TI 2015-102/III
Tinbergen Institute Discussion Paper

Assigning Multiple Job Types to Parallel Specialized Servers

Dinard van der Laan

Faculty of Economics and Business Administration, VU University Amsterdam, and Tinbergen Institute, the Netherlands.

Tinbergen Institute is the graduate school and research institute in economics of Erasmus University Rotterdam, the University of Amsterdam and VU University Amsterdam.

More TI discussion papers can be downloaded at http://www.tinbergen.nl

Tinbergen Institute has two locations:

Tinbergen Institute Amsterdam
Gustav Mahlerplein 117
1082 MS Amsterdam
The Netherlands
Tel.: +31(0)20 525 1600

Tinbergen Institute Rotterdam
Burg. Oudlaan 50
3062 PA Rotterdam
The Netherlands
Tel.: +31(0)10 408 8900
Fax: +31(0)10 408 9031

Duisenberg school of finance is a collaboration of the Dutch financial sector and universities, with the ambition to support innovative research and offer top quality academic education in core areas of finance.

DSF research papers can be downloaded at: http://www.dsf.nl/

Duisenberg school of finance
Gustav Mahlerplein 117
1082 MS Amsterdam
The Netherlands
Tel.: +31(0)20 525 8579

Assigning multiple job types to parallel specialized servers

Dinard van der Laan *

August 18, 2015

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 which is modeled as continuous time Markov decision process. Performance optimization is difficult for this assignment problem, but optimization over the class of static policies is tractable. By applying the described mixing methods a suitable static decision rule is mixed with some dynamic decision rules which are easy to describe and implement. For the discussed mixing methods optimization is performed over corresponding classes of so-called mixing policies. These mixing policies are still easy to describe and implement and for all investigated instances the optimized mixing policies perform substantially better than optimal static policies. Moreover, the optimized mixing policies perform better than stationary dynamic policies which apply at decision epochs one of the dynamic rules to which the mixing methods have been applied.

Keywords: Job Assignment; Specialized Servers; Markov Decision Process; Mixing Decision Rules.

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 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 [8] and [3]). 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 can be mixed with some dynamic heuristic rule) it is possible to obtain policies which perform better and are as easy to implement than the best performing

*Tinbergen Institute, and Department of Econometrics and Operations Research, VU University, De Boelelaan 1105, 1081 HV Amsterdam, The Netherlands, Email: d.a.vander.laan@vu.nl

policies within classes which have been investigated as for example the class of static policies. In [24] for discrete-time MDP (DTMDP) with finite state space the mixing of decision rules is considered. Structural results on optimization by mixing methods are derived and the concept of mixing decision rules is illustrated for some examples with a small finite state space. 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. Such issues have been investigated in [24]. In the current paper the implementation and practical use of two so-called mixing methods are investigated for a problem which can be modeled as 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 and will be considered in detail. Subsequently policies generated by mixing methods are implemented in simulations and the approximated performances obtained by simulations will be compared with each other and with the exact calculable performances of static policies. The results are discussed and conclusions are drawn.

The problem for which in this paper mixing methods are investigated and compared with other methods with respect to performance optimization is described in [8]. In the current paper the problem will be referred to as "the multiple job type assignment problem with specialized servers" which will be abbreviated as MJTAPSS. It is a problem of assigning arriving (according to a given stochastic process) jobs (tasks) 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 [8]. 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 the expert(s) 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 the expert 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 occupance 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 [10] a problem resembling MJTAPSS is considered and optimization is performed over so-called static probabilistic policies. In [22] 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 [8] 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 [2] 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 [16]. Motivated by call center applications asymptotic optimality results for dynamic policies have been obtained especially by focusing on heavy traffic limits. For example in [13] 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 [4] and [5]. Moreover, in [11] asymptotic optimality results for assignment to parallel servers are obtained for the diffusion limit. 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 [12] and [17]. 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 seems to complicate the problem even more.

In Section 2 the system is specified including all system parameters. The performance objective is defined and the optimization problem is modeled as CTMDP. In Section 3 variety of Markovian decision rules and corresponding assignment policies for MJTAPSS are introduced. First the so-called static policies as applied in [8] 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 MJTTAPSS is transformed to a DTMDP after which mixing of decision rules as considered in [24] is generalized for the infinite state space DTMDP modeling MJTAPSS. In particular a randomized mixing method and a deterministic mixing method are discussed and compared. The static and dynamic assignment rules which have been introduced will be mixed to obtain new policies. An optimization over subclasses over such policies will be performed to obtain a better performance than the performance of any of the original policies. 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 after which conclusions are drawn from the obtained numerical results.

2 The model and performance objective

We consider a queueing system where different types $i = 1, 2, \ldots, M$ 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 $j = 1, 2, \ldots, N$. 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. Unless some other specification on the service times is made 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 $i = 1, 2, \ldots, M$ and $n = 1, 2, \ldots$ 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 $n = 1, 2, \ldots$ let W_n and S_n be respectively the 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 $i = 1, 2, \ldots, M$ and $t \geq 0$ let $N^i(t)$ 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 $i = 1, 2, \ldots, M$ 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 $\lim_{t\to\infty} \frac{1}{t} \sum_{n=1}^{t} V_n^i = V^i$ with probability one. Thus if for given policy ψ and $i \in \{1, 2, \ldots, M\}$ the Cesàro mean $\lim_{t\to\infty} \frac{1}{t} \sum_{n=1}^{t} V_n^i$ 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 $\lim_{t\to\infty} \frac{1}{t} \sum_{n=1}^{t} V_n = V$ with probability one. Thus if for given policy ψ the Cesàro mean $\lim_{t\to\infty} \frac{1}{t} \sum_{n=1}^{t} V_n$ 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 $i \in \{1, 2, \ldots, M\}$ 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*

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

Similarly for $i = 1, 2, \ldots, M$ 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 $\lim_{t\to\infty} \frac{1}{t} \int_{s=0}^{t} N^i(s)ds = L^i$ 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 $\lim_{t\to\infty} \frac{1}{t} \int_{s=0}^{t} L(s)ds = V$ with probability one.
By Little's Law (see [18]) it holds for $i = 1, 2, \ldots, M$ that $L^i(\psi)$ exists if and only if $V^i(\psi)$ exists. Moreover, in that case it holds that

$$L^i = \lambda_i V^i. \tag{2}$$

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

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

2.1 Performance Objective

In this paper the focus is on optimization of the performance of policies ψ to assign arriving jobs to the parallel servers. The objective is to minimize a (weighted) average of the long-run average sojourn times V^i , $i = 1, 2, \ldots, M$ of the different types of arriving jobs. By (2) it follows that this is equivalent to minimizing a weighted average (with adjusted weight factors) of the L^i. We assume that for the classes of policies which are considered in this paper the limits defining $V^i(\psi)$ and $L^i(\psi)$ will exist or be infinite. In other words it is claimed that almost surely the limit inferior equals the limit superior for the considered Cesàro means. Indeed the considered classes of policies will induce Markov chains satisfying conditions from which this follows. However, we note that the results can be extended to minimization over classes of policies for which this is not necessarily true by defining both $V^i(\psi)$ and $L^i(\psi)$ with the limit superior instead of the limit. In that case (2) still holds from which it follows that minimizing some weighted average of the V^i is still equivalent to minimizing some weighted average of the L^i by adjusting the weight factors according to (2).

Definition 2 *Let $\mathcal{F}$ be a class of policies for which $V^i(\psi)$ and thus $L^i(\psi)$ exists for all $i \in \{1, 2, \ldots, M\}$ and $\psi \in \mathcal{F}$. Moreover, let $w = (w_1, w_2, \ldots, w_M)$ with $w_i > 0$ for $i = 1, 2, \ldots, M$ be a given vector of weight factors. Then for $\psi \in \mathcal{F}$ we define*

$$f_w(\psi) := \sum_{i=1}^{M} w_i L^i(\psi) \tag{4}$$

as the performance of ψ with respect to w. For $\psi_1, \psi_2 \in \mathcal{F}$ we say that policy ψ_1 performs better than policy ψ_2 with respect to w if $f_w(\psi_1) < f_w(\psi_2)$. Moreover, we define $g(w) := \inf_{\psi \in \mathcal{F}} f_w(\psi)$ to be the optimal performance with respect to w and we say that policy $\psi \in \mathcal{F}$ is optimal with respect to w if $f_w(\psi) = g(w)$.

Remark. For $i = 1, 2, \ldots, M$ the weight factor w_i can be seen as the holding cost per time-unit per type i job in which case the minimization of $f_w(\psi)$ as defined by (4) comes down to minimizing the long-run average total holding costs for jobs present in the system. Moreover, unless specified else the minimization will be considered with respect to the particular weight vector w given by $w_i = 1$ for $i = 1, 2, \ldots, M$. To ease notation any reference to this standard $w = (1, 1, \ldots, 1)$ will be omitted. Note that for all $\psi \in \mathcal{F}$ it follows by (3) that

$$f(\psi) = L(\psi) = \lambda V(\psi). \tag{5}$$

In other words for the standard weight vector $w = (1, 1, \ldots, 1)$ the long-run average number of total jobs present in the system is minimized which is equivalent to minimizing the long-run average sojourn time over all arriving jobs.

2.2 Modeling

Now that we have defined the performance objective by (4) the multiple job type assignment problem with specialized servers (MJTAPSS) can be modeled as a continuous time Markov decision process (CTMDP). First we model MJTAPSS as a continuous time Markov decision process (CTMDP). Later in the paper a well-known uniformization method is utilized to transform the CTMDP to an equivalent discrete time Markov decision process (DTMDP).

For the CTMDP modeling first of all an appropriate state space S should be described. For this a state $s \in S$ should describe the situation for each queue $j \in \{1, 2, \ldots, N\}$. That is for each queue j the state description should specify all the jobs which have been assigned to server j and are still in the system and

moreover, it should be specified in which order these jobs will be served by server j because of the FCFS queueing discipline. Let K be the free monoid generated by the set of job types $\{1, 2 \ldots, M\}$. It is clear that for each queue j the situation at an decision epoch can be represented as a (possibly empty) sequence of jobs (including the type) to be served in order by server j. The space of all such sequences corresponds one to one to K. Therefore let $y_j \in K$ correspond to the current situation for queue j at an decision epoch. Then for $j = 1, 2, \ldots, N$, y_j will be a component of the current state $s \in S$. Moreover, for actual decision epochs the current state s should also specify the type z of the currently arriving job which has to be assigned to one of the queues. Therefore for actual decision epochs we have that $z \in \{1, 2, \ldots, M\}$. For convenience we include for now in the model also states describing situations at so-called virtual decision epochs. At such a moment there is some occurrence, but an actual decision is not made. For the MJTAPSS this happens at the moment when the service of some job was just completed and thus that job is no longer in the system. For these virtual decision epochs occurring at departures we will specify that $z = 0$ which means that there is no arriving job and thus no actual decision is made at these epochs. To conclude with this modeling the state space S is given by

$$S = \{(y_1, y_2, \ldots, y_N, z) : y_j \in K \text{ for } j = 1, 2, \ldots, N \text{ and } z \in \{0, 1, \ldots, M\}\}.$$

We note that S is an infinite countable state space which is easily seen from the well-known fact that the free monoid K is infinite and countable.

