Algorithmic challenges in the theory of queueing networks

David Gamarnik *
Notes prepared by Shubham Gupta †

October 25, 2011

Abstract

The theory of queueing systems is traditionally considered as a branch of applied probability. Hence the toolkit used in the analysis of queueing system draws heavily on the theory of stochastic processes. Many problems in the area of queueing networks, however, are of algorithmic nature, and thus require algorithmic/complexity theoretic approaches. In this tutorial we will discuss several such questions, ranging from some older results on designing optimal control of queueing networks, to more recent results on determining stability properties of queueing networks leading to the fascinating theory of algorithmic undecidability (non-computability). We will illustrate these approaches on a broad scope of models, including multiclass queueing networks, constrained random walks and Skorokhod mapping problem, the latter arising in studying queueing networks in the heavy traffic regime.

Contents

*MIT; e-mail: gamarnik@mit.edu.
†MIT; e-mail: shubhamg@mit.edu.

1 Languages, algorithms and Turing Machine

We start by discussing the notions of "*tractability*" and "*undecidability*". We formalize the notion of the complexity of an algorithm for solving a problem and the question of what can be solved, and what cannot.

A *Turing machine* (TM henceforth) is an abstract model of a general purpose computer (similar to a finite automaton but with an infinite and unrestricted memory). They can be classified as deterministic or nondeterministic.

Turing Machines (TM). A Turing machine is a 7-tuple [Sip97], $(\mathcal{S}, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where $\mathcal{S}, \Sigma, \Gamma$ are all finite sets and

1. $\mathcal{S}$ is the set of states,
2. Σ is the input alphabet not containing the special $BLANK$ symbol,
3. Γ is the tape alphabet which is defined to be Σ plus the special $BLANK$ symbol.
4. $\delta : \mathcal{S} \times \Gamma \to \mathcal{S} \times \Gamma \times \{L, R\}$ is the transition function. Here L and R wills stand for "move left" and "move right" commands.
5. $q_0 \in \mathcal{S}$ is the start state, $q_{\text{accept}} \in \mathcal{S}$ is the accept state and $q_{\text{reject}} \in \mathcal{S}$ is the reject state.

Given this Turing Machine operates as follows. It begins with state q_0. The infinite tape contains a sequence of elements of σ followed by an infinite sequence of $BLANKs$. For example, say $\Sigma = \{a, b, c\}$ and $\mathcal{S} = \{q_0, q_1, q_{\text{accept}}, q_{\text{reject}}\}$. Then the tape configuration can be $a, a, a, b, a, c, c, a, a, BLANK, BLANK, BLANK, \ldots$. A reading/writing device, called HEAD sits on top of the first symbol which in our case happens to be a. To indicate this we write $*a, a, a, b, a, c, c, a, a, BLANK, BLANK, BLANK, \ldots$ (star in front of the first symbol). At every moment of time the TM looks into the current state and the current element of Γ, and uses the δ function to update. Specifically, it uses δ to update the state to a new state, rewrite the current element of Γ by a new element of Γ and uses L and R to decide whether to move to the left or to the right. See Figure 1.

Say in our example, $\delta : (q_0, a) = (q_1, b, R)$. Then the state becomes q_1, the first symbol becomes b and the HEAD is on top of the second symbol. We obtain $b, *a, a, b, a, c, c, a, a, BLANK, BLANK, BLANK, \ldots$. We continue this process until either we reach state q_{accept}, in which case we stop and output $ACCEPT$, or reach state q_{reject}, in which case we stop and output $REJECT$. If neither of the terminating states $q_{\text{accept}}, q_{\text{reject}}$ is reached, the TM runs forever.

The sequence $a, a, a, b, a, c, c, a, a, BLANK, BLANK, BLANK, \ldots$ is what we call problem instance. Any countable set A of instances (strings) is called *language*, which for us means a collection of instances satisfying some properties. Every instance comes with either YES label or NO label and the corresponding subsets are denoted A_{YES}

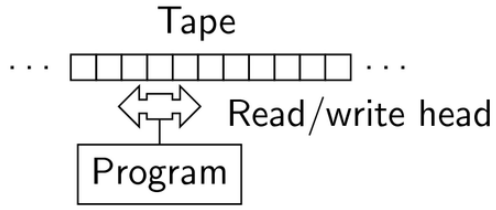

Figure 1: Illustration of a Turing Machine.

and A_{NO}. For example a language can be encoding collection of connected graphs. Given a language we say that a given Turing Machine is an *Algorithm* for A if for every sequence in A_{YES} the TM stops after finite time and outputs $ACCEPT$, and for every sequence in A_{NO} the TM stops after finite time and outputs $REJECT$. Thus TM is capable of determining in finite time whether we have an instance satisfying the needed property, for example whether the given graph is connected. Namely, A is a family of graphs, A_{YES} is the family of connected graphs and A_{NO} is the family of unconnected graphs.

