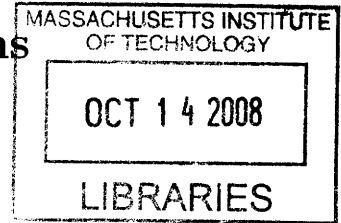


Algorithmic issues in queueing systems and  
combinatorial counting problems

by

Dmitriy A. Katz-Rogozhnikov



Submitted to the Sloan School of Management  
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Operations Research

at the

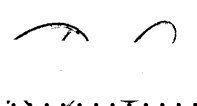
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2008

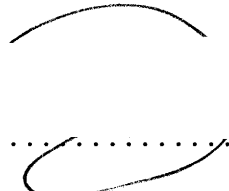
© Massachusetts Institute of Technology 2008. All rights reserved.

Author .....

Sloan School of Management  
August 15, 2008

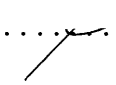
Certified by ..... 

David Gamarnik  
Associate Professor  
Thesis Supervisor

Certified by ..... 

Dimitris Bertsimas  
Boeing Professor of Operations Research  
Thesis Supervisor

Accepted by .....

  
Cynthia Barnhart  
Co-director, Operations Research Center



# Algorithmic issues in queueing systems and combinatorial counting problems

by

Dmitriy A. Katz-Rogozhnikov

Submitted to the Sloan School of Management  
on August 15, 2008, in partial fulfillment of the  
requirements for the degree of  
Doctor of Philosophy in Operations Research

## Abstract

Multiclass queueing networks are used to model manufacturing, computer, supply chain, and other systems. Questions of performance and stability arise in these systems. There is a body of research on determining stability of a given queueing system, which contains algorithms for determining stability of queueing networks in some special cases, such as the case where there are only two stations. Yet previous attempts to find a general characterization of stability of queueing networks have not been successful.

In the first part of the thesis, we contribute to the understanding of why such a general characterization could not be found. We prove that even under a relatively simple class of static buffer priority scheduling policies, stability of deterministic multiclass queueing network is, in general, an undecidable problem. Thus, there does not exist an algorithm for determining stability of queueing networks, even under those relatively simple assumptions. This explains why such an algorithm, despite significant efforts, has not been found to date.

In the second part of the thesis, we address the problem of finding algorithms for approximately solving combinatorial graph counting problems. Counting problems are a wide and well studied class of algorithmic problems, that deal with counting certain objects, such as the number of independent sets, or matchings, or colorings, in a graph. The problems we address are known to be  $\#P$ -hard, which implies that, unless  $P = \#P$ , they can not be solved exactly in polynomial time. It is known that randomized approximation algorithms based on Monte Carlo Markov Chains (MCMC) solve these problems approximately, in polynomial time. However, these randomized algorithms can never provide proven upper or lower bounds on the number of objects they are counting, but can only give probabilistic estimates.

We propose a set of *deterministic* algorithms for counting such objects for three classes of counting problems. They are interesting both because they give an alternative approach to solving these problems, and because unlike MCMC algorithms, they provide provable bounds on the number of objects. The algorithms we propose are for special cases of counting the number of matchings, colorings, or perfect matchings (permanent), of a graph.

Thesis Supervisor: David Gamarnik  
Title: Associate Professor

Thesis Supervisor: Dimitris Bertsimas  
Title: Boeing Professor of Operations Research

# Acknowledgments

First and foremost I am indebted to my advisors, Dimitris Bertsimas and David Gamarnik, for their help and support in conducting this research, and throughout my graduate studies at MIT.

I would like to thank professor Glen Urban for the opportunity to conduct interesting and intellectually stimulating research with him as well as for his support during a difficult period in my MIT graduate studies, and professor Jim Orlin for his extremely valuable advice.

I would like to thank professor Devavrat Shah for his help in conducting this research, and for agreeing to be on my thesis committee.

I would like to thank Paulette Mosley for helping me navigate the administrative side of MIT graduate studies, and for her patience at times when I was less than responsive to concerns not directly related to research.

I would also like to thank all of the ORC community – students, faculty, and the administrative staff, for the excellent environment I enjoyed during my graduate studies at MIT.



# Contents

<b>1</b>	<b>Introduction</b>	<b>15</b>
1.1	Multiclass Queueing Networks . . . . .	15
1.1.1	Thesis contribution to the problem of deciding stability of multiclass queueing networks . . . . .	18
1.2	Counting problems and complexity classes . . . . .	19
1.2.1	NP complexity class . . . . .	19
1.2.2	#P complexity class . . . . .	21
1.2.3	RP complexity class . . . . .	22
1.2.4	Approximation algorithms . . . . .	23
1.3	Prior work on algorithms for solving counting problems . . . . .	24
1.3.1	Solving counting problems by computing marginal distribution . . . . .	24
1.3.2	Prior work on counting colorings and list colorings of a graph . . . . .	25
1.3.3	Prior work on counting matchings . . . . .	26
1.3.4	Prior work on computing the permanent of a matrix . . . . .	26
1.4	Thesis contribution to solving counting problems . . . . .	27
1.4.1	Contribution to counting colorings of a graph . . . . .	29
1.4.2	Markov Random Fields . . . . .	30
1.4.3	Contribution to counting matchings in a graph . . . . .	30
1.4.4	Contribution to computing the permanent of a matrix . . . . .	31
<b>2</b>	<b>On deciding stability of multiclass queueing networks under buffer priority</b>	

<b>scheduling policies</b>	<b>33</b>
2.1 Model description and the main result . . . . .	33
2.1.1 Deterministic multiclass queueing networks and static buffer priority scheduling policy . . . . .	33
2.1.2 The main result . . . . .	36
2.2 Counter Machine, Halting Problem and undecidability . . . . .	36
2.2.1 Counter Machine and the Halting Problem . . . . .	37
2.2.2 Simplified Counter Machine (SCM), stability and decidability . . . . .	38
2.3 Description of the queueing network corresponding to a SCM . . . . .	39
2.3.1 The description of the subnetwork $SN_i$ , $i = 1, 2$ . . . . .	39
2.3.2 The description of the main network $MN$ . . . . .	43
2.4 Proof of Theorem 1 . . . . .	45
2.4.1 Proof of the induction step of Theorem 4 . . . . .	46
2.5 Conclusion . . . . .	58
<b>3 Counting list colorings of a graph</b>	<b>59</b>
3.1 Definitions and the main result . . . . .	59
3.2 Preliminary technical results . . . . .	61
3.2.1 Basic recursion . . . . .	61
3.2.2 Upper and lower bounds . . . . .	63
3.3 Algorithm and complexity . . . . .	66
3.3.1 Description of an algorithm . . . . .	66
3.3.2 Some properties . . . . .	67
3.3.3 Complexity . . . . .	68
3.4 Correlation decay . . . . .	69
3.5 Comparison of the correlation decay on a computation tree and the spatial correlation decay property . . . . .	75
3.6 Conclusions . . . . .	76



<b>4</b>	<b>Markov random field and partition function</b>	<b>77</b>
4.1	Model and the preliminary results . . . . .	77
4.1.1	Basic recursion and the algorithm . . . . .	80
4.1.2	Complexity . . . . .	83
4.1.3	Correlation decay analysis . . . . .	84
4.1.4	Example: Potts model . . . . .	89
4.2	Conclusions . . . . .	90
<b>5</b>	<b>Counting matchings in a graph</b>	<b>91</b>
5.1	Definitions, preliminaries and the main result . . . . .	91
5.2	Basic recursion and correlation decay analysis . . . . .	93
5.3	Algorithm . . . . .	97
5.4	Conclusions . . . . .	98
<b>6</b>	<b>Computing the Permanent of a matrix</b>	<b>101</b>
6.1	Preliminaries and the main result . . . . .	101
6.2	Constant degree expanders . . . . .	104
6.3	General graphs . . . . .	107
6.4	Conclusions . . . . .	110



# List of Figures

2-1	Subnetwork $SN_i$ . . . . .	41
2-2	Main network $MN$ . . . . .	42
2-3	Workload $W_i(s)$ . Case 1 . . . . .	48
2-4	Workload $W_i(s)$ . Case 2 . . . . .	49
2-5	Workload $W_i(s)$ . Case 3 . . . . .	50



# List of Tables

2.1	Servers and classes in $SN_i$ . . . . .	40
2.2	Arrival processes into $SN_i$ . . . . .	41
2.3	Servers and classes in $MN$ . . . . .	44
2.4	Arrival processes into $MN$ . . . . .	44

‘

# Chapter 1

## Introduction

### 1.1 Multiclass Queueing Networks

Queueing network is a ubiquitous tool for modeling a large variety of real life processes such as communication and data networks, manufacturing processes, call centers and service networks, and many other real life systems. It is an important task to design and operate queueing networks so that the performance is acceptable. One of the key qualitative performance measures is stability. Roughly speaking, a queueing network is stable if the total expected number of jobs in the network is bounded as a function of time. In probabilistic framework, which is typically used to formalize the stability question, it means that the underlying queue length process is positive (Harris) recurrent [CY01],[MT93],[Dai95]. We do not provide a formal definition of this notion here as in this thesis we consider exclusively deterministic queueing networks, for which stability simply means that the total number of jobs in the network remains bounded as a function of time. The formalities of the model and stability notions are delayed until Chapter 2.

The research on stability questions started with the works of Kumar and Seidman [KS90], Lu and Kumar [LK91], and Rybko and Stolyar [RS92], who identified for the first time queueing networks and work-conserving scheduling policies leading to instability, even though every processing unit was nominally underloaded (the condition  $\rho_\sigma < 1$  was satisfied by

every server  $\sigma$ ). This initiated the search for tight stability conditions. Important advances were obtained in this direction, most notably the development of the fluid model methodology which significantly simplifies the stability issue by reducing the underlying stochastic problem to a simpler deterministic continuous time continuous state problem [Dai95],[Sto95]. It was established that stability of the fluid model implies stability of the underlying stochastic network [Dai95],[Sto95], and partially the converse result holds as well [Mey95], [Dai96],[PR00],[GH05], although not always [Bra99],[DHV99]. Yet, even characterizing stability of fluid models turned out to be non-trivial [DM97],[DW96],[BGT96],[DV00] and no full characterization is available either. Meanwhile it was discovered that certain classes of networks and scheduling policies are universally stable. For example, networks with feedforward (acyclic) structure were proven to be stable under an arbitrary work-conserving scheduling policy [DM95],[Dai95],[CY01]. A First-Buffer-First-Serve, Last-Buffer-First-Serve static buffer priority type scheduling policies were shown to stabilize an arbitrary queueing network satisfying a certain topological restriction (so-called reentrant line) [KK01],[DW96]. A certain simple scheduling policy based on due dates was shown to stabilize an arbitrary network [Bra01]. At the same time some simple static buffer priority policies are not necessarily stable, as was shown in the original works on instability [KS90],[LK91],[RS92]. Also First-In-First-Out (FIFO) policy can lead to instability [Sei94],[Bra94].

While most of the aforementioned research activity was conducted by the operations research, electrical engineering and mathematics communities, in parallel and independently the stability problem was investigated by the theoretical computer science community using the *Adversarial Queueing Network (AQN)* model. The motivation there coming from data networks, the model is somewhat different: no probabilistic assumptions are made on either the arrival or service processes. Instead an adversary is assumed to inject jobs (communication packets) into the network which is represented as a graph. The links of the graph serve the roles of processing units and the processing times are typically assumed to be equal to one unit of time deterministically. In this setting the model is defined to be stable if for every pattern of packet injections, subject to certain load conditions,



the total number of packets remains bounded as a function of time. The AQN was introduced by Borodin et al. [BKR<sup>+</sup>01] and further researched by many authors [AAF<sup>+</sup>96], [And00],[AZ00],[AKOR98],[Gam00],[Gam03],[Goe99], [LPSR02],[Ros02],[Tsa97]. Many results similar to the stochastic networks counterpart were established. It was shown that while AQN corresponding to an acyclic graph is always stable [BKR<sup>+</sup>01], there are AQN and scheduling policies (usually called protocols) which are work-conserving (usually called greedy) and which lead to instability [AAF<sup>+</sup>96],[Goe99]. It was also established that FIFO can lead to instability [AAF<sup>+</sup>96], even with arbitrary small injection rates [BG03]. The relevance of fluid models to AQN was established in [Gam00]: stability of the fluid model implies stability of AQN. Partially a converse result holds as was shown also in [Gam00]. Yet, despite an impressive progress in the area and interesting parallel development to the stochastic counterpart, tight characterization of stable AQN is still not known.

Stability theory comes in various flavors. Recently a lot of attention was devoted to stability of certain utility maximization and max/min type scheduling policies in stochastic models of internet congestion control [RM00],[dVLK01],[BM01],[LS04],[Sri04], [Bra05],[GW06],[CST]. In such model arriving jobs are flows which simultaneously occupy several processing servers. We do not discuss this type of stability questions in this thesis.

In this thesis we frame the problem of characterizing stable queueing networks as an algorithmic decision problem: given a queueing network and an appropriately defined scheduling policy determine whether the network is stable. In order to introduce the problem formally we consider the simplest possible setting - the interarrival times and service times are assumed to take deterministic rational values. We focus exclusively on a simple class of scheduling policies, namely non-preemptive static buffer priority scheduling policies. We assume that buffers have finite or infinite capacity. Jobs which upon arrival see a full (finite) buffer are dropped from the network. We note that the assumption of finite buffers is the only departure from models studied in the context of stability of queueing networks problem. The details of the model are given in Section 2.1.

### 1.1.1 Thesis contribution to the problem of deciding stability of multiclass queueing networks

Our main result is that stability of queueing networks operating under the class of the so called static buffer priority scheduling policies is an undecidable property. Thus no constructive means of characterizing stable queueing networks for this broad class of policies is possible. This resolves the open problem of providing tight characterization of stable queueing networks for this class of policies. Our work builds partially on an earlier work [Gam02] where undecidability result was established for the class of so-called *generalized priority* scheduling policy. There are important difference between the current work and [Gam02]. The class of generalized priority policies was not considered in the literature prior to [Gam02]. Additionally, generalized priority policies allow idling, whereas most of the work on stability analysis focuses on work-conserving scheduling policy. Also [Gam02] considered single-server setting, whereas here we consider the network setting. We note that for the class of buffer priority policies (as well as any other work-conserving scheduling policy) the question of stability of a single server model is decidable: one needs to compute the load factor  $\rho$ . Then the system is stable if and only if  $\rho < 1$  ( $\rho \leq 1$  if all of the interarrival and service times are deterministic).

The concept of undecidability was introduced in the classical works of Alan Turing in 1930's and it is one of the principal tools for establishing limitations of certain computational problems. The first problems which were established to be undecidable included Turing Halting Problem, Post Correspondence Problem and several related problems [Sip97]. Typically one establishes undecidability of a given problem by taking a problem which is already known to be undecidable, and establishing a reduction from this problem to the given problem of interest. This method is well known in the computer science literature as the *reduction method*. Lately several problems were proven to be undecidable in the area of control theory [BBK<sup>+</sup>01],[BT00b], [BT00a]. In particular the work of Blondel et al. [BBK<sup>+</sup>01] used a device known as *Counter Machine* or *Counter Automata* as a reduction tool. In this thesis as in [BBK<sup>+</sup>01] as well as in [Gam02], our proof technique is also based on a reduction to a Counter Machine model, though the construction details are substantially different from

[Gam02]. We use a well-known Rybko-Stolyar network [RS92] as a gadget and construct an elaborate queueing network which is able to emulate the dynamics of an arbitrary Counter Machine. The undecidability result is then a simple consequence of undecidability of the Halting Problem for Counter Machine, which is a classical result [HU69].

## 1.2 Counting problems and complexity classes

In this section, we give an overview of the  $\#P$  complexity class and related complexity classes, since the counting problems we discuss belong to this complexity class.

$\#P$  complexity class for counting problem is analagous to the NP complexity class for decision problems, so we will start by briefly describing NP complexity class. For more information on the NP complexity class, and complexity and computability theory in general, we recommend Sipser's book, [Sip97].

### 1.2.1 NP complexity class

Perhaps the most used characterization of an algorithm in terms of performance in theoretical computer science is whether its running time is bounded by a polynomial in the size of the input. If so, the algorithm is said to be *polynomial time* and belongs to the complexity class P. Thus, complexity class P is classically defined to be a class of yes/no problems for which there exists an algorithm which runs in time bounded by a polynomial in the size of the input. Formally, the definition of P is given in terms of the running time of a *Turing machine*, but it is equivalent for any reasonable computational model (for more formal definitions, see [Sip97]). This characterization is so prevalent, that in some literature such algorithms are simply referred to as “fast” or “efficient”. While there are a lot of problems which are proven to be in P, often called “easy” problems, it is usually impossible to prove that a problem which is considered to be hard is not in P. A few problems are known to be undecidable, such as the Turing Halting Problem, or, as we discussed in Section 1.1, the problem of determining stability of a queueing network operating under a static buffer priority policy,

and thus are trivially not in P. A few other problems can be directly shown to be outside of P. But for most of the problems, perhaps the vast majority of those arising in practice, such a proof seems out of reach. The NP complexity class, first introduced by Cook (see [Sip97]), is invaluable in proving that a certain problem is “unlikely” to be in P, considered by most to be a sufficient proof that a problem is “hard”, and that efforts to find polynomial time algorithm are extremely unlikely to be fruitful. Cook defines NP complexity class as a class of yes/no problems which can be solved in Polynomial time by a *non-deterministic Turing Machine*, or, equivalently, for which there exists a solution that can be verified in polynomial time. Naturally, all problems in P are in NP, but it is unknown and is one of the most well known open problems in theoretical computer science whether P and NP are indeed equivalent. It is considered likely that P does not equal NP, partly because decades of research, including the time before NP complete class has even been defined, have failed to produce a polynomial algorithm for an NP-hard problem. Indeed a lot of statements are made modulo the  $P \neq NP$  assumption.

We say that problem A is *NP-hard* if, given an oracle (or a polynomial algorithm) for A, any problem in NP can be solved in polynomial time. Thus, unless P equals NP, an NP-hard problem can not be solved in polynomial time. A proof that a problem is NP-hard is considered to be sufficient evidence that a polynomial algorithm for such a problem is impossible to construct, and is probably by far the most used way of proving that a given problem is “hard”.

If a problem is both in NP, and NP-hard, we say that it is *NP-complete*. NP-complete problems are “hardest in NP”, and are all polynomially equivalent – if one of them can be solved in polynomial time, all can be. Examples of NP-complete problems include Travelling Salesman Problem, determining if a graph contains an independent set of a given size, determining if a graph has a Hamiltonian path, and many others.

NP complexity class proved invaluable in classifying yes/no decision problems. However it is not directly applicable to counting problems. Although counting problems can be NP-hard, only Yes/No decision problems fall under definition of NP-complete or “in NP”.

## 1.2.2 #P complexity class

While NP class proved very useful for characterizing difficulty of NP problems, its definition does not directly apply to counting problems. In order to deal with counting type problems, in 1979, Valiant introduced a #P complexity class [Val].

Formally, while NP problem can be phrased as “does a given non-deterministic Turing Machine have an accepting path”, #P problem can be phrased as “how many accepting paths does a given non-deterministic Turing machine have”? Less formally, #P complexity class is a class of problems corresponding to counting the number of objects, existence of which can be verified in polynomial time, such as “How many Hamiltonian cycles does a given graph have”, or “How many independent sets of a given size does a given graph contain”?

In analogy with the NP complexity class, we say that problem A is #P-hard if an oracle (or a polynomial time algorithm) for problem A enables us to solve any #P-hard problem in polynomial time. This implies that if a #P-hard problem can be solved in polynomial time, all #P problems can be. Similarly, #P-complete problems are those that are both in #P and are #P-hard, or “the hardest problems in #P”.

It is not surprising that most problems of the form “count the number of objects, finding one of which is NP-hard”, such as “how many independent sets of a given size does a given graph have?” are #P-hard. However, surprisingly, some problems of the form “Count the number of objects which *are* possible to find in polynomial time”, such as the well studied “Count the number of perfect matching of a given bipartite graph”, turn out to be #P-hard as well. That is, while finding a perfect matching in a given bipartite graph, if one exists, can be done in polynomial time [Kuh55], determining how many matchings exist is #P-hard [Val79].

Proving that a given counting problem is #P-hard (or #P-complete) is considered sufficient evidence for it to be extremely unlikely that such a problem can be solved in polynomial time.

However #P-hard problems do arise in theory and in practice. While exact polynomial-time algorithm is extremely unlikely to exist, it can still be possible to solve such counting

problems *approximately*. Thus most of the effort in solving #P-hard problems has been focused on finding *efficient approximation algorithms*. Approximation algorithms are further discussed in Subsection 1.2.4.

### 1.2.3 RP complexity class

While we do not directly address the randomized polynomial time (RP) complexity class in the body of this thesis, our works has some bearing on research surrounding RP, so we will briefly mention it here.

RP is the class of problems solvable in polynomial time by an algorithm that is allowed to make random choices. If the answer to the problem is “no”, the algorithm must correctly return “no”, but if the answer is “yes”, a small probability of a mistake is allowed.

More formally, RP is the class of problems for which a probabilistic Turing machine exists such that:

- The running time is bounded by a polynomial in the size of the input
- If the correct answer is “no”, the Turing machine returns “no”
- If the correct answer is “yes”, the Turing machine returns “yes” with probability at least  $1/2$

Note that  $1/2$  in the definition can be any fraction less than one, by running the Turing machine on the same input repeatedly.

In practice, randomization has proven to be useful in designing efficient algorithm. However, unlike the case with the P vs. NP question, a large number of researchers believe that RP is indeed equal to P.

The evidence for  $RP = P$  includes, among other things, the fact that most randomized algorithms can be *de-randomized* – that is, based on the randomized algorithm, a deterministic one for solving the same problem can be constructed. One of the few important classes algorithms which resisted de-randomization attempts is Monte Carlo Markov Chains

(MCMC). In the present thesis we do not derandomize MCMC. However we do show how to solve deterministically a large class of problems that previously was only solvable in polynomial time by a randomized MCMC algorithm. Thus we shed some, albeit limited, light on the fundamental  $P = RP$  question.

For more discussion of RP, and related complexity classes, see [Sip97]

## 1.2.4 Approximation algorithms

As we mentioned in 1.2.2, some problems that seem to be unsolvable precisely can be solvable approximately. Of particular interest is a class of algorithms which solve such problems to an arbitrarily high precision.

**Definition 1.** *An algorithm  $\mathcal{A}$  is defined to be a Fully Polynomial Time Approximation Scheme (FPTAS) for a computing  $Z$  if given an arbitrary  $\delta > 0$  it produces a value  $\hat{Z}$  satisfying*

$$\frac{1}{1 + \delta} \leq \frac{\hat{Z}}{Z} \leq 1 + \delta,$$

*in time which is polynomial in the length of input and in  $\frac{1}{\delta}$ . An algorithm is defined to be Polynomial Time Approximation Scheme (PTAS) if the computation time is polynomial in the length of input, but not necessarily in  $\frac{1}{\delta}$*

In practice, the approximate solution of high enough precision can often be as useful as an exact one, thus FPTAS and PTAS are of great interest in both theoretical and applied computer science.

Problems we address in Chapters 3 through 6 belong to the #P-hard complexity class, making an exact solution by a deterministic or randomized polynomial algorithm unlikely. Thus previous research and our efforts have been focused on finding approximation algorithms for those problems.

## 1.3 Prior work on algorithms for solving counting problems

In this section we will discuss existing methods for solving approximately #P-hard counting problems. By far the most successful method for solving counting problems to date has been Monte Carlo Markov Chain (MCMC) algorithms.

### 1.3.1 Solving counting problems by computing marginal distribution

Most of the #P-hard counting problem can not be solved by counting the desired object directly, as their number can be exponential in size of the input. Thus straightforward counting would not result in a polynomial time algorithm.

Most methods for solving counting problems instead compute marginal distributions, appropriately defined. As an example, we will use the problem of counting number of independent sets in a graph.

Consider the following sub-problem:

**Problem A:** “Given graph  $\mathbb{G}$  and a vertex  $v$ , find the probability that an independent set chosen uniformly at random in  $\mathbb{G}$  does not contain  $v$ .”

Suppose we have an algorithm, called *ProblemASolver*, which can solve **Problem A**. Then, using it, we can construct the following algorithm for counting independent sets in a graph.

#### Algorithm CountIndepSets

INPUT: Graph  $\mathbb{G}$ .

BEGIN

Set *ProbabilityProduct* = 1.

While  $\mathbb{G}$  is not empty

    Pick any vertex  $v$  in  $\mathbb{G}$ .



Set  $ProbabilityProduct = ProbabilityProduct * ProblemASolver(\mathbb{G}, v)$

Set  $\mathbb{G} = \mathbb{G} \setminus v$

END

OUTPUT:  $1/ProbabilityProduct$ .

If `ProblemASolver` gives precise answer, `CountIndepSets` will output the precise number of independent sets in a graph. If `ProblemASolver` gives an approximate answer, so does `CountIndepSets`. We will discuss the proof of this fact and the effect of the precision of `ProblemASolver` on the precision of `CountIndepSets` elsewhere. Here we just note that the precision of `CountIndepSets` is a function of the precision of `ProblemASolver`, and the problem size.

### 1.3.2 Prior work on counting colorings and list colorings of a graph

The setting for the problem of counting the number of list colorings of a graph is as follows. Each node of a given graph is associated with a list of colors. An assignment of nodes to colors is called list coloring if every node is assigned to some color from its list and no two nodes sharing an edge are assigned to the same color. When all the lists are identical, the problem reduces to the problem of coloring of a graph. The problem of determining whether a list coloring exists is NP-hard, but provided that the size of each list is strictly larger than the degree for each node, a simple greedy algorithm produces a coloring. We are concerned with the corresponding counting problem – compute the total number of list colorings of a given graph/list pair. This problem is known to be  $\#P$  hard even for the restricted problem of counting the colorings, and the focus is on the approximation algorithms. The existing approximation schemes are based on the rapidly mixing Markov chain technique, also known as Glauber dynamics approach. It was established by Jerrum [Jer95] that the Glauber dynamics corresponding to graphs where the ratio of the number of colors to degree satisfies  $q/\Delta \geq 2$ , mixes rapidly. This leads to a randomized approximation algorithm for enumerating the number of colorings. The 2-barrier was first broken by Vigoda [Vig00], who lowered the ratio requirement to 11/6. Many further significant improvements were obtained

subsequently. The state of the art is summarized in [FV06]. For a while the improvement over  $11/6$  ratio came at a cost of lower bound  $\Omega(\log n)$  on the maximum degree, where  $n$  is the number of nodes. This requirement was lifted by Dyer et al. [DFHV04].

### 1.3.3 Prior work on counting matchings

We now discuss the problem of computing the total number of (full and partial) matchings in a given graph. This problem, along with many other combinatorial counting problems falls into the class of  $\#P$ -complete problems. Thanks to Vadhan’s work [Vad01], the problem of counting matchings even in regular graphs with bounded degree (to be precise, degree at least five) is also known to be  $\#P$ -complete. FPRAS based on MCMC was established for arbitrary graphs [JS97].

### 1.3.4 Prior work on computing the permanent of a matrix

A permanent of an  $n$  by  $n$  matrix  $A = (a_{i,j})$  is  $\text{Perm}(A) \triangleq \sum_{\sigma} \prod_{1 \leq i \leq n} a_{i,\sigma(i)}$ , where  $\sigma$  runs over the elements of the permutation group on the set  $1, 2, \dots, n$ . When  $A$  is a zero-one matrix,  $\text{Perm}(A)$  counts the number of perfect matching in the bi-partite graph  $\mathbb{G}$  corresponding to the adjacency matrix  $A$ . Permanent of a matrix  $A$  is naturally related to the determinant of  $A$  (signs in front of products are removed). Yet, while the determinant of a matrix can be computed in polynomial time, the problem of computing the permanent belongs to the  $\#P$  class even when  $A$  is a zero-one matrix [Val79]. Thus, modulo a basic complexity theoretic conjecture, no polynomial time algorithm exists for computing the permanent of a matrix.

A significant research effort was devoted to constructing approximation algorithms for computing the permanent of a matrix. The first breakthrough came from Jerrum and Sinclair [JS89] who constructed a *randomized* fully polynomial time approximation scheme (FPTAS) for 0, 1 matrices satisfying certain “polynomial growth” condition. This condition relates to the ratio of near-perfect to perfect matchings and requires that this ratio grows at most polynomially in  $n$ . The algorithm is based on constructing a rapidly mixing Markov

chain which runs on perfect and near perfect matchings. Using this algorithm as a black-box Jerrum and Vazirani [JS89] constructed a randomized approximation algorithm for an arbitrary zero-one matrix, but with mildly exponential running time  $\exp(O(n^{\frac{1}{2}} \log^2 n))$ . A recent dramatic improvement was obtained by Jerrum, Sinclair and Vigoda [JSV04], who constructed an FPTAS for an arbitrary matrix with non-negative entries.

Unfortunately, the randomization aspect of the algorithm of [JSV04] is quite crucial. It is not known how to derandomize their Markov chain based algorithm, and the best known deterministic approximation algorithm is due to Linial, Samoridnitsky and Wigderson [LSW00], who provide only  $e^n$  multiplicative approximation factor guarantee. Their algorithm is based on an FPTAS for a related matrix scaling problem and uses van der Waerden's conjecture, which states that a permanent of every doubly stochastic matrix is at least  $n!/n^n$ . The  $e^n$  approximation factor can be improved to  $(k/(k-1))^{kn}$  for the case of matrices with row and column sums bounded by  $k$ , using the Gurvits' proof [Gur06] of Schrijver's bound [Sch98]. Thus approximation of a permanent is one of the famous algorithmic problems where the existing gap between the randomized and deterministic algorithms is so profound.

## 1.4 Thesis contribution to solving counting problems

Our approach is based on establishing a certain *correlation decay* property which has been considered in many settings [SS97], [GMP05],[BW02],[BW04],[Jon02] and has been recently a subject of interest. In particular, the correlation decay has been established in [GMP05] for coloring triangle-free graphs under the assumption that  $\alpha > \alpha^* = 1.763\dots$ , the unique solution of  $\alpha e^{-\frac{1}{\alpha}} = 1$ . (Some mild additional assumptions were adopted). The principal motivation for establishing the correlation decay property comes from statistical physics, in particular the connection with the uniqueness of the associated Gibbs measure (uniform measure in our setting) on infinite versions of the graph, typically lattices. Recently, however, a new approach linking correlation decay to counting algorithms was proposed in Bandyopadhyay and Gamarnik [BG06] and Weitz [Wei06]. The idea is to use correlation decay property

instead of Markov sampling for computing marginals of the Gibbs (uniform) distribution. This leads to a deterministic approach since the marginals are computed using a dynamic programming like scheme (also known as Belief Propagation (BP) algorithm [YFW00]). This approach typically needs a locally-tree like structure (large girth) [Sha05] in order to be successful. The large girth assumption was explicitly assumed in [BG06], where the problems of computing the number of independent sets and colorings in some special structured (regular) graphs was considered. Weitz [Wei06] cleverly by-passes the large girth assumption by using a certain self-avoiding tree construction thus essentially reducing the original problem to a problem on a tree with appropriate boundary conditions implied by independent sets. This idea was used recently by Jung and Shah [JS06] to introduce a version of a BP algorithm which works on a non-locally-tree like graphs, where appropriate correlation decay can be established. This approach works for binary type problems (independent sets, matchings, Ising model) but does not apparently extend to multi-valued problems.

We propose a general deterministic approximate counting algorithm which can be used for arbitrary multi-valued counting problem. We also by-pass the large girth assumption by considering a certain *computation tree* corresponding to the Gibbs (uniform for the case of colorings) measure. Our principal insight is establishing correlation decay for the computation tree as opposed to the conventional correlation decay associated with the graph-theoretic structure of the graph. We provide a discussion explaining why it is crucial to establish the correlation decay in this way in order to obtain FPTAS. Contrast this with [GMP05] where correlation decay is established for the coloring problem but in the conventional graph-theoretic distance sense. Our method is similar to the self-avoiding walk method of Weitz but somewhat more direct as the step of relating the marginal probability on a graph to the marginal probability on the tree is by-passed in our computation tree approach. The advantage of establishing correlation decay on a computation tree as opposed to the original graph has been highlighted also in [TJ02] in the context of BP algorithms and the Dobrushin's Uniqueness condition. More importantly our approach works for general, not necessarily two-valued model.

