stages to be processed by machine m, where stages are notated as couples (p, n). Every stage belongs to exactly one machine. When there is only 1 part type then we suppress $p = 1$ from the notation.

As an example, consider the model of Figure 3. Here $P = 2$, $M = 2$, $\mathcal{A}_1 = \{(1,1),(2,1)\}$, and $\mathcal{A}_2 = \{(2,2)\}$.

Parts requiring the same processing stage are scheduled in a FCFS order. When multiple stages share the same machine, then a central controller can decide on the type of job that is to be scheduled next. We assume that this can be done in a pre-emptive manner, making it possible to interrupt jobs if a state change in the system happens that makes a change in scheduled jobs desirable.

The results in this section are quite obvious but we could not find them in the literature. Therefore we supply proves.

In this paper we focus on the construction of scheduling policies that perform close to optimality. An important step is finding some initial policy that is stable. While in Section 1 the FBFS and LBFS policies were mentioned, remind that this section deals with Bernoulli policies.

An obvious condition for stability is that the load on each machine should be smaller than 1. To formalize this, define $\rho_m = \sum_{(p,n)\in\mathcal{A}_m} \lambda_p/\mu_{pn}$. Then a stable policy exists if $\rho_m < 1$ for all m.

Throughout this paper we make the following assumption.

Assumption 2.1 *$\rho_m < 1$, for $m = 1, \ldots, M$.*

We construct a scheduling policy R^β as follows. There are numbers $\beta_{pn} \geq 0$, meaning that the machine m for which $(p,n) \in \mathcal{A}_m$ schedules a job of type (p,n) after each event in the system (i.e., an arrival or a service completion) with probability β_{pn}. Of course we require that $\sum_{(p,n)\in\mathcal{A}_m} \beta_{pn} = 1$. It might occur that a job of type (p,n) is scheduled while there is no such job waiting to be scheduled. In that case the machine idles until the next event.

Lemma 2.2 *The performance of the policy R^β is equal to that of P parallel tandem queueing systems with arrival rates λ_p, $p = 1,\ldots,P$, where stage n in system p has it own dedicated server with service rate $\beta_{pn}\mu_{pn}$, $p = 1,\ldots,P$, $n = 1,\ldots,N_p$.*

Proof We show that the re-entrant line and the tandem queueing systems can be coupled. Assume that at time 0 the states are equal (i.e., the numbers of customers in stage n of tandem queue p is equal to the number of jobs of type (p,n)).

Events at the tandem system are generated as follows. Take a Poisson process with rate $\gamma = \sum_p \lambda_p + \sum_m \sum_{(p,n)\in\mathcal{A}_m} \beta_{pn}\mu_{pn}$. At each point of this Poisson process one (potential) event occurs: with probability λ_p/γ this is an arrival at tandem queue p, and with probability $\beta_{pn}\mu_{pn}/\gamma$ this is a potential departure at queue n of tandem system p. This leads to a correctly specified tandem system, thanks to properties of the Poisson process.

Now consider the re-entrant system. We couple it to the tandem system by making arrivals of type p (departures of type (p,n)) occur at the same time as arrivals at tandem queue p (departures at queue n at tandem p ,respectively). It is easily seen that this specifies the re-entrant line correctly, where indeed each machine is scheduled again after each event. This is where the preemptive nature of the policy R^β is used.

As events happen at the same time we conclude that the paths of both systems are the same. □

Theorem 2.3 *For any re-entrant line satisfying Assumption 2.1 there exists a pre-emptive work-conserving stable policy.*

Proof It suffices to show that there exists a policy for which a stationary distribution exists. First we show that there exist a pre-emptive stable policy, that possibly idles one or more of the machines while there are jobs waiting to be processed by this machine. Because of Lemma 2.2 it suffices to show that the tandem system is stable. This is the case if $\lambda_p/(\beta_{pn}\mu_{pn}) < 1$. The choice $\beta_{pn} = \lambda_p/(\rho_m\mu_{pn})$ will do. Indeed, $\lambda_p/(\beta_{pn}\mu_{pn}) = \rho_m < 1$ by Assumption 2.1, and also $\sum_{(p,n)\in\mathcal{A}_m} \beta_{pn} = 1$.

It remains to show there exists a work-conserving policy R, i.e., a policy that does not idle at a machine as long as there is work. This policy R is constructed from R^β by taking the same actions at the machines where a non-idling action is taken, and by serving an arbitrary part if R^β idles (for example, according to generalized processor sharing). A simple coupling argument shows that R leads to parts leaving the system earlier than under R^β, leading the lower total numbers of parts in the system. □

We conjecture that this random selection rule will also lead to stable policies in the non-preemptive case and in the case of general service times. In this case we have to choose $\beta_{pn} = \lambda_p s_{pn}/\rho_m$ with s_{pn} the expectation of the service time of type (p, n).

Note that the same argument as in the proof of Theorem 2.3 can be used to show that it is never optimal to idle unless there are no jobs at a machine if it is the objective to minimize the total workload.

In the next sections we will minimize the workload in re-entrant systems using the fact that the tandem system is stable for suitably chosen parameters.

3 Optimal control and the power-series algorithm

In this section we present our optimization method. Relevant subjects are Markov decision theory, optimization methods and techniques for performance evaluation. We focus on a special type of algorithm for performance evaluation, namely the power series algorithm.

3.1 Markov decision process

We formulate the model as a Markov decision process. The state space, actions, transitions and the objective function are introduced next.

State space

We define the state space of the model by a vector with the number of jobs at each station for all types, denoted with $\mathcal{X}$. These numbers are positive integers, including zero. This is sufficient if we assume exponentially distributed times in the system. Consequently, the dimension of the state space is determined by $\sum_m |\mathcal{A}_m|$ with $|\mathcal{A}_m|$ the number of different parts that are served by machine m. The ordering of these numbers in the state can be chosen arbitrarily.

Actions

In the model we consider unrandomized policies. A policy dictates the action to be taken in each state of the system. Here, an action determines the part type that each machine works on. We denote the action space with $\mathcal{R}(x)$ for state x and $\mathcal{R}$ is the space for all states

$$\mathcal{R} = \cup_{x \in X} \mathcal{R}(x).$$

We define $R(x) \equiv (R_1(x), ..., R_M(x)) \in \mathcal{R}(x)$ and it should hold that $R_m(x) \in \{1, ..., P\}$ and $(R_m(x), n) \in \mathcal{A}_m$. In words, machine m can logically only serve part type p if parts of that type arrive at that machine in one of the stages. The length of the action vector is the number of machines.