Nondeterministic Turing Machines (NTM). We will not be working with this notion a lot, but I still provide it for reference. The transition function for NTM has the form $\delta : \mathcal{S} \times \Gamma \to \mathcal{P}(\mathcal{S} \times \Gamma \times \{L, R\})$. In a DTM, the set of rules prescribes at most one action to be performed for any given situation. A NTM, by contrast, may have a set of rules that prescribes more than one action for a given situation.

Example. Let us build a Turing machine which determines whether a given sequence of letters a is even. We set $\Sigma = \{a\}, \mathcal{S} = \{q_0, q_1, q_{\text{accept}}, q_{\text{reject}}\}$. Set

$$\begin{aligned}
&\delta(q_0, a) = (q_1, a, R),\\
&\delta(q_1, a) = (q_0, a, R),\\
&\delta(q_0, Blank) = (q_{\text{accept}}, \cdot, \cdot),\\
&\delta(q_1, Blank) = (q_{\text{reject}}, \cdot, \cdot),
\end{aligned}$$

We do not define δ on states $q_{\text{accept}}, q_{\text{reject}}$ as on these states the TM stops. For the same reason we have dots in the last two cases. Now it is easy to see that if we have a string $a, a, \cdots, a, BLANK, BLANK, \cdots$, then the TM will end in q_{accept} if the number of a-s in the string is even, and ends in q_{reject} otherwise.

Exercise. Show that TM M recognizes the language consisting of all strings of zeros whose length is a power of 2: $A = \{0^{2^n} \mid n \geq 0\}$.

2 Complexity Classes

Based on the amount of time and space that problems require to be computed on a Turing Machine, various complexity classes can be defined.

P is the class of languages that are decidable in polynomial time by a Turing Machine, whereas NP is the class of languages that are decidable in polynomial time on a Non-deterministic TM. Polynomiality is measured in terms of the size of the instance which is defined to be the length of the string in A. So in our example of strings of a-s, the running time of the algorithm is the same as the length of the string, so it is linear and therefore polynomial time algorithm. An example of a problem in P is the Shortest Path problem: given a graph $G = (V, E)$ and two nodes u, v find the shortest path from u to v. Another example is Maximum Matching Problem: given a graph G find the cardinality of a largest matching. To be exact, the decision problem which is in P is given a graph G and number K determine whether there exists a matching of size at least K. Examples of problems in NP include various combinatorial optimization problems such as Independent Set (IS), CLIQUE, SUBSET-SUM and many others. Independent set of a graph $G = (V, E)$ is a set of nodes $I \subset V$ such for no pair of nodes $i, j \in I$ is an edge, that is $(i, j) \notin E$.

PSPACE is the class of languages that are decidable in polynomial space by a TM. Examples include the so-called True Fully Quantified Boolean Formula. Finally, EXPTIME is the class of languages that are decidable in exponential time on a DTM. The hierarchy is,

$$\text{P} \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \subseteq \text{NEXPTIME} \subseteq \text{EXPSPACE} \tag{1}$$

Another important and relevant for us complexity class is $\#P$. Intuitively this class corresponds to problems which can be counted by non-deterministic TM. Example of a problem in $\#P$ is the problems of counting the number of independent sets in the graph. To be exact given a graph $G = (V, E)$ and number K determine whether the number of independent sets in G is at most K. A major open question along with whether $P = NP$ is whether $P = \#P$, namely problems in $\#P$ can be solved using deterministic TM which run in polynomial time.

NP-hard. NP-Hard in computational complexity theory, is a class of problems that are, informally, "at least as hard as the hardest problems in NP".

- A problem H is NP-hard if and only if there is an NP-complete problem L that is polynomial time Turing-reducible to H (i.e., $L \leq TH$). In other words, L can be solved in polynomial time by an oracle machine with an oracle for H. Informally, we can think of an algorithm that can call such an oracle machine as a subroutine for solving H, and solves L in polynomial time, if the subroutine call takes only one step to compute.
- Alternatively, a problem H is NP-hard if all problems in NP are polynomial time reducible to it.

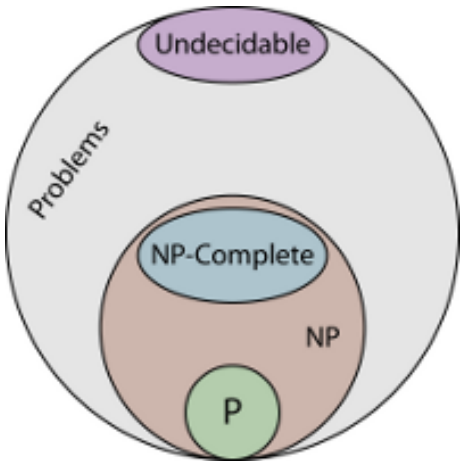

Figure 2: Illustration of P, NP and Undecidable.