The connection between the correlation decay property and the mixing rates of the Markov chain corresponding to the counting problems has been investigated recently [DSVW04],[GMP05],[BKMP01],[MS06]. It is known that if the underlying graph satisfies a certain sub-exponential growth condition, the spatial correlation decay implies rapid mixing (see e.g., [DSVW04], [GMP05]). The converse, however, does not hold in general, as shown by Berger et al. [BKMP01].

Our work, along with [BG06],[BN06],[Wei06],[NT] reinforces this connection, as well as, broadly speaking, contributes to the exciting and emerging connection between theoretical computer science, probability theory and statistical physics.

### 1.4.1 Contribution to counting colorings of a graph

As noted before, all the approximation algorithms known so far for solving the problem of counting the number of colorings of a (non-tree-like) graph relied on Monte Carlo Markov Chains, and produce a randomized algorithm. In this thesis we focus on a different approach to the counting list colorings problem. Our setting is a list coloring problem. We require that the size of every list is at least  $\alpha\Delta + \beta$ , where  $\alpha > \alpha^{**} = 2.8432\dots$  - the unique solution to  $\alpha e^{-\frac{1}{\alpha}} = 2$ , and  $\beta$  is a large constant which depends on  $\alpha - \alpha^{**}$ . Our girth restriction is  $g \geq 4$ , namely, the graph is triangle-free. We obtain the following results. First, assuming that the size of each list is at most a constant, we construct a *deterministic* Fully Polynomial Time Approximation Scheme (FPTAS) for the problem of computing the total number of list colorings of a given graph/list pair. Second, for an arbitrary graph/list pair (no assumptions on the list sizes) we construct an approximation algorithm with complexity  $2^{O(\log^2 n)}$ . Namely, our algorithm is super-polynomial but still significantly quicker than exponential time.

Although our regime  $\alpha > 2.8432\dots$  is weaker than  $q/\Delta > 2$ , for which the Markov chain is known to mix rapidly, the important contribution of our method is that it provides a *deterministic* algorithm. Presently no deterministic algorithms are known for counting approximately the number of coloring of a graph.

## 1.4.2 Markov Random Fields

We extend our approach used to solve the problem of counting the number of colorings of a graph to Markov random field model and show that under certain conditions, the computation tree satisfies the correlation decay property and, as a result, one obtains a deterministic algorithm for computing approximately the associated partition function.

## 1.4.3 Contribution to counting matchings in a graph

In this thesis we use the correlation decay approach for constructing a fully polynomial deterministic approximation scheme for counting the total number of (partial and full) matchings of a graph. In fact we solve a more general problem - the one of computing the partition function corresponding to matchings with a given activity level  $\lambda$ . The result is obtained by establishing the correlation decay property on the computation tree *for every*  $\lambda > 0$  and *every* degree  $\Delta$  (contrast with  $\Delta \leq 5$  condition for counting independent sets [Wei06]). Interestingly, the analysis becomes far easier, when compared to its counterparts for independent sets and coloring problems. The proof of the correlation decay property is done differently than in [Wei06] and uses mean value theorem to establish a required contraction. A similar approach was used in [KK98], but in the present context of counting matchings it is particularly simple. Using a two-level recursion analysis, we show that the rate of the correlation decay is  $\approx 1 - O(\frac{1}{\sqrt{\Delta}})$ , where  $\Delta$  is the maximum degree of the graph. As a corollary we construct a deterministic FPTAS for computing the number of matchings in any graph with  $\Delta = O(1)$ . For the case of arbitrary graphs (with no restriction on the maximum degree) we construct a deterministic approximation scheme which runs in time  $\exp(O(\sqrt{n} \log^2 n))$ , where  $n$  the number of vertices in the graph.

The problem of computing the number of matchings in graphs with large girth was addressed recently by Bayati and Nair [BN06] in the context of Belief Propagation algorithm and the validity of the cavity method. The use of tree like recursions, similar to the one in this thesis, for computing the matching polynomials can also be found in the work of [God81]. The fact that the Gibbs distribution corresponding to matchings exhibits a spa-

tial correlation decay was already established by Heilmann and Lieb [OE72] by looking at complex roots of the partition function and later by van den Berg [vdB98] using more probabilistic/combinatorial arguments. We also note that the work of Kahn and Kim [KK98] is pertinent here. Using Godsil’s (self-avoiding) tree construction, they show that in any  $\Delta$ -regular graph, the probability that any fixed vertex is *not* in a (uniformly chosen) random matching is asymptotically  $1/\sqrt{\Delta}$ ; moreover, as we realized since doing the present work, their proof establishing this fact (see Claim (2.6) in their paper) also uses a two level recursion and partial derivatives to show convergence to the above probability estimate.

It should be noted that, while, as we mentioned, an FPRAS for counting matchings is known to exist thanks to the MCMC method, constructing a deterministic counterpart was an open problem, prior to this work, and in general constructing non-randomized approximate counting algorithms is a very challenging task.

#### 1.4.4 Contribution to computing the permanent of a matrix

In this thesis we establish the following two results. First we construct a polynomial time algorithm which for every  $\epsilon > 0$  provides factor  $(1 + \epsilon)^n$  multiplicative approximation for the permanent of a 0, 1 matrix, when the underlying graph is a constant degree expander. The definition of the latter is given in the subsequent section. Thus we significantly improve the  $e^n$  factor of [LSW00] for this class of graphs. While our algorithm requires polynomial time, it is not fully polynomial, as the term  $1/\epsilon$  appears in the exponent of the running time. Next we construct an algorithm providing the same factor  $(1 + \epsilon)^n$  approximation for the permanent of an *arbitrary* 0, 1 matrix. The running time of the algorithm is  $\exp(O(n^{\frac{2}{3}} \log^3 n))$ , namely the algorithm is mildly exponential. The main technical ingredient of our results is the reduction of the problem of computing a permanent to the problem of computing a partition function corresponding to the collection of all partial and full matchings of the graph. In this thesis we obtain the required bounds on the permanent of a graph in terms of the partition function of partial matchings of the same graph. Both of our results rely on techniques developed by Jerrum and Vazirani [JV96] for constructing a mildly exponential randomized

approximation algorithm, specifically we use their expander decomposition procedure. An important technical difficulty arising in our context is the fact that, while in [JV96] it sufficed to obtain appropriate bounds on the ratio of near perfect to perfect matchings, here we need to upper bound the ratio of the number of  $k$ -matchings to perfect matchings, for all  $k \leq n - 1$ . The straightforward application of the bound in [JV96] unfortunately gives a superexponential bound on this quantity and thus is of no use here. We circumvent this problem by utilizing a more careful counting argument. The difference between our argument and the argument of [JV96] will be highlighted in the body of this thesis.



# Chapter 2

## On deciding stability of multiclass queueing networks under buffer priority scheduling policies

This chapter contains our contribution to the problem of deciding the stability of queueing networks, and is organized as follows. The model description and the main result are provided in the following section. A background material on a Counter Machine and undecidability is given in Section 2.2. Section 2.3 is devoted to constructing a reduction from a Counter Machine to a queueing network. Section 2.4 is devoted to the proof of the main result. Some concluding thoughts and questions for further research are given in Section 2.5.

### 2.1 Model description and the main result

#### 2.1.1 Deterministic multiclass queueing networks and static buffer priority scheduling policy

A multiclass queueing network is described as a collection of  $J$  service nodes  $\sigma_1, \dots, \sigma_J$  and  $N$  job classes  $1, 2, \dots, N$ . Each node is assumed to be single-server type. Each class  $i$  is associated with a unique finite or infinite capacity buffer  $B_i$  which stores jobs corresponding

to this class, and is assigned to a unique server node, denoted by  $\sigma(i)$ . For simplicity we sometimes identify classes  $i$  with the corresponding buffers  $B_i$ . The number of jobs in buffer  $B_i$  at time  $s$  is the queue length corresponding to class  $i$  and is denoted by  $Q_i(s)$ . The total queue length  $\sum_{i \in \sigma_j} Q_i(s)$  corresponding to the server  $\sigma_j$  at time  $t$  is denoted by  $Q_{\sigma_j}(s)$ .

Each class  $i$  is associated with an external arrival process  $A_i(0, s)$  which denotes the total number of jobs which arrived externally to the buffer  $B_i$  during the time interval  $[0, s]$ . The arrival processes typically considered in the literature are either a random renewal process in the stochastic queueing networks literature or an adversarial process in the computer science literature. Throughout this chapter we adopt the following simple assumption: the intervals between the arrivals of jobs is a *deterministic* class dependent rational quantity  $a_i$  and the initial delay is some rational  $b_i$ . Thus the external arrivals corresponding to the class  $i$  occur exactly at times  $na_i + b, n = 0, 1, \dots$  and  $A_i(0, s) = \lceil (t - b)/a_i \rceil$ . Some classes may not have an associated external arrival process in which case  $a_i = \infty$  and  $A_i(0, s) = 0$  for all  $s \geq 0$ . We will also write  $A_i(s) = 1$  if there was an arrival at time  $s$  (that is  $s = a_i n + b_i$  for some  $n \in \mathbb{Z}_+$ ), and  $A_i(s) = 0$  otherwise. Each class  $i$  is associated with a deterministic service time  $0 \leq m_i < \infty$  which takes a non-negative rational values. It is possible that some of the service times are equal to zero. We say that at a given time  $s$  server is busy only if at time  $s$  it is working on a job which requires a non-zero service time. Specifically, for every collection of classes  $S$ , the associated workload  $W_S(s)$  at time  $s$  is the total time required to serve jobs which are presently in the network and which will *eventually* arrive into classes in  $S$ .

The routing of jobs in the network after the service completions is controlled as follows. A zero-one  $N$  by  $N$  sub-stochastic matrix  $R$  is fixed. Namely, the row sums of this matrix add up to at most unity and the spectral radius of this matrix is strictly less than unity. For every pair of classes  $i, l$  such that  $R_{i,l} = 1$ , every job which completes service in class  $i$  at some time  $s$  is immediately routed to buffer  $B_l$  after the service completion. If the buffer is not full  $Q_l(s) < B_l$ , then the job is added to the end of the queue in the buffer. If the buffer is full  $Q_l(s) = B_l$ , then the job is dropped from the network. If class  $i$  is such that  $R_{i,l} = 0$  for all  $l$ , then the jobs in class  $i$  after the service completion depart from the network.

The selection of jobs for processing is controlled using some *scheduling policy*  $\pi$ . In this thesis we consider exclusively a *static buffer priority scheduling policy*  $\pi$  which is described as follows. For each server  $\sigma_j$  a permutation  $\theta_j$  of the elements of classes belonging to  $\sigma_j$  is fixed.

At time  $s = 0$  and at every time instance  $s$  corresponding to the service completion in  $\sigma_j$ , the server  $\sigma_j$  finds the index  $i \in \sigma_j$  with the smallest value  $\theta_j(i)$  such that  $Q_i(s) > 0$ , selects the job in the head of this queue and begins working on it. If  $\sum_{i \in \sigma_j} Q_i(s) = 0$  then the server idles till the first time that a job appears in one of the classes and starts working on this job. The vector  $\theta = (\theta_j), 1 \leq j \leq J$  completely specifies the scheduling policy  $\pi$ . In particular, the scheduling policy is non-preemptive and non-idling. Static buffer priority policy is a widely studied scheduling policy [BPT94],[BNM99],[DW96],[DM97],[LK91],[KK94],[KM04],[BGT01],[RS92]

A queueing network, described by servers  $\sigma_j, 1 \leq j \leq J$ , classes  $i = 1, 2, \dots, N$ , the routing matrix  $R$ , interarrival times  $a_i$ , delays  $b_i$  and service times  $m_i$  will be denoted by  $\mathcal{Q}$  for brevity. The queueing network  $\mathcal{Q}$  together with the scheduling policy  $\pi$  and the vector of initial queue lengths  $(Q_i(0)), 1 \leq i \leq N$  completely determines the queue length dynamics of the network, namely the vector process  $Q(s) = (Q_i(s)), s \geq 0$ .

**Definition 2.** A triplet  $(\mathcal{Q}, \pi, Q(0))$  is defined to be stable if

$$\limsup_{s \geq 0} \sum_{1 \leq i \leq N} Q_i(s) < \infty, \quad (2.1)$$

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

In models with probabilistic settings,  $Q(s)$  is typically a stochastic process, in which case the queueing networks is defined to be stable if the process is so-called positive Harris recurrent [Dai95],[MT93],[CY01]. This usually implies the property  $\limsup_{s \geq 0} \sum_{1 \leq i \leq N} \mathbb{E}[Q_i(s)] < \infty$ . In our deterministic setting, however, this reduces to the simple condition (2.1). The principle goal of the stability research is developing algorithms for determining stability of a given triplet  $(\mathcal{Q}, \pi, Q(0))$  or a pair  $(\mathcal{Q}, \pi)$ . In many interesting special cases stability of  $(\mathcal{Q}, \pi)$  is implied by stability of  $(\mathcal{Q}, \pi, Q(0))$  for a given starting state  $Q(0)$ . For example, in the stochastic setting, this would be the case provided that the underlying Markov chain is irreducible. Due to the deterministic nature of our model, though, this implication does not necessarily hold and it is important to make the distinction. Our results apply only to the stability of triplets  $(\mathcal{Q}, \pi, Q(0))$ . We certainly expect that the problem of determining stability of pairs  $(\mathcal{Q}, \pi)$  is undecidable and leave it as an open problem. Note that undecidability of pairs  $(\mathcal{Q}, \pi)$  was established in [Gam02] for the class of generalized priority policies  $\pi$ .

### 2.1.2 The main result

The main result of this chapter is establishing the undecidability (non-computability) of stability property for the class of buffer priority policies  $\theta$ . Precisely stated

**Theorem 1.** *No algorithm can exist which on every input  $(\mathcal{Q}, \theta, Q(0))$  outputs YES if the triplet  $(\mathcal{Q}, \theta, Q(0))$  is stable and outputs NO otherwise, where  $\mathcal{Q}$  is an arbitrary multiclass queueing network,  $\theta$  is an arbitrary non-preemptive buffer priority scheduling policy, and  $Q(0)$  is an arbitrary vector of initial queue lengths. Namely, the underlying problem is undecidable.*

To prove Theorem 1, we introduce in Section 3 a counter Machine and its stability. Stability of a Counter Machine is a property tightly related to the so-called Halting Property (see Section 3), which is a classical undecidable property.

## 2.2 Counter Machine, Halting Problem and undecidability

A Counter Machine (see [BBK<sup>+</sup>01], [HU69]) is a deterministic computing machine which is a simplified version of a Turing Machine – a formal description of an algorithm performing a certain computational task or solving a certain decision problem. In his classical work on the Halting Problem, Alan Turing showed that certain decision problems simply cannot have a corresponding solving algorithm, and thus are undecidable. For a definition of a Turing Machine and the Turing Halting Problem see [Sip97]. Ever since many quite natural problems in mathematics and computer science were found to be undecidable, Hilbert’s tenth problem [Mat93] being one of the most notable examples. The famous Church-Turing thesis states that whatever is computable in principle can be computed by a Turing Machine. Thus undecidable problems, that is problems for which a Turing Machine cannot be built, are truly problems not allowing constructive solution.

More recently several undecidability results were obtained in the area of control theory, some of them using the device known as a Counter Machine, see Blondel et al. [BBK<sup>+</sup>01]. For a survey of decidability results in control theory area see Blondel and Tsitsiklis [BT00b]. We use the Counter Machine device as our reduction tool as well, and thus in the next subsection

we provide a detailed description of a Counter Machine and state relevant undecidability results.

### 2.2.1 Counter Machine and the Halting Problem

A Counter Machine is described by 2 counters  $R_1, R_2$  and a finite collection of states  $S$ . Each counter  $R_i$  contains some nonnegative integer  $z_i$  in its register. Depending on the current state  $s \in S$  and depending on whether the content of the registers is positive or zero, the Counter Machine is updated as follows: the current state  $s$  is updated to a new state  $s' \in S$  and one of the counters has its number in the register incremented by one, decremented by one or no change in the counters occurs.

Formally, a Counter Machine is a pair  $(S, \Gamma)$ .  $S = \{s_1, s_2, \dots, s_m\}$  is a finite set of states and  $\Gamma$  is configuration update function  $\Gamma : S \times \{0, 1\}^2 \rightarrow S \times \{(-1, 0), (0, -1), (0, 0), (1, 0), (0, 1)\}$ . A configuration of a Counter Machine is an arbitrary triplet  $(s, z_1, z_2) \in S \times \mathbb{Z}_+^2$ . A configuration  $(s, z_1, z_2)$  is updated to a configuration  $(s', z'_1, z'_2)$  as follows. Let  $1\{\cdot\}$  be the indicator function. Specifically, for every integer  $z$ ,  $1\{z\} = 1$  if  $z > 0$ , and  $= 0$  otherwise. Given the current configuration  $(s, z_1, z_2)$  suppose, for example  $\Gamma(s, 1\{z_1\}, 1\{z_2\}) = (s', 1, 0)$ . Then the current state is changed from  $s$  to  $s'$ , 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  $\Gamma : (s, z_1, z_2) \rightarrow (s', z_1 + 1, z_2)$  and  $\Gamma : s \rightarrow s', \Gamma : z_1 \rightarrow z_1 + 1, \Gamma : z_2 \rightarrow z_2$ . If  $\Gamma(s, 1\{z_1\}, 1\{z_2\}) = (s', (-1, 0))$ , then the current state becomes  $s'$ ,  $z'_1 = z_1 - 1, z'_2 = z_2$ . Similarly, if  $\Gamma(s, b) = (s', (0, 1))$  or  $\Gamma(s, b) = (s', (0, -1))$ , the new configuration becomes  $(s', z_1, z_2 + 1)$  or  $(s', z_1, z_2 - 1)$ , respectively. If  $\Gamma(s, b) = (s', (0, 0))$  then the state is updated to  $s'$ , but the contents of the counters do not change. It is assumed that the configuration update function  $\Gamma$  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 necessary for our purposes.

Given an initial configuration  $(s^0, z_1^0, z_2^0) \in S \times \mathbb{Z}_+^2$  the Counter Machine uniquely determines subsequent configurations  $(s^1, z_1^1, z_2^1), (s^2, z_1^2, z_2^2), \dots, (s^t, z_1^t, z_2^t), \dots$ . We fix a certain configuration  $(s^*, z_1^*, z_2^*)$  and call it the *halting* configuration. If this configuration is reached then the process halts and no additional updates are executed. The following theorem es-

establishes the undecidability (also called non-computability) of the halting property.

**Theorem 2.** *Given a Counter Machine  $(S, \Gamma)$ , initial configuration  $(s^0, z_1^0, z_2^0)$  and the halting configuration  $(s^*, z_1^*, z_2^*)$ , the problem of determining whether the halting configuration is reached in finite time (the Halting Problem) is undecidable. It remains undecidable even if the initial and the halting configurations are the same with both counters equal to zero:  $s^0 = s^*, z_1^0 = z_2^0 = z_1^* = z_2^* = 0$ .*

The first part of this theorem is a classical result and can be founded in [Hoo66]. The restricted case of  $s^0 = s^*, z_i^0 = z_i^*, i = 1, 2$  can be proven similarly by extending the set of states and the set of transition rules. It is the restricted case of the theorem which will be used in this thesis.

## 2.2.2 Simplified Counter Machine (SCM), stability and decidability

We say that a Counter Machine is stable if the value of counters is bounded as time goes to infinity. Namely  $\sup_t z_1^t < \infty, \sup_t z_2^t < \infty$ . It is shown in [Gam00] that determining whether a Counter Machine which started in a given configuration  $(s_1, 0, 0)$  is stable, is an undecidable problem, by a simple reduction to the Halting Problem.

**Definition 3.** *A simplified Counter Machine (SCM) is a Counter Machine satisfying the following condition: there exist two functions  $\alpha : S \times \{0, 1\}^2 \rightarrow S, \beta : S \rightarrow \{-1, 0, 1\}^2$ , such that  $\Gamma(s, z_1, z_2) = (\alpha(s, 1\{z_1 > 0\}, 1\{z_2 > 0\}), \beta((\alpha(s, 1\{z_1 > 0\}), 1\{z_2 > 0\}))$ ). In other words, while the new state  $s'$  depends on the entire current configuration  $(s, z_1, z_2)$ , the incrementing or decrementing of counters at the next step depends only on the new state  $s'$ .*

It turns out that this restrictive version of a Counter Machine is still sufficiently general for our purposes:

**Proposition 1.** *Given a Counter Machine, a SCM can be constructed, such that the SCM is stable if and only if the given Counter Machine is stable.*

*Proof.* We modify the state space  $\{s_j\}, 1 \leq j \leq m$  to  $\{s_j^{\text{odd}}\}_{1 \leq j \leq m} \cup \{(s_j^{\text{even}}, b_1, b_2)\}_{1 \leq j \leq m, b_1, b_2 \in \{-1, 0, 1\}}$ . The transition rules are defined as follows

$\alpha(s_j^{\text{odd}}, b_1, b_2) = (s_l^{\text{even}}, \Delta_1, \Delta_2)$ , if and only if  $\Gamma(s_j, b_1, b_2) = (s_l, \Delta_1, \Delta_2)$ , and  $\beta(s_l^{\text{even}}, \Delta_1, \Delta_2) = (\Delta_1, \Delta_2)$ . Also  $\alpha(s_l^{\text{even}}, \Delta_1, \Delta_2) = s_l^{\text{odd}}$  and  $\beta(s_l^{\text{odd}}) = (0, 0)$ . It is not hard to observe then that each transition  $(s_j, z_1, z_2) \rightarrow (s_l, z'_1, z'_2)$  with  $b_1 = z'_1 - z_1, b_2 = z'_2 - z_2$ , is emulated by two transitions in the SCM:  $(s_j^{\text{odd}}, z_1, z_2) \rightarrow ((s_l^{\text{even}}, b_1, b_2), z'_1, z'_2) \rightarrow (s_l^{\text{odd}}, z'_1, z'_2)$ .  $\square$

**Corollary 1.** *Determining the stability of SCMs with a given starting configuration  $s^*, z_1^* = 0, z_2^* = 0$  is an undecidable problem.*

## 2.3 Description of the queueing network corresponding to a SCM

Given an SCM with states  $\{s_1, s_2, \dots, s_m\}$  and counter update rules  $\alpha, \beta$ , we construct a certain queueing network, buffer priority policy and the vector of queue lengths at time zero. This network/policy/initial state triplet will have the property that it is stable if and only if the underlying SCM is stable, thus the reduction goal will be achieved.

We now proceed to the details of the construction. The queueing network consist of three subnetworks denoted respectively  $SN_1, SN_2$  and  $MN$ , which stand for *Subnetwork 1*, *Subnetwork 2*, and the *Main Network*, see Figures 2-1,2-2. The subnetwork  $SN_i, i = 1, 2$  will be in charge of the updates of the counter readings  $z_i$ . The network  $MN$  will be in charge of updating the state  $s_i$  of the SCM. We also describe the buffer priority scheduling policy implemented in this queueing network. The policy is denoted henceforth by  $\theta$ . Both in  $SN_i, i = 1, 2$  and in the main network  $MN$  the buffer capacities are all 0 or infinite.

The subnetworks  $SN_i, i = 1, 2$  are identical in their topological description. They will only differ in their buffer contents. Hence we only need to describe one of these subnetworks. On Figures 2-1,2-2 the buffers with infinite capacities are marked by a vertical bar.

### 2.3.1 The description of the subnetwork $SN_i, i = 1, 2$

The subnetwork  $SN_i$  consists of five servers,  $S_{ij}, j = 1, \dots, 5$  and several classes. (see Figure 2-1). The classes (buffers) corresponding to server  $S_{ij}$  are denoted by triplets  $ijk$ . Table 2.1 lists servers, classes (buffers), next classes, corresponding (deterministic) service

Server	Classes	Next Class	Service Time	Priority	Capacity
$S_{i1}$	$i11$	$i21$		2	$\infty$
	$i12$		.5	1	$\infty$
	$i13$	$i31$		3	0
	$i14$	$i31$		4	0
$S_{i2}$	$i21$		.5	1	$\infty$
	$i22$	$i12$		2	$\infty$
	$i23$	$i31$		3	0
$S_{i3}$	$i31$		.04	2	$\infty$
	$i32$		1.1	1	$\infty$
	$i33$	01 <i>i</i> of the network $MN$		3	0
$S_{i4}$	$i41$		.2	1	$\infty$
	$i42$	$i11$		2	0
$S_{i5}$	$i51$	$i11$	.02	1	$\infty$

Table 2.1: Servers and classes in  $SN_i$

times, priorities and the buffer capacities. Service times are shown in column 4 and only non-zero service times are shown. Thus the non-listed service time entries correspond to zero service time. For each class we also provide the next class to where the jobs are routed after the service completion. If the corresponding entry is empty, it means that the job leaves the network after the service completion. The fifth column corresponds to the priority of this class within the server. For examples the order of priority of classes in server  $S_{i1}$  is  $i12, i11, i13, i14$ , meaning  $i12$  has the highest priority,  $i11$  has the next highest priority, etc. The collection of classes  $i11, i12, i21, i22$  is defined to be "Rybko-Stolyar sub-network", or  $RSSN_i$ . It indeed describes the well-known Rybko-Stolyar network [RS92],[CY01].

There are seven external arrival processes into subnetwork  $SN_i$ , denoted by  $A_j^i(0, s)$ ,  $j = 1, \dots, 7$ . The corresponding information is summarized in Table 2.2. For each arrival process we describe exact arrival times as well as the class to which the arriving job is routed. For example the entry  $i42$  corresponding to the arrival process  $A_2^i$  indicates that the job arriving according to the arrival process  $A_2^i$  is routed to class  $i42$ . The arrival times are represented in the form  $an + b$  for some explicit constants  $a, b$ . Here  $a$  is the interarrival time and  $b$  is the initial delay. This means that for every non-negative integer  $n$ , an arrival occurs at time  $an + b$ .



Arrival process	Classes	Arrival times
$A_1^i$	$i22$	$n$
$A_2^i$	$i42$	$n + .02$
$A_3^i$	$i13$	$3n + 1.6$
$A_4^i$	$i23$	$3n + 2.1$
$A_5^i$	$i14$	$3n + 2.6$
$A_6^i$	$i32$	$3n + 1.5$
$A_7^i$	$i33$	$3n + 2.7$

Table 2.2: Arrival processes into  $SN_i$

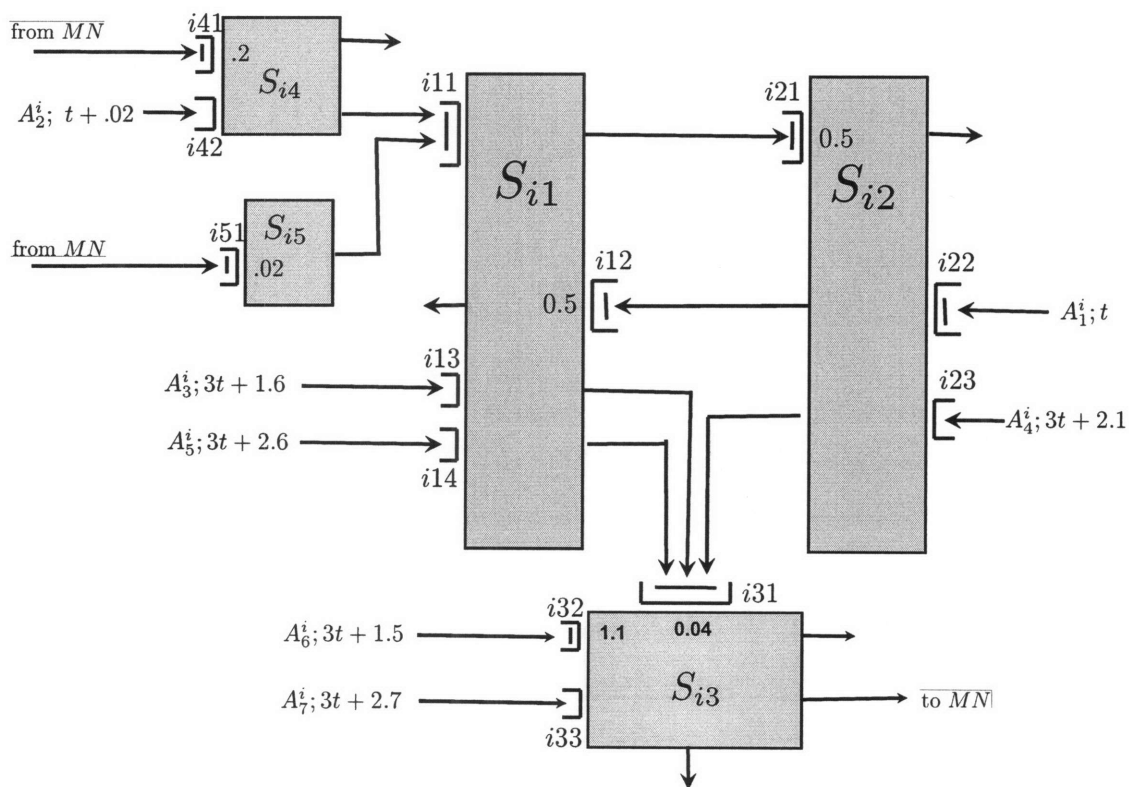


Figure 2-1: Subnetwork  $SN_i$

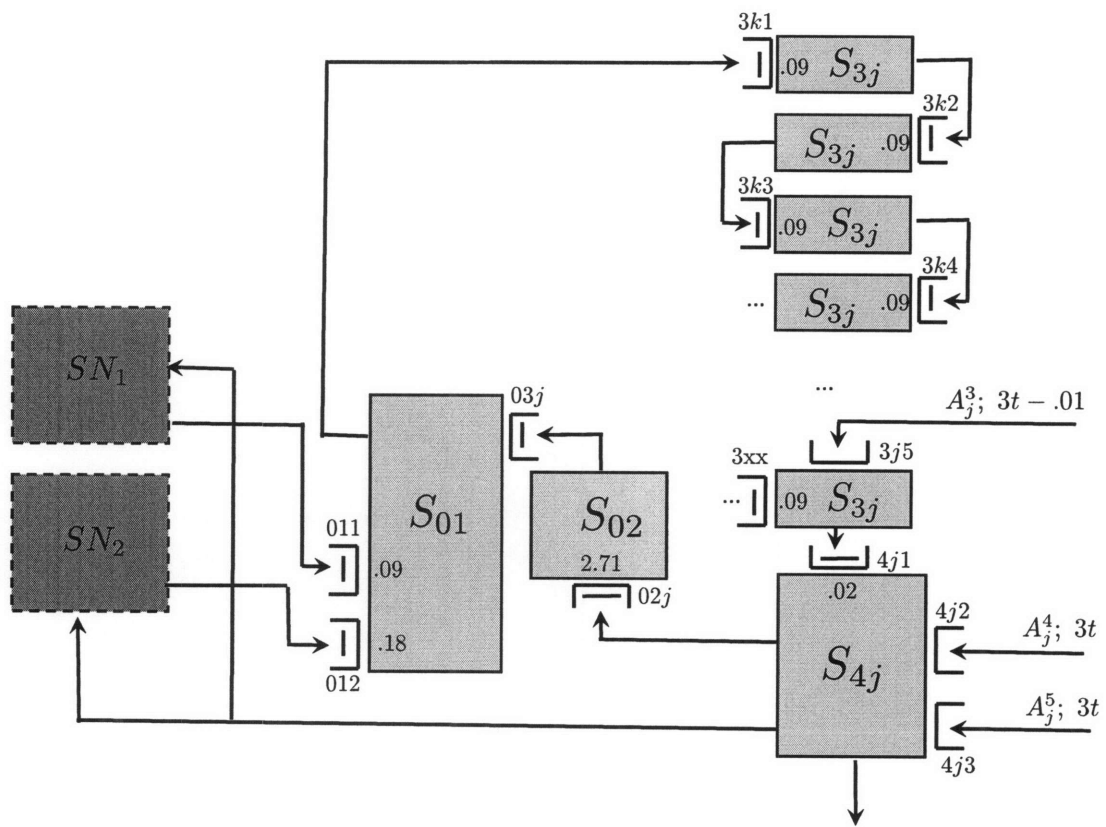


Figure 2-2: Main network  $MN$

### 2.3.2 The description of the main network $MN$

The main network consists of  $2m + 2$  servers, where  $m$  is the number of states in the SCM. The servers are  $S_{01}, S_{02}, S_{3j}, S_{4j}, j = 1, 2, \dots, m$ . The table describing, servers, classes, next classes, service times, priorities and buffer capacities is given below as Table 2.3. The interpretation is the same as for the table for subnetworks  $SN_i$ . Specific attention is paid to classes  $4j3$ ,  $1 \leq j \leq m$  and the next classes described generically as "i41, i51 or exit". The jobs departing from class  $4j3$  are routed to

1. class 141 if  $\beta(j) = (-1, 0)$ ;
2. class 151 if  $\beta(j) = (1, 0)$ ;
3. class 241 if  $\beta(j) = (0, -1)$ ;
4. class 251 if  $\beta(j) = (0, 1)$ ;
5. exit the network if  $\beta(j) = (0, 0)$ ;

In Table 2.3 some classes within the same server are assigned the same priority level. This means that the tie is broken arbitrarily. We prefer to assign the same priority level for simplicity. In reality, as we will see, the server will never have to prioritize between these classes, as at most one of the corresponding buffers will be non-empty. In order to avoid cluttering of the figure, the servers  $3j$  are described separately for classes  $3k1, 3k2, 3k3, 3k4$  and classes  $3j5$  although these belong to the same group of servers  $3j, j = 1, \dots, m$ . Arrivals into the main network are summarized in Table 2.4. There are  $3m$  external arrival processes into subnetwork  $MN$ , denoted by  $A_j^i(0, s), i = 3, 4, 5, j = 1, 2, \dots, m$ . We have started the index  $i$  from 3 to avoid confusion with arrival processes  $A_j^1, A_j^2$  in networks  $SN_i, i = 1, 2$ . The corresponding information is summarized in the table below. The arrival times are again represented in the form  $an + b$  for some explicit constants  $a, b$ .

We now describe the initial state of our queueing network at time  $s = 0$ , namely  $Q(0)$ . At this time there is one job in class  $02j$  in the main network, where  $j$  is such that  $s_j = s^*$  is the initial state of the SCM. The service is initiated at time  $s = 0$ , so the processing of this job will be over at time  $2.71$ . All other buffers in the queueing network are empty.

Server	Classes	Next Classes	Service Time	Priority	Capacity
$S_{01}$	011		.09	1	$\infty$
	012		.18	2	$\infty$
	all 03j	3j1		3	$\infty$
$S_{02}$	all 02j	03j	2.71	1	$\infty$
$S_{3j}$	3k1, for all $k$ s.t. $\alpha(s_k, 1, 1) = s_j$	3k2	.09	1	$\infty$
	3k2, for all $k$ s.t. $\alpha(s_k, 0, 1) = s_j$	3k3	.09	1	$\infty$
	3k3, for all $k$ s.t. $\alpha(s_k, 1, 0) = s_j$	3k4	.09	1	$\infty$
	3k4, for all $k$ s.t. $\alpha(s_k, 0, 0) = s_j$		.09	1	$\infty$
	3j5	4j1	.02	2	0
$S_{4j}$	4j1		.02	1	$\infty$
	4j2	02j		2	0
	4j3	i41, i51 or exit		3	0

Table 2.3: Servers and classes in  $MN$

Arrival process	Classes	Arrival times
$A_j^3$	3j5	$3n - .01$
$A_j^4$	4j2	$3n$
$A_j^5$	4j3	$3n$

Table 2.4: Arrival processes into  $MN$

## 2.4 Proof of Theorem 1

Our main result, Theorem 1 follows immediately from Corollary 1 and theorem below.

**Theorem 3.** *The queueing network constructed in the previous section with the prescribed initial state  $Q(0)$  is stable if and only if the SCM is stable.*

For the remainder of this chapter we focus on establishing Theorem 3. We first introduce the following definitions. Let  $W_i(s)$  be the combined workload of the servers  $S_{i1}, S_{i2}$  in the network  $SN_i$  at time  $s$ . Namely, it is the amount of service required to serve all jobs in servers  $S_{i1}, S_{i2}$  at time  $s$  when the scheduling policy  $\theta$  is implemented. Observe that  $W_i(s) = W_{i12}(s) + W_{i21}(s) + .5Q_{i22}(s) + .5Q_{i11}(s)$ , where  $W_{i12}(s)$  and  $W_{i21}(s)$  stand for the time required to process jobs currently in buffers  $i12, i21$  (if any) respectively. We will specifically focus on workloads  $W_i(s^-)$  where  $s^-$  indicates time immediately preceding  $s$ . Thus if there is an arrival at time  $s$ , this arrival is not showing up at  $s^-$ .

For every integer time instance  $t = 1, 2, \dots$ , we define the status of the main network  $MN$  to be the following quantity: for every  $k = 1, 2, \dots, m$ ,  $Status_{MN}(t) = k$  if at time  $t - 1$  server  $S_{02}$  of the network  $MN$  started working on a job in class  $02k$ , and there are no other jobs anywhere in the network at time  $t$ . Otherwise  $Status_{MN}(t) = -1$ .

For each  $i = 1, 2$  we also introduce status of the subnetwork  $SN_i$  at a given time  $3t + 1$  for  $t \in \mathbb{Z}_+$  as follows.  $Status_{SN_i}(3t + 1) = 2W_i((3t + 1)^-)$  if  $Q_{i12}(3t + 1)Q_{i21}(3t + 1) = 0$  and there are no jobs anywhere else in the subnetwork  $SN_i$ , that is other than the four classes of  $RSSN_i$ . Otherwise,  $Status_{SN_i}(3t + 1) = -1$ . We do not define  $Status_{SN_i}(t)$  at other values of  $t$ . As we will see shortly the status functions at time  $3t + 1$  will represent the configuration of the SCM at time  $t$ . Provided that we have initialized our queueing network appropriately, the status functions will never take values  $-1$ .

**Theorem 4.** *If the configuration of the SCM after  $t$  steps is  $(s_q, z_1, z_2)$ , then  $Status_{MN}(3t + 1) = q$  and  $Status_{SN_i}(3t + 1) = z_i$ ,  $i = 1, 2$ .*

*Proof.* The proof is by induction. For  $t = 0$ , the statement of Theorem 4 holds because the queueing network initialization makes it so. The remainder of this chapter is devoted to proving the induction step. It is given in Section 5.1.  $\square$

We now show how this result implies Theorem 3.

*Proof of Theorem 3.* The idea of the proof is to show that a bound on the value of counters of SCM implies a bound on the number of jobs in the queueing network at any one time, and vice versa.

Suppose SCM is stable. That means that there is a bound  $M$  on the maximum value of counters, so that  $z_1$  and  $z_2$  never exceed  $M$ . Let  $(s_j, z_1, z_2)$  be the configuration of the SCM at time  $t$ . Then by Theorem 4, at time  $(3t + 1)^-$  there are  $z_1 \leq M$  jobs in  $SN_1$ ,  $z_2 \leq M$  jobs in  $SN_2$ , and one job in the main network. So at time  $(3t + 1)^-$  there can be no more than  $2M + 1$  jobs in the queueing network. Since there is only a constant number of arrival processes in the network and the arrival process is deterministic, then for every time period  $[3t + 1, 3(t + 1) + 1)$  the total number of jobs in the network is bounded by  $2M + C$  for some constant  $C$  which only depends on the network parameters. Thus if SCM is stable, so is the queueing network.

Conversely, suppose the network is stable and at any time  $t$ , the total number of jobs in the network does not exceed  $M$  for some finite value  $M$ . Then  $M$  is also an upper bound on  $Status_{SN_i}(3t + 1)$  for every  $t$ . By Theorem 4, this implies that the values  $z_1, z_2$  of the counters of SCM are bounded by  $M$  and therefore the SCM is also stable.  $\square$

### 2.4.1 Proof of the induction step of Theorem 4

This subsection proves induction step of Theorem 4. Thus we assume that its statement holds after  $t$  steps, and prove that it holds after  $t + 1$  steps. Assume that the configuration of the SCM at time  $t$  is  $(s_q, z_1, z_2)$ ;  $Status_{MN}(3t + 1) = q$ ,  $Status_{SN_i}(3t + 1) = z_i$ ,  $i = 1, 2$ . Assume that the configuration of SCM at time  $t + 1$  is  $\Gamma(s_q, z_1, z_2) = (s_r, y_1, y_2)$ . We need to show that  $Status_{MN}(3t + 4) = r$ ,  $Status_{SN_i}(3t + 4) = y_i$ ,  $i = 1, 2$ .

#### Dynamics in subnetwork $SN_i$

**Lemma 1.** *For every time  $s \geq 0$  either  $Q_{i12}(s) = 0$  or  $Q_{i21}(s) = 0$ . Moreover,  $\dot{W}_i(s) = -1$ , whenever  $W_i(s) > 0$  and  $s \in \mathbb{R}_+$  is not an instance of arrivals into servers  $S_{i1}, S_{i2}$ .*

**Remark :** The first part of the lemma is a well-known fact from the stability literature, stating that the classes  $i12, i21$  constitute a *virtual server* such that only one of the two classes can be served at any given time [DV00],[DHV99].

*Proof.* Suppose the statement of the lemma does not hold. Then let  $u = \inf\{s : Q_{i12}(s) > 0 \text{ and } Q_{i21}(s) > 0\}$ . That means that both buffers  $i12$  and  $i21$  are non-empty at time  $u^+$ , but at least one of the two is empty at time  $u^-$ . Suppose this holds for buffer  $i12$ . This implies that there was an (instantaneous) service completion in buffer  $i22$  at time  $u$ . Class  $i21$  has higher priority than class  $i22$  (consult Table 2.1). This implies that the server  $S_2$  was not working on the job in class  $i21$  at time  $u^-$ . Since however, class  $i21$  is non-empty at time  $u^+$ , then we conclude that there was an arrival into buffer  $i21$  exactly at time  $u$ . We conclude that there was a simultaneous arrival into buffers  $i12$  and  $i21$  at time  $u$  and buffers  $i12$  and  $i21$  were empty at time  $u^-$ .

Now we show that such a thing is impossible. Since jobs arrive to  $i12$  from  $i22$  and into  $i22$  from outside at integer times  $n$ , we see that  $u$  must take integer values. We now obtain a contradiction. The jobs arrive into  $i11$  only from classes  $i42$  and  $i51$ . Jobs arriving into  $i42$  arrive from outside at non-integer times  $n + .02$ . Buffer  $i42$  has no capacity and the processing time for this class is zero. Therefore, these jobs can ultimately arrive into  $i21$  only at times  $n + .02$  and not integer times. Jobs arriving into  $i51$  have a non-zero processing time  $.02$ . These jobs arrive from the main network  $MN$  from classes  $4j3$  which correspond to zero capacity buffers and zero processing times. Jobs arrive into  $4j3$  from outside at integer times  $3n$ . Thus these jobs can ultimately arrive into class  $i21$  only at times  $3n + .02$  and not integer times. We conclude that jobs cannot ever arrive into  $i21$  at integer times.

Similarly we consider the case when  $Q_{i21}(u^-) = 0$ . Since  $Q_{i21}(u^+) > 0$  then there was a service completion in buffer  $i11$  at time  $u$ . We already showed above that this can only occur at times of the form  $n + .02$ . Also this means  $Q_{i12}(u^-) = 0$ , since class  $i12$  has higher priority than class  $i11$ . Thus there was an arrival into  $i12$  at time  $u$ , namely there was a service completion in  $i22$  at time  $u$ . Since  $Q_{i21}(u^-) = 0$  and service time in  $i22$  is zero, there was arrival into  $i22$  at  $u$ . But these arrivals occur only at integer times  $n$ . Again we obtain a contradiction.

To establish the last part regarding  $\dot{W}_i(s)$  observe that only jobs in buffers  $i12, i21$  have non-zero processing times. Since only one of these buffers can contain a job, the case  $W_i(s) > 0$  corresponds to the case of exactly one of these buffers having jobs, as otherwise, if both  $i12, i21$  are empty, the remaining jobs in servers  $S_{i1}, S_{i2}$  are processed immediately since they have zero service time requirement. The assertion then follows.  $\square$

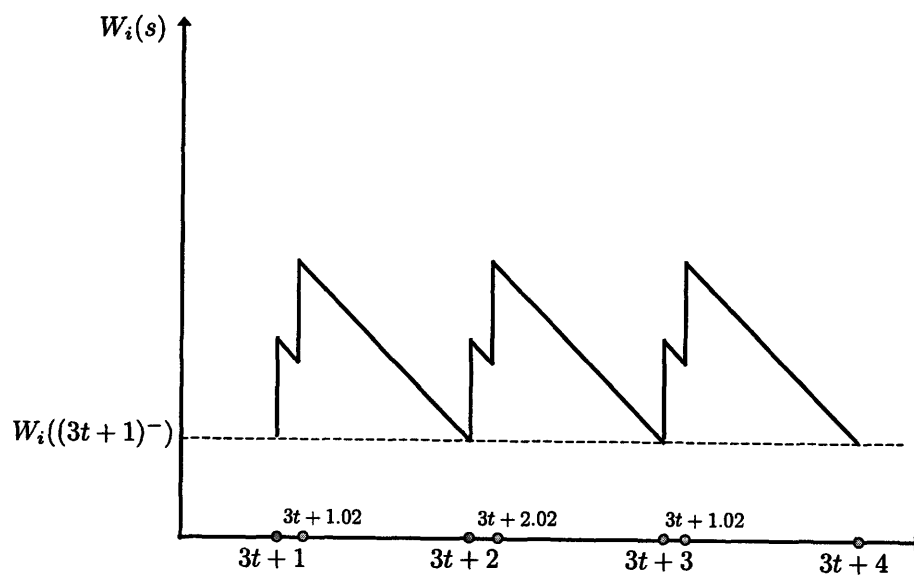


Figure 2-3: Workload  $W_i(s)$ . Case 1



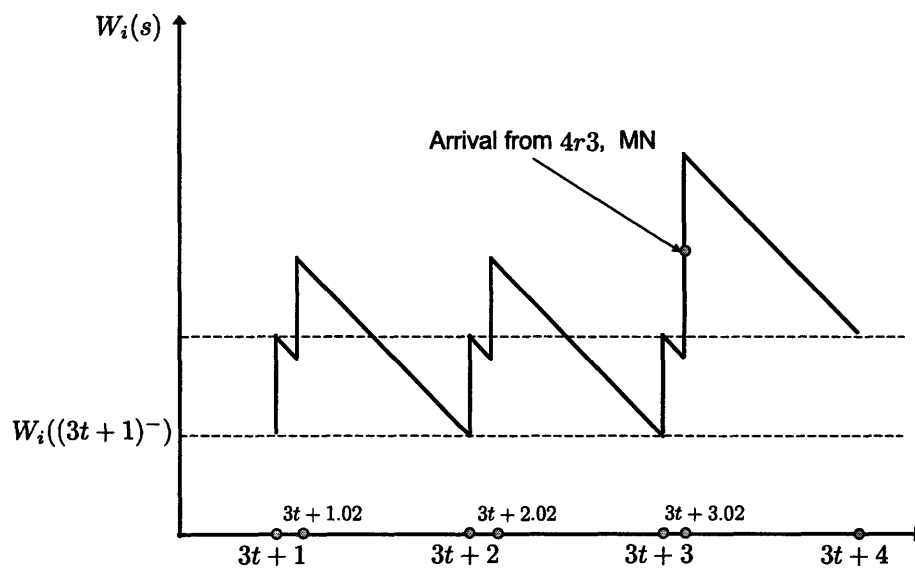


Figure 2-4: Workload  $W_i(s)$ . Case 2

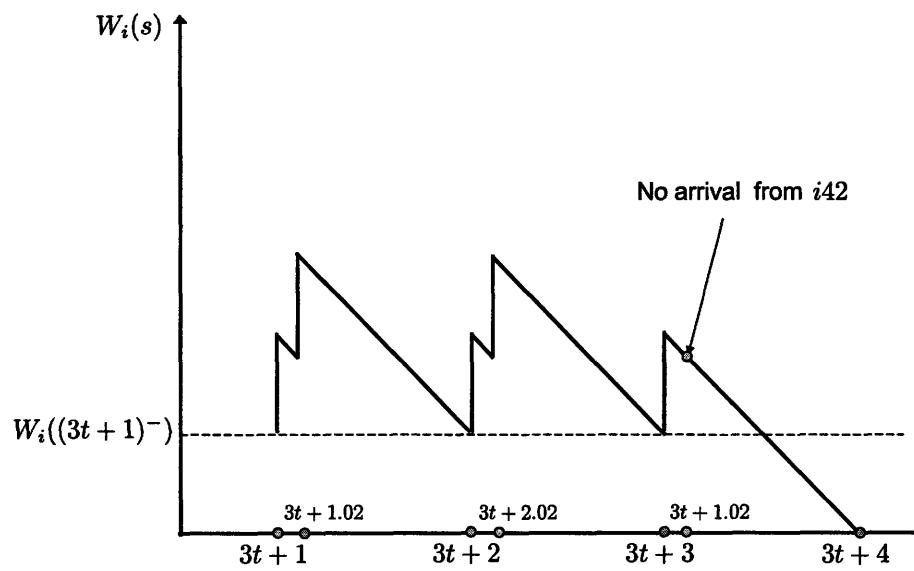


Figure 2-5: Workload  $W_i(s)$ . Case 3

**Lemma 2.** *There are no arrivals into buffers  $i41, i51$  during the time interval  $[3t+1, 3t+3)$ .*

*Proof.* Arrivals into class  $A_{i41}$  and  $A_{i51}$  can happen as a result of a departure from one of the classes  $4j3$  of the network  $MN$ . The buffers  $4j3$  have zero capacity and zero processing time. Therefore service completions happen there simultaneously with arrivals from arrival processes  $A_j^5$ . But those arrivals occur only at times  $3t$ . Thus the first arrival after  $3t$  can occur only at time  $3t+3$ . The assertion then follows.  $\square$

**Lemma 3.** *During the time interval  $[3t+1, 3t+3)$ , exactly one of servers  $S_{i1}$  and  $S_{i2}$  is busy, and  $W_i((3t+2)^-) \geq W_i((3t+1)^-)$ . In addition, during this time period, jobs in classes  $i12$  and  $i21$  finish service only at times which are multiples of .5.*

*Proof.* By Lemma 1, at most one of servers  $S_{i1}, S_{i2}$  does work at any given time. Thus we need to show that at least one server works during this time period.

By Lemma 2 there are no arrivals into buffers  $i41, i51$  during  $[3t+1, 3t+3)$ . By the inductive assumption  $Status_{SN_i}(3t+1) = z_i \geq 0$ , implying in particular that there are no jobs in buffer  $i41$  at time  $3t+1$ . Thus buffer  $i41$  is empty during  $[3t+1, 3t+3)$ . This means that the jobs arriving into class  $i42$  at times  $3t+1.02$  and  $3t+2.02$  will arrive instantly into buffer  $i11$ . Also one job will arrive into  $i22$  at time  $3t+1, 3t+2$ . By Lemma 1 only one of the jobs in buffers  $i12, i21$  can be served at a time. Thus the dynamics of the number of jobs in the subnetwork  $RSSN_i$  can be viewed as dynamics of a single server queue with service time .5 and arrivals at times  $3t+1, 3t+1.02, 3t+2, 3t+2.02$ . It is easy then to construct explicitly  $W_i(s)$  during the time period  $s \in [3t+1, 3t+3)$ , given the initial value  $W_i((3t+1)^-)$ , and the graph of  $W_i(s)$  is depicted on Figures 2-3,2-4,2-5. The part  $[3t+1, 3t+3)$  is identical in all of the three figures. The differing parts of the graph corresponding to the interval  $[3t+3, 3t+4)$  will be used later on in Subsection 2.4.1. In particular we see that if  $W_i((3t+1)^-) > 0$ , then  $W_i(s)$  is always positive during the time interval  $[3t+1, 3t+3)$ , and if  $W_i((3t+1)^-) = 0$ , then  $W_i(s)$  is equal to zero only at time  $s = 3t+2$ , In particular at least one (and therefore exactly one) of the servers  $S_{i1}, S_{i2}$  was busy during the time interval  $[3t+1, 3t+3)$ . We also see by inspection that  $W_i((3t+2)^-) \geq W_i((3t+1)^-)$ . Finally, by the inductive assumption  $Status_{SN_i}(3t+1) = z_i = 2W_i((3t+1)^-)$ , in particular it is an integer. This means there is no service in progress in buffers  $i12, i21$  at time  $3t+1$ . Thus, whether or not there are prior jobs in buffers  $i12, i21$  at time  $3t+1$ , there will be service completions exactly at times

$3t + 1.5, 3t + 2, 3t + 2.5$  and  $3t + 3$ , as seen again by inspecting Figures 2-3,2-4,2-5. This proves the second assertion of the lemma.  $\square$

**Lemma 4.** *Suppose  $Status_{SN_i}(3t + 1) \geq 1$ . Then the job  $\mathcal{J}$  arriving at time  $3t + 2.7$  from outside according to the arrival process  $A_i^j$  will be routed to buffer  $01i$  of the network  $MN$  at time  $3t + 2.7$ .*

*Proof.* At time  $3t + 1.5$  a job arrives into class  $i32$  which requires 1.1 processing time. Since  $i32$  is the highest priority class in server  $S_{i3}$ , then this server will be busy until time  $3t + 2.6$ . Also the highest priority of this class implies that there is only one job of this class at a time. Thus at time  $3t + 2.6$  buffer  $i32$  is empty. Buffer  $i31$  has the second highest priority and buffer  $i33$  to where the job  $\mathcal{J}$  arrives has the lowest priority. Thus, whether  $\mathcal{J}$  will be blocked from service at arrival time  $3t + 2.7$  then depends on the number of jobs in buffer  $i31$  at time  $3t + 2.7$ . The processing time for these jobs is .04. Therefore  $\mathcal{J}$  will not be blocked if and only if there are at most two jobs in  $i31$  since then these jobs will be processed not later than  $3t + 2.6 + .04 + .04 < 3t + 2.7$ , and otherwise they will be processed at time  $3t + 2.6 + .04 + .04 + .04 > 3t + 2.7$  or later. We conclude that  $\mathcal{J}$  will be blocked if and only if there are at most two jobs in buffer  $i31$ . We now show that this is indeed the case provided  $Status_{SN_i}(3t + 1) \geq 1$ .

Jobs arriving into buffer  $i31$  depart from classes  $i13, i14$  and  $i23$ . These buffers have zero capacity and zero service time. Therefore they can arrive into  $i31$  only at a time of arrival into these three buffers namely at times  $3t + 1.6, 3t + 2.1$  and  $3t + 2.6$ . In particular there will be up to three jobs in buffer  $i31$  at time  $3t + 2.6$ . Thus we need to show that it is impossible for all of these three jobs to arrive into  $i31$ . We will show that at least one of these jobs is blocked. By Lemma 3 either server  $S_{i1}$  or  $S_{i2}$  is busy during  $[3t + 1, 3t + 3)$ . Suppose job arriving into  $i13$  at time  $3t + 1.6$  is not blocked. This means  $S_{i2}$  is busy at time  $3t + 1.6$ . By Lemma 3 it will remain busy till  $3t + 2$ . If it remains busy after this time then it will remain busy till  $3t + 2.5$ , the job arriving into  $i23$  at time  $3t + 2.1$  is blocked and the assertion is established. Thus the only remaining possibility that  $S_{i2}$  finishes service at time  $3t + 2$  and remains idle after this. We will show that then a job arriving into  $i14$  at time  $3t + 2.6$  will be blocked and then the proof is complete. By Lemma 3  $W_i((3t + 2)^-) \geq W_i((3t + 1)^-) \geq 1$ . Thus there is at least one job either in  $S_{i1}$  or  $i21$  at time  $(3t + 2)^-$  which still requires .5 processing time. We claim that at time  $(3t + 2)^+$  it is in  $i12$ . Indeed it cannot be in  $i12$  since server is idle at this time. For the same reason it cannot be in  $i22$  since service time

in this buffer is zero. Also it cannot be in  $i11$  since  $S_{i1}$  was idle at  $(3t + 2)^-$  and the arrivals into  $i11$  do not occur at integer times. We conclude that there is at least one job in  $i12$  at time  $(3t + 2)^+$  and no jobs in  $i11, i21, i22$  at this time. At time  $3t + 2$  there is an arrival into  $i22$  which then immediately proceeds to  $i12$ . Thus we have at least two jobs in  $i12$  at time  $(3t + 2)^+$ . The server will work on them during  $[3t + 2, 3t + 3)$  and will block a job arriving into  $i14$  at time  $3t + 2.6$ . This completes the proof.  $\square$

**Lemma 5.** *Suppose  $Status_{SN_i}(3t + 1) = 0$ . Then a job  $\mathcal{J}$  arriving at time  $3t + 2.7$  from outside according to the arrival process  $A_7^i$  will exit the system immediately.*

*Proof.* The proof is very similar to the proof of the previous lemma. We need to show that all three jobs arriving into classes  $i13, i23, i14$  at times  $3t + 1.6, 3t + 2.1$  and  $3t + 2.6$ , respectively, will not be blocked and will be in buffer  $i31$  at time  $3t + 2.6$ . Suppose  $Status_{SN_i}(3t + 1) = 0$ . Namely  $W_i((3t + 1)^-) = 0$ . Then the job arriving at time  $3t + 1$  into buffer  $i22$  according to  $A_1^i$  will immediately proceed to buffer  $i12$  and occupy server  $S_{i1}$  during the time interval  $(3t + 1, 3t + 1.5)$ . By Lemma 2 the job arriving into buffer  $i24$  at time  $3t + 1.02$  according to  $A_2^i$  will be processed immediately in buffer  $i42$  and proceed to buffer  $i11$ . It will be delayed in buffer  $i11$  till  $3t + 1.5$  and at this time will depart to buffer  $i21$  and occupy server  $S_{i2}$  during the time interval  $(3t + 1.5, 3t + 2)$ . Then again a job arriving at  $3t + 2$  into  $i22$  will proceed into  $i12$  and occupy the server  $S_{i1}$  during the time interval  $(3t + 2, 3t + 2.5)$ . Finally, the job arriving into  $i42$  at time  $3t + 2.02$  will be delayed in  $i11$  till  $3t + 2.5$  and then it will occupy  $S_{i2}$  during  $(3t + 2.5, 3t + 3)$ . It is clear from this dynamics that all of the three jobs arriving at times  $3t + 1.6, 3t + 2.1$  and  $3t + 2.6$  into buffers  $i13, i23, i14$  will be processed immediately and arrive into buffer  $i31$  at the same times  $3t + 1.6, 3t + 2.1$  and  $3t + 2.6$ .  $\square$

Combining the results of Lemmas 4 and 5 we obtain the following conclusion.

**Corollary 2.** *Exactly one job arrives into the class  $01i$  of network  $MN$  at time  $3t + 2.7$  if and only if  $Status_{SN_i}(3t + 1) \geq 1$ .*

### Dynamics in $MN$

We now switch to the analysis of the dynamics in network  $MN$ . Recall that by the inductive assumption  $Status_{MN}(3t + 1) = q$ , we have one job in class  $02q$  at time  $3t + 1$  which started service at time  $3t$ , and there are no other jobs in  $MN$  at time  $3t + 1$ . We call this unique job

$\mathcal{K}$ . Recall, that the configuration  $(q, x_1, x_2)$  of the SCM at time  $t$  is assumed to be updated to the configuration  $(r, y_1, y_2)$  at time  $t + 1$ . Introduce  $m_1 = \alpha(s_q, 1, 1)$ ,  $m_2 = \alpha(s_q, 0, 1)$ ,  $m_3 = \alpha(s_q, 1, 0)$ ,  $m_4 = \alpha(s_q, 0, 0)$ . Namely,  $m_1, m_2, m_3, m_4$  are the four possible values of the state  $r$ .

**Lemma 6.** *During the time interval  $(3t + 2.98, 3t + 3.07)$  the job  $\mathcal{K}$  will be in server  $3r$ , buffer  $3m_4$  (respectively buffer  $3m_3$  or  $3m_2$  or  $3m_1$ ) if and only if  $x_1 = x_2 = 0$  (respectively if and only if  $x_1 = 1, x_2 = 0$  or  $x_1 = 0, x_2 = 1$  or  $x_1 = x_2 = 0$ ). This job will leave the network before time  $3t + .34$ .*

*Proof.* By the inductive assumption the job  $\mathcal{K}$  will finish service in buffer  $02q$  at time  $3t + 2.71$  and will arrive into buffer  $03q$ . It will possibly experience a delay in the corresponding server  $S_{01}$  which depends on the presence/absence of jobs in buffers  $011, 012$ . We now consider four possible cases.

1. Case  $x_1 = x_2 = 0$ . By the inductive assumption this means  $Status_{SN_1}(3t + 1) = Status_{SN_2}(3t + 1) = 0$ . By Corollary 2 this means that at time  $3t + 2.7$  no jobs arrive into buffers  $011, 012$ . Since only jobs arriving from buffer  $i33$ , that is ultimately from  $A_7^i$  can possibly get into buffers  $011, 012$ , then these buffers are empty at least till  $3(t + 1) + 2.7$ . In particular the job  $\mathcal{K}$  arriving into  $03q$  at time  $3t + 2.71$  will find an idle server and will proceed immediately to buffers  $3m_1, 3m_2, 3m_3, 3m_4$ . In each of these buffers it has the highest priority. Since the service time in each of these buffers is  $.09$ , then it will arrive into these four buffers exactly at times  $3t + 2.71, 3t + 2.8, 3t + 2.89, 3t + 2.98$ . In particular it will be in buffer  $3m_4$  during the time interval  $(3t + 2.98, 3t + 3.07)$  and the assertion is established.
2. Case  $x_1 = 1, x_2 = 0$ . By the inductive assumption this means  $Status_{SN_1}(3t + 1) > 0, Status_{SN_2}(3t + 1) = 0$ . By Corollary 2 this means that at time  $3t + 2.7$  no job arrives into buffers  $012$  and one job arrives into buffer  $011$ . This job has the highest priority and requires  $.09$  processing time. The only difference with the previous case is then that the job  $\mathcal{K}$  now experiences a delay  $.09$  in server  $S_{01}$ . Thus it will arrive into buffers  $m_1, m_2, m_3, m_4$  exactly at times  $3t + 2.8, 3t + 2.89, 3t + 2.98, 3t + 3.07$ . In particular it will be in buffer  $3m_3$  during the time interval  $(3t + 2.98, 3t + 3.07)$  and the assertion is established.

3. Case  $x_1 = 0, x_2 = 1$ . The analysis is similar. We observe that we will have one job in buffer 012 and no jobs in buffer 011 at time  $3t + 2.7$ . This buffer 012 has the second highest priority, the job  $\mathcal{K}$  will experience the delay  $.18$  - the processing time of a job in buffer 012.
4. Case  $x_1 = x_2 = 1$ . The analysis is similar. In this case we have one job in buffer 011 and one job in buffer 012. The job  $\mathcal{K}$  is delayed by  $.18 + .09 = .27$  time units.

Finally, we see again by considering the four cases that the job  $\mathcal{K}$  will depart from the network at time  $3t + 3.34$  the latest. This completes the proof of the lemma.  $\square$

**Lemma 7.** *At time  $(3t + 3)^-$ , the server  $S_{4r}$  is idle, and the servers  $S_{4j}, j \neq r$  are busy processing jobs in buffers  $4j1$ .*

*Proof.* At time  $(3t + 3)^-$  the servers  $S_{4j}$  can be busy only serving jobs in buffer  $4j1$ . These jobs arrive from zero capacity buffer  $3j5$ . These jobs have the highest priority in server  $S_{4j}$  and the second highest in  $S_{3j}$ . Also these jobs arrive at time  $3(t+1) - .01$  into  $3j5$ . The only way for these jobs to be dropped from zero capacity buffer  $3j5$  is by higher priority buffer in these server, that is one serving possibly job  $\mathcal{K}$ , to be occupied. By Lemma 6 this is the case exactly for one server, namely server  $3r$ .  $\square$

**Lemma 8.**  $Status_{MN}(3t + 4) = r$ .

*Proof.* We need to show that at time  $3t + 4$  in network  $MN$  there is one job in class  $02r$  which initiated service at time  $3t + 3$  and no jobs elsewhere. By Lemma 6, the job  $\mathcal{K}$  will leave the network before time  $3t + 3.34 < 3t + 4$ . The jobs arriving into zero capacity buffers  $4j2, 4j3, j \neq r$  at time  $3t + 3$  will find, by Lemma 7 a busy server  $4j$  and will be dropped from the network. The job arriving into buffer  $4r3$  at time  $3t + 3$  will find by Lemma 7 an idle buffer and will immediately proceed to one of the subnetworks  $SN_i$ . The jobs arriving into buffers  $3j5$  at time  $3t + 3 - .01$  will either be dropped from the network or will proceed to buffers  $4j1$  and after an additional service time  $.02$  will leave the network. Thus they will leave the network before time  $3t + 3 + .01 < 3t + 4$ . We conclude that only the job arriving into buffer  $4r2$  at time  $3t + 3$  can remain in the network. By Lemma 7 it will find an idle server  $S_{4r}$  and will proceed immediately to buffer  $02r$  and begin service there at time  $3t + 3$ . This completes the proof.  $\square$

**Lemma 9.** *There are no arrivals into classes  $i41, i51$  during the time period  $[3t + 1, 3t + 4]$  other than possibly at time  $3t + 3$ . At time  $3t + 3$  at most one job arrives into four classes  $141, 151, 241, 251$ . Specifically,*

1.  $A_{141}(3t + 3) = 1$  if  $\beta(s_r) = (-1, 0)$ ;
2.  $A_{151}(3t + 3) = 1$ , if  $\beta(s_r) = (1, 0)$ ;
3.  $A_{241}(3t + 3) = 1$ , if  $\beta(s_r) = (0, -1)$ ;
4.  $A_{251}(3t + 3) = 1$ , if  $\beta(s_r) = (0, 1)$ ;
5. No arrivals, if  $\beta(s_r) = (0, 0)$ .

*Proof.* Arrivals into  $i42, i52$  can occur only from buffers  $4j3$ . These buffers have zero capacity and zero processing times. The arrivals into these buffers occurs at times  $3n$ ,  $n = 0, 1, \dots$ . By Lemma 7 only server  $4r$  will process a job at time  $3t + 3$  in buffer  $4r3$ . According to Table 2.3 and the corresponding description it will be routed to one of the buffers  $i41, i51$  or leave the network precisely as described by the lemma.  $\square$

**Lemma 10.** *The following holds for each  $i = 1, 2$ :*

1.  $Status_i(3t + 4) = Status_i(3t + 1)$  if  $A_{i41}(3t + 3) = A_{i51}(3t + 3) = 0$ ;
2.  $Status_i(3t + 4) = Status_i(3t + 1) - 1$  if  $A_{i41}(3t + 3) = 1$ ;
3.  $Status_i(3t + 4) = Status_i(3t + 1) + 1$  if  $A_{i51}(3t + 3) = 1$ ;

*Proof.* By Lemma 1 we have  $Q_{i12}(3t + 4)Q_{i21}(3t + 4) = 0$ . Let us show that at time  $3t + 4$  there are no jobs in  $SN_i$  other than possibly  $RSSN_i$ . By the inductive assumption we have  $Status_{SN_i}(3t + 1) \geq 0$ . In particular at this time there are no jobs in  $SN_i$  outside of  $RSSN_i$ . We need to show that no jobs arriving during  $(3t + 1, 3t + 4]$  can be outside of  $RSSN_i$  at time  $3t + 4$ .

By Lemma 9 jobs can arrive into  $i41, i51$  during  $(3t + 1, 3t + 4]$  only at time  $3t + 3$  and only one such job can arrive. Upon arrival they will experience service time either  $.2$  in  $i41$  or  $.02$  in buffer  $i51$  and thus will leave the network before time  $3t + 3.2$  the latest.



The jobs arriving into  $i42$  at times  $3t + 2, 3t + 3, 3t + 4$  either will be dropped or proceed to buffer  $i11$  which is a part of  $RSSN_i$ . Thus at time  $3t + 4$  these jobs either will be in  $RSSN_i$  or leave the network (no jobs in  $RSSN_i$  feed buffers outside of  $RSSN_i$ ).

We have already analyzed the dynamics of the jobs arriving into buffers  $i13, i14, i23, i32, i33$  at times  $3t + 1.6, 3t + 2.1, 3t + 2.6$  as a part of the proofs of Lemmas 4,5. In particular, we saw that these jobs leave  $SN_i$  before time  $3t + 2.72$ . We have established that there are no jobs in  $SN_i$  outside of  $RSSN_i$  at time  $3t + 4$ .

It remains to analyze the value of  $Status_i$  at time  $3t + 4$ . We consider the corresponding three cases.

1.  $A_{i41}(3t + 3) = A_{i51}(3t + 3) = 0$ . Then, by Lemma 9 there were no arrivals into classes  $i41, i51$  in time interval  $[3t + 1, 3t + 4]$ . Consider the quantity  $W_i(s)$  during this time interval. As long as  $W_i(s) > 0$ , by Lemma 1,  $\dot{W}_i(s) = -1$  at time instances  $s$  not corresponding to the arrival instances. But we have arrivals into  $i22$  at times  $3t + 1, 3t + 2,$  and  $3t + 3$ , and into  $i42$  at times  $3t + 1 + .02, 3t + 2 + .02$  and  $3t + 3 + .02$ , ensuring that  $W_i(s)$  is not 0 for any period of positive length during  $[3t + 1, 3t + 4]$ , (consult Figure 2-3). In this situation,  $W_i(s)$ , over the time interval  $[3t + 1, 3t + 4]$  increases by 3 units due to 6 arrivals, and decreases by 3 due to 6 service completions. Thus  $W_i((3t + 4)^-) = W_i((3t + 1)^-)$ .
2.  $A_{i51}(3t + 3) = 1$ . Then, the job arriving into  $i51$  at time  $3t + 3$  after a delay of .02 will arrive into  $i11$ , thus increasing  $W_i(s)$  by .5 at time  $s = 3t + 3.02$  (consult Figure 2-4). Therefore  $W_i((3t + 4)^-) = W_i((3t + 1)^-) + .5$  and  $Status_{SN_i}(3t + 4) = Status_{SN_i}(3t + 1) + 1$ .
3.  $A_{i41}(3t + 3) = 1$ . Then the job arriving into  $i41$  at time  $3t + 3$  will occupy server  $S_{i4}$  for .2 time units. As a result the job arriving into  $i42$  at time  $3t + 3.02$  will find a busy server and will be dropped from the network. Comparing this situation with the case  $A_{i41}(3t + 3) = A_{i51}(3t + 3) = 0$ , and consulting Figure 2-5, we obtain the same situation, except that there are no arrival into  $i11$  at time  $3t + 3.02$ . The net result is  $W((3t + 4)^-) = W((3t + 1)^-) - .5$  and  $Status_{SN_i}(3t + 4) = Status_{SN_i}(3t + 1) - 1$

This completes the proof. □

As an immediate corollary of Lemma 9 and Lemma 10 we obtain

**Corollary 3.**  $Status_1(3t + 4) = y_1$ , and  $Status_2(3t + 4) = y_2$ .

Lemmas 8 and Corollary 3 prove the induction step for Theorem 4, so its proof is now complete. □

## 2.5 Conclusion

We have shown that there does not exist an algorithm for determining stability of multiclass queueing networks operating under the class of static non-preemptive buffer priority scheduling policies. Namely, the underlying problem is undecidable. There are, however, special cases for which the stability can be determined. Characterization of those special cases is of interest. Also of interest is whether the above result holds for FIFO scheduling policy, another frequently studied scheduling policy. Our model made several simplifying assumptions. We allow for some buffers to be finite and we allow zero service times. We have little doubt that the stability property remains undecidable even without these assumptions, but at present we do not have a proof.

# Chapter 3

## Counting list colorings of a graph

This chapter contains our contribution to the problem of counting the number of list colorings of a graph, and has the following structure. The model description and the main result are stated in Section 3.1. Some preliminary technical results are established in Section 3.2. The description of the algorithm and its complexity are subject of Section 3.3. The principal technical result is established in Section 3.4. The key result is Theorem 6, which establishes the correlation decay result on a computation tree arising in computing the marginals of the uniform distribution on the set of all list colorings. Section 3.5 provides a brief comparison between the correlation decay result on a computation tree and the correlation decay in conventional sense. Some conclusions and open problems are in Section 3.6.

### 3.1 Definitions and the main result

We consider a simple graph  $\mathbb{G}$  with the node set  $V = \{v_1, v_2, \dots, v_{|V|}\}$ . Our graph is assumed to be triangle-free, namely the girth  $g$  (the size of the smallest cycle) is at least 4. Let  $E$  denote the set of edges,  $\Delta(v)$  denote the degree of the node  $v$ , and  $\Delta(\mathbb{G}) = \max_v \Delta(v)$  denote the degree of the graph. Each node  $v$  is associated with a list of colors  $L(v) \subset \{1, 2, \dots, q\} = \cup_{v \in V} L(v)$ , where  $\{1, 2, \dots, q\}$  is the total universe of colors. We let  $\mathbf{L} = (L(v), 1 \leq v \leq n)$  denote the vector of lists. We also let  $\|\mathbf{L}\| = \max_v |L(v)|$  be the size of the largest list. The list-coloring problem on  $\mathbb{G}$  is formulated as follows: associate each node  $v$  with a color  $c(v) \in L(v)$  such that no two nodes sharing an edge are associated with the same color.

When all the lists are identical and contain  $q$  elements, the corresponding problem is the problem of coloring  $\mathbb{G}$  using  $q$  colors. We let  $|L(v)|$  denote the cardinality of  $L(v)$ . It is easy to see that if

$$|L(v)| \geq \Delta(v) + 1 \tag{3.1}$$

for every node  $v$ , then a simple greedy procedure produces a list-coloring. We adopt here a stronger assumption

$$|L(v)| \geq \alpha\Delta(v) + \beta, \tag{3.2}$$

where  $\alpha$  is an arbitrary constant strictly larger than  $\alpha^{**}$ , the unique solution of  $\alpha^{**} \exp(-\frac{1}{\alpha^{**}}) = 2$ . That is  $\alpha^{**} \approx 2.8432\dots$ . We also assume that  $\beta$  is a large constant which depends on  $\alpha$ . To be more specific we assume that  $\beta = \beta(\alpha)$  is large enough to satisfy

$$(1 - \frac{1}{\beta})\alpha e^{-\frac{1}{\alpha}(1+\frac{1}{\beta})} > 2, \tag{3.3}$$

which is always possible when  $\alpha > \alpha^{**}$ .

Let  $Z(\mathbb{G}, \mathbf{L})$  denote the total number of possible list-colorings of a graph/list pair  $(\mathbb{G}, \mathbf{L})$ . The corresponding counting problem is to compute (approximately)  $Z(\mathbb{G}, \mathbf{L})$ . In statistical physics terminology,  $Z(\mathbb{G}, \mathbf{L})$  is the partition function. We let  $Z(\mathbb{G}, \mathbf{L}, \chi)$  denote the number of list colorings of  $(\mathbb{G}, \mathbf{L})$  which satisfy some condition  $\chi$ . For example  $Z(\mathbb{G}, \mathbf{L}, c(v) = i, c(u) = j)$  is the number of list colorings such that the color of  $v$  is  $i$  and the color of  $u$  is  $j$ .

On the space of all list colorings of  $\mathbb{G}$  we consider a uniform probability distribution, where each list coloring assumes weight  $1/Z(\mathbb{G}, \mathbf{L})$ . For every node/color pair  $v \in V, i \in L(v)$ ,  $\mathbb{P}_{\mathbb{G}, \mathbf{L}}(c(v) = i)$  denotes the probability that node  $v$  is colored  $i$  with respect to this probability measure. The size of the instance corresponding to a graph/list pair  $(\mathbb{G}, \mathbf{L})$  is defined to be  $n = \max\{|V|, |E|, q\}$ .

For convenience, we restate here the definition of a FPTAS.

**Definition 4.** *An approximation algorithm  $\mathcal{A}$  is defined to be a Fully Polynomial Time Approximation Scheme (FPTAS) for a computing  $Z(\mathbb{G}, \mathbf{L})$  if given arbitrary  $\delta > 0$  it produces*

a value  $\hat{Z}$  satisfying

$$1 - \delta \leq \frac{\hat{Z}}{Z(\mathbb{G}, \mathbf{L})} \leq 1 + \delta,$$

in time which is polynomial in  $n, \frac{1}{\delta}$ .

We now state our main result.

**Theorem 5.** *There exist a deterministic algorithm which provides a FPTAS for computing  $Z(\mathbb{G}, \mathbf{L})$  for arbitrary graph list pair  $\mathbb{G}, \mathbf{L}$  satisfying (3.2), when the size of the largest list  $\|\mathbf{L}\|$  is constant. The same algorithm has complexity  $2^{O(\log^2 n)}$ , without any restriction on  $\|\mathbf{L}\|$ , where  $n$  is the size of the instance.*

## 3.2 Preliminary technical results

### 3.2.1 Basic recursion

We begin by establishing a standard relationship between the partition function  $Z(\mathbb{G}, \mathbf{L})$  and the marginals  $\mathbb{P}_{\mathbb{G}, \mathbf{L}}(c(v) = i)$ . The relation, also known as cavity method, is also the basis of the Glauber dynamics approach for computing partition functions.

**Proposition 2.** *Consider an arbitrary list coloring  $i_1, \dots, i_{|V|}$  of the graph  $\mathbb{G}$  (which can be constructed using a simple greedy procedure). For every  $k = 0, 1, \dots, |V| - 1$  consider a graph list pair  $\mathbb{G}_k, \mathbf{L}_k$ , where  $(\mathbb{G}_0, \mathbf{L}_0) = (\mathbb{G}, \mathbf{L})$ ,  $\mathbb{G}_k = \mathbb{G} \setminus \{v_1, \dots, v_k\}$ ,  $k \geq 1$  and the list  $\mathbf{L}_k$  is obtained by deleting from each list  $L(v_l)$ ,  $l > k$  a color  $i_r$ ,  $r \leq k$  if  $(v_l, v_r) \in E$ . Then*

$$Z(\mathbb{G}, \mathbf{L}) = \prod_{0 \leq k \leq |V| - 1} \mathbb{P}_{\mathbb{G}_k, \mathbf{L}_k}^{-1}(c(v_k) = i_k).$$

*Proof.* We have

$$\mathbb{P}_{\mathbb{G}, \mathbf{L}}(c(v_1) = i_1) = \frac{Z(\mathbb{G}, \mathbf{L}, c(v_1) = i_1)}{Z(\mathbb{G}, \mathbf{L})} = \frac{Z(\mathbb{G}_1, \mathbf{L}_1)}{Z(\mathbb{G}, \mathbf{L})},$$

from which we obtain

$$Z(\mathbb{G}, \mathbf{L}) = \mathbb{P}_{\mathbb{G}, \mathbf{L}}(c(v_1) = i_1)^{-1} Z(\mathbb{G}_1, \mathbf{L}_1).$$

Iterating further for  $k \geq 2$  we obtain the result.  $\square$

Our algorithm is based on a recursive procedure which relates the number of list colorings of a given graph/list pair in terms of the number of list colorings of some reduced graph/list pairs.

Given a pair  $(\mathbb{G}, \mathbf{L})$  and a node  $v \in \mathbb{G}$ , let  $v_1, \dots, v_m$  be the set of neighbors of  $v$ . For every pair  $(k, i) \in \{1, \dots, m\} \times L(v)$  we define a new pair  $(\mathbb{G}_v, \mathbf{L}_{k,i})$  as follows. The set of nodes of  $\mathbb{G}$  is  $V_k = V \setminus \{v\}$  and  $L_{k,i}(v_r) = L(v_r) \setminus \{i\}$  for  $1 \leq r < k$ ,  $L_{k,j}(u) = L(u)$  for all other  $u$ . Namely, we first delete node  $v$  from the graph. Then we delete color  $i$  from the lists corresponding to the nodes  $v_r, r < k$ , and leave all the other lists intact.

**Lemma 11.** *The graph/list pair  $(\mathbb{G}_v, \mathbf{L}_{k,j})$  satisfies (3.2) for every  $1 \leq k \leq m, j \in L(v)$ , provided that  $(\mathbb{G}, \mathbf{L})$  does.*

*Proof.* When we create graph  $\mathbb{G}_v$  from  $\mathbb{G}$  the list size of every remaining node either stays the same or is reduced by one. The second event can only happen for neighbors  $v_1, \dots, v_m$  of the deleted node  $v$ . When the list is reduced by one the degree is reduced by one as well. Since  $\alpha > 1$ , the assertion follows by observing that  $|L(v_k)| \geq \alpha \Delta(v_k) + \beta$  implies  $|L(v_k)| - 1 \geq \alpha(\Delta(v_k) - 1) + \beta$ .  $\square$

The basis of our algorithm is the following simple result.

**Proposition 3.** *Given a graph/list pair  $(\mathbb{G}, \mathbf{L})$  and a node  $v$ , suppose  $\Delta(v) = m > 0$ . For every  $i \in L(v)$*

$$\mathbb{P}_{\mathbb{G}, \mathbf{L}}(c(v) = i) = \frac{\prod_{1 \leq k \leq m} (1 - \mathbb{P}_{\mathbb{G}_v, \mathbf{L}_{k,i}}(c(v) = i))}{\sum_{j \in L(v)} \prod_{1 \leq k \leq m} (1 - \mathbb{P}_{\mathbb{G}_v, \mathbf{L}_{k,j}}(c(v) = j))}. \quad (3.4)$$

The recursion as well as the proof is similar to the one used by Weitz in [Wei06], except we bypass the construction of a self-avoiding tree, considered in [Wei06].

*Proof.* Consider a graph/list  $(\mathbb{G}_v, \mathbf{L})$  obtained simply by removing node  $v$  from  $\mathbb{G}$ , and leaving  $\mathbf{L}$  intact for the remaining nodes. We have

$$\begin{aligned}
\mathbb{P}_{\mathbb{G}, \mathbf{L}}(c(v) = i) &= \frac{\mathbb{P}_{\mathbb{G}, \mathbf{L}}(c(v) = i)}{\sum_{j \in L(v)} \mathbb{P}_{\mathbb{G}, \mathbf{L}}(c(v) = j)} \\
&= \frac{Z(\mathbb{G}, \mathbf{L}, c(v) = i) Z^{-1}(\mathbb{G}, \mathbf{L})}{\sum_{j \in L(v)} Z(\mathbb{G}, \mathbf{L}, c(v) = j) Z^{-1}(\mathbb{G}, \mathbf{L})} \\
&= \frac{Z(\mathbb{G}_v, \mathbf{L}, c(v_k) \neq i, 1 \leq k \leq m)}{\sum_{j \in L(v)} Z(\mathbb{G}_v, \mathbf{L}, c(v_k) \neq j, 1 \leq k \leq m)} \\
&= \frac{\mathbb{P}_{\mathbb{G}_v, \mathbf{L}}(c(v_k) \neq i, 1 \leq k \leq m)}{\sum_{j \in L(v)} \mathbb{P}_{\mathbb{G}_v, \mathbf{L}}(c(v_k) \neq j, 1 \leq k \leq m)}
\end{aligned}$$

Now, for every  $j \in L(v)$

$$\mathbb{P}_{\mathbb{G}_v, \mathbf{L}}(c(v_k) \neq j, 1 \leq k \leq m) = \mathbb{P}_{\mathbb{G}_v, \mathbf{L}}(c(v_1) \neq j) \prod_{2 \leq k \leq m} \mathbb{P}_{\mathbb{G}_v, \mathbf{L}}(c(v_k) \neq j | c(v_r) \neq j, 1 \leq r < k)$$

We observe that  $\mathbf{L}_{1,j} = \mathbf{L}$  for every  $j$  (no colors are removed due to the vacuous condition  $r < 1$ ), and  $\mathbb{P}_{\mathbb{G}_v, \mathbf{L}}(c(v_k) \neq j | c(v_r) \neq j, 1 \leq r < k) = \mathbb{P}_{\mathbb{G}_v, \mathbf{L}_{k,j}}(c(v_k) \neq j)$ . Namely

$$\mathbb{P}_{\mathbb{G}_v, \mathbf{L}}(c(v_k) \neq j, 1 \leq k \leq m) = \prod_{1 \leq k \leq m} \mathbb{P}_{\mathbb{G}_v, \mathbf{L}_{k,j}}(c(v_k) \neq j) = \prod_{1 \leq k \leq m} (1 - \mathbb{P}_{\mathbb{G}_v, \mathbf{L}_{k,j}}(c(v_k) = j)).$$

Substituting this expression we complete the proof.  $\square$

### 3.2.2 Upper and lower bounds

The condition (3.2) allows us to obtain the following simple bounds.

**Lemma 12.** *For every  $\mathbb{G}, \mathbf{L}$ , node  $v$  and a color  $i \in L(v)$*

$$\mathbb{P}_{\mathbb{G}, \mathbf{L}}(c(v) = i) \leq \frac{1}{\beta}.$$

*Proof.* Observe that given an arbitrary coloring of the neighbors  $v_1, \dots, v_m$  of  $v$ , there are at least  $|L(v)| - \Delta(v) \geq \beta$  colors remaining. Then the upper bound holds.  $\square$

From this simple bound we now establish a different upper bound and also a lower bound

using the triangle free assumption.

**Lemma 13.** *There exist  $\epsilon_0 = \epsilon_0(\alpha) \in (0, 1)$  and  $\beta > 0$  such that for every  $\mathbb{G}, \mathbf{L}$ , node  $v$  and a color  $i \in L(v)$*

$$q^{-1}(1 - \beta^{-1})^\Delta \leq \mathbb{P}_{\mathbb{G}, \mathbf{L}}(c(v) = i) \leq \frac{1}{2\Delta(v)(1 + \epsilon_0)}.$$

We note that the upper bounds of this lemma and Lemma 12 are not comparable, since values of  $\Delta(v)$  could be smaller and larger than  $\beta$ .

*Proof.* We let  $v_1, \dots, v_m$  denote the neighbors of  $v$ ,  $m = \Delta(v)$  and let  $v_{kr}$  denote the set of neighbors of  $v_k$ , other than  $v$  for  $k = 1, \dots, m$ . We will establish that for any coloring of nodes  $(v_{kr})$ , which we generically denote by  $\mathbf{c}$ , we have

$$q^{-1}(1 - \beta^{-1})^\Delta \leq \mathbb{P}_{\mathbb{G}, \mathbf{L}}(c(v) = i | \mathbf{c}) \leq \frac{1}{2m(1 + \epsilon_0)}.$$

The corresponding inequality for the unconditional probability then follows immediately. Now observe that, since the girth is at least 4, then there are no edges between  $v_k$ . Then  $\mathbb{P}_{\mathbb{G}, \mathbf{L}}(c(v) = i | \mathbf{c})$  is the probability  $\mathbb{P}_{\mathbb{T}}(c(v) = i)$  that  $v$  is colored  $i$  in a depth-1 tree  $\mathbb{T} \triangleq \{v, v_1, \dots, v_m\}$ , where the lists  $\hat{L}(v_k)$  of  $v_k$  are obtained from  $L(v_k)$  by deleting the colors used by the neighbors  $v_{kr}$  by coloring  $\mathbf{c}$ . From the assumption (3.2) we have that the remaining lists  $\hat{L}(v_k)$  have size at least  $|L(v_k)| - \Delta(v_k) \geq \beta$  each. Let  $t_i = \mathbb{P}_{\mathbb{T}}(c(v) = i)$ . For each color  $j \in L(v)$  let  $t_{j,k} = 1/|\hat{L}(v_k)|$  if  $j \in \hat{L}(v_k)$  and  $= 0$  otherwise. Proposition 3 then simplifies to

$$t_i = \frac{\prod_{1 \leq k \leq m} (1 - t_{i,k})}{\sum_{j \in L(v)} \prod_{1 \leq k \leq m} (1 - t_{j,k})} \leq \frac{1}{\sum_{j \in L(v)} \prod_{1 \leq k \leq m} (1 - t_{j,k})}, \quad (3.5)$$

for every  $i \in L(v)$ , where  $\prod_{1 \leq k \leq m}$  is defined to be equal to unity when  $m = 0$ . From the equality part, applying  $t_{j,k} \leq 1/\beta$ , we get

$$t_i \geq |L(v)|^{-1} (1 - \beta^{-1})^m \geq q^{-1} (1 - \beta^{-1})^\Delta,$$

and the lower bound is established.

We now focus on the upper bound and use the inequality part of (3.5). Thus it suffices



to show that

$$\sum_{j \in L(v)} \prod_k (1 - t_{j,k}) \geq 2(1 + \epsilon_0)m \quad (3.6)$$

for some constant  $\epsilon_0 > 0$ . Using the first order Taylor expansion for  $\log z$  around  $z = 1$ ,

$$\begin{aligned} \prod_{1 \leq k \leq m} (1 - t_{j,k}) &= \prod_{1 \leq k \leq m} e^{\log(1-t_{j,k})} \\ &= \prod_{1 \leq k \leq m} e^{-t_{j,k} - \frac{1}{2(1-\theta_{j,k})^2} t_{j,k}^2}, \end{aligned}$$

for some  $0 \leq \theta_{j,k} \leq t_{j,k}$ , since  $-1/z^2$  is the second derivative of  $\log z$ . Again using the bound  $t_{j,k} \leq 1/\beta$ , we have  $(1 - \theta_{j,k})^2 \geq (1 - 1/\beta)^2$ . We assume that  $\beta$  is a sufficiently large constant ensuring  $(1 - 1/\beta)^2 > 1/2$ . Thus we obtain the following lower bound

$$\prod_{1 \leq k \leq m} (1 - t_{j,k}) \geq \prod_{1 \leq k \leq m} e^{-t_{j,k} - \frac{t_{j,k}^2}{2(1-1/\beta)^2}} \geq e^{-(1+\frac{1}{\beta})\sum_k t_{j,k}} \triangleq e^{-(1+\frac{1}{\beta})T_j},$$

where  $T_j$  stands for  $\sum_k t_{j,k}$ . Then

$$\sum_{j \in L(v)} \prod_{1 \leq k \leq m} (1 - t_{j,k}) \geq \sum_{j \in L(v)} e^{-(1+\frac{1}{\beta})T_j} \geq |L(v)| e^{-\frac{1}{|L(v)|}(1+\frac{1}{\beta})\sum_j T_j},$$

where we have used an inequality between the average arithmetic and average geometric. Finally we observe

$$\sum_{j \in L(v)} T_j = \sum_{j,k} t_{j,k} = \sum_{1 \leq k \leq m} \sum_{j \in \hat{L}(v_k)} \frac{1}{|\hat{L}(v_k)|} = m.$$

Thus

$$\sum_{j \in L(v)} \prod_{1 \leq k \leq m} (1 - t_{j,k}) \geq |L(v)| e^{-\frac{m}{|L(v)|}(1+\frac{1}{\beta})} \geq (\alpha m + \beta) e^{-\frac{1}{\alpha}(1+\frac{1}{\beta})} > \alpha m e^{-\frac{1}{\alpha}(1+\frac{1}{\beta})}$$

The condition  $\alpha > \alpha^{**}$  implies that there exists a sufficiently large  $\beta$  such that  $\alpha e^{-\frac{1}{\alpha}(1+\frac{1}{\beta})} > 2$ . We find  $0 < \epsilon_0 < .1$  such that  $\alpha e^{-\frac{1}{\alpha}(1+\frac{1}{\beta})} = 2(1 + \epsilon_0)$ . We obtain a required lower bound

(3.6).

□

### 3.3 Algorithm and complexity

#### 3.3.1 Description of an algorithm

Our algorithm is based on the idea of trying to approximate the value of  $\mathbb{P}_{\mathbb{G}, \mathbf{L}}(c(v) = i)$ , by performing a certain recursive computation using (3.4) a fixed number of times  $d$  and then using a correlation decay principle to guarantee the accuracy of the approximation. Specifically, introduce a function  $\Phi$  which takes as an input a vector  $(\mathbb{G}, \mathbf{L}, v, i, d)$  and takes some values  $\Phi(\mathbb{G}, \mathbf{L}, v, i, d) \in [0, 1]$ . The input  $(\mathbb{G}, \mathbf{L}, v, i, d)$  to  $\Phi$  is any vector, such that such that  $v$  is a node in  $\mathbb{G}$ ,  $i$  is an arbitrary color, and  $d$  is an arbitrary non-negative integer. Function  $\Phi$  is defined recursively in  $d$ . The quantity  $\Phi$  "attempts" to approximate  $\mathbb{P}_{\mathbb{G}, \mathbf{L}}(c(v) = i)$ . The quality of the approximation is controlled by  $d$ . We define  $\Phi$  as follows. For every input  $(\mathbb{G}, \mathbf{L}, v, i, d)$  such that  $i \notin L(v)$  we set  $\Phi(\mathbb{G}, \mathbf{L}, v, i, d) = 0$ . Otherwise we set the values as follows.

- When  $d = 0$ , we set  $\Phi(\mathbb{G}, \mathbf{L}, v, i, d) = 1/|L(v)|$  for every input  $(\mathbb{G}, \mathbf{L}, v, i)$ . (It turns out that for our application the initialization values are not important, due to the decay of correlations).
- For every  $d \geq 1$ , if  $\Delta(v) = 0$ , then  $\Phi(\mathbb{G}, \mathbf{L}, v, i, d) = 1/|L(v)|$  for all  $i \in L(v)$ . Suppose  $\Delta(v) = m > 0$  and  $v_1, \dots, v_m$  are the neighbors of  $v$ . Then for every  $i \in L(v)$  we define

$$\Phi(\mathbb{G}, \mathbf{L}, v, i, d) = \min \left[ \frac{1}{2(1 + \epsilon_0)m}, \frac{1}{\beta}, \frac{\prod_{1 \leq k \leq m} (1 - \Phi(\mathbb{G}_v, \mathbf{L}_{k,i}, v_k, i, d - 1))}{\sum_{j \in L(v)} \prod_{1 \leq k \leq m} (1 - \Phi(\mathbb{G}_v, \mathbf{L}_{k,j}, v_k, j, d - 1))} \right]. \quad (3.7)$$

The last part of the expression inside  $\min[\cdot]$  corresponds directly to the expression (3.4) of Proposition 3. Specifically, if it was true that  $\Phi(\mathbb{G}_v, \mathbf{L}_{k,j}, v_k, j, d - 1) = \mathbb{P}_{\mathbb{G}_v, \mathbf{L}_{k,j}}(c(v_k) = j)$ , then, by Lemmas 12,13, the minimum in (3.7) would be achieved by the third expression, and then the value of  $\Phi(\mathbb{G}, \mathbf{L}, v, i, d)$  would be exactly  $\mathbb{P}_{\mathbb{G}, \mathbf{L}}(c(v) = i)$ .

We will use the correlation decay property to establish that the difference between the two values, modulo rescaling, is diminishing as  $d \rightarrow \infty$ . Note that the computation of  $\Phi$  can be done recursively in  $d$  and it involves a dynamic programming type recursion. The underlying computation is done essentially on a tree of graph list pairs  $\mathbb{G}_s, \mathbf{L}_s$  generated during the recursion. We refer to this tree as *computation tree* with depth  $d$ .

We now describe our algorithm for approximately computing  $Z(\mathbb{G}, \mathbf{L})$ . The algorithm is parametrized by the "quality" parameter  $d$ .

### Algorithm CountCOLOR

INPUT: A graph/list pair  $(\mathbb{G}, \mathbf{L})$  and a positive integer  $d$ .

BEGIN

Set  $\hat{Z} = 1, \hat{\mathbb{G}} = \mathbb{G}, \hat{\mathbf{L}} = \mathbf{L}$ .

While  $\hat{\mathbb{G}} \neq \emptyset$ , find an arbitrary node  $v \in \hat{\mathbb{G}}$  and a color  $i \in \hat{\mathbf{L}}(v)$ . Compute

$$\hat{p}(v, i) \triangleq \Phi(\hat{\mathbb{G}}, \hat{\mathbf{L}}, v, i, d). \tag{3.8}$$

Set  $\hat{Z} = \hat{p}^{-1}(v, i)\hat{Z}, \hat{\mathbb{G}} = \hat{\mathbb{G}} \setminus \{v\}, \hat{\mathbf{L}}(u) = \hat{\mathbf{L}}(u) \setminus \{i\}$  for all neighbors  $u$  of  $v$  in  $\hat{\mathbb{G}}$ , and  $\hat{\mathbf{L}}(u)$  remains the same for all other nodes.

END

OUTPUT:  $\hat{Z}$ .

### 3.3.2 Some properties

We now establish some properties of  $\Phi$ .

**Lemma 14.** *The following holds for every  $\mathbb{G}, \mathbf{L}, v, i \in L(v), d \geq 0$ .*

$$\Phi(\mathbb{G}, \mathbf{L}, v, i, d) \leq \min \left[ \frac{1}{\beta}, \frac{1}{2(1 + \epsilon_0)\Delta(v)} \right], \quad (3.9)$$

$$\sum_{i \in L(v)} \Phi(\mathbb{G}, \mathbf{L}, v, i, d) \leq 1, \quad (3.10)$$

$$\Phi(\mathbb{G}, \mathbf{L}, v, i, d) \geq q^{-1}(1 - 1/\beta)^\Delta. \quad (3.11)$$

*Proof.* (3.10) follows directly from the definition of  $\Phi$ . To show (3.9) we consider cases. For  $d \geq 1$  this follows directly from the recursion (3.7). For  $d = 0$ , this follows since  $\Phi(\mathbb{G}, \mathbf{L}, v, i, 0) = 1/|L(v)| \leq 1/(\alpha\Delta(v) + \beta)$  and  $2(1 + \epsilon_0) < 2.2 < \alpha$ . We now establish (3.11). For the case  $d = 0$  this follows since  $1/|L(v)| \geq 1/q$ . For the case  $d \geq 1$  this follows from the recursion (3.7) since  $1/\beta, 1/(2(1 + \epsilon_0)\Delta(v)) > 1/q$  and the third term inside the minimum operator is at least  $q^{-1}(1 - 1/\beta)^\Delta$ , using upper bound  $\Phi(\mathbb{G}, \mathbf{L}, v, i, d - 1) \leq 1/\beta$  which we have from (3.9). □

### 3.3.3 Complexity

We begin by analyzing the complexity of computing function  $\Phi$ . Recall that  $n = \max(|V|, |E|, q)$  is the size of the instance.

**Proposition 4.** *For any given node  $v$ , the function  $\Phi$  can be computed in time  $2^{O(d(\log \|L\| + \log \Delta))}$ . In particular when  $d = O(\log n)$ , the overall computation is  $2^{O(\log^2 n)}$ . If in addition the size of the largest list  $\|L\|$  is constant then the computation time is polynomial in  $n$ .*

*Proof.* Let  $T(d)$  denote the complexity of computing function  $\Phi(\cdot, d)$ . Clearly,  $T(0) = O(\|L\|)$ . We now express  $T(d)$  in terms of  $T(d - 1)$ . Given a node  $v$ , in order to compute  $\Phi(\mathbb{G}, \mathbf{L}, v, i, d)$  we first identify the neighbors  $v_1, \dots, v_m$  of  $v$ . Then we create graph/list pairs  $\mathbb{G}_v, \mathbf{L}_{j,k}, 1 \leq k \leq m, j \in L(v)$ , compute  $\Phi(\cdot, d - 1)$  for each of this graphs, and use this to compute  $\Phi(\mathbb{G}, \mathbf{L}, v, i, d)$ . The overall computation effort is then

$$T(d) = O(\|L\|\Delta T(d - 1)).$$

Iterating over  $d$  we obtain  $T(d) = O(\|L\|^{d+1}\Delta^d) = O(2^{(d+1)(\log \|L\| + \log \Delta)}) = 2^{O(d(\log \|L\| + \Delta))}$ .

When  $d = O(\log n)$ , we obtain a bound  $2^{O(\log^2 n)}$ . If in addition  $\|L\| = O(1)$ , then the assumption (3.2) implies  $\Delta = O(1)$ , and then  $T(d) = n^{O(1)}$ .  $\square$

The following is then immediate.

**Corollary 4.** *Suppose  $d = O(\log n)$ . Then the complexity of the algorithm CountCOLOR is  $2^{O(\log^2 n)}$ . If in addition the size of the largest list  $\|L\|$  is constant, then CountCOLOR is a polynomial time algorithm.*

### 3.4 Correlation decay

The following is the key correlation decay result.

**Theorem 6.** *Consider a triangle-free graph/list pair  $(\mathbb{G}, \mathbf{L})$  satisfying (3.2), (3.3). There exist constants  $0 < \epsilon < 1$  which depend only on  $\alpha$ , such that for all nodes  $v$ , colors  $i \in L(v)$  and  $d \geq 0$*

$$\max_{i \in L(v)} \left| \log \mathbb{P}_{\mathbb{G}, \mathbf{L}}(c(v) = i) - \log \Phi(\mathbb{G}, \mathbf{L}, v, i, d) \right| \leq O(n^2(1 - \epsilon)^d). \quad (3.12)$$

This theorem is our key tool for using the values of  $\Phi$  for computing the marginals  $\mathbb{P}_{\mathbb{G}, \mathbf{L}}(c(v) = i)$ . We first establish that this correlation decay result implies our main result, Theorem 5.

*Proof of Theorem 5.* We consider an arbitrary instance  $(\mathbb{G}, \mathbf{L})$  with size  $n$  and arbitrary  $\delta > 0$ . We may assume without the loss of generality that  $n$  is at least a large constant bigger than  $C/\delta$ , for any universal constant  $C$ , since we can simply extend the size of the instance by adding isolated nodes. The proof uses a standard idea of approximating marginals  $\mathbb{P}_{\mathbb{G}, \mathbf{L}}(c(v) = i)$  and then using Proposition 2 for computing  $Z(\mathbb{G}, \mathbf{L})$ . From Proposition 2, if the algorithm CountCOLOR produces in every stage  $k = 1, 2, \dots, |V| - 1$  a value  $\hat{p}(v, i)$  which approximates  $\mathbb{P}_{\mathbb{G}_v, \mathbf{L}_k}(c(v_k) = i)$  with accuracy

$$1 - \frac{\delta}{n} \leq \frac{\hat{p}(v, i)}{\mathbb{P}_{\mathbb{G}_v, \mathbf{L}_k}(c(v_k) = i)} \leq 1 + \frac{\delta}{n} \quad (3.13)$$

then the output  $\hat{Z}$  of the algorithm satisfies

$$\left(1 - \frac{\delta}{n}\right)^n \leq \left(1 - \frac{\delta}{n}\right)^{|V|} \leq \frac{Z(\mathbb{G}, \mathbf{L})}{\hat{Z}} \leq \left(1 + \frac{\delta}{n}\right)^{|V|} \leq \left(1 + \frac{\delta}{n}\right)^n$$

Since  $|V| \leq n$  and  $n$  is at least a large constant, we obtain an arbitrary accuracy of the approximation. Thus it suffices to arrange for (3.13). We run the algorithm CountCOLOR with  $d = \lceil \frac{4 \log n}{\log \frac{1}{1-\epsilon}} \rceil$ , where  $\epsilon$  is the constant from Theorem 6. This choice of  $d$  gives  $(1 - \epsilon)^d \leq 1/n^4$ . Theorem 6 with the given value of  $d$  then implies

$$\left| \log \frac{\mathbb{P}_{\hat{\mathbb{G}}, \hat{\mathbf{L}}}(c(v) = i)}{\hat{p}(v, i)} \right| = \left| \log \frac{\mathbb{P}_{\hat{\mathbb{G}}, \hat{\mathbf{L}}}(c(v) = i)}{\Phi(\hat{\mathbb{G}}, \hat{\mathbf{L}}, v, i, d)} \right| \leq O(n^2) \frac{1}{n^4} = O\left(\frac{1}{n^2}\right).$$

Thus

$$1 - O\left(\frac{1}{n^2}\right) \leq \exp\left(-O\left(\frac{1}{n^2}\right)\right) \leq \frac{\hat{p}(v, i)}{\mathbb{P}_{\mathbb{G}, \mathbf{L}}(c(v) = i)} \leq \exp\left(O\left(\frac{1}{n^2}\right)\right) = 1 + O\left(\frac{1}{n^2}\right)$$

This gives us (3.13) for all  $n > C/\delta$  where  $C$  is the universal constant appearing in  $O(\cdot)$ . This completes the analysis of the accuracy. The complexity part of the theorem follows directly from Corollary 4.  $\square$

The rest of the section is devoted to establishing this Theorem 6. The basis of the proof is the recursion (3.4). As before, let  $v_1, \dots, v_m$  be the neighbors of  $v$  in  $\mathbb{G}$ ,  $m = \Delta(v)$ . Observe that (3.12) holds trivially when  $m = 0$ , since both expression inside the absolute value become  $1/|L(v)|$  and the left-hand side becomes equal to zero. Thus we assume that  $m \geq 1$ . Denote by  $m_k$  the degree of  $v_k$  in the graph  $\mathbb{G}_v$ . In order to ease the notations, we introduce

$$\begin{aligned} x_i &= \mathbb{P}_{\mathbb{G}, \mathbf{L}}(c(v) = i), \quad i \in L(v), \\ x_{i,k} &= \mathbb{P}_{\mathbb{G}_v, \mathbf{L}_{k,i}}(c(v_k) = i), \quad i \in L(v), 1 \leq k \leq m \\ x_i^* &= \Phi(\mathbb{G}, \mathbf{L}, v, i, d), \quad i \in L(v), \\ x_{i,k}^* &= \Phi(\mathbb{G}_v, \mathbf{L}_{k,i}, v_k, i, d-1), \quad i \in L(v) \cap L_{i,k}(v_k), 1 \leq k \leq m \end{aligned}$$

**Proposition 5.** *There exists a constant  $\epsilon > 0$  which depends on  $\alpha$  only such that*

$$\frac{1}{m} \max_{i \in L(v)} |\log(x_i) - \log(x_i^*)| \leq (1 - \epsilon) \max_{j \in L(v), k: m_k > 0} \frac{1}{m_k} |\log(x_{j,k}) - \log(x_{j,k}^*)| \quad (3.14)$$

First we show how this result implies Theorem 6:

*Proof of Theorem 6.* Applying this proposition  $d$  times and using the fact that we are summing over  $k : m_k > 0$ , we obtain

$$\frac{1}{m} \max_{i \in L(v)} [\log(x_i) - \log(x_i^*)] \leq M(1 - \epsilon)^d,$$

where

$$M = \max_{l,s} \left| \log \mathbb{P}_{\mathbb{G}_s, \mathbf{L}_s}(c(v) = l) - \log \Phi(\mathbb{G}_s, \mathbf{L}_s, v, l, 0) \right|$$

and the maximum is over all graph/list pairs  $\mathbb{G}_s, \mathbf{L}_s$  appearing during the computation of  $\Phi$  and over all colors  $l$ . Recall that if  $l$  does not belong to the list associated with node  $v$  and list vector  $\mathbf{L}_s$ , then  $\mathbb{P}_{\mathbb{G}_s, \mathbf{L}_s}(c(v) = l) = \Phi(\mathbb{G}_s, \mathbf{L}_s, v, l, 0) = 0$  (the first is equal to zero by definition, the second by the way we set the values of  $\Phi$ ). Otherwise we have from Lemma 12 and part (3.11) of Lemma 14 that absolute value of the difference is at most

$$\log q + \Delta \log(\beta/(\beta - 1)).$$

Since  $m \leq \Delta \leq n$ ,  $\beta$  is a constant which only depends on  $\alpha$ , and  $q \leq n$ , then we obtain  $M = O(n)$  and  $mM = O(n^2)$ .  $\square$

Thus we focus on establishing Proposition 5.

*Proof of Proposition 5.* Observe that for every  $i \in L(v) \setminus L_{k,j}(v_k)$  we have  $x_{i,k} = x_{i,k}^* = 0$ . This is because the probability of node  $v_k$  obtaining color  $i$  is zero when this color is not in its list. Similarly, the corresponding value of  $\Phi$  is zero, since we set it to zero for all colors not in the list. For every  $i \in L(v)$  introduce

$$A_i \triangleq \prod_{1 \leq k \leq m} (1 - x_{i,k}) \quad (3.15)$$

and

$$A \triangleq \sum_{j \in L(v)} A_j \quad (3.16)$$

Introduce  $A_i^*, A^*$  similarly. Applying Proposition 3 we obtain

$$x_i = \frac{A_i}{A}, \quad (3.17)$$

$$x_i^* = \min \left[ \frac{1}{2(1 + \epsilon_0)m}, \frac{1}{\beta}, \frac{A_i^*}{A^*} \right]. \quad (3.18)$$

Let

$$\tilde{x}_i^* = \frac{A_i^*}{A^*}.$$

We claim that in order to establish (3.14) it suffices to establish the bound

$$\frac{1}{m} |\log x_i - \log \tilde{x}_i^*| \leq (1 - \epsilon) \max_{j \in L(v), k: m_k > 0} \frac{1}{m_k} |\log(x_{j,k}) - \log(x_{j,k}^*)|$$

Indeed, if  $\tilde{x}_i^* \neq x_i^*$ , then  $x_i^* = \min[\frac{1}{2(1+\epsilon_0)m}, \frac{1}{\beta}]$ . On the other hand, by Lemmas 12,13 we have  $x_i \leq \min[\frac{1}{2(1+\epsilon_0)m}, \frac{1}{\beta}]$ , implying  $x_i \leq x_i^* \leq \tilde{x}_i^*$ , and the bound for  $\tilde{x}_i^*$  implies the bound (3.14).

We have

$$\max_{i \in L(v)} |\log(x_i) - \log(x_i^*)| = \max_{i \in L(v)} \left| \log A_i - \log A_i^* - \log A + \log A^* \right|. \quad (3.19)$$

We introduce auxiliary variables  $y_i = \log(x_i)$ ,  $y_{i,k} = \log(x_{i,k})$ . Similarly, let  $y_i^* = \log(\tilde{x}_i^*)$ ,  $y_{i,k}^* = \log(x_{i,k}^*)$ . Define  $\mathbf{y} = (y_{i,k})$ ,  $\mathbf{y}^* = (y_{i,k}^*)$ . Observe that if  $m_k = 0$  then for every color  $i$   $x_{i,k} = x_{i,k}^*$ . This follows since both values are  $1/|L_{i,k}|$  when  $i \in L_{i,k}$  and zero otherwise. This implies  $y_{i,k} = y_{i,k}^*$ . Then we rewrite (3.19) as

$$\begin{aligned} \max_{i \in L(v)} |y_i - y_i^*| &= \max_{i \in L(v)} \left| \sum_{k: m_k > 0} \log(1 - \exp(y_{i,k})) - \sum_{k: m_k > 0} \log(1 - \exp(y_{i,k}^*)) \right. \\ &\quad \left. - \log \left( \sum_{j \in L(v)} \prod_{1 \leq k \leq m} (1 - \exp(y_{j,k})) \right) + \log \left( \sum_{j \in L(v)} \prod_{1 \leq k \leq m} (1 - \exp(y_{j,k}^*)) \right) \right|, \end{aligned} \quad (3.20)$$



where the sums  $\sum_{1 \leq k \leq m}$  were replaced by  $\sum_{k:m_k > 0}$  due to our observation  $y_{i,k} = y_{i,k}^*$  when  $m_k = 0$ .

For every  $i$  denote the expression inside the absolute value in the right-hand side of equation (3.20) by  $\mathcal{G}_i(\mathbf{y})$ . That is we treat  $\mathbf{y}^*$  as constant and  $\mathbf{y}$  as a variable. It suffices to prove that for each  $i$

$$\mathcal{G}_i(\mathbf{y}) \leq (1 - \epsilon) \max_{j \in L(v), k: m_k > 0} \frac{1}{m_k} \left| \log(x_{j,k}) - \log(x_{j,k}^*) \right| \quad (3.21)$$

Observe that  $\mathcal{G}_i(\mathbf{y}^*) = 0$ . Let  $g_i(t) = \mathcal{G}_i(\mathbf{y}^* + t(\mathbf{y} - \mathbf{y}^*))$ ,  $t \in [0, 1]$ . Then  $g_i$  is a differentiable function interpolating between 0 and  $\mathcal{G}_i(\mathbf{y})$ . In particular,  $g_i(1) = \mathcal{G}_i(\mathbf{y})$ . Applying the Mean Value Theorem we obtain

$$\begin{aligned} |g_i(1) - g_i(0)| &= |g_i(1)| \leq \sup_{0 \leq t \leq 1} |\dot{g}_i(t)| \\ &= \sup_{0 \leq t \leq 1} \left| \nabla \mathcal{G}_i(\mathbf{y}^* + t(\mathbf{y} - \mathbf{y}^*))^T (\mathbf{y} - \mathbf{y}^*) \right| \end{aligned}$$

where the supremum is over values of  $t$ . We use a short-hand notation

$$\Pi_j = \prod_{1 \leq k \leq m} (1 - \exp(y_{j,k} + t(y_{j,k} - y_{j,k}^*)))$$

For each  $t$  we have

$$\begin{aligned} \nabla \mathcal{G}_i(\mathbf{y}^* + t(\mathbf{y} - \mathbf{y}^*)) (\mathbf{y} - \mathbf{y}^*) &= \sum_{k: m_k > 0} \frac{-\exp(y_{i,k} + t(y_{i,k} - y_{i,k}^*))}{1 - \exp(y_{i,k} + t(y_{i,k} - y_{i,k}^*))} (y_{i,k} - y_{i,k}^*) \\ &+ \frac{\sum_{j \in L(v)} \sum_{1 \leq k \leq m} \frac{\exp(y_{j,k} + t(y_{j,k} - y_{j,k}^*))}{1 - \exp(y_{j,k} + t(y_{j,k} - y_{j,k}^*))} (y_{j,k} - y_{j,k}^*) \Pi_j}{\sum_{j \in L(v)} \Pi_j}. \end{aligned}$$

Again using the fact  $y_{j,k} = y_{j,k}^*$  when  $m_k = 0$ , we can replace the sum  $\sum_{1 \leq k \leq m}$  by  $\sum_{k: m_k > 0}$

in the expression above. For each  $j$  we have from convexity of  $\exp$

$$\begin{aligned} \exp(y_{j,k} + t(y_{j,k} - y_{j,k}^*)) &\leq (1-t)\exp(y_{j,k}^*) + t\exp(y_{j,k}) \\ &= (1-t)x_{j,k} + tx_{j,k}^* \\ &\leq \frac{1}{2(1+\epsilon_0)m_k}. \end{aligned}$$

where the last inequality follows from Lemma 13 and part (3.9) of Lemma 14. This bound is useful for terms with  $m_k > 0$  (for this reason we only kept these terms in the sum  $\sum_{k:m_k>0}$ ). Similarly using Lemma 12 and again part (3.9) of Lemma 14 we obtain

$$\begin{aligned} \frac{1}{1 - \exp(y_{j,k} + t(y_{j,k} - y_{j,k}^*))} &\leq \frac{1}{1 - (1-t)\exp(y_{j,k}^*) - t\exp(y_{j,k})} \\ &= \frac{1}{1 - (1-t)x_{j,k} - tx_{j,k}^*} \\ &\leq \frac{1}{1 - \frac{1}{\beta}}. \end{aligned}$$

We obtain

$$\begin{aligned} \sup_{0 \leq t \leq 1} \left| \nabla \mathcal{G}_i(\mathbf{y}^* + t(\mathbf{y} - \mathbf{y}^*))(\mathbf{y} - \mathbf{y}^*) \right| &\leq \sum_{k:m_k>0} \frac{1}{(1 - \frac{1}{\beta})2(1 + \epsilon_0)m_k} |y_{i,k} - y_{i,k}^*| \\ &\quad + \frac{\sum_{j \in L(v)} \sum_{k:m_k>0} (1 - \frac{1}{\beta})^{-1} 2^{-1} (1 + \epsilon_0)^{-1} m_k^{-1} |y_{j,k} - y_{j,k}^*| \Pi_j}{\sum_{j \in L(v)} \Pi_j} \\ &\leq \frac{m}{(1 - \frac{1}{\beta})2(1 + \epsilon_0)} \max_{j \in L(v), k:m_k>0} \frac{|y_{j,k} - y_{j,k}^*|}{m_k} \\ &\quad + \frac{m}{(1 - \frac{1}{\beta})2(1 + \epsilon_0)} \max_{j \in L(v), k:m_k>0} \frac{|y_{j,k} - y_{j,k}^*|}{m_k} \\ &= \frac{m}{(1 - \frac{1}{\beta})(1 + \epsilon_0)} \max_{j \in L(v), k:m_k>0} \frac{|y_{j,k} - y_{j,k}^*|}{m_k}. \end{aligned}$$

Combining with (3.20) we conclude

$$\max_{i \in L(v)} \frac{|y_i - y_i^*|}{m} \leq \frac{1}{(1 - \frac{1}{\beta})(1 + \epsilon_0)} \max_{j \in L(v), k:m_k>0} \frac{|y_{j,k} - y_{j,k}^*|}{m_k}.$$

We now select a sufficiently large constant  $\beta = \beta(\epsilon_0)$  such that

$$1 - \epsilon \triangleq \frac{1}{(1 - \frac{1}{\beta})(1 + \epsilon_0)} < 1.$$

This completes the proof of Proposition 5. □

### 3.5 Comparison of the correlation decay on a computation tree and the spatial correlation decay property

As we have mentioned above, the (spatial) correlation decay is known to hold for the coloring problem in a stronger regime  $\alpha > \alpha^* \approx 1.763\dots$ , then the regime  $\alpha > \alpha^{**}$  considered in this paper [GMP05]. This decay of correlation is established in a conventional sense: for every node  $v$  the marginal probability  $\mathbb{P}(c(v) = i)$  is asymptotically independent from changing a color on a boundary of the depth- $d$  neighborhood  $B(v, d)$  of  $v$  in the underlying graph. In fact it is established that the decay of correlation is exponential in  $d$ . It is natural to try to use this result directly as a method for computing approximately the marginals  $\mathbb{P}(c(v) = i)$ , for example by computing the marginal  $\mathbb{P}_{B(v, d)}(c(v) = i)$  corresponding to the neighborhood  $B(v, d)$ , say using brute force computation. Unfortunately, this conventional correlation decay result is not useful because of the computation growth. In order to obtain  $\epsilon$ -approximation of the partition function, we need order  $O(\epsilon/n)$  approximation of the marginals, which means the depth  $d$  of the neighborhood  $B(v, d)$  needs to be at least  $O(\log n)$ . Here  $n$  is the number of nodes. But the resulting cardinality of  $B(v, d)$ , even for the case of constant degree graphs is  $O(\Delta^{\log n}) = n^{O(1)}$  - polynomial in  $n$  and the brute-force computation effort would be exponential in  $n$ . Notice that even if the underlying graph has a polynomial expansion  $|B(v, d)| \leq d^r$ , for some power  $r \geq 1$ , the brute-force computation would still be  $O(\exp(\log^r n))$  which is super-polynomial. This is where having correlation decay on computation tree as opposed to the conventional graph theoretic sense helps.

## 3.6 Conclusions

We have established the existence of a deterministic approximation algorithm for counting the number of list colorings for certain classes of graphs.

Along with [BG06] and [Wei06] this work is another step in the direction of developing a new powerful method for solving counting problems using insights from statistical physics. This method provides an important alternative to the existing MCMC sampling based method as it leads to a deterministic as opposed to a randomized algorithm.

The principle insight from this work, along with the work of Weitz [Wei06] is the advantage of establishing the correlation decay property on the *computation tree* as opposed to the original graph theoretic structure. While we have established such correlation decay only in the regime  $\alpha > 2.8432\dots$ , we conjecture that it holds for much lower values of  $\alpha$ . In fact, just as it is conjectured that the Markov chain is rapidly mixing in the regime  $q \geq \Delta + 2$ , we conjecture that the correlation decay on the computation tree holds in this regime as well, at least for the case of constant number of colors  $q$ .

# Chapter 4

## Markov random field and partition function

In this chapter we extend some of the results of the previous chapter to Markov random fields.

The main conceptual point of Chapter 3, namely construction a recursion of the form (3.4), construction of a corresponding computation tree, establishing correlation decay property and application to a counting problem, can be extended to an arbitrary model of random constraint satisfaction problems with multiple values. In this chapter we provide details using a very general framework of Markov random fields (MRF), also known as *graphical model*. We show that generalizing (3.4) is straightforward. It is establishing the decay of correlation which presents the main technical difficulty. We provide a simple and general sufficient condition and then illustrate the approach on specific statistical physics problem, namely  $q$ -state Potts model. Here we restrict ourselves for simplicity to MRF defined on simple graphs. Extensions to multi-graphs are possible as well.

### 4.1 Model and the preliminary results

A Markov random field (MRF) is given as a graph  $\mathbb{G}$  with node set  $V = \{v_1, \dots, v_{|V|}\}$ , edge set  $E$ , an alphabet  $\mathcal{X}$  and set of functions  $\phi_v : \mathcal{X} \rightarrow \mathbb{R}_+$ ,  $v \in V$ ,  $f_{v,u} : \mathcal{X} \rightarrow \mathbb{R}_+$ ,  $(v, u) \in E$ .

Consider a probability measure on  $\mathcal{X}^{|V|}$  defined by

$$\mathbb{P}(\mathbf{X} = \mathbf{x}) = \frac{\prod_{v \in V} \phi_v(x_v) \prod_{(v,u) \in E} f_{v,u}(x_v, x_u)}{Z},$$

for every  $\mathbf{x} = (x_v) \in \mathcal{X}^{|V|}$ , where  $Z = \sum_{\mathbf{x}} \prod_{v \in V} \phi_v(x_v) \prod_{(v,u) \in E} f_{v,u}(x_v, x_u)$  is the normalizing constant called the partition function. Here  $\mathbf{X} = (X_v)$  is the random vector selected according to this probability measure. In the case  $Z = 0$ , the MRF is not defined. From now on assume that

$$\prod_{v \in V} \phi_v(x_v) \prod_{(v,u) \in E} f_{v,u}(x_v, x_u) > 0 \text{ for at least one } \mathbf{x} = (x_v) \in \mathcal{X}^{|V|}.$$

Let us see that the problem of list-coloring can be cast as a Markov random field, where  $\mathbb{P}(\cdot)$  corresponds to the uniform probability distribution on the set of valid colorings. Given an instance of a list-coloring problem  $(\mathbb{G}, \mathbf{L})$  with a universe of colors  $\{1, \dots, q\}$ , we set  $\mathcal{X} = \{1, \dots, q\}$ ,  $\phi_v(i) = 1\{i \in L(v)\}$  for all node/color pairs  $v, i$ , and  $f_{v,u}(i, j) = 1\{i \neq j\}$ , where  $1\{\cdot\}$  is the indicator function. It is not hard to see that  $\mathbb{P}(\mathbf{x}) = 1/Z$  if  $\mathbf{x}$  corresponds to a valid coloring and  $= 0$  otherwise, and  $Z = Z(\mathbb{G}, \mathbf{L})$  is the total number of valid list-colorings. Thus this MRF corresponds to the uniform distribution on the set of proper colorings.

An instance of a MRF is denoted by  $\mathbb{M} = (\mathbb{G}, \mathcal{X}, \phi, f)$ , with  $\phi = (\phi_v)$ ,  $f = (f_{v,u})$ . We will write  $\mathbb{P}_{\mathbb{M}}$  and  $Z_{\mathbb{M}}$  for the corresponding probability measure and the partition function, respectively, in order to emphasize the dependence on the particular instance of the MRF. Computation of  $Z_{\mathbb{M}}$  is the principle goal of this chapter. As in the case of list-coloring model, denote by  $Z_{\mathbb{M}}[\chi]$  the sum of the terms in the partition function which satisfy some condition  $\chi$ .

Observe that if  $\phi_v, f_{v,u} > 0$  for all nodes and edges then  $\mathbb{P}_{\mathbb{M}}(\mathbf{X} = \mathbf{x}) > 0$  for every  $\mathbf{x} = (x_v), v \in V$ . Moreover, if  $f_{v,u} = c$  for all edges for some constant  $c$ , then we obtain a product form solution

$$\mathbb{P}_{\mathbb{M}}(\mathbf{X} = \mathbf{x}) = \prod_v \frac{\phi_v(x_v)}{\sum_{y \in \mathcal{X}} \phi_v(y)}.$$

Thus we might expect the correlation decay to take place when the values of  $f_{v,u}$  are *close* to each other. This is the regime within which we will establish our results. Let  $\phi_{\min} = \min_{v,x} \phi_v(x)$ ,  $\phi_{\max} = \max_{v,x} \phi_v(x)$  and  $c_{\phi} = \phi_{\max}/\phi_{\min}$ . Also let  $f_{\min} = \min_{(v,u) \in E, x, y \in \mathcal{X}} f_{v,u}(x, y)$ ,  $f_{\max} =$

$\max_{(v,u) \in E, x, y \in \mathcal{X}} f_{v,u}(x, y)$  and let  $c_f = f_{\max}/f_{\min}$ . From now on we assume that the following conditions hold

$$\phi_v(x) > 0, \quad \forall v \in V, x \in \mathcal{X} \quad (4.1)$$

$$f_{v,u}(x, y) > 0, \quad \forall (v, u) \in E, x, y \in \mathcal{X} \quad (4.2)$$

These conditions in particular ensure that  $c_f < \infty$ . The following condition will be used in lieu of (3.2)

$$\gamma \triangleq (c_f^\Delta - c_f^{-\Delta})\Delta|\mathcal{X}|^\Delta < 1. \quad (4.3)$$

The size of an instance  $\mathbb{M}$  is

$$n = \max \left( |V|, |E|, |\mathcal{X}|, |\log \phi_{\max}|, |\log \phi_{\min}^{-1}|, |\log f_{\max}|, |\log f_{\min}^{-1}| \right).$$

We now state the main result of this chapter.

**Theorem 7.** *There exist a deterministic algorithm which provides an FPTAS for computing  $Z_{\mathbb{M}}$  for an arbitrary MRF instance  $\mathbb{M}$  satisfying (4.1), (4.2), (4.3), whenever  $|\mathcal{X}|$  and  $\Delta$  are constants.*

Our first task is obtaining a generalization of the cavity recursion given by Proposition 2. Given a MRF  $\mathbb{M} = (\mathbb{G}, \mathcal{X}, \phi, f)$ , an arbitrary node  $v$  and an arbitrary element  $x^* \in \mathcal{X}$  we consider a new MRF instance  $\mathcal{T}_{v, x^*}[\mathbb{M}] = (\tilde{\mathbb{G}}, \tilde{\mathcal{X}}, \tilde{\phi}, \tilde{f})$  defined as follows. The graph  $\tilde{\mathbb{G}}$  is the subgraph of  $\mathbb{G}$  induced by all nodes other than  $v$ .  $\tilde{\mathcal{X}} = \mathcal{X}$  and  $\tilde{f} = f$ .  $\tilde{\phi}$  is defined as follows. For every  $u$  which is a neighbor of  $v$ ,  $\tilde{\phi}_u(x) = \phi_v^{\frac{1}{\Delta(v)}} f_{v,u}(x^*, x)$ , where  $\Delta(v)$  is the degree of  $v$  in  $\mathbb{G}$ . For all the remaining nodes  $u$  we set  $\tilde{\phi}_u = \phi_u$ .

