Assigning multiple job types to parallel specialized servers

Dinard van der Laan¹

Received: 2 August 2016 / Accepted: 20 April 2018 / Published online: 17 May 2018
© The Author(s) 2018

Abstract In this paper methods of mixing decision rules are investigated and applied to the so-called multiple job type assignment problem with specialized servers. This problem is modeled as continuous time Markov decision process. For this assignment problem performance optimization is in general considered to be difficult. Moreover, for optimal dynamic Markov decision policies the corresponding decision rules have in general a complicated structure not facilitating a smooth implementation. On the other hand optimization over the subclass of so-called static policies is known to be tractable. In the current paper a suitable static decision rule is mixed with dynamic decision rules which are selected such that these rules are relatively easy to describe and implement. Some mixing methods are discussed and optimization is performed over corresponding classes of so-called mixing policies. These mixing policies maintain the property that they are easy to describe and implement compared to overall optimal dynamic Markov decision policies. Besides for all investigated instances the optimized mixing policies perform substantially better than optimal static policies.

Keywords Assignment · Specialized servers · Markov decision process · Mixing decision rules · Implementation

1 Introduction

The control of stochastic systems modeling applications from telecommunication like call centers is commonly modeled as a Markov decision process (MDP). However, for such real-life applications the resulting stochastic system usually has a huge multi-dimensional

✉ Dinard van der Laan
d.a.vander.laan@vu.nl

¹ Department of Econometrics and Operations Research, Vrije Universiteit Amsterdam, De Boelelaan 1105, 1081 HV, Amsterdam, The Netherlands

state space which makes control optimization a problem which is difficult in general. It is usually untractable to obtain an optimal policy by well-known MDP optimization methods as policy iteration and value iteration. Moreover, even if an optimal control policy could be calculated it is in general practically impossible to represent and implement such policies in case of a huge (in general infinite and multi-dimensional) state space. For such huge complex stochastic systems a control policy should have some specific structural properties to make the policy implementable in practice. Then having such structural properties the policy can usually also be described in a comprehensive way. Therefore both static policies and policies generated by dynamic rules according to straightforward heuristics have been investigated and applied (see for example Becker et al. 2000 and Anselmi and Casale 2013). Such policies are suboptimal in general, but by optimization over a class of such policies the expectance is to obtain an implementable policy which performs reasonably well.

In the current paper it is investigated whether by so-called mixing of Markovian decision rules with varying characteristics (for example a static decision rule may be mixed with a dynamic heuristic rule), it is possible to obtain policies which perform better and are as easy to implement than commonly applied policies. For example we compare the performance of these mixing policies with static policies which are optimized by the method given in Becker et al. (2000). For discrete-time MDP (DTMDP) with finite state space the mixing of decision rules is investigated in van der Laan (2011). In that paper structural results on optimization by mixing methods are derived and the concept of mixing decision rules was illustrated by examples in which the state space is finite. Since the mixing of decision rules generates policies which are in general non-stationary it is not obvious under which conditions the performance of such policies is independent of the initial state. These theoretical issues have been investigated in van der Laan (2011). In the current paper the implementation and practical use of two so-called mixing methods are investigated for a problem which is modeled as a continuous time Markov decision process (CTMDP) with a huge multi-dimensional state space consisting of multiple infinite components. Therefore the concepts connected to mixing decision rules will be generalized to be applied to CTMDP with infinite countable state space. For large complex state spaces the implementation of policies generated by the applied mixing methods is an important issue which is considered in detail. Subsequently policies generated by mixing methods will be implemented in simulations. The simulated performances will be compared with each other and with the performances of static policies. It is convenient that the performances of static policies can also be obtained by exact methods. The results are discussed and conclusions are drawn.

In this paper we will apply mixing methods to improve on the performance of various established policies. Improving the performance by mixing policies is applied to an assignment problem which is described and investigated in Becker et al. (2000). This assignment problem is usually referred to as “the multiple job type assignment problem with specialized servers” which in this paper will be abbreviated as MJTAPSS. It is a problem of assigning arriving (according to a given stochastic process) jobs of different types at the moment of arrival irrevocably to one of a group of parallel servers where each of the parallel servers has its own infinite First Come First Served (FCFS) queue to buffer jobs awaiting service. Moreover, service time distributions are allowed to depend both on the type of job and the server to which the job is assigned. The performance of an assignment policy is the (weighed) long-run average sojourn time of the arriving jobs where jobs of different types can have different weight factors. Details of the problem including a description of all system parameters can be found in Section 2. The described problem is a rather general assignment problem with many applications. An obvious application is a call center as described in Becker et al. (2000). In the call center incoming calls about a specific issue

are ideally assigned to an expert on that issue who can answer the call in the most efficient manner. However, in case that at the moment of arrival all the experts are rather occupied it could be beneficial to assign the call to one of the non-experts (a generalist or expert on another issue) who is not so occupied at that moment. Such a call center can indeed be modeled as an instance of MJTAPSS since service time distributions may depend both on job type and server. In such applications the parameters are usually such that it is for each job type quite obvious which server is most efficient for that particular job type. Such straightforward relation between job types and most efficient server(s) for that type will usually be the case in considered instances of MJTAPSS. However, the parameters can also be such that it is not so obvious which server should ideally handle a particular type of job regarding the efficiency following from the system parameters. Anyway for MJTAPSS for an arriving job the optimal assignment will in general depend not only on general system parameters but also on current occupancy of the servers and these criteria may easily conflict with each other.

MJTAPSS is a difficult optimization problem and no heuristic is known to obtain an optimal assignment in all situations. Moreover, simplified variants of MJTAPSS (like for example the problem with only one type of job but there are parallel servers with different service rates) are already known to be difficult. In Borst (1995) a problem resembling MJTAPSS is considered and optimization is performed over so-called static probabilistic policies. In Sethumaran and Squillante (1999) a similar assignment problem is optimized for static policies when in each queue instead of FCFS scheduling an optimal priority over the different job classes is allowed for the scheduling. In Becker et al. (2000) the optimization of static policies is investigated both for FCFS scheduling and optimal priority scheduling for the MJTAPSS problem with the same system specifications as in the current paper. In Altman et al. (2011) the focus is on the performance of static policies for other service disciplines as in particular processor sharing.

If dynamic policies are allowed for optimization only partial optimality results have been obtained as for example in Hordijk and Koole (1992). Motivated by call center applications asymptotic optimality results for dynamic policies have been obtained especially by focusing on heavy traffic limits. For example in Glazebrook and Nino-Mora (2001) the assignment of multiple classes of customers to parallel identical servers is investigated for heavy traffic and the assignment of customers to heterogeneous servers in the Halfin-Whitt heavy traffic regime is considered in Armony (2005) and Armony and Ward (2010). Moreover, in Chen and Ye (2012) asymptotic optimality results for assignment to parallel servers are obtained for the diffusion limit.

A problem resembling MJTAPSS is investigated in Stolyar (2005) which concentrates on the so-called output-queued model satisfying the immediate routing condition. The difference with MJTAPSS is that in these output-queued models the jobs waiting in a queue do not have to be served according to FCFS discipline. Instead for each server, the order in which the different types of jobs waiting in the queue of that server are served, may be chosen by the controller. In Stolyar (2005) for this output-queued model a so-called MinDrift routing rule is introduced which is shown to be asymptotically optimal in the heavy traffic regime. Moreover, so-called MaxWeight policies are introduced to schedule in output-queued models the order in which the different types of waiting jobs are served. In Al-Azzoni and Down (2008) some issues with this MaxWeight scheduling are identified when backing off from heavy traffic. In Stolyar and Teczan (2010) a so-called shadow routing approach is introduced which addresses most issues with MaxWeight policies and which is even applicable if the system input parameters are unknown.

Another variant of the problem occurs if the exact size of an arriving job is known before the assignment which gives additional information compared to plain knowledge of service time distributions as is the case in the current paper. However, also if exact job sizes are known before the assignment only partial optimality results have been obtained as for example in Feng et al. (2005) and Hyttiä et al. (2012).

Recapitulating for a variety of variants of problems with multiple job classes and/or parallel heterogeneous servers partial and asymptotic optimality results have been obtained, but an overall optimal dynamic assignment rule is not known for such problems. Obtaining dynamic optimal assignment policies is known to be difficult in case of heterogeneous servers and multiple job classes make the problem even more complicated.

This paper is organised as follows. In Section 2 the system is specified including all system parameters. The performance objective is defined and the optimization problem is modeled as CTMDP. Next in Section 3 a variety of Markovian decision rules and corresponding assignment policies for MJTAPSS are introduced. First the so-called static policies as considered in Becker et al. (2000) are introduced including the most important issues with respect to implementation and optimization. Next also some dynamic policies based on heuristic decision rules are introduced with focus on the implementation of these policies. These heuristic rules are the myopic so-called selfish rule and the more sophisticated so-called virtual cost rule. In Section 4 the CTMDP modeling MJTAPSS is transformed to a DTMDP. Subsequently a method of mixing decision rules is applied to infinite state space DTMDP modeling MJTAPSS. For this some concepts from van der Laan (2011) on mixing of decision rules for DTMDP with finite state space are generalized to be applied to the the infinite state space DTMDP modeling MJTAPSS. In particular a randomized mixing method and a deterministic mixing method are explained and compared. The static and dynamic assignment rules which were introduced in Section 3 are mixed according to these methods to obtain new mixing policies. Optimization over subclasses of these mixing policies is performed to improve the performance. For the considered subclasses of mixing policies important issues are the tractability of optimization and the practical implementation policies within such a subclass. In Section 5 for instances of MJTAPSS with various levels of traffic load the mixing methods are implemented and the performance of the corresponding policies is approximated by simulation. The obtained performances for the different methods are compared and the mixing parameter is optimized. Finally in Section 6 conclusions are drawn from the obtained numerical results.

2 The model and performance objective

We consider a queueing system where different types $i = 1, 2, \dots, M$ of jobs arrive to be served. Each arriving job has to be assigned at the moment of its arrival irrevocably to one of N parallel queues $j = 1, 2, \dots, N$. Each queue j has its own server j which serves jobs under the FCFS queueing discipline. Moreover the service time distributions S^{ij} may depend both on the job type i and the queue j to which the job is assigned. The service time of a type i job assigned to server j is assumed to be exponentially distributed with parameter μ_{ij} . The objective is to assign arriving jobs to servers such that the weighted average (for some given weight factors) of the long-run average sojourn times of the different job types is minimized. We assume that for every type i the arrival process is a Poisson process and that the M Poisson processes are independent of each other and also independent of service times, etcetera. We denote with λ_i the arrival rate of the type i Poisson process and then $\lambda := \sum_{i=1}^M \lambda_i$ is the total arrival rate of the Poisson process induced by all arrivals of jobs.

We introduce some more notation and definitions. For $i = 1, 2, \dots, M$ and $n = 1, 2, \dots$ let W_n^i and S_n^i be respectively the waiting time and service time of the n -th arriving type i job. Then $V_n^i := W_n^i + S_n^i$ is the sojourn time of the n -th arriving type i job, i.e. it is the total time elapsed between the arrival in the system and departure from the system for the n -th arriving type i job. Similarly for $n = 1, 2, \dots$ let W_n and S_n be respectively the waiting time and service time of the overall n -th arriving job and $V_n := W_n + S_n$ be the sojourn time of the overall n -th arriving job. Besides for $i = 1, 2, \dots, M$ and $t \geq 0$ let $L^i(t)$ be the number of type i jobs present (in service or in one of the waiting queues) in the system at time t . Moreover, for $t \geq 0$ let $L(t) := \sum_{i=1}^M L^i(t)$ be the total number of jobs present in the system at time t .

Let the policy which is applied to assign arriving jobs to queues be denoted by ψ . Then for $i = 1, 2, \dots, M$ we say that $V^i = V^i(\psi)$ is the almost sure long-run average sojourn time of type i jobs if jobs are assigned to servers according to policy ψ if $\lim_{t \rightarrow \infty} \frac{1}{t} \sum_{n=1}^t V_n^i = V^i$ with probability one. Thus if for given policy ψ and $i \in \{1, 2, \dots, M\}$ the Cesàro mean $\lim_{t \rightarrow \infty} \frac{1}{t} \sum_{n=1}^t V_n^i$ of the sojourn times of type i jobs almost surely exists then V^i exists and any sample path realization of the Cesàro mean of sojourn times of type i jobs is with probability one equal to V^i . Analogously we say that $V = V(\psi)$ is the almost sure long-run average sojourn time for all arriving jobs if these jobs are assigned to servers according to policy ψ if $\lim_{t \rightarrow \infty} \frac{1}{t} \sum_{n=1}^t V_n = V$ with probability one. Thus if for given policy ψ the Cesàro mean $\lim_{t \rightarrow \infty} \frac{1}{t} \sum_{n=1}^t V_n$ of all the sojourn times of arriving jobs almost surely exists then V exists and any sample path realization of the Cesàro mean of the sojourn times of all jobs equals V with probability one. The following lemma follows in a straightforward manner from these definitions.