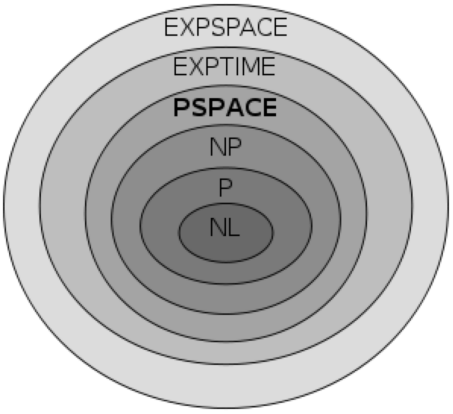

Figure 3: Hierarchy of various complexity classes.

NP-hard problems may be of any type: decision problems, search problems, or optimization problems. The NP-hard nomenclature is sort of misleading because NP-Hard problems are not necessarily in NP. Nonetheless, this notation is well-itched into the research community and unlikely to be changed.

EXP-hard. In computational complexity theory, the complexity class EXPTIME (sometimes called EXP) is the set of all decision problems solvable by a Deterministic Turing Machine in $O(2^{p(n)})$ time, where $p(n)$ is a polynomial function of n, and n is the length of the input.

A decision problem is EXPTIME-complete if it is in EXPTIME, and every problem in EXPTIME has a polynomial-time many-one reduction to it. In other words, there is a polynomial-time algorithm that transforms instances of one to instances of the other with the same answer. Problems that are EXPTIME-complete might be thought of as the hardest problems in EXPTIME. Notice that although we don't know if NP is a subset of P or not, *we do know that EXPTIME-complete problems are not in P*; it has been proven that these problems cannot be solved in polynomial time, by the time hierarchy theorem.

Examples of EXP-Complete problems: evaluating a position in generalized chess, Checkers, Go, and as we will see optimal control of closed queueing networks.

3 Undecidability

In computability theory and computational complexity theory, an *undecidable* problem is a decision problem for which it is **impossible to construct a an algorithm (Turing Machine) that always leads to a correct yes-or-no answer**. The most famous example of an undecidable problem is the so-called Halting Problem which we discuss below. Other examples include PCP (Post Correspondence Problem), Hilbert's tenth problem, that is whether a Diophantine equation (multivariable polynomial equation) has a solution in integers, and many others. There is a wealth of interesting undecidable problem and I recommend a recent article [GS10] for a non-technical survey of the topic, and [BT00] for a technical survey of hard and undecidable problems in the area of control theory.

One way to get some intuition behind the concept of undecidability is perhaps to make a comparison with the classification problem in learning. Consider the classification problems which are decidable by the set of *linear classifiers*. For a particular dataset, one can think of the set of all linear classifiers as the set of admissible functions from which a mapping to YES/NO is desired. There exist datasets for which there does not exist a mapping from this class of functions to correct YES/NO answers. Thus, these datasets are undecidable by the class of linear classifiers. In a similar vein, one can think of the class of functions induced by the set of Turing machines. Consequently, the set of problems for which there is a mapping from the TM induced functions to YES/NO answers are *decidable* and those for which there cannot exist a mapping are *undecidable*.

Alan Turing proved in 1936 his famous Halting Problem theorem, which roughly speaking says that an algorithm running that solves the halting problem for all possible program-input pairs cannot exist. That is there is no algorithm which when takes a pair of string and TM needs to output whether the TM will halt on this string, cannot exist. Hence, the halting problem is undecidable. The proof of the undecidability of the halting problem uses a technique called *diagonalization*, which was also used by George Cantor in 19th century to prove that the set of real values is uncountable. We now provide a brief overview of the proof idea.

Theorem 1. *The following language which we call HALTING PROBLEM is undecidable:*

$$A_{TM} = \{(M, w) |\ w \text{ is a string, } M \text{ is a TM, and } M \text{ accepts } w\} \tag{2}$$

Proof. We assume that A_{TM} is decidable and obtain a contradiction. Suppose that H is a TM which is a decider for A_{TM}, i.e.,

$$H(\langle M, w\rangle) = \begin{cases} accept, & \text{if } M \text{ accepts } w, \\ reject, & \text{if } M \text{ does not accept } w. \end{cases}$$

We construct a new TM D with H as a subroutine with the following mechanics on input $\langle M\rangle$, where M is a TM. Here $\langle M\rangle$ is a description of the TM M written using some appropriate alphabet Γ.

- Run H on input $\langle M, \langle M \rangle \rangle$.
- Output the opposite of what H outputs; i.e., if H accepts, reject and if H rejects, accept.