For $s = (y_1, y_2, \ldots, y_N, z) \in S$ with $z \geq 1$ the action space is given by $A(s) = \{1, 2, \ldots, N\}$. On the other hand in case of a virtual decision epoch ($z = 0$) there is no actual choice of action which is modeled by $A(s) = \{0\}$. To complete the CTMDP modeling for all $s, t \in S$ and $a \in A(s)$ the expected transition times $\tau_s(a)$, the expected direct costs $c_s(a)$ and the transition probabilities $p_{st}(a)$ still should be specified. The specification of all these elements is straightforward but somewhat cumbersome. Now the most important aspects of this specification are clarified.

Note that $\tau_s(a)$, the expected time between the current and the next decision epoch, will depend on the type of job that each server $j \in \{1, 2, \ldots, N\}$ will serve (or will be idle) from the current decision epoch until the next decision epoch. Note that by including the moments of departure of jobs from the system as virtual decision epochs of the CTMDP for each server the (type of) job being served is fixed in any period between consecutive decision epochs. Therefore for given $s \in S$, $a \in A(s)$ we may for each server server j define $u_j = u_j(s, a) \in \{0, 1, \ldots, M\}$ as the type of job being served by server j between the current and the next decision epoch, with the stipulation that $u_j = 0$ if server j will be idle during that period. Then the $u_j(s, a)$ for $j = 1, 2, \ldots, N$ are well defined and are determined by s and a. Moreover, it holds for all $s \in S$ and $a \in A(s)$ that

$$\tau_s(a) = (\lambda + \sum_{j:u_j>0} \mu_{u_j j})^{-1}. \tag{6}$$

For $s, t \in S$ and $a \in A(s)$ the CTMDP modeling with virtual decision epochs makes it also straightforward to determine transition probabilities $p_{st}(a)$, but we do not need now to explicitize these transition probabilities. Therefore we conclude the CTMDP modeling by determining explicitly for $s \in S$ and $a \in A(s)$ the incurred costs $c_s(a)$, between the current and the next decision epoch. Note that from the inclusion of virtual decision epochs it also follows that in the period between decision epochs the set of jobs which are present in the system is fixed. Therefore for given state $s \in S$ we may for each type i define $q_i = q_i(s)$ as the number of type i jobs present in the system between the current decision epoch and the next decision epoch. Then the

$q_i(s)$ for $i = 1, 2, \ldots, M$ are well defined and are determined by the state s at the current decision epoch. Thus the q_i can be considered as partial state information and this information is sufficient to determine the incurred costs. Indeed for any weight vector $w = (w_1, w_2, \ldots, w_M)$ as in Definition 2 it holds for all $s \in S$ and $a \in A(s)$ that the incurred costs $c_s(a)$ between the current and the next decision epoch are given by

$$c_s(a) = \tau_s(a) \sum_{i=1}^{M} w_i q_i, \tag{7}$$

where $\tau_s(a)$ is determined by (6).

For the problem specified in this section with multiple types of jobs and service time distributions which may depend both on the job type i and on the queue j it is in general difficult to obtain the optimal performance and/or an optimal assignment policy. The state space S 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 the most common methods and heuristics by which easily implementable (but in general suboptimal) policies can be obtained. After that we introduce a more sophisticated heuristic by which usually a rather good performing policy is obtained which is still 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 the most commonly applied dynamic policies for MJTAPSS.

3 Implementation and optimization of Markovian policies

We have seen that the problem of minimizing the performance $f_w(\pi)$ over assignment policies π can be modeled as a CTMDP. To optimize the performance we consider Markovian policies π which can be represented as an infinite sequence $\pi = (d_1, d_2, \ldots)$ where d_n, $n = 1, 2, \ldots$ 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 specified in the previous section 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 at virtual decision epochs not matter which decision rule is applied. 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 $\pi = (d_1, d_2, \ldots)$ 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.
We briefly recall some basic terminology for Markovian policies and decision rules. A decision rule is Markovian if only the current system state $s \in S$ at the decision epoch is used as input for the choosing an action $a \in A(s)$. 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 $d : S \to \mathcal{P}(A)$. A Markovian policy $\pi = (d_1, d_2, \ldots)$ is called stationary if for some decision rule d it holds that $d_n = d$ for all $n \in N$. Thus a stationary Markovian policy is determined by a mapping $d : S \to \mathcal{P}(A)$ which is applied at every (actual) decision epoch.

Note that for any actual decision epoch the applied decision rule $d : S \to P(A)$ is determined by specifying $d(s)$ for states $s = (y_1, y_2, \ldots, y_M, z) \in S$ with $z \geq 1$. For states s with $z = 0$ the $d(s)$ could

be formally specified, but such a specification for states with $z = 0$ is irrelevant for performance and implementation. Therefore in the sequel we identify Markovian decision rules d as mappings with domain $S^1 = \{(y_1, y_2, \ldots, y_M, z) \in S : z \geq 1\}$ instead of domain S.
For all $s \in S^1$ it holds that $A(s) = \{1, 2, \ldots, N\}$ and thus for the common finite action space $A = \{1, 2, \ldots, N\}$ we have that any decision rule d can be described as a mapping from S^1 to the standard $N-1$-dimensional simplex $X := \Delta^{N-1} = \{(x_1, x_2, \ldots, x_N) : \sum_{j \in A} x_j = 1, x_j \geq 0 \ \forall j \in A\}$ 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 to the cooresponding stationary Markovian policy $(d, d, \ldots)$.
If for a decision rule $d : S^1 \to X$ it holds that for all $s \in S^1$ there exists some $j(s) \in A$ such that $d(s)$ is the unit vector $e_{j(s)}$, then d is a so-called deterministic Markovian decision rule. Note that deterministic Markovian decision rules could also 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 $d : S^1 \to X$. 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. Such subclasses of decision rules and Markovian policies using only some partial state information are described in the following subsections.

3.1 Implementation and optimization of static policies

Considering that in the interest of implementation it is better to use a minimal amount of state information consider the following class of Markovian policies. Let $s = (y_1, y_2, \ldots, y_M, z) \in S$ be the state at an decision epoch. Then in addition to stationarity for the policies in this class 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. As in [8] this will be called the class of static policies. In [8] the optimization over this class of so-called static policies is investigated for MJTAPSS. In the sequel we call both a decision rule d and the corresponding stationary policy $(d, d, \ldots)$ static if the choice of action only depends on the z component of state s. Unlike as in [8] in the current paper also some dynamic (non-static) decision rules and corresponding policies will be considered for the optimization of MJTAPSS, but the performance of the optimal static policy is a good reference to assess 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 that dynamic rule with a well-chosen static rule. We start with the following definition of static decision rules and corresponding static Markovian policies.

Definition 3 *A decision rule $d : S \to \mathcal{P}(A)$ is so-called* **static** *if for all $s_1 = (y_1^1, y_2^1, \ldots, y_N^1, z^1) \in S$ and $s_2 = (y_1^2, y_2^2, \ldots, y_N^2, z^2) \in S$ it holds that $d(s_1) = d(s_2)$ if $z^1 = z^2$.*
A stationary Markovian policy π is called static if $\pi = (d, d, \ldots)$ for some static decision rule d.

Recall that for the given CTMDP model of MJTAPSS at actual decision epochs the common action space for $s \in S_1$ is $A = \{1, 2, \ldots, N\}$ and thus $\mathcal{P}(A)$ is the $N-1$-dimensional standard simplex $\Delta^{N-1} =$

$\{(x_1, x_2, \ldots, x_N) : \sum_{j \in A} x_j = 1, x_j \geq 0 \ \forall j \in A\}$. From this it follows that any static decision rule d can be 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 also clear that static decision rules are easy to implement. If we denote with $\mathcal{G}$ the class of Markovian stationary policies $\pi = (d, d, \ldots)$ determined by a static decision rule d then it follows that any policy $\pi \in \mathcal{G}$ is also uniquely represented by such a matrix $R = (r_{ij})$ of assignment probabilities. In other words there is a one-to-one correspondence between $\mathcal{G}$ and the space $\mathcal{H}$ of nonnegative $M \times N$ matrices with row sums equal to one which is a convex and compact subspace of $\mathbb{R}^{MN}$.

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

Definition 4 *A static decision rule d is a so-called 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 $i \in \{1, 2, \ldots, M\}$ and $j \in \{1, 2, \ldots, N\}$. A stationary Markovian policy π is called deterministic static if $\pi = (d, d, \ldots)$ for some deterministic static decision rule d.*

We note that there exist exactly MN different deterministic static decision rules and corresponding $M \times N$ matrices. Moreover, is is obvious that $\mathcal{H}$ is the convex hull of the MN different matrices representing deterministic static rules.

For more details we refer to [8], but we now summarize the key results for application and optimization of static policies adapted to the notation and objectives of the current paper.
For static policy $\psi \in \mathcal{G}$ let $R = (r_{ij})$ be the unique matrix of assignment probabilities representing policy ψ. By (4) the performance of policy $\psi = \psi(R)$ is defined as $f_w(\psi(R)) = \sum_{i=1}^{M} w_i L^i(\psi(R))$. The key observation to minimize this performance measure $\sum_{i=1}^{M} w_i L^i(\psi(R))$ is that for static policies all queues $j \in \{1, 2, \ldots, N\}$ 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 [23]) 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 Program (MP) to optimize the performance over all static policies.
To apply the Pollaczek-Khintchine formula for $j \in \{1, 2, \ldots, M\}$ it is sufficient to determine for queue j the aggregated arrival rate λ_j, the traffic intensity $\rho_j := \frac{\lambda_j}{\mathrm{E}[S_j]}$ and the second moment $\mathrm{E}[(S_j)^2]$ where S_j is the service time distribution for jobs assigned to 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 [8]) for $j = 1, 2, \ldots, N$ that

$$\lambda_j = \sum_{i=1}^{M} \lambda_i r_{ij}, \tag{8}$$

$$\rho_j = \sum_{i=1}^{M} \lambda_i r_{ij} \mathrm{E}[S^{ij}] = \sum_{i=1}^{M} \frac{\lambda_i r_{ij}}{\mu_{ij}} \tag{9}$$

and

$$\mathrm{E}[(S_j)^2] = (1/\lambda_j)\sum_{i=1}^{M} \lambda_i r_{ij} \mathrm{E}[(S^{ij})^2] = \frac{2\sum_{i=1}^{M} \frac{\lambda_i r_{ij}}{(\mu_{ij})^2}}{\sum_{i=1}^{M} \lambda_i r_{ij}}. \tag{10}$$

Let $W(j)$ be the mean waiting time of all jobs assigned to queue j if static policy $\psi(R)$ is applied. By (8), (9), (10) and the Pollaczek-Khintchine formula it follows that

$$W(j) = \frac{\lambda_j \mathrm{E}[(S_j)^2]}{2(1-\rho_j)} = \frac{\sum_{i=1}^{M} \frac{\lambda_i r_{ij}}{(\mu_{ij})^2}}{1 - \sum_{i=1}^{M} \frac{\lambda_i r_{ij}}{\mu_{ij}}} \text{ if } \rho_j < 1, \tag{11}$$

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 (11) it follows for all $i \in \{1, 2, \ldots, M\}$ that $V^i(\psi(R))$ is finite if $\rho_j < 1$ for all $j \in \{1, 2, \ldots, N\}$. In that case it follows for $i = 1, 2, \ldots, M$ that

$$V^i(\psi(R)) = \sum_{j=1}^{N} r_{ij}(W(j) + \mathrm{E}[S^{ij}]) = \sum_{j=1}^{N} \left(\frac{r_{ij} \sum_{k=1}^{M} \frac{\lambda_k r_{kj}}{(\mu_{kj})^2}}{1 - \sum_{k=1}^{M} \frac{\lambda_k r_{kj}}{\mu_{kj}}} + \frac{r_{ij}}{\mu_{ij}}\right) \tag{12}$$

and thus by Little's law it follows that

$$L^i(\psi(R)) = \lambda_i V^i(\psi(R)) = \lambda_i \sum_{j=1}^{N} \left(\frac{r_{ij} \sum_{k=1}^{M} \frac{\lambda_k r_{kj}}{(\mu_{kj})^2}}{1 - \sum_{k=1}^{M} \frac{\lambda_k r_{kj}}{\mu_{kj}}} + \frac{r_{ij}}{\mu_{ij}}\right) \tag{13}$$

Let $\mathcal{H}' = \{R \in \mathcal{H} : \rho_j < 1 \text{ for } j = 1, 2, \ldots, N\}$ where ρ_j is determined by (9) for $j = 1, 2, \ldots, N$. If $\mathcal{H}'$ is nonempty and let $L^i(\psi(R))$ be determined by (13) for $i = 1, 2, \ldots, M$ it follows that an optimal solution R^* of the mathematical program

$$\min_{R \in \mathcal{H}'} f_w(\psi(R)) = \sum_{i=1}^{M} w_i L^i(\psi(R)) \tag{14}$$

yields an optimal static policy ψ by $\psi = \psi(R^*)$. It also follows (see [8]) that if $\mathcal{H}'$ is nonempty that then an optimal solution of (14) and thus a corresponding optimal static policy exists. Namely in that case replacing the search space $\mathcal{H}'$ by $\{R \in \mathcal{H} : \rho_j \leq 1 - \epsilon \text{ for } j = 1, 2, \ldots, N\}$ for some $\epsilon > 0$ chosen small enough does not affect the optimal solution. In this way (14) 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 that for this kind of mathematical program (see [22]) 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 $\pi = (d, d, \ldots)$ 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 an 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 $i = 1, 2, \ldots, M$, $j = 1, 2, \ldots, N$ determine q_{ij}, the number of type i jobs present (including jobs being served) in the queue for server j.
Next determine $j' \in \{1, 2, \ldots, N\}$ as the minimal number of a server such that $s_{kj'} = \min_j s_{kj}$ where for $j = 1, 2, \ldots, N$,

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

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 $\pi = (d, d, \ldots)$ is applied.

3.3 The virtual cost policy

Simulation will confirm that the myopic SF policy can perform very poor especially in case of heavy traffic. In this subsection we introduce another dynamic policy which especially in case of heavy traffic is expected to 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 $\pi = (d, d, \ldots)$ will be as straightforward as for the SF policy. It turns out 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 $i = 1, 2, \ldots, M$, $j = 1, 2, \ldots, N$ 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 $j = 1, 2, \ldots, N$

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

Next determine $j' \in \{1, 2, \ldots, N\}$ 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 $\pi = (d, d, \ldots)$ is applied.

Remark. Note that in case of $M = 1$ (only one type of job) it holds that $s_{kj} = u_{kj}$ for $j = 1, 2, \ldots, N$. 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. Namely the number of future assignments to queue j for which the sojourn time is increased by μ_{kj} because this virtual type k job first has to be served 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.

Moreover, it seems intuitively that the VC rule considers the cost of the assignment for future arrivals while the SF rule at first 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 provided later in this paper) 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 in general 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 so-called mixing of the three types of decision rules it is often possible to obtain a performance which clearly improves on the best performance among all stationary Markovian policies which apply the same decision rule (static, SF or VC) at all decision epochs. For DTMDP a method to improve on a given set of decision rules by mixing them was investigated in [24]. 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 [24] 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 mixing policy which is applied. We would like like to apply these conditions to MJTAPSS. We note that MJTAPSS is modeled as a CTMDP while results in [24] to which sometimes is referred are obtained for general DTMDP. However, by a well-known uniformization technique (see for example [21]) the CTMDP corresponding to the MJTAPSS can be transformed to an equivalent DTMDP. Therefore we will utilize this uniformization of the CTMDP modeling MJTAPSS to a DTMDP to describe the policies which can be obtained by a so-called mixing of the three (static, SF and VC) types of decision rules which were described in the previous section.

4.1 Uniformization

It is clear that for the MJTAPSS all state transition rates (corresponding to either arrival of jobs or service completions) are bounded and by choosing for example an uniform jump time $\tau = (\sum_{i=1}^{M} \lambda_i + \sum_{j=1}^{N} \max_{i\in\{1,2,\ldots,M\}} \mu_{ij})^{-1}$ the CTDMP modeling MJTAPSS can be uniformized to an equivalent DTMDP. The transformed transition probabilities $\overline{p}_{st}(a)$ of the DTMDP are given by (see for example [23])

$$\overline{p}_{st}(a) = \begin{cases} (\tau/\tau_s(a))p_{st}(a) & \text{voor } t \neq s, \\ (\tau/\tau_s(a))p_{st}(a) + [1 - (\tau/\tau_s(a))] & \text{voor } t = s. \end{cases}$$

Moreover, the transformed expected direct costs $\overline{c}_s(a)$ are given by

$$\overline{c}_s(a) = \frac{c_s(a)}{\tau_s(a)}. \tag{15}$$

From (7) and (15) it follows for the equivalent DTMDP that the direct costs $\overline{c}_s(a)$ are given by

$$\overline{c}_s(a) = \sum_{i=1}^{M} w_i q_i \tag{16}$$

, where $q_i = q_i(s)$ is the number of type i customers which will be present in the system between the current and the next decision epoch. Thus for the standard weight vector $w = (1, 1, \ldots, 1)$ we have that $\overline{c}_s(a) = q$, where $q = q(s) := \sum_{i=1}^{M} q_i(s)$, is the total number of jobs present in the system between the current and next decision epoch.

4.2 Mixing of decision rules

In [24] a description is given of so-called mixing of Markovian decision rules for DTMDP with a finite state space. In this subsection we recall some definitions and concepts which are used in the sequel of this paper. In addition to [24] adjustments are made to deal with the infinite countable state space to model MJTAPSS in the present paper.

The main idea of so-called 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 $\mathcal{D} = \{d^1, d^2, \ldots, d^k\}$ 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 $S \to \mathcal{P}(A)$ representing so-called $\mathcal{D}$-mixing rules is generated. Definition 5 shows how convex combinations of decision rules are constructed to generate the space $\mathcal{F}_D$.

Definition 5 *Let $\mathcal{D} = \{d^1, d^2, \ldots, d^k\}$ be a given set of k Markovian decision rules with common state space S and action space A and let*

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

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

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

$$d_\theta(s) := \sum_{l=1}^{k} \theta_l d^l(s) \text{ for all } s \in S. \tag{17}$$

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

$$\mathcal{F}_D := \cup_{\theta \in \Theta_k} d_\theta. \tag{18}$$

Remark 6 *Since $\mathcal{P}(A)$ is convex and according to (17) $d_\theta(s)$ 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 17 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 $d^l \in \mathcal{D}$. Namely at any decision epoch d_θ may be implemented by first choosing $d^l \in \mathcal{D}$ with probability θ_l for $l = 1, 2, \ldots, k$ and then implementing d^l.

Since for DTMDP modeling MJTAPSS the state space S is infinite countable we have that decision rules $d \in \mathcal{D}$ induce some infinite transition probability matrix P, i.e. any decision rule induces 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 $\mathcal{D} = \{d^1, d^2, \ldots, d^k\}$ let P^l, $l = 1, 2, \ldots, k$ denote the infinite transition matrix induced by d^l. Then we have the following lemma.

Lemma 7 *Let P_θ be the infinite transition matrix induced by d_θ for $\theta \in \Theta_k$. Then it holds that*

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

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

Definition 8 *Let $\mathcal{D}$ be given and $\mathcal{F}_D$ be defined by (18). Then for all $\theta \in \Theta_k$, $d_\theta \in \mathcal{F}_D$ let $\pi_\theta = (d_\theta, d_\theta, \ldots)$ 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*

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

The performance can be optimized over the class $\mathcal{G}_D$. The idea is that in this way policies can be obtained with substantially better performance than any of the stationary policies for which some decision rule $d \in \mathcal{D}$ is applied. Such performance improvement is most interesting if $\mathcal{D}$ consists of a (small) selection of reasonable decision rules. This selection can for example consist of some of the rules introduced in subsections 3.1, 3.2 and 3.3.

The performance of policies $\pi_\theta \in \mathcal{G}_D$ will be approximated by simulation of such stationary $\mathcal{D}$-mixing policies by implementing these policies as described in Remark 6. If k is not too large it turns out to be tractable to optimize in this manner 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 [24] such optimization utilizing exact performance computations is illustrated for problems with finite (and rather small) state and action spaces, but this method is 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 performance of $\pi_\theta \in \mathcal{G}_D$ exact by analytical methods even for the complicated MDP modeling MJTAPSS. We note that this is for example 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 $l = 1, 2, \ldots, k$. 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 Subsection 3.1 that for static decision rules d it is possible to compute exactly the performance $f_w(\psi)$ (as defined by (4)) of the stationary policy $\psi = (d, d, \ldots, d)$. Thus indeed exact computations of the performances of $\pi_\theta \in \mathcal{G}_D$ are possible if $\mathcal{D}$ consists of only static decision rules.

Remark 9 *It is clear that if $\mathcal{D}$ consists of all the $k = MN$ different deterministic static decision rules that then the class $\mathcal{G}_D$ of $\mathcal{D}$-mixing policies consists of all stationary static Markovian policies as described in subsection 3.1. Hence the optimization over all static policies as investigated in [8] could be regarded as a special case of optimization over stationary $\mathcal{D}$-mixing policies with a particular choice of $\mathcal{D}$.*

In the current paper we will focus on $\mathcal{D}$ consisting of only a very few (in particular $\mathcal{D}$ with $k = 2$ will be considered) different decision rules. The idea is that despite a small value of k good performances can be obtained if the k different decision rules have "good mixing" properties. In particular the following choice of $\mathcal{D}$ with $k = 2$ turns out to give good results. Choose a (deterministic) static decision rule as the first rule and choose either the SF rule or the VC rule as the second decision rule for mixing with the (deterministic) static rule. On the other hand the simulations will indicate that the performance of the VC rule can in general not be improved by mixing it with the SF rule.