Lemma 1 *If for some policy ψ it holds for all $i \in \{1, 2, \dots, M\}$ that $V^i = V^i(\psi)$, the (almost sure) Cesàro mean of the sojourn times of type i jobs, exists then $V = V(\psi)$, the almost sure Cesàro mean of the sojourn times of all arriving jobs, exists. In that case we have that*

$$V = V(\psi) = \sum_{i=1}^M \frac{\lambda_i}{\lambda} V^i(\psi). \quad (1)$$

Similarly for $i = 1, 2, \dots, M$ we say that $L^i = L^i(\psi)$ is the almost sure long-run average number of type i jobs present in the system if jobs are assigned to servers according to policy ψ if $\lim_{t \rightarrow \infty} \frac{1}{t} \int_{s=0}^t L^i(s) ds = L^i$ with probability one. Also we say that $L = L(\psi)$ is the almost sure long-run average total number of jobs present in the system if these jobs are assigned to servers according to policy ψ if $\lim_{t \rightarrow \infty} \frac{1}{t} \int_{s=0}^t L(s) ds = L$ with probability one.

By Little's Law (see Little 2011) it holds for $i = 1, 2, \dots, M$ that $L^i(\psi)$ exists if and only if $V^i(\psi)$ exists. Moreover, in that case it holds that

$$L^i = \lambda_i V^i. \quad (2)$$

Moreover, if L^i exists for all $i \in \{1, 2, \dots, M\}$ it follows that L and V exists and it follows that

$$L = L(\psi) = \sum_{i=1}^M L^i(\psi) = \lambda V(\psi) = \sum_{i=1}^M \lambda_i V^i(\psi). \quad (3)$$

Performance objective To compare the performance of policies we will use L (or equivalently V) as objective. For the subclass of so-called static policies which is defined in

Section 3.1 it is possible to calculate L exactly. Within this subclass L and thus also V can be minimized by solving some mathematical programming problem. We will also consider policies for which the values of L and V can not be computed exactly. In Section 5 the long-run average sojourn times V will be compared by simulation.

2.1 MDP modeling

For a performance objective defined by L or V (which are equivalent criteria by (3)) the multiple job type assignment problem with specialized servers (MJTAPSS) can be modeled as continuous time Markov decision process (CTMDP). In the sequel we need a description of an appropriate state space S for the MDP formulation. For this a state $s \in S$ should describe the situation for each queue $j \in \{1, 2, \dots, N\}$. A decision epoch occurs at the arrival of a new job or at the completion of a job previously assigned to one of the servers. Then the state s at a decision epoch is an element of $S = \{(y_1, y_2, \dots, y_N, z) \in K^N \times \{0, 1, \dots, M\}\}$, where K is the set of finite sequences taking values in $\{1, 2, \dots, M\}$. For all j , y_j stands for the list of jobs previously assigned to server j for which the service has not yet completed. The y_j are represented as lists because of the assumption of FCFS queueing discipline per queue. The variable z represents the newly arriving job which has to be assigned at the decision epoch. In case of a service completion instead of arriving job, we use the convention that $z = 0$. We note that S is an infinite countable state space since K is infinite countable.

Next we describe the action space for the MDP formulation. For $s = (y_1, y_2, \dots, y_N, z) \in S$ with $z \geq 1$ the action space is given by $A(s) = \{1, 2, \dots, N\}$. In case $z = 0$ there is no actual choice of action (a so-called virtual decision epoch) which is modeled by $A(s) = \{0\}$.

The CTMDP modeling can be completed by specifying all expected transition times, expected direct costs and transition probabilities. We omit this since these elements of the CTMDP model are not necessary to describe and apply the mixing methods which are investigated in this paper.

For CTMDP modeling MJTAPSS it is in general difficult to obtain the optimal performance and/or an optimal assignment policy. We have seen that the state space S needed to describe the problem is huge consisting of multiple components of infinite size. This makes it untractable to apply standard MDP methods like policy iteration or value iteration to obtain optimal policies. In the next section we will first discuss some common methods and heuristics by which easily implementable (but in general suboptimal) policies can be obtained. After that we introduce some other sophisticated heuristic which gives good performances and is also easy to implement. Next methods are introduced to improve the performance even more while the obtained policies remain easy to implement. Examples will be given for which these methods are applicable and it is shown how the obtained policies can be implemented. In Section 5 simulations will confirm that by applying these methods the performance can be substantially improved compared to static policies and commonly applied dynamic policies for MJTAPSS.

3 Implementation and optimization of Markovian policies

We have seen that the problem of minimizing the performance $V(\pi)$ over assignment policies π can be modeled as a CTMDP. We will optimize the performance within the class of Markovian policies π which can be represented as an infinite sequence $\pi = (d_1, d_2, \dots)$

where $d_n, n = 1, 2, \dots$ is a Markovian decision rule to be applied at the n -th **actual** decision epoch. For the considered assignment problem the n -th actual decision epoch occurs at the moment of the n -th arrival of a job. Remark that formally for the CTMDP model also at the virtual decision epochs (occurring at departures) some Markovian decision rule is applied. However, it is obvious that for both the implementation and the performance of a Markovian policy it does not matter which decision rule is applied at virtual decision epochs. Therefore slightly abusing notation we represent a Markovian policy such that only at actual decision rules it is indicated which decision rule is applied. The advantage is that for such representation $\pi = (d_1, d_2, \dots)$ we have for all $n \in \mathbb{N}$ that the Markovian decision rule d_n regulates the choice of server for the n -th arriving job.

Let $A = \cup_{s \in S} A(s)$ be the common action space for all states. Then a Markovian decision rule d is a mapping from the state space S into the set of probability distributions $\mathcal{P}(A)$ on A , that is $d : S \rightarrow \mathcal{P}(A)$. A Markovian policy $\pi = (d_1, d_2, \dots)$ is called stationary if for some decision rule d it holds that $d_n = d$ for all $n \in \mathbb{N}$. Thus a stationary Markovian policy is determined by a mapping $d : S \rightarrow \mathcal{P}(A)$ which is applied at every (actual) decision epoch.

For states s corresponding to virtual decision epochs the value of $d(s) \in \mathcal{P}(A)$ could be formally specified, but this formal specification would be irrelevant for the performance and implementation of d . Therefore in the sequel we identify Markovian decision rules d as mappings with domain $S^1 = \{(y_1, y_2, \dots, y_M, z) \in S : z \geq 1\}$ instead of domain S .

An advantage is that all $s \in S^1$ have the same finite action space $A = \{1, 2, \dots, N\}$ from which it follows that the decision rule d can be described by a mapping from S^1 to the standard $N - 1$ -dimensional simplex $X := \Delta^{N-1} = \{(x_1, x_2, \dots, x_N) : \sum_{j \in A} x_j = 1, x_j \geq 0 \forall j \in A\}$ having the N unit vectors $e_j, j \in A$ of $\mathbb{R}^N$, as vertices. Vice versa it is clear that any point in the huge infinite dimensional space X^{S^1} uniquely corresponds to a feasible Markovian decision rule d and thus also uniquely corresponds to the stationary Markovian policy $(d, d, \dots)$.

In case it holds for all $s \in S^1$ that there exists some $j(s) \in A$ such that $d(s)$ is the unit vector $e_{j(s)}$, then d is called a *deterministic* Markovian decision rule. From this definition it follows that deterministic Markovian decision rules can be identified with mappings from S^1 into A instead of X .

Since S^1 is a large space it is in general difficult to give a comprehensive description of a decision rule $d : S^1 \rightarrow X$. This difficulty of description is then also the case for Markovian policies and for subclasses which are important in practical applications like the subclass of stationary deterministic Markovian policies. From a practical viewpoint this implies that the implementation of Markovian decision rules and policies is an issue for MJTAPSS. Therefore an important consideration for investigating some subclass of policies is that policies in that subclass should be relatively easy to represent and to implement. Indeed for subclasses investigated in this paper the corresponding decision rules will use only some partial state information to choose a server for an arriving job. For these subclasses the decision rules can be described easily and the corresponding Markovian policies are easy to implement. In the following subsections some of these subclasses of easily implementable Markovian policies will be described.

3.1 Static policies

For an easy implementation of a decision rule it is best to use not too much information about the current system state to assign a job. In particular the following class of Markovian

policies fulfills this condition for an easy implementation. Let $s = (y_1, y_2, \dots, y_M, z) \in S$ be the state at an decision epoch. Then for this class of policies the choice of action $a \in A(s)$ may depend on z , the type of the arriving job, but not on the other components of the current state s . In other words such policies do not consider the current congestion of the servers to choose a server to which the arriving job is assigned. As in Becker et al. (2000) we call this the class of static policies. In Becker et al. (2000) the optimization over these static policies is investigated for MJTAPSS. We call a decision rule d itself a static rule and the corresponding stationary policy $(d, d, \dots)$ a static stationary policy if d has the property that the choice of action only depends on the z component of state s . Later in this paper we will also investigate some dynamic (non-static) decision rules and corresponding policies for the optimization of MJTAPSS. Then the best performance among static stationary policies will be a good reference to compare and evaluate the performance of the considered dynamic policies. We will show that for several dynamic decision rules and corresponding policies a performance improvement is possible by so-called mixing of the dynamic rule with an appropriate static rule. For reference we have the following formal definition of static decision rules and corresponding static stationary Markovian policies.

Definition 1 A decision rule $d : S \rightarrow \mathcal{P}(A)$ is said to be **static** if for all $s_1 = (y_1^1, y_2^1, \dots, y_N^1, z^1) \in S$ and $s_2 = (y_1^2, y_2^2, \dots, y_N^2, z^2) \in S$ it holds that $d(s_1) = d(s_2)$ if $z^1 = z^2$. Moreover, a stationary Markovian policy π is said to be a stationary static policy if $\pi = (d, d, \dots)$ for some static decision rule d .

Since for actual decision epochs $s \in S_1$ we have identified $\mathcal{P}(A)$ with $A = \{1, 2, \dots, N\}$ with the $N - 1$ -dimensional standard simplex $\Delta^{N-1} = \{(x_1, x_2, \dots, x_N) : \sum_{j \in A} x_j = 1, x_j \geq 0 \forall j \in A\}$, it follows that any static decision rule d is uniquely represented by an $M \times N$ matrix $R = (r_{ij})$ for which all the elements r_{ij} are nonnegative and all row sums are equal to one. In this representation r_{ij} is the probability that a type i job is assigned to server j if the static decision rule d is applied. Thus in contrast to general Markovian decision rules any static decision rules can be represented in a comprehensive way and by this representation it is obvious that static decision rules are relatively easy to implement. If we denote with $\mathcal{G}$ the class of Markovian static stationary policies $\pi = (d, d, \dots)$ for some static Markovian decision rule d then it follows that any policy $\pi \in \mathcal{G}$ is also uniquely represented by a matrix $R = (r_{ij})$ of assignment probabilities.

A subclass of static decision rules and corresponding static stationary policies are the so-called **deterministic** static decision rules and corresponding policies. This subclass is defined as follows.

Definition 2 A static decision rule d is said to be a deterministic static decision rule if for $M \times N$ matrix $R = (r_{ij})$ of assignment probabilities which uniquely represents d it holds that $r_{ij} \in \{0, 1\}$ for all $i \in \{1, 2, \dots, M\}$ and $j \in \{1, 2, \dots, N\}$.

A Markovian policy π is said to be a deterministic static stationary policy if $\pi = (d, d, \dots)$ for some deterministic static decision rule d .

We note that there exist N^M different deterministic static decision rules and corresponding matrices. Moreover, the convex hull of these N^M matrices representing deterministic static decision rules is exactly the space $\mathcal{H}$ representing all decision rules which we have defined to be static.

We now summarize the key results for application and optimization of static policies adapted to the notation and objectives of the current paper. For more details on these results

we refer to Becker et al. (2000). For static policy $\psi \in \mathcal{G}$ let $R = (r_{ij})$ be the unique matrix of assignment probabilities representing policy ψ . The performance of static policy $\psi = \psi(R)$ is defined by the formula

$$L(\psi(R)) = \sum_{i=1}^M L^i(\psi(R)). \quad (4)$$

The key observation to minimize this performance measure $\sum_{i=1}^M L^i(\psi(R))$ is that for static policies all queues $j \in \{1, 2, \dots, N\}$ are independent of each other and thus each queue becomes a standard $M/G/1$ queue for which mean waiting times can be determined exactly by applying the Pollaczek-Khintchine (see for example Tijms 2003) formula. After that the results for the separate queues can be combined to express for each type i the $L^i(\psi(R))$ in terms of decision variables r_{ij} which results in a mathematical programming problem to optimize the performance over all stationary static policies.