In summary,

$$D(\langle M \rangle) = \begin{cases} accept, & \text{if } M \text{ rejects } \langle M \rangle, \\ reject, & \text{if } M \text{ accepts } \langle M \rangle. \end{cases}$$

Now, when D is run with its own description:

$$D(\langle D \rangle) = \begin{cases} accept, & \text{if } D \text{ does not accept } \langle D \rangle, \\ reject, & \text{if } D \text{ accepts } \langle D \rangle. \end{cases}$$

This is a contradiction, therefore, neither TM D nor TM H can exist.

Here is an illustration of the same argument.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	...
M_1	a		a	
M_2		a	a	
M_3	a	a		
$\vdots$				

Table 1: Entry i, j is "a" if M_i accepts $\langle M_j \rangle$.

Table 2 contains the results of running H on inputs corresponding to Table 1.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	...
M_1	a	r	a	r
M_2	r	a	a	r
M_3	a	a	r	r
$\vdots$	r	r	r	r

Table 2: Entry i, j is the value of H on $\langle M_i, \langle M_j \rangle \rangle$.

Since, H and D are TMs, they must occur amongst $M_1, M_2, \ldots$. Since, D computes the opposite of the diagonal entries, the contradiction occurs at the point of question mark.

□

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	...	$\langle D \rangle$
M_1	a	r	a	r	
M_2	r	a	a	r	
M_3	a	a	r	r	$\vdots$
$\vdots$	r	r	r	r	
D		...			?

Table 3: If D is in the figure, a contradiction occurs at ?.

4 Intractability of performance analysis and optimization of queuing networks

4.1 The Complexity of Optimal Queueing Network Control by Papadimitriou and Tsitsiklis [PT94]

The authors in [PT94] prove that the problem of *optimal control (routing and sequencing) of a network of queues* is EXP-complete and therefore, provably intractable. A weaker result is also established for the *restless bandit problem*, namely, it is PSPACE-hard.

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 $\sigma(c) \in S$ and the mean service time $\mu(c)$. The service times are assumed to be exponentially distributed i.i.d. The set C is partitioned into two subsets R and D. $\forall c \in D$, we are given a set $N(c) \in C$. $\forall c \in R$, the new class $c^{'}$ is determined at random according to given probabilities $p_{cc'}$.

The queueing network described here is *closed*. This means that there is a fixed set of jobs permanently trapped in the system. The *state* is: 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 for routing of jobs in classes C and servicing jobs in all classes. We consider the family of non-preemptive policies. Namely, we servers cannot interrupt servicing one job and start servicing another job. Thus service decisions are made only after service completions. We assume that idling is allowed. Let $a_c^{\pi}(t)$ be the number of class c service completions until time t.

$$J^{\pi} = \limsup_{t \to \infty} \frac{1}{t} \sum_{c \in C} w(c) E[a_c^{\pi}(t)], \tag{3}$$

Goal. Goal is to find a policy that maximizes J^{π}. Formally, the problem is given a rational number K, decide whether there exists a policy π for which $J^{\pi} > K$.

Theorem 2. *NETWORK OF QUEUES problem is EXP-complete.*

This theorem was proved via reduction from the so-called STOCHASTIC IN-PLACE ACCEPTANCE problem. We skip details of the problem statement and the

reduction.

4.2 Restless Bandit problem

RESTLESS BANDITS. We are given n Markov chains (bandits) $X_i(t), i = 1, \ldots, n, t = 0, 1, \ldots$ that evolve on a finite state space $S = \{1, \ldots, M\}$. At each time t, bandit $i(t)$ is played. For $i = i(t)$, $X_i(t+1)$ is determined according to a transition probability matrix P and all other i according to some other transition probability matrix Q. We incur a cost:

$$C(t) = c(X_{i(t)}) + \sum_{i \neq i(t)} d(X_i(t)), \tag{4}$$

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

MULTI-ARMED BANDIT. Special case of the above in which we have $Q = I$ and $d = 0$ (bandits not played do not move and do not incur any costs).

In the same paper [PT94] the following result is established.

Theorem 3. *RESTLESS BANDITS with deterministic transition rules is PSPACE-hard.*

4.3 Performance of wireless networks

We now use $\#P$ complexity class to establish intractability of the performance analysis of some loss networks which one can think of idealized wireless network. Consider a wireless communication network as a graph $G = (V, E)$. Every node $i \in V$ of the graph is associated with a rate ν_i, Poisson process corresponding to wireless calls corresponding to some fixed frequency h. The wireless call is accepted if and only if there are no prior wireless calls originating from this node and furthermore, no (graph-theoretic) neighbors of i have calls in progress. The latter requirement is to ensure that calls using the same frequency h should not interfere and a link between two nodes means the two nodes are within the interference proximity range. Assume each call on node i has an exponentially distributed length with rate equal to unity (assumed for simplicity, though since we will establish certain hardness result, this restriction is strength not a weakness of the assumption). If the arriving call cannot be accepted, it is dropped. It is not hard to see then that the set $I(t)$ of calls in progress at time t is an independent set in G. Using the reversibility theory, it one can show that the stationary distribution π exists, is unique and has the following simple product form. For every independent set I,

$$\mathbb{P}_\pi(I) = Z^{-1} \prod_{i \in I} \nu_i,$$

where Z is the normalizing constant defined by

$$Z = \sum_{I} \prod_{i \in I} \nu_i,$$

where the sum is over all independent sets. This explicit form of the stationary distribution is deceiving, as in order to compute it we need to compute Z which involves a sum of potentially exponentially many independent sets in G. Thus we formulate the following decision problem which we call *Performance of Loss Network*: given a graph $G = (V, E)$, rates $\nu_i, i \in V$ and number K, determine if $Z \leq K$. Considering the special case $\nu_i = 1$, we see that Z is nothing else but the number of independent sets in our communication graph G. We obtain

Theorem 4. *The problem* Performance of Loss Network *is #P-complete.*

A variant of this theorem and many extensions was established by Louth, Mitzenmacher and Kelly [LMF94]. Analysis of wireless networks using independent sets problem and its variants is a very active area of research now [SS],[MSZ06],[UNS09],[STT]

5 Constrained homogeneous random walks and undecidability

Let us now introduce a much simpler variant of the Turing Machine and Halting Problem which will be useful for our analysis of constrained random walks, queueing networks and Skorokhod problem.

5.1 Counter Machine and Halting Problem

A Counter Machine which we define below is a deterministic computing machine which is a simplified version of a Turing Machine A Counter Machine is described by 2 counters R_1, R_2 and a finite collection of states $\mathcal{S} = \{1, 2, \ldots, m\}$. Each counter R_i contains some nonnegative integer in its register. Depending on the current state $i \in \mathcal{S}$ and depending on whether the content of the registers is positive or zero, the Counter Machine is updated as follows: the current state i is updated to a new state $j \in \mathcal{S}$ and one of the counters has its number in the register incremented by one, decremented by one or no change in the counters occurs. More specifically, a Counter Machine is a pair $(\mathcal{S} = \{1, \ldots, m\}, \Gamma)$. where Γ is configuration update function $\Gamma : \mathcal{S} \times \{0,1\}^2 \to \mathcal{S} \times \{(-1,0),(0,-1),(0,0),(1,0),(0,1)\}$. A configuration of a Counter Machine is an arbitrary triplet $(i, z_1, z_2) \in \mathcal{S} \times \mathbb{Z}_+^2$. A configuration (i, z_1, z_2) is updated as follows. Given (i, z_1, z_2) suppose $\Gamma(i, \mathbf{1}\{z_1 > 0\}, \mathbf{1}\{z_2 > 0\}) = (i', 1, 0)$. Then the current state is changed from i to i', the content of the first counter is incremented by one and the second counter does not change: $z_1' = z_1 + 1, z_2' = z_2$. We will also write $\Gamma : (i, z_1, z_2) \to (i', z_1 + 1, z_2)$. Suppose, on the other hand, $\Gamma(i, \mathbf{1}\{z_1 >$

$0\}, \mathbf{1}\{z_2 > 0\}) = (i', -1, 0)$. Then the current state becomes i', $z_1' = z_1 - 1, z_2' = z_2$. Similarly, if $\Gamma(i, \mathbf{1}\{z_1 > 0\}, \mathbf{1}\{z_2 > 0\}) = (i', 0, 1)$ or $\Gamma(i, \mathbf{1}\{z_1 > 0\}, \mathbf{1}\{z_2 > 0\}) = (i', 0, -1)$, the new configuration becomes (i, z_1, z_2+1) or (i', z_1, z_2-1), respectively. If $\Gamma(i, \mathbf{1}\{z_1 > 0\}, \mathbf{1}\{z_2 > 0\}) = (i', 0, 0)$ then the state is updated to i', but the contents of the counters do not change. It is assumed that the configuration update function Γ is consistent in the sense that it never attempts to decrement a counter which is equal to zero. The present definition of a Counter Machine can be extended to the one which incorporates more than two counters, but such an extension is not needed for our purposes.

Given an initial configuration $(i^0, z_1^0, z_2^0) \in \mathcal{S} \times \mathbb{Z}_+^2$ the Counter Machine uniquely determines the subsequent configurations $(i^1, z_1^1, z_2^1), (i^2, z_1^2, z_2^2), \ldots, (i^t, z_1^t, z_2^t), \ldots$. We fix a certain configuration $(i^*, z_1^*, z_2^*) \in \mathcal{S} \times \mathbb{Z}_+^2$ and call it the *halting* configuration. If this configuration is reached then the process halts and no additional updates are executed. The following theorem establishes the undecidability (also called non-computability) of the halting property. It is a classical result and can be found in [Hoo66].

