Small-Depth Counting Networks and Related Topics

by

Michael Richard Klugerman

B.S. Mathematics and Computer Science
Yale University (1986)

Submitted to the Department of Mathematics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1994

Author . . . .
Department of Mathematics
June 15, 1994

Certified by . . . .
Frank Thomson Leighton
Professor of Applied Mathematics
Thesis Supervisor

Accepted by . . . .
David Vogan
Chairman, Departmental Graduate Committee

Small-Depth Counting Networks and Related Topics

by

Michael Richard Klugerman

B.S. Mathematics and Computer Science

Yale University (1986)

Submitted to the Department of Mathematics

on June 15, 1994, in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

Abstract

In [5], Aspnes, Herlihy, and Shavit generalized the notion of a sorting network by introducing a class of so called "counting" networks and establishing an $O(\lg^2 n)$ upper bound on the depth complexity of such networks. Their work was motivated by a number of practical applications arising in the domain of asynchronous shared memory machines. In this thesis, we continue the analysis of counting networks and produce a number of new upper bounds on their depths. Our results are predicated on the rich combinatorial structure which counting networks possess. In particular, we present a simple explicit construction of an $O(\lg n \lg\lg n)$-depth counting network, a randomized construction of an $O(\lg n)$-depth network (which works with extremely high probability), and we present an existential proof of a deterministic $O(\lg n)$-depth network. The latter result matches the trivial $\Omega(\lg n)$-depth lower bound to within a constant factor. Our main result is a uniform polynomial-time construction of an $O(\lg n)$-depth counting network which depends heavily on the existential result, but makes use of extractor functions introduced in [25]. Using the extractor, we construct regular high degree bipartite graphs with extremely strong expansion properties. We believe this result is of independent interest.

Thesis Supervisor: Frank Thomson Leighton

Title: Professor of Applied Mathematics

Acknowledgments

There are many people I would like to thank for giving me their friendship and support during my stay at MIT. Among others, these include people from the vast ultimate community in the Boston area and the theory of computation group here at MIT.

I would especially like to thank Tom Leighton, Mauricio Karchmer, and Michel Goemans who served on my committee but more importantly were friends who helped me in my technical development and who were generous with their time and their support. Greg Plaxton, with whom I did a good deal of my thesis work, was equally supportive, helpful and brilliant.

My officemates Stephen Ponzio and Drew Sutherland are friends who were instrumental in helping me maintain the right balance between sanity and insanity during my hours spent at the lab and for that I am very thankful.

Be Hubbard has been a great source of motherly love for the five years I have spent here. Her smile has brightened even the darkest days. Thanks, Be.

I thank my mother, father, and sister for making this possible and encouraging me to make the most of myself.

Most of all, I thank Kristin Wulfsberg. She has helped me in too many ways to enumerate. No one could ask for a better friend.

This research was supported in part by AFOSR Grant F49620-92-J-0125 and ARPA Grants N00014-91-J-1698 and N00014-92-1799.

Contents

Chapter 1

Introduction

One of the fundamental tools used in parallel computation is the *shared counter* [13, 15, 17, 28]. A shared counter permits several processors to request and increment the value of a shared variable. If a counter's value begins at 0 and r requests are made by various processors, a shared counter returns r distinct values between 0 and $r-1$ inclusive to the requesting processors. Thus, each request is fulfilled with a distinct value and precisely the first r values are returned, leaving no unused values.

One solution to the problem of implementing a shared counter is to use a single shared Fetch-and-Increment variable which is incremented each time a processor makes a request. Because a large number of processors may be attempting to access this single variable at the same time, this can lead to high memory contention. In order to avoid this sequential bottleneck, Aspnes, Herlihy, and Shavit [5] introduced the concept of a "counting network", and they showed that such networks can be simulated efficiently on an asynchronous shared memory machine as a means of implementing shared counters. By significantly reducing the memory contention, counting networks allow for a much higher degree of concurrency. More specifically, a number of shared variables are used to implement a single counter in such a way that a processor incrementing the counter need only access a small number of memory locations and not all processors making requests access the same set of variables, thus providing fast response time and high throughput.

The elementary building block of a counting network is a *balancer* which is a 2-

input, 2-output device similar to a comparator. Whereas a comparator receives a number along each input wire and outputs them in sorted order, a balancer receives multiple anonymous tokens along both input wires in an asynchronous fashion and sends the tokens alternately to the upper and lower output wires. Thus, if a total of m tokens are received by the two input wires of a particular balancer, that balancer emits $\lceil m/2 \rceil$ tokens via the top output wire and $\lfloor m/2 \rfloor$ tokens via the bottom output wire (see Figure 1-1). A balancer is in its *initial state* if the next token it outputs will leave the balancer along its top output wire. Thus, a balancer is in its initial state if an even number of tokens have passed through it.

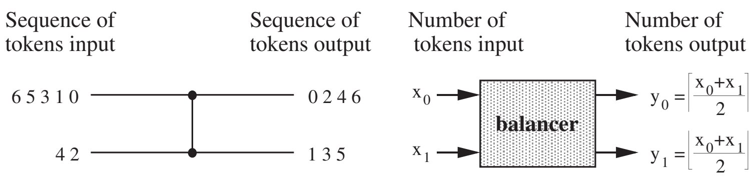

Figure 1-1: A balancer.

A *balancing network* is an acyclic circuit made up of balancers, just as a comparator network is an acyclic circuit made up of comparators. Similarly, a counting network is a balancing network with additional properties in the same way that a sorting network [3, 6, 9, 23] is a comparator network with additional properties. An n-input balancing network has n input wires and n output wires. We say that such a network has *width* n. In the remainder of this thesis we will refer to the input (resp., output) wires as $x_0, \ldots, x_{n-1}$ (resp., $y_0, \ldots, y_{n-1}$). We also use these values to denote the number of tokens input (resp. output) on these wires.

Each token enters the network through some input wire and then passes through one or more balancers before arriving at an output wire. Figure 1-2 is an example of a balancing network.

We now define two special types of balancing networks that guarantee certain properties concerning the nature of the output.

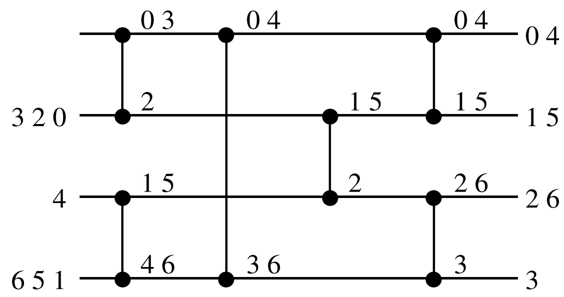

Figure 1-2: A sequential execution of a counting (balancing) network. Tokens are labelled with distinct numbers.

An n-input balancing network k-smooths if for any input sequence, $|y_i - y_j| \leq k$, $0 \leq i < j < n$. The output of such a network is said to be *smoothed to within k*.

An n-input network is a *smoothing network* if it 1-smooths. The output of such a network is said to have a *smoothed shape*.

An n-input *counting network* is a balancing network such that for any input sequence, $1 \geq y_i - y_j \geq 0$, $0 \leq i < j < n$. Note that a counting network is a restricted form of smoothing network. The output of such a network is said to have a *counted shape*.

In our constructions we will find it useful to consider balancing networks that count certain restricted classes of input sequences. An input sequence $x_0, \ldots, x_{n-1}$ is *k-smoothed* if $|x_i - x_j| \leq k$, $0 \leq i < j < n$. A *k-smoother* is a balancing network that smooths any k-smoothed input sequence. A *k-counter* is a balancing network that counts any k-smoothed input sequence.

It is obvious from the definition of a counting network that assigning the value $in + j$ to the i^{th} token (counting from 0) output from wire j (counting from 0) of a counting network gives the tokens distinct integer values ranging from 0 to $p - 1$ where p is the total number of tokens input to the network. Thus, the tokens are said to be "counted". Figure 1-2 shows a sequence of tokens passing through a counting network. Note that the output has a counted shape.