Given a MRF  $\mathbb{M} = (\mathbb{G}, \mathcal{X}, \phi, f)$  let  $v_1, \dots, v_{|V|}$  be an arbitrary enumeration of nodes. Consider an arbitrary  $\mathbf{x}^* = (x_{v_1}^*, \dots, x_{v_{|V|}}^*)$  such that  $\mathbb{P}(\mathbf{x}^*) > 0$ . Define  $\mathbb{M}_0 = \mathbb{M}$  and  $\mathbb{M}_k = \mathcal{T}_{v_k, x_{v_k}^*}[\mathbb{M}_{k-1}]$ ,  $k = 1, 2, \dots, |V|$ , where  $\mathbb{M}_{|V|}$  is an empty MRF and its partition function is set by default to unity.

**Proposition 6.** *The following identity holds.*

$$Z_{\mathbb{M}} = \prod_{1 \leq k \leq |V|} \mathbb{P}_{\mathbb{M}_{k-1}}^{-1}(X_{v_k} = x_{v_k}^*).$$

*Proof.* Let  $\mathcal{A}(v_1) = \prod_{(v,u) \in E, v,u \neq v_1} f_{v,u}(x_v, x_u)$

We have

$$\begin{aligned} \mathbb{P}_{\mathbb{M}}(X_{v_1} = x_{v_1}^*) &= \frac{\sum_{\mathbf{x} \in \mathcal{X}^{|V|}: x_{v_1} = x_{v_1}^*} \mathcal{A}(v_1) \phi_{v_1}(x_{v_1}^*) \prod_{u: (v_1, u) \in E} f_{v_1, u}(x_{v_1}^*, x_u) \prod_{u \neq v_1} \phi(x_u)}{Z_{\mathbb{M}}} \\ &= \frac{\mathcal{A}(v_1) \prod_{u \neq v_1} \phi_u^1(x_u)}{Z_{\mathbb{M}}} \\ &= \frac{Z_{\mathbb{M}_1}}{Z_{\mathbb{M}}}, \end{aligned}$$

where the second equality follows since  $\phi(x_{v_1}^*) \prod_{u: (v_1, u) \in E} f(x_{v_1}^*, x_u) = \prod_u \phi^1(x_u)$  and the second product is over neighbors  $u$  of  $v_1$  in  $\mathbb{G}$ . Iterating further for  $k \geq 2$  we obtain the result.  $\square$

The identity in Proposition 6 provides an important representation of the partition function in terms of marginal probabilities. Thus, if we compute (approximately) these marginal probabilities, we can use them to obtain the value of the underlying partition function.

### 4.1.1 Basic recursion and the algorithm

Our next task is constructing a generalization of  $(\mathbb{G}_v, \mathbf{L}_{k,i})$  and extending Proposition 3 to MRF. Given a MRF  $\mathbb{M} = (\mathbb{G}, \mathcal{X}, \phi, f)$  and a node  $v$  let  $\mathbb{M}_v$  denote the MRF instance obtained naturally by removing node  $v$ . Namely, we keep  $\phi_u$  and  $f_{u,w}$  intact for all the nodes  $u \neq v$  and edges  $(u,w), u,w \neq v$ . Also, given a MRF  $\mathbb{M} = (\mathbb{G}, \mathcal{X}, \phi, f)$ , a set of nodes  $v_1, \dots, v_r \subset V$  and a set of elements  $x_1, \dots, x_r \in \mathcal{X}$  we construct a MRF denoted by  $\mathbb{M}[v_1, x_1; \dots; v_r, x_r] = (\tilde{\mathbb{G}}, \tilde{\phi}, \tilde{f})$  as follows. The corresponding graph  $\tilde{\mathbb{G}}$  is the subgraph induced by nodes  $V \setminus \{v_1, \dots, v_r\}$ . For every node  $u \in \tilde{\mathbb{G}}$  which has at least one neighbor among  $v_1, \dots, v_r$  we set  $\tilde{\phi}_u(x) = \prod_i f_{v_i, u}(x_i, x) \phi_u(x)$ , where the product is over  $i = 1, 2, \dots, r$  such that  $(v_i, u)$  is an edge in  $\mathbb{G}$ . For all the remaining  $u$  we set  $\tilde{\phi}_u = \phi_u$ . We also set  $\tilde{f} = f$ . The interpretation for  $\mathbb{M}[v_1, x_1; \dots; v_r, x_r]$  comes from the following simple fact.



**Lemma 15.** For every event  $\mathcal{E}$  corresponding to the probability measure  $\mathbb{P}_{\mathbb{M}}$ , the following holds

$$\mathbb{P}_{\mathbb{M}}(\mathcal{E} \mid \bigwedge_{k \leq r} X_{v_k} = x_k) = \mathbb{P}_{\mathbb{M}[v_1, x_1; \dots; v_r, x_r]}(\mathcal{E}).$$

*Proof.* The proof is obtained immediately by summing over all of the elementary events  $\mathbf{x} \in \mathcal{E}$  and observing that the terms  $\phi_{v_k}(x_k)$  cancel in the ratio  $\mathbb{P}_{\mathbb{M}}(\mathcal{E} \mid \bigwedge_{k \leq r} X_{v_k} = x_k) / \mathbb{P}_{\mathbb{M}}(\bigwedge_{k \leq r} X_{v_k} = x_k)$ .  $\square$

Observe that the value of  $c_f$  corresponding to the MRF  $\mathbb{M}[v_1, x_1; \dots; v_r, x_r]$  is the same of  $\mathbb{M}$ . Thus, should  $\mathbb{M}$  satisfy conditions (4.1), (4.2), (4.3), so does the instance  $\mathbb{M}[v_1, x_1; \dots; v_r, x_r]$ . Moreover,  $c_{\bar{\phi}}$  defined for this MRF satisfies

$$c_{\bar{\phi}} \leq c_{\phi} c_f^{\Delta} \tag{4.4}$$

Now we obtain a recursion which serves as a basis for our correlation decay analysis and construction of an algorithm.

**Proposition 7.** For every node  $v$  and its neighbors  $v_1, \dots, v_m$ , the following identity holds for every  $x_0 \in \mathcal{X}$ :

$$\mathbb{P}_{\mathbb{M}}(X_v = x_0) = \frac{\phi_v(x_0) \sum_{x_1, \dots, x_m \in \mathcal{X}} \prod_{k=1}^m f_{v, v_k}(x_0, x_k) \mathbb{P}_{\mathbb{M}[v_1, x_1; \dots; v_{k-1}, x_{k-1}]}(X_{v_k} = x_k)}{\sum_{x \in \mathcal{X}} \phi_v(x) \sum_{x_1, \dots, x_m \in \mathcal{X}} \prod_{k=1}^m f_{v, v_k}(x, x_k) \mathbb{P}_{\mathbb{M}[v_1, x_1; \dots; v_{k-1}, x_{k-1}]}(X_{v_k} = x_k)}, \tag{4.5}$$

where the sum  $\sum_{x_1, \dots, x_m \in \mathcal{X}} = 1$  when  $m = 0$ .

*Proof.* The case  $m = 0$  is immediate. Assume  $m > 0$ . For every  $x_0 \in \mathcal{X}$  we have the following identity

$$\mathbb{P}_{\mathbb{M}}(X_v = x_0) = \frac{\phi_v(x_0) \sum_{x_1, \dots, x_m \in \mathcal{X}} Z_{\mathbb{M}_v}[X_{v_1} = x_1, \dots, X_{v_m} = x_m] \prod_{k=1}^m f_{v, v_k}(x_0, x_k)}{\sum_{x \in \mathcal{X}} \phi_v(x) \sum_{x_1, \dots, x_m \in \mathcal{X}} Z_{\mathbb{M}_v}[X_{v_1} = x_1, \dots, X_{v_m} = x_m] \prod_{k=1}^m f_{v, v_k}(x, x_k)}$$

We divide both parts by  $Z_{\mathbb{M}_v}$  and write

$$\frac{Z_{\mathbb{M}_v}[X_{v_1} = x_1, \dots, X_{v_m} = x_m]}{Z_{\mathbb{M}_v}} = \prod_{k=1}^m \frac{Z_{\mathbb{M}_v}[X_{v_1} = x_1, \dots, X_{v_k} = x_k]}{Z_{\mathbb{M}_v}[X_{v_1} = x_1, \dots, X_{v_{k-1}} = x_{k-1}]},$$

where the term corresponding to  $k = 0$  is identified with  $Z_{M_v}$ . Applying Lemma 15, we recognize the  $k$ -th term in this product as  $\mathbb{P}_{M[v_1, x_1; \dots; v_{k-1}, x_{k-1}]}(X_{v_k} = x_k)$  (note that the terms  $\phi_{v_j}(x_j), j \leq k - 1$  cancel out).

□

Proposition 7 also allows us to obtain upper and lower bounds on the marginal probabilities:

**Lemma 16.** *For every node  $v$  and  $x_0 \in \mathcal{X}$*

$$c_f^{-\Delta} \frac{\phi_v(x_0)}{\sum_x \phi_v(x)} \leq \mathbb{P}_M(X_v = x_0) \leq c_f^{\Delta} \frac{\phi_v(x_0)}{\sum_x \phi_v(x)}.$$

*Proof.* The proof follows from Proposition 7. We have for every  $x \in \mathcal{X}$ , node  $v$  and its neighbors  $v_1, \dots, v_m$  that  $f_{\min}^m \leq \prod_{k=1}^m f_{v, v_k}(x, x_k) \leq c_f^m f_{\min}^m$ . Applying this bound to the numerator of (4.5) for  $x = x_0$  we obtain the required upper bound. Applying the same to the denominator, we obtain the required lower bound. □

We now provide sufficient conditions under which the construction of a computation tree for computing approximately marginal probabilities  $\mathbb{P}_M(X_v = x)$  as well as the partition function  $Z_M$  can be performed in polynomial time.

Similarly to the problem of coloring, we introduce  $\Phi(\cdot)$  – a surrogate for computing the marginal probabilities  $\mathbb{P}_M(\cdot)$ . Consider a function  $\Phi_M(v, x, d)$  defined recursively for an arbitrary instance of a MRF  $M = (\mathbb{G}, \mathcal{X}, \phi, f)$ , arbitrary node  $v$ , element  $x \in \mathcal{X}$  and a non-negative integer  $d$  as follows.

- We set  $\Phi_M(v, x, 0) = 1$ . As in the case of coloring, it turns out that the initialization values are not particularly important, due to the decay of correlations.
- For every node  $v$  with neighbors  $v_1, \dots, v_m$ , every  $x_0 \in \mathcal{X}$  and  $d \geq 1$

$$\Phi_M(v, x_0, d) = \frac{\phi_v(x_0) \sum_{x_1, \dots, x_m \in \mathcal{X}} \prod_{k=1}^m \Phi_{M[v_1, x_1; \dots; v_{k-1}, x_{k-1}]}(v_k, x_k, d-1) f_{v, v_k}(x_0, x_k)}{\sum_{x \in \mathcal{X}} \phi_v(x) \sum_{x_1, \dots, x_m \in \mathcal{X}} \prod_{k=1}^m \Phi_{M[v_1, x_1; \dots; v_{k-1}, x_{k-1}]}(v_k, x_k, d-1) f_{v, v_k}(x, x_k)}, \quad (4.6)$$

where the sum  $\sum_{x_1, \dots, x_m \in \mathcal{X}} = 1$  when  $m = 0$ .

Assumptions (4.1) and (4.2) guarantee that  $\Phi > 0$ .

We now describe our algorithm for approximately computing  $Z_{\mathbb{M}}$ . The algorithm is parametrized by  $d$ . It is based on computing recursively the values of  $\Phi_{\mathbb{M}}$ .

### Algorithm ComputeZ

INPUT: A MRF instance  $\mathbb{M} = (\mathbb{G}, \mathcal{X}, \phi, f)$  and a positive integer  $d$ .

BEGIN

Set  $\hat{Z} = 1, \hat{\mathbb{M}} = \mathbb{M}$ .

While  $\hat{\mathbb{G}} \neq \emptyset$ , fix an arbitrary node  $v \in \hat{\mathbb{G}}$  and element  $x \in \mathcal{X}$ . Compute  $\Phi_{\hat{\mathbb{M}}}(v, x, d)$ .

Set  $\hat{Z} = \Phi_{\hat{\mathbb{M}}}^{-1}(v, x, d)\hat{Z}$ .

Set  $\hat{\mathbb{M}} = \mathcal{T}_{v,x}[\hat{\mathbb{M}}]$ , where the operator  $\mathcal{T}$  was defined before Proposition 6.

END

OUTPUT:  $\hat{Z}$ .

## 4.1.2 Complexity

We begin by analyzing the complexity of computing function  $\Phi$ .

**Proposition 8.** *For every  $v \in V, x \in \mathcal{X}$ , the function  $\Phi_{\mathbb{M}}(v, x, d)$  can be computed in time  $O(2^{d\Delta \log |\mathcal{X}|} n^2)$ . In particular when  $d = O(\log n)$ , and  $|\mathcal{X}|, \Delta = O(1)$ , the computation is polynomial in  $n$ .*

We note that the dependence on  $\Delta$  is not as nice as in the case of the list-coloring problem, as it appears as  $\Delta$  not  $\log \Delta$  in the exponent. Thus we can no longer claim that the computation time is  $2^{O(\log^2 n)}$  in this case.

*Proof.* Let  $T(d)$  denote the complexity of computing function  $\Phi(\cdot, d)$ . Clearly,  $T(0) = O(1)$ . We now express  $T(d)$  in terms of  $T(d-1)$ . Given a node  $v$ , in order to compute  $\Phi_{\mathbb{M}}(v, x, d)$  we identify the neighbors  $v_1, \dots, v_m$  of  $v$ . For every sequence  $x_1, \dots, x_m \in \mathcal{X}$  and every  $k = 1, 2, \dots, m$  we compute  $\Phi_{\mathbb{M}[v_1, x_1; \dots; v_{k-1}, x_{k-1}]}(v_k, x_k, d-1)$ . The computation of each such

quantity is  $T(d - 1)$ . We use the obtained values to compute  $\Phi_{\mathbf{M}}(v, x, d)$  via (4.6). We also need  $O(n^2)$  time to "take care" of multiplying by  $f_{v,v_k}$  and by  $\phi_v$ . The overall computation effort then satisfies

$$T(d) = O(|\mathcal{X}|^\Delta T(d - 1) + n^2).$$

Iterating over  $d$  we obtain  $T(d) = O(|\mathcal{X}|^{d\Delta} n^2) = O(2^{d\Delta \log |\mathcal{X}|} n^2)$ , and the first part is established. When  $d = O(\log n)$  and  $\Delta, |\mathcal{X}|$  are constants, we obtain  $T(d) = n^{O(1)}$ .  $\square$

The following is then immediate.

**Corollary 5.** *Suppose  $d = O(\log n)$  and  $|\mathcal{X}|, \Delta = O(1)$ . Then ComputeZ is a polynomial time algorithm.*

### 4.1.3 Correlation decay analysis

We now establish a correlation decay result which is a key to proving our main result, Theorem 7.

**Theorem 8.** *Given an arbitrary MRF satisfying conditions (4.1), (4.2), (4.3), the following holds for every node  $v$  and  $d \geq 1$*

$$\begin{aligned} & \max_{x \in \mathcal{X}} \left| \log \mathbb{P}_{\mathbf{M}}(X_v = x) - \log \Phi_{\mathbf{M}}(v, x, d) \right| \\ & \leq (1 - \gamma) \max_{1 \leq k \leq m, y \in \mathcal{X}} \left| \mathbb{P}_{\mathbf{M}[v_1, x_1, \dots, v_{k-1}, x_{k-1}]}(X_{v_k} = y) - \Phi_{\mathbf{M}[v_1, x_1, \dots, v_{k-1}, x_{k-1}]}(v_k, y, d - 1) \right| \end{aligned} \quad (4.7)$$

where  $v_1, \dots, v_m$  are the neighbors of  $v$ .

We first show how this theorem implies our main algorithmic result.

*Proof of Theorem 7.* We claim that ComputeZ provides FPTAS for computing partition function  $Z_{\mathbf{M}}$  when  $d = O(\log n)$  under the setting of Theorem 7. We have already established in Corollary 5 that the algorithm is polynomial time.

Consider any MRF instance  $\tilde{\mathbf{M}}$  obtained during the computation of  $\Phi_{\mathbf{M}}(\cdot)$  as a part of performing algorithm ComputeZ. Applying (4.4) and Lemma 16 we obtain that for every

node  $v$  in  $\tilde{M}$  and every  $x \in \mathcal{X}$

$$\mathbb{P}_{\tilde{M}}(X_v = x) \geq c_f^{-\Delta} \frac{1}{|\mathcal{X}|} c_\phi^{-1} \geq c_f^{-\Delta} \frac{1}{|\mathcal{X}|} c_f^{-\Delta d} c_\phi^{-1} = c_f^{-\Delta(d+1)} \frac{1}{|\mathcal{X}|} c_\phi^{-1}.$$

Then applying the result of Theorem 8  $d$  times and recalling  $\Phi_{\tilde{M}}(v, x, 0) = 1$ , we obtain for  $d = O(\log n)$

$$\begin{aligned} \left| \log \mathbb{P}_{\tilde{M}}(X_v = x) - \log \Phi_{\tilde{M}}(v, x, d) \right| &\leq (1 - \gamma)^d \left( \Delta(d+1) \log c_f + \log |\mathcal{X}| + \log c_\phi \right) \\ &= (1 - \gamma)^d O(dn \log n) \\ &= \frac{1}{n^{O(\log \frac{1}{1-\gamma})}} O(n \log^2 n) \\ &= \frac{1}{n^{O(1)}} \end{aligned}$$

where the last step is obtained by selecting  $d = C \log n$  for sufficiently large constant  $C$ .

$$\left| \frac{\mathbb{P}_{\tilde{M}}(X_v = x)}{\Phi_{\tilde{M}}(v, x, d)} - 1 \right| \leq \exp(n^{-\Omega(1)}) - 1 = \frac{1}{n^{\Omega(1)}}.$$

We conclude that  $\Phi_{\tilde{M}}(v, x, d)$  provides an approximation of marginal probability  $\mathbb{P}_{\tilde{M}}(X_v = x)$  with an inverse polynomial error. The remainder of the proof is the same as for Theorem 3.1.  $\square$

*Proof of Theorem 8.* Fix a node  $v$  and an element  $x_0 \in \mathcal{X}$ . Let  $v_1, \dots, v_m$  be neighbors of  $v$ . When  $m = 0$  we the left-hand side of (4.7) is zero. Thus assume  $m > 0$ . In order to ease the exposition we introduce some notations. Set  $z = \log \mathbb{P}_{\tilde{M}}(X_v = x_0)$ ,  $z_{x_1, \dots, x_k} = \log \mathbb{P}_{\tilde{M}[v_1, x_1; \dots; v_{k-1}, x_{k-1}]}(X_{v_k} = x_k)$ . Similarly  $\tilde{z} = \log \Phi_{\tilde{M}}(v, x_0, d)$ ,  $\tilde{z}_{x_1, \dots, x_k} = \log \Phi_{\tilde{M}[v_1, x_1; \dots; v_{k-1}, x_{k-1}]}(v_k, x_k, d - 1)$ . Also let  $\mathbf{z}$  denote the vector  $(z_{x_1, \dots, x_k})$ ,  $1 \leq k \leq m$ ,  $x_1, \dots, x_m \in \mathcal{X}$  and  $\tilde{\mathbf{z}}$  denote the vector  $(\tilde{z}_{x_1, \dots, x_k})$ ,  $1 \leq k \leq m$ ,  $x_1, \dots, x_m \in \mathcal{X}$ . Both vectors have dimension  $\sum_{1 \leq k \leq m} |\mathcal{X}|^m$ . Then we can rewrite (4.5) as

$$z = \log \frac{\phi_v(x_0) \sum_{x_1, \dots, x_m \in \mathcal{X}} \prod_{k=1}^m f_{v, v_k}(x_0, x_k) \exp(z_{x_1, \dots, x_k})}{\sum_{x \in \mathcal{X}} \phi_v(x) \sum_{x_1, \dots, x_m \in \mathcal{X}} \prod_{k=1}^m f_{v, v_k}(x, x_k) \exp(z_{x_1, \dots, x_k})}, \quad (4.8)$$

and rewrite (4.6) as

$$\tilde{z} = \log \frac{\phi_v(x_0) \sum_{x_1, \dots, x_m \in \mathcal{X}} \prod_{k=1}^m f_{v, v_k}(x_0, x_k) \exp(\tilde{z}_{x_1, \dots, x_k})}{\sum_{x \in \mathcal{X}} \phi_v(x) \sum_{x_1, \dots, x_m \in \mathcal{X}} \prod_{k=1}^m f_{v, v_k}(x, x_k) \exp(\tilde{z}_{x_1, \dots, x_k})}, \quad (4.9)$$

Introduce a function  $\mathcal{G}$  defined on a vector  $\mathbf{w} = (w_{x_1, \dots, x_k}), 1 \leq k \leq m, x_1, \dots, x_m \in \mathcal{X}$  with the same dimension  $\sum_{1 \leq k \leq m} |\mathcal{X}|^m$  as follows:

$$\mathcal{G}(\mathbf{w}) = \log \frac{\phi_v(x_0) \sum_{x_1, \dots, x_m \in \mathcal{X}} \prod_{k=1}^m f_{v, v_k}(x_0, x_k) \exp(w_{x_1, \dots, x_k})}{\sum_{x \in \mathcal{X}} \phi_v(x) \sum_{x_1, \dots, x_m \in \mathcal{X}} \prod_{k=1}^m f_{v, v_k}(x, x_k) \exp(w_{x_1, \dots, x_k})}, \quad (4.10)$$

which we rewrite as

$$\log \phi_v(x_0) + \log \mathcal{G}_1(\mathbf{w}) - \log \mathcal{G}_2(\mathbf{w})$$

where the definition of  $\mathcal{G}_1$  and  $\mathcal{G}_2$  is immediate.

$$\begin{aligned} \mathcal{G}_1(\mathbf{w}) &= \sum_{x_1, \dots, x_m \in \mathcal{X}} \prod_{k=1}^m f_{v, v_k}(x_0, x_k) \exp(w_{x_1, \dots, x_k}) \\ \mathcal{G}_2(\mathbf{w}) &= \sum_{x \in \mathcal{X}} \left( \phi_v(x) \sum_{x_1, \dots, x_m \in \mathcal{X}} \prod_{k=1}^m f_{v, v_k}(x, x_k) \exp(w_{x_1, \dots, x_k}) \right) \end{aligned}$$

For convenience, let

$$D(v, x_0, \mathbf{w}) = \sum_{x_1, \dots, x_m \in \mathcal{X}} \prod_{k=1}^m f_{v, v_k}(x_0, x_k) \exp(w_{x_1, \dots, x_k})$$

then

$$\begin{aligned} \mathcal{G}_1(\mathbf{w}) &= D(v, x_0, \mathbf{w}) \\ \mathcal{G}_2(\mathbf{w}) &= \sum_{x \in \mathcal{X}} \left( \phi_v(x) D(v, x_0, \mathbf{w}) \right) \end{aligned}$$

We have  $z - \tilde{z} = \mathcal{G}(z) - \mathcal{G}(\tilde{z})$ . Thus establishing (4.7) reduces to showing

$$|\mathcal{G}(z) - \mathcal{G}(\tilde{z})| \leq (1 - \gamma)\|z - \tilde{z}\|_{L_\infty}.$$

Applying Mean Value Theorem, there exists  $t \in [0, 1]$  such that

$$z - \tilde{z} = \nabla \mathcal{G}(tz + (1 - t)\tilde{z})^T(z - \tilde{z})$$

further implying

$$\|z - \tilde{z}\| \leq \|\nabla \mathcal{G}(tz + (1 - t)\tilde{z})\|_{L_1}\|z - \tilde{z}\|_{L_\infty}.$$

It then suffices to establish

$$\|\nabla \mathcal{G}(tz + (1 - t)\tilde{z})\|_{L_1} \leq 1 - \gamma.$$

In the following lemma we show that this bound holds for an arbitrary input vector  $\mathbf{w}$  and thus complete the proof of Theorem 8.  $\square$

**Lemma 17.** *For every vector  $\mathbf{w}$*

$$\|\nabla \mathcal{G}(\mathbf{w})\|_{L_1} \leq 1 - \gamma.$$

*Proof.* Fix an arbitrary sequence  $x_1^0, \dots, x_{k_0}^0 \in \mathcal{X}$  and the corresponding variable  $w_{x_1^0, \dots, x_{k_0}^0}$ .

We have

$$\frac{\partial \mathcal{G}}{\partial w_{x_1^0, \dots, x_{k_0}^0}} = \mathcal{G}_1^{-1} \frac{\partial \mathcal{G}_1}{\partial w_{x_1^0, \dots, x_{k_0}^0}} - \mathcal{G}_2^{-1} \frac{\partial \mathcal{G}_2}{\partial w_{x_1^0, \dots, x_{k_0}^0}}$$

For convenience, let  $B = \prod_{k=1}^{k_0} f_{v, v_k}(x_0, x_k^0) \exp(w_{x_1^0, \dots, x_k^0})$

We have

$$\begin{aligned} \mathcal{G}_1^{-1} \frac{\partial \mathcal{G}_1}{\partial w_{x_1^0, \dots, x_{k_0}^0}} &= \\ &= \frac{B \sum_{x_{k_0+1}, \dots, x_m \in \mathcal{X}} \prod_{k=k_0+1}^m f_{v, v_k}(x_0, x_k) \exp(w_{x_1^0, \dots, x_{k_0}, x_{k_0+1}, \dots, x_k})}{\sum_{x_1, \dots, x_m \in \mathcal{X}} \prod_{k=1}^m f_{v, v_k}(x_0, x_k) \exp(w_{x_1, \dots, x_k})} \end{aligned}$$

Using  $f_{\min} \leq f_{v,v_k}(x_0, x_k^0) \leq c_f f_{\min}$ , we obtain

$$\begin{aligned} \mathcal{G}_1^{-1} \frac{\partial \mathcal{G}_1}{\partial w_{x_1^0, \dots, x_{k_0}^0}} &\leq c_f^m \frac{\left( \prod_{k=1}^{k_0} \exp(w_{x_1^0, \dots, x_k^0}) \right) \sum_{x_{k+1}, \dots, x_m \in \mathcal{X}} \prod_{k=k_0+1}^m \exp(w_{x_1, \dots, x_k})}{\sum_{x_1, \dots, x_m \in \mathcal{X}} \prod_{k=1}^m \exp(w_{x_1, \dots, x_k})} \\ &\leq c_f^m \\ &\leq c_f^\Delta. \end{aligned}$$

Similarly, we obtain

$$\mathcal{G}_1^{-1} \frac{\partial \mathcal{G}_1}{\partial w_{x_1, \dots, x_k}} \geq c_f^{-\Delta}.$$

Using again  $f_{\min} \leq f_{v,u}(x, y) \leq c_f f_{\min}$  we also obtain

$$\begin{aligned} \mathcal{G}_2^{-1} \frac{\partial \mathcal{G}_2}{\partial w_{x_1^0, \dots, x_{k_0}^0}} &= \\ &= \frac{\sum_{x \in \mathcal{X}} \phi_v(x) B \sum_{x_{k+1}, \dots, x_m \in \mathcal{X}} \prod_{k=k_0+1}^m f_{v,v_k}(x_0, x_k) \exp(w_{x_1^0, \dots, x_{k_0}^0, x_{k_0+1}, \dots, x_k})}{\sum_{x \in \mathcal{X}} \phi_v(x) \sum_{x_1, \dots, x_m \in \mathcal{X}} \prod_{k=1}^m f_{v,v_k}(x, x_k) \exp(w_{x_1, \dots, x_k})} \\ &\leq c_f^m \frac{\left( \sum_{x \in \mathcal{X}} \phi_v(x) \right) \left( \prod_{k=1}^{k_0} \exp(w_{x_1^0, \dots, x_k^0}) \right) \sum_{x_{k+1}, \dots, x_m \in \mathcal{X}} \prod_{k=k_0+1}^m \exp(w_{x_1, \dots, x_k})}{\left( \sum_{x \in \mathcal{X}} \phi_v(x) \right) \sum_{x_1, \dots, x_m \in \mathcal{X}} \prod_{k=1}^m \exp(w_{x_1, \dots, x_k})} \\ &= c_f^m \frac{\left( \prod_{k=1}^{k_0} \exp(w_{x_1^0, \dots, x_k^0}) \right) \sum_{x_{k+1}, \dots, x_m \in \mathcal{X}} \prod_{k=k_0+1}^m \exp(w_{x_1, \dots, x_k})}{\sum_{x_1, \dots, x_m \in \mathcal{X}} \prod_{k=1}^m \exp(w_{x_1, \dots, x_k})} \\ &\leq c_f^m \\ &\leq c_f^\Delta. \end{aligned}$$

Similarly,

$$\mathcal{G}_2^{-1} \frac{\partial \mathcal{G}_2}{\partial w_{x_1^0, \dots, x_{k_0}^0}} \geq c_f^{-\Delta}.$$

Since the dimension of the argument  $\mathbf{w}$  is  $\sum_{1 \leq k \leq m} |\mathcal{X}|^m < \Delta \|X\|^\Delta$ , then we conclude

$$\|\nabla G(\mathbf{w})\|_{L_1} \leq (c_f^\Delta - c_f^{-\Delta}) \Delta |X|^\Delta \leq 1 - \gamma.$$



This concludes the proof. □

#### 4.1.4 Example: Potts model

One of the most widely studied objects in the statistical physics is  $q$ -state Potts model. It is described in the terminology of MRF as follows. Given a graph  $\mathbb{G}$  we set  $\phi_v = 1$  for all nodes  $v$ .  $\mathcal{X} = \{1, 2, \dots, q\}$ . A parameter  $\beta$  called inverse temperature is fixed. The coupling functions  $f$  are set as  $f_{u,v}(x, y) = \exp(\beta 1\{x = y\})$  for all nodes  $u, v$  and all elements  $x, y \in \mathcal{X}$ . The case  $\beta > 0$  corresponds to the *ferromagnetic* Potts model. In this case the distribution  $\mathbb{P}_{\mathbb{M}}(\cdot)$  "favors" assignments which select the same element along the edges. The case  $\beta < 0$  corresponds to the *anti-ferromagnetic* Potts model, and in this case the distribution favors assignments with different elements along the edges. The extreme case  $\beta = -\infty$  corresponds to the usual coloring problem, where monochromatic coloring are simply forbidden. The special case  $q = 2$  is called *Ising* model - one of the cornerstone models of the statistical physics.

It is immediate that conditions (4.1) and (4.2) are satisfied by this model provided  $|\beta| < \infty$ . Thus an immediate corollary of Theorem 7 is the following algorithmic result.

**Corollary 6.** *There exists a deterministic FPTAS for computing a partition function for a family of Potts model  $(\mathbb{G}, q, \beta)$  with constant constant degree  $\Delta$ , constant number of colors  $q$ , and satisfying*

$$(e^{\beta\Delta} - e^{-\beta\Delta})\Delta q^\Delta < 1.$$

Observe that for large  $\Delta$ , the largest inverse temperature  $\beta$  satisfying this condition behaves like  $O(\frac{1}{\Delta q^\Delta})$ . We believe that this is an overly conservative estimate. We conjecture that in fact the correlation decay property can be established in the regime

$$\beta = O\left(\frac{1}{\Delta}\right), \tag{4.11}$$

leading to a deterministic FPTAS.

## 4.2 Conclusions

We have extended our approach from Chapter 3 to constructing deterministic approximation algorithm for computing a partition function of a Markov random field satisfying certain conditions.

We conjecture that the polynomial time algorithms for computing partition function of a MRF can be constructed under weaker an assumption than (4.3). Specifically, we conjecture that for the case of Potts model, the critical inverse temperature  $\beta^*$  under which the correlation decay can be established on a computation tree behaves like (4.11).

# Chapter 5

## Counting matchings in a graph

The work in this chapter is joint with Mohsen Bayati, Chandra Neir, and Prasad Tetali.

This chapter focuses on the problem of counting the number of matchings in a graph and has the following structure. The definitions and the main result Theorem 9 are presented in Section 5.1. The correlation decay analysis is the subject of Section 5.2. The approximate counting algorithm and the complexity analysis is presented in Section 5.3. Some concluding remarks and further open questions are presented in Section 5.4.

### 5.1 Definitions, preliminaries and the main result

We consider a simple labeled undirected graph  $\mathbb{G}$  with  $n$  vertices. The vertex set is denoted by  $V = \{v_1, \dots, v_n\}$ .  $E$  denotes the edge set.  $N(v, \mathbb{G}) \subset V$  denotes the set of neighbors of  $v$ . We will also use  $N(v)$ , when the underlying graph  $\mathbb{G}$  is unambiguous. The degree of the graph is  $\Delta \triangleq \max_v |N(v, \mathbb{G})|$ . We abuse notations by letting  $\mathbb{G} \setminus \{v\}$  denote a subgraph of  $\mathbb{G}$  induced by nodes  $V \setminus \{v\}$ . A matching is a subset  $M \subseteq E$  such that no two edges in  $M$  share a vertex. We denote by  $\mathcal{M} = \mathcal{M}(\mathbb{G})$  the set of all matchings of  $\mathbb{G}$ .

Given a fixed parameter  $\lambda > 0$ , called the *activity*, a natural (Gibbs) probability distribution on the set  $\mathcal{M}$  of matchings is defined as:

$$\mathbb{P}_{\mathbb{G}}(M) = \frac{\lambda^{|M|}}{Z(\mathbb{G})},$$

where the normalizing constant  $Z(\mathbb{G})^1$  is called the *partition function* corresponding to  $\lambda$ , and is expressed as

$$Z(\mathbb{G}) = \sum_{M \in \mathbb{M}} \lambda^{|M|}.$$

When  $\lambda = 1$ ,  $Z(\mathbb{G})$  is simply the number of partial matchings  $|\mathbb{M}|$ . Our goal is constructing an algorithm which computes  $Z(\mathbb{G})$  approximately. The instance size of the problem is  $O(\max(|V|, |E|, |\log \lambda|))$ .

For convenience, we restate here the definition of a FPTAS.

**Definition 5.** *An approximation algorithm  $\mathcal{A}$  is a Fully Polynomial Time Approximation Scheme (FPTAS) for computing  $Z(\mathbb{G})$  if, given an arbitrary  $\epsilon > 0$ , it produces a value  $\hat{Z}$  satisfying*

$$\exp(-\epsilon) \leq \frac{\hat{Z}}{Z(\mathbb{G})} \leq \exp(\epsilon),$$

*in time which is polynomial in  $1/\epsilon$  and size instance.*

We now state our main result. Here and throughout  $\log$  denotes natural logarithm.

**Theorem 9.** *There exists a deterministic algorithm which provides FPTAS for computing  $Z(\mathbb{G})$  corresponding to a constant activity  $\lambda > 0$  for any graph, whose degree is bounded by a constant  $\Delta$ . The above algorithm has complexity  $O\left(\left(\frac{n}{\epsilon}\right)^{\kappa \log \Delta + 1}\right)$ , where  $\kappa = -2 / \log\left(1 - \frac{2}{\sqrt{1 + \lambda \Delta + 1}}\right)$ .*

Thus, while the running time of the algorithm depends polynomially on  $1/\epsilon$  (hence *Fully Polynomial* approximation scheme) the degree of the polynomial depends on both  $\Delta$  and  $\lambda$ .

Additionally, we will show that the algorithm of Theorem 9 has subexponential complexity when the graph is general (non-constant degree).

**Corollary 7.** *The complexity of the algorithm of Theorem 9 is  $\exp(O(\sqrt{n} \log^2 n))$  for general graphs and constant  $\lambda > 0$ .*

---

<sup>1</sup>For ease of notation we suppress the dependence of  $Z(\mathbb{G})$  on  $\lambda$ .

We now introduce an identity which appears in various forms in the context of Markov chain sampling method as well. The identity is also the essence of the cavity method in statistical physics. It shows that the problem of computing the partition function can be reduced to the problem of computing certain marginal probabilities. Problems admitting such reduction are called self reducible.

Denote by  $\mathbf{M}$  a random matching selected according to the Gibbs measure  $\mathbb{P}_{\mathbb{G}}$ . Abusing notation slightly, we write  $v \in \mathbf{M}$  to say that matching  $\mathbf{M}$  contains an edge incident to  $v$ .

**Proposition 9.** *Let  $\mathbb{G}_k = \mathbb{G} \setminus \{v_1, \dots, v_{k-1}\}$  with  $\mathbb{G}_0 = \mathbb{G}$ . Then*

$$Z(\mathbb{G}) = 1 / \prod_{1 \leq k \leq |V|} \mathbb{P}_{\mathbb{G}_k}(v_k \notin \mathbf{M}).$$

*Proof.* Observe that for every graph  $\mathbb{G}$  and vertex  $v \in \mathbb{G}$

$$\mathbb{P}_{\mathbb{G}}(v \notin \mathbf{M}) = \frac{Z(\mathbb{G} \setminus \{v\})}{Z(\mathbb{G})}. \quad (5.1)$$

Applying this identity recursively to  $v_k, \mathbb{G}_k$ , and using the convention that the partition function for the graph with no edges equals 1, we obtain the result.  $\square$

The following corollary is a straightforward application of Proposition 9 and therefore we shall focus our attention on constructing an algorithm for computing an approximation of  $\mathbb{P}_{\mathbb{G}}(v \notin \mathbf{M})$ .

**Corollary 8.** *Given any  $\epsilon > 0$ , if there exists a fully polynomial time algorithm  $\mathcal{A}$ , which on input  $(\mathbb{G}, v)$ , computes a value  $\hat{p}(v)$  satisfying*

$$\exp(-\frac{\epsilon}{n}) \leq \frac{\hat{p}(v)}{\mathbb{P}_{\mathbb{G}}(v \notin \mathbf{M})} \leq \exp(\frac{\epsilon}{n}), \quad (5.2)$$

*then one immediately obtains a fully polynomial time approximation algorithm for  $Z(\mathbb{G})$ .*

## 5.2 Basic recursion and correlation decay analysis

The following recursion is similar to the one obtained by Godsil in [God81] and Heilmann and Lieb [OE72]. It was also used by Kahn and Kim [KK98] for the analysis of random

matchings on regular large degree graphs.

**Proposition 10.** *The following holds for every vertex  $v$ :*

$$\mathbb{P}_{\mathbb{G}}(v \notin M) = \frac{1}{1 + \lambda \sum_{u \in N(v, \mathbb{G})} \mathbb{P}_{\mathbb{G} \setminus \{v\}}(u \notin M)}. \quad (5.3)$$

*Proof.* We have

$$Z(\mathbb{G}) = \sum_{M \in \mathbb{M}(\mathbb{G}): v \notin M} \lambda^{|M|} + \sum_{u \in N(v, \mathbb{G})} \sum_{M \in \mathbb{M}(\mathbb{G}): (v, u) \in M} \lambda^{|M|}. \quad (5.4)$$

Observe that the set of all matchings  $M$  such that  $v \notin M$  is the set of all matchings in  $\mathbb{G} \setminus \{v\}$ . Also for every matching  $M$  containing  $(v, u)$ ,  $M \setminus \{(v, u)\}$  induces a matching in the graph  $\mathbb{G} \setminus \{v, u\}$ . Conversely, for every matching  $M$  in  $\mathbb{G} \setminus \{v, u\}$ ,  $M \cup \{(v, u)\}$  creates a matching in  $\mathbb{G}$  containing the edge  $(v, u)$ . Thus we can rewrite (5.4) as

$$Z(\mathbb{G}) = Z(\mathbb{G} \setminus \{v\}) + \sum_{u \in N(v, \mathbb{G})} \lambda Z(\mathbb{G} \setminus \{v, u\}).$$

Dividing both parts by  $Z(\mathbb{G} \setminus \{v\})$  and using the identity (5.1) we obtain the result.  $\square$

**Definition 6.** *For every induced subgraph  $\mathbb{H}$  of the graph  $\mathbb{G}$  every vertex  $v \in \mathbb{H}$  and every  $t \in \mathbb{Z}_+$  let  $\Phi_{\mathbb{H}}(v, t)$  be defined inductively as follows:*

$$\Phi_{\mathbb{H}}(v, t) = \begin{cases} \Phi_{\mathbb{H}}(v, 0) = 1, & \text{for all } \mathbb{H}, v; \\ \left(1 + \lambda \sum_{u \in N(v, \mathbb{H})} \Phi_{\mathbb{H} \setminus \{v\}}(u, t-1)\right)^{-1}, & t \geq 1. \end{cases} \quad (5.5)$$

While we have introduced the values  $\Phi_{\mathbb{H}}(v, t)$  for potentially exponentially many subgraphs of  $\mathbb{G}$ , it is only a small family of subgraphs of  $\mathbb{G}$  for which the value of  $\Phi$  will be relevant to us. The quantity  $\Phi_{\mathbb{H}}(v, t)$  will serve as an approximation of  $\mathbb{P}_{\mathbb{H}}(v \notin M)$ . The essence of this approximation is described in the following result which is the basis of our algorithm.

**Theorem 10 (Correlation Decay).** *The following holds for every vertex  $v$  and every*

positive even value  $t$ :

$$\left| \log \mathbb{P}_{\mathbb{G}}(v \notin \mathbf{M}) - \log \Phi_{\mathbb{G}}(v, t) \right| \leq \left( 1 - \frac{2}{(\sqrt{1 + \lambda\Delta} + 1)} \right)^{t/2} \log(1 + \lambda\Delta).$$

*Proof.* Fix a vertex  $v \in \mathbb{G}$ , and let  $N(v, \mathbb{G}) = \{u_1, \dots, u_m\}$ ,  $N(u_i, \mathbb{G} \setminus \{v\}) = \{w_1^{(i)}, \dots, w_{m_i}^{(i)}\}$ . We introduce the following shorthand notations, with  $x$ 's representing the true probabilities (of certain vertices not being in random matchings) and  $y$ 's representing the corresponding approximations:

$$x = \log \mathbb{P}_{\mathbb{G}}(v \notin \mathbf{M}), \quad x_i = \log \mathbb{P}_{\mathbb{G} \setminus \{v\}}(u_i \notin \mathbf{M}), \quad x_j^{(i)} = \log \mathbb{P}_{\mathbb{G} \setminus \{v, u_i\}}(w_j^{(i)} \notin \mathbf{M}), \quad (5.6)$$

$$y = \log \Phi_{\mathbb{G}}(v, t), \quad y_i = \log \Phi_{\mathbb{G} \setminus \{v\}}(u_i, t - 1), \quad y_j^{(i)} = \log \Phi_{\mathbb{G} \setminus \{v, u_i\}}(w_j^{(i)}, t - 2), \quad (5.7)$$

for  $i = 1, \dots, m$ ,  $j = 1, \dots, m_i$ .

Let  $M = \sum_{i=1}^m m_i$ ,  $\vec{z} = (z_1^{(1)}, \dots, z_{m_1}^{(1)}, \dots, z_1^{(m)}, \dots, z_{m_m}^{(m)})$ . Let  $f : [0, 1]^M \rightarrow [0, 1]$  be given as

$$f(\vec{z}) = \log \left( 1 + \lambda \sum_{i=1}^m \frac{1}{1 + \lambda \sum_{j=1}^{m_i} e^{z_j^{(i)}}} \right).$$

Then we can rewrite (5.3) and (5.5) as  $x = -f(\vec{x})$ ,  $y = -f(\vec{y})$ . We consider  $g(\alpha) = f(\alpha\vec{x} + (1 - \alpha)\vec{y})$  as a function of one-dimensional parameter  $\alpha \in [0, 1]$  and fixed vectors  $\vec{x}, \vec{y}$ . Applying the mean value theorem, there exists  $\alpha \in [0, 1]$  such that for  $\vec{z}_\alpha = \alpha\vec{x} + (1 - \alpha)\vec{y}$ ,

$$|x - y| = |\nabla f(\vec{z}_\alpha) \cdot (\vec{x} - \vec{y})| \stackrel{(a)}{\leq} \|\nabla f(\vec{z}_\alpha)\|_{L_1} \|\vec{x} - \vec{y}\|_{L_\infty}, \quad (5.8)$$

where (a) follows from  $|\sum a_i b_i| \leq (\sum |a_i|) \max |b_i| = \|a\|_{L_1} \|b\|_{L_\infty}$ . It is easy to see that

$$\|\nabla f(\vec{z})\|_{L_1} = \frac{1}{1 + \lambda \sum_{i=1}^m \frac{1}{1 + \lambda \sum_{j=1}^{m_i} e^{z_j^{(i)}}}} \sum_{i=1}^m \lambda \left( \frac{1}{1 + \lambda \sum_{j=1}^{m_i} e^{z_j^{(i)}}} \right)^2 \lambda \sum_{j=1}^{m_i} e^{z_j^{(i)}}.$$

**Lemma 18.** *For every  $\vec{z}$ ,*

$$\|\nabla f(\vec{z})\|_{L_1} \leq 1 - \frac{2}{(\sqrt{1 + \lambda\Delta} + 1)}.$$

*Proof.* Define  $A_i = 1 + \lambda \sum_{j=1}^{m_i} e^{z_j^{(i)}}$ . The  $L_1$ -norm can be re-written, in terms of  $A_i$  as

$$\|\nabla f(\vec{z})\|_{L_1} = \frac{1}{1 + \lambda \sum_{i=1}^m \frac{1}{A_i}} \sum_{i=1}^m \frac{\lambda(A_i - 1)}{A_i^2} = 1 - \frac{1 + \lambda \sum_{i=1}^m \frac{1}{A_i^2}}{1 + \lambda \sum_{i=1}^m \frac{1}{A_i}}.$$

It is not difficult to see that the expression  $\frac{1 + \lambda \sum_{i=1}^m \frac{1}{A_i^2}}{1 + \lambda \sum_{i=1}^m \frac{1}{A_i}}$  is minimized, for  $0 \leq 1/A_i \leq \infty$ , when  $1/A_i = \frac{\sqrt{1 + \lambda m} - 1}{\lambda m}$ . To show this, first observe by taking partial derivatives that the minimum occurs when all of  $A_i$  are equal. Then the solution for optimal  $A_i$  reduces to a quadratic equation. Substituting for the minimum value, one obtains

$$\|\nabla f(\vec{z})\|_{L_1} = 1 - \frac{1 + \lambda \sum_{i=1}^m \frac{1}{A_i^2}}{1 + \lambda \sum_{i=1}^m \frac{1}{A_i}} \leq 1 - \frac{2}{(\sqrt{1 + \lambda m} + 1)} \leq 1 - \frac{2}{(\sqrt{1 + \lambda \Delta} + 1)}.$$

□

Applying Lemma 18 to (5.8) we obtain

$$\begin{aligned} & \left| \log \mathbb{P}_{\mathbb{G}}(v \notin \mathbf{M}) - \log \Phi_{\mathbb{G}}(v, t) \right| \\ & \leq \left( 1 - \frac{2}{(\sqrt{1 + \lambda \Delta} + 1)} \right) \max_{i,j} \left| \log \mathbb{P}_{\mathbb{G} \setminus \{v, u_i\}}(w_j^{(i)} \notin \mathbf{M}) - \log \Phi_{\mathbb{G} \setminus \{v, u_i\}}(w_j^{(i)}, t - 2) \right|. \end{aligned}$$

Iterating this bound  $t/2$  times we obtain that  $\left| \log \mathbb{P}_{\mathbb{G}}(v \notin \mathbf{M}) - \log \Phi_{\mathbb{G}}(v, t) \right|$  is at most  $(1 - \frac{2}{(\sqrt{1 + \lambda \Delta} + 1)})^{t/2}$  times  $\max_{\mathbb{H}, u} \left| \log \mathbb{P}_{\mathbb{H}}(u \notin \mathbf{M}) - \log \Phi_{\mathbb{H}}(u, 0) \right|$ , where the maximum is over all subgraph/vertex pairs ( $\mathbb{H} \subset \mathbb{G}, u \in \mathbb{H}$ ). Observe from (5.3) that

$$\mathbb{P}_{\mathbb{G}}(v \notin \mathbf{M}) \geq \frac{1}{1 + \lambda \Delta}.$$

Applying this bound and the fact  $\Phi_{\mathbb{H}}(u, 0) = 1$ ,  $\max_{\mathbb{H}, u} \left| \log \mathbb{P}_{\mathbb{H}}(u \notin \mathbf{M}) - \log \Phi_{\mathbb{H}}(u, 0) \right|$  is at most  $\log(1 + \lambda \Delta)$ . □

**Remark :** In the proof below we analyzed two steps of the recursions (5.3) and (5.5). This leads to a correlation decay rate  $\approx (1 - \frac{1}{\sqrt{\lambda \Delta}})$ . We could instead use a one-step analysis, but this would give us a decay rate only  $\approx (1 - \frac{1}{\lambda \Delta})$ . While this would not make a big difference in the case  $\lambda, \Delta = O(1)$ , it does make a difference in the general case, since  $\Delta$



could be as large as  $n - 1$ . The case  $\lambda = O(n^{\frac{1}{3}})$  will be used in Chapter 6 for constructing an approximation algorithm for computing a permanent. It also seems that 3-step analysis would not buy us better correlation decay as the rate  $\approx (1 - \frac{1}{\sqrt{\lambda\Delta}})$  is tight, as can be checked on a regular tree. This observation is also implicit in [KK98].

## 5.3 Algorithm

Our algorithm is based on computing the values  $\Phi_{\mathbb{G}}(v, t)$ . In our analysis of algorithm complexity, we assume that each arithmetic operation takes one unit of time. This can be done since arithmetic operations introduce at most a polynomial time overhead in the computation.

**Lemma 19.** *The values  $\Phi_{\mathbb{G}}(v, t)$  can be computed in time  $O(\Delta^t)$ . In particular when  $t = O(\log n)$  and  $\Delta = O(1)$ ,  $\Phi_{\mathbb{G}}(v, t)$  can be computed in polynomial time.*

*Proof.* The proof follows immediately from recursion (5.5). □

Based on this lemma, we propose the following algorithm for estimating the partition function  $Z(\mathbb{G})$ .

### Algorithm CountMATCHINGS

INPUT: A graph/activity pair  $(\mathbb{G}, \lambda)$  and a positive integer  $t$ .

BEGIN

Set  $Z = 1, \mathbb{H} = \mathbb{G}$ .

While  $\mathbb{H} \neq \emptyset$ , find an arbitrary node  $v \in \mathbb{H}$ . Compute  $\Phi_{\mathbb{H}}(v, t)$ .

Set  $Z = \frac{Z}{\Phi_{\mathbb{H}}(v, t)}$ ,  $\mathbb{H} = \mathbb{H} \setminus \{v\}$ .

END

OUTPUT:  $Z$ .

As a final step we show that  $\Phi$  can be used to approximate the marginal probabilities  $\mathbb{P}_{\mathbb{G}}(v \notin \mathcal{M})$  with polynomial accuracy.

**Lemma 20.** Let  $\delta = -\log\left(1 - \frac{2}{(\sqrt{1+\lambda\Delta}+1)}\right)$ . If  $t = 2\lceil(\log n + \log \log(1 + \lambda\Delta) - \log \epsilon)/\delta\rceil$ , then

$$e^{-\frac{\epsilon}{n}} \leq \frac{\Phi_{\mathbb{G}}(v, t)}{\mathbb{P}_{\mathbb{G}}(v \notin \mathbf{M})} \leq e^{\frac{\epsilon}{n}}.$$

*Proof.* Theorem 10 implies

$$\begin{aligned} \left| \log \frac{\Phi_{\mathbb{G}}(v, t)}{\mathbb{P}_{\mathbb{G}}(v \notin \mathbf{M})} \right| &\leq \left(1 - \frac{2}{(\sqrt{1 + \lambda\Delta} + 1)}\right)^{(\log n + \log \log(1 + \lambda\Delta) - \log \epsilon)/\delta} \log(1 + \lambda\Delta) \\ &= \frac{\epsilon}{n}. \end{aligned}$$

The result then follows. □

*Proof of Theorem 9.* First we assume the case of bounded degree graph and constant activity:  $\lambda, \Delta = O(1)$ . The bound on  $t$  given by Lemma 20 becomes in this case  $t = O(\log n)$ . The algorithm providing FPTAS is CountMATCHINGS, with input  $\mathbb{G}, \lambda$ , and  $t = O(\log n)$  as in Lemma 20. We can combine Lemma 19, Lemma 20 and Corollary 8 and observe that the complexity of the algorithm is bounded by  $O(n\Delta^t)$  where  $t = 2\lceil(\log n + \log \log(1 + \lambda\Delta) - \log \epsilon)/\delta\rceil$ . This gives the desired FPTAS with the complexity bounded as stated in the theorem.

In the general case we have for  $\delta$  defined in Lemma 20 that  $\delta^{-1} = O(\sqrt{\lambda\Delta})$ . Due to our assumption  $\lambda = O(n)$ , this gives  $t = O(\frac{1}{\delta} \log \frac{n}{\epsilon}) = O(\sqrt{\lambda\Delta} \log \frac{n}{\epsilon})$ . Thus again applying Lemma 19, the complexity of the algorithm CountMATCHINGS is

$$O(n\Delta^t) = O(n^t) = \exp(O(\sqrt{\lambda\Delta} \log^2 n)),$$

where  $\Delta = O(n)$  is used. (Here we ignore the explicit dependence on  $\epsilon$  but it is easy to see that the complexity depends polynomially on  $\frac{1}{\epsilon}$ .) The special case corresponding to counting matchings  $\lambda = 1$  leads to an upper bound  $\exp(O(\sqrt{n} \log^2 n))$ . □

## 5.4 Conclusions

We have constructed a deterministic algorithm for counting approximately the number of matchings of a given graph. The algorithm runs in polynomial time for the class of bounded

degree graphs, and in subexponential time  $\exp(O(\sqrt{n} \log^2 n))$  for the class of all graphs, where  $n$  is the number of nodes.

A natural open question is whether there is a FPTAS for counting matchings in graphs of unbounded degrees? There seem to be some fundamental limitations of the approach proposed in this thesis – the correlation decay rate corresponding to the case of matchings seems to be of order  $1 - O(\frac{1}{\sqrt{\Delta}})$ , and thus we speculate that the improvement should come along some combinatorial, (rather than statistical physics), type arguments. In general, it is of interest to see to what extent the correlation decay approach can be used for solving approximately other counting problems for which the MCMC method has been successful. This line of investigation might also bring us a step closer to understanding the extent to which randomized algorithms are more powerful than deterministic algorithms.



# Chapter 6

## Computing the Permanent of a matrix

The chapter contains our contribution to the problem of computing a permanent of a 0,1 matrix, and has the following structure. Definitions, preliminaries and the statements of the main results are subject of the following section. Section 6.2 is devoted to constructing approximation algorithm for the case when the underlying graph is a constant degree expander. The general case is considered in Section 6.3. We conclude with some possible further directions in Section 6.4.

### 6.1 Preliminaries and the main result

Consider a simple undirected  $n$  by  $n$  bi-partite graph  $\mathbb{G}$  with the node set  $V = V_1 \cup V_2$ ,  $|V_1| = |V_2| = n$ . Let  $E$  be the set of edges of the graph.  $N(v, \mathbb{G}) \subset V$  denotes the set of neighbors of  $v \in V$  and  $\Delta(v) = |N(v, \mathbb{G})|$  denotes the degree of the node  $v$ . The degree of the graph is  $\Delta \triangleq \max_{v \in V} \Delta(v)$ . Given a set of nodes  $A$ , we denote by  $N(A)$  or specifically by  $N(A, \mathbb{G})$  the set of neighbors of  $A$  (in  $\mathbb{G}$ ). Given  $\alpha > 0$ , a graph  $\mathbb{G}$  is defined to be an *expander* or specifically  $\alpha$ -*expander* if for every set of nodes  $A \subset V_i, i = 1, 2$  such that  $|A| \leq n/2$ , the inequality  $|N(A)| \geq (1 + \alpha)|A|$  holds. We also define

$$\alpha(\mathbb{G}) \triangleq \max_A \frac{|N(A)|}{|A|} - 1$$

to be the *expansion* of the graph  $\mathbb{G}$ , where the maximum is over all subsets  $A \subset V_i, i = 1, 2$  with  $|A| \leq n/2$ . Clearly,  $\mathbb{G}$  is  $\alpha$ -expander if its expansion is at least  $\alpha$ .

A matching is a subset  $M \subset E$  such that no two edges in  $M$  share a node. For every  $k \leq n$  let  $M(k)$  be the number of size  $k$  matchings in  $\mathbb{G}$ . Specifically,  $M(n)$  is the number of full matchings.

Given a graph  $\mathbb{G}$  let  $A = (a_{ij})$  be the corresponding adjacency matrix. The rows and columns of  $A$  are indexed by nodes of  $V_1$  and  $V_2$  respectively, and  $a_{ij} = 1$  if  $(v_i, v_j) \in E$  and  $a_{ij} = 0$  otherwise. It is immediate that  $M(n) = \text{Perm}(A)$ .

A parameter  $\lambda > 0$  is fixed and is called the *activity*. The *partition function* corresponding to  $\lambda$  is defined as

$$Z(\lambda, \mathbb{G}) = \sum_M \lambda^{|M|} = \sum_{0 \leq k \leq n} \lambda^k M(k).$$

This object is also known as matching polynomial [God81]. A partition function is an important object in statistical physics. The case of matching is usually called monomer-dimer model in the statistical physics literature.

For convenience, we restate here the definition of a FPTAS.

**Definition 7.** *An approximation algorithm  $\mathcal{A}$  is defined to be a Fully Polynomial Time Approximation Scheme (FPTAS) for a computing  $Z(\lambda, \mathbb{G})$  if given an arbitrary  $\delta > 0$  it produces a value  $\hat{Z}$  satisfying*

$$\frac{1}{1 + \delta} \leq \frac{\hat{Z}}{Z(\lambda, \mathbb{G})} \leq 1 + \delta,$$

*in time which is polynomial in  $n$  and  $\frac{1}{\delta}$ . An approximation algorithm is defined to be Polynomial Time Approximation Scheme (PTAS) if the computation time is polynomial in  $n$ , but not necessarily in  $\frac{1}{\delta}$*

The following result was established in Chapter 5

**Theorem 11.** *There exist a deterministic algorithm which provides a FPTAS for computing  $Z(\lambda, \mathbb{G})$  for an arbitrary graph/activity pair  $(\mathbb{G}, \lambda)$  when  $\Delta$  and  $\lambda$  are constants. The complexity of the same algorithm is  $\exp(O(\sqrt{\lambda\Delta} \log \Delta \log n))$  for general  $\Delta$  and  $\lambda$ .*

The case  $\lambda = 1$  corresponds to counting the number of partial matchings of a graph. In this chapter we use the algorithm underlying Theorem 11 as a subroutine to devise an approximation algorithm for computing a permanent. For this purpose we will be making  $\lambda$  to be appropriately large. Throughout this chapter we assume  $\lambda > 10$ .

We now state the two main results of this chapter. In the next and the following results the notion of  $(1 + \epsilon)^n$  multiplicative approximation factor of  $\text{Perm}(\mathbb{G})$  corresponds to obtaining a value  $\hat{Z}$  satisfying  $(1 + \epsilon)^{-n} \leq \frac{\hat{Z}}{\text{Perm}(\mathbb{G})} \leq (1 + \epsilon)^n$ .

**Theorem 12.** *Let  $\mathbb{G}$  be an  $n$  by  $n$  bi-partite  $\alpha$ -expander and let  $\epsilon > 0$ . There exist a deterministic  $(1 + \epsilon)^n$  approximation algorithm for computing  $\text{Perm}(\mathbb{G})$  with running time*

$$n^{O\left(\sqrt{\frac{\Delta \log \Delta}{\epsilon \alpha}} \log \Delta\right)}. \tag{6.1}$$

*Specifically, the running time is polynomial (PTAS) in  $n$  whenever  $\Delta, \alpha$  are constants.*

Thus we construct a  $(1 + \epsilon)^n$  polynomial time approximation algorithm when the underlying graph is a constant degree expander. Our algorithm corresponding to the second part of the theorem, while polynomial, is not strongly polynomial, as the dependence of the running time on the approximation parameter  $\epsilon$  is of the form  $n^{O\left(\frac{1}{\epsilon}\right)^{\frac{1}{2}}}$ . While our approximation factor  $(1 + \epsilon)^n$  is a far cry from PTAS (namely  $1 + \epsilon$  approximation factor), it is still a significant improvement over  $e^n$  factor constructed in [LSW00].

Our second result does not require any restrictions on the underlying graph.

**Theorem 13.** *There exist a deterministic  $(1 + \epsilon)^n$  approximation algorithm for computing  $\text{Perm}(\mathbb{G})$  of an arbitrary  $n$  by  $n$  bi-partite graph  $\mathbb{G}$  which runs in time  $\exp(O(\epsilon^{-\frac{1}{2}} n^{\frac{2}{3}} \log^3 n))$ .*

Thus, similarly to [JV96], our algorithm provides a mildly exponential algorithm for approximating a permanent of a graph. While the approximation factor  $(1 + \epsilon)^n$  is weaker than the one of [JV96], our algorithm is deterministic.

## 6.2 Constant degree expanders

Proof of Theorem 12 is given in this section. We begin by establishing some preliminary results. We assume without the loss of generality that  $M(n) \geq 1$ . Consider an arbitrary  $k$  matching  $M$  between sets  $A_1 \subset V_1, A_2 \subset V_2, |A_1| = |A_2| = k$ . A path  $v_1, v_2, \dots, v_{2r}$  is defined to be an *alternating path* wrt  $M$  if  $v_1 \in V_1 \setminus A_1, v_{2r} \in V_2 \setminus A_2$ , if  $(v_{2l}, v_{2l+1}) \in M, 1 \leq l \leq r-1$  and  $(v_{2l-1}, v_{2l}) \notin M, 1 \leq l \leq r$ . Observe that given  $M$  and an alternating path  $P$ , one can construct a  $k+1$  matching, by subtracting from  $M$  all the edges  $(v_{2l}, v_{2l+1}), 1 \leq l \leq r$  and adding all the edges  $(v_{2l-1}, v_{2l}), 1 \leq l \leq r-1$ . The length of this alternating path is defined to be  $2r-1$ .