Transitions

Three different type of transitions can be distinguished: job arrivals, part movements between machines and job completions. Consider a transition from state x to $\bar{x}$, $x, \bar{x} \in \mathcal{X}$.

- Arrivals occur in every state of the system. When an arrival of type p occurs, the only difference between the old and new state is $x_{mp} + 1 = \bar{x}_{mp}$, with m determined by $(p, 1) \in \mathcal{A}_m$.
- Part movements between machines occur when there are jobs present at a machine and this machine is not the last production stage of the job. When a part of type p moves from machine m to m', the only differences between the old and new state are $x_{mp} - 1 = \bar{x}_{mp}$ and $x_{m'p} + 1 = \bar{x}_{m'p}$. This transition is possible if $(p, i) \in \mathcal{A}_m$, $(p, i+1) \in \mathcal{A}_{m'}$, $i < N_p$, and the action of machine m, say $R_m(x)$, is p.
- With job completions parts leave the system because their last stage finished. This is the case when $x_{mp} + 1 = \bar{x}_{mp}$, $(p, N_p) \in \mathcal{A}_m$ and $R_m(x) = p$.

Uniformization

We look at an equivalent system in discrete time. We multiply all transition rates by α such that the sum of all transition rates in each state is smaller than one. The outcomes can be considered as transition probabilities in discrete time. This is called uniformization, see Lippman [20]. If we observe the original system at exponentially distributed times with parameter α, it behaves the same as the original one. We choose

$$\alpha = \sum_{p=1}^{P} \left(\lambda_p + \sum_{n=1}^{N_p} \mu_{pn} \right).$$

Objective

The purpose of introducing actions is controlling the system such that the long-run average total number of jobs in the system is minimized. The total number of jobs in state x is xe, with e the unity vector.

Consider the process $\{x_t^R, t \geq 0\}$, with x_t^r the state at time t under policy R. Decision variable $R(x_t^R)$ determines the resource allocation at time t. The average number of jobs in the system under policy R is

$$g^R = \lim_{k \to \infty} \frac{\sum_{t=0}^{k-1} e x_t^R}{k}$$

with t the state at time t. We define an optimal policy R^* by

$$R^* = \arg\min_{R \in \mathcal{R}} g^R.$$

3.2 Optimization

Standard methods to obtain optimal policies in systems with finite state spaces, such as policy iteration and value iteration (see for example Puterman [22]), rely on iterative methods to solve one or more matrix equations. The size of the solutions is equal to the size of the state space, making it quickly infeasible to solve this type of problem, certainly if the state is multi-dimensional: the total number of states is exponential in the dimension of the state space. This is the so-called curse of dimensionality.

The policy iteration method consists of repeatedly executing an optimization and an evaluation step. It is the evaluation step that is numerically demanding; by utilizing the so-called power series algorithm (PSA) instead of the standard recursive method we can handle significantly bigger systems.

Policy evaluation step

In this section we formulate the Poisson equation corresponding with our model and the Markov decision process that we defined. The purpose of this equation is that it allows us to evaluate policies. (Under a fixed policy a Markov decision chain becomes a Markov reward chain.)

Consider the finite state Markov reward chain with state space X, transition probability matrix P and cost vector c. Assume that there is a single recurrent class (by choosing R properly, non-idling, work-conserving policy). We are interested in the solution of the Poisson equation

$$v + ge = c + P^R v. \tag{1}$$

Here the average cost is denoted by g (a real value), v represents the bias or value vector of the states and P^R is the transition matrix under policy R. We denote the element p^R_{xy} of P^R as the transition probability of going from state x to y. The vector g is equal for all states because there is a single recurrent class.

There exist several methods to solve Equation (1), value iteration or any matrix inversion method. In this paper we will use the power series algorithm for this task.

Policy improvement step

Given an initial policy, improvements are achievable by policy iteration. In matrix notation, policy evaluation consists of solving $v^0, R^0, v^1, R^1, ...$ repeatedly using

$$v^{i-1} + g^{i-1}e = c + P(R^{i-1})v^{i-1}; \qquad r^i = \arg\min_{R\in\mathcal{R}}\{c + P(R^{i-1})v^{i-1}\} \tag{2}$$

until R^{i-1} and R^i are equal. After each improvement step the policy evaluation step takes place. What we mentioned about the PSA for solving the value function of a Markov reward chain is relevant here, because in this paper the PSA is used for policy evaluation. In other words, the policy evaluation step of the policy iteration algorithm is performed with the PSA.

In Chen & Meyn [4] the importance of a good initialization was shown for the value iteration algorithm. In the policy iteration algorithm as we use it in conjunction with the PSA we have to choose an initial policy. The Bernoulli policy of Section 2 is a good choice with respect to stability, because the value function of the corresponding fluid limit is quadratic in the queue lengths (see [4]).

For general text on the control of stochastic systems we refer the reader to Puterman [22].

3.3 Power-series algorithm applied to the Poisson equation

The PSA was introduced by Hooghiemstra et al. [11] to compute in a numerically efficient way the equilibrium probabilities of Markov chains, i.e., the solution of $\pi = \pi P$ and $\pi e = 1$ with P the transition matrix of the Markov chain. Its possibilities were explored in a series of papers by Blanc, summarized in [2].

In Katehakis and Levine [13] it was used in the context of control problems, to find the solution to dynamic programming optimality equation for some limiting case. See Koole & Passchier [15] for more details on this method to obtain asymptotically optimal policies.

In this paper we apply the PSA to solve the Poisson equation. We first have to define a level function.

Level function

We introduce a level function $L : \mathcal{X} \to \mathbb{N}_0$. By level k we denote all states $x \in X$ such that $L(x) = k$. We assume that the level function has the following properties:
- Level 0 consists of only one state;
- The states within each level can be ordered such that there are no transitions to higher ordered states within that level;
- Transitions to lower level states are possible in each state unless $L(x) = 0$.

For a recurrent system such a level function can always be chosen (Koole [14]).

The ordenings per type are independent of eachother, since transitions change the state of exactly one type. Therefore, we can consider the state space of each type separately.

For the re-entrant system we define the level as the total number of jobs in the system, specified per part type. We first have to define the ordening of the states within each level and for each type before we can show that this definition satisfies the properties of L.

Theorem 3.1 *For each type of part and level, a lexicographical ordening on the production stages satisfies the properties of L.*

Proof We consider the service completions. It is obvious that the next state is in the same level as the original state, in case that the service was not the last production stage. The next state goes one level lower if the part leaves the system. In addition, level 0 contsists indeed of one state and states to lower level states are possible if $L(x) > 0$. □

What remains to show is a procedure for traversing states within a level in the right order. This is given in Appendix A.

The systems from our numerical analysis in Section 4 satisfy the conditions and definitions from this section.

Power series expansion

We replace the unknown variables v and g of Equation (1) with power series of parameter λ. This parameter must be chosen in such a way that the Markov chain has transition probabilities of the form

$$p_{xy} = \lambda^{(L(y)-L(x))^+} p'_{xy}, \tag{3}$$

where all p'_{xy} are independent of λ. This restricts the choice of the level function. In many queueing systems we take λ equal to the arrival rate, explaining the choice of notation. Then level k corresponds to all states with k customers in the system. In case there is no obvious choice for λ, then it can be chosen to be an artificial parameter which is later replaced by 1. We define the power series of g and v as

$$v(x,\lambda) = \sum_{k=0}^{\infty} v^{(k)}(x)\lambda^k, \tag{4}$$

$$g(\lambda) = \sum_{k=0}^{\infty} g^{(k)}\lambda^k, \tag{5}$$

in which x denotes the state and $v^{(k)}(x)$ and $g^{(k)}$ the coefficients that need to be solved. The Poisson equation (1) is rewritten with v and g replaced by the power series of λ, as defined in Equation (4) and (5).

Recursive solution

The PSA is a recursive process to solve the coefficients of the power series expansion around $\lambda = 0$. If the power series converges, a solution of (1) is found.

First we consider Equation (1) in state x

$$\sum_{x \neq y} p_{xy} v(x) + g = c(x) + \sum_{x \neq y} p_{xy} v(y).$$

We replace v and g by the power series as defined in Equation (4) and (5), yielding

$$\sum_{x \neq y} p_{xy} \sum_{k=0}^{\infty} v^{(k)}(x)\lambda^k + \sum_{k=0}^{\infty} g^{(k)}\lambda^k = c(x)\mathbb{I}\{k=0\} + \sum_{x \neq y} p_{xy} \sum_{k=0}^{\infty} v^{(k)}(y)\lambda^k.$$

We make a distinction between transitions to lower level states and the others

$$\sum_{y:L(x)<L(y)} p_{xy} \sum_{k=0}^{\infty} v^{(k)}(x)\lambda^k + \sum_{y:L(x)\geq L(y)} p_{xy} \sum_{k=0}^{\infty} v^{(k)}(x)\lambda^k + \sum_{k=0}^{\infty} g^{(k)}\lambda^k =$$

$$c(x)\mathbb{I}\{k=0\}+\sum_{y:L(x)<L(y)}p_{xy}\sum_{k=0}^{\infty}v^{(k)}(y)\lambda^k+\sum_{y:L(x)\geq L(y)}p_{xy}\sum_{k=0}^{\infty}v^{(k)}(y)\lambda^k.$$

The above equation can be expressed explicitly in power series of λ

$$\sum_{y:L(x)<L(y)}p'_{xy}\sum_{k=0}^{\infty}v^{(k)}(x)\lambda^{k+1}+\sum_{y:L(x)\geq L(y)}p'_{xy}\sum_{k=0}^{\infty}v^{(k)}(x)\lambda^k+\sum_{k=0}^{\infty}g^{(k)}\lambda^k=$$
$$c(x)\mathbb{I}\{k=0\}+\sum_{y:L(x)<L(y)}p'_{xy}\sum_{k=0}^{\infty}v^{(k)}(y)\lambda^{k+1}+\sum_{y:L(x)\geq L(y)}p'_{xy}\sum_{k=0}^{\infty}v^{(k)}(y)\lambda^k.$$

We consider the equation for a fixed order k

$$\sum_{y:L(x)<L(y)}p'_{xy}v^{(k-1)}(x)+\sum_{y:L(x)\geq L(y)}p'_{xy}v^{(k)}(x)+g^{(k)}= \tag{6}$$
$$c(x)\mathbb{I}\{k=0\}+\sum_{y:L(x)<L(y)}p'_{xy}v^{(k-1)}(y)+\sum_{y:L(x)\geq L(y)}p'_{xy}v^{(k)}(y). \tag{7}$$

From this equation we can calculated the coefficients of each order recursively.

The first step of the algorithm is the initialization

$$v^{(k)}(0)=0, k\geq 0;$$

$$v^{(0)}(x)=0, L(x)\neq 0.$$

Thereafter, the coefficients of order $k=1,2,3,...$ are calculated. The coefficients of order k are determined as follows. All states within level l are traversed according to the ordering that was defined before in this section, iterating on $l=0,1,2$ etc. In this way each coefficient is determined uniquelly, see Theorem 3.2. Finally, the values and the average cost can be determined with (4) and (5).

The state spaces of the Markov chains that are discussed in this paper, are not finite. Before applying the PSA we choose a state level H and a minimum order for the power series K, such that the algorithm determines for all states up to level H the first K coefficients of the power series. Necessarily, the first $K-1$ coefficients of $v(x)$ are calculated for states $x\in L(H+1)$, the first $K-2$ coefficients for states $x\in L(H+2)$,..., and only the first order coefficients for states in level $K+H$.

In the applications that are presented later we take in all cases $H=0$.

Theorem 3.2 *The coefficients $v^{(k)}(x)$ and $g^{(k)}$ have a unique solution.*

Proof This can be done recursively, starting with the initialization that we described above. Consider the description of the algorithm, specifying the order in which the coefficients are calculated. Assume that the coefficients $v^{(i)}(x)$ and $g^{(i)}$ up to order $k-1$ and level $l+1$ are calculated. The coefficients of the states with order k can be determined up

to level l with Equation (6). The terms of order $k-1$ are already known as we assumed. Coefficient $g^{(k)}$ is determined by considering Equation (6) in state 0. Then the transitions to lower level states vanish and only terms of order $k-1$ remain. Coefficients $v^{(k)}$ are solved consequetively according to the ordening; the equation is solved for each state. □

The PSA and re-entrant lines

Some of the initial policies that we consider in Section 4 are Bernoulli policies. These do not fall in the class of deterministic policies, but in the class of randomized policy. We did not include these in our model to keep notation simple. Under Bernoulli policies the difference, in comparison to deterministic policies, is that the transition matrix P changes. Consider the rates from a state that correspond with transitions to lower level states or same level states. Under deterministic policies only one of them is positive, depending on the action, while under Bernoulli policies the rates are multiplied by different scalars and the outcomes could be but are not necessarily positive. These scalars all sum up to one, for each state. As a result, the value function of re-entrant lines can be simplified. It will correspond with the value function of a tandem system, assuming one server per station.

We applied the PSA algorithm to re-entrant systems with a single arrival process as well as to a system with two arrival processes. In the first case we generated v and g as power series of the arrival rate λ and in the second case as power series of the arrival rates $\lambda_i = \alpha_i \lambda$. It is straightforward to see that in both cases, and for all possible policies, for this choice of parameter the corresponding level function satisfies the conditions.

3.4 The ϵ-algorithm

In our study we experienced that in some situations the PSA did not produce satisfying answers; we observed that the power series were not always stable, depending on the choice of the model parameters. Instead, the power series were divergent and alternating.

For that reason we decided to implement the ϵ-algorithm, which converts a divergent series into a convergent series. It was first used in the context of the PSA in Blanc [1] and it works as follows. We denote the variable of the power series by x and define S_m as the partial sums of the power series

$$S_m = \sum_{i=0}^{m} c_s x^s. \tag{8}$$

The algorithm is initialized by setting

$$\epsilon_{-1}^{(m)} = 0 \qquad \text{and} \qquad \epsilon_0^{(m)} = S_m.$$

The quantities $\epsilon_s^{(m)}$ for $s > 0$ are determined by

$$\epsilon_{s+1}^{(m)} = \epsilon_{s-1}^{(m+1)} + (\epsilon_s^{(m+1)} - \epsilon_s^{(m)})^{-1}.$$

Afterwards, the converging series that corresponds with the series from Equation (8) are the quantities $\epsilon_{2r}^{(m)}$, with $r = 0, 1, ...$

The ϵ-algorithm is stable when the power series describes a quotient of two polynomials. Around $x = 0$ the partial sums S_m of the power series may be convergent (if the function is analytic). From the moment that the denominator reaches zero (in the complex plane) the partial sums diverge. The benefit of the ϵ-algorithm is that these poles are cancelled out. As a result the power series remain converging. See Wynn [23] for more details.

We applied the algorithm to the partial sums of Equation (4) and (5). In general these functions are not a quotient of two polynomials (but they do have poles). Therefore it is hard to predict the stability of the ϵ-algorithm when it is applied to power series that are a solution of Equation (1). The effectiveness of the algorithm in our experiments will be discussed in Section 4.

From earlier experiments it appears that the ϵ-algorithm is often remarkable stable, see Wynn [23]. For queuing systems this concerns calculations of the stationary probabilities, see Hooghiemstra and Koole [12]. For finite chains the algorithm is stable, given that m, the size of the input data, is chosen properly (see [14]).

However, in general not much is known about the required number of partial sums, i.e. parameter m, since analytic expressions of the stationary probabilities or value function are not always known. In general there is no guarantee that the PSA, in conjunction with the ϵ-algorithm, will produce correct results. For this reason it is a good habit to check the outcomes of the PSA by filling them in in the equation for which they are supposed to give a solution. In our studies we compared the left-hand and right-hand side of Equation (1) after filling in the approximations for v and g. In case that the difference between both sides was significantly non-equal to zero, we concluded that no solution was found. By increasing m the accuracy of the approximation will improve.

3.5 Implementation

So far we discussed only the mathematical aspects of the PSA. However, implementation details determine to a large extent the performance of the algorithm. In Appendix A we show how the implementation of the algorithm can be considerably improved by choosing an appropriate data structure.

4 Experimental results

In this section we present the results that we obtained for various re-entrant systems. Different initial policies are considered: last buffer first served (LBFS), first buffer first served (FBFS) and a Bernoulli policy of which the parameters will be defined later in this section. In the numerical studies one or more policy improvement and evaluation steps are executed. It is known from the literature (e.g, by Ott and Krishnan [21]) that only one step of policy improvement can already give an enormous improvement and can bring us quite close to the optimal value. This is confirmed by our computations.

4.1 Tandem systems

We start by analyzing a tandem system, with parameters as indicated in Figure 1. We have $P = 1$, M servers, $\mathcal{A}_m = \{(1, m)\}$.

The holding costs are h_j for each customer in queue j, the direct costs c from the Poisson equation are therefore given by $c(x) = h^T x = \sum_n h_n x_{1n}$, where $x = (x_{11}, \ldots, x_{1M})$ represents the numbers of customers in the queues, and h_j are the holding costs of keeping one job one time unit at queue j.

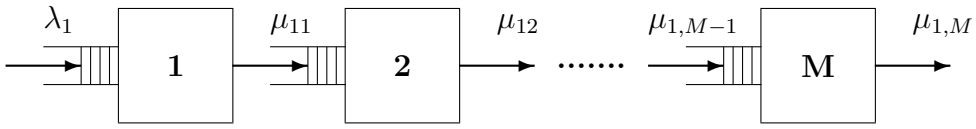

Figure 1: Tandem system

First we derive formulas to calculate the coefficients of each power series. Equation (1) becomes in this application:

$$(\lambda_1 + \sum_{i=1}^{M} \mu_{1i} \mathbb{I}\{x_{1i} > 0\}) v(x) + g =$$

$$h^T x + \lambda_1 v(x + e_1) + \sum_{i=1}^{N} \mu_{1i} v(x - e_i + e_{i+1} \mathbb{I}\{i < M\}) \mathbb{I}\{x_{1i} > 0\}. \tag{9}$$

We replace the variables by power series of λ, defined by Equation (4) and (5):

$$(\lambda_1 + \sum_{i=1}^{M} \mu_{1i} \mathbb{I}\{x_{1i} > 0\}) \sum_{k=0}^{\infty} v^{(k)}(x) \lambda_1^k + \sum_{k=0}^{\infty} g^{(k)} \lambda_1^k =$$

$$h^T x + \lambda_1 \sum_{k=0}^{\infty} v^{(k)}(x + e_1) \lambda_1^k + \sum_{i=1}^{M} \mu_{1i} \sum_{k=0}^{\infty} v^{(k)}(x - e_{1i} + e_{1,i+1} \mathbb{I}\{i < M\}) \mathbb{I}\{x_{1i} > 0\} \lambda_1^k.$$