Theorem 5. *Given a Counter Machine $(\mathcal{S}, \Gamma)$, initial configuration (i^0, z_1^0, z_2^0) and the halting configuration (i^*, z_1^*, z_2^*), the problem of determining whether the halting configuration is reached in finite time (the Halting Problem) is undecidable. Without the loss of generality it may be assumed that $z_1^* = z_2^* = 0$.*

5.2 Constrained Random Walk.

Constrained random walks in non-negative orthant $\mathbb{Z}_+^d$ are generalization of stochastic processes arising in studying certain types of queueing systems. Consider, for example, a multiclass queueing network with exponentially distributed interarrival times, service times, and probabilistic routing where each server uses a preemptive static buffer priority policy as a scheduling rule (see Section 4 as well as [Dai95],[DW96],[GKR09] for formal definitions of multiclass queueing networks). The vector of queue length $Q(t)$ observed at event times (arrivals or service completions) is a Markov chain in $\mathbb{Z}_+^d$, where d is the number of classes. It has the property that the transition probabilities depend only on which buffers are empty and which buffers are non-empty, but does not depend on the actual size of queues in buffers. This motivates the following definition which goes back to the papers by Malyshev and his co-authors [Mal72b],[Mal72a],[Men74],[Mal93],[FMM95].

Markov chain $Q(t), t = 1, 2, \ldots$ has a nonnegative orthant $\mathbb{Z}_+^d$ as its state space. For each $\Lambda \subset \{1, 2, \ldots, d\}$, let $\mathbb{Z}_\Lambda$ be the corresponding face:

$$\mathbb{Z}_\Lambda = \{(z_1, \ldots, z_d) \in \mathbb{Z}_+^d : z_i > 0 \text{ for } i \in \Lambda, z_i = 0 \text{ for } i \notin \Lambda\}. \tag{5}$$

The transition probabilities depend entirely on the face, which the random walk currently belongs to, and the transition vectors have at most unit length in max norm.

$$\sum_{\Delta \in \{-1,0,1\}^d} p(\Lambda, \Delta) = 1, \tag{6}$$

This is motivated by the fact that the queue length change at most by one at the transition epochs. Given a current state $Q(t)$, the next state is chosen to be $Q(t) + \Delta$ with probability $p(\Lambda, \Delta)$, if the state $Q(t)$ belongs to the face $\mathbb{Z}_\Lambda$. The following variation, called *deterministic homogeneous walk in a nonnegative orthant*, is one for which $p(\Lambda, \Delta) \in \{0, 1\}$ will be important for us.

For a given initial state $Q(0)$, the constrained random walk is defined to be *stable* if there exists some $C > 0$ such that the random walk visits the set $\{z \in \mathbb{Z}_+^d : \sum_{i=1}^d z_i \leq C\}$ i.o. w.p 1. and the expectation of the time intervals between the visits is finite. Namely, the walk is stable if it contains positive recurrent states. It is easy to see that if the walk is deterministic, then stability is equivalent to the condition that $\sup_t \|Q(t)\|_1 < \infty$, where $\|Q(t)\|_1 = \sum_{1 \leq i \leq d} Q_i(t)$ (in fact any norm can be used in this definition).

The following result was established in [Gam02].

Theorem 6. *Stability of a deterministic homogeneous walk in $\mathbb{Z}_+^d$ is undecidable.*

Proof. We prove the theorem by a simple reduction from a counter machine. CM $(\mathcal{S}, \Gamma)$ with initial configuration (s^0, z_1^0, z_2^0) and halting configuration $(s^*, 0, 0)$, we will construct a deterministic walk that has dynamics very similar to the dynamics of a counter machine. Subsequently, if we had an algorithm for checking stability of a deterministic walk we could use this algorithm for checking whether the counter machine halts.

Let $\mathcal{S} = \{1, 2, \ldots, m\}$ and let $i^* \in \{1, 2, \ldots, m\}$ be the halting state. That is the halting configuration is $(i^*, 0, 0,)$. Our deterministic walk has a state space $\mathbb{Z}_+^{m+2}$. The first m coordinates will be used to encode the states of the counter machine. We encode the state $i \in \mathcal{S}$ by an m-dimensional unit vector e_i. The last two coordinates contain the values of the counters of the counter machine. Thus, the configuration (i, z_1, z_2) corresponds to the following state $q = (q_1, \ldots, q_{m+2}) \in \mathbb{Z}_+^{m+2}$ of the walk: $q_i = 1, q_{m+1} = z_1, q_{m+2} = z_2$, and $q_j = 0$ for all other coordinates j.

