Ninth Annual Conference on Structure in Complexity Theory, Amsterdam, The Netherlands, June 1994.

APRIL 1994

LIDS-P-2241

The Complexity of Optimal Queueing Network Control

Christos H. Papadimitriou
U. of California at San Diego
La Jolla, CA 92093

John N. Tsitsiklis
M.I.T.
Cambridge, MA 02139

Abstract

We consider the classical problem of optimal control (routing and sequencing) of a network of queues. We prove that this problem is EXP-complete and, therefore, provably intractable. Similar results are established for restricted versions of the problem. A weaker result is also established for the restless bandit problem.

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 — presumably based on the load of the other queues — so as to optimize weighted throughput.

Networks of queues have many applications. If you lived in the former Soviet Union, your whole life was a network of queues. Many of us are sitting many hours every day in front of a complex network of queues: a computer. But the most important applications of networks of queues are related to communication networks or manufacturing systems. There are preciously few cases of the problem that have been satisfactorily solved; for example, the problem is wide open even for the case of *two-server* networks and exponential service time distributions. The few problems that have been solved are reviewed in [10]. Besides some *ad hoc* techniques for very special cases, and computationally explosive dynamic programming algorithms for others, we can only solve certain single-server problems [6,11] 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 [4], 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 [5] and certain other rigorous approximation algorithms [2,7].

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

There are many such intractability results in the literature, starting from the classical ones about regular expression equivalence [9], Presburger arithmetic [3] 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. 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* (recall the formulation of polynomial *space* in [8] as polynomial *time* bounded stochastic computation; stochastic machines behave like alternating ones, transforming time to space and space to the next exponential of time). 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 is easily shown to be PSPACE-complete.

Given that the multi-armed bandit problem is the main tool for solving the few cases of networks of queues that we *can* solve, we study its most promising extension, the *restless bandit problem* [12,13], and show that it is PSPACE-complete even for deterministic problems.

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 class c customer completes service, it gets transformed into a customer of some new class c'. For each $c \in D$, we are given a set $N(c) \subset C$ 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 queueing 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 *non-preemptive policies*; that is, once a server starts serving a customer, it must continue until service is completed.

A queueing 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) 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 $N(c) \subset C$ 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 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 $J^\pi > K$.

Theorem 1. NETWORK OF QUEUES is EXP-complete.

Outline of the proof: For the upper bound, notice that the problem is a *Markov decision process*, that is, 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 it is known that if the optimal average cost is the same for any initial state, then such a problem can be reformulated as a linear program, and thus solved in polynomial time [1]. For the general case in which the optimal average cost may be different for different initial states, the problem is still solvable in polynomial time by means of linear programming techniques. 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 data.

To show completeness, we shall rely on a novel characterization of EXP. A *stochastic* Turing machine is a Turing machine whose states are divided into two parts: the *nondeterministic* states and the *stochastic* states; assume, without loss of generality, that all state-symbol combinations, except 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 $2^{|x|}$ steps. We say that the machine *accepts* an input x if in the (doubly exponential) tree of computations on input x, it has the following property: there is a way to choose one branch out of every nondeterministic node such that the *majority* of the leaves are accepting. We assume that the machine has only two tape symbols, 0 and 1.

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?

Lemma 1. STOCHASTIC IN-PLACE ACCEPTANCE is EXP-complete.

Sketch of proof: In [8] it was shown that stochastic Turing machines operating within polynomial time accept all of PSPACE. It is also known that alternating Turing machines with polynomial space comprise all of EXP. Using the same technique as in [8] starting from this result, we show that EXP is precisely the class of languages decided by stochastic Turing machines within polynomial space. To convert to linear space is standard (use padding), and to rephrase the problem in a form with no explicit input, just absorb the input in the machine itself. □

We shall reduce this 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 on M. We first assume that the start state is never visited again. Furthermore, given an arbitrary configuration of the Turing machine (state, head position, tape contents), the computation is guaranteed to end after at most 2^n steps. When the computation ends, the tape has again n zeroes, and the head is at the initial position. We finally assume that the transition function of the machine has been modified so that when an end state is reached, the machine does not halt but starts running again with the same initial configuration.

We shall now construct a network of queues. The different servers and customers in the network will be used to simulate different parts of the machine.

a) 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. However, if G_i wants to move from one of these two servers to the other, it must go through a special server that we call the *router* server.

b) If the machine has k states, we introduce servers $q_1, \ldots, q_k$, and a single customer Q that can only be served by one of these servers.

c) Finally, we have a set of servers $h_1, \ldots, h_n$, and a single customer H that can only be served by one of these servers.