Counting networks are implemented in software, rather than in hardware. Specifically, each balancer in a network is represented by three memory locations. Because each balancer on the network generally outputs to two other balancers, the first two memory locations are used to identify the locations of the two following balancers. The third location is used to specify the state of the balancer indicating to which of the two balancers the next token will be sent. This last location is represented with a single bit that acts as a toggle mechanism. As each token passes through a balancer the state of the toggle bit is flipped. The exceptions to this representation are the balancers with output wires which are output wires of the network. These balancers have a location specifying the number of tokens which have left along the output wire and the number of the wire so that the value assigned to the token can easily be computed.

In the shared memory model, a processor wishing to increment a counter begins with the address of a balancer at depth 1 of a counting network. The processor then moves the token through the network obtaining the value of the toggle bit and flipping the bit by using a Test&Set operation. By obtaining the value of the toggle bit, the processor obtains the address of the next balancer in the network that the token is to pass through. The processor repeats this process until the token is "output" on an output wire. At this point the processor obtains the value of the counter by determining the height of the output wire from the top (say j) and the number of tokens that have previously been output along this wire (say i). The value of the counter is then $in + j$ where the network is an n-input n-output network.

Additional uses of counting networks, as well as other practical issues such as implementation and simulations of counting networks are discussed in [5] and [18]. In [5], the authors describe a number of data structures which are of great importance to parallel and distributed computing which may be implemented with the use of counting networks. In particular, they discuss producer/consumer buffers, and synchronization barriers.

In a producer/consumer buffer, processors are producing tasks which need to be acted upon while others are taking these tasks and performing the work that needs to be done. The producer/consumer buffer deals with the problem of distributing the tasks so that the processors looking for work to do can find it quickly. In essence, it is a problem in load balancing.

In the case of barrier synchronization, processors are performing tasks in an asynchronous environment. However, when processors reach a certain point in their computation, they are required to wait for other processors to reach a certain point before continuing to their next phase. Counting networks provide a data structure which allows this synchronization to take place efficiently.

Two useful measures of the complexity of a balancing network, and thus a counting network, are its size and depth. The *size* of a balancing network is the number of balancers in the network. The *depth* of a balancing network is the maximum number of balancers a token may be required to pass through when moving from an input wire to an output wire. More formally, we first define the depth of each balancer and wire in the network. The input wires of a balancing network have depth 0. Given this, and because a balancing network is required to be acyclic, the following pair of rules can be used to determine the depth of all balancers and all remaining wires in the network: (i) the depth of a balancer is 1 greater than the maximum depth of its two input wires, and (ii) the depth of an output wire of a given balancer is equal to the depth of that balancer. The *depth* of a balancing network may then be defined as the maximum depth of any output wire in the network. Because the depth of the network is the maximum number of balancers that a token may have to travel through to leave the network depth is a lower bound on the latency of such a network.

In this thesis, we focus on several constructions of counting networks with small depth. Though small depth is important for practical reasons, the focus in this thesis is on the combinatorial nature of these networks. These networks have a rich mathematical structure which we explore in order to obtain our results.

In the original paper on counting networks, Aspnes, Herlihy, and Shavit [5] provide two $O(\lg^2 n)$-depth families of n-input counting networks by proving that the balancing network isomorphic to Batcher's bitonic sorting network [6, 9, 23] and isomorphic to the balanced periodic sorting network of Dowd, Perl, Rudolph, and Saks [10] are counting networks.

In [22] Klugerman presents an $O(\lg n \lg \lg n)$-depth counting network construction.

We present this construction in this thesis. This result has great simplicity and displays some of the concepts used in later constructions.

The main result in this thesis is Klugerman's uniform polynomial-time construction of an $O(\lg n)$-depth counting network. A slightly weaker result presented by Klugerman and Plaxton in [21] provides an existential proof for such a network. Our result answers the question posed in [5], which asks whether such an optimal-depth counting network exists. The technique used in order to obtain the existential result involves constructing a set of networks $\mathcal{N}^*$ such that for any fixed input sequence I, if a network $\mathcal{N}$ is chosen uniformly at random from $\mathcal{N}^*$, then $\mathcal{N}$ will count I with extremely high probability. "Good" networks are then chosen non-uniformly from $\mathcal{N}^*$ and are used to construct a deterministic counting network with logarithmic depth. A similar technique has recently been used by Ajtai, Komlós and Szemerédi [8] to improve the constant factor in their $O(\lg n)$-depth sorting network, and by Plaxton [26] in order to obtain a $2^{O(\sqrt{\lg \lg n})} \lg n$-depth sorting network from an $O(\lg n)$-depth random sorting network [24] that sorts with extremely high probability. This existential result is presented in this thesis and is built upon in order to obtain the uniform polynomial-time construction.

The other result in [21] is an explicit construction of a counting network of depth $O(c^{\lg^* n} \lg n)$ (for some positive constant c). However, this construction is superseded by the constructive proof for an $O(\lg n)$-depth network presented in this thesis which uses many aspects of the existential proof, but makes use of *extractors* constructed in [25]. These extractors are functions which extract a great deal of randomness from a source with limited randomness by using a small number of truly random bits. These extractors have been used to show that randomized *space*$(S(n))$ using only *poly*$(S(n))$ random bits can be simulated deterministically in *space*$(S(n))$, for $S(n) \geq \lg n$ [25]. In addition, the extractor function has been used to construct high degree expanders in polynomial-time [29]. Using techniques similar to those found in [29], we construct regular, high degree bipartite graphs with the expansion properties necessary to obtain an $O(\lg n)$-depth counting network. In essence, this bipartite expander graph allows us to find the desired network in $\mathcal{N}^*$ deterministically in polynomial-time. We believe that the bipartite graph constructed is of independent interest.

Discussions in this thesis include relationships between sorting and counting. As we discuss in Chapter 2, the ability to count depends, in part, on the ability to sort. The $O(\lg n)$-depth and $O(\lg n \lg\lg n)$-depth constructions both make use of the $O(\lg n)$-depth AKS sorting network construction [3]. Unfortunately, the constant in the Big-Oh of the AKS construction is extremely large. As a result, the constants in the counting network constructions are quite large, as well. With the dependence of counting on sorting, one cannot hope to build an $O(\lg n)$-depth counting network with small constants without an improvement in the construction of sorting networks. However, smoothing networks, which are somewhat weaker, are not so clearly dependent on sorting. There is hope that these weaker networks can be constructed without depending so heavily on sorting. This thesis begins to address the question of how much an $O(\lg n)$-depth network can smooth its input without using such powerful sorting tools as AKS. We present a network which $O(\lg\lg n)$-smooths any input. This construction is based on the construction of the optimal-depth counting network but does not use the AKS sorting network as a subroutine. Though this network has weaker properties than either a counting network or a smoothing network, it may provide insight into future constructions.

In addition to constructions of counting networks using 2-input 2-output balancers, we discuss constructions of counting networks with balancers having more inputs and outputs. This model was introduced by Aharonson and Attiya [1]. In [1], the authors discuss limitations on the number of input and output wires a counting network may have in this generalized model. In independent work, Klugerman [22] shows that any counting networks comprised of 2-input 2-output balancers must contain 2^k input and output wires for some integer k. The approaches used in [22] and [1] are similar and are described in this thesis.

The remainder of this thesis is organized as follows: In Chapter 2, we provide lemmas about balancing and counting networks that will be of use in later chapters. In Chapter 3, we discuss negative results and lower bounds pertaining to counting networks. Chapter 4 contains constructions of simple small-depth counting networks. Sections 4.1 and 4.2 describe the $O(\lg^2 n)$-depth counting networks presented in [5]. In Section 4.3 we describe the 2-smoother, a tool used to aid in the construction of the small-depth counting networks described in Section 4.4 and Chapter 5. Section 4.4 contains the construction and analysis of our $O(\lg n \lg\lg n)$-depth counting network. Section 4.5 contains the construction of a network for general n under a more general counting network model. Chapter 5 contains the main results in this thesis, namely the construction of an optimal-depth counting network. Section 5.1 contains more tools used in these constructions. In Section 5.2, we present the $O(\lg n)$-depth random counting network. In Section 5.3, we use the random network to construct a non-uniform deterministic counting network. In Section 5.4 we present the construction of a k-smoother, which is used in Section 5.5. In Section 5.5 we transform the existential proof into a uniform polynomial-time constructive proof. In Section 5.6 we described a small-depth balancing network which smooths all wires to within $O(\lg\lg n)$ of one another without making use of the AKS balancing network. In Chapter 6 we discuss modifications that have been made to the counting network model and other potential solutions to the problem of shared counting. Finally, in Chapter 7 we offer some concluding remarks.