We now describe the transition vectors Δ for each face $\mathbb{Z}_\Lambda$ of the state space $\mathbb{Z}_+^{m+2}$.

- Suppose $\Lambda = \{i, m+1, m+2\}$ for some $i \in \{1, 2, \ldots, m\}$. Suppose $\Gamma(i, (1, 1)) = (j, (1, 0))$ and $i \neq j$. Namely, the state needs to become j, the first counter needs to increment by one and the second counter needs to remain the same. Then we set $\Delta_i(\Lambda) = -1, \Delta_j(\Lambda) = 1, \Delta_{m+1}(\Lambda) = 1$, and $\Delta_k(\Lambda) = 0$ for all other k. This simply means that if the current state $Q(t)$ of the walk encodes a state i and both coordinates $m+1, m+2$ correspond to positive contents, then $Q(t+1) = Q(t) + \Delta(\Lambda)$ will encode the state j, and the coordinate $m+1$ increases its value by 1. If $i = j$, then we set $\Delta_{m+1}(\Lambda) = 1, \Delta_i(\Lambda) = 0$ for all other i. This corresponds to the case when the state i does not change.
- If $\Lambda = \{i, m+1\}$ or $\Lambda = \{i, m+2\}$ or $\Lambda = \{i\}$, we construct $\Delta(\Lambda)$ similarly by applying the rule Γ to $(i, (1, 0))$, $(i, (0, 1))$ and $(i, (0, 0))$ respectively. Specifically,

for $\Lambda = \{i^*\}$ we put $\Delta(\Lambda) = 0$, meaning that once the configuration $(i^*, 0, 0)$ is reached the walk will remain in this state forever.

- Finally we set $\Delta(\Lambda) = 0$ for all other Λ.

With this set of transition vectors $\Delta(\Lambda)$, if $Q(t) \in \mathbb{Z}_\Lambda$ and $Q(t)$ encodes the current configuration (s^t, z_1^t, z_2^t) of the counter machine, then $Q(t+1) = Q(t) + \Delta(\Lambda)$ encodes the updated configuration $(s^{t+1}, z_1^{t+1}, z_2^{t+1})$

Finally, let us show that if we had an algorithm $\mathcal{E}$ for checking stability of a deterministic homogeneous walk, we would have an algorithm for checking whether a counter machine halts on a given initial configuration (s^0, z_1^0, z_2^0). Given a counter machine, construct a deterministic walk with initial state and transition rules as described above. Suppose we have an algorithm for checking the stability of this walk. We now construct an algorithm for solving the Halting Problem as follows. If the walk is *unstable*, we declare the counter machine *nonhalting*. This is accurate, because if it were halting, then the walk would end up in a "trapping" face $\mathbb{Z}_\Lambda$ with $\Lambda = \{i^*\}$ and would stay in the same state forever.

If, on the other hand, the walk is determined to be *stable*, then we simply follow the dynamics of our counter machine until either it halts or a certain configuration is repeated, i.e., for some $t_1 < t_2$ $(s^{t_1}, z_1^{t_1}, z_2^{t_1}) = (s^{t_2}, z_1^{t_2}, z_2^{t_2})$. By stability, the content of the counters in the counter machine remains bounded as a function of time, so some configuration must be repeated at some point. If this is the terminating configuration, then we declare the Counter Machine halting. If it is not, then we know that the Counter Machine will be repeating a cycle containing the repeated configuration indefinitely, and we declare the Counter Machine non-halting. We have constructed an algorithm for determining whether our Counter Machine halts, which is a contradiction. □

6 Multiclass Queueing Networks (MQNET) and undecidability

The authors [GK07] show that characterizing stable queueing networks is an algorithmically undecidable problem for the case of nonpreemptive static buffer priority scheduling policies and deterministic interarrival and service times.

6.1 Mqnet Setting

Deterministic Multiclass Queueing Network. Collection of J service nodes, $S_1, \ldots, S_J$, and N job classes, $1, \ldots, N$. Each node is assumed to be single-server type and can process at most one job at a time. Class i has buffer i with capacity B_i which can be finite or infinite. $Q_i(t)$ is the queue length corresponding to class i.

Each class i is associated with an external arrival process $A_i(0,t)$. In particular deterministic class-dependent arrivals occur at $na_i + b, n = 0, 1, \ldots)$ which arrived externally to the buffer B_i during $[0,t]$. Routing of jobs is done by a zero-one $N \times N$ sub-stochastic matrix R. Here $\forall i, l, R_{i,l} = 1$, if every job which completes service in class i at some time t is immediately routed to buffer B_l after the service completion and $R_{i,l} = 0$ is otherwise.

Static Buffer Priority Scheduling Policy. For each server S_j, a permutation θ_j of the elements belonging to S_j is fixed. $\forall t$ corresponding to service completion in S_j, the server S_j finds the index $i \in S_j$ with the smallest value $\theta_j(i)$ such that $Q_i(t) > 0$, selects the job in the head of this queue and begins working on it. The vector $\theta = (\theta_j)$, $1 \leq j \leq J$, then completely specifies the scheduling policy.

A queueing network described by servers $S_j, 1 \leq j \leq J$, classes $i = 1, 2, \ldots, N$, the routing matrix R. interarrival times a_i, delays b_i and service times m_i will be denoted by $\mathcal{Q}$.

The key differences in the setting of this queueing system from the NETWORK OF QUEUES in Section 4 is: i) the setting here is not *closed*; and ii) there is no *routing* here, i.e., each the sub-stochastic matrix exactly determines the next customer class for after each service; and iii) there is no question of control in the current. On the other hand there is a stability question in the current setting.

Definition 6.1. A triplet $(\mathcal{Q}, \Pi, Q(0))$ is defined to be stable if

$$\sup_{s \geq 0} \sum_{1 \leq i \leq N} Q_i(s) < \infty \tag{7}$$

A queueing network $\mathcal{Q}$ together with the scheduling policy Π is defined to be stable if $(\mathcal{Q}, \Pi, Q(0))$ is stable for every $Q(0)$.

The following theorem is established in [GKR09].

Theorem 7. *The problem of characterizing stable queueing networks for the case of nonpreemptive static buffer priority scheduling policies and deterministic interarrival and service times is undecidable.*

References

[BT00] V. D. Blondel and J. N. Tsitsiklis, *A survey of computational complexity results in systems and control*, Automatica **36** (2000), no. 9, 1249–1274.

[Dai95] J. G. Dai, *On the positive Harris recurrence for multiclass queueing networks: A unified approach via fluid models*, Ann. Appl. Probab. **5** (1995), 49–77.

[DW96] J. G. Dai and G. Weiss, *Stability and instability of fluid models for certain re-entrant lines*, Mathematics of Operations Research **21** (1996), 115–134.

[FMM95] G. Fayolle, V. A. Malyshev, and M. V. Menshikov, *Topics in the constructive theory of countable Markov chains*, Cambridge University Press, Cambridge, UK, 1995.

[Gam02] D. Gamarnik, *On deciding stability of constrained homogeneous random walks and queueing systems*, Mathematics of Operations Research **27** (2002), no. 2, 272–293.

[GK07] D. Gamarnik and D. Katz, *Correlation decay and deterministic FPTAS for counting list-colorings of a graph*, Proceedings of 18th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2007.

[GKR09] D. Gamarnik and D. Katz-Rogozhnikov, *On deciding stability of queueing networks under priority scheduling policy*, Annals of Applied Probability **19** (2009), 2008–2037.

[GS10] C. Goodman-Strauss, *Cant decide? undecide!*, Notices of the American Mathematical Society **57** (2010), 343–356.

[Hoo66] P. Hooper, *The undecidability of the Turing machine immortality problem*, The Journal of Symbolic Logic **2** (1966), 219–234.

[LMF94] G. Louth, M.Mitzenmacher, and F.Kelly, *Computational complexity of loss networks*, Theoretical Computer Science **14** (1994), 45–59.

[Mal72a] V. A. Malyshev, *Analytic method in the theoy of two-dimensional random walks*, Sib.Math.J. **13** (1972), no. 6, 1314–1327.

[Mal72b] ———, *Classification of two-dimensional positive random walks and almost linear semimartingales*, Dokl. Akad. Nauk SSSR **202** (1972), 526–528.

[Mal93] ———, *Networks and dynamical systems*, Adv.Appl.Prob. **25** (1993), 140–175.

[Men74] M. V. Menshikov, *Ergodicity and transience conditions for random walks in the positive octant of space*, Soviet.Math.Dokl. **217** (1974), 755–758.

[MSZ06] E. Modiano, D. Shah, and G. Zussman, *Maximizing throughput in wireless networks via gossiping*, ACM SIGMETRICS Performance Evaluation Review **34** (2006).

[PT94] C. Papadimitriou and J. Tsitsiklis, *The complexity of optimal queueing network control*, Proceedings of the Ninth Annual Structure in Complexity Theory Conference, 1994, 1994.

[Sip97] M. Sipser, *Introduction to the theory of computability*, PWS Publishing Company, 1997.

[SS] D. Shah and J. Shin, *Randomized scheduling algorithm for queueing networks*, Annals of Applied Probability, under revision.

[STT] D. Shah, D. N. C. Tse, and J. N. Tsitsiklis, *Hardness of low delay network scheduling*, Submitted.

[UNS09] P. Gupta U. Niesen and D. Shah., *On capacity scaling in arbitrary wireless networks*, IEEE Trans. on Information Theory **55** (2009).