**Lemma 21.** *Let  $M$  be a  $k$  matching between  $A_1$  and  $A_2$  for  $k \leq n-1$ . For every set  $L \subset V_1 \setminus A_1$  with  $|L| \geq (n-k)/2$  there exists an alternating path  $P$  with end points in  $L$  and  $V_2 \setminus A_2$  and with length at most*

$$O\left(\log\left(\frac{2n}{n-k}\right)\log^{-1}(1+\alpha)\right). \quad (6.2)$$

*Proof.* The proof is similar to the argument of Lemma 2 [JV96]. Let  $R = V_2 \setminus A_2$ . Let  $L_r$  be the set of nodes in  $V_2$  reachable from  $L$  en route of alternating paths with length at most  $2r$ . Let  $r_0$  be a minimum  $r$  satisfying  $(1+\alpha)^r(n-k)/2 \geq n$ . Note  $r_0 = O\left(\log\left(\frac{2n}{n-k}\right)\log^{-1}(1+\alpha)\right)$ . Since  $|L| \geq (n-k)/2$ , then  $l(r_0) \triangleq \min((1+\alpha)^{r_0}|L|, \frac{n}{2}+1) = \frac{n}{2}+1$ . For every  $r \leq r_0$ , by the expansion property either  $|L_r| \geq \min((1+\alpha)^r|L|, \frac{n}{2}+1)$ , or  $L_{r'} \cap R \neq \emptyset$  for some  $r' \leq r$ . In the second case we found an alternating path with length  $\leq l(r_0)$ . In the first case we have  $|L_{r_0}| > n/2$ . We now claim that for every  $r > r_0$  either  $L_r \cap V_2 \setminus A_2 \neq \emptyset$  or  $|V_2 \setminus L_r| \leq \frac{1}{(1+\alpha)^{r-r_0}}(n/2)$ . Indeed, let  $L'_r \subset V_1$  be the set of nodes matched to  $L_r$ . In particular  $|L'_{r'}| > n/2$ . Since  $|L_{r+1}| = |N(L'_{r'})| > n/2$ , then

$$|V_2 \setminus L_r| = |V_1 \setminus L'_{r'}| \geq |N(V_2 \setminus L_{r+1})| \geq (1+\alpha)|V_2 \setminus L_{r+1}|,$$

and the assertion is established by induction. For  $r \geq 2r_0$  we obtain  $\frac{1}{(1+\alpha)^{r-r_0}}(n/2) < n-k$  and thus we must have  $L_r \cap V_2 \setminus A_2 \neq \emptyset$ . We established that for some  $r \leq 2r_0$  there exists an alternating path with end points in  $L$  and  $V_2 \setminus A_2$ . The length of this path is  $O(r_0) = O(\log(\frac{2n}{n-k})\log^{-1}(1+\alpha))$ . This completes the proof.  $\square$



**Lemma 22.** For every  $k \leq n - 1$

$$\frac{M(k)}{M(k+1)} \leq \left(\frac{2n}{n-k}\right)^{O\left(\log^{-1}(1+\alpha)\log\Delta\right)}. \quad (6.3)$$

As a result

$$\frac{M(k)}{M(n)} \leq \left(\frac{2en}{n-k}\right)^{O\left((n-k)\log^{-1}(1+\alpha)\log\Delta\right)}. \quad (6.4)$$

**Remark :** Before we prove the lemma, we compare this result with a similar result established in [JV96], namely, Lemma 2. It is established there that  $M(n-1)/M(n) = O(\exp(\log^2 n))$  (treating  $\alpha$  as a constant). One could extend their argument to show that  $M(k)/M(k+1) = \exp(O(\log n \log(n/(n-k))))$ . Applying this bound straightforwardly, we obtain  $M(k)/M(n) = \exp(O(n \log^2 n))$  when  $n-k = \Omega(n)$ . Unfortunately, this bound grows superexponentially. This is not good enough, as shall shortly see. We need to obtain a bound which grows at most exponentially in  $n$  when  $n-k = \Omega(n)$ , and the bound (6.4) achieves just that. This bound is achieved in our Lemma 22 using a more careful counting argument.

*Proof.* Fix an arbitrary  $k$  matching  $M$  between  $A_1 \subset V_1$  and  $A_2 \subset V_2$ . We claim that there exist at least  $(n-k)/2$   $k+1$ -matchings obtained from  $M$  via an alternating path with length at most the value given by (6.2). Indeed, consider the set of all nodes  $v$  in  $V_1 \setminus A_1$  such that the shortest alternating path starting from  $v$  is larger than the required bound. By Lemma 21 this set contains less than  $(n-k)/2$  nodes and the assertion follows.

Now consider the following bi-partite graph. The nodes on the left (right) are all  $k$  ( $k+1$ )-matchings. We put an edge between two matchings  $M, M'$  if  $M'$  is obtained from  $M$  via an alternating path with length bounded by the expression in (6.2). The total number of edges in this graphs is at least  $M(k)(n-k)/2$  by our observation above. For every matching  $M'$  on the right side of the graph the number of edges pointing to it is at most the number of alternating paths with length at most (6.2) which result in  $M'$ . For every possible starting node of an alternating path, the number of such alternating paths is at most

$$\Delta^{O\left(\log\left(\frac{2n}{n-k}\right)\log^{-1}(1+\alpha)\right)}.$$

The number of starting nodes is bounded by  $n$ . Then the total number of edges in this graph is at most  $M(k+1)n\Delta^{O\left(\log\left(\frac{2n}{n-k}\right)\log^{-1}(1+\alpha)\right)}$ . We conclude that

$$M(k+1)n\Delta^{O\left(\log\left(\frac{2n}{n-k}\right)\log^{-1}(1+\alpha)\right)} \geq M(k)(n-k)/2.$$

The bounds (6.3) then follows. From this bound we also obtain

$$\frac{M(k)}{M(n)} \leq \left(\frac{(2n)^{n-k}}{(n-k)!}\right)^{O\left(\log^{-1}(1+\alpha)\log\Delta\right)} = \left(\frac{(2en)^{n-k}}{(n-k)^{n-k}}\right)^{O\left(\log^{-1}(1+\alpha)\log\Delta\right)},$$

where the Stirling's approximation was used in the equality. This is the required bound (6.4).  $\square$

**Corollary 9.** *The following holds*

$$1 \leq \frac{Z(\lambda, \mathbb{G})}{\lambda^n \text{Perm}(\mathbb{G})} \leq \exp(O(n\lambda^{-1}\log^{-1}(1+\alpha)\log\Delta)). \quad (6.5)$$

*Proof.* The inequality  $Z(\lambda, \mathbb{G}) \geq \lambda^n \text{Perm}(\mathbb{G})$  is immediate. We focus on the second inequality in (6.5). We need to analyze the ratio

$$\frac{\lambda^k M(k)}{\lambda^n M(n)} = \lambda^{-(n-k)} \frac{M(k)}{M(n)}. \quad (6.6)$$

We set  $c_n \triangleq \log^{-1}(1+\alpha)\log\Delta$  for simplicity. Applying the second part of Lemma 22

$$\lambda^{-(n-k)} \frac{M(k)}{M(n)} \leq \exp\left(O\left(c_n(n-k)\left(\log n - \log(n-k) - \log\frac{\lambda}{2e}\right)\right)\right)$$

Consider the problem of maximizing

$$g(x) \triangleq x \log n - x \log x - x \log \frac{\lambda}{2e}$$

in the range  $x \in [1, n]$ . The boundary cases  $x = 1$  and  $x = n$  give, respectively, the values  $\log(2en/\lambda)$ ,  $-n \log(\lambda/2e)$ . The second quantity is negative when  $\lambda > 2e$  (recall our assumption  $\lambda > 10$ ). To find another candidate for the largest value, we take the derivative with

respect to  $x$  and equating it to zero we obtain

$$\log n - \log x - \log \frac{\lambda}{2} = 0,$$

giving  $x = 2n/\lambda$ . Evaluating  $g(x)$  at this value simplifies to  $2n/\lambda$  and this gives the largest value of  $g$  when  $n$  is larger than some  $\lambda$  dependent constant. We conclude that the left-hand side of (6.6) is at most  $\exp(O(\frac{c_n n}{\lambda}))$ , implying

$$\frac{Z(\lambda, \mathbb{G})}{\lambda^n \text{Perm}(\mathbb{G})} \leq n \exp(O(\frac{c_n n}{\lambda})) = \exp(O(\frac{c_n n}{\lambda} + \log n)) = \exp(O(\frac{c_n n}{\lambda})).$$

□

*Proof of Theorem 12.* Fix an arbitrary constant  $\epsilon > 0$ . We select the smallest  $\lambda$  so that  $\exp(O(\lambda^{-1} \log^{-1}(1 + \alpha) \log \Delta)) < 1 + \epsilon$ . It is clear that  $\lambda = O(\epsilon^{-1} \alpha^{-1} \log \Delta)$ . We compute an  $\epsilon$ -approximation  $\tilde{Z}$  of  $Z(\lambda, \mathbb{G})$  using the algorithm underlying Theorem 11 for computing  $Z(\lambda, \mathbb{G})$ . By Corollary 9, it satisfies

$$(1 - \epsilon) \leq \frac{\tilde{Z}}{\lambda^n \text{Perm}(\mathbb{G})} \leq (1 + \epsilon)^{n+1}.$$

Then  $\tilde{Z}/\lambda^n$  provides the required approximation. The complexity of this algorithm is

$$\exp(O(\sqrt{\lambda \Delta} \log \Delta \log n))$$

which is (6.1) for our choice of  $\lambda$ , and the first part of the theorem is established.

For the second part we observe that  $\lambda = O(\epsilon^{-1} \alpha^{-1} \log \Delta)$  is a constant whenever  $\alpha$  and  $\Delta$  are constants. We recall from Theorem 11 that the algorithm for computing  $Z(\lambda, \mathbb{G})$  is polynomial time, under these assumptions. □

## 6.3 General graphs

*Proof of Theorem 13.* Our approach uses Jerrum-Vazirani expander decomposition approach [JV96]. The idea is to decompose the underlying graph into a collection of subgraphs with

a suitable expansion properties and apply an algorithm for computing the permanent recursively. In [JV96] the subroutine used is based on the algorithm relying on rapidly mixing Markov chain. Here we use the deterministic algorithm constructed in the proof of Theorem 12.

The following result is established in [JV96] (Lemma 4). There exists an algorithm, called

**TestExpansion** which on input  $\mathbb{G}, \alpha$  either identifies that  $\mathbb{G}$  is an  $\alpha$ -expander, or identifies a set  $A \subset V_1, |A| \leq n/2$  such that  $N(A) \leq (1 + 2\alpha)|A|$ . The running time of the algorithm is  $\exp(O(\alpha n \log n))$ .

Given an arbitrary set  $A \subset V_1$  it is straightforward to observe that

$$\text{Perm}(\mathbb{G}) = \sum_{B \subset N(A), |B|=|A|} \text{Perm}(A, B) \text{Perm}(A^c, B^c) \quad (6.7)$$

where  $A^c = V_1 \setminus A, B^c = V_2 \setminus B$  and  $\text{Perm}(A, B)$  is the permanent of the subgraph induced by  $A$  and  $B$ . Based on this observation we propose the following recursive algorithm for computing  $\text{Perm}(\mathbb{G})$ . We suppose that we have an algorithm  $\mathcal{A}_r$  which computes  $(1 + \epsilon)^r$  factor approximation of a permanent of any  $r$  by  $r$  bipartite graph in time  $g(r)$ , for every  $r \leq n - 1$ . We use it to construct  $\mathcal{A}_n$  as follows. Set  $\alpha = n^{-1/3}$ . Run algorithm **TestExpansion** on  $\mathbb{G}$ . The running time is  $\exp(O(n^{2/3} \log n))$ . If the algorithm returns no set  $A$ , then the underlying graph is an  $\alpha$ -expander and we use algorithm of Theorem 12 to obtain an  $(1 + \epsilon)^n$  approximation of  $\text{Perm}(\mathbb{G})$ . Using a bound  $\Delta \leq n$ , the running time of this algorithm is  $\exp(O(\sqrt{\epsilon^{-1} n^{1/3} \log^3 n})) = \exp(O(\epsilon^{-1/2} n^{2/3} \log^3 n))$ , and the overall running time is  $\exp(O(n^{2/3} \log n)) + \exp(O(\epsilon^{-1/2} n^{2/3} \log^3 n)) = \exp(O(\epsilon^{-1/2} n^{2/3} \log^3 n))$ . Let  $c_0$  be the constant hidden in  $O(\cdot)$ . From now on, treating  $\epsilon$  as constant, we hide  $\epsilon^{-1/2}$  in the  $O(\cdot)$  term. In the end we observe that the dependence of the running time on  $\epsilon$  is of the form  $\exp(O(\epsilon^{-1/2}))$ .

Otherwise the algorithm **TestExpansion** identifies a set  $A$  with  $|N(A)| \leq (1 + 2\alpha)|A|$ . We estimate  $\text{Perm}(A, B)$  and  $\text{Perm}(A^c, B^c)$  using algorithm  $\mathcal{A}_r$  with  $r = |A|, |A^c|$  respectively. Then we estimate  $\text{Perm}(\mathbb{G})$  using the expression (6.7). For every product  $\text{Perm}(A, B) \text{Perm}(A^c, B^c)$  our approximation factor is  $(1 + \epsilon)^{|A|} (1 + \epsilon)^{|A^c|} = (1 + \epsilon)^n$  by the recursive assumption.

Now we obtain an upper bound on  $g(n)$  and specifically show that it is  $\exp(O(n^{2/3} \log^3 n))$ . Let  $c > \max(c_0, 8)$ . By the recursive assumption  $g(r) \leq \exp(cr^{2/3} \log^3 r)$ ,  $r \leq n - 1$ . The

function  $g$  satisfies the following bound

$$g(n) \leq \max \left( \exp(c_0 n^{\frac{2}{3}} \log^3 n), \max_{1 \leq r \leq n/2} \binom{\min(n, r(1+2\alpha))}{r} (g(r) + g(n-r)) \right). \quad (6.8)$$

Here the term  $\binom{\min(n, r(1+2\alpha))}{r}$  comes from performing the computation over all subsets  $B$  of  $N(A)$  with size  $|B| = |A|$ . We have

$$\binom{\min(n, r(1+2\alpha))}{r} < (\min(n, r(1+2\alpha)))^{\min(n, r(1+2\alpha))-r} \leq n^{2\alpha r}$$

We have

$$n^{2\alpha r} = \exp(2n^{-\frac{1}{3}} r \log n) \quad (6.9)$$

$$\leq \exp(n^{\frac{2}{3}} \log n). \quad (6.10)$$

Now, by the recursive assumption we have for  $r \leq n/2$  that  $g(r) \leq \exp(cr^{\frac{2}{3}} \log^3 r) \leq \exp((3/4)cn^{\frac{2}{3}} \log^3 n)$ , which gives

$$n^{2\alpha r} g(r) \leq \exp((7/8)cn^{\frac{2}{3}} \log^3 n),$$

since  $1 + (3/4)c < c + (3/4)c = (7/8)c$ . As for  $g(n-r)$ , we have for sufficiently large  $n$  using (6.9)

$$\begin{aligned} n^{2\alpha r} g(n-r) &\leq \exp(2n^{-\frac{1}{3}} r \log n) \exp(c(n-r)^{\frac{2}{3}} \log^3(n-r)) \\ &\leq \exp(2n^{-\frac{1}{3}} r \log n) \exp(c(n-r)^{\frac{2}{3}} \log^3 n) \\ &\stackrel{(a)}{\leq} \exp(2n^{-\frac{1}{3}} r \log n) \exp(cn^{\frac{2}{3}} \log^3 n - (cr/2)n^{-\frac{1}{3}} \log^3 n) \\ &\stackrel{(b)}{\leq} \exp(cn^{\frac{2}{3}} \log^3 n - 2n^{-\frac{1}{3}}) \\ &\leq (1 - n^{-\frac{1}{3}}) \exp(cn^{\frac{2}{3}} \log^3 n), \end{aligned}$$

where in (a) we use  $(n-r)^{\frac{2}{3}} \leq n^{\frac{2}{3}} - \frac{r}{2}n^{-\frac{1}{3}}$  (obtained, for example, using Taylor's expansion around  $n^{\frac{2}{3}}$ ); and in (b) we use  $r \geq 1$  and  $c > 8$ . Using  $\exp((7/8)cn^{\frac{2}{3}} \log^3 n) \leq$

$n^{-\frac{1}{3}} \exp(cn^{\frac{2}{3}} \log^3 n)$  for large  $n$ , we obtain that for every  $1 \leq r \leq n - 1$

$$n^{2\alpha r} (g(r) + g(n - r)) \leq \exp(cn^{\frac{2}{3}} \log^3 n).$$

Combining with (6.8) we obtain the required bound  $g(n) \leq \exp(cn^{\frac{2}{3}} \log^3 n)$ . □

## 6.4 Conclusions

We proposed a new deterministic approximation algorithm for computing a permanent of an  $n$  by  $n$   $0, 1$  matrix. Our algorithm provides a multiplicative approximation factor  $(1 + \epsilon)^n$  and runs in polynomial time for a matrix corresponding to a constant degree expander, and in time  $\exp(O(n^{\frac{2}{3}} \log^3))$  for an arbitrary matrix. It is natural to try to extend our results in several directions. One possibility is lifting  $0, 1$  requirement. This entails extending the result of Chapter 5 to the case of weighted graphs. It is reasonable to expect that for some special cases such an approach will work. In this case one can try to extend the results of the present chapter to the case of general matrix entries.

A second interesting direction is utilizing the ideas in [JSV04], developed in the Markov chain setting, as a possibility of getting stronger deterministic approximation algorithms for computing a permanent. The technique developed in [JSV04] allow one to deal with the case when the ratio  $M(n - 1)/M(n)$  is exponentially large. Since obtaining amenable bounds on the ratios  $M(k)/M(n)$  was required for our algorithms to work, this might be indeed a fruitful direction for further research.

# Bibliography

- [AAF<sup>+</sup>96] M. Andrews, B. Awerbuch, A. Fernandez, Jon Kleinberg, T. Leighton, and Z. Liu, *Universal stability results for greedy contention-resolution protocols*, Proc. 27th IEEE Symposium on Foundations of Computer Science (1996), 380–389.
- [AKOR98] W. Aiello, E. Kushilevitz, R. Ostrovsky, and A. Rosen, *Adaptive packet routing for bursty adversarial traffic*, Proc. 30th Ann. ACM Symposium on the Theory of Computing (1998).
- [And00] M. Andrews, *Instability of FIFO in session-oriented networks*, Proc. 11th ACM-SIAM Symposium on Discrete Algorithms (2000).
- [AZ00] M. Andrews and L. Zhang, *The effects of temporary sessions on network performance*, Proc. 11th ACM-SIAM Symposium on Discrete Algorithms (2000).
- [BBK<sup>+</sup>01] V. D. Blondel, O. Bournez, P. Koiran, C. H. Papadimitriou, and J. N. Tsitsiklis, *Deciding stability and mortality of piecewise affine systems*, Theoretical Computer Science **225** (2001), no. 1-2, 687–696.
- [BG03] R. Bhattacharjee and A. Goel, *Instability of FIFO at arbitrarily low rates in the adversarial queueing model*, Proc. 44th IEEE Symposium on Foundations of Computer Science, 2003.
- [BG06] A. Bandyopadhyay and D. Gamarnik, *Counting without sampling. New algorithms for enumeration problems using statistical physics.*, Proceedings of 17th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2006.
- [BGT96] D. Bertsimas, D. Gamarnik, and J. Tsitsiklis, *Stability conditions for multiclass fluid queueing networks*, IEEE Trans. Automat. Control **41** (1996), 1618–1631.

- [BGT01] ———, *Performance of multiclass Markovian queueing networks via piecewise linear Lyapunov functions*, Ann. of Appl. Prob. **11** (2001), no. 4, 1384–1428.
- [BKMP01] N. Berger, C. Kenyon, E. Mossel, and Y. Peres, *Glauber dynamics on trees and hyperbolic graphs*, Proc. 42nd IEEE Symposium on Foundations of Computer Science (2001).
- [BKR<sup>+</sup>01] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. Williamson, *Adversarial queueing theory*, Journal of ACM **48** (2001), 13–38.
- [BM01] T. Bonald and L. Massoulié, *Impact of fairness on internet performance*, Proceedings of ACM Sigmetrics, 2001.
- [BN06] M. Bayati and C. Nair, *A rigorous proof of cavity method for counting matchings*, Annual Allerton Conference on Communication, Control and Computing, 2006.
- [BNM99] D. Bertsimas and J. Nino-Mora, *Optimization of multiclass queueing networks with changeover times via the achievable region method: Part I, the single-station case*, Mathematics of Operations Research **24** (1999), 306–330.
- [BPT94] D. Bertsimas, I. Paschalidis, and J. Tsitsiklis, *Optimization of multiclass queueing networks: Polyhedral and nonlinear characterization of achievable performance*, The Annals of Applied Probability **4** (1994), 43–75.
- [Bra94] M. Bramson, *Instability of FIFO queueing networks*, Ann. Appl. Probab. **2** (1994), 414–431.
- [Bra99] ———, *A stable queueing network with unstable fluid model*, Ann. Appl. Probab. **9** (1999), no. 3, 818–853.
- [Bra01] M. Bramson, *Stability of earliest-due-date first-served queueing networks*, Queueing Syst. **39** (2001), no. 1, 79–102.
- [Bra05] M. Bramson, *Stability of networks for max-min fair routing*, Presentation at the 13th INFORMS Applied Probability Conference (2005).
- [BT00a] V. D. Blondel and J. N. Tsitsiklis, *The boundedness of all products of a pair of matrices is undecidable*, Systems and control letters **41** (2000), no. 2, 135–140.



- [BT00b] ———, *A survey of computational complexity results in systems and control*, *Automatica* **36** (2000), no. 9, 1249–1274.
- [BW02] G. Brightwell and P. Winkler, *Random colorings of a Cayley tree*, in *Contemporary Combinatorics*, B. Bollobas, ed., Bolyai Society Mathematical Studies, 2002, pp. 247–276.
- [BW04] ———, *Graph homomorphisms and long range action*, in *Graphs, Morphisms and Statistical Physics* (Nesetril and Winkler eds.), DIMACS series in discrete mathematics and computer science, 2004, pp. 29–47.
- [CST] M. Chiang, D. Shah, and A. Tang, *Stochastic stability under network utility maximization: General file size distribution*, Preprint.
- [CY01] H. Chen and D. Yao, *Fundamentals of queueing networks: Performance, asymptotics and optimization*, Springer-Verlag, 2001.
- [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.
- [Dai96] ———, *A fluid-limit model criterion for instability of multiclass queueing networks*, *Ann. Appl. Probab.* **6** (1996), 751–757.
- [DFHV04] M. Dyer, A. Frieze, T. Hayes, and E. Vigoda, *Randomly coloring constant degree graphs*, in *Proceedings of 45th IEEE Symposium on Foundations of Computer Science*, 2004.
- [DHV99] J. G. Dai, J. J. Hasenbein, and J. H. Vande Vate, *Stability of a three-station fluid network*, *Queueing Systems* **33** (1999), 293–325.
- [DM95] D. D. Down and S. P. Meyn, *Stability of acyclic multiclass queueing networks*, *IEEE Trans. Automat. Control* **40** (1995), no. 5, 916–920.
- [DM97] ———, *Piecewise linear test functions for stability and instability of queueing networks*, *Queueing Systems* **27** (1997), 205–226.
- [DSVW04] M. Dyer, A. Sinclair, E. Vigoda, and D. Weitz, *Mixing in time and space for lattice spin systems: a combinatorial view*, *Random Struct. & Alg.* **24** (2004), 461–479.

- [DV00] J. G. Dai and J. H. Vande Vate, *The stability of two-station multi-type fluid networks*, Operations Research **48** (2000), 721–744.
- [dVLK01] G. de Veciana, T. Lee, and T. Konstantopoulos, *Stability and performance analysis of networks supporting elastic services*, IEEE/ACM Transactions on Networking **9** (2001), no. 1, 2–14.
- [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.
- [FV06] A. Frieze and E. Vigoda, *Survey of Markov chains for randomly sampling colorings*, To appear in Festschrift for Dominic Welsh (2006).
- [Gam00] D. Gamarnik, *Using fluid models to prove stability of adversarial queueing networks*, IEEE Transactions on Automatic Control. (Conference version in FOCS98.) **4** (2000), 741–747.
- [Gam02] ———, *On deciding stability of constrained homogeneous random walks and queueing systems*, Mathematics of Operations Research **27** (2002), no. 2, 272–293.
- [Gam03] ———, *Stability of adaptive and non-adaptive packet routing policies in adversarial queueing networks*, SIAM Journal on Computing. (Conference version in STOC99.) (2003), 371–385.
- [GH05] D. Gamarnik and J. Hasenbein, *Instability in stochastic and fluid queueing networks*, Ann. Appl. Probab. **15** (2005), no. 3, 1652–1690.
- [GMP05] L. A. Goldberg, R. Martin, and M. Paterson, *Strong spatial mixing with fewer colors for lattice graphs*, SIAM J. Comput. **35** (2005), no. 2, 486–517.
- [God81] C. D. Godsil, *Matchings and walks in graphs*, J. Graph Th. **5** (1981), 285–297.
- [Goe99] A. Goel, *Stability of networks and protocols in the adversarial queueing model for packet routing*, Proc. 10th ACM-SIAM Symposium on Discrete Algorithms (1999).

- [Gur06] L. Gurvits, *Hyperbolic polynomials approach to Van der Waerden/Schrijver-Valiant like conjectures: sharper bounds, simpler proofs and algorithmic applications*, Proceedings of the thirty-eighth annual ACM symposium on Theory of computing, 2006.
- [GW06] H. C. Gromoll and R. Williams, *Fluid limit of a network with fair bandwidth sharing and general document size distribution*, Preprint (2006).
- [Hoo66] P. Hooper, *The undecidability of the Turing machine immortality problem*, The Journal of Symbolic Logic **2** (1966), 219–234.
- [HU69] J. Hopcroft and J. Ullman, *Formal languages and their relation to automata*, Addison-Wesley. Boston, MA, 1969.
- [Jer95] M. Jerrum, *A very simple algorithm for estimating the number of  $k$ -colourings of a low-degree graph*, Random Structures and Algorithms **7** (1995), no. 2, 157–165.
- [Jon02] J. Jonasson, *Uniqueness of uniform random colorings of regular trees*, Statistics and Probability Letters **57** (2002), 243–248.
- [JS89] M. Jerrum and A. Sinclair, *Approximating the permanent*, SIAM journal on computing **18** (1989), 1149–1178.
- [JS97] ———, *The Markov chain Monte Carlo method: an approach to approximate counting and integration*, Approximation algorithms for NP-hard problems (D. Hochbaum, ed.), PWS Publishing Company, Boston, MA, 1997.
- [JS06] K. Jung and D. Shah, *On correctness of belief propagation algorithm*, Preprint (2006).
- [JSV04] M. Jerrum, A. Sinclair, and E. Vigoda, *A polynomial-time approximation algorithms for permanent of a matrix with non-negative entries*, Journal of the Association for Computing Machinery **51** (2004), no. 4, 671–697.
- [JV96] M. Jerrum and V. Vazirani, *A mildly exponential approximation algorithm for the permanent*, Algorithmica **16** (1996), no. 4-5, 392–401.

- [KK94] S. Kumar and P. R. Kumar, *Performance bounds for queueing networks and scheduling policies*, IEEE Transactions on Automatic Control **8** (1994), 1600–1611.
- [KK98] J. Kahn and J. H. Kim, *Random matchings in regular graphs*, Combinatorica **8** (1998), 201–226.
- [KK01] S. Kumar and P. R. Kumar, *Queueing network models in the design and analysis of semiconductor wafer fabs*, IEEE Transactions on Robotics and Automation **17** (2001), no. 5, 548–561.
- [KM04] P. R. Kumar and J. Morrison, *New linear program performance bounds for queueing networks*, Journal of Optimization Theory and Applications (2004), 575–597.
- [KS90] P. R. Kumar and T. I. Seidman., *Dynamic instabilities and stabilization methods in distributed real-time scheduling of manufacturing systems*, IEEE Trans. Automat. Control **AC-35** (1990), 289–298.
- [Kuh55] H. Kuhn, *The hungarian method for the assignment problem*, Naval Research Logistic Quarterly (1955), no. 2, 83–97.
- [LK91] S. H. Lu and P. R. Kumar, *Distributed scheduling based on due dates and buffer priorities*, IEEE Trans. Automat. Control **36** (1991), 1406–1416.
- [LPSR02] Z. Lotker, B. Patt-Shamir, and A. Rosen, *New stability results for adversarial queueing*, SIAM Journal on Computing **33** (2002), no. 2, 286–303.
- [LS04] X. Lin and N. Shroff, *On the stability region of congestion control*, Proceedings of Allerton Conference, 2004.
- [LSW00] N. Linial, A. Samorodnitsky, and A. Wigderson, *A deterministic strongly polynomial algorithm for matrix scaling and approximate permanents*, Combinatorica **20** (2000), no. 4, 545–568.
- [Mat93] Y. Matiyasevich, *Hilbert’s tenth problem*, Nauka Publishers. English translation published by the MIT Press, 1993.
- [Mey95] S. P. Meyn, *Transience of multiclass queueing networks and their fluid models*, Ann. of Appl. Prob. **5** (1995), 946–957.

- [MS06] A. Montanari and G. Semerjian, *Rigorous inequalities between length and time scales in glassy systems*, Preprint in arXiv.org (2006).
- [MT93] S. P. Meyn and R. L. Tweedie, *Markov chains and stochastic stability*, Springer-Verlag. London, UK., 1993.
- [NT] C. Nair and P. Tetali, *The correlation decay (CD) tree and strong spatial mixing in multi-spin systems*, Preprint on <http://front.math.ucdavis.edu/math.PR/0701494>.
- [OE72] O.J.Heilman and E.H.Lieb, *Theory of monomer-dimer systems*, Comm. Math. Phys. **25** (1972), 190–232.
- [PR00] A.A. Puhalskii and A.N. Rybko, *Nonergodicity of a queueing network under nonstability of its fluid model*, Problems of Information Transmission **36** (2000), 26–41.
- [RM00] J. Roberts and L. Massoulié, *Bandwidth sharing and admission control for elastic traffic*, Telecommunication Systems **15** (2000), 185–201.
- [Ros02] A. Rosen, *A note on models for non-probabilistic analysis of packet-switching networks*, Information Processing Letters **84** (2002), no. 5, 237–240.
- [RS92] A. Rybko and A. Stolyar, *On the ergodicity of stochastic processes describing open queueing networks*, Problemi Peredachi Informatsii **28** (1992), 3–26.
- [Sch98] A. Schrijver, *Counting 1-factors in regular bipartite graphs*, Journal of Combinatorial Theory, Series B **72** (1998), 122135.
- [Sei94] T. I. Seidman, *First come first serve can be unstable*, IEEE Trans. Autom. Control **39** (1994), 2166–2170.
- [Sha05] D. Shah, *Max weight independent set and matching via max-product*, Preprint (2005).
- [Sip97] M. Sipser, *Introduction to the theory of computability*, PWS Publishing Company, 1997.
- [Sri04] R. Srikant, *The mathematics of internet congestion control*, Birkhauser, 2004.

- [SS97] J. Salas and A. D. Sokal, *Absence of phase transition for antiferromagnetic Potts models via the Dobrushin uniqueness theorem*, J. Statist. Phys. **86** (1997), no. 3-4, 551–579.
- [Sto95] A. Stolyar, *On the stability of multiclass queueing networks: A relaxed sufficient condition via limiting fluid processes*, Markov Processes and Related Fields (1995), 491–512.
- [TJ02] S. Tatikonda and M. I. Jordan, *Loopy belief propagation and Gibbs measures*, In Uncertainty in Artificial Intelligence (UAI), D. Koller and A. Darwiche (Eds)., 2002.
- [Tsa97] P. Tsaparas, *Stability in adversarial queueing theory*, M.Sc. Thesis, University of Toronto, 1997.
- [Vad01] S. Vadhan, *The complexity of counting in sparse, regular, and planar graphs*, SIAM Journal on Computing (2001), no. 2, 398–427.
- [Val] L. G. Valiant.
- [Val79] ———, *The complexity of computing the permanent*, Theoretical Computer Science **8** (1979), 189–201.
- [vdB98] J. van den Berg, *On the absence of phase transition in the monomer-dimer model*, CWI reports, PNA-R9813, ISSN 1386-3711 (1998).
- [Vig00] E. Vigoda, *Improved bounds for sampling colorings*, Journal of Mathematical Physics (2000).
- [Wei06] D. Weitz, *Counting independent sets up to the tree threshold*, Proc. 38th Ann. Symposium on the Theory of Computing (2006).
- [YFW00] J. Yedidia, W. Freeman, and Y. Weiss, *Understanding Belief Propagation and its generalizations*, Mitsubishi Elect. Res. Lab. (2000), no. TR-2001-22.