Chapter 2

Basic lemmas

In this section we present some elementary lemmas about balancing networks that will be useful in later proofs.

2.1 Asynchronous vs. Synchronous Balancing Networks

The first lemma shows that given a specific balancing network and an input sequence $x_0, \ldots, x_{n-1}$ to this network, the output sequence $y_0, \ldots, y_{n-1}$ is well-defined. This is a simple extension of the serialization lemma given in [5].

Lemma 2.1.1 *The order in which tokens pass through the network does not affect the number of tokens output on each wire.*

Proof: We prove the claim by induction on the depth of the network. If the depth is 0, then each output wire corresponds to a single input wire and the result is immediate. Now assume that the claim holds for balancing networks of depth k, $k \geq 0$, and consider any maximum-depth balancer x in a network of depth $k+1$. By the induction hypothesis, the number of tokens arriving along each input wire of x is well-defined. Applying the definition of a balancer, we see that the number of tokens received by each of the two outputs of x is also well-defined. ∎

Since we are only concerned with the number of tokens output per wire from the network and not the specific ordering of the tokens, as a result of Lemma 2.1.1, we can choose the order in which we wish tokens to traverse a network when we analyze the properties of a specific network. Note that there is no guarantee, that the i^{th} token input to the network will be output on the "i^{th}" wire.

2.2 Relationships between sorting, smoothing, and counting

The following lemma is stated in [5] and is very useful in our constructions. Given the $O(\lg n)$-depth AKS sorting network result [3], this lemma shows that the problem of constructing a small-depth counting network can be reduced to that of constructing a small-depth smoothing network.

Lemma 2.2.1 *A sorting network (with comparators replaced by balancers) when applied to the output of a smoothing network, produces a counting network.*

∎

Proof: As a consequence of Lemma 2.1.1, we can analyze the output of such a network for a particular input sequence by permitting all tokens to pass through the smoothing network before entering the sorting network. For a particular input sequence, suppose z or $z+1$ tokens are output from each wire of the smoothing network. We then pass z tokens from each output wire of the smoothing network entirely through the sorting network. When all nz of these tokens are output from the network, it is easily shown by induction on the depth of the balancers that there will be z tokens output per wire from all the balancers at any depth k and that these balancers will be in their initial state. All that remains is to pass the remaining 0 or 1 tokens per wire from the smoothing network through the sorting network. When inputs are restricted to 0 or 1 tokens, the balancers act just as comparators would, yielding an output which is counted. ∎

Next, we examine the relationships between counting, smoothing and sorting networks. In this thesis, we make use of sorting networks by replacing the comparators with balancers. We then say that the network with comparators and the network with balancers are *isomorphic* to each other.

Lemma 2.2.2 *Every counting network is isomorphic to some sorting network.*

Proof: By the 0-1 sorting lemma, it suffices to prove that the resulting network sorts any sequence of zeros and ones.

It is clear from inspection that a balancer acts just as a comparator would on the input of zero or one tokens along its input wires. But if the original network is a counting network, then any sequence of 0-1 tokens at the input will result in a sorted sequence of 0-1 tokens at the output. Therefore, this will also be true in the comparison network. ∎

Lemma 2.2.2 states that any counting network is at least as powerful as a sorting network. In out constructions of $O(\lg n)$-depth counting networks, the constants involved in the Big-Oh are quite large due to the use of the $O(\lg n)$-depth AKS sorting network. However, Lemma 2.2.2 states that one cannot hope to construct an $O(\lg n)$-depth counting network with small constants until improvements in constructions of sorting networks are done.

The next lemma shows that, in fact, counting networks are strictly stronger than counting networks.

Lemma 2.2.3 *A sorting network is not necessarily isomorphic to a smoothing (or counting) network.*

Proof: Consider the following n-input sorting network. Connect wires x_0 and x_1 with a balancer. Next, connect x_1 to x_2. Continue this process until x_{n-2} is connected to x_{n-1}. These $n-1$ balancers represent a single state of the sorting network. Repeat the stage $n-1$ times. After the i^{th} stage, the i^{th} largest number is guaranteed to be in the i^{th} wire, so the network sorts. When the network is viewed as a balancing network and $n-i$ tokens are input to wire x_i, then $y_i = x_i$ for all $0 \leq i < n$ (no

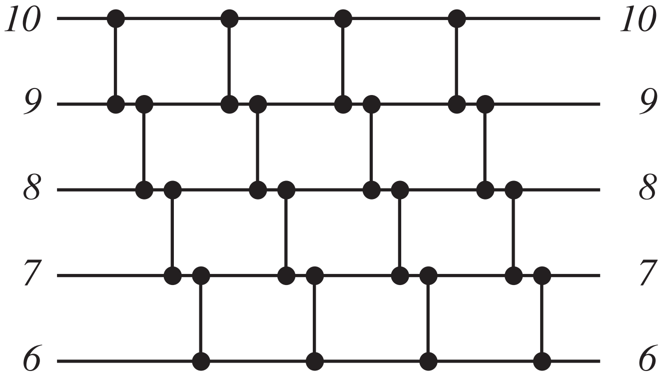

Figure 2-1: A sorter is not necessarily isomorphic to a smoother.

smoothing occurs). We provide an example for $n = 5$ (see Figure 2-1). Indeed, if 10, 9, 8, 7 and 6 tokens enter from top to bottom, then 10, 9, 8, 7 and 6 tokens will be output from the network and no smoothing will occur. ■

Our understanding of the relationship between counting networks and sorting networks far exceeds our understanding of the relationship between smoothing networks and a sorting network. Perhaps, there is no strong connection. What is true is that one does not necessarily imply the other.

Lemma 2.2.4 *All smoothing networks are not necessarily isomorphic to a sorting network.*

Proof: Consider the network illustrated in fig. 2-2. This network is a smoothing network. Indeed, for any set of inputs, the tokens leaving the upper (resp. lower) counting network have a counted shape. The $n/2$ balancers as the end of the network superpose the upper counted shape with the reverse of the lower counted shape. This results in a smoothed shape.

On the other hand, this network is not a sorting network. Treating this network as a comparison network we see that, if the largest element is input on the lower half, it will leave on the top wire of the lower half after going through the lower sorting network and, after going through the rightmost balancer, it will leave the network on the bottom wire of the upper half, rather than the top wire, where it belongs. ■

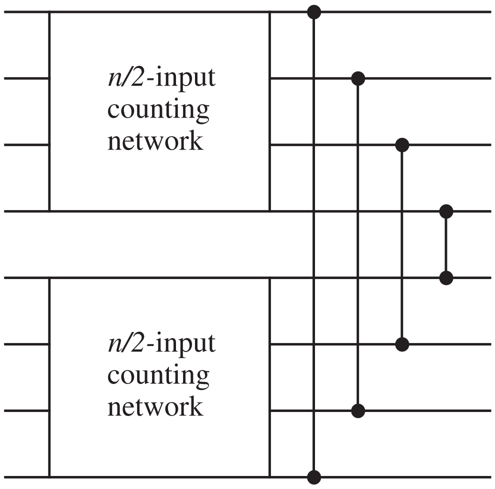

Figure 2-2: A smoother is not necessarily isomorphic to a sorter,

Chapter 3

Impossibility Results and Lower Bounds

3.1 Testing a counting network

We begin by addressing the issue of how to test a network to see if, in fact, it is a counting network. This section provides an attempt to provide a lemma similar to the 0-1 sorting lemma for sorting networks [23]. In [5], the authors provide a method for testing a network by testing the network with a large number of tokens.