To apply the Pollaczek-Khintchine formula for $j \in \{1, 2, \dots, M\}$ it is sufficient to determine for queue j the aggregated arrival rate λ_j , the traffic intensity $\rho_j := \lambda_j E[S_j]$ and the second moment $E[(S_j)^2]$ where S_j is the service time distribution for jobs assigned to queue j . Note that since the S^{ij} are exponentially distributed it follows that the mixture S_j is hyperexponentially distributed.

It follows (see Becker et al. 2000) for $j = 1, 2, \dots, N$ that

$$\lambda_j = \sum_{i=1}^M \lambda_i r_{ij}, \quad (5)$$

$$\rho_j = \sum_{i=1}^M \lambda_i r_{ij} E[S^{ij}] = \sum_{i=1}^M \frac{\lambda_i r_{ij}}{\mu_{ij}} \quad (6)$$

and

$$E[(S_j)^2] = (1/\lambda_j) \sum_{i=1}^M \lambda_i r_{ij} E[(S^{ij})^2] = \frac{2 \sum_{i=1}^M \frac{\lambda_i r_{ij}}{(\mu_{ij})^2}}{\sum_{i=1}^M \lambda_i r_{ij}}. \quad (7)$$

Let $W(j)$ be the mean waiting time of all jobs assigned to queue j if static policy $\psi(R)$ is applied. By (5), (6), (7) and the Pollaczek-Khintchine formula it follows that

$$W(j) = \frac{\lambda_j E[(S_j)^2]}{2(1 - \rho_j)} = \frac{\sum_{i=1}^M \frac{\lambda_i r_{ij}}{(\mu_{ij})^2}}{1 - \sum_{i=1}^M \frac{\lambda_i r_{ij}}{\mu_{ij}}} \text{ if } \rho_j < 1, \quad (8)$$

while $W(j) = \infty$ if $\rho_j \geq 1$. Recall that $V^i(\psi(R))$ denotes the almost sure long-run average sojourn time of type i customers if static policy $\psi(R)$ is applied. By (8) it follows for all $i \in \{1, 2, \dots, M\}$ that $V^i(\psi(R))$ is finite if $\rho_j < 1$ for all $j \in \{1, 2, \dots, N\}$. In that case it follows for $i = 1, 2, \dots, M$ that

$$V^i(\psi(R)) = \sum_{j=1}^N r_{ij} (W(j) + E[S^{ij}]) = \sum_{j=1}^N \left(\frac{r_{ij} \sum_{k=1}^M \frac{\lambda_k r_{kj}}{(\mu_{kj})^2}}{1 - \sum_{k=1}^M \frac{\lambda_k r_{kj}}{\mu_{kj}}} + \frac{r_{ij}}{\mu_{ij}} \right) \quad (9)$$

and thus by Little's law it follows that

$$L^i(\psi(R)) = \lambda_i V^i(\psi(R)) = \lambda_i \sum_{j=1}^N \left(\frac{r_{ij} \sum_{k=1}^M \frac{\lambda_k r_{kj}}{(\mu_{kj})^2}}{1 - \sum_{k=1}^M \frac{\lambda_k r_{kj}}{\mu_{kj}}} + \frac{r_{ij}}{\mu_{ij}} \right) \quad (10)$$

Let $\mathcal{H}' = \{R \in \mathcal{H} : \rho_j < 1 \text{ for } j = 1, 2, \dots, N\}$ where ρ_j is determined by (6) for $j = 1, 2, \dots, N$. If $L^i(\psi(R))$ is determined by (10) for $i = 1, 2, \dots, M$ and $\mathcal{H}'$ is nonempty it follows that an optimal solution R^* of the mathematical programming problem

$$\min_{R \in \mathcal{H}'} L(\psi(R)) = \sum_{i=1}^M L^i(\psi(R)) \quad (11)$$

yields an optimal static policy ψ by $\psi = \psi(R^*)$. It is also shown in Becker et al. (2000) that if $\mathcal{H}'$ is nonempty that then an optimal solution of (11) and thus a corresponding optimal static policy exists. Indeed in that case replacing the search space $\mathcal{H}'$ by $\{R \in \mathcal{H} : \rho_j \leq 1 - \epsilon \text{ for } j = 1, 2, \dots, N\}$ for some $\epsilon > 0$ chosen small enough does not affect the optimal solution. In this way (11) is transformed to a problem of minimizing a continuous function on a compact space for which standard methods are available. However, given that the problem is nonlinear and has MN decision variables r_{ij} the problem appears rather difficult to solve unless M and N are small. On the other hand it is very helpful that it can be shown (see Sethumaran and Squillante 1999) that for this kind of mathematical programming problem a locally optimal solution is also a globally optimal solution.

3.2 The selfish policy

The so-called selfish policy π is a stationary Markovian policy $\pi = (d, d, \dots)$ where the decision rule d which is applied at all decision epochs is such that based on current state information the arriving job is assigned to a server such that the expected sojourn time of the arriving job is minimized without any consideration of the effect on sojourn times of future arrivals. In other words it is a myopic policy called to be selfish (SF) because this would happen if arriving jobs were allowed to make an individual decision based on current state information at the moment of arrival with only objective the minimization of the own expected sojourn time. Technically the SF decision rule d and corresponding policy π can be described as follows.

The SF decision rule and corresponding policy Given a current system state $s \in S$ determine k , the type of the arriving job, and for $i = 1, 2, \dots, M$, $j = 1, 2, \dots, N$ determine q_{ij} , the number of type i jobs present (including jobs being served) in the queue for server j .

Next determine $j' \in \{1, 2, \dots, N\}$ as the minimal number of a server such that $s_{kj'} = \min_j s_{kj}$ where for $j = 1, 2, \dots, N$,

$$s_{kj} := \sum_{i=1}^M \frac{q_{ij}}{\mu_{ij}} + \frac{1}{\mu_{kj}}$$

is the expected sojourn time for a type k job if it would be assigned to server j given the current partial state information determined by the q_{ij} . Then $d(s) = j'$. In other words if at the moment of arrival the system state is $s \in S$ then the arriving job is assigned to server $d(s) = j'$ if the SF policy $\pi = (d, d, \dots)$ is applied.

3.3 The virtual cost policy

Simulation confirms that the myopic SF policy can perform very poor especially in case of heavy traffic. In this subsection we introduce another dynamic policy which could perform

better than the SF policy. The assignment of jobs will depend on similar partial state information as for the SF rule and therefore the description and implementation of this policy $\pi = (d, d, \dots)$ will be as straightforward as for the SF policy. It turns out (see the results in Section 5) that this policy performs in general much better than the SF policy and the performance seems to be robust for more extreme cases. This new policy will be called the VC (virtual cost) policy and the corresponding decision rule for the assignment of arriving jobs the VC rule. Now a technical description of this VC rule is provided.

The VC rule and corresponding policy Given a current system state $s \in S$ determine k , the type of the arriving job. Moreover, for $i = 1, 2, \dots, M$, $j = 1, 2, \dots, N$ let q_{ij} be the number of type i jobs present in the queue for server j (as it is for the SF rule) and determine for $j = 1, 2, \dots, N$

$$q_j := \sum_{i=1}^M q_{ij}, \text{ the total number of jobs present in queue } j \text{ at the decision epoch.}$$

Next determine $j' \in \{1, 2, \dots, N\}$ as the minimal number of a server such that $u_{kj'} = \min_j u_{kj}$ where

$$u_{kj} := \frac{1 + q_j}{\mu_{kj}}.$$

Then $d(s) = j'$. In other words if at the moment of arrival the system state is $s \in S$ then the arriving job is assigned to server $d(s) = j'$ if the VC policy $\pi = (d, d, \dots)$ is applied.

Remark 1 Note that in case of $M = 1$ (only one type of job) it holds that $s_{kj} = u_{kj}$ for $j = 1, 2, \dots, N$. Thus in such a case the SF rule and VC rule in fact coincide.

Since the SF rule appears to be a reasonable decision rule one could wonder why in case of $M \geq 2$ the VC rule (which is apparently at least as simple to implement) in general performs better than the SF rule. An intuitive explanation of the relatively good performance of the VC rule is that $\frac{q_j}{\mu_{kj}}$ is a reasonable estimator of the total extra sojourn time of future arrivals assigned to queue j if virtually one extra type k job would be present in queue j . Indeed the number of future assignments to queue j for which the expected sojourn time is increased by $\frac{1}{\mu_{kj}}$ because of virtually adding one type k job to the jobs waiting in queue j , is estimated by q_j , the number of jobs currently present in the queue j . This is also the reason why the decision rule will be called the VC rule. Thus the VC rule takes into account the cost of the assignment for future arrivals while the SF rule only considers the cost for the currently arriving job. This could explain the possibly major differences in performance for a criterium function based on long-term average costs.

4 Performance improvement by mixing of decision rules

In the previous section we have introduced several decision rules for MJTAPSS. When we compare the performance of these decision rules we will confirm by simulation (simulation results will be presented in Section 5) that the SF decision rule has a reasonable performance in cases of light traffic, but its performance can be very poor in case of high traffic load. On the other hand static job-type policies with optimized assignment fractions perform rather well in case of heavy traffic, but relatively poor in case of light traffic. The performance of the VC rule appears to be robust in the sense that it is performing reasonably well for all kind of traffic load. However, simulation results will also show that by mixing of the three

types of decision rules it is often possible to obtain a performance which clearly improves on all the performances. For DTMDP a method to improve on a given set of decision rules by mixing them was investigated in van der Laan (2011). General results on the Markov chains induced by mixing decision rules were obtained and several technical issues were resolved. In particular it was investigated whether the performance of a policy obtained by mixing decision rules is independent of the initial state which is important to have a well defined performance of policies obtained by mixing decision rules. In van der Laan (2011) sufficient conditions are obtained for this. If these sufficient conditions are satisfied then it follows that the performance of mixing policies can be arbitrarily well approximated by simulation since in that case it follows that a simulated Cesàro mean of (weighted) sojourn times of arriving jobs converges with probability one to the performance of the applied mixing policy. We consider these conditions for MJTAPSS. An issue to consider is that MJTAPSS is modeled as a CTMDP while the results in van der Laan (2011) have been obtained for DTMDP. However, by a well-known uniformization technique (see for example Serfozo 1979) the CTMDP corresponding to the MJTAPSS can be transformed to an equivalent DTMDP. Thus uniformization of the CTMDP modeling MJTAPSS to a DTMDP can be used to apply the theoretical results from van der Laan (2011) to MJTAPSS.

In van der Laan (2011) a description is given of mixing Markovian decision rules for DTMDP with a finite state space. In the next subsection we recall some of these definitions and concepts. Some concepts are generalized from van der Laan (2011) to be applicable for the infinite countable state space associated with MJTAPSS.

4.1 Stationary mixing of decision rules

The main idea of mixing of Markov decision rules is to optimize over large class(es) of Markovian policies which are generated from a given finite set of Markovian decision rules. Let $\mathcal{D} = \{d^1, d^2, \dots, d^k\}$ be a given finite set of Markov decision rules from which the class(es) of policies are generated. It is assumed that all decision rules $d \in \mathcal{D}$ are applicable to the same DTMDP and thus all $d \in \mathcal{D}$ can be considered as mappings from some common state space S to $\mathcal{P}(A)$ for some common action space A .

By constructing for given $\mathcal{D}$ convex combinations of the elements of $\mathcal{D}$ a space $\mathcal{F}_D$ of mappings $S \rightarrow \mathcal{P}(A)$ representing so-called $\mathcal{D}$ -mixing rules is generated. Definition 3 shows how convex combinations of decision rules are constructed to generate the space $\mathcal{F}_D$.

Definition 3 Let $\mathcal{D} = \{d^1, d^2, \dots, d^k\}$ be a given set of k Markovian decision rules with common state space S and action space A and let

$$\Theta_k := \left\{ (\theta_1, \theta_2, \dots, \theta_k) \in \mathbb{R}^k : \theta_l \geq 0 \text{ for } l \in \{1, 2, \dots, k\}, \sum_{l=1}^k \theta_l = 1 \right\}$$

be the $k - 1$ -dimensional standard simplex.

Then for each $\theta \in \Theta_k$ a corresponding decision rule d_θ is defined by

$$d_\theta(s) := \sum_{l=1}^k \theta_l d^l(s) \text{ for all } s \in S. \quad (12)$$

Moreover, the space $\mathcal{F}_D$ of $\mathcal{D}$ -mixing rules is defined by

$$\mathcal{F}_D := \{d_\theta : \theta \in \Theta_k\}. \quad (13)$$

Remark 2 Since $\mathcal{P}(A)$ is convex and according to (12) $d_\theta(s)$ is a convex combination of elements in $\mathcal{P}(A)$ it is obvious that $d_\theta(s) \in \mathcal{P}(A)$ for all $s \in S$. Hence for all $\theta \in \Theta_k$ it follows that d_θ as defined by Definition 12 is indeed a randomized Markovian decision rule with state space S and action space A .