The coefficients can be determined recursively by

$$\begin{aligned} g^{(k)} &= v^{(k-1)}(e_1) \mathbb{I}\{k > 0\}, \\ v^{(k)}(x) &= \Big[- g^{(k)} + h^T x \mathbb{I}\{k = 0\} - v^{(k-1)}(x) \mathbb{I}\{k > 0\} + v^{(k-1)}(x + e_1) \mathbb{I}\{(k > 0)\} + \\ &\quad \sum_{i=1}^{M} \mu_{1i} v^{(k)}(x - e_{1i} + e_{1,i+1} \mathbb{I}\{i < M\}) \mathbb{I}\{x_{1i} > 0\} \Big] \Big/ \Big[\sum_{i=1}^{M} \mu_{1i} \mathbb{I}\{x_{1i} > 0\} \Big]. \end{aligned}$$

The performance of the algorithm, which is applied to a tandem system with five servers and the tree as main data structure (see Appendix A) can be found in Table 1. The arrival rate and service time parameters are taken fixed: $\lambda_1 = 1$ and $\mu_{1i} = 5/4$ for all i. Each row shows for different orders the memory usage, the computation time and the approximation of the average cost. The accuracy of the approximations are shown since the real average cost is determined by $g = \sum_{i=1}^{5} \lambda_1/(\mu_{1i} - \lambda_1) = 20$, from standard results for queuing methods.

K	Memory (Mb)	Time (seconds)	Approximation of g
20	2	2	19.7694
25	3	4	19.9244
30	5	9	19.9752
35	9	23	19.9919
40	15	51	19.9973
45	24	100	19.9991
50	37	184	19.9997

Table 1: Results for a tandem system with 5 servers

In Table 2 several performance measures are presented for different numbers of servers. The parameters are the same as in Table 1. The order of the power series is 19 (because of hardware limitations).

#Servers	Memory (Mb)	Time (seconds)
5	2	1
6	4	4
7	12	17
8	43	73
9	140	266
10	431	875

Table 2: Results for different tandem systems

Note that no closed-form expression is known for the value function of the tandem system. Only the average cost g is known from queueing theory.

We conclude that all values of g that we found with the PSA are quite accurate.

4.2 Re-entrant system with three waiting queues

The simplest non-trivial re-entrant system is that of Figure 2. Note that the same model has also been analyzed in Chen & Meyn [4], using value iteration. We have $P = 1, M =$

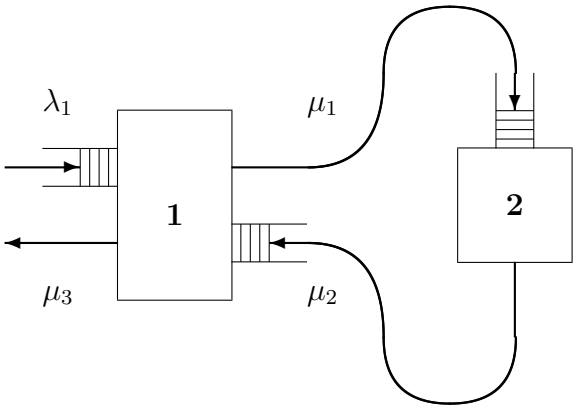

Figure 2: A re-entrant system with three queues

$2, \mathcal{A}_1 = \{(1,1),(1,3)\}$ and $\mathcal{A}_2 = \{(1,2)\}$. It is well-known that this system is stable under any non-idling policy, see Dai & Weiss [8].

Our objective is to minimize the total weighted number of clients in the system. We allow different positive holding costs of the queues. First we study the Bernoulli policy R^0 that minimizes the total weighted number of clients in the system. As we saw in Section 2, the system then becomes a tandem system with three servers. Only the parameter of one Bernoullli distribution has to be determined: the fraction of capacity β_1 of machine 1 that is assigned to stage 1. From the system it follows that $\beta_2 = 1$ and $\beta_3 = 1-\beta_1$. The optimal allocation vector β can simply be determined analytically. Let $N(\beta)$ be the total weighted queue length. Then

$$N(\beta) = \frac{h_1\lambda_1}{\beta_1\mu_{11} - \lambda_1} + \frac{h_2\lambda_1}{\mu_{12} - \lambda_1} + \frac{h_3\lambda_1}{(1-\beta_1)\mu_{13} - \lambda_1}.$$

The minimum of $N(\beta)$ is obtained at

$$\beta_1 = \frac{\sqrt{h_1 a_1} - a_3\sqrt{h_1 a_1} + a_1\sqrt{h_3 a_3}}{\sqrt{h_1 a_1} + \sqrt{h_3 a_3}}, \tag{10}$$

with $a_1 = \lambda_1/\mu_{11}$ and $a_3 = \lambda_1/\mu_{13}$.
The values of $v^0(x)$ and the average cost g^0 can be determined with the PSA. For a fixed β these can be obtained from Equation (9), taking $N = 3$ and service rates $\mu_{1i}\beta_i$.

The one-step improved policy R^1 is found by applying one policy iteration step:

$$R^1 = \arg\min_{R\in\mathcal{R}}\{c + P^{R^0}v^0\}, \tag{11}$$

with $\mathcal{R}$ the set of all non-randomizing policies, such that a policy $R \in \mathcal{R}$ specifies for each state x and machine m which job to serve. If there are no jobs at a machine then we take

action $R_m(x) = 0$, which means idling. Assume in the current three-dimensional model that either $x_1 > 0$ or $x_3 > 0$. Then $R^1_m(x) = 1$ if $x_1 > 0$ and

$$\mu_{11} v^{R^\beta}(x - e_1 + e_2) + (\mu_{13} - \mu_{11}) v^{R^\beta}(x) \leq \mu_{13} v^{R^\beta}((x - e_3)^+),$$

and $R^1_m(x) = 2$ otherwise.

The one-step improvement is evaluated by numerical study, in the sense that we compare the outcomes to the most optimal policies that we could obtain. The holding costs are set to 1, as well as the arrival rate. Table 3 presents the parameters of the different model instances that we consider. The order is the level at which the state space is truncated. Optimal solutions were approximated with backward recursion. Parameter k is the number of iterations until optimality was reached. Taking the starting value equal to the value of the fluid limit, as proposed in Chen & Meyn [4], reduces the number of iterations significantly in comparison to starting with initial value 0.

Instance	(μ_1, μ_2, μ_3)	Order	k
1	(3, 2, 4)	20	690
2	(3, 2, 3)	30	891
3	(4, 2, 4)	20	631
4	(3.5, 2, 2.5)	35	1044
5	(2.5, 2, 2.5)	55	1184

Table 3: Parameters of the re-entrant systems with three queues, $\lambda = 1$

In Table 4 results on one-step improvement are shown for different initial policies: Bernoulli, FBFS and LBFS. The last column presents approximations of average cost under policy R^*. The outcomes of R^1 are evaluated by comparing the outcomes of the left-hand and right-hand side of

$$v + g = c + P(R^1)v, \tag{12}$$

as we already mentioned in Section 3.4. The maximum order of the power series was in all instances taken equal to 40. This number of coefficients appeared to be sufficient to find very accurate approximations under modest workloads. We observed that values of states within higher levels were solved less accurately. To obtain more accurate result it is necessary to calculate coefficients of higher order terms. The table shows that the one-step improvement policy leads to quite a big improvement for the Bernoulli policy. We should also note that LBFS performs initially already performance better than Bernoulli after the first policy iteration (in all instances except 4). The choice of the initial policy has not much influence on the quality after one-step improvement, but also these minor differences from optimal should not be ignored. LBFS is the best initial policy.

Instance	Bernoulli		LBFS		FBFS		
	g^0	g^1	g^0	g^1	g^0	g^1	g^{R^*}
1	3.7856	2.3294	2.3214	2.2970	2.6506	2.3214	2.2944
2	5.0000	2.9025	2.9006	2.8178	3.3129	2.8665	2.8210
3	3.0000	1.9425	1.9929	1.9357	2.0667	1.9532	1.9316
4	5.3331	3.0550	3.1744	3.0465	3.3691	3.0504	2.9950
5	9.0000	4.8351	4.5623	4.4853	6.0720	4.5227	4.3804

Table 4: Re-entrant system with three queues ($\lambda = 1$)

4.3 Criss-cross system

In this section we will apply the PSA to another three-dimensional system, the criss-cross system of Harrison & Wein [10]. The system is depicted in Figure 3. It is well-known that this system is stable under any non-idling policy.

We have $P = 2, M = 2, \mathcal{A}_1 = \{(1,1),(2,1)\}$ and $\mathcal{A}_2 = \{(2,2)\}$.

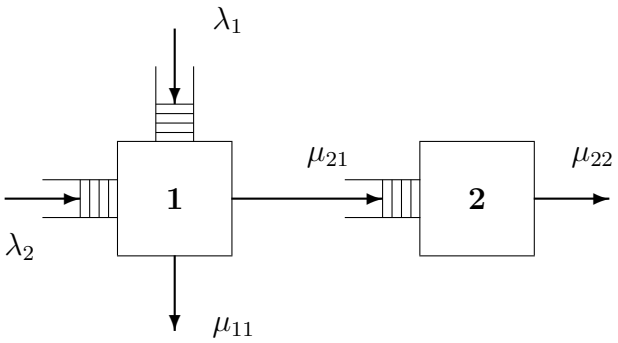

Figure 3: Criss-cross system

Our objective is to minimize the total number of clients in the system, therefore we take for each queue the same holding cost.

We will discuss several policies and start as usual with the Bernoulli policy that assigns a fraction β_1 of the total service capacity of server 1 to stage 1 and the remaining part $\beta_2 = 1 - \beta_1$ to stage 2. Because machine 2 serves only one type of job we take $\beta_3 = 1$. We minimize the total average queue length

$$N(\beta_1) = \frac{\lambda_1}{\beta_1\mu_{11} - \lambda_1} + \frac{\lambda_2}{(1-\beta_1)\mu_{21} - \lambda_2} + \frac{\lambda_2}{\mu_{22} - \lambda_2}. \tag{13}$$

The optimal value for β_1 is

$$\beta_1 = \frac{\sqrt{a_1} + a_1\sqrt{a_2} - \sqrt{a_1}a_2}{\sqrt{a_1} + \sqrt{a_2}}, \tag{14}$$

with $a_1 = \lambda_1/\mu_{11}$ and $a_2 = \lambda_2/\mu_{21}$.

In subsection 4.1 we derived formulas to calculate the coefficients of the power series for tandem systems. This method is not directly applicable to systems with several arrival processes. We solved this problem in the following way. We defined $\lambda_1 = \alpha_1\lambda$ and $\lambda_2 = \alpha_2\lambda$. The Poisson equation for R^0 that was given in (1) is here

$$(\sum_{i=1}^{P}\alpha_i\lambda + \sum_{pn}\mu'_{pn}\mathbb{I}\{x_{pn}>0\})v(x) + g =$$
$$h^Tx + \sum_{i=1}^{P}\alpha_i\lambda v(x+e_{i1}) + \sum_{pn}\mathbb{I}\{x_{pn}>0\}\mu'_{pn}v(x-e_{pn}+e_{p,n+1}\mathbb{I}\{n<N_p\})$$

with $\mu'_{11} = \beta_1\mu_{11}$, $\mu'_{21} = (1-\beta_1)\mu_{21}$ and $\mu'_{22} = \mu_{22}$. Next, by inserting the usual power series we get

$$\Big(\sum_{i=1}^{P}\alpha_i\lambda + \sum_{pn}\mu'_{pn}\mathbb{I}\{x_{pn}>0\}\Big)\sum_{k=0}^{\infty}v^{(k)}(x)\lambda^k + \sum_{k=0}^{\infty}g^{(k)}\lambda^k = h^Tx+$$
$$\sum_{i=1}^{P}\alpha_i\lambda\Big(\sum_{k=0}^{\infty}v^{(k)}(x+e_i)\lambda^k\Big) + \sum_{pn}\mu'_{pn}\mathbb{I}\{x_{pn}>0\}\Big(\sum_{k=0}^{\infty}v^{(k)}(x-e_{pn}+e_{p,n+1}\mathbb{I}\{n<N_p\})\lambda^k\Big).$$

Solving the $v^{(k)}$s and $g^{(k)}$s gives

$$g^{(k)} = \sum_{i=1}^{P}\alpha_i v^{(k-1)}(e_{p1}),$$

$$v^{(k)}(x) = \Big(-g^{(k)} - \sum_{i=1}^{P}\alpha_i v^{(k-1)}(x) + h^Tx\mathbb{I}\{k=0\} +$$
$$\sum_{pn}\mu'_{pn}v^{(k)}(x-e_{pn}+e_{p,n+1}\mathbb{I}\{n<N_p\}) + \sum_{i=1}^{P}\alpha_i v^{(k-1)}(x+e_{i1})\Big)\Big/\Big(\sum_{pn}\mu'_{pn}\mathbb{I}\{x_{pn}>0\}\Big).$$

Table 5 contains the average costs for the initial policy and after one and k steps of policy improvement. The right column also contains the number k, i.e., the maximum number of policy improvements for which we obtained accurate outcomes. The one-step improved policy is again surprisingly good.

5 Folded systems with 4 and 8 waiting queues

In this section we analyze re-entrant systems of the type that is drawn in Figure 4. Consider first the one that consists of four waiting queues, studied also in Kumar & Seidman [19] and

$(\mu_{11}, \mu_{21}, \mu_{22})$	g^0 (exact)	g^1 (PSA)	g^k (PSA)
(3, 4, 2)	3.786	2.351	2.109 (k=7)
(3, 3, 2)	5.000	2.930	2.844 (4)
(3, 2, 4)	10.232	5.373	4.868 (2)
(2.5, 3, 3)	5.989	3.221	3.083 (4)
(3, 2.5, 3)	5.989	3.229	3.155 (4)

Table 5: Criss-Cross system for $\lambda_1 = \lambda_2 = 1$ and $K = 70$

Dai & Weiss [8]. The formulas are equivalent to those in the previous sections. We note that the actions are choosen at several places in the system, namely at both servers, instead of one server as in the systems from subsections 4.3 and 4.2. This caused no problems but the ϵ-algorithm was needed again to find a solution of the optimality equation under the policies that resulted from the policy iterations. We calculated the one-step improved policy again using equation (11). Results for the case $N = 4$ can be found in Table 6.

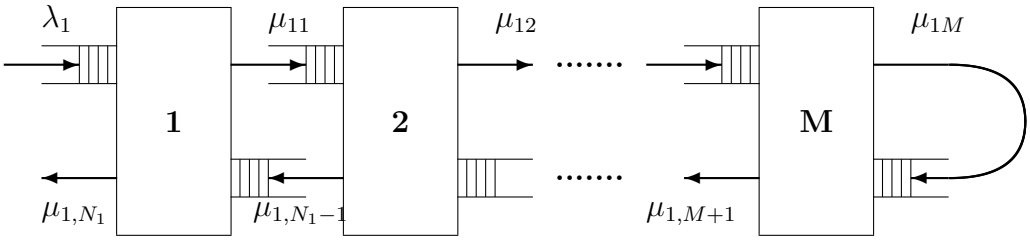

Figure 4: Folded re-entrant system

$(\mu_{11}, \mu_{12}, \mu_{13}, \mu_{14})$	g^0	g^1	g^k
(3, 3, 3, 3)	8.000	3.661	3.323 (k=2)
(4, 3, 4, 2)	8.711	4.060	3.816 (2)
(4, 5, 2, 2)	10.455	4.780	4.508 (2)
(5, 2.5, 4, 3)	7.012	3.096	2.505 (4)
(4, 4, 2.5, 2.5)	7.328	3.359	3.119 (2)

Table 6: Results for a two-machine folded re-entrant line with $\lambda = 1$ and $K = 60$

We just saw that re-entrant lines with a state dimension of 4 can be handled easily. Now we will try the same for a system with 8 queues or stages. The calculation of the values and average cost under the initial policy were easily obtained. Next we determined the one-step improved policy and solved equation (12) with the PSA again. This yielded

the average cost and values that corresponds to that policy. In the second iteration we encountered problems, because the PSA did not converge to a solution of (12). This is probably due to the low number of terms in the power series, higher-order terms probably contribute significantly to the answer. This low number of terms is a consequence of the big state space, the number of states in a level increases exponentially in the dimension. In summary, it was possible to execute the second policy iteration step, but not to determine the corresponding values and average costs.

Numerical results are shown in Table 7. It presents for different service time parameters the average cost under the initial policy and after one-step of policy improvement. The columns have the same meaning as these from Table 4. A distinction is made between the three initial policies that are considered in this paper. The LBFS also performs well for this system.

$(\mu_{11}, ..., \mu_{18})$	Bernoulli		LBFS		FBFS	
	g^0	g^1	g^0	g^1	g^0	g^1
$(10, \ldots, 10)$	2.000	0.965	0.9747	0.9644	1.0326	0.9653
$(6, \ldots, 6)$	4.000	1.873	1.9209	1.8627	2.1217	1.8653
$(5, \ldots, 5)$	5.333	2.456	2.5446	2.4356	2.8722	2.4391
$(4, \ldots, 4)$	8.000	3.607	3.7847	3.53(4)–	4.4167	3.534

Table 7: Four-machine folded re-entrant line with $\lambda = 1$ and $K = 20$

6 Conclusions

We have studied several systems in the previous sections. We conclude that the one-step improvement yields very good policies for re-entrant lines. In comparison with the static allocation policy it reduces the system load with almost 50 percent. Under moderate workload the greedy policy LBFS yields near-optimal performance.

Systems containing up to 4 queues can be analyzed very extensively with the current hardware because several policy improvement steps are possible and the outcomes are computed fast, in less than a quarter of an hour. Since we can determine the values of systems that contain up to 8 queues very accurately, also for these systems the one-step improvement policies are easily obtained.

We conclude that it is hard to determine the solution of the Poisson equation under the dynamic policies, which is firstly required to evaluate the system under one step of policy improvement and secondly if a second step of policy iteration is desired. But we can expect that the quality is comparable to that of smaller systems.

In this paper we showed that the PSA is useful for solving the Poisson equations of high-dimensional systems. Especially when it desired to evaluate systems under different arrival rates the PSA is superior to other techniques, because the coefficient of the power series only need to be solved once.

Acknowledgments We thank Floske Spieksma and Sandjai Bhulai for their remarks that considerably improved the paper.