We assume that the mean service time for the customers Q, H, and G_i, $i = 1, \ldots, n$, is unity at every eligible server. We will say that the network is in a *busy state* if each one of these customers is being served at some eligible server. A busy state in which no customer G_i is at the router server is called *definite*. Note that there is a one–to–one correspondence between definite states of the network and a configuration of the Turing machine, according to the following conventions: customer G_i is served at g_{i0} (respectively, g_{i1}) if and only if the ith square contains the symbol 0 (respectively, 1); customer Q is served at q_i if and only if the machine is at state i; customer H is served at h_i if and only if the head scans the ith square.

Each transition of the Turing machine can be viewed as a 6–tuple of the form $(s, i, r; s', i', r')$ where s is the current state, i is the current position of the head, r is the symbol in the ith square, s' is the new state, i' is the new position of the head, and r' is the symbol written on the tape. The rules of operation of the Turing machine can always be described by specifying for each (s, i, r), two different continuations (s', i', r'). (If s is a stochastic state, one of the two continuations is chosen at random; for nondeterministic states, we are free to choose one of the two options.)

We now introduce a *test* customer T whose properties will encourage the network to simulate the machine. There is a *base* server at which the test customer is served for zero time. Then, the test customer makes an excursion through the network along one of several possible routes, each possible route corresponding to one of the possible transitions $(s, i, r; s', i', r')$ of the Turing machine. Choosing a particular route is done as follows. The test customer first chooses (s, i, r). If $s \in D$, the test customer is also free to choose one of the two available choices for (s', i', r'). If $s \in R$, then (s', i', r') is chosen at random among the two possibilities. [In queueing network terms, these choices can be viewed as routing steps first to a server labeled by (s, i, r) and then to another server labeled by $(s, i, r; s', i', r')$.]

Once $(s, i, r; s', i', r')$ has been selected, the route to be followed during the excursion is the following:

a) Go through all of the servers $q_1, \ldots, q_k, h_1, \ldots, h_n$, except for q_s and h_i that are skipped; finally, go through server $g_{i,1-r}$.

b) Go through a *special* server.

c) Go through servers h_i, g_{ir}, and $q_{s'}$.

d) Go through the router server. Then, go through the servers $q_1, \ldots, q_k, h_1, \ldots, h_n$, except for servers $q_{s'}$ and $h_{i'}$ that are skipped. Also, go through server $g_{i,1-r'}$.

We assume that each service along the above described route takes zero time, with the following two exceptions: the mean service time at the special server is 1; also, the mean service time at server q_s in part (c) of the route is $1 + \epsilon$ if s is a final nonaccepting state and 1 otherwise. Here, ϵ is a very small number.

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 unity at the base server and zero elsewhere.

The network is initialized with the test customer at the base server, each customer G_i at server g_{i0}, customer Q at server q_i, where i is the start state of the Turing machine, and customer H at state h_i where i 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 queueing network and which simulates the stochastic Turing machine. (We will argue later that this policy is optimal.) Under policy π^*, the customers Q, H, and 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 the base server and that the network is at a definite state. Then, the test customer will make an excursion corresponding to the transition $(s, i, r; s', i', r')$ of the Turing machine. Let t_0 be the time that the excursion starts. As long as (s, i, r) corresponds to the current configuration of the Turing machine and as long as the network is simulating it correctly, the test customer will not meet any other customer during part (a) of its route and that part gets traversed in zero time. Part (b) takes unit expected time. By the memoryless property of exponential distributions, we may assume that customers Q, H and G_i started service at the servers q_s, h_i and g_{ir}, respectively, at time t_0. When each one of them completes service, it moves to a new server, as follows: Q goes to $q_{s'}$, H goes to $h_{i'}$, and G_i goes to $g_{ir'}$ via the router server. Thus, the time elapsed from time t_0, until the test customer can start service at q_s is equal to the maximum of four independent unit mean exponential random variables. When the test customer reaches the router server and if the service of G_i there has not been yet completed, the

test customer will have to wait there until the router server is free. The test customer will not experience any other delay during part (d) of the route, because it skips the servers to which Q, H and G_i went. At the end of the excursion, the network is in a new definite state that encodes the new configuration of the Turing machine. If s is not a nonaccepting final state, the mean duration of this excursion is always the same and equal to some (easily computable) rational number d; otherwise, the mean duration is $d+\epsilon$. The second possibility occurs with probability p, once every 2^n excursions where p is the fraction of nonaccepting leaves in the stochastic Turing machine being simulated. Since there are $n+2$ customers other than the test customer, the weighted throughput corresponding to this policy is equal to $B(n+2)+1/(d+\epsilon 2^{-n}p)$. Recall that we have a "yes" instance of STOCHASTIC IN–PLACE ACCEPTANCE if and only if $p < 1/2$; equivalently, if and only if the weighted throughput of policy π^* is larger than $K = B(n+2)+1/(d+\epsilon 2^{-n+1})$.

We will now argue that no other policy could have better weighted throughput. This will complete the reduction. We first note that since the weight given to the customers Q, H, and G_i is very large, an optimal policy must keep these customers busy all of the time.

No matter which route is chosen by the test customer, parts (a) and (c) of the route go through servers that are currently busy by customers Q, H and G_i for some i. The time that the test customer will have to wait at these servers will be minimized if these servers appear in part (c) of the route, right after the special server, and not in part (a). (The idea is that while the test customer is at the special server, Q, H and G_i have an opportunity to free up their respective servers.) Thus, the test server must choose a route that corresponds to a transition $(s,i,r;s',i',r')$ of the Turing machine or else suffer some additional delay. Also, customers Q, H, and G_i must go to servers $q_{s'}$, $h_{i'}$ and $g_{i'r'}$; otherwise, the test customer will suffer extra delay during part (d) of the route. 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 penalty 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. (This argument can made formal by using an easy lemma on optimality conditions in Markov decision theory.)

There is one last detail that must be dealt with. We define the class of the test customer to be what is left of its route until it returns to the base server. We also define the class of customers Q, H, and G_i, to be the server at which they happen to be located. In this way, the network that we have constructed here is an instance of NETWORK OF QUEUES and the reduction is now 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 result is also proved by a similar reduction of the STOCHASTIC IN–PLACE ACCEPTANCE PROBLEM. However, the encoding of the Turing machine and the specifics of the reduction are quite different. The proof will be provided in the full paper.

Theorem 2: The problem remains EXP–complete if the service times are random variables taking values in a finite range, even if routing is entirely deterministic.

If all sources of randomness are removed, however, the problem may become much easier. The proof of the next result is similar to the proof of Theorem 1, except that instead of a stochastic Turing machine, we are now simulating a space–bounded deterministic Turing machine.

Theorem 3: If routing and the service times are deterministic, then the problem is PSPACE–complete.

A special case of the queueing network problem is obtained if we put some restrictions on the mechanism with which the customers change classes. More precisely, let us assume that there are several *types* of customers and that the type of a customer never changes. Let us then assume that the class of a customer is a pair consisting of the customer's type and the customer's location in the network. Note that this restriction is violated by the network that we constructed in the proof of Theorem 1. (For example, the test customer may visit the same server in parts (a) and (d) of its route, but its class will be different at each visit.) However, we can show that Corollary 2 and Theorems 2–3 still hold.

Out of all queueing 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 [6]. The fundamental reason why these problems are solvable is that they can be reformulated as "branching bandits problems" [11], which is one of the succesfull extensions of the multi-armed bandit problem [4]. 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 queueing 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" [12,13] 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 $S = \{1, \ldots, M\}$. 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. □

Acknowledgment: The second author wishes to thank Dimitris Bertsimas for discussions on the subject. The second author's research was supported by the ARO under contract DAAL03–92–G–0115.

References

1. D. P. Bertsekas, *Dynamic Programming*, Academic Press, 1987.
2. D. Bertsimas, I. C. Paschalidis, and J. N. Tsitsiklis, "Optimization of Multiclass Queueing Networks: Polyhedral and Nonlinear Characterizations of Achievable Performance", to appear in the *Annals of Applied Probability*, 1994.
3. M. J. Fischer and M. O. Rabin "Super-exponential complexity of Presburger arithmetic," *Complexity of Computation* (R. M. Karp, ed.), SIAM-AMS Symp. in Applied Mathematics, 1974.
4. J. C. Gittins, *Multi-Armed Bandit Allocation Indices*, J. Wiley, New York, 1989.
5. J. M. Harrison, *Brownian Motion and Stochastic Flow Systems*, Prentice Hall, 1985.
6. G. P. Klimov, "Time sharing service systems I", *Theory of Probability and Applications*, Vol. 19, 1974, pp. 532–551.
7. S. Kumar, P.R. Kumar, "Performance bounds for queueing networks and scheduling policies", to appear in the *IEEE Trans. on Automatic Control*, 1994.
8. C. H. Papadimitriou "Games against nature," *Proc. of the 24th FOCS Conf.*, pp. 446–450; also *J.CSS 31*, pp. 288–301, 1985.
9. L. J. Stockmeyer and A. R. Meyer "Word problems requiring exponential time," *Proc. of the 5th STOC Conf.*, pp.1 –9, 1973.
10. J. Walrand, *An Introduction to Queueing Networks*, Prentice Hall, Englewood Cliffs, NJ, 1988.
11. G. Weiss, "Branching bandit processes", *Probability in the Engineering and Informational Sciences*, Vol. 2, 1988, pp. 269-278.
12. R. R. Weber nd G. Weiss, "On an index policy for restless bandits", *J. of Applied Probability*, Vol. 27, 1990, pp. 637–648; addendum in Vol. 23, 1991, pp. 429–430.
13. P. Whittle, "Restless bandits: activity allocation in a changing world", in *A Celebration of Applied Probability*, J. Gani (ed.), *J. Applied Probability*, Vol. 25A, 1988, pp. 287–298.