MATHEMATICS OF OPERATIONS RESEARCH
Vol. 24, No. 2, May 1999
Printed in U.S.A.

THE COMPLEXITY OF OPTIMAL QUEUING NETWORK CONTROL

CHRISTOS H. PAPADIMITRIOU AND JOHN N. TSITSIKLIS

We show that several well-known optimization problems related to the optimal control of queues are *provably intractable*—independently of any unproven conjecture such as $P \neq NP$. In particular, we show that several versions of the problem of optimally controlling a simple network of queues with simple arrival and service distributions and multiple customer classes is complete for exponential time. This is perhaps the first such intractability result for a well-known optimization problem. We also show that the restless bandit problem (the generalization of the multi-armed bandit problem to the case in which the unselected processes are not quiescent) is complete for polynomial space.

1. Introduction. The optimal control of a network of queues is a well-known, much studied, and notoriously difficult problem. We are given several servers, a set of customer classes, and class-dependent probability distributions for the service times. For each customer class, there is only one server that can serve customers of that class, but the same server might be eligible for several classes. The class of a customer can change at each service completion time; for some customer classes, the new class is under our control; for others, the class change is probabilistic. We restrict ourselves to closed networks in which there is a finite number of customers that never leave the system, and no external arrivals. The throughput of a class of customers is defined as the steady-state average number of service completions for that class per unit time; our performance measure will be a weighted sum of the throughputs of the different classes. Operating the network amounts to choosing the new class of a customer whose service has just been completed (routing) and choosing at each server which customer to serve next, out of all eligible customers (sequencing). The problem is to come up with a routing and sequencing strategy—with decisions presumably based on the load of the queues—so as to optimize weighted throughput.

Networks of queues have many applications, the most important ones related to communication networks or manufacturing systems. There are precious few cases of the problem that have been satisfactorily solved and are reviewed by Walrand (1988); for example, the problem is wide open even for the case of *two-server* networks and exponential service time distributions. Besides some *ad hoc* techniques for very special cases, and computationally explosive dynamic programming algorithms for others, we can only solve to optimality certain single-server problems (Klimov 1974, Weiss 1988), by reducing them to extensions of the *multi-armed bandit problem*, the problem of repeatedly selecting one among many Markov processes, each with known transition probabilities and costs. The latter problem can be solved by an ingeniously simple index calculation (Gittins 1989), and an optimal policy corresponds to prioritizing the different classes by sorting their respective indices. Due to the difficulty of the problem, research in this area has been deflected to approaches such as diffusion approximations (Harrison 1985) and certain other rigorous approximation algorithms (Bertsimas et al. 1994, Kumar and Kumar 1994).

In this paper *we prove that the problem of finding an optimal control policy in a multiclass*

Received July 8, 1994; revised December 20, 1998.
AMS 1991 subject classification. Primary: 68Q15, 68Q25; Secondary: 60K20, 60K25.
OR/MS subject classification. Primary: Analysis of Algorithms/Computational Complexity; Secondary: Queues/Networks.
Key words. Computational complexity, optimal control, queuing network.

0364-765X/99/2402/0293/$05.00

closed queuing network is an intractable problem. Nobody was really expecting an efficient algorithm for this problem, at least in this generality, and it would be trivial to show it NP-hard (the vanilla variety of intractability available in the literature). However, our result is much stronger: we show that the problem *provably* requires exponential time for its solution, *independently* of the P vs. NP question. In particular, we show that it is *EXP-complete*.

EXP is the class of all problems solvable in time 2^{n^k} for some k. EXP-complete problems are the problems in EXP to which every other problem in EXP is efficiently reducible; such problems are *known* (Papadimitriou 1994) to require exponential time—independently of the status of the P = NP question. There are many such intractability results in the literature, starting from the classical ones about regular expression equivalence (Stockmeyer and Meyer 1973), Presburger arithmetic (Fischer and Rabin 1974), and other logics such as, more recently, variants of Temporal Logic. However, in our experience this is the first intractability result for a practical and important optimization problem that had been attacked in earnest over many decades—in contrast, NP-completeness theory is teeming with optimization problems.

In the next section, we introduce the problem NETWORK OF QUEUES, a relatively simplified version of the problems one finds in the literature (Walrand 1988, Bertsimas et al. 1994, Kumar and Kumar 1994). The proof that it can be solved in exponential time relies on the fact that it can be rendered as a Markov decision process with exponentially many states, which can then be solved by linear programming. To prove completeness, we rely on a heretofore untapped alternative characterization of EXP, namely in terms of *polynomial space bounded stochastic computation* (a similar formulation of polynomial *space* as polynomial *time* bounded stochastic computation was proposed by Papadimitriou 1985). Besides the case of exponentially distributed service times, we show that the lower bound also holds for the case of deterministic service times, and for the case where service times have a discrete probability distribution and routing is deterministic. However, if both routing and the service times are deterministic, the problem can be easily shown to be *PSPACE-complete*. Recall that PSPACE is the class of problems solvable in polynomial *space*. The inclusions P $\subseteq$ NP $\subseteq$ PSPACE $\subseteq$ EXP are well known (Papadimitriou, 1994), and are all strongly believed to be proper (we only know that at least one of them is, since it can be proved that EXP properly contains P). Unlike EXP-completeness, PSPACE-completeness does not immediately imply intractability, but it strongly suggests it—it is considered a much more convincing evidence of intractability than NP-hardness.

Given that the multi-armed bandit problem is the main tool for solving the few cases of networks of queues that we *can* solve, it is interesting to study the complexity of its most promising extension, the *restless bandit problem* (Whittle 1988, Weber and Weiss 1990). We show that this problem is also PSPACE-complete, even in the deterministic case.

2. Networks of queues. A *network of queues* consists of a finite set of servers S and a finite set C of customer classes. For each class $c \in C$, we are given the identity $\sigma(c) \in S$ of the only server who can serve customers of that class, and the mean service time $\mu(c)$. Service times are independent exponentially distributed random variables with the prescribed mean. The set C is partitioned into two subsets, R and D. Whenever a customer currently in class c completes service, its class changes to some new class c'. For each $c \in D$, we are given a set $N(c) \subseteq C$ and c' is allowed to be an element of $N(c)$ of our choice. If on the other hand $c \in R$, the new class c' is determined at random according to given probabilities $p_{cc'}$.

The queuing network is controlled by making decisions of the following nature: each time that a customer of some class $c \in D$ completes service, we choose its next class $c' \in N(c)$—these are *routing* decisions. In addition, at each service completion time, any free server can choose to remain idle or to start serving an eligible customer. We only consider

nonpreemptive policies; that is, once a server starts serving a customer, it must continue until service is completed.

A queuing network of the type described here is *closed*: no new customers arrive and no customers can leave the network; in particular, the total number of customers is conserved. At any point in time, the *state* of the network consists of the following information: (a) how many customers of each class are present in the system, and (b) the class of the customer (if any) being served at each server.

A *policy* is a rule for making decisions at service completion times, as a function of the current state of the network. Due to the independence and exponentiality of the service times, the state of the network evolves as a Markov chain under any fixed policy.

Let us fix the initial state of the network. For any policy π and any class $c \in C$, the number $a_c^\pi(t)$ of class c service completions until time t is a well-defined random variable. We then consider as our performance measure the weighted throughput

$$J^\pi = \limsup_{t\to\infty} \frac{1}{t} \sum_{c\in C} w(c)E[a_c^\pi(t)],$$

where $w(c)$ are given weights and $E[\,\cdot\,]$ denotes expectation. We are interested in finding a policy that maximizes J^π, as well as the corresponding optimal value of J^π.

We can now provide a formal definition of the problem NETWORK OF QUEUES. An instance is specified by the finite sets C, S, the function $\sigma : C \mapsto S$, rational-valued functions μ and w defined on C, subsets D and R of C, a set $N(c) \subseteq C$ for each $c \in D$, rational coefficients $p_{cc'}$ for every $c \in R$ and $c' \in C$, the initial numbers $n_i(c)$ of customers of each class, the set S_0 of busy servers at time zero, the class c_s of the customer being served at time zero by each server $s \in S_0$, and a rational number K. The problem is to decide whether there exists a policy π for which $J^\pi > K$.

We shall establish that this problem is complete for the class EXP of all problems solvable in deterministic exponential time. In our proof we shall use a novel characterization of EXP, in terms of *stochastic Turing machines*, which we now proceed to develop. A stochastic Turing machine is a Turing machine whose states are divided into two parts: the *nondeterministic* states and the *stochastic* states. We assume that all state-symbol combinations, except of course for the final ones, have *two* possible continuations. We also assume that the machine is *precise*, that is, on input x it only visits the first $|x|$ tape squares, and it stops after exactly $f(|x|)$ steps, where $f(n)$ is some function depending on the machine (we can take $f(n) = 2^{\lfloor n^{1/2} \rfloor}$). For each input x this machine defines a tree of computations, of depth $f(|x|)$. We evaluate this tree starting from the leaves (where a leaf has value one of zero, depending whether it is accepting or rejecting), so that the value of each internal node is the maximum of its two children if the state of the node is nondeterministic, and the average of its two children if it is stochastic. We say that the machine *accepts* input x if this tree of computations evaluates to more than half. Notice that, since we assume that the machine is precise, this is equivalent to saying that there is a way to choose one branch out of every nondeterministic node such that the *majority* of the leaves are accepting.

Our proof relies on the fact that stochastic Turing machines operating in linear space capture the class EXP:

PROPOSITION. *The class of all languages decided by stochastic Turing machines in polynomial space is precisely EXP.*

The proof of this proposition is contained in the proof of the following more technical lemma: The STOCHASTIC IN-PLACE ACCEPTANCE problem is the following: Given a stochastic Turing machine M and an integer n in unary, does M accept the input consisting of n zeroes, without using any space beyond that occupied by the input?

LEMMA 1. *STOCHASTIC IN-PLACE ACCEPTANCE is EXP-complete.*

PROOF. The addition of stochastic states and acceptance convention to Turing machines is known to have the effect of turning time into space: It was shown by Papadimitriou (1985) that the languages decidable by polynomial time-bounded stochastic Turing machines comprise all of PSPACE. The present lemma essentially says that, as is the case with the *alternating* machines (another much studied variant (Chandra et al. 1981, Papadimitriou 1994)), stochastic machines also turn space into time *one exponential higher*.

We first show that any space-bounded stochastic Turing machine can be simulated in exponential time. The simulation algorithm is simple: We visit the exponentially many configurations of the machine one-by-one, in decreasing time (we assume without loss of generality that the configuration embodies also the time at which the configuration can occur), and we compute the probability of acceptance (based on the results of the same calculation one time step later). For stochastic configurations, this entails averaging over the two possible successor configurations; for nondeterministic (existential) configurations, we take the maximum. Since there are exponentially many possible configurations, we can carry out this computation in exponential time.

To show completeness, we shall show how to simulate by a stochastic machine any *alternating* machine with polynomial space bounds; it is known (Chandra et al. 1981) that the power of such alternating machines is precisely EXP. The idea is similar to that in Papadimitriou (1985) for space: Existential states are simulated by nondeterministic states, and universal states by stochastic states. It then follows that the stochastic machine accepts with probability 1, if and only if the alternating machine accepts. Let us now modify the stochastic machine so that it starts at a stochastic state and one of the continuations can only lead to nonaccepting states, while the other continuation leads to a state from which the alternating machine will be simulated as just described. Then, the acceptance probability of the stochastic machine is $\frac{1}{2}$ if and only if the alternating machine accepts, and is smaller otherwise.

To convert polynomial space to linear space is standard (use "padding" to transform any input x to x followed with polynomially many zeros). To rephrase the problem in a form with no explicit input (all-zero input), just absorb any given input x in the description of the machine; this machine always starts by printing x on its tape. □

We are now ready for the proof of the main result:

THEOREM 1. *NETWORK OF QUEUES is EXP-complete.*

PROOF. For the upper bound (that is, in order to show that the problem can be solved in exponential time) notice that the problem is a *Markov decision process*: We are given a Markov chain with possible decisions that affect the transition probabilities and costs, and we seek to find a policy that minimizes the average cost. It is known that such problems can be reformulated as a linear program, and thus solved in polynomial time (Puterman 1994). But of course, the number of states in the Markov process (and thus, the number of variables in the linear program), is exponential in the size of the data—hence membership in EXP.

To show completeness, we shall reduce the STOCHASTIC IN-PLACE ACCEPTANCE problem to NETWORK OF QUEUES. We are given a stochastic Turing machine M and an integer n. We will be making the following additional assumptions about M, besides the fact that it is precise, halting always after $f(|x|)$ steps. We first assume that M's alphabet consists only of the symbols 0 and 1 (any alphabet can be thus encoded). Also, the start state is never visited again during the computation (this is easily guaranteed by introducing an extra state). Furthermore, when the computation ends, the tape has again n zeroes (recall that the machine starts in this configuration), and the head is at the first position (this only requires a final "clean-up" phase). We next assume that, when a final state is reached, the machine does not halt but starts running again from the initial configuration (this can be achieved by simply

treating final states identically with the start state). We also assume that the transition function of the machine has been modified so that at each step the symbol scanned is always overwritten by a different symbol—which can of course be again overwritten in the next step; this is no loss of generality. Finally, let $f(n)$ be the exact number of steps the machine takes to halt when started on n zeroes.

We shall now give a high-level overview of the network of queues we construct from the given Turing machine. There are $n + 1$ customers; the first n customers G_i, $i = 1, \ldots, n$ will encode the tape contents, while the $n + 1$st customer, the *test* customer T, will encode the current state and tape square (it will become clear later in the proof that T also encodes the next move chosen by the machine). Each of these customers potentially belongs to several classes, and is served by several servers. In general, the class of a customer will be such that it fully describes the present server of the customer, as well as a part of the future route (sequence of servers) to be followed by this customer. Only one server, the *router* server r, can serve all customers; all other servers are customer-specific.

More specifically, corresponding to the ith square of the tape, $i = 1, \ldots, n$, we have two servers, g_{i0} and g_{i1}. For each i, there is a single customer G_i that can be served by either of these two servers. In order for G_i to move from one of these two servers to the other, it must go through the router server r. If G_i goes from g_{ib} to r, it can get served there any number of times, but when it leaves, it can only go to server $g_{i,1-b}$. We assume that the mean service time of G_i is unity at all three relevant servers.

Each transition of the Turing machine can be viewed as a 6-tuple of the form $(s, i, b; s', i', b')$ where s is the current state, i is the current position of the head, b is the symbol (bit) in the ith square, s' is the new state, i' is the new position of the head, and b' is the symbol written on the tape (in the ith square). We have assumed that $b' = 1 - b$; thus, b' can be omitted, and we will work instead with 5-tuples $(s, i, b; s', i')$. The rules of operation of the Turing machine can be described by specifying for each (s, i, b), two different continuations (s', i'). (If s is a stochastic state, one of the two continuations is chosen at random, with probability $\frac{1}{2}$ each; for nondeterministic states, we are free to choose one of the two options.)

The current class (alternatively, current server) of the test customer T will encode the state of the machine, the position of the head, and the chosen continuation. For every possible state s and for $i = 1, \ldots, n$, customer T can be of class (s, i) in which case it is to be served at a corresponding server (s, i). Then, the test customer makes an *excursion* through the network along one of a few possible routes, each possible route corresponding to a possible transition $(s, i, b; s', i')$ of the Turing machine. (The choice of a route is enforced by having the route become part of the description of the customer's class. In this way, the class of a customer can "remember" the customer's eventual destination.) The symbol $b \in \{0, 1\}$ above is chosen by the policy (to be defined soon) so as to correctly reflect the currently scanned symbol. If s is a nondeterministic state, then the policy also chooses one of the two available continuations (s', i') of (s, i, b). If s is a stochastic state, then (s', i') is chosen at random among the two possibilities, each possibility being equally likely.

Once the transition $(s, i, b; s', i')$ has been selected, the route to be followed during the test customer's excursion is the following: Get served once by server $g_{i,1-b}$, then by g_{ib}, and finally by the router server, to end up at server (s', i'), which is the end of the excursion. In the beginning of an excursion implementing the choice $(s, i, b; s', i')$, the test customer assumes a class specific to that excursion and choice. We assume that the mean service time of the test customer is zero at all servers except for servers g_{i0}, g_{i1}, and the router server, where the mean service time is unity.

We choose the weights $w(c)$ as follows. The weight of all customers other than the test customer is a very large number B. The weight of the test customer is zero at all servers, except for servers of the form (s, i). There, the weight is unity if s is a nonaccepting state

and $1 + \epsilon$ if s is an accepting state, where ϵ is a very small number, e.g., smaller than $f(n)$. (The number B can be chosen independently of n.)

The network is initialized with each customer G_i at server g_{i0} and with the test customer at server (s_0, i), where s_0 is the start state of the Turing machine, and $i = 1$, is the initial position of the head. Thus, the initial state of the network encodes the starting configuration of the Turing machine.

We will now describe a particular policy π^* for controlling the queuing network which simulates the stochastic Turing machine. (We will argue later that this policy is optimal.) Under policy π^*, the customers G_i, $i = 1, \ldots, n$, remain always busy. In general, they keep getting served over and over at the same server with some exceptions to be described shortly.

Suppose that the test customer is at some server (s, i) and that the router server is free. Then, the policy observes the content b of the ith tape square (which is b if and only if G_i is at g_{ib}). At that point, $(s, i, b; s', i')$ is chosen according to the transition rules of the stochastic Turing machine. If s is a stochastic state, (s', i') is chosen randomly; if s is a nondeterministic state, (s', i') is chosen in an optimal way, that is, according to a policy that maximizes the probability of acceptance in the stochastic Turing machine.

The test customer moves to $g_{i,1-b}$, finds it idle (because G_i is at g_{ib}) and starts service there. Once service is completed, it moves to server $g_{i,b}$. Recall that customer G_i was initially at g_{ib}. As soon as its service is completed, G_i moves to the router server, and g_{ib} becomes available to the test customer.

Customer G_i gets served at the router server and then moves to server $g_{i,1-b}$. (However, it will only choose to move to $g_{i,1-b}$ if T has left $g_{i,1-b}$. In particular, G_i may have to be served more than once at the router server.) Eventually, T finishes service at g_{ib} and moves to the router server where it may or may not have to wait for customer G_i. Once T finishes service at the router server, it assumes its new class (s', i') and the excursion has been completed. Notice that at the time that the excursion is completed, the router server has just been freed from T and none of the customers G_i will be found there. In particular, the class of the test server and the location of the customers G_i encode completely the new configuration of the Turing machine.

It is clear that under policy π^*, each excursion corresponds to a correct simulation of one transition of the stochastic Turing machine. In addition, the mean duration of each excursion is constant, and equal to 4. To see why, note that initially, T is served at $g_{i,1-b}$ and G_i is served at $g_{i,b}$, and the mean time until the first move is $\frac{1}{2}$. If T was the first to complete service, it moves to $g_{i,b}$ and waits for an average of one time unit until G_i moves to the router. If G_i was the first one to complete service, it moves to the router and gets served there over and over until T moves to $g_{i,b}$, which happens one time unit later on the average. At this point, and under either scenario, the expected elapsed time is 1.5, T is at $g_{i,b}$, and G_i is at the router. Since they are served simultaneously, the expected time until the next move is $\frac{1}{2}$. If G_i is the first to complete service, it moves to $g_{i,1-b}$ and it remains for T to be served at $g_{i,b}$ and at the router. If T is the first to complete service, it moves to the router, and it remains for G_i and T to be served at the router. In either case, the last two services take an expected time of 2, bringing the total to 4.

We now evaluate the weighted throughput of policy π^*. The n customers G_i are always busy, and they get one service completion per unit time, on the average. Hence, their contribution to the weighted throughput is nB. Regarding the test customer, it goes through a state of the form (s, i) once every 4 time units on the average. A fraction $1/f(n)$ of these visits correspond to times at which the computation has ended, and a further fraction p corresponds to accepting states, where p is the acceptance probability for the stochastic Turing machine. Hence, the weighted throughput under this policy is

$$nB + \frac{1}{4}\left(1 + \frac{\epsilon p}{f(n)}\right).$$

Recall that we have a "yes" instance of STOCHASTIC IN-PLACE ACCEPTANCE if and only if $p > \frac{1}{2}$; equivalently, if and only if the weighted throughput of policy π^* is larger than

$$nB + \frac{1}{4}\left(1 + \frac{\epsilon}{2f(n)}\right).$$

We will now argue that no other policy could achieve better weighted throughput. We need to consider all possible deviations from policy π^* and argue that they are not profitable.

(a) Because B is assumed very large, G_i must always stay busy. Whenever T and G_i are at the same server, we must always give priority to G_i.

(b) Given that the test customer is at server (s, i), would it ever pay to make a choice of b that is different than the choice under policy π^*? To see what happens in this case, suppose that G_i is at g_{i0} and T chooses $b = 1$ and goes to g_{i0}. We need one time unit for G_i to be served at g_{i0} and move to the router. If G_i moves from the router to g_{i1} before T is served at g_{i1}, either G will have to wait there (which is unprofitable) or T will never be able to start service at g_{i1}. Hence, we must first serve T at g_{i0} and at g_{i1} (two time units), while G_i keeps being served over and over at the router. Once T is finished at g_{i1}, we must wait for the next service completion for G_i at the router, and finally T must also be served at the router, which brings the total expected time to 5 (as compared to 4).

(c) Another possible deviation from π^* is to have some other customer G_j, $j \neq i$, move from some g_{jr} to the router and then to $g_{j,1-r}$. Since this will occupy the router for some time, it can only prolong the duration of T's excursion by an $O(1)$ amount.

(d) Finally, if the current state s is nonstochastic and the test customer chooses (s', i') in a way that does not maximize the acceptance probability in the Turing machine, the frequency p with which final states are accepting goes down, and the weighted throughput is reduced.

The conclusion is that if π^* is not followed, the mean duration of the test customer's excursion increases by a quantity of size $O(1)$. This is not enough to establish the optimality of π^*, for the following reason: it might still be profitable to violate π^* for some time in order to bring the network to a more favorable state that will result in much higher payoffs down the line. However, the only possible future payoff would be to bring the network to a state from which the loss due to a final nonaccepting state will become less likely. This payoff is at most ϵ, and with ϵ chosen small, the expected future payoff cannot exceed the price paid.

To make the preceding argument more rigorous, notice that we are dealing with an average reward dynamic programming problem, whose state evolves as a controlled Markov process $x(t)$, in which we wish to maximize

$$\lim_{t\to\infty} E\left[\frac{1}{t}\sum_{t_k \le t} r(t_k)\right],$$

where t_k is the time of the kth event and $r(t_k)$ is the reward obtained at that time. (In our case, rewards are associated with service completions that have positive weights.) Given a stationary policy π^* with average reward J, we define the corresponding value function by

$$V(x) = \lim_{t\to\infty} E\left[\sum_{t_k \le t} r(t_k) - Jt \,\middle|\, x(0) = x\right],$$

where the expectation is taken with respect to the probability distribution determined by π^*.

Let S be a subset of the state space which is recurrent under π^*. For any initial state $x(0)$,

let $t_{x(0)}$ be the first time that the state enters the set S from outside S. As a consequence of standard dynamic programming theory, if our policy π^* maximizes

$$E\left[\sum_{t_k \le t_{x(0)}} r(t_k) - Jt_{x(0)} + V(x(t_{x(0)}))\,\middle|\, x(0)\right]$$

for all initial states $x(0)$, then this policy is optimal. We wish to apply this result to our context.

Let S be the set of all states of the queueing network in which the test customer is at some server of the form (s, i). Consider now any initial state $x(0)$ and consider the particular policy π^* which we wish to prove optimal. Since the customers other than the test customer are always kept busy, we can ignore their rewards as well as their contribution to J. Let a^* be the average reward to the test customer under π^* (which is close to $\frac{1}{4}$). Recall also that there are no rewards received by the test customer while away from the set S. Thus, to verify that π^* is optimal, it suffices to establish that π^* maximizes $E[-a^* t_{x(0)} + V(x(t_{x(0)}))]$. This is indeed the case because, as argued earlier, any action that deviates from π^* increases the expected value of $t_{x(0)}$ by at least 1, whereas it is easily checked that on the set S the value function V does not vary by more than ϵ.

Having established that π^* is an optimal policy, it follows that we have a "yes" instance of STOCHASTIC IN-PLACE ACCEPTANCE if and only if the optimal performance criterion in the NETWORK OF QUEUES instance is larger than K, and the reduction is complete. □

3. Extensions and special cases. There are several variations of the problem NETWORK OF QUEUES that are also intractable; we review some of them below.

The following two results are obtained with minor modifications of the proof of Theorem 1.

COROLLARY 1. *NETWORK OF QUEUES remains EXP-complete under each one of the following alternative performance measures*:

(a) $\sum_c w(c) E[a_c^{\pi}(t^*)]$, *where* t^* *is a given deterministic terminal time.*

(b) $\sum_c \sum_{i=1}^{\infty} w(c) E[e^{-\alpha t_i(c)}]$, *where* α *is a positive discount rate and* $t_i(c)$ *is the* ith service completion time of some customer of class c. □

COROLLARY 2. *NETWORK OF QUEUES remains EXP-complete if the service times are deterministic (instead of exponentially distributed).* □

The following "dual" result can also be proved, using again a reduction from the STOCHASTIC IN-PLACE ACCEPTANCE problem. However, the encoding of the Turing machine and the specifics of the reduction are quite different.

THEOREM 2. *NETWORK OF QUEUES remains EXP-complete if the service times are random variables taking values in a finite range, even if* $R = \varnothing$ *(that is, there are no classes for which routing is stochastic).*

PROOF OUTLINE. The proof of membership in EXP is very similar to that in Theorem 1. To show completeness, we shall outline a construction reducing STOCHASTIC IN-PLACE ACCEPTANCE to this problem. We start with a version of the problem in which the machine always moves its tape head to the right, coming back to the left end immediately after it visits the right end (it is best to think of it as having a *circular* tape). The machine performs its main computation on the even-numbered squares, and keeps in the odd-numbered squares markings which identify by a 1 the current square—all other markings are zeroes. To simulate a leftward move, the machine goes all the way to the right, then starts at the leftmost square, guesses (using nondeterminism) the square to be visited, and checks that the guess was correct. Furthermore, each move of the machine as in the form "if the state is s and the

head scans the ith square and the ith square contains symbol r, then the next state is s';" or of the form "if the state is s and the head scans the ith square and the ith square contains symbol r, then symbol r' is written"—that is, at each step we allow only one of the two changes. This is not a loss of generality, since a change of both state and symbol can be accomplished in two stages.

Starting from any such machine, we shall construct a network of queues (S, C). The servers in S are divided (for the purpose of this exposition) into *clusters*, each cluster simulating a different part of the machine. As before, the customer classes are divided into partitions which we call *customers*; customers in the same partition are transformed from one class in the partition to another.

For each square of the machine we have a cluster of two servers, s_{i0} and s_{i1}, $i = 1, \ldots, n$, simulating the contents of the ith square. For the states we have a cluster $q_1, \ldots, q_k$ simulating the k states in K. We also have a cluster of n servers $h_1, \ldots, h_n$ simulating the head positions.

Each cluster of servers services at all times certain customers. One such server at all times is servicing the *main* customer of the cluster (its presence indicates that the machine indeed is in this state, scans this square, sees this symbol, etc.), and all others service *dummy* customers (just placeholders) or the *test* customer to be introduced later. The head cluster has no dummy customers. All dummy and main customers have deterministic service times on all machines in their cluster equal to 2. But they are not synchronized: The main customers start service at times 0, 2, 4, . . . , while the dummy customers at 1, 3, 5, $\cdots$ (we shall see that it is easy to start this).

Each tape square cluster s_{i0}, s_{i1}, for example, has two customers, a main and a dummy one. (Actually, the n tape square clusters have one main customer each, and a collective set of $n - 1$ dummy customers; the remaining queue will service a *test customer*, defined later.) Each customer returns to its server after it completes service, and so the queue that has the main customer, suppose it is s_{i1}, will continue to service it time after time until something extraordinary happens (namely the *test*, explained below). The intended meaning of this situation is that *the ith square contains symbol* 1.

The state cluster $q_1, \ldots, q_k$ has one main and $k - 2$ dummy customers. Again, if q_s is servicing the main customer between times $2t$ and $2(t + 1)$, this implies that the machine at step t is at state s. The kth customer serviced will again be a test customer.

The head cluster has a main customer, serviced for 2 units at queue h_1, then at h_2, then at h_3, and so on up to h_n, and then back to h_1 (recall the circular tape).

Recall that each move of the machine as a rule of the form "if the state is s and the head scans the ith square and the ith square contains symbol r, then the next state is s';" or of the form "if the state is s and the head scans the ith square and the ith square contains symbol r, then symbol r' is written"—that is, at each step we allow only one of the two changes. Each such rule is implemented by a *test customer* specific to that rule. This customer has very high weight, and is serviced for most of the time in its own dedicated server; however, it requires service at times 1, 3, 5, $\cdots$ for *zero* time units by any one these servers: q_s, h_i, and s_{ir}. If any one of the three servers q_s, h_i, and s_{ir} is free at the time (because its dummy customer has not arrived yet), then the test customer executes there for zero time, and is routed to its dedicated server, to return after 2 time units. But if all three are occupied, then they are occupied by main customers, *and thus the rule is applicable.* The test customer in this case requires service for 1 time unit on queue $q_{s'}$ and for another 1 on q_s, *thus reversing the role dummy/main of the customers of* $q_{s'}$ *and* q_s*—exactly as required.* In the case of a symbol-writing rule, the test customer performs in exactly the same manner a transition on s_{ir}.

So far we have been simulating a deterministic machine. To simulate our stochastic machine, we modify this construction as follows: For nondeterministic moves, say with next state either s' or s'', the test customers can be serviced, after q_s, either at $q_{s'}$ or at $q_{s''}$, and

the control decides which to choose. The stochastic steps are simulated by having a transition to $q_{s'}$, with service time a stochastic variable that is either 0 or 1, with probability half for each. Then the test customer is routed either to q_s, or to $q_{s''}$ and then to q_s, in both servers with deterministic service time 1. If the test customer completes service at $q_{s'}$ in zero time (this will happen an expected half of the time for stochastic transitions), the optimal routing is to $q_{s''}$ and then to q_s, thus reversing the role of s and s'', effecting a transition from state s to state s''. If however the test customer completes service at $q_{s'}$ in time 1, then the optimum routing is to go directly to q_s, thus changing the state from s to s'. Any other routing decision will cause the test customer to lose synchronization with the other customers, with a substantial loss in the weighted objective function.

We have not addressed two questions: First, how is this process started? Second, why can't we deviate from the above schedule, thus failing to simulate a computation? Both issues are taken care of as follows: We introduce a set of *elite* customers, one for each server, plus a set of new dedicated servers, one for each elite customer. The elite customers have exponentially higher weight (even when compared to the test customers). They require a service of zero time from each server, and then some of them go to their own dedicated servers, where they stay for 2^{n+1} steps, while certain others (those that correspond to the initial configuration) stay for another unit of time, thus starting the process. Only the elite customer that corresponds to the accepting state returns one unit earlier, to make sure that the computation ended up accepting. The proof that no deviation from the intended schedule is possible now can be structured as follows: To achieve the goal completion rate K, all elite customers must always be serviced with no delay (any deviation here prevents us from achieving K). Then, to get the next lower order bits of K right, the test customers must always be serviced with no delay. Finally, the remaining customers (main and dummy) must also be serviced as intended. We omit the full proof. □

If all sources of randomness are removed, then the problem can be solved in *polynomial space*. The reason is that the problem is now one of guessing a policy and testing that it indeed leads to a periodic behavior with the right performance (in terms of weighted throughput). Guessing such a policy, and evaluating its weighted throughput, can be done in nondeterministic polynomial space—which is well known to be the same as deterministic polynomial space. The next result in fact states that this special case is complete for polynomial space; its proof is similar to the proof of Theorem 2, except that, instead of a stochastic Turing machine, we are now simulating a space-bounded nondeterministic Turing machine.

THEOREM 3. *If $R = \varnothing$ (that is, there are no classes for which routing is stochastic) and the service times are deterministic, then the NETWORKS OF QUEUES problem is PSPACE-complete.* □

Out of all queuing control problems, there is a relatively small class for which optimal policies can be efficiently computed. These are problems involving a single server who chooses between one of several customer streams (Klimov 1974). The fundamental reason why these problems are solvable is that they can be reformulated as "branching bandits problems" (Weiss 1988), which is one of the successful extensions of the multi-armed bandit problem (Gittins 1989). Optimal policies in such problems can be found by computing a number of indices—known as Gittins indices—and there are polynomial time algorithms available for doing so. Thus, there may be some hope of enlarging the class of efficiently solvable queuing control problems, by deriving efficient solution procedures for other generalizations of the multi-armed bandit problem. The most interesting generalization that has been proposed so far is the "restless bandit problem" (Whittle 1988, Weber and Weiss 1990) and this raises the question whether the restless bandit problem is as "easy" as the original multi-armed bandit problem. Our results below establish that this is unlikely to be the case.

In the RESTLESS BANDITS problem we are given n Markov chains (bandits) $X_i(t)$, $i = 1, \ldots, n$, $t = 0, 1, \ldots$, that evolve on a common finite state space $S = \{1, \ldots, M\}$. We are also given the initial state of each chain. These Markov chains are coupled (and controlled) as follows. At each time t we choose one of the bandits, say bandit $i(t)$, to be played. For $i = i(t)$, $X_i(t+1)$ is determined according to a transition probability matrix P, and for every $i \neq i(t)$, $X_i(t+1)$ is determined according to some other transition probability matrix Q. At each time step, we incur a cost of the form

$$C(t) = c(X_{i(t)}) + \sum_{i \neq i(t)} d(X_i(t)),$$

where c and d are given rational-valued functions defined on the state space S. A *policy* π is a mapping $\pi : S^n \mapsto \{1, \ldots, n\}$ which at any time decides which bandit is to be played next, as a function of the states of the different bandits; that is, $i(t) = \pi(X_1(t), \ldots, X_n(t))$. Let the average expected cost of a policy be defined as

$$\limsup_{t \to \infty} \frac{1}{T} \sum_{t=1}^{T} E[C(t)].$$

We are interested in finding a policy with minimal average expected cost.

The classical multi-armed bandit problem is the special case of the above in which we have Q equal to the identity matrix and $d = 0$ (bandits not played do not move and do not incur any costs).

We shall actually show that the restless bandits problem is difficult even for the special case where the transition probability matrices P, Q correspond to deterministic transition rules, with one transition rule applying to all the bandits that are not played and another applying to the one which is played.

THEOREM 4. *RESTLESS BANDITS with deterministic transition rules is PSPACE-hard.*

PROOF. We will start from the following problem. We are given a circular tape, with squares $1, \ldots, n$; each square can take values in a set $\{1, \ldots, A\}$. At any given time, if the head is pointing at square i, then, the value v_i in that square is updated according to the rules $v_i := F(v_{i-1}, v_i)$ if $i > 1$, and $v_1 := F(v_n, v_1)$, where F is a known function satisfying $F(1, 1) = 1$; then the head moves to the next square (or to the first square if it is currently visiting the last one). That is, at time t the values are updated as follows:

$$v_i(t+1) = \begin{cases} v_i(t), & \text{if } i = t \bmod n; \\ F(v_{i-1 \bmod n}, v_i), & \text{otherwise.} \end{cases}$$

We say that this process "halts" if all squares have the value 1.

We claim that the problem of deciding, given an initial configuration of the tape, and a rule F, whether the process will eventually halt, is PSPACE-complete. To see why, start with a Turing machine computing in-place with a circular tape (such machines were argued in the proof of Theorem 1 to simulate arbitrary polynomial space-bounded machines). The symbols in $\{1, \ldots, A\}$ encode the square contents and the current state of the machine when the square was last entered, and at the next step. It is then clear that the next symbol of square i can be determined from the current symbols in squares i and $i - 1 \bmod n$.

For the reduction, we will describe a state space, a transition structure, and cost structure for a restless bandit problem in such a manner that solving the restless bandit problem is equivalent to solving the halting problem. A key idea is to introduce in the restless bandits

problem a high-cost trapping state and to arrange so that the the restless bandits can avoid the trapping states if and only if their evolution simulates the computation on the circular tape. We also set up the restless bandits problem so that the time-average cost is zero if and only if the computation on the circular tape halts.

There are n bandits, one for each square of the tape. The common state space of the bandits consists of a trapping state and of n consecutive "segments." Each segment contains states of the form (k, s) or (l, k, s), where $l, k, s \in \{1, \ldots, A\}$. These are arranged into A layers, with k serving as the layer index. The transition rules will be such that we always move from one layer to a state in the next layer (modulo nA).

All layers in any one of the segments $3, \ldots, n$ are fairly simple: they consist of A states; from a state s in one layer, we always move to state s in the next layer, as long as the bandit is not played (this serves to remember the value in the corresponding tape square); if the bandit is played, its next state is a trapping state with very high cost. This effectively prevents us from ever playing a bandit in segments $3, \ldots, n$.

In general, when the head is at cell i, bandit i will be traversing the first segment, bandit $i - 1$ the second segment, bandit $i - 2$ the third, etc., all the way to bandit $i + 1$ who will be traversing the nth segment. In particular, we only have to choose between playing bandit i or $i - 1$.