We also note that for any $\theta \in \Theta_k$ the decision rule d_θ is not (substantially) more difficult to implement than any of the decision rules $d^l \in \mathcal{D}$. In particular at any decision epoch d_θ may be implemented by first choosing $d^l \in \mathcal{D}$ with probability θ_l for $l = 1, 2, \dots, k$ and then implementing d^l .

Utilizing the space $\mathcal{F}_D$ of $\mathcal{D}$ -mixing rules several classes of Markovian policies can be considered for performance optimization. In particular consider the following subclass $\mathcal{G}_D$ of stationary policies.

Definition 4 Let $\mathcal{D}$ be given and $\mathcal{F}_D$ be defined by (13). Then for all $\theta \in \Theta_k$, $d_\theta \in \mathcal{F}_D$ let $\pi_\theta = (d_\theta, d_\theta, \dots)$ be the stationary Markovian policy for which d_θ is applied at every (actual) decision epoch. The class $\mathcal{G}_D$ of stationary $\mathcal{D}$ -mixing policies is then defined by

$$\mathcal{G}_D := \{\pi_\theta : \theta \in \Theta_k\}.$$

Now the performance can be optimized over this class $\mathcal{G}_D$. Optimization over this class comes down to finding the best convex combination of a finite number of points. To be precise note that any $d \in \mathcal{D}$ induces some infinite transition probability matrix P , i.e. some real-valued function P on $S \times S$ satisfying $P(x, y) \geq 0$ for $x, y \in S$ and $\sum_{y \in S} P(x, y) = 1$ for all $x \in S$. Given $\mathcal{D} = \{d^1, d^2, \dots, d^k\}$ let P^l , $l = 1, 2, \dots, k$ denote the infinite transition matrix induced by d^l . Then for any $\theta \in \Theta_k$ we have that

$$P_\theta = \sum_{l=1}^k \theta_l P^l,$$

where P_θ is the infinite transition matrix induced by d_θ .

The main idea of the optimization is that an optimal convex combination $P_\theta = \sum_{l=1}^k \theta_l P^l$ may have substantially better performance than any stationary policies induced by P^l , $l = 1, 2, \dots, k$, the extreme points of the convex combination. Such performance improvement is most interesting if $\mathcal{D}$ consists of a (small) selection of reasonable decision rules. For example the extreme points of the convex combination could be restricted to rules introduced in Sections 3.1, 3.2 and 3.3.

The performance of policies $\pi_\theta \in \mathcal{G}_D$ can be approximated by simulation of these policies. In such simulations these policies can be implemented as we have described in Remark 2. If k is not too large it turns out to be tractable to optimize in such a way the performance of π_θ over the $(k - 1)$ -dimensional simplex Θ_k .

In some cases it is besides approximation by simulation also possible to compute the performance π_θ exact by analytical methods for all $\theta \in \Theta_k$ and exact computations could enhance the optimization of the performance over Θ_k . In van der Laan (2011) such optimization utilizing exact performance computations is illustrated for problems with finite (and rather small) state and action spaces, but this method is in general not applicable for the infinite state space associated with MJTAPSS.

On the other hand it is for particular choices of $\mathcal{D}$ possible to compute the exact performance of $\pi_\theta \in \mathcal{G}_D$ by analytical methods. For example this is the case if all decision rules

$d^l \in \mathcal{D}$ are static decision rules. This follows from the fact that if all the $d^l \in \mathcal{D}$ are static that then $d_\theta \in \mathcal{F}_D$ is static for all $\theta \in \Theta_k$. Indeed let R_l be the matrix of assignment probabilities associated with static rule d^l for $l = 1, 2, \dots, k$. Then for all $\theta \in \Theta_k$ it is easily seen that $\sum_{l=1}^k \theta_l R_l$ is also a nonnegative $M \times N$ matrix with row sums equal to one representing the assignment probabilities of static rule d_θ . Moreover, we recall from Section 3.1 that for static decision rules d it is possible to compute exactly the performance $V(\psi)$ of the stationary policy $\psi = (d, d, \dots, d)$. Thus exact computations of the performances of $\pi_\theta \in \mathcal{G}_D$ are possible if $\mathcal{D}$ consists of only static decision rules.

4.2 Non-stationary mixing of decision rules

Recall (see 13) the definition of the space $\mathcal{F}_D$ of $\mathcal{D}$ -mixing rules and (see Definition 4) the class $\mathcal{G}_D$ of *stationary* $\mathcal{D}$ -mixing policies. With this in mind we now define a large class $\mathcal{K}_D$ of (possibly non-stationary) $\mathcal{D}$ -mixing policies and a subclass $\mathcal{L}_D$ of $\mathcal{K}_D$.

Definition 5 The class $\mathcal{K}_D$ consists of all Markovian policies $\pi = (d_1, d_2, \dots)$ for which $d_t \in \mathcal{F}_D$ for $t = 1, 2, \dots$

The subclass $\mathcal{L}_D \subseteq \mathcal{K}_D$ consists of all $\pi = (d_1, d_2, \dots) \in \mathcal{K}_D$ for which $d_t \in \mathcal{D}$ for $t = 1, 2, \dots$

From the definitions it is obvious that $\mathcal{G}_D \subseteq \mathcal{K}_D$ and thus the optimal performance within the class $\mathcal{K}_D$ is at least as good as the optimal performance within the class $\mathcal{G}_D$. However, even for small $k = |\mathcal{D}|$ the class $\mathcal{K}_D$ gives a very large and complex search space for optimization. Therefore it is not tractable to optimize the performance over the whole space $\mathcal{K}_D$. Also the subclass $\mathcal{L}_D$ is too large as search space for optimization. Moreover, most non-stationary policies in $\mathcal{K}_D$ and $\mathcal{L}_D$ can not be described in a comprehensive way and are in general also difficult to implement. Recall that as acceptable policies to apply for MJTAPSS we searched for Markovian policies which are easy to implement. Thus many policies in $\mathcal{K}_D$ and $\mathcal{L}_D$ are to our standard not acceptable for the assignment of arriving jobs to the servers. On the other hand in general (in van der Laan (2011) an explicit example is given) we have that $\mathcal{L}_D$ (and thus also $\mathcal{K}_D$) contain policies which perform better than the best performing policy in $\mathcal{G}_D$. In the sequel of this section we search for subclasses of $\mathcal{K}_D$ which ideally should have the following properties.

Criterion 1

1. Optimization over the whole subclass is tractable.
2. Policies within the subclass are not (substantially) more difficult to implement than policies in $\mathcal{G}_D$.
3. The subclass contains policies which perform better than the optimal policy within the class $\mathcal{G}_D$.

4.2.1 Periodic mixing policies

Subclasses which at first could be considered to satisfy Criterion 1 consist of periodic policies in $\mathcal{K}_D$ with some upper limit on the period of the policy. A Markovian policy $(d_1, d_2, \dots) \in \mathcal{K}_D$ is called periodic with period p if $d_t = d_{t+p}$ for all $t \in \mathbb{N}$. Note that the subclass of $\mathcal{K}_D$ of policies with period $p = 1$ is exactly the subclass $\mathcal{G}_D$ of stationary

$\mathcal{D}$ -mixing policies. To generalize this for any $p_0 \in \mathbb{N}$ we denote with $\mathcal{G}_D^{p_0}$ the subclass of $\mathcal{K}_D$ of policies with period $p \leq p_0$. Then $\mathcal{G}_D = \mathcal{G}_D^1 \subseteq \mathcal{G}_D^2 \subseteq \mathcal{G}_D^3 \subseteq \dots$ and the question is whether Criterion 1 is satisfied by a subclass $\mathcal{G}_D^{p_0}$ for some $p_0 > 1$. From a practical viewpoint it is problematic that $|\mathcal{G}_D^{p_0}|$ grows rapidly if p_0 increases. Therefore optimization over the whole space is tractable only if p_0 is small. Another issue to be considered is that the implementation of periodic policies becomes more difficult if the period grows. Thus to fulfill the first two points of Criterion 1 the maximal period p_0 should be quite small. On the other hand for smaller p_0 it is less likely that the third issue (improvement) of Criterion 1 is fulfilled. How large p_0 should be for an improvement on the optimal policy within $\mathcal{G}_D$ depends on the particular characteristics of the MDP and the choice of $\mathcal{D}$. In some cases a small period could be sufficient for an improvement, but for the complex problem of MJTAPSS in general no guarantees can be given.

Instead of such periodic policies in $\mathcal{K}_D$ in similar manner periodic policies within the smaller subclass $\mathcal{L}_D$ could be considered. Then the optimization should remain tractable for somewhat larger values of the maximal period p_0 . However to find an improvement over the optimal policies in $\mathcal{G}_D$ the necessary value of p_0 could increase in that case. Thus the issue remains that by considering subclasses of periodic policies one has a priori no guarantee whether Criterion 1 will be satisfied.

4.3 Mixing policies corresponding to billiard sequences

Apart from periodic policies with a bound on the period there is another subclass of $\mathcal{L}_D$ which according to our investigation is more promising to satisfy all aspects of Criterion 1. This is the subclass of policies $(d_1, d_2, \dots) \in \mathcal{K}_D$ which can be obtained as a so-called billiard sequence (see for example Arnoux et al. 1994; Baryshnikov 1995). This subclass will be denoted by $\mathcal{B}_D$. For given $\mathcal{D} = \{d^1, d^2, \dots, d^k\}$ billiard sequences and corresponding policies in $\mathcal{B}_D$ are obtained as follows.

Definition 6 A k -dimensional billiard sequence is determined by an initial position $x_0 \in \mathbb{R}^k$ and a normalized direction $\theta \in \Theta_k$ as follows. Let Ω be the subset of points in $\mathbb{R}^k$ having at least one integer coordinate. Then given x^0 and θ the corresponding billiard sequence is constructed from the intersection of the halfline $\{x^0 + t\theta | t > 0\}$ and Ω . We assume that x^0 is chosen such that the intersection consists only of points with exactly one integer coordinate. Let $0 < t_1 < t_2 < \dots$ be the unique increasing sequence of numbers in the countable set $\{t > 0 : x^0 + t\theta \in \Omega\}$. Then by the assumption it follows for all $n \in \mathbb{N}$ that there exists a unique $w_n \in \{1, 2, \dots, k\}$ for which $x_{w_n}^0 + t_n \theta_{w_n} \in \mathbb{Z}$. Thus the infinite sequence $w := (w_1, w_2, \dots)$ is well defined. This sequence is referred to as the k -dimensional billiard sequence determined by the initial position $x^0 \in \mathbb{R}^k$ and normalized direction $\theta \in \Theta_k$.

Remark 3 In Definition 6 it is assumed that all intersection points of the halfline and Ω have exactly one integer coordinate. Consider now otherwise that intersection points may contain multiple integer coordinates. Also in such a case it is possible to construct k -dimensional sequences from the intersection of the halfline and Ω . Namely suppose that $I = \{i \in \{1, 2, \dots, k\} : x_i^0 + t\theta_i \in \mathbb{Z}\}$ consist of multiple elements for some $t > 0$. Then at the place associated with such $t > 0$ some subsequence consisting of all $i \in I$ should be included in the sequence. Then these multiple elements of I could be permuted arbitrarily, but to obtain a sequence which is considered to be a proper billiard sequence (see for example Arnoux et al. 1994; Baryshnikov 1995) some order should be consequently applied for all intersection points with multiple integer coordinates. Namely for all intersection points for

which simultaneously $1 \leq i < j \leq k$ are associated with the integer coordinates of the intersection point either i should always be put before j in the permutation or vice versa.

It can be shown that all infinite sequences that can be obtained with such a construction can also be obtained as in Definition 6 by an appropriate adjustment of the initial position x^0 such that intersection points have not more than one integer coordinate. Thus the assumption in Definition 6 can be made without consequences.

Remark 4 From Definition 6 it is clear that the initial position and normalized direction vector uniquely determine an infinite billiard sequence. Vice versa given a k -dimensional billiard sequence the initial position $x^0 \in \mathbb{R}^k$ is not uniquely determined, but the element of Θ_k representing the normalized direction vector is uniquely determined by the infinite sequence. Namely let $w := (w_1, w_2, \dots)$ be an infinite k -dimensional billiard sequence and for $i = 1, 2, \dots, k$ let $N_i(n)$ be the number of i 's in the prefix $(w_1, w_2, \dots, w_n)$ of w . For k -dimensional billiard sequences it is known that for all $i \in \{1, 2, \dots, k\}$ the limit $\lim_{n \rightarrow \infty} \frac{N_i(n)}{n}$ exists. From the construction in Definition 6 it is easily seen that the normalized direction $\theta = (\theta_1, \theta_2, \dots, \theta_k)$ is uniquely determined by $\theta_i = \lim_{n \rightarrow \infty} \frac{N_i(n)}{n}$ for $i = 1, 2, \dots, k$.

Definition 7 Given $\mathcal{D} = \{d^1, d^2, \dots, d^k\}$ we define $\mathcal{B}_D \subseteq \mathcal{L}_D$ as the subset of $\mathcal{L}_D$ of all policies which correspond to k -dimensional billiard sequences $w = (w_1, w_2, \dots)$ which can be generated from an initial position and normalized direction as in Definition 6. Let $w = (w_1, w_2, \dots)$ be such a k -dimensional billiard sequence. Then the corresponding policy $\pi = (d_1, d_2, \dots) \in \mathcal{B}_D$ is defined by $d_t = d^{w_t}$ for $t = 1, 2, \dots$

We will investigate whether the three issues formulated in Criterion 1 hold for the subset $\mathcal{B}_D$ of $\mathcal{D}$ -mixing policies defined by Definition 7. The first issue is discussed immediately for two distinct cases in Sections 4.3.1 and 4.3.2. The second issue is discussed in Section 4.4 and the third issue is discussed in Section 5 with numerical results.

The first issue is whether performance optimization over the subclass $\mathcal{B}_D$ is tractable. The brief answer to that question is that indeed it is tractable provided that $k = |\mathcal{D}|$ is not too large and we are satisfied obtaining a performance within ε of the optimal performance with ε chosen arbitrarily small.

Namely under some mild assumptions which are satisfied for MJTAPSS optimization over $\mathcal{B}_D$ is similar to optimization over $\mathcal{G}_D$ (recall Definition 4). Moreover, recall that in case of $\mathcal{G}_D$ a continuous function should be optimized over the compact and convex $(k-1)$ -dimensional simplex Θ_k .

4.3.1 Billiard mixing for $k = 2$

Consider the case of $k = 2$ which is the most straightforward and arguably also the most important case. For $k = 2$ billiard sequences as defined by Definition 6 (which can be seen as playing billiards on a square table) there are other constructions and characteristic properties of sequences constructed in this way. Therefore these sequences are also known under other notions (see for example Morse and Hedlund 1940; Lothaire 2002) from which regular sequences and Sturmian words (sequences) are the most commonly used notions. A well-known property is that for $k = 2$ the subsequences of a billiard sequence do not depend on the initial position $x^0 \in \mathbb{R}^2$ with x^0 as in Definition 6. As a consequence of this we have the following proposition.

Proposition 1 *In case of $k = 2$ the performance of a policy in $\mathcal{B}_D$ is determined by the normalized direction vector $\theta \in \Theta_2$ of the billiard sequence that generates the policy. This implies that for $k = 2$ the performance optimization over $\mathcal{B}_D$ can be done by optimizing over the one-dimensional simplex $\Theta_2 = \{(\theta_1, \theta_2) : \theta_1 \geq 0, \theta_2 \geq 0, \theta_1 + \theta_2 = 1\}$.*

Note that the search space Θ_2 for optimization over $\mathcal{B}_D$ is the same (recall Definition 4) as for optimization over the class $\mathcal{G}_D$ of stationary $\mathcal{D}$ -mixing policies. On the other hand it should be noted that for the classes $\mathcal{B}_D$ and $\mathcal{G}_D$ the applicable methods to optimize over the associated search space Θ_2 may be different. Namely for the stationary policies in $\mathcal{G}_D$ it is sometimes possible (for example as we have seen if both decision rules in $\mathcal{D}$ are static) to obtain analytically the exact performance as function of $\theta \in \Theta_2$, while for the non-stationary policies in $\mathcal{B}_D$ generated by billiard sequences in general numeric methods like simulation should be used to estimate the performance for $\theta \in \Theta_2$. Optimization over $\mathcal{G}_D$ by exact methods can be considerably faster and with more precision than optimization over $\mathcal{B}_D$. On the other hand for general $\mathcal{D}$ some approximation of the performance has to be performed to optimize over $\mathcal{G}_D$. Then the speed and precision of optimization over $\mathcal{G}_D$ and $\mathcal{B}_D$ is comparable.

4.3.2 Billiard mixing for $k > 2$

After establishing for $k = 2$ the tractability of optimization of the performance over $\mathcal{B}_D$, we now consider the tractability for $k > 2$. An issue is that Proposition 1 can not be generalized to cases with $k > 2$. Indeed for $k > 2$ the performance of a policy in $\mathcal{B}_D$ depends in general not only on the normalized direction vector $\theta \in \Theta_k$, but also on the initial position $x^0 \in \mathbb{R}^k$. This follows from the fact that for $k > 2$ billiard sequences constructed with the same $\theta \in \Theta_k$ can have different finite subsequences if they are generated from different initial positions.

Therefore for $k > 2$ optimization over $\mathcal{B}_D$ is considerably more difficult than optimization over $\mathcal{G}_D$ since not only $\theta \in \Theta_k$ has to be optimized, but also the initial position $x^0 \in \mathbb{R}^k$ has to be optimized. On the other hand for fixed $\theta \in \Theta_k$ the number of different performances of policies generated by billiard sequences with direction θ is finite, but this finite number depends on θ . Moreover, if k gets larger then these numbers grow quickly. Only for small k and fixed $\theta \in \Theta_k$ it is tractable to optimize the performance by varying the initial position.

For larger values of k full optimization becomes untractable, but there is a practical approach to obtain solutions which should be close to optimal. This approach is based on the assumption that differences in performances caused by varying the initial position $x^0 \in \mathbb{R}^k$ are expected to be relatively small compared to differences in performance caused by varying the normalized direction vector $\theta \in \Theta_k$. In other words optimizing the normalized direction has much more effect than possible improvements obtained by varying the initial position keeping θ fixed. If the influence of the initial position is ignored then any fixed initial position can be chosen such that the assumption in Definition 6 is satisfied. Then the functions $f(\theta)$ and $g(\theta)$ for optimization over respectively $\mathcal{B}_D$ and $\mathcal{G}_D$ can be computed and compared as in the $k = 2$ case described earlier. However, it should be noted that by fixing some initial position it is if $k > 2$ not expected that the performance function $f(\theta)$ will be a smooth function. Therefore, despite the relaxation by fixing the initial position, optimization over $\mathcal{B}_D$ remains in case $k > 2$ more complicated than optimization over $\mathcal{G}_D$. We conclude that the optimization problem is tractable if k is very small, but it soon becomes much more complicated if k gets larger.

4.4 Implementation

Now that for $\mathcal{B}_D$ the question about the first aspect of Criterion 1 is answered we consider the second aspect (implementation) of Criterion 1 for subclass $\mathcal{B}_D$. Thus the question is whether the implementation of non-stationary policies in $\mathcal{B}_D$ generated by billiard trajectories can be done as easy and fast as the implementation of stationary policies in the class $\mathcal{G}_D$. The following remark is needed for this.

Remark 5 Let $\mathcal{D} = \{d^1, d^2, \dots, d^k\}$ and $\theta = (\theta_1, \theta_2, \dots, \theta_k) \in \Theta_k$ be given and let π be a Markovian policy with $\pi \in \mathcal{K}_D$ (recall Definition 5). If policy π is implemented let p_l denote for $l = 1, 2, \dots, k$ the long-run fraction of arriving jobs which are assigned to a server by applying decision rule d^l . Then for the subclasses $\mathcal{G}_D$ and $\mathcal{B}_D$ of $\mathcal{K}_D$ the following holds for the fractions $p_l, l = 1, 2, \dots, k$.

1. If $\pi = \pi_\theta \in \mathcal{G}_D$ is the randomized stationary policy given by Definition 4 and implemented as described in Remark 2 then with probability one it holds that $p_l = \theta_l$ for $l = 1, 2, \dots, k$.
2. If $\pi \in \mathcal{B}_D$ corresponds to a billiard sequence with normalized direction vector θ then (independent of the initial position) it holds that $p_l = \theta_l$ for $l = 1, 2, \dots, k$.

We have seen how the simplex Θ_k is utilized for optimization over both $\mathcal{G}_D$ and $\mathcal{B}_D$. From Remark 5 it can be deduced that the search space Θ_k also links the implementation of policies in these two classes. Namely it follows that all policies in these classes are relatively easy to implement by utilizing the unique $\theta \in \Theta_k$ that is associated to the policy. In the case of a policy $\pi_\theta \in \mathcal{G}_D$ at every actual decision epoch the decision rule $d^l \in \mathcal{D}$ to assign the arriving job can be determined by generating a new random number uniformly distributed on $[0, 1]$. Since an implementation (or computer simulation) of the policy is in practice always performed over a finite time horizon only a finite number of random numbers has to be generated to implement such a policy. In the second case for the implementation of a policy in $\mathcal{B}_D$ it is essential that (given normalized direction vector θ and some initial position) not the entire infinite billiard sequence corresponding to the policy has to be constructed, but only a finite prefix of the sequence. Moreover, for an online implementation it is not necessary to keep the constructed prefix in memory, but only the normalized direction vector and a current position has to be kept in memory. Then the decision rule in $\mathcal{D}$ to be applied at the next actual decision epoch is determined by an easy calculation after which the information on current position can be updated immediately.

We conclude that the difficulty of implementation of policies in $\mathcal{G}_D$ and $\mathcal{B}_D$ is comparable. The only difference in implementation is that at actual decision epochs in the former case a random number should be generated while in the latter case an easy calculation and update of a k -dimensional vector should be performed. In the section on numerical results more details on the implementation of the applied policies are given.

4.5 Comparing performances

In Section 5 simulation will be applied to simultaneously estimate performances for policies in $\mathcal{G}_D$ and $\mathcal{B}_D$. Moreover, for both methods the performance will be optimized over $\theta \in \Theta_2$. Performing these optimizations simultaneously it is natural to compare for any $\theta \in \Theta_2$ the performance $g(\theta)$ of the corresponding policy in $\mathcal{G}_D$ with the performance $f(\theta)$ of a corresponding policy in $\mathcal{B}_D$.

Under certain conditions it is shown in Hajek (1985) and Altman et al. (2003) that the implementation of a billiard (regular) sequence gives the best performance if θ is fixed. In particular results have been obtained if the performance function can be shown to be multimodal. In Hajek (1985) multimodality of the performance function is established for the admission of arriving jobs in a single queue. In Altman et al. (2003) multimodality is obtained for a variety of (in particular open-loop) queueing problems. A similar result for mixing decision rules for MJTAPSS would imply for $k = 2$ that $f(\theta) \leq g(\theta)$ if both are finite. However, for mixing decision rules for MJTAPSS the methods and conditions given in Altman et al. (2003) to obtain multimodality are not directly applicable. On the other hand in van der Laan (2011) the optimality of regular sequences for mixing $k = 2$ decision rules for any $\theta \in \Theta_2$ is considered for general DTMDP. In that paper the optimality of regular sequences is under some minor condition (but without any assumption on multimodality) established for DTMDP with two states, but also that partial result is not directly applicable to MJTAPSS. Thus for MJTAPSS it is not yet established that for all $\theta \in \Theta_2$ it holds that $f(\theta) \leq g(\theta)$. However, in general such inequality should hold if function $g(\theta)$ is locally convex around θ . Such convexity property is usually satisfied and confirmed by numerical results. In the next section for cases of MJTAPSS with varying parameters and specified $\mathcal{D}$ the optimization over $\mathcal{G}_D$ and $\mathcal{B}_D$ is performed by numerical methods. Simultaneously the numerical estimations of $f(\theta)$ and $g(\theta)$ will be compared over the search space Θ_2 . If $f(\theta) \leq g(\theta)$ for $\theta \in \Theta_2$ is confirmed then optimization over $\mathcal{B}_D$ provides a better optimized performance than optimization over $\mathcal{G}_D$ while the required computation time is comparable. For the subclass $\mathcal{B}_D$ the last aspect (improvement) of Condition 1 will be established in this way.

5 Numerical results

For MJTAPSS performance optimization by mixing decision rules as described in the previous section we present in this section some numerical results obtained by simulation. In these numerical experiments first several instances of MJTAPSS with $M = 2$ types of arriving customers and $N = 2$ servers are considered. The performance measure V is the long-run average sojourn time as defined by (1). Note that for MJTAPSS with these characteristics the six parameters $\lambda_1, \lambda_2, \mu_{11}, \mu_{12}, \mu_{21}, \mu_{22}$ completely determine the instance. For the first four instances the experiments on performance improvement by mixing decision rules are performed for $\mathcal{D}$ consisting of $k = 2$ different decision rules selected from the following three basic decision rules denoted by DS12, SF and VC. Decision rule DS12 stands for the deterministic static decision rule by which type 1 jobs are always assigned to server 1 and type 2 jobs are always assigned to server 2, SF stands for the SF rule as described in Section 3.2 and VC stands for the virtual cost rule as described in Section 3.3. In the fifth instance not DS12, but another static rule is applied. In the sixth and final instance of MJTAPSS we will investigate we have $M = 3$ types of arriving customers and $N = 3$ servers.

We first discuss the technical details about the simulations which are performed. Consider the case that performances of mixing policies are estimated by simulation for some given $\mathcal{D}$ consisting of $k = 2$ decision rules. Then for both Bernoulli-mixing and billiard-mixing the performance will be estimated simultaneously for several points $(\theta, 1 - \theta) \in \Theta_2$ spread out over the search space Θ_2 . Typically in the first simulation round the performance estimation will be done for both Bernoulli-mixing and billiard-mixing for θ in the set

$\{\frac{a}{10}, a = 0, 1, \dots, 10\}$. Thus a total of 22 performance values are then simulated in the first round. To get a good comparison between the performances of all the corresponding simulated mixing policies the simulation runs are performed with common random numbers. Each simulation run takes at least 10000 decision epochs. Starting from a empty system the chosen startup period before the performance is measured varies from 1000 epochs for the investigated instance with relatively light traffic until 10000 epochs for the investigated instance with the most heavy traffic. After that each simulation run continues for 10000 decision epochs. By using common random numbers the simulated arrival process is for each simulation run the same for all simulated mixing policies and differences in performance only happen because in the simulated sample path the policies assign the jobs differently to the servers. Such common random number simulation runs are repeated until for all simulated policies the width of the 95 % confidence interval for the evaluated performance is considered to be small enough compared to the average (over the performed simulation runs) performance of the corresponding policy. In particular a simulation round is terminated when for all simulated policies the computed quotient of the half-width of the confidence interval and the average performance is smaller than some chosen precision parameter ε . In the first simulation round typically $\varepsilon = 0.05$ in case of moderate traffic and $\varepsilon = 0.10$ in case of heavy traffic will be chosen as stop criterium.

When the first round is terminated it can from the simulation results be inspected around which point $(\theta, 1 - \theta) \in \Theta_2$ the performance is optimal. Typically there is for Bernoulli-mixing and billiard-mixing a small difference for which point in Θ_2 the performance is optimal. After the first simulation round for both methods the optimal point $(\theta, 1 - \theta) \in \Theta_2$ and corresponding performance will be obtained more precisely by zooming in around the best performing points in the first simulation round. This is done by performing a second simulation round by simulating performances for additional points around the points which had the best performance in the first round. Because in the smaller interval the differences in performance will be relatively small more precision in the performance estimations is required. Therefore the second simulation round is performed with ε at least a factor two smaller than in the first round. This zooming in around the optimal point may be repeated, but the duration of the simulation rounds grows quickly when ε is chosen smaller. Therefore for the current problem the zooming is terminated after the second round when the differences in performance are usually already relatively small.

Now that the technical details of the performed simulations and the optimization procedure have been discussed the obtained numerical results for several instances of MJTAPSS will be presented and evaluated. Representing these results graphs are included where the x-axis represents the value of $\theta \in [0, 1]$ determining $(\theta, 1 - \theta) \in \Theta_2$ and the y-axis represents the average sojourn time as defined by (1) of the simulated Bernoulli-mixing and billiard-mixing policies. Whether a dot in the graph corresponds to a Bernoulli-mixing policy performance or a billiard-mixing policy performance is indicated by the color of the dot. Red dots are used for performances for Bernoulli-mixing and blue dots for billiard-mixing.

Instance 1: The first instance is determined by the system parameters $\lambda_1 = 1.0, \lambda_2 = 1.0, \mu_{11} = 1.3, \mu_{12} = 2.0, \mu_{21} = 0.4$ and $\mu_{22} = 1.2$ for which the traffic load can be regarded as moderately heavy. Recall that the performance of static policies such as DS12 can be computed exactly by applying the Pollaczek-Khintchine formula for each queue separately. In this case the exact computation is very easy since for the pure DS12 policy the two queues become independent $M/M/1$ queues. For type 1 jobs it follows that the average sojourn time is $V^1(DS12) = \frac{1}{\mu_{11} - \lambda_1} = \frac{1}{1.3 - 1.0} = \frac{10}{3}$. Similarly it follows

for the average sojourn time of type 2 jobs that $V^2(DS12) = \frac{1}{\mu_{22}-\lambda_2} = \frac{1}{1.2-1.0} = 5$ and thus the overall average sojourn time for DS12 is $V(DS12) = \frac{\lambda_1}{\lambda_1+\lambda_2}V^1(DS12) + \frac{\lambda_2}{\lambda_1+\lambda_2}V^2(DS12) = \frac{5}{3} + \frac{5}{2} = \frac{25}{6} \approx 4.167$. By carrying out the optimization program described in Becker et al. (2000) it can be verified that for this instance DS12 is optimal among all static assignment policies. Thus for this instance we have that among static policies the best possible performance is $\frac{25}{6}$.

We first investigate the policy mixing for $\mathcal{D} = \{DS12, SF\}$ to see if the performance of 4.167 can be improved by this mixing. The first simulation round is performed with $\varepsilon = 0.05$ and the results are presented in Fig. 1.

About the results presented in Fig. 1 the following is noticed. First of all it should be clear that for pure policies corresponding to $\theta = 0$ and $\theta = 1$ there never is a difference in simulated performances between Bernoulli-mixing and billiard-mixing since common random numbers are used and for the corresponding pure SF and DS12 policies the implemented policy will be exactly the same for the two methods. The pure SF policy corresponding to $\theta = 0$ seems to have simulated performance around 5 which is worse than the simulated performance of the pure DS12 policy corresponding to $\theta = 1$ which appears to be slightly more than 4. Note that this simulation result for $\theta = 1$ is in accordance with the theoretical value of 4.167 which we have computed. In general it is known that the SF policy performs poorly for heavy traffic, but in this case also for moderate traffic DS12 already performs considerably better. According to Fig. 1 if θ is chosen around 0.8 the performance of the corresponding mixing policy is about 3.5 which is considerably better than $\frac{25}{6}$ for the static policy DS12. Moreover, in this region of the search space Θ_2 the performances for billiard-mixing are apparently slightly better than for Bernoulli-mixing. The difference in performance between the two methods seems to be about 0.10. For an average sojourn time around 3.5 this means that the relative improvement in performance by applying billiard mixing instead compared to Bernoulli-mixing is about 3%. In the second simulation

Fig. 1 Mixing SF and DS12 for Instance 1

round we zoom in on the region with the best performing values of θ . Therefore in the second round the performance is simulated for $\theta = 0.60 + 0.05k$ for $k = 0, 1, \dots, 8$ and ε is chosen to be 0.025. The results of this second simulation round are presented in Fig. 2.

Figure 2 reinforces the results of the first round. For both Bernoulli-mixing and billiard-mixing the optimal performance for $(\theta, 1 - \theta) \in \Theta_2$ seems to be attained for θ somewhere between 0.70 and 0.80. It is hard to obtain the optimal value of θ more precisely because then considerably lower values of ε would be necessary and the simulation would then take a very long time. Methods like gradient estimation by simulation (see for example Heidergott et al. 2010; Bhulai et al. 2012) could speed up the optimization of θ , but in regions where the function is almost flat it remains difficult. Therefore we focus on the relative size of performance improvements which has more practical value than obtaining the optimal value of θ with high precision. Indeed by focusing on performance the optimal value of θ does not have to be very precise since the performance function is more or less flat around the optimal value of θ . The results represented in Fig. 2 confirm that for mixing SF and DS12 the best performances are obtained by billiard-mixing. By optimal billiard-mixing of SF and DS12 the relative improvement compared to DS12 seems to be around 15% which is quite significant. Additionally it is confirmed that the performance of the optimal billiard-mixing policy is a few percent better than for the optimal Bernoulli-mixing policy.

Now that we have investigated the mixing of decision rules for $\mathcal{D} = \{DS12, SF\}$ we similarly investigate the mixing of decision rules for $\mathcal{D} = \{DS12, VC\}$. In this case $\theta = 0$ corresponds to the pure VC policy while $\theta = 1$ corresponds to the pure DS12 policy. The first simulation round is again performed with $\varepsilon = 0.05$ and the results are presented in Fig. 3.

According to Fig. 3 the performance of the VC policy is around 3.5 which is by coincidence as good as the best performance which could be obtained by mixing SF and DS12. It is clear that the VC policy performs considerably better than the pure SF and pure DS12 rule. Moreover, From Fig. 3 it is clear that by mixing VC with the DS12 rule the

Fig. 2 Second round of mixing SF and DS12 for Instance 1

Fig. 3 Mixing VC and DS12 for Instance 1

performance can be further improved and that for this mixing the optimal performance is attained for values of θ around 0.50. Therefore in the second round the performance is simulated for $\theta = 0.30 + 0.05k$ for $k = 0, 1, \dots, 8$ and ε is chosen to be 0.025. The results of this second simulation round are presented in Fig. 4.

In Fig. 4 it is confirmed that the optimal value of θ is close to 0.50 and the slight difference in optimal performance between Bernoulli-mixing and billiard-mixing is visible once more. This difference is around 0.05 which is smaller than it was for mixing SF and DS12. The overall optimal performance for billiard-mixing of VC and DS12 seems to be slightly below 3.40 which improves slightly on the performance of the pure VC policy. An

Fig. 4 Second round of mixing VC and DS12 for Instance 1

explanation for the relatively small improvement is that for this instance the VC policy performs already rather well and thus it is much more difficult to improve upon than it was for the pure SF and DS12 policies. By mixing the VC policy with DS12 still a few percent of improvement is achieved which could well be worth the effort especially since it does not take much time and it is not difficult to implement.

Finally investigating the mixing with $\mathcal{D} = \{SF, VC\}$ illustrates that not always an improvement is achieved if two suboptimal decision rules are mixed. In this case $\theta = 0$ corresponds to the pure SF policy while $\theta = 1$ corresponds to the pure VC policy. The simulation is performed with $\varepsilon = 0.05$ and the results are presented in Fig. 5. Figure 5 shows that the pure VC policy performs better than any genuine mixing between the SF and VC policy.

Instance 2: The second instance is determined by the parameters $\lambda_1 = 2.00$, $\lambda_2 = 1.00$, $\mu_{11} = 2.10$, $\mu_{12} = 0.80$, $\mu_{21} = 1.30$ and $\mu_{22} = 1.10$.

It should be noted that these parameters give a rather heavy traffic load. Similar to Instance 1 the performance of the pure DS12 policy can also be computed exactly by applying the Pollaczek-Khintchine formula for each queue separately. For type 1 jobs it follows that the average sojourn time is $V^1(DS12) = \frac{1}{\mu_{11} - \lambda_1} = \frac{1}{2.10 - 2.00} = 10$. It follows for the average sojourn time of type 2 jobs that $V^2(DS12) = \frac{1}{\mu_{22} - \lambda_2} = \frac{1}{1.10 - 1.00} = 10$ and thus the exact overall average sojourn time for DS12 is $V(DS12) = \frac{2}{3}V^1(DS12) + \frac{1}{3}V^2(DS12) = 10$. For this instance performing the optimization over all static policies as described in Becker et al. (2000) it turns out that DS12 is not the optimal static policy but it is very close to optimal. Indeed in this case the optimal static policy assigns a small fraction (about 3 out of 1000) of type 1 jobs to queue 2 which gives a performance of about 9.936.

For $\mathcal{D} = \{DS12, SF\}$ a first simulation round has been performed with $\varepsilon = 0.10$. The pure SF policy (induced by choosing $\theta = 0$) performs in general very bad in instances with heavy traffic load. For this particular instance it turned out that the system is not even

Fig. 5 Mixing VC and SF for Instance 1

stable for low values of θ and simulation results indicate that the system is unstable for $\theta = 0.60$ or lower. Moreover, for $\theta = 0.70$ the simulated performance value is around 25 which is much worse than the simulated average sojourn time for the pure DS12 policy corresponding to $\theta = 1.0$ which as expected is about the theoretical value of 10. The differences in the performances of Bernoulli-mixing policies and billiard-mixing policies with the same $(\theta, 1 - \theta) \in \Theta_2$ seem to be relatively small for this instance. Indeed these performance differences are very small compared to the performance value of almost 25 in case $\theta = 0.70$ when the system is hardly stable. By zooming in on the interval $[0.8; 1]$ we expect that differences in performance become more apparent and for both methods the optimal value of θ and the corresponding performance can be determined more precisely. Therefore in the second round the performances have been simulated with $\varepsilon = 0.05$ for $\theta \in \{0.80, 0.85, 0.90, 0.95, 1.00\}$. The results of this second round of simulations are presented in Fig. 6.

According to Fig. 6 for both Bernoulli-mixing and billiard-mixing the optimal value of θ seems to be about 0.95. The differences in performance between Bernoulli-mixing and billiard-mixing still seem to be relatively small for this instance. On the other hand for both mixing methods the performance for the optimal mixing policy seems to be close to 8.5 which is considerably better (the improvement is about 15%) than the performance of 10 of the pure DS12 policy and the performance of 9.936 of the optimal static policy. This improvement is quite remarkable since in this instance the optimal mixing policy applies DS12 on a large majority (about 95%) of the decision epochs while the SF rule is only applied on the remaining 5% of the decision epochs.

Next the optimal mixing is investigated for $\mathcal{D} = \{DS12, VC\}$ for which we expect (as for Instance 1) to obtain even better performances than for $\mathcal{D} = \{DS12, SF\}$ -mixing. For $\mathcal{D} = \{DS12, VC\}$ we have that $\theta = 0$ corresponds to applying the pure VC policy and $\theta = 1$ corresponds to applying the pure DS12 policy. Performing a first simulation round with $\varepsilon = 0.10$ it turns out that for this mixing the system is stable for all mixing parameters $\theta \in [0, 1]$ and thus otherwise than for $\{DS12, SF\}$ -mixing for $\{DS12, VC\}$ -mixing

Fig. 6 Second round of mixing SF and DS12 for Instance 2

performance values can be simulated over the entire interval. Moreover, the performance of the pure VC policy seems to be slightly above 8.5 which is close to the best performance which could be obtained by mixing SF and DS12. Recall that by coincidence also for Instance 1 the performance of the pure VC policy was similar to the best performance which could be obtained by mixing SF and DS12. Again the performance of the pure VC policy can be improved upon by mixing VC with the DS12 rule. Indeed it seems that the optimal performance is attained for values of θ around 0.8 and that then a performance around 8 or even better can be obtained. Also around the optimal value of θ billiard-mixing seems to perform slightly better than Bernoulli-mixing. To investigate this in more detail in the second round we have chosen to simulate the performance for $\theta = 0.70 + 0.05k$ for $k = 0, 1, \dots, 6$ with $\varepsilon = 0.05$. The results of this second simulation round are presented in Fig. 7.

Figure 7 shows that for both mixing methods the optimal value of θ is slightly above 0.8 and the corresponding performances are around 8. The improvement compared to the optimal static policy is then about 20% while the improvement compared to optimal mixing of SF and DS12 or the pure VC policy is about 5%. Moreover, around the optimal value of θ billiard-mixing consistently performs better than Bernoulli-mixing, but performance differences between the two methods seem to be not more than 1% and thus relatively small.

To conclude the investigation of Instance 2 also simulation have been performed for $\mathcal{D} = \{SF, VC\}$ -mixing. However, as in Instance 1 it turns out that the pure VC policy performs better than any genuine mixing between the SF and VC policy. Thus the corresponding performance graph turns out to be monotonically decreasing as in Fig. 5 for Instance 1. In other words once more $\{SF, VC\}$ -mixing does not provide results which can be used to improve on the pure VC policy.

Fig. 7 Second round of mixing VC and DS12 for Instance 2

Instance 3: The third instance is determined by the parameters $\lambda_1 = 3.00$, $\lambda_2 = 2.00$, $\mu_{11} = 5.00$, $\mu_{12} = 1.00$, $\mu_{21} = 2.00$ and $\mu_{22} = 3.00$.

Note that the traffic load is rather light compared to the previous instances. Again the performance of the pure DS12 policy can be computed exactly by applying the Pollaczek-Khintchine formula for each queue separately. For type 1 jobs it follows that the average sojourn time is $V^1(DS12) = \frac{1}{\mu_{11}-\lambda_1} = \frac{1}{5.0-3.0} = 0.5$. Similarly it follows for the average sojourn time of type 2 jobs that $V^2(DS12) = \frac{1}{\mu_{22}-\lambda_2} = \frac{1}{3.0-2.0} = 1.0$ and thus the overall average sojourn time for DS12 is $V(DS12) = \frac{\lambda_1}{\lambda_1+\lambda_2} V^1(DS12) + \frac{\lambda_2}{\lambda_1+\lambda_2} V^2(DS12) = \frac{3}{10} + \frac{2}{5} = \frac{7}{10} = 0.7$ which is in accordance with the following results obtained by simulation. By carrying out the optimization program described in Becker et al. (2000) it can be verified that for this instance DS12 is optimal among all static assignment policies. Thus among static policies the best possible performance is 0.7. For $\mathcal{D} = \{DS12, SF\}$ a first simulation round has been performed with $\varepsilon = 0.05$. Then the simulated performance of the pure SF policy corresponding to $\theta = 0$ seems to be around 0.9 which is worse than the simulated performance of the pure DS12 policy corresponding to $\theta = 1$. Moreover, the results indicate that by mixing SF and DS12 the best performing policies are obtained for values for θ around 0.8. Also around this optimal value of θ billiard-mixing seems to perform slightly better than Bernoulli-mixing. Therefore in the second round we have simulated the performance for $\theta = 0.60 + 0.05k$ for $k = 0, 1, \dots, 8$ with $\varepsilon = 0.025$. The results of this second simulation round are presented in Fig. 8.

Figure 8 is in accordance with the results of the first round. For both Bernoulli-mixing and billiard-mixing the optimal performance seems to be attained for θ somewhere between 0.80 and 0.90 and the results of this second round confirm that for the mixing policy with optimal θ the performance is a few percent better than the 0.7 of the pure DS12 policy.

Fig. 8 Second round of mixing SF and DS12 for Instance 3

Moreover, again it is seen that around the optimal value of θ corresponding billiard-mixing policies perform slightly better than corresponding Bernoulli-mixing policies.

Next the optimal mixing is investigated for $\mathcal{D} = \{DS12, VC\}$ for which we now expect (as for Instance 1 and Instance 2) to obtain even better performances than for $\mathcal{D} = \{DS12, SF\}$. For $\mathcal{D} = \{DS12, VC\}$ we have that $\theta = 0$ corresponds to applying the pure VC policy and $\theta = 1$ corresponds to applying the pure DS12 policy. A first simulation round has been performed with $\varepsilon = 0.05$ and the results indicate that by $\mathcal{D} = \{DS12, VC\}$ -mixing performances close to 0.62 can be obtained which would improve more than 10% on the optimal static policy DS12. Also for $\theta < 0.7$ the performance function appears to be rather flat while for $\theta > 0.7$ the simulated performance values are clearly increasing in θ . Therefore in the second simulation round we zoom in on the region with $\theta < 0.7$ to determine the optimal value of θ more precisely. In the second round the performance values have been simulated for $\theta = 0.05k$ for $k = 0, 1, \dots, 13$ with $\varepsilon = 0.01$. The results of this second simulation round are presented in Fig. 9.

From Fig. 9 it seems that for billiard-mixing the optimal value of θ is around 0.5 while for Bernoulli-mixing it seems to be somewhat below 0.4. However, the flatness of the function around the optimum makes it for both methods difficult to determine the optimal point very precise. Besides from Fig. 9 it is apparent that around the optimal point billiard-mixing gives consistently better performances than Bernoulli-mixing although these differences are relatively small.

To conclude the investigation of Instance 3 numerical results for $\mathcal{D} = \{SF, VC\}$ -mixing confirm also for Instance 3 that no improvement is obtained by $\mathcal{D} = \{SF, VC\}$ -mixing since again it turns out that the pure VC policy performs better than any genuine mixing between the SF and VC policy.

Instance 4: The fourth instance is determined by the parameters $\lambda_1 = 1.00$, $\lambda_2 = 2.00$, $\mu_{11} = 5.00$, $\mu_{12} = 4.00$, $\mu_{21} = 3.00$ and $\mu_{22} = 6.00$.

The traffic load is light in this instance. Again the performance of the pure DS12 policy can be computed exactly by applying the Pollaczek-Khintchine formula for each queue

Fig. 9 Second round of mixing VC and DS12 for Instance 3

separately. For type 1 jobs it follows that the average sojourn time is $V^1(DS12) = \frac{1}{\mu_{11} - \lambda_1} = \frac{1}{5.0 - 1.0} = 0.25$. Similarly it follows for the average sojourn time of type 2 jobs that $V^2(DS12) = \frac{1}{\mu_{22} - \lambda_2} = \frac{1}{6.0 - 2.0} = 0.25$ and thus the overall average sojourn time for DS12 is 0.25 which is in accordance with the results obtained by simulation. Moreover, among static policies the best possible performance is 0.25 since for this instance DS12 is optimal among the static policies. For $\mathcal{D} = \{DS12, SF\}$ a first simulation round has been performed with $\varepsilon = 0.05$. The simulated performance of the pure SF policy corresponding to $\theta = 0$ seems to be below 0.24 which is slightly better than the simulated performance of the pure DS12 policy corresponding to $\theta = 1$. It also seems that by mixing SF and DS12 a slight improvement on both SF and DS12 can be obtained and that around the optimal value of θ billiard-mixing seems to perform better than Bernoulli-mixing. In the second simulation round we have simulated the performance for $\theta = 0.05k$ for $k = 0, 1, \dots, 12$ with $\varepsilon = 0.01$. The results of this simulation are presented in Fig. 10.

From Fig. 10 it appears that for Bernoulli-mixing $\theta = 0$ is optimal which implies that the performance of SF can not be improved by Bernoulli-mixing with DS12. However, for billiard-mixing the optimal performance seems to be attained around $\theta = 0.5$. Around this value for θ billiard-mixing performs clearly better than Bernoulli-mixing. Moreover, billiard-mixing with θ around 0.5 slightly improves the performance of the pure SF policy.

Next the optimal mixing is investigated for $\mathcal{D} = \{DS12, VC\}$ where $\theta = 0$ corresponds to applying the pure VC policy and $\theta = 1$ corresponds to applying the pure DS12 policy. The simulation has been performed with $\varepsilon = 0.01$ for which the results are presented in Fig. 11.

Fig. 10 Second round of mixing SF and DS12 for Instance 4

Fig. 11 Mixing VC and DS12 for Instance 4

From the simulation results it seems that $\theta = 0$ is optimal for both Bernoulli-mixing and billiard-mixing. Thus it appears that the performance of VC can not be improved by mixing with DS12. The performance of the VC policy seems to be slightly below 0.23 which is better than the performance of the SF policy. To conclude the investigation of Instance 4 numerical results for $\mathcal{D} = \{SF, VC\}$ -mixing confirm as in the previous instances that no improvement is obtained by $\mathcal{D} = \{SF, VC\}$ -mixing. Also in this light traffic instance the pure VC policy performs better than any genuine mixing between the SF and VC policy.

In this light traffic instance we have seen that the performances of all considered heuristics are relatively close to each other. The performance of SF is in this case slightly better than the performance of DS12 which is different from the previous instances with heavier traffic. VC still performs better than SF. In this instance the performance of VC can not be improved by mixing with DS12.

Instance 5: The fifth instance we present is determined by the parameters $\lambda_1 = 5.00$, $\lambda_2 = 2.00$, $\mu_{11} = 4.00$, $\mu_{12} = 3.00$, $\mu_{21} = 3.00$ and $\mu_{22} = 6.00$. We first note that if policy DS12 would be applied that queue 1 is not stable since $\lambda_1 > \mu_{11}$. Some of the type 1 jobs have to be routed to queue 2 to get a stable system. To obtain a good randomized static policy for this instance the mathematical programming problem as described Section 3.1 has been solved. For the optimal solution in variables r_{ij} we have obtained that $r_{11} \approx 0.70$, $r_{12} = 1 - r_{11} \approx 0.30$, $r_{21} = 0$ and $r_{22} = 1$. The static policy with routing probabilities $r_{11} = 0.70$, $r_{12} = 0.30$, $r_{21} = 0$ and $r_{22} = 1$ will be denoted by R70 (Randomized 70). We will mix R70 with the dynamic policies SF and VC. The first simulation round has been performed with $\varepsilon = 0.10$ for $\mathcal{D} = \{R70, SF\}$ -mixing. The results are presented in Fig. 12.

From Fig. 12 it seems that the performance of SF ($\theta = 0$) is around 1.8 while the performance of R70 ($\theta = 1$) is slightly below 1.8. To verify these simulation results we have

Fig. 12 Mixing SF and R70 for Instance 5

also computed the performance of static policy R70 by applying the Pollaczek-Khintchine formula. This calculation gives 1.786 (rounded to three decimals) which is in accordance with the simulation results. According to Fig. 12 the performance is optimized for θ around 0.8. Therefore in the second round the performance is simulated for $\theta = 0.60 + 0.05k$ for $k = 0, 1, \dots, 8$ and ε is chosen to be 0.05. The results of this second simulation round are presented in Fig. 13.

Fig. 13 Second round of mixing SF and R70 for Instance 5

As for previous instances we observe that billiard mixing performs slightly better than Bernoulli mixing around the optimal θ . The optimal value for θ seems to be around 0.75 and the corresponding performance is less than 1.4. This is a substantial improvement of more than 20 % on the performances of SF and R70.

Next the mixing policies are simulated for $\mathcal{D} = \{R70, VC\}$ where $\theta = 0$ corresponds to applying the VC policy and $\theta = 1$ corresponds to applying the R70 policy. The simulation has been performed with $\varepsilon = 0.05$ for which the results are presented in Fig. 14. It seems that the performance of VC is around 1.0 and the average sojourn time is slowly increasing in θ . Thus it seems that in this instance the performance of VC can not be improved by mixing with R70.

Finally the mixing policies have been simulated for $\mathcal{D} = \{SF, VC\}$. As in the previous instances the results indicate that VC performs better than SF and the performance of VC is not improved by mixing with SF.

Instance 6: In the sixth and final instance we present we have three job types and three servers. The system is determined determined by the parameters $\lambda_1 = 4.00$, $\lambda_2 = 5.00$, $\lambda_3 = 3.00$, $\mu_{11} = 4.50$, $\mu_{12} = 3.00$, $\mu_{13} = 2.00$, $\mu_{21} = 3.00$, $\mu_{22} = 6.00$, $\mu_{23} = 4.00$, $\mu_{31} = 2.00$, $\mu_{32} = 3.00$ and $\mu_{33} = 4.00$. The static policy we utilize for mixing policies is the policy which assigns type i customers to server i for $i = 1, 2, 3$ which is the best deterministic static policy given the system parameters. This static policy will be denoted by DS123. The performance of the pure DS123 policy can be computed exactly by applying the Pollaczek-Khintchine formula for each queue separately. For type 1 jobs it follows that the average sojourn time is $V^1(DS123) = \frac{1}{\mu_{11} - \lambda_1} = \frac{1}{4.5 - 4.0} = 2.0$. Similarly it follows for the average sojourn time of type 2 jobs that $V^2(DS12) = \frac{1}{\mu_{22} - \lambda_2} = \frac{1}{6.0 - 5.0} = 1.0$ and

Fig. 14 Mixing VC and R70 for Instance 5

for type 3 jobs we have $V^3(DS123) = \frac{1}{\mu_{33} - \lambda_3} = \frac{1}{4.0 - 3.0} = 1.0$. Thus the overall average sojourn time for DS123 is $V(DS123) = \frac{2}{3} + \frac{5}{12} + \frac{1}{4} = \frac{4}{3} \approx 1.33$ which is in accordance with the following results which have been obtained by simulation. For $\mathcal{D} = \{DS123, SF\}$ a first simulation round has been performed with $\varepsilon = 0.10$. Then the simulated performance of the pure SF policy corresponding to $\theta = 0$ appears to be around 10 which is much worse than the simulated performance of the pure DS123 policy corresponding to $\theta = 1$. Moreover, the results indicate that by mixing SF and DS123 the best performing policies are obtained for values for θ around 0.8. Also around this optimal value of θ billiard-mixing seems to perform slightly better than Bernoulli-mixing. Therefore in the second round we have simulated the performance for $\theta = 0.60 + 0.05k$ for $k = 0, 1, \dots, 8$ with $\varepsilon = 0.05$. The results of this second simulation round are presented in Fig. 15.

Figure 15 is in accordance with the results of the first round. For both Bernoulli-mixing and billiard-mixing the optimal performance seems to be attained for θ around 0.85 with a corresponding performance around 1.0 which is a considerable improvement compared to the 1.33 of the pure DS123 policy. Moreover, again it is seen that around the optimal value of θ corresponding billiard-mixing policies perform slightly better than corresponding Bernoulli-mixing policies.

Next the optimal mixing is investigated for $\mathcal{D} = \{DS123, VC\}$ for which we expect to obtain even better performances than for $\mathcal{D} = \{DS123, SF\}$. For $\mathcal{D} = \{DS123, VC\}$ we have that $\theta = 0$ corresponds to applying the pure VC policy and $\theta = 1$ corresponds to applying the pure DS123 policy. A first simulation round has been performed with $\varepsilon = 0.10$ and the results indicate that by $\mathcal{D} = \{DS123, VC\}$ -mixing performances below 0.9 can be obtained. Moreover, the optimal value for θ seems to be around 0.5. In the second simulation round we zoom in to determine the optimal value of θ more precisely. In the second round

Fig. 15 Second round of mixing SF and DS123 for Instance 6

Fig. 16 Second round of mixing VC and DS123 for Instance 6

the performance values have been simulated for $\theta = 0.2 + 0.05k$ for $k = 0, 1, \dots, 12$ with $\varepsilon = 0.05$. The results of this second simulation round are presented in Fig. 16.

From Fig. 16 it seems that for billiard-mixing the optimal value of θ is around 0.5 while for Bernoulli-mixing it seems to be around 0.45. From Fig. 16 it is apparent that around the optimum billiard-mixing yields a better performances than Bernoulli-mixing. The optimal performance for billiard mixing with $\mathcal{D} = \{DS123, VC\}$ is apparently around 0.85 which is much better than the performance of 1.33 of the pure DS123 policy. Moreover, it is also a considerable improvement compared to the performance around 1.0 which was obtained by mixing the DS123 and SF policy.

To conclude the investigation of Instance 6 numerical results for $\mathcal{D} = \{SF, VC\}$ -mixing confirm as for previous instances that no improvement is obtained by $\mathcal{D} = \{SF, VC\}$ -mixing. Again it turns out that the pure VC policy performs better than any genuine mixing between the SF and VC policy.

6 Conclusions

The numerical results obtained by simulation for policy mixing for MJTAPSS instances with various traffic loads have been shown and interpreted. Static rules (DS12, R70 and DS123) and dynamic rules (SF and VC) have been utilized for policy mixing. We conclude that to improve on the performance of such rules it is in general a good idea to mix a static rule with dynamic rule(s). Since a mixture of static rules remains a static rule it is reasonable to use only one static rule for mixing, but this static rule should be chosen carefully. For

most of the investigated instances with two job types and two servers DS12 is the most obvious choice as static rule. Namely DS12 is a deterministic static rule which is very easy to implement and it corresponds to an extreme point of the space $\mathcal{H}$ of all static decision rules which is a good property for mixing with other rules. For the first four investigated instances DS12 is optimal or very close to optimal within the class of static decision rules. Thus for these instances the performance of DS12 is a good reference to evaluate the other performances obtained by policy mixing. For the fifth instance we have chosen R70 as static policy for that reason. Finally for the sixth instance with three job types and three servers we have chosen DS123 as static rule for the same reason as DS12 has been chosen for most instances with two job types and two servers.

As could be expected the numerical results confirm that applying the stationary policy induced by the dynamic myopic SF rule gives in general a bad performance especially in case of heavy traffic load. Except for Instance 4 where the traffic is rather light in all other instances the performance of the pure SF policy is worse than the performance of the selected static policy. For Instance 2 applying the pure SF policy does even result in an unstable system. However, by mixing the SF rule with the static rule rather good performances can be achieved. In each instance the performance of the best static rule could be substantially improved by mixing with an easy implementable dynamic rule. The improvement in performance compared to the best static policies can easily be more than 10% while the resulting mixing policies are still relatively easy to describe and implement. Besides the SF rule the VC rule which is somewhat more sophisticated but also easy to implement has been utilized for mixing. For the investigated instances the policy induced by the VC rule turns out to have a rather good performance which is in all investigated instances better than the static rule and the SF rule. According to the numerical results the VC rule can not be improved upon by mixing it with the SF rule. Any genuine mixing of these two rules give worse performance than the pure VC rule. A possible explanation for this is that both rules are dynamic for which the current state information is used in a rather similar way to assign the arriving job. In many situations the chosen server will be the same for these two rules which could explain why no improvement is obtained by mixing these two rules. On the other hand it turns out for most of the investigated instances the dynamic VC rule can be improved up by mixing it with the selected static rule. Moreover, in all cases by mixing the static and VC rule also a better performance is obtained than by mixing the static and the SF rule. Both the Bernoulli-mixing method and billiard-mixing method give good results, but it seems that in general the best billiard-mixing policies have a slightly better performance than the best Bernoulli-mixing policies. We conclude that for all the investigated MJTAPSS instances the best performance has been obtained by mixing the selected static rule and the VC rule according to the billiard-mixing method. The improvement in performance compared to the best static policies can be 15% or even more.

For future research also the mixing of more than two decision rules can be investigated. For example a static rule, the SF rule and the VC rule could all three be mixed simultaneously. However, in this case we have seen that the two dynamic rules SF and VC do not mix well. Therefore no further improvement can be expected by mixing all three rules while the optimization takes considerably more time than for two rules. Other possible extensions could be to utilize other dynamic decision rules for mixing or investigate MJTAPSS instances for which it is more complicated to choose the most suitable static rule for mixing with dynamic rules. Finally the Bernoulli-mixing method and billiard-mixing method can also be applied to other decision problems which are modeled as MDP. By the results in the current paper it seems a promising approach to mix a well-chosen static rule with easy implementable dynamic decision rule(s). It is possible that for other type of MDP

problems performances mixing multiple dynamic rules also gives a substantial improvement in performance.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

- Al-Azzoni I, Down D (2008) Linear programming-based affinity scheduling of independent tasks on heterogeneous computing systems. *IEEE Trans Parallel Distrib Syst* 19:1671–1682
- Altman E, Gaujal B, Hordijk A (2003) Discrete-event control of stochastic networks: multimodularity and regularity. lecture notes in mathematics. Springer, Berlin
- Altman E, Ayetsa U, Prabhu B (2011) Load balancing in processor sharing systems. *Telecommun Syst* 47:35–48
- Anselmi J, Casale G (2013) Heavy-traffic revenue maximization in parallel multiclass queues. *Perform Eval* 70:806–821
- Armony M (2005) Dynamic routing in large-scale service systems with heterogeneous servers. *Queueing Systems* 51:287–329
- Armony M, Ward A (2010) Fair dynamic routing in large-scale heterogeneous-server systems. *Oper Res* 58:624–637
- Arnoux P, Maudit P, Shiokawa I, Tamura J (1994) Complexity of sequences defined by billiards in the cube. *Bull Soc Math France* 122:1–12
- Baryshnikov Y (1995) Complexity of trajectories in rectangular billiards. *Commun Math Phys* 175:43–56
- Becker K, Gaver D, Glazebrook K, Jacobs P, Lawphongpanich S (2000) Allocation of tasks to specialized processors: a planning approach. *Eur J Oper Res* 126:80–88
- Bhulai S, Fahrenhorst-Yuan T, Heidergott B, van der Laan D (2012) Optimal balanced control for call centers. *Ann Oper Res* 201:39–62
- Borst S (1995) Optimal probabilistic allocation of customer types to servers. In: *Proceedings of the ACM sigmetrics conference on measurement and modeling of computer systems*, pp 116–125
- Chen H, Ye H (2012) Asymptotic optimality of balanced routing. *Oper Res* 60:163–179
- Feng H, Misra V, Rubenstein D (2005) Optimal state-free, size-aware dispatching for heterogeneous $m/g/-$ type systems. *Perform Eval* 62:475–492
- Glazebrook K, Nino-Mora J (2001) Parallel scheduling of multiclass $m/m/m$ queues: approximate and heavy-traffic optimization of achievable performance. *Oper Res* 49:609–623
- Hajek B (1985) Extremal splittings of point processes. *Math Oper Res* 10(4):543–556
- Heidergott B, Vázquez-Abad F, Pflug G, Fahrenhorst-Yuan T (2010) Gradient estimation for discrete-event systems by measure-valued differentiation. *Trans Model Comput Simul* 20:5.1–5.28
- Hordijk A, Koole G (1992) On the assignment of customers to parallel queues. *Probab Eng Inf Sci* 6:495–511
- Hyytiä E, Penttinen A, Aalto S (2012) Size- and state-aware dispatching problem with queue-specific job sizes. *Eur J Oper Res* 217:357–370
- Little J (2011) Little’s law as viewed on its 50th anniversary. *Oper Res* 59:536–549
- Lothaire M (2002) Algebraic combinatorics on words. Cambridge University Press, Cambridge
- Morse M, Hedlund G (1940) Symbolic dynamics ii - sturmian trajectories. *Am J Math* 62:1–42
- Serfozo R (1979) An equivalence between continuous and discrete time markov decision processes. *Oper Res* 27:616–620
- Sethumaran J, Squillante M (1999) Optimal stochastic scheduling in multiclass parallel queues. In: *Proceedings of the ACM sigmetrics conference on measurement and modeling of computer systems*, pp 93–102
- Stolyar A (2005) Optimal routing in output-queued flexible server systems. *Probab Eng Inf Sci* 19:141–189
- Stolyar A, Teczan T (2010) Control of systems with flexible multi-server pools: a shadow routing approach. *Queueing Systems* 66:1–151
- Tijms H (2003) A first course in stochastic models. Wiley, England
- van der Laan D (2011) Optimal mixing of markov decision rules for mdp control. *Probab Eng Inf Sci* 25:307–342

Dinard van der Laan received M.Sc. in mathematics from Leiden University, Leiden, the Netherlands in 1999, and Ph.D. in Mathematics from Leiden University, Leiden the Netherlands in 2003. In 2003-2004 he did research at the INRIA Rhône-Alpes research institute in Montbonnot, France. Since 2004 he is Assistant Professor at the department of Econometrics and Operations Research of VU University, Amsterdam, the Netherlands. His current research interests include Markov decision theory, reinforcement learning and dynamic pricing.