Apart from improvements in performance the advantage of having a small number k of different decision rules in $\mathcal{D}$ is that the search space Θ_k is relatively small and therefore the optimization over Θ_k is tractable and can be done rather fast. The disadvantage compared to optimizing over (all) static policies is that for non-static policies it is in general not possible to compute the performance with analytical methods and then we have to rely on simulation to approximate the performance of such policies.

4.3 Non-stationary mixing of decision rules

Recall (18) the definition of the space $\mathcal{F}_D$ of $\mathcal{D}$-mixing rules and (Definition 8) 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 all (possibly non-stationary) $\mathcal{D}$-mixing policies and a subclass $\mathcal{L}_D$ of $\mathcal{K}_D$.

Definition 10 *The class $\mathcal{K}_D$ consists of all Markovian policies $\pi = (d_1, d_2, \ldots)$ for which $d_t \in \mathcal{F}_D$ for $t = 1, 2, \ldots$.*
The subclass $\mathcal{L}_D \subseteq \mathcal{K}_D$ consists of all $\pi = (d_1, d_2, \ldots) \in \mathcal{K}_D$ for which $d_t \in \mathcal{D}$ for $t = 1, 2, \ldots$.

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 solutions for MJTAPSS we searched for Markovian policies which are easy to implement. Thus most policies in $\mathcal{K}_D$ and $\mathcal{L}_D$ are not acceptable for the assignment of arriving jobs to the servers. On the other hand it is in general possible (in [24] an explicit example is given) that $\mathcal{L}_(D)$ (and thus also $\mathcal{K}_D$) contains policies which perform better than the best performing policy in $\mathcal{G}_D$.

Therefore for MJTAPSS with given $\mathcal{D}$ as described in Remark 9 we search for subclass(es) of $\mathcal{K}_D$ with the following properties.

Criterion 11

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 $\mathcal{G}_D$.*

Subclasses which at first could be considered to satisfy Criterion 11 consist of periodic policies in $\mathcal{K}_D$ with some upper limit on the period of the policy. A Markovian policy $(d_1, d_2, \ldots) \in \mathcal{K}_D$ 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 $\mathcal{G}_D = \mathcal{G}_D^1 \subseteq \mathcal{G}_D^2 \subseteq \mathcal{G}_D^3 \subseteq \ldots$ and the question is whether Criterion 11 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 only tractable 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 11 the maximal period p_0 should be quite small. On the other hand for smaller p_0 it is in general not likely that point 3 of Criterion 11 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 no guarantees ca be given in general.
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 no useful guarantees whether Criterion 11 will be satisfied.

4.3.1 Billiard sequences and corresponding policies

Apart from periodic policies with a bound on the period there is another subclass of $\mathcal{L}_D$ which is more promising to satisfy all aspects of Criterion 11. This is the subclass of policies $(d_1, d_2, \ldots) \in \mathcal{K}_D$ which can be obtained as a so-called billiard sequence (see for example [6], [7]). This subclass will be denoted by $\mathcal{B}_D$.

Definition 12 *For given $\mathcal{D} = \{d^1, d^2, \ldots, d^k\}$ billiard sequences and corresponding policies in $\mathcal{B}_D$ are obtained as follows. Consider the k-dimensional unit cube where orthogonal faces of the cube have been given different labels in $\{1, 2, \ldots, k\}$ while parallel faces have the same label in $\{1, 2, \ldots, k\}$. Then consider the infinite sequence $(w_1, w_2, \ldots)$ which is obtained by following the trajectory of a (billiard) point moving with (constant) nonzero speed and $w_n \in \{1, 2, \ldots, k\}$ is the label of the face on which the trajectory has its n -th reflection. A priori it must be specified in which order the labels are sequenced if the trajectory hits multiple sides simultaneously or alternatively it may be assumed that the initial position and direction are such that it never hits more than one face simultaneously. Either way it follows that the generated billiard sequence depends then only on the initial position and direction of the moving point. An infinite sequence generated in this way is called a k-dimensional billiard sequence. Then any k-dimensional billiard sequence $(w_1, w_2, \ldots)$ defines a corresponding policy $\pi = (d_1, d_2, \ldots) \in \mathcal{L}_D$ by putting $d_t = d^{w_t}$ for $t = 1, 2, \ldots$.*
Finally we define $\mathcal{B}_D \subseteq \mathcal{L}_D$ as the subset of $\mathcal{L}_D$ of policies which correspond to k-dimensional billiard sequences.

Remark 13 *Given $\mathcal{D}$ the mixing of decision rules by applying policies in $\mathcal{B}_D$ is called billiard-mixing while the mixing of decision rules by applying stationary policies $\pi_\theta \in \mathcal{G}_D$ is called Bernoulli-mixing.*

Now that the subclass $\mathcal{B}_D$ is defined we consider whether the three issues formulated in Criterion 11 hold for $\mathcal{B}_D$. The first issue is whether performance optimization over the whole 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 by obtaining an approximation with prescribed precision of the optimal performance.
This follows from the following explanation which shows that optimization over $\mathcal{B}_D$ is (under some mild assumptions which are always satisfied for the problem considered in this paper) similar to optimization over $\mathcal{G}_D$ (recall Definition 8) which is an optimization of a continuous function over the compact and convex $(k-1)$-dimensional simplex Θ_k. Proposition 16 shows that (at least for $k=2$) the (initial) direction of the trajectory is of utter importance to characterize a billiard sequence.Therefore we specify now the meaning of the direction of the trajectory of a billiard sequence.

It is clear that the initial direction at time $t=0$ can be described by some nonzero vector $v=(v_1,v_2,\ldots,v_k)\in\mathbb{R}^k$. Moreover, because of the reflections occurring in a billiard trajectory for all components $l\in\{1,2,\ldots,k\}$ only the relative size of the absolute value of v_l compared to the other components of v in essence determines the direction of the trajectory. Therefore for given nonzero vector $v=(v_1,v_2,\ldots,v_k)\in\mathbb{R}^k$ the direction vector is normalized by putting

$$\theta_l := \frac{|v_l|}{\sum_{i=1}^{k}|v_i|} \text{ for } l=1,2,\ldots,k. \tag{19}$$

Remark 14 *For any nonzero vector v we have for the corresponding normalized direction vector $\theta=(\theta_1,\theta_2,\ldots,\theta_k)$ determined by (19) that $\theta\in\Theta_k$ where Θ_k is as in Definition 5. Vice versa for any $\theta\in\Theta_k$ there exists some nonzero $v\in\mathbb{R}^k$ such that the vector determined by (19) is θ. For example for $v=\theta$.*

Next we define the (essential) direction of the trajectory of a billiard sequence as follows.

Definition 15 *The (essential) direction of the trajectory of a k-dimensional billiard sequence is determined by some $\theta\in\Theta_k$. For any k-dimensional billiard sequence the essential direction of the trajectory is obtained by (19) where v is the nonzero initial direction vector at time $t=0$ which together with some initial point determines the trajectory which generates the sequence according to Definition 12.*
We say that trajectories generating k-dimensional billiard sequences are parallel to each other if they have the same (essential) direction $\theta\in\Theta_k$.

Consider now the case of $k=2$ which is the most simple and arguably also the most important case. For $k=2$ billiard sequences as defined by Definition 12 are generated by playing billiards on a square table. Apart from playing billiards there are for $k=2$ other constructions and characteristic properties of such sequences and therefore these sequences are also known under other notions (see for example [20] and [19]) from which regular sequences and Sturmian words (or sequences) are the most commonly used notions. Non-periodic Sturmian words are known for the property to have minimal complexity among all non-periodic words where the complexity function $c(n)$, $n\in\mathbb{N}$ of an infinite sequence is defined by the number of different subsequences of length n. With respect to the finite subsequences of $k=2$-dimensional billiard sequences another related, well-known and important property is the following.

Proposition 16 *In case of $k=2$ for billiard sequences the set of all finite subsequences is completely determined by the (essential) direction $\theta\in\Theta_2$ of the billiard trajectory and thus the finite subsequences of the billiard sequence do not depend on the initial point within the unit square from which the trajectory is generated.*

In other words according to Proposition 16 all $k=2$ billiard sequences for which the trajectories are parallel to each other are essentially the same since these trajectories generate the same subsequences. This is a very useful property if it implies that policies corresponding to parallel billiard sequences have the same

performance. Namely in that case the optimization over $\mathcal{B}_D$ can be done by an optimization over only the (essential) direction since the initial position would not influence the performance of the generated policy. In [24] it was investigated whether policies in $\mathcal{L}_D$ for which the corresponding infinite sequences have the same set of finite subsequences indeed have the same performance. Some counterexamples were given, but also sufficient conditions for this property have been established. In the counterexamples given in [24] some policy in $\mathcal{L}_D$ is applied for which the performance does depend on the initial system state of the MDP. Moreover, from [24] it follows that for the property to hold it is sufficient that for the considered policies in $\mathcal{L}_D$ the performance does not depend on the initial system state. In [24] in general the independence of performance of the initial system state could be established for all policies in $\mathcal{L}_D$ under some conditions on the transition matrices associated with the decision rules $d^l \in \mathcal{D}$. However, for the particular MDP associated with MJTAPSS the fact that the performance of policies in $\mathcal{L}_D$ does not depend the initial system state follows directly from the characteristics of MJTAPSS and it is not necessary to consider $\mathcal{D}$ to establish this. Namely suppose that the considered policy ψ in $\mathcal{L}_D$ is such that the system is stable if the policy is applied. A stable system means that for any initial system state that with probability one all the queues will empty at some point. Then it follows by standard arguments that the performance of this policy ψ given by Definition 2 does not depend on the initial system state. On the other hand if the policy is such that the system is not stable then there is at least one queue for which the number jobs which are present in that queue grows to infinity if the time $t \to \infty$. Then there is also at least time one type of jobs for which the total number that is present in the system grows to infinity and thus there exist $i \in \{1, 2, \ldots, M$ with $L^i(\psi) = \infty$. Hence for the performance of such a policy we have that $f_w(\psi) = \infty$ independent of the initial state of the system. We note that the independence of the initial system state of the performances of Markov policies applied to MJTAPSS was implied earlier, but now it is explicitly established. Combining this with Proposition 16 we have the following result.

Proposition 17 *In case of $k = 2$ the performance of a policy in $\mathcal{B}_D$ is determined by the (essential) direction $\theta \in \Theta_2$ of the trajectory 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 $\Theta_2 = \{(\theta_1, \theta_2) : \theta_1 \geq 0, \theta_2 \geq 0, \theta_1 + \theta_2 = 1\}$.*

From Proposition 17 it is clear that at least for $k = 2$ a full optimization over $\mathcal{B}_D$ is tractable. Note that the search space Θ_2 for the optimization is the same (recall Definition 8) as for optimization over the class $\mathcal{G}_D$ of stationary $\mathcal{D}$-mixing policies. However, it should be noted that for the classes $\mathcal{B}_D$ and $\mathcal{G}_D$ in some cases the applicable methods to optimize over the associated search space Θ_2 are 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 the exact performance as function of $\theta \in \Theta_2$ by an analytical method, while for the non-stationary policies in $\mathcal{B}_D$ generated by billiard sequences numeric methods like simulation are used to approximate the performance for $\theta \in \Theta_2$. In such a case optimization over $\mathcal{G}_D$ by the analytical method can be completed considerably faster and with more precision than optimization over $\mathcal{B}_D$. On the other hand for general $\mathcal{D}$ both the optimization over $\mathcal{G}_D$ and $\mathcal{B}_D$ should be done by approximation of performances. Then the applied method is similar and thus the speed and precision of optimizing over respectively $\mathcal{G}_D$ and $\mathcal{B}_D$ are expected to be comparable.
In the following section indeed simulation is applied to simultaneously approximate performances for policies in $\mathcal{G}_D$ and $\mathcal{B}_D$ and these performances will be optimized over $\theta \in \Theta_2$. Performing both 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 $\mathcal{B}_D$. For some problems (see [14] and [1]) it was shown that regular ($k = 2$ billiard sequences) have the best performance if (the direction) θ is fixed.

These results have been obtained in cases that the function which has to be minimized is multimodular. In [14] this was shown for the admission of arriving jobs in a single queue. In [1] multimodularity was shown under some conditions 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 the methods and conditions given in [1] to obtain multimodularity are not applicable. In [24] the optimality of regular sequences for mixing $k = 2$ decision rules for any $\theta \in \Theta_2$ is considered for general DTMDP. The optimality of regular sequences was established under some minor condition for DTMDP with only two states, but that partial result is not applicable to MJTAPSS which are modeled by MDP with an infinite state space. Therefore for MJTAPSS it is an open problem whether $f(\theta) \leq g(\theta)$ for $\theta \in \Theta_2$ if both are finite, but it seems to hold at least when $g(\theta)$ is locally convex around θ which is usually the case. In the next section for several examples of MJTAPSS and $\mathcal{D}$ with $k = 2$ optimizations over $\mathcal{G}_D$ and $\mathcal{B}_D$ are performed by numerical methods. Simultaneously the numerical approximations of $f(\theta)$ and $g(\theta)$ will be compared over the search space $\theta \in \Theta_2$. If $f(\theta) \leq g(\theta)$ for $\theta \in \Theta_2$ is confirmed in the numerical experiments then optimization over $\mathcal{B}_D$ gives better results than optimization over $\mathcal{G}_D$ while the required time for the optimization is similar. The numerical results resolve then the question about the third aspect (improvement) of Condition 11 for the subclass $\mathcal{B}_D$.

After discussing for $k = 2$ the tractability of optimization of the performance over $\mathcal{B}_D$, we consider now the tractability for $k \geq 2$ in general. An issue is that Proposition 17 can not be generalized to cases with $k > 2$. Namely for $k > 2$ the performance of a policy in $\mathcal{B}_D$ depends in general not only on the (essential) direction $\theta \in \Theta_k$ of the trajectory, but the performance may also depend on the initial position in the unit cube of the trajectory. This follows from the fact that for $k > 2$ parallel billiard sequences can have different finite subsequences if they are generated from different initial positions. As example with $k = 3$ consider the essential direction $\theta = (\frac{1}{2}, \frac{1}{3}, \frac{1}{6}) \in \Theta_3$. Since all three components of this θ are rational numbers with least common denominator equal to 6 it follows from the construction that all billiard sequences with this direction are periodic with period 6. Moreover, it is easily seen that both $(1, 1, 2, 3, 1, 2)$ and $(1, 2, 1, 3, 1, 2)$ are period cycles of billiard sequences with this direction θ but different initial position. Then for example 1,1 is a subsequence of the first billiard sequence one but not of the second while 1,3 is a subsequence of the second billiard sequence but not of the first. Since they have different subsequences it follows that the policies in $\mathcal{B}_D$ corresponding to these two billiard sequences will in general have different performances despite the fact that the policies are generated from parallel trajectories.

From this example it follows that for $k > 2$ optimization over $\mathcal{B}_D$ is more difficult than optimization over $\mathcal{G}_D$ since not only the direction $\theta \in \Theta_k$ has to be optimized since the performance is also influenced by the initial position of the trajectory. 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. The magnitude of this finite number depends on the direction θ. For some θ it can become large very soon if k gets larger. Only for very small values of $k > 2$ the number is always rather small and then it is tractable to obtain for billiard sequences with given direction $\theta \in \Theta_k$ the best possible performance by an exhaustive search over the initial position of the trajectory.

A more practical approach is based on the assumption that differences in performances because of initial position of the trajectory of policies in $\mathcal{B}_D$ are expected to be relatively small compared to differences in performance because of different direction $\theta \in \Theta_k$. In other words the optimization of the direction of the trajectory is much more important than the relatively small gains which can be obtained from optimizing the initial position. If the initial position is ignored for optimization then the initial position of the trajectory can be chosen arbitrarily (take for example always the origin as initial position) and just as for case $k = 2$ only optimize over the direction $\theta \in \Theta_k$. Then the functions $f(\theta)$ and $g(\theta)$ for optimization over

respectively $\mathcal{B}_D$ and $\mathcal{G}_D$ can be computed and compared similar to the $k = 2$ case described above. However, it should be noted that with this approach it is not be expected that the function $f(\theta)$ will be smooth as $g(\theta)$ is. Therefore despite the simplification by ignoring the initial position optimization over $\mathcal{B}_D$ is more complicated than optimization over $\mathcal{G}_D$. We conclude that the optimization problem is tractable as long as k is very small, but that it soon becomes difficult if k gets larger.

Now that for $\mathcal{B}_D$ the question about the first aspect of Criterion 11 is answered and meanwhile also the third aspect has been considered (concluding that this can be resolved by numerical experiments) we consider the second (the implementation) aspect of Criterion 11 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 $\mathcal{G}_D$. To confirms this the following remark is useful.

Remark 18 *Let $\mathcal{D} = \{d^1, d^2, \ldots, d^k\}$ and $\theta = (\theta_1, \theta_2, \ldots, \theta_k) \in \Theta_k$ be given and let π be a Markovian policy with $\pi \in \mathcal{K}_D$ (recall Definition 10). If policy π is implemented let p_l denote for $l = 1, 2, \ldots, k$ the long-run fraction of arriving jobs for which is server is assigned by 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, $l = 1, 2, \ldots, k$.*

1. *If $\pi = \pi_\theta \in \mathcal{G}_D$ is the randomized stationary policy given by Definition 8 and implemented as described in Remark 6 then with probability one it holds that $p_l = \theta_l$ for $l = 1, 2, \ldots, k$.*
2. *If $\pi \in \mathcal{B}_D$ is generated according to a billiard trajectory with essential direction θ then (independent of the initial position of the trajectory) it holds that $p_l = \theta_l$ for $l = 1, 2, \ldots, k$.*

We have seen how the simplex Θ_k is utilized for optimization over both $\mathcal{G}_D$ and $\mathcal{B}_D$. From Remark 18 it can be deduced that Θ_k is also a link for the implementation of policies in these two classes. Namely it follows that any policy in these classes is relatively easy to implement by utilizing the unique vector $\theta \in \Theta_k$ that is associated to the policy. In the first case for 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-interval only a finite number of random numbers has to 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 initial direction θ and initial position) not the whole infinite billiard trajectory and corresponding infinite sequence has to be calculated, but only a finite part of such a sequence. Moreover, for an online implementation it is not necessary to keep track of the whole trajectory, but only the direction and position at the last (actual) decision epoch 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 a minor 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 a minor calculation and update of a k -dimensional vector should be performed. In the following section on numerical experiments more specific implementation details on for example this information update at decision epochs are given.

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 several instances of MJTAPSS with $M = 2$ types of arriving customers and $N = 2$ servers are considered. The performance measure $V(\psi)$ 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 selected 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 subsection 3.2 and VC stands for the virtual cost rule as described in subsection 3.3.

We first discuss the technical details about the simulations which are performed. Consider the case that performances of mixing policies are approximated by simulation for some given $\mathcal{D}$ consisting of $k = 2$ decision rules. Then for both Bernoulli-mixing and billiard-mixing (recall Remark 13) the performance will be approximated 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 approximation will be done for both Bernoulli-mixing and billiard-mixing for θ in the set $\{\frac{a}{10}, a = 0, 1, \ldots, 10\}$. 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. Namely 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 approximations 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 with $M = 2$ types of arriving customers and $N = 2$ servers 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. We first investigate the policy mixing for $\mathcal{D} = \{DS12, SF\}$. The first simulation round is performed with $\varepsilon = 0.05$ and the results are presented in Figure 1.

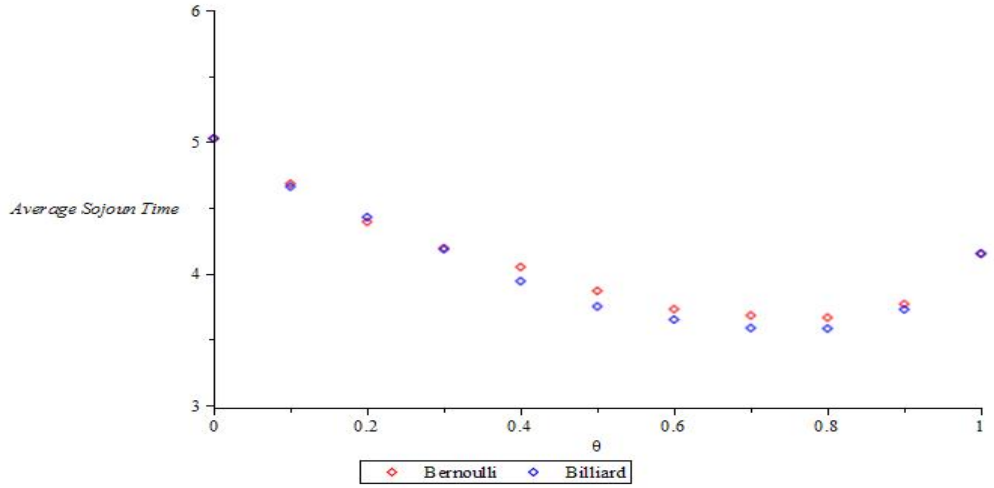

Figure 1: Mixing SF and DS12 for Instance 1

About the results presented in Figure 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. In general it is known that the SF policy performances poorly for heavy traffic, but in this case also for moderate traffic DS12 already performs considerably better. Recall that the pure DS12 policy is a static stationary policy and we have seen that the performance of static policies 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 $V(DS12) = \frac{\lambda_1}{\lambda_1+\lambda_2}V^1(DS12) + \frac{\lambda_2}{\lambda_1+\lambda_2}V^2(DS12) = \frac{5}{3} + \frac{5}{2} = \frac{25}{6} \sim 4.167$ which is in accordance with the simulation performance for $\theta = 1$ represented in Figure 1. By carrying out the optimization program described in [8] 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 $\frac{25}{6}$. However, according to Figure 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 which is close to 3% of 3.5. In the second simulation 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 $k = 0, 1, \ldots, 8$ and ε is chosen to be 0.025. The results of this second simulation round are presented in Figure 2.

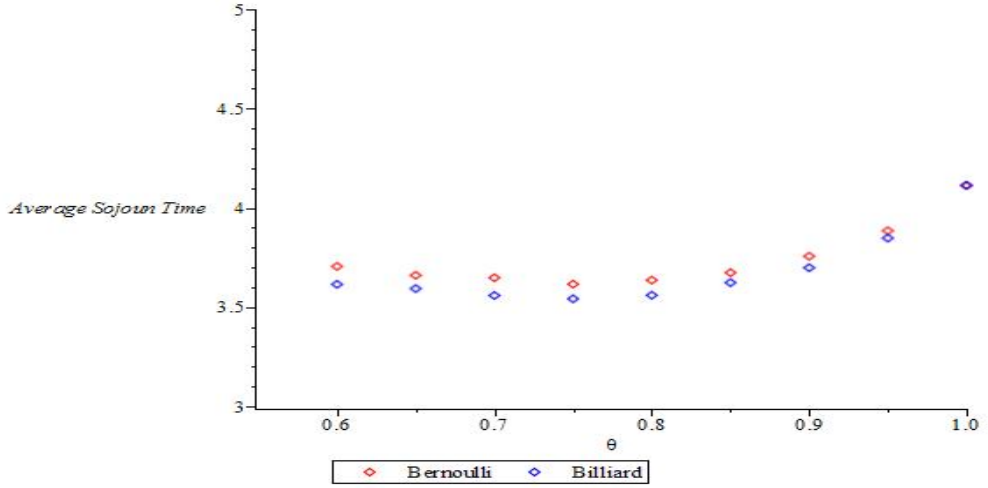

Figure 2: Second round of mixing SF and DS12 for Instance 1

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 [15] and [9]) 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 then obtaining the optimal value of θ with high precision. Indeed by focusing on performance values the optimal value of θ has not be obtained very precise since the performance function is more or less flat around the optimal value of θ. The results represented in Figure 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 decsion rules for $\mathcal{D} = \{DS12, SF\}$ we similarly investigate the mixing of decsion 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 Figure 3.

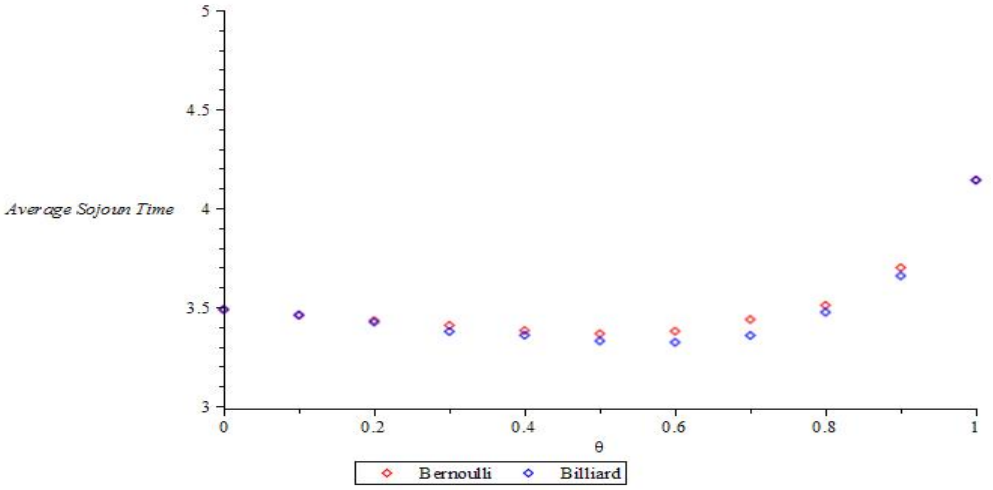

Figure 3: Mixing VC and DS12 for Instance 1

According to Figure 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 Figure 3 it is clear that by mixing VC with the DS12 rule the 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 $k = 0, 1, \ldots, 8$ and ε is chosen to be 0.025. The results of this second simulation round are presented in Figure 4.

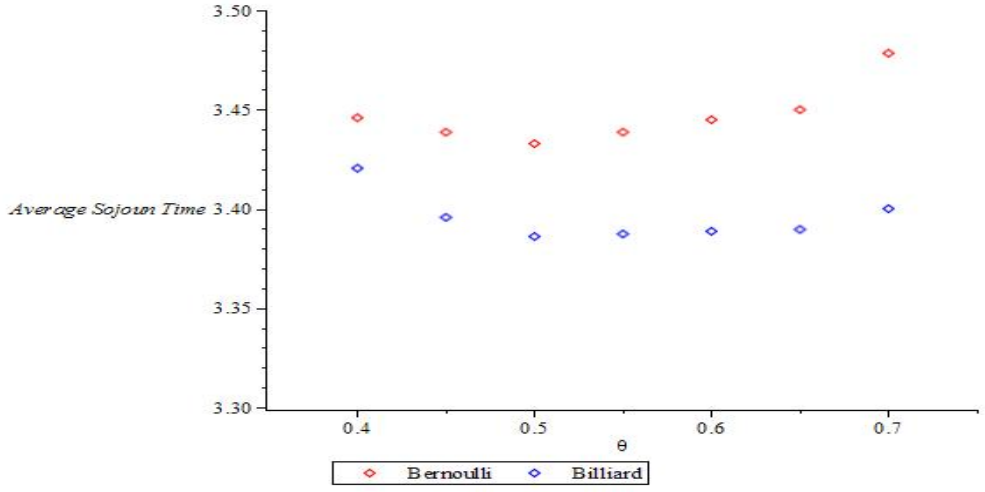

Figure 4: Second round of mixing VC and DS12 for Instance 1

In Figure 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 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 Figure 5. Figure 5 shows that the pure VC policy performs better than any genuine mixing between the SF and VC policy.

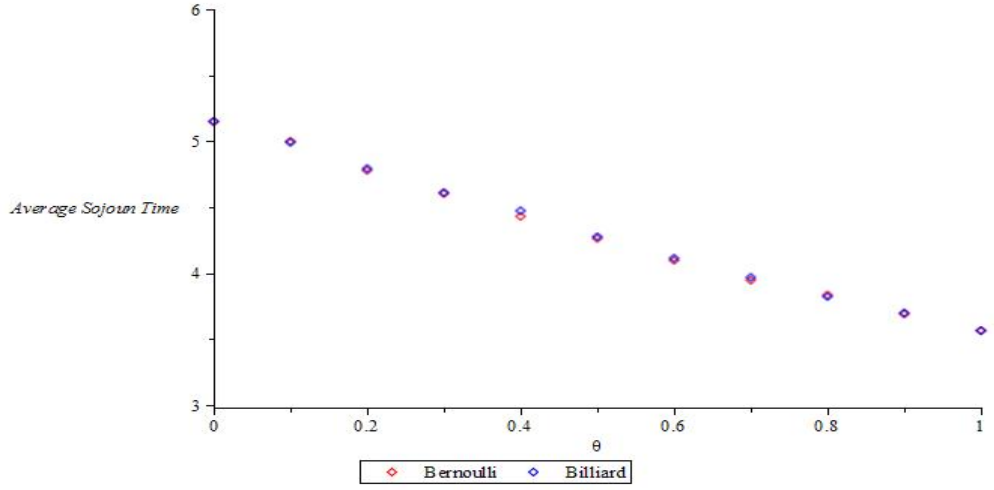

Figure 5: Mixing VC and SF for Instance 1

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. For $\mathcal{D} = \{DS12, SF\}$ the first simulation round is performed with $\varepsilon = 0.10$ and the results are presented in Figure 6. The pure SF policy (induced by choosing $\theta = 0$) performs very bad in such instances with rather heavy traffic load. In this instance it turned out that the system is not even stable for low values of θ. Therefore in Figure 6 only $\theta \in \{0.70, 0.80, 0.90, 1.00\}$) the simulated performances are represented since the system seems to be unstable for $\theta = 0.60$ or lower.
According to Figure 6 the simulated average sojourn time for the pure DS12 policy is about 10. Moreover, the The pure DS12 policy is a static stationary policy and we have seen that the performance of such policies 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 $V^1(DS12) = \frac{1}{\mu_{11}-\lambda_1} = \frac{1}{2.10-2.00} = 10$. Similarly it follows for the average sojourn time of type 2 jobs that $V^2(DS12) = \frac{1}{\mu_{22}-\lambda_2} = \frac{1}{1.10-1.00} = 10$ and thus the overall average sojourn time for DS12 is $V(DS12) = \frac{2}{3}V^1(DS12) + \frac{1}{3}V^2(DS12) = 10$ which is in accordance with the obtained numerical results.
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 negligible. 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 $[0.8; 1]$ which for both Bernoulli-mixing and billiard-mixing of SF and DS12 seems to contain the

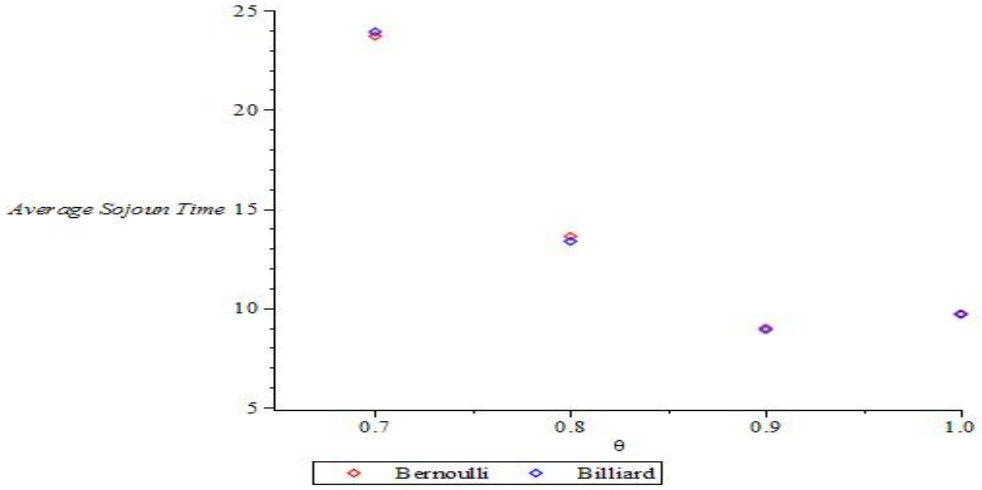

Figure 6: Mixing SF and DS12 for Instance 2

optimal value of θ 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 Figure 7.

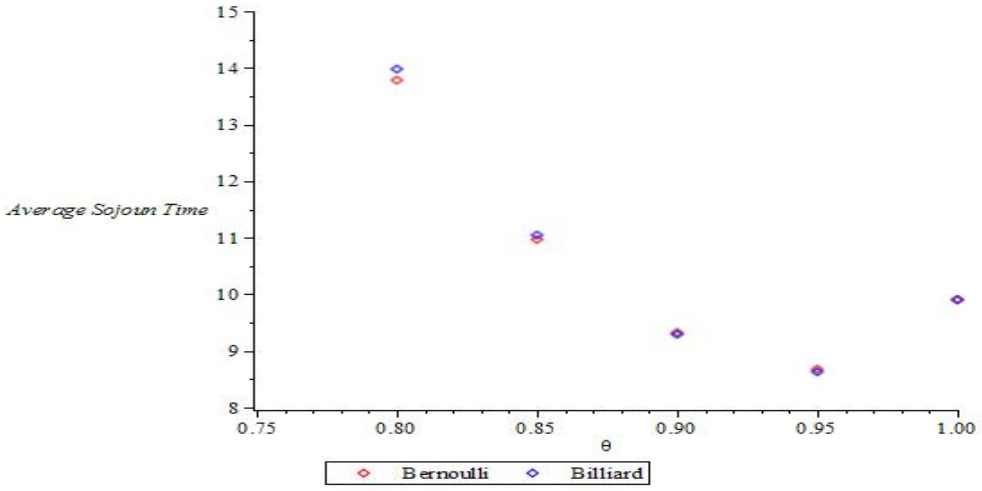

Figure 7: Second round of mixing SF and DS12 for Instance 2

According to Figure 7 for both Bernoulli-mixing and billiard-mixing the optimal value of θ seems to be about 0.95. Also after zooming in the differences in performance between Bernoulli-mixing and billiard-mixing seem to be negligible 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 (again the improvement is about 15 %) than the performance of 10 of the pure DS12 policy. This improvement is quite remarkable since in this instance the optimal mixing policies apply 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. For this instance performing the optimization over all static policies as described in [8] 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 which is very slightly better than 10, but still is much worse than performances close to 8.5 which can be obtained by mixing for $\mathcal{D} = \{DS12, SF\}$.

Next the optimal mixing is investigated for $\mathcal{D} = \{DS12, VC\}$ when possibly (as was the case for Instance 1) even better performances can be obtained 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. For these two decision rules the system turns out to be stable for all mixing parameters $\theta \in [0, 1]$ and thus it is possible to simulate performance values over the entire interval. The first simulation round is performed with $\varepsilon = 0.10$ and the results are presented in Figure 8.

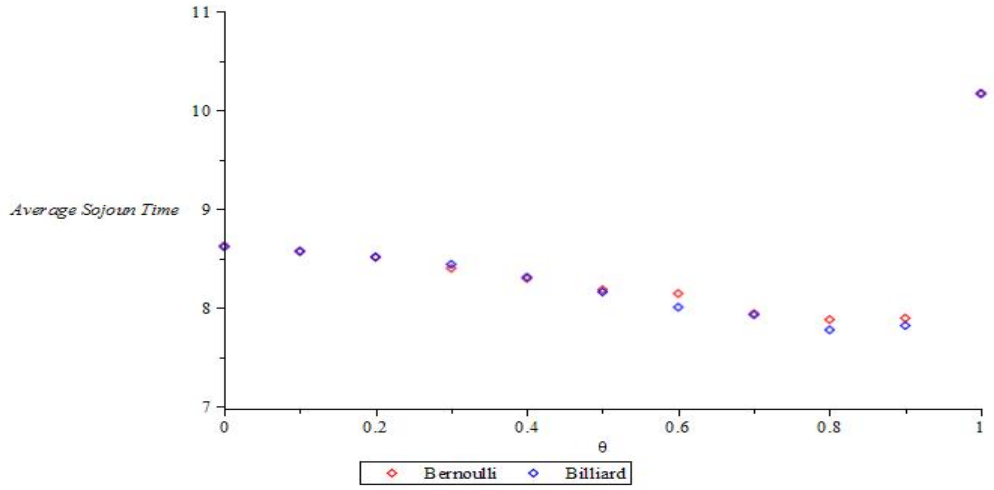

Figure 8: Mixing VC and DS12 for Instance 2

According to Figure 8 the performance of the VC policy is slightly above 8.5 which is (again by coincidence) close to the best performance which could be obtained by mixing SF and DS12. Moreover, by mixing VC with the DS12 rule the performance can be further improved. The optimal performance is attained for values of θ around 0.8 and it seems that then a performance around 8 or even slightly better is obtained. Also around the optimal value of θ billiard-mixing seems to perform slightly better than Bernoulli-mixing. To zoom in the second round the performance is simulated for $\theta = 0.70 + 0.05k$ for $k = 0, 1, \ldots, 6$ and ε is chosen to be 0.05. The results of this second simulation round are presented in Figure 9.
Figure 9 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 consequently performs better than Bernoulli-mixing, but performance differences between the two methods seem to be not more than 1% and thus relatively small.

Finally for Instance 2 also the mixing with $\mathcal{D} = \{SF, VC\}$ can be investigated. 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 would be monotonically decreasing as it is in Figure 5 for Instance 1.

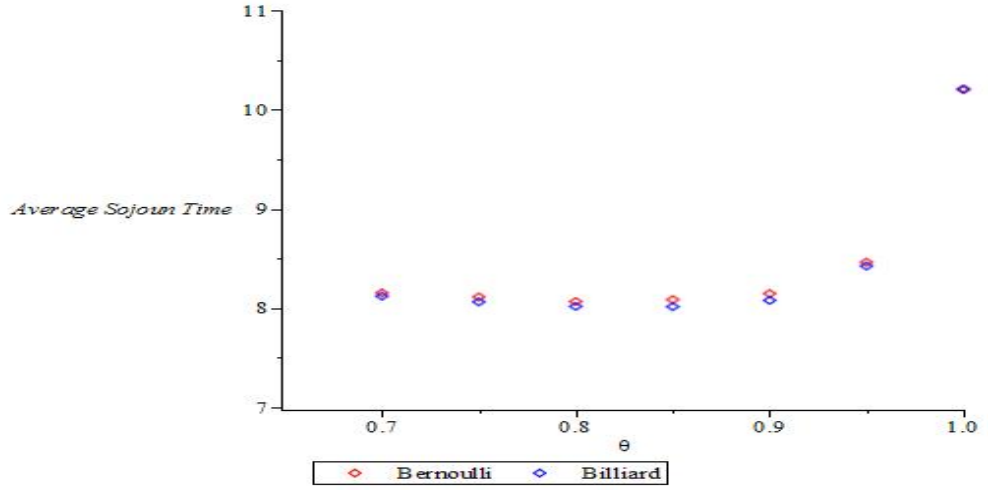

Figure 9: Second round of mixing VC and DS12 for Instance 2

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. For $\mathcal{D} = \{DS12, SF\}$ the first simulation round is performed with $\varepsilon = 0.05$ and the results are presented in Figure 10.

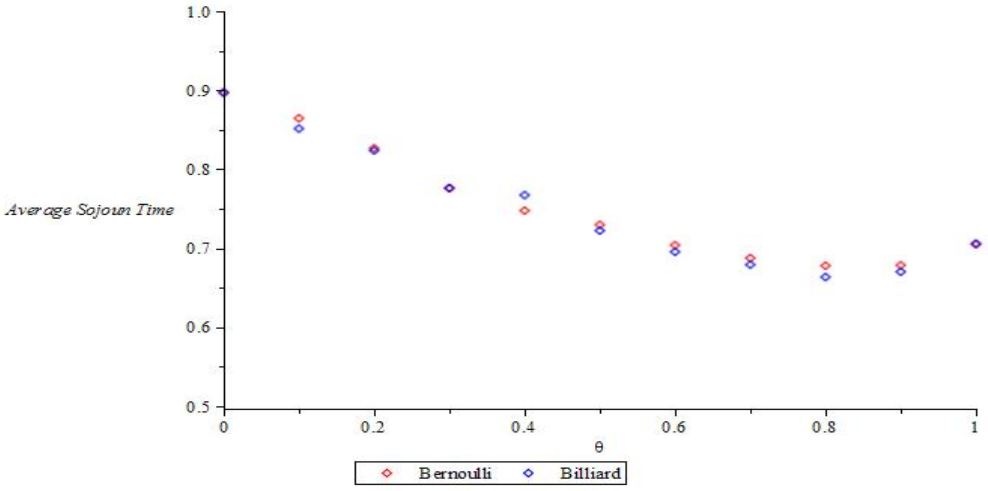

Figure 10: Mixing SF and DS12 for Instance 3

The pure SF policy corresponding to $\theta = 0$ seems to have simulated performance around 0.9 which is worse than the simulated performance of the pure DS12 policy corresponding to $\theta = 1$. Again 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 $V^1(DS12) = \frac{1}{\mu_{11}-\lambda_1} = \frac{1}{5.0-3.0} = 0.5$. 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 simulation performance for $\theta = 1$ represented in Figure 10. By carrying out the optimization program described in [8] 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. According to Figure 10 it seems that by mixing SF and DS12 a slightly better 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. To zoom in the second round the performance is simulated for $\theta = 0.60 + 0.05k$ for $k = 0, 1, \ldots, 8$ and ε is chosen to be 0.025. The results of this second simulation round are presented in Figure 11.

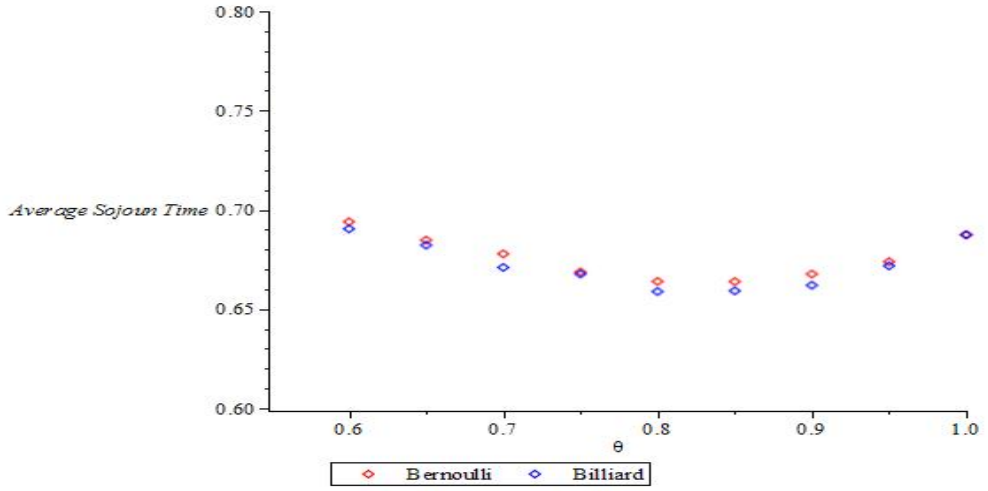

Figure 11: Second round of mixing SF and DS12 for Instance 3

Figure 11 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 after zooming in it is clear that for the mixing policy with optimal θ the performance is a few percent better than the 0.7 of the pure DS12 policy. Moreover, it is confirmed that as we have seen before 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 $\mathcal{D} = \{DS12, VC\}$ when it can be expected (see 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. The first simulation round is performed with $\varepsilon = 0.05$ and the results are presented in Figure 12.
From Figure 12 it follows that $\mathcal{D} = \{DS12, VC\}$-mixing performances close to 0.62 can be reached which improves more than 10% on the optimal static policy DS12. In Figure 12 the performance function appears to be rather flat for $\theta < 0.7$ and therefore in the second round we zoom in on this part to determine the optimal value of θ more precisely. In the second round the performance is simulated for $\theta = 0.0 + 0.05k$ for $k = 0, 1, \ldots, 13$ and ε is chosen to be 0.01. The results of this second simulation round are presented in Figure 13.
From Figure 13 it seems that for billiard-mixing the optimal value of θ is slightly above 0.4 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. On the other hand it is apparent that billiard-mixing gives around the optimal point consistently better performances than Bernoulli-mixing although the differences are relatively small.
Once more no improvement is obtained by $\mathcal{D} = \{SF, VC\}$-mixing since it again turns out that the pure VC policy performs better than any genuine mixing between the SF and VC policy.

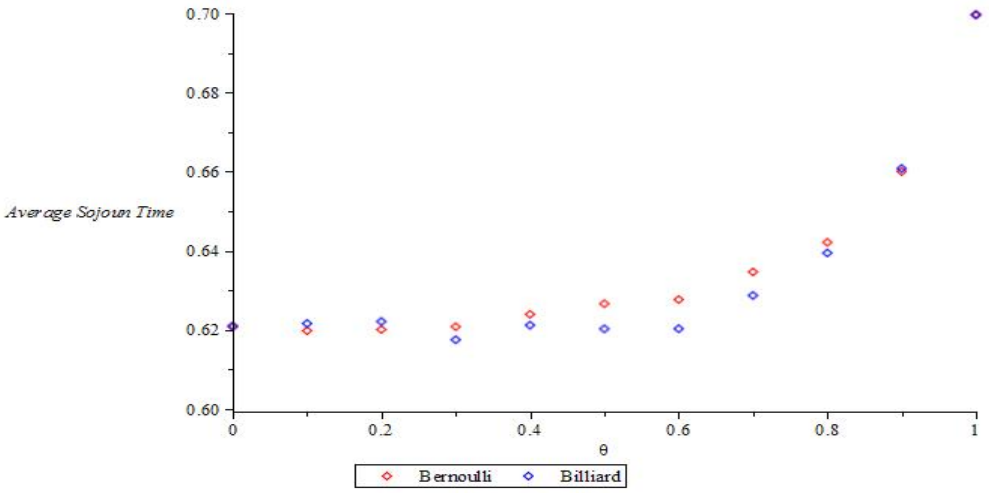

Figure 12: Mixing VC and DS12 for Instance 3

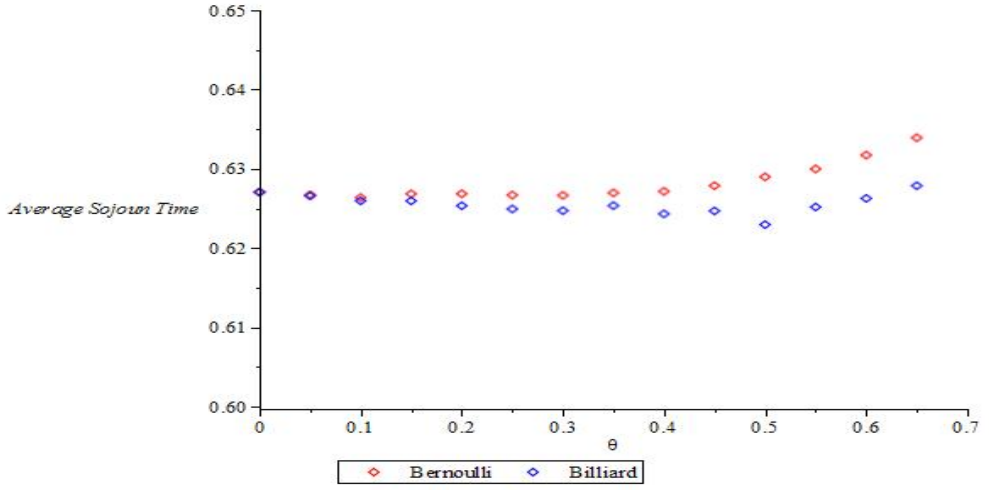

Figure 13: Second round of mixing VC and DS12 for Instance 3

5.1 Conclusions and remarks

The numerical results obtained by simulation for policy mixing for MJTAPSS instances with various traffic loads have been shown and interpreted. One static rule (DS12) and two dynamic rules (SF and VC) have been utilized for policy mixing. To obtain improvements by mixing decision 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 should suffice to use only one static rule for mixing, but then this static rule should be chosen carefully. For the investigated instances DS12 is for two reasons the most obvious choice as static rule. The first reason is that DS12 is a deterministic static rule. Thus the rule 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. The second and most important reason is that for all investigated instances DS12 is optimal or very close to optimal within the class of static decision rules. This also implies for these instances the exactly computable performance of DS12 is a good reference to evaluate the performance obtained by policy mixing.

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. In the investigated instances the performance of the pure SF policy is worse than for the static policy induced by DS12. For Instance 2 applying the SF policy does even result in an unstable system. However, by mixing this inferior dynamic rule with the static DS12 rule rather good performances can be achieved. In each case the performance of DS12 and thus the best static policies could be substantially improved by this mixing. 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 a more sophisticated but also easy to implement dynamic rule 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 better than for the DS12 rule and thus also much better than for 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 all the investigated instances that the dynamic VC rule can be improved up by mixing it with the static DS12 rule. By mixing the DS12 and VC rule also a better performance can be obtained than by mixing the DS12 and SF rule. Moreover, both the Bernoulli-mixing method and billiard-mixing method give good results, but that in general for the best billiard-mixing policies have a slightly better performance than the best Bernoulli-mixing policies. To summarize for all the investigated MJTAPSS instances the best performance was obtained by mixing the DS12 and VC rule according to the billiard-mixing method. In that way the improvement in performance compared to the best static policies can be 15% or even more.

We note that in general also the mixing of more than two decision rules can be investigated. For example the DS12, SF and 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 of course besides for MJTAPSS 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), but maybe for some problems mixing multiple dynamic rules gives also good results.

References

[1] B. Altman, E. Gaujal and A. Hordijk. *Discrete-Event Control of Stochastic Networks: Multimodularity and Regularity.* Lecture Notes in Mathematics. Springer Verlag, 2003.

[2] E. Altman, U. Ayetsa, and B. Prabhu. Load balancing in processor sharing systems. *Telecommunication Systems*, 47:35–48, 2011.

[3] J. Anselmi and G. Casale. Heavy-traffic revenue maximization in parallel multiclass queues. *Performance Evaluation*, 70:806–821, 2013.

[4] M. Armony. Dynamic routing in large-scale service systems with heterogeneous servers. *Queueing Systems*, 51:287–329, 2005.

[5] M. Armony and A.R. Ward. Fair dynamic routing in large-scale heterogeneous-server systems. *Operations Research*, 58:624–637, 2010.

[6] P. Arnoux, P. Mauduit, I. Shiokawa, and J. Tamura. Complexity of sequences defined by billiards in the cube. *Bull. Soc. Math. France*, 122:1–12, 1994.

[7] Y. Baryshnikov. Complexity of trajectories in rectangular billiards. *Comm. Math. Phys.*, 175:43–56, 1995.

[8] K.J. Becker, D.P. Gaver, K.D. Glazebrook, P.A. Jacobs, and S. Lawphongpanich. Allocation of tasks to specialized processors: A planning approach. *European Journal of Operations Research*, 126:80–88, 2000.

[9] S. Bhulai, T. Farenhorst-Yuan, B. Heidergott, and D.A. van der Laan. Optimal balanced control for call centers. *Annals of Operations Research*, 201:39–62.

[10] S.C. Borst. Optimal probabilistic allocation of customer types to servers. In *Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 116–125, 1995.

[11] H. Chen and H. Ye. Asymptotic optimality of balanced routing. *Operations Research*, 60:163–179, 2012.

[12] H. Feng, V. Misra, and D. Rubenstein. Optimal state-free, size-aware dispatching for heterogeneous m/g/- type systems. *Performance Evaluation*, 62:475–492, 2005.

[13] K.D. Glazebrook and J. Nino-Mora. Parallel scheduling of multiclass $m/m/m$ queues: approximate and heavy-traffic optimization of achievable performance. *Operations Research*, 49:609–623, 2001.

[14] B. Hajek. Extremal splittings of point processes. *Mathematics of Operations Research*, 10(4), 1985.

[15] B. Heidergott, F. Vázquez-Abad, G. Pflug, and T. Farenhorst-Yuan. Gradient estimation for discrete-event systems by measure-valued differentiation. *Transactions on Modeling and Computer Simulation*, 20:5.1–5.28, 2010.

[16] A. Hordijk and G. Koole. On the assignment of customers to parallel queues. *Probability in the Engineering and Informational Sciences*, 6:495–511, 1992.

[17] E. Hyytiä, A. Penttinen, and S. Aalto. Size- and state-aware dispatching problem with queue-specific job sizes. *European Journal of Operations Research*, 217:357–370, 2012.

[18] J.D.C. Little. Little's law as viewed on its 50th anniversary. *Operations Research*, 59:536–549, 2011.

[19] M. Lothaire. *Algebraic Combinatorics on Words*. Cambridge University Press, 2002.

[20] M. Morse and G.A. Hedlund. Symbolic dynamics ii - sturmian trajectories. *American Jornal of Mathematics*, 62:1–42, 1940.

[21] R.F. Serfozo. An equivalence between continuous and discrete time markov decision processes. *Operations Research*, 27:616–620, 1979.

[22] J. Sethumaran and M.S. Squillante. Optimal stochastic scheduling in multiclass parallel queues. In *Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 93–102, 1999.

[23] H.C. Tijms. *A first course in stochastic models*. Wiley, 2003.

[24] D.A. van der Laan. Optimal mixing of markov decision rules for mdp control. *Probability in the Engineering and Informational Sciences*, 25:307–342, 2011.