A Implementation and resource usage of the PSA

An important aspect of the implementation is the data structure in which the values and coefficients of the power series are stored. We have analyzed two types of structures that we expected to be appropriate. The first type of structure is discussed in Blanc [1] where a method is proposed to store the coefficients of the power series in an array. A map function is derived to determine the index in the array where each value is stored. In case the number of states that we want to solve is limited at level H and we are only interested in the coefficients up to order K (at level zero), then the number of coefficients is equal to

$$\binom{K+S+1}{S+1} \tag{15}$$

The index of the k-th coefficient of the power series that approximates the value of state s was defined as

$$i = \sum_{j=0}^{S} \binom{k+j+\sum_{l=1}^{j} s_l}{j+1}. \tag{16}$$

In our study we were especially interested in the solution of the Poisson equation and not really in each coefficient of the power series. A big reduction of storage requirement can be obtained without this condition. We used two main data structures, one to store the most recent calculated coefficient of all states and another to store the values. After all coefficients of a certain order had been calculated, the values were incremented. In this way the required number of memory places to store all coefficients is

$$\#\{s : \sum_{i=1}^{S} s_i \leq K\} = \binom{K+S}{S} \tag{17}$$

The index of the coefficients that are related to state s can then be determined by

$$\sum_{j=0}^{S-1} \binom{j+\sum_{i=1}^{j+1} s_i}{j+1}. \tag{18}$$

An alternative data structure is a tree that has a depth equal to the dimension of the state space and the data stored in the leaves. If the number of states is bounded at level H, the number of different paths is determined by equation (17). A disadvantage of the tree structure is the high number of pointers involved, namely

$$\sum_{i=1}^{S-1} \binom{H+i}{i}. \tag{19}$$

If it is desired that the data structure is removed from memory at run time, an extra integer is required in each node that stores the branch width.

However, the access time of this structure is much faster than an array, because the number of calculations that is required to access the data is only the dimension of the state space. A problem that we encountered while using an array was the calculation of high factorial numbers. The number 20 is the highest factional number that can be processed on a 64-bits processor and this was not sufficient in our study. Although the memory usage is an important aspect and can sometimes form a bottle-neck, we prefer the tree structure above an array because in this way more complex systems can be analyzed and more accurate results can be found. The C++ code of the tree structure can be found at `www.math.vu.nl/~sapot/pub/re-entrant/src` .

One of our aims was to do research on the resource usage, speed and accuracy of the PSA. We wrote a program in C++ to solve the Poisson equation for tandem systems with an arbitrary number of servers, denoted by N. Because the dimension of the state space was not fixed, it was a big challenge to program the code to traverse the states. The pseudo-code to achieve this (in a recursive way) is given below.

```
Integer DIMENSION;
State state(DIMENSION);
Integer MAX_LEVEL;
FunctionPtr function_to_apply;

// apply in each state function_to_apply
forAll(MAX_LEVEL, function_to_apply);

forAll(Integer max_level, FunctionPtr function){
  for(Integer level = 0; level <= max_level; ++level)
    forAll(0, level, function);
}

forAll(Integer server, Integer level, FunctionPtr function){
  Integer surplus = 0;
  for(Integer index = 0; index < server; ++index)
    surplus += state[index];
  if(server == DIMENSION - 1){
    state[server] = level - surplus;
    function();
  }
  else{
    for(Integer index = 0; index <= level - surplus; ++index){
      state[server] = index;
      forAll(server + 1, level, function);
    }
```

```
  }
}
```

Acknowledgment We would like to thank the referee for the suggestions that greatly improved the paper.

References

[1] J.P.C. Blanc. A numerical approach to cyclic-service queueing models. *Queueing Systems*, 6:173–188, 1990.

[2] 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.

[3] M. Bramson. Instability of FIFO queueing networks. *Annals of Applied Probability*, 6:414–431, 1996.

[4] R.-R. Chen and S. Meyn. Value iteration and optimization of multiclass queueing networks. *Queueing Systems*, 32:65–97, 1999.

[5] J.G. Dai. On positive harris recurrence of multiclass queueing networks: A unified approach via fluid limit models. *Annals of Applied Probability*, 5:49–77, 1995.

[6] J.G. Dai and H. Vande Vate. Global stability of two-station queueing networks. In *Proceedings of Workshop on Stochastic Networks: Stability and Rare Events*, pages 1–26, 1996.

[7] J.G. Dai and J.H. Vande Vate. The stability of two-station multi-type fluid network. *Operations Research*, 48:721–744, 2000.

[8] J.G. Dai and G. Weiss. Stability and instability of fluid models for reentrant lines. *Mathematics of Operations Research*, 21:115–134, 1996.

[9] V. Dumas. A multiclass network with non-linear, non-convex, non-monotonic stability conditions. *Queueing Systems*, 25:1–43, 1997.

[10] J.M. Harrison and L.M. Wein. Scheduling networks of queues: heavy traffic analysis of a simple open network. *Queueing Systems*, 5:265–279, 1989.

[11] 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.

[12] G. Hooghiemstra and G.M. Koole. On the convergence of the power series algorithm. *Performance Evaluation*, 42:21–39, 2000.

[13] M.N. Katehakis and A. Levine. A dynamic routing problem—numerical procedures for light traffic conditions. *Applied Mathematics and Computation*, 17:267–276, 1985.

[14] G.M. Koole. On the use of the power series algorithm for general Markov processes, with an application to a Petri net. *INFORMS Journal on Computing*, 9:51–56, 1997.

[15] G.M. Koole and O. Passchier. Optimal control in light traffic Markov decision processes. *Mathematical Methods of Operations Research*, 45:63–79, 1997.

[16] G.M. Koole and R. Righter. Optimal control of tandem reentrant queues. *Queueing Systems*, 28:337–347, 1998.

[17] P.R. Kumar. Re-entrant lines. *Queueing Systems: Theory and Applications*, 13:87–110, 1993. Special Issue on Queueing Networks.

[18] P.R. Kumar. Scheduling queueing networks: Stability, performance, analysis and design. *IMA Volumes in Mathematics and its Applications, Frank P. Kelly and Ruth Williams, Editors*, 71:21–70, 1995. Springer-Verlag, New York.

[19] P.R. Kumar and T.I. Seidman. Dynamic instabilities and stabilization methods in distributed real-time scheduling of manufacturing systems. *IEEE Transactions on Automatic Control*, 35:289–298, 1990.

[20] S.A. Lippman. Applying a new device in the optimization of exponential queueing systems. *Operations Research*, 23:687–710, 1975.

[21] T.J. Ott and K.R. Krishnan. Separable routing: A scheme for state-dependent routing of circuit switched telephone traffic. *Annals of Operations Research*, 35:43–68, 1992.

[22] M.L. Puterman. *Markov Decision Processes*. Wiley, 1994.

[23] P. Wynn. On the convergence and stability of the epsilon algorithm. *SIAM Journal on Numerical Analysis*, 3:91–122, 1966.