Let us consider bandit $i - 1$ who is traversing the second segment. Let (k, s), $s = 1, \ldots, A$, be the states in the kth layer of that segment. As with later segments, from state (k, s), we can only go to state $(k + 1, s)$. [If $k = A$, we move to a state $(1, s)$ in the first layer of the next segment.] If $k \neq s$, playing bandit $i - 1$ leads to a high-cost trapping state and, therefore, bandit i will be played. If on the other hand, $k = s$, a similar transition to a high-cost trapping state forces us to play bandit $i - 1$, and, therefore, not play bandit i.

Let us now look at the evolution of the state of bandit i who is traversing the first segment. At the first layer of the segment, the bandit's state is $(1, s')$ where s' is the value in the ith square of the tape. Recall that at each one of the A layers of that segment, that bandit will be played, with the exception of the sth layer, where s is the value in square $i - 1$. (This is the time k that bandit $i - 1$ is at state $(k, s) = (s, s)$ of the second segment.) So, let the state evolve from (k, s') to $(k + 1, s')$ each time that the bandit is played; on the other hand, if the bandit is not played, the state goes from (k, s') to $(k + 1, k, s')$; thereafter, for every $l \geq k$ and until the end of the segment, bandit i is played and the state moves from (l, k, s') to $(l + 1, k, s')$. Notice that the value of k for which bandit i was not played is equal to s, the value in square $i - 1$. Thus, the state (l, k, s') of bandit i encodes both the content s' of the ith square as well as the content s of square $i - 1$. At the end of the segment, bandit i ends up at state (A, s, s').

There is one exception to the above. If the value in square $i - 1$ is A, then bandit i is played for the first $A - 1$ layers and ends up in state (A, s'). Since this allows the ith bandit to infer that the value in cell $i - 1$ is A, we can identify state (A, s') with state (A, A, s').

At the end of the segment, the ith bandit makes a transition from state (A, s, s') to state $(1, F(s, s'))$ in the first layer of the second segment. In particular, the new state of bandit i is the correct new value in square i.

At this point, bandit i starts traversing the second segment and the process we have described is repeated, with bandits i and $i + 1$ playing the role of $i - 1$ and i, respectively. It is then clear that a policy that avoids getting into the high-cost trapping states must simulate the computation on the circular tape.

Besides the trapping state, let the costs per stage be unity at any state of the form (k, s) with $s \neq 1$, and zero at all remaining states. It is clear that the infinite-horizon average cost is positive except if every tape square eventually becomes equal to 1. But this is the same as the halting condition. We conclude that the average cost is zero if and only if the computation halts, thus proving PSPACE-completeness. □

Acknowledgments. We are grateful to the referees for their suggestions, including a simplification of the proof of Theorem 1. The first author's research was supported by an NSF grant. The second author's research was supported by the ARO under contract DAAL03-92-G-0115. A preliminary version of this paper was presented at the Ninth Annual Conference on Structure in Complexity Theory, June 1994, Amsterdam, The Netherlands.

References

Bertsimas, D., I. C. Paschalidis, J. N. Tsitsiklis 1994. Optimization of multiclass queueing networks: Polyhedral and nonlinear characterizations of achievable performance. *Ann. Appl. Probab.* **4** 43–75.

Chandra, A., D. Harel, D. Kozen 1981. Alternation. *J. Assoc. Comput. Mach.* **28** 114–133.

Fischer, M. J., M. O. Rabin 1974. Super-exponential complexity of Presburger arithmetic. R. M. Karp, ed. *Complexity of Computation.* SIAM-AMS Symposia in Applied Mathematics.

Gittins, J. C. 1989. *Multi-Armed Bandit Allocation Indices.* Wiley, New York.

Harrison, J. M. 1985. *Brownian Motion and Stochastic Flow Systems,* Prentice Hall, Englewood Cliffs, NJ.

Klimov, G. P. 1974. Time sharing service systems I. *Theory Probab. Appl.* **19** 532–551.

Kumar, S., P. R. Kumar 1994. Performance bounds for queuing networks and scheduling policies. *IEEE Trans. Automat. Control* **39** 1600–1611.

Puterman, M. L. 1994. *Markov Decision Processes.* Wiley, New York.

Papadimitriou, C. H. 1985. Games against nature. *Proc. of the 24th FOCS Conf.* 446–450; also *J. Comput. Systems Sci.* **31** 288–301.

——— 1994. *Computational Complexity.* Addison Wesley, Reading, MA.

Stockmeyer, L. J., A. R. Meyer 1973. Word problems requiring exponential time. *Proc. of the 5th STOC Conf.* 1–9.

Walrand, J. 1988. *An Introduction to Queueing Networks.* Prentice Hall, Englewood Cliffs, New Jersey.

Weiss, G. 1988. Branching bandit processes. *Probab. Engin. Inf. Sci.* **2** 269–278.

Weber, R. R., G. Weiss 1991. On an index policy for restless bandits. *J. Appl. Probab.* **27** 637–648; addendum in **23** 429–430.

Whittle, P. 1988. Restless bandits. *J. Appl. Probab.* **25A** 301–313.

C. H. Papadimitriou: University of California at San Diego, La Jolla, California; Current address: Computer Science Division, University of California at Berkeley, Berkeley, California; e-mail: christos@cs.berkeley.edu

J. N. Tsitsiklis: Department of EE and CS, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139; e-mail: jnt@mit.edu