Theorem 3.1.1 *[5] A balancing network with b balancers is a counting network iff it counts for all possible inputs of up to 3×2^b tokens.*

Because b will be at least $\Omega(n \lg n)$, the number of inputs which need to be tested is $2^{\Omega(n^2 \lg n)}$ We improve this theorem by making the number of tokens required to test the network exponential in the depth of the network rather than in the size of the network:

Theorem 3.1.2 *[22] A balancing network with depth d is a smoothing (counting) network iff it smooths (counts) on all possible inputs of up to 2^d tokens per wire.*

Because the depth of a network will typically be $O(\lg^c n)$ this means that the number of tests which need to be performed is $2^{O(n \lg^c n)}$, a significant improvement over $2^{\Omega(n^2 \lg n)}$.

To prove theorem 3.1.2, we need the following lemma:

Lemma 3.1.1 *If 2^d tokens are input to any single input wire of a balancing network of depth d, all the balancers in the network remain in their initial state.*

Proof: We show that if a multiple of 2^k tokens are input on a single input wire of a network, then wires at depth D will receive a multiple of 2^{k-D} tokens and the balancers at depth D will be in their initial state at quiescence. Our result then immediately follows for $k = d$ and $D = d$.

Base case: $D = 0$ is immediate.

Inductive step: Assume the hypothesis is true for wires of depth $< D$. Consider a balancer which has as its output a wire of depth D. By induction, this balancer receives a multiple of $2^{k-(D-1)}$ tokens on each of its input wires. Thus, a multiple of 2^{k-D} tokens must be output along both of its output wires. Since the same number of tokens leave each output of a balancer, the balancer remains in its initial state.

■

Theorem 3.1.3 *If 2^d tokens are input into any single input wire of a depth d smoothing network, the same number of tokens are output on each wire.*

Proof: Suppose not. Consider the input wire for which when 2^d tokens are input to that wire, the network outputs a different number of tokens on two output wires. By the preceding lemma, the balancers are in their initial state after all tokens leave the network. Using Lemma 2.1.1 we can input another 2^d tokens into the same input wire. Since the balancers were in their initial states, the gap between the two output wires will double. As a result, the network cannot possibly be a smoothing network.

■

We now return to the proof of theorem 3.1.2.

Proof: Our test is sufficient due to the fact that after 2^d tokens are input to a wire, all balancers are left in their initial state. Suppose the network counts for all inputs with at most 2^d tokens per wire. Consider any input with more than 2^d tokens on some input wires. Suppose the i^{th} input wire has more than 2^d tokens. By Lemma 2.1.1 the order with which we input the tokens does not affect the number of tokens per output wire. Input the first 2^d tokens into wire i. We have tested the network to make sure that this input is counted. By Theorem 3.1.3, the network outputs precisely the same number of tokens on each output wire. By Lemma 3.1.1 the balancers remain in their initial state. We repeat the process of inputting sets of 2^d tokens into individual input wires until no more than 2^d tokens per wire remain on each of these wires. At this point we know the remaining input will be counted by the test we performed. ∎

Note that this testing algorithm requires $O(size \times 2^{nd})$ time.

In [7], a testing algorithm with the same asymptotic testing time is presented. The techniques used in that paper use combinatorial and linear algebra techniques.

3.2 Restriction on number of input wires

In what follows, we show that only counting networks having a number of input wires equal to some integer power of 2 are constructible. This result was independently proved by Klugerman [22] and Aharonson and Attiya [1].

Theorem 3.2.1 *The width of a balancing network must be a power of two in order to be a smoothing network.*

Proof: Consider a balancing network of depth d and width n. By Theorem 3.1.3, the number of tokens output per wire when 2^d tokens are input into a single wire is $p = 2^d/n$. Since p is an integer, the result follows. ∎

In [1], the authors introduce a more general model of balancer networks they call *arbitrary fan-out networks*. Rather than restricting the balancers to be 2-input 2-output devices, they allow balancers of variable size.

Definition 3.2.1 *A b-balancer, is a b-input, b-output device, which outputs the i^{th} token received on the i mod b wire.*

Thus, if a total of k tokens enter a b-balancer, then $\lceil k/b \rceil$ tokens are output on the top $k \bmod b$ wires and $\lfloor k/b \rfloor$ tokens are output on the bottom $k - (k \bmod b)$ wires (see Figure 3-1). In their paper, the authors consider the case where a network is constructed from balancers of sizes taken from a set of integers B. They prove the following impossibility result:

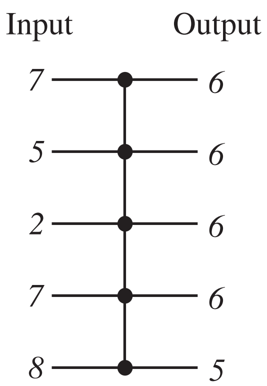

Figure 3-1: A 5-balancer

Theorem 3.2.2 *If there exists a prime factor of n, p, such that $p \not| b$ for all $b \in B$, then there is no acyclic smoothing network with fan-out n over B.*

Note that Theorem 3.2.1 is simply a special case of this theorem where $B = \{2\}$. We will now provide a proof which is similar in spirit to that provided in [1] but which concentrates on the depth of networks rather than the size. As a result, this proof follows the proof of Theorem 3.2.1 quite closely and thus provides a means for testing networks with arbitrary fan-out.

Proof: (of Theorem 3.2.2). Consider an arbitrary fan-out network of depth d with balancers of sizes chosen from the set B. We enter $\left(\prod_{b_i \in B} b_i\right)^d$ tokens into an arbitrary input wire of this network. We can easily show that the number of tokens output on any wire in the network of depth D is divisible by $\left(\prod_{b_i \in B} b_i\right)^{d-D}$ and that the balancers remain in their initial state. This is easily proved by induction on the depth of the network in precisely the same manner as was done in Lemma 3.1.1. By the same reasoning as the proof of Theorem 3.1.3, if the shape output by this network is counted, then the number of tokens per wire must be the same among all wires. But this means $n \mid \left(\prod_{b_i \in B} b_i\right)^d$, leading immediately to the result. ∎

Thus, one can test such a network by making sure that the network smooths (or counts) when up to $\prod_{b_i \in B} b_i$ tokens are entered into each input wire.

3.3 An $\Omega(\lg n)$-depth lower bound on smoothing and counting

Thus far we have not shown any lower bound on the depth of a smoothing network since there is no clear relationship between smoothing and sorting.

Lemma 3.3.1 *A smoothing network on n inputs has $\Omega(\lg n)$-depth.*

Proof: Each output has to depend on all inputs (otherwise, we could increase the number of inputs at a given wire by an arbitrary large amount without increasing the number of outputs at a given wire.) However, at depth d, a wire depends on at most 2^d inputs. ∎

This is true of 2-smoothers as well.

Lemma 3.3.2 *A 2-smoother on n inputs has $\Omega(\lg n)$-depth.*

Proof: Again each input depends on every other input. Suppose not. Then there are two inputs which are independent of one another. Input 2 tokens along one of these wires and 0 tokens along the other. Input 1 token on the remaining wires. Because these two wires have no effect on one another, the wire with 2 tokens and the one with 0 cannot be smoothed. ∎

Chapter 4

Simple Counting Networks

4.1 A Bitonic Counter

In the original paper introducing counting networks [5], the authors present two small-depth counting networks, the bitonic counting network and the periodic counting network. From a practical perspective these networks are the most efficient counting networks to implement (for any reasonable value of n). In addition, the networks are quite simple and again show the close relationship between sorting and counting.

We now construct the bitonic counting network of depth $\frac{1}{2}\lg n(\lg n+1)$. This counter is isomorphic to the bitonic sorting network [6, 9, 23], also known as the even-odd or Batcher sorting network. Because of the simplicity of the network we provide both a construction and a proof that the network is a counting network.

The bitonic counting network is constructed in two phases, each of which is recursive (see also Figure 4-1).

Phase 1: Recursively apply $n/2$-input bitonic counting networks to both the top $n/2$ input wires and the bottom $n/2$ input wires.

Phase 2: Apply a n-input merger to the output of Phase 1.

The n-input merger is designed to input two counted sequences $x_0, x_1, \ldots, x_{n/2-1}$ and $x'_0, x'_1, \ldots, x'_{n/2-1}$ and output a counted shape. The merger is constructed recursively as follows (see also Figure 4-1):

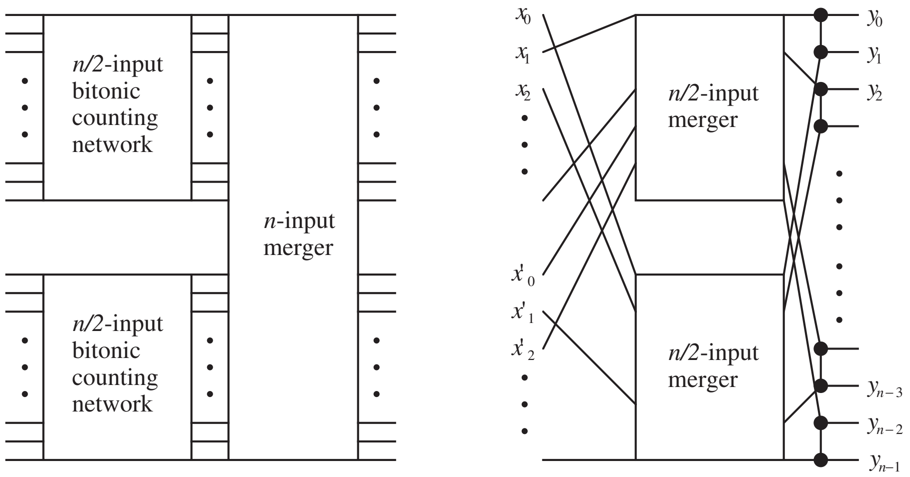

Figure 4-1: To the left: a bitonic counter on n inputs. To the right: a bitonic merger on n inputs.

Phase 1: Apply a $n/2$-input merger to the odd-indexed subsequence $x_1, x_3, \ldots, x_{n/2-1}$ and the even-indexed subsequence $x'_0, x'_2, \ldots, x'_{n/2-2}$. Similarly, apply a $n/2$-input merger to the even-indexed subsequence of x and the odd-indexed subsequence of x'.

Phase 2: Apply a depth one level of balancers. The i^{th} balancer from the top receives, as input, the i^{th} output wire of both mergers from Phase 1.

If $S(n)$ and $M(n)$ are respectively the depth of the counting and merging network, we have $M(n) = M(n/2) + 1 = \lg n$ and $S(n) = S(n/2) + M(n) = S(n/2) + \lg n = \frac{1}{2} \lg n(\lg n + 1)$.

Lemma 4.1.1 *If two counted shapes are input to a merger, then the output will be counted.*

Proof: The two input sequences to each of the mergers in Phase 1 are each counted sequences. So, by induction, the shape output from each merger is counted. To prove that this merger actually merges, we note that

$$\sum_{i=0}^{k/2-1} x_{2i+1} = \lceil S/2 \rceil \text{ and } \sum_{i=0}^{k/2-1} x_{2i} = \lfloor S/2 \rfloor,$$

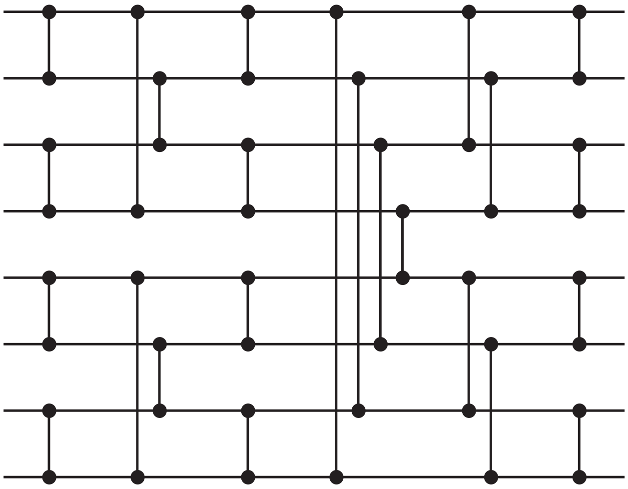

Figure 4-2: A bitonic counter on 8 inputs.

where $S = \sum_{i=0}^{k-1} x_i$. Therefore, if we denote by $z_0, z_1, \ldots, z_{k-1}$ and $z'_0, z'_1, \ldots, z'_{k-1}$ the outputs of the top and bottom merger ($k = n/2$), we have

$$\sum_{i=0}^{k-1} z'_i = \lceil S/2 \rceil + \lfloor S'/2 \rfloor \text{ and } \sum_{i=0}^{k-1} z_i = \lfloor S/2 \rfloor + \lceil S'/2 \rceil, \tag{4.1}$$

where $S' = \sum_{i=0}^{k-1} x'_i$. Eq. 4.1 shows that the sum of the sequences z and z' differs by at most one. Since z and z' have the counted shape by induction on the size of the merger, this implies that z and z' have the same values for all but at most one index i, and they differ at this index by at most one value. The i^{th} balancer in Phase 2 ensures that the final shape is counted. ∎

Theorem 4.1.1 *The bitonic counting network is a counting network.*

Proof: By induction Phase 1 of the construction outputs two counted shapes. By definition of the merger, the final output is counted. ∎

Fig. 4-2 illustrates a bitonic counter on 8 inputs.

4.2 A Periodic Counter

The second counting network that the authors of [5] describe is known as the periodic counting network. This network is isomorphic to the network described by Dowd, Perl, Rudolph, and Saks [10]. In this construction, a depth $\lg n$ block is repeated $\lg n$ times producing a $\lg^2 n$-depth network. In this section we provide a description of the construction and refer the reader to [5] for a proof of its correctness.

One of the basic building blocks for this construction is the *ladder* balancing network. In later sections we examine this object more closely, but for now, we simply define it.

Definition 4.2.1 *For any positive integer n, a $2n$-input* ladder network *is a depth 1 balancing network constructed by connecting x_i to x_{2n-1-i} with a balancer, for $0 \leq i < n$ (as indicated in Figure 4-3).*

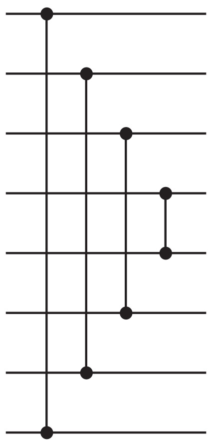

Figure 4-3: An 8-input ladder.

We use the ladder to define a $2n$-input *block* recursively as follows (see also Figure 4-4):

Phase 1: Apply a n-input ladder to all input wires.

Phase 2: Apply $n/2$-input blocks to both the top $n/2$ wires and the bottom $n/2$ wires.

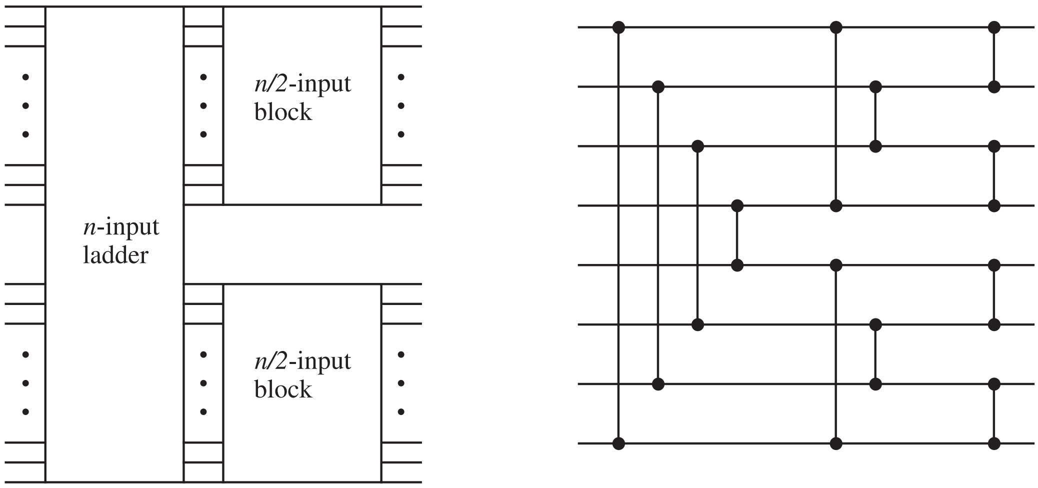

Figure 4-4: On left: a n-input block. On right: an 8-input block.

A n-input periodic counting network is then formed by repeating an n-input block $\lg n$ times (see also Figure 4-5).

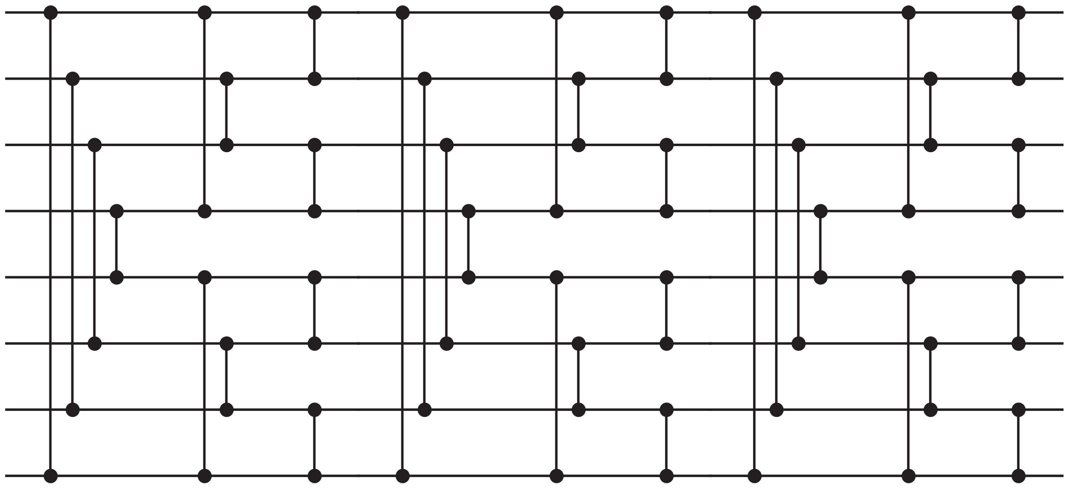

Figure 4-5: An 8-input periodic counting network.

4.3 A deterministic 2-smoother

In this section, we present an explicit construction of a n-input 2-smoother, where n is any positive integer. The 2-smoother is used in the next section in the $O(\lg n \lg\lg n)$-

depth counting network construction and in the optimal-depth construction of Chapter 5. Our construction makes use of two primitives as subroutines: *ladder networks* and *sorting networks*.

4.3.1 Ladder networks

Recall the definition of a ladder from Section 4.2.

The power of the ladder stems from the following lemma:

Lemma 4.3.1 *If a counted shape is input to the top n wires of a $2n$-input ladder, and another counted shape is input to the bottom n wires, then the output of the ladder is smoothed.*

Proof: There exist integers n_0 and k_0 such that n_0 of the top n inputs receive k_0 tokens each, and the remaining $n - n_0$ top inputs receive $k_0 + 1$ tokens each. Let n_1 and k_1 be defined similarly for the bottom inputs. If $n_0 \geq n - n_1$, then every output of the ladder will receive at least $\left\lfloor \frac{k_0+k_1}{2} \right\rfloor$ and at most $\left\lceil \frac{k_0+k_1+1}{2} \right\rceil = \left\lfloor \frac{k_0+k_1}{2} \right\rfloor + 1$ tokens. If $n_0 \leq n - n_1$, then every output of the ladder will receive at least $\left\lfloor \frac{k_0+k_1+1}{2} \right\rfloor$ and at most $\left\lceil \frac{k_0+k_1+2}{2} \right\rceil = \left\lfloor \frac{k_0+k_1+1}{2} \right\rfloor + 1$ tokens. In either case, the output is smoothed. ∎

4.3.2 Sorting networks

We also make use of n-input *sorting networks* in our counting network constructions. Any sorting network may be used at these locations by replacing the comparators with balancers. To obtain small depth we use the $O(\lg n)$-depth AKS sorting network [3]. We refer to the AKS network with balancers as the AKS balancing network.

As we discussed in Chapter 2, when the sorting network is applied to the end of a smoothing network, the resulting network becomes a counting network. We make use of this property in this section. In addition, the sorting network provides another useful property:

Lemma 4.3.2 *If at most k (resp. at least 0) tokens are input into any input wire of a sorting network (with comparators replaced by balancers) and n_k (resp. n_0) input-wires receive k (resp. 0) tokens, then all output-wires containing k (resp. 0) tokens will reside in the top n_k (resp. bottom n_0) wires.*

Proof: We provide a proof by contradiction. Consider an input sequence I for which the property above does not hold. We compare the output from the network with comparators with the output from the network with balancers. Suppose without loss of generality, that an output wire outputs k tokens from the balancing network while it outputs a number smaller than k from the sorting network. Find a balancer b in the network of minimum depth where one of its output wires, say w output k tokens, but the corresponding comparator outputs less than the number k on wire w. There are 3 cases to consider:

1. Both inputs to b were less than k. This case cannot happen because both outputs of b would be less than k.

2. Exactly one input to b is less than k. At most one of b's output wires will contain k tokens and it will be the "larger" output wire. Since b is minimum depth, the corresponding comparator will also receive a k as input, so it is guaranteed to output k on this wire.

3. Both inputs to b contain k tokens. The comparator will receive a k on both its input wires (by minimal depth of b) and so will output k on both its output wires.

So there can be no minimal depth balancer b with the stated property. ∎

4.3.3 Construction of the 2-smoother

Here we consider the case of constructing a k-smoother with $k = 2$. Later we will consider more general k. To simplify the analysis of our constructions we note that we can simply analyze the network for input wires receiving at most k tokens. If the network smooths all such possible inputs, then the network smooths all inputs that are smoothed to within k.

Lemma 4.3.3 *A network that smooths any input sequence with no more than k tokens per wire is a k-smoother.*

Proof: Suppose that a given n-input network $\mathcal{N}$ smooths every input sequence with no more than k tokens per wire, and let a k-smoothed input sequence be input to $\mathcal{N}$. There exists some integer a such that $a \leq x_i \leq a + k$, for $0 \leq i < n$. Using Lemma 2.1.1, we begin by passing all but a tokens from each input wire through the network. By our assumption, a smoothed shape will be produced at the output. Next, we pass the remaining a tokens from each input wire through the network. By a simple induction on the depth of network $\mathcal{N}$, we find that each output wire will receive an additional a tokens as a result of this pass. Hence, the final shape will be smoothed. ∎

We now provide a construction for a $2n$-input 2-smoother. The 3 phase construction (see also Figure 4-6) defined below produces an $O(\lg n)$-depth network.

Phase 1: Apply a $2n$-input sorting network to the $2n$ input wires.

Phase 2: Apply a n-input sorting network to the top n wires and another n-input sorting network to the bottom n wires.

Phase 3: Apply a ladder to all $2n$ wires.

By lemma 4.3.3, it is sufficient to prove that our network counts when each x_i is drawn from $\{0, 1, 2\}$, for $0 \leq i < n$. Fixing a particular input sequence of this type, let n_0, n_1, and n_2 denote the number of wires receiving 0, 1, and 2 tokens, respectively.

Lemma 4.3.4 *After applying the first phase of the construction (i.e., a $2n$-input sorting network), all wires containing 2 tokens are located in the top n_2 wires. Similarly, all wires containing 0 tokens are located in the bottom n_0 wires.*

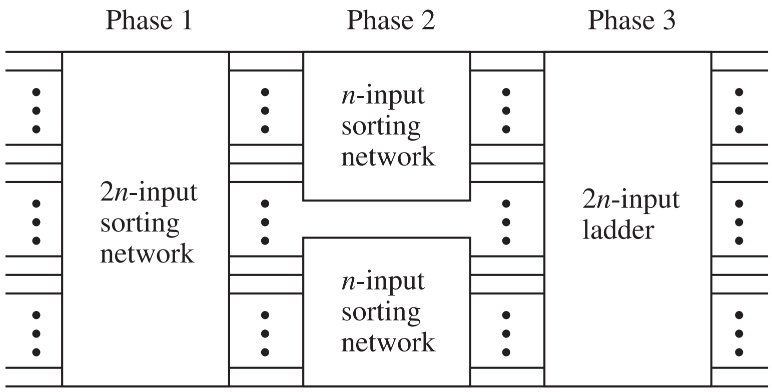

Figure 4-6: A $2n$-input 2-smoother.

Proof: Immediate from lemma 4.3.2. ∎

We now complete the proof that the network described in this section is a 2-smoother by considering two cases.

Case (i): $n_0 \leq n$ and $n_2 \leq n$.

As a result of Lemma 4.3.4, the inputs to each of the two n-input sorting networks in Phase 2 will be smoothed, so the outputs from each of these networks will have a counted shape (Lemma 2.2.1). The two counted shapes are then passed through a ladder, which produces a smoothed output (Lemma 4.3.1).

Case (ii): $n_0 > n$ or $n_2 > n$.

Without loss of generality it may be assumed that $n_2 = n + k$ for some $k > 0$. Arguing as in the proof of Lemma 4.3.4, the output of the first $2n$-input sorting network consists of $n_0 - m$ 0's, m 1_0's, n_1 1's, m 1_2's, and $n_2 - m$ 2's for some nonnegative integer m. Furthermore, the top n wires receive only 2's and 1_2's, so the output of the top n wires is smoothed and at least $n - m$ of these wires receives a 2. At the same time, at most $n-k-m < n-m$ of the bottom n wires receive a 0. Thus, after applying the two n-input sorting networks, the output of the top n wires will be counted and each of the top $n - m$ wires will receive a 2 (Lemma 2.2.1). Furthermore, every 0 must appear on one of the bottom

$n - m$ wires (Lemma 4.3.4). Hence, when the ladder is applied, all existing 0's will be paired with 2's, leaving a smoothed output consisting of 1's and 2's. ■

To obtain an $O(\lg n)$-depth 2-smoother with $2n+1$ (an odd number of) input wires, we apply the three phases described abov to the top $2n$ wires and then perform the following three additional phases (see also Figure 4-7).

Phase 4: Apply a $2n$-input sorting network to the wires from Phase 3.

Phase 5: Apply a balancer to the bottom output wire of Phase 4 and the wire w which has not yet been connected to a balancer.

Phase 6: Apply a balancer to the top output wire of Phase 4 and the low output from the balancer in Phase 5.

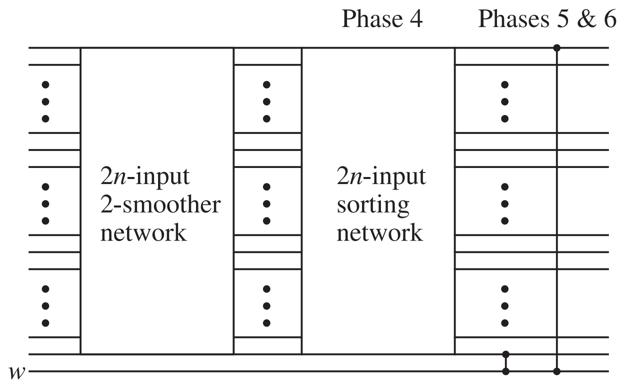

Figure 4-7: A $2n+1$-input 2-smoother.

Lemma 4.3.5 *The six phase network described above is a $2n+1$-input 2-smoother network.*

Proof: The first three phases smooth $2n$ of the wires. Suppose these $2n$ wires each contain either a or $a+1$ tokens at the end of Phase 3. Phase 4 counts these wires by Lemma 2.2.1. Let the number of tokens on these counted wires be either a or $a+1$. We consider all possible numbers of tokens on wire w upon entering Phase 5.

w **contains** a **or** $a+1$ **tokens:** all wires are already smoothed.

w **contains** $a+2$ **tokens:** then the balancer in Phase 5 smooths all the wires.

w **contains** $a-1$ **tokens:** Phase 5 will output $a-1$ tokens on the wire connected to a balancer in Phase 6. This balancer will ensure that all wires are smoothed.

w **contains** $a-2$ **tokens:** This is only possible if all other wires contains a tokens. Thus the balancer in Phase 5 will complete the smoothing.

■

Theorem 4.3.1 *There exists an explicitly constructible family of $O(\lg n)$-depth, n-input 2-smoothers.* ■

4.4 An $O(\lg n \lg\lg n)$-depth counting network

We now present an $O(\lg n \lg\lg n)$-depth counting network construction for all $n = 2^d$, d a positive integer. We make use of the 2-smoother balancing network from Section 4.3. This construction is interesting because it improves on the $(\lg^2 n)$-depth constructions using a very simple approach (given the existence of the AKS balancing network). The $O(\lg n)$-depth construction which is described in a later section is far more complex. In addition, this construction is easily generalizable to the case of arbitrary fan-out balancers described in Section 3.2. This more general network is described in Section 4.5 below.

4.4.1 Construction

The counting network consists of a smoothing network followed by the AKS balancing network. The smoothing network consists of 3 phases (see also Figure 4-8), the first two of which are recursive:

Phase 1:

Assign the n input wires to distinct elements of a $r \times c$ grid, where $r = 2^{\lceil \frac{\lg n}{2} \rceil}$

and $c = 2^{\lfloor \frac{\lg n}{2} \rfloor}$. Recursively apply a smoothing network to each of the rows of this grid.

Phase 2:

Recursively apply a smoothing network to each of the columns. Note: it is not important which input wire is associated with which element of the grid.

Phase 3: Apply a 2-smoother to the outputs of Phase 2.

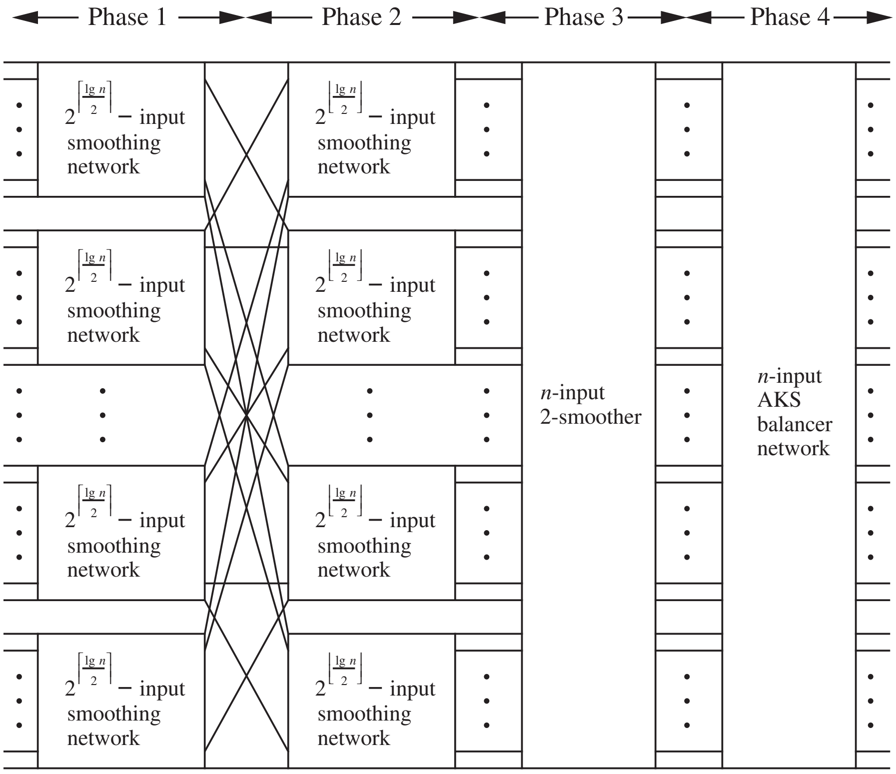

Figure 4-8: An $O(\lg n \lg \lg n)$-depth counting network

4.4.2 Analysis

Let $D_C(n)$ be the depth of the counting network on n input wires. Let $D_S(n)$ be the depth of the smoothing network on n input wires. Then,

$$\begin{aligned} D_S(n) &= D_S(2^{\lceil \frac{\lg n}{2} \rceil}) + D_S(2^{\lfloor \frac{\lg n}{2} \rfloor}) + O(\lg n) \\ &\leq 2D_S(2\sqrt{n}) + O(\lg n) \\ &= O(\lg n \lg\lg n) \end{aligned}$$

Therefore, $D_C(n) = D_S(n) + O(\lg n) = O(\lg n \lg\lg n)$.

4.4.3 Correctness

We now show that after Phase 2 the input is 2-smoothed.

Lemma 4.4.1 *If a total of k tokens enter a smoothing network with n wires, when the network reaches a quiescent state, each wire will contain either $\lceil \frac{k}{n} \rceil$ or $\lfloor \frac{k}{n} \rfloor$ tokens.*

Proof: Immediate from the definition of a smoothing network. ∎

Lemma 4.4.2 *After Phase 2, the difference between the number of tokens on any pair of wires is at most 2.*

Proof: Consider the $r \times c$ grid of wires defined in Phase 1. There exist r_i, $1 \leq i \leq r$ such that after the rows are smoothed, either r_i or $r_i + 1$ tokens are output from each wire in row i. Let $R = \sum_{i=1}^{r} r_i$. Let C_j denote the number of tokens in column j after the rows are smoothed. Then $R \leq C_j \leq R + r$, for all $1 \leq j \leq c$. As a result, the average number of tokens per wire in one column is within one of the average number of tokens per wire in any other column. Thus, after the columns are smoothed, lemma 4.4.1 yields the proof. ∎

Lemma 4.4.3 *After Phase 3 the output is smoothed.*

Proof: Immediate from Lemma 4.4.2 and the definition of a 2-smoother. ∎

Theorem 4.4.1 *The exist polynomial-time constructible $O(\lg n \lg\lg n)$-depth counting networks.*

Proof: We apply the AKS sorting network to all the wires output from the 2-smoother described above and by Lemma 2.2.1 this network becomes a 2-counter. ■

4.5 An arbitrary fan-out counting network

Consider a counting network with n inputs and n outputs where n is no longer a power of 2. Instead, $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$ where each of the p_i are distinct primes. Let the set of balancer sizes available to construct the network be $\{p_1, p_2, \cdots, p_k\}$ (see Section 3.2).

In this section we show how to construct such a network using precisely the same technique as shown in Section 4.4. In [7], the authors present a small depth construction of width $p2^k$ with balancers of size $\{2, p\}$ and they construct a network of width pq^k with balancers of size $\{p, q\}$. In this section we address the more general problem.

Theorem 4.5.1 *If balancers of sizes $\{p_1, p_2, \cdots, p_k\}$ are available. Then an n-input n-output counting network can be constructed where $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$.*

Proof: The construction is recursive. In the base case, $n = p_1$ a prime. Here a p_1-balancer can be applied. In the more general case, $n = m \times l$ where m and l are positive integers greater than 1. Many such factorings may be possible. One can choose that which will minimize the depth of the network using dynamic programming techniques. We treat the n input wires of the network as a m by l grid. First we build smoothing networks on the rows of this grid (m-input m-output smoothers, and then we apply a smoothing network to each of the columns (l-input l-output smoothers). By Lemma 4.4.2, the output will be 2-smoothed.

We are now left to construct an n-input n-output 2-smoother from balancers of size $\{p_1, p_2, \cdots, p_k\}$. To do this we mimick the construction of Section 4.3. If n is even, then $p_i = 2$ for some i and the construction is identical to that of Section 4.3.

If n is odd then we require a small-depth sorting network which makes use of p-comparators rather than comparators. The p-comparator has p inputs and p outputs and it sorts the p inputs. In [8], Chvátal presents a construction of such a network with depth $O(\log_p n)$. We will refer to this network as the pAKS sorting network. When the p-comparators are replaced with p-balancers, the network becomes the pAKS balancing network.

The pAKS balancing network holds similar properties to the AKS balancer network in that it counts a smoothed input (for the same reasons as offered in Lemma 2.2.1) and it possesses the same properties with respect to Lemma 4.3.2 by the same reasoning offered in the proofs of these lemma. In the first phase of the 2-counter we apply the pAKS network to all n wires. In the second phase of the 2-counter we apply the pAKS network to the top $(n-1)/2$-wires and also to the bottom $(n-1)/2$ wires. This leaves one wire which has not yet been involved. However, this middle wire w_m need not be balanced with any other wire as we now argue. Assume without loss of generality that 0,1 and 2 tokens are input per wire to the network. If w_m contains 1 token after the initial pAKS network, then when all other wires are smoothed, the entire network is clearly smoothed. If w_m contains a 0, then by Lemma 4.3.2 there are more than $n/2$ 0's in the network. But this means that once the wires are all smoothed, 0's will still remain. Thus smoothing the other wires will result in a smoothed network. By a symmetric argument, w_m need not be smoothed if it contains 2 tokens.

After the $(n-1)/2$-input pAKS networks we are left to perform the same function as the ladder on $n-1$ wires (all wires excluding w_m). Because 2-balancers are not available we will modify the original ladder construction by replacing the 2-balancers with p-balancers for some p. Note that the original ladder simply smooths pairs of wires. By doing this, the ladder (as proved before) smooths the entire input. We partition the balancers of the ladder into blocks of $(p+1)/2$ balancers. Note that because of issues of divisibility, one block may contain fewer balancers. We now offer a construction which smooths the wires in each block. We replace each block of balancers with 2 p-balancers. The first p-balancer connects the same inputs and outputs as the $(p+1)/2$ 2-balancers leaving out the bottom most of these input output wires. The second p-balancer connects the same inputs and outputs as the $(p+1)/2$ 2-balancers except for the second wire from the top (see Figure 4-9). This smooths the full-sized blocks. Note that it is not critical that the second wire from the top was left out of the second balancers. The wires which must be included are the top most and bottom most wire from the first p-balancer and the bottom wire which has not yet been smoothed. In addition, the remaining $p-3$ input wires should be taken from the same block.

The block that contains fewer than $(p+1)/2$ balancers is replaced with a single p-balancer. This p balancer uses the same input and output wires as those in the block. Because there may be fewer than p wires in this block the remaining wires are chosen arbitrarily from wires taken from the full-sized blocks. This stage occurs after the full-sized block have been smoothed.

Lemma 4.5.1 *The above construction performs the same function as a ladder in the original 2-smoother construction.*

Proof: The input to the ladder is 2-smoothed. Let us assume that there are a, $a+1$, or $a+2$ tokens per wire. Because the ladder smooths the entire input, each block must contain an average number of token which lie between a and $a+1$ or $a+1$ and $a+2$. Note that all block will be in the same on of these two cases. Thus we need only show that each block is smoothed. After the first p-balancer is applied to the top p wires of a block, all wires but the bottom wire in the block are smoothed. These p wires now contain either a and $a+1$ tokens per wire or $a+1$ and $a+2$ tokens per wire. In the former case, if the bottom wire contains a or $a+1$ tokens then the block is smoothed. If the bottom wire contains $a+2$ tokens then the second p-balancer ensures that this wire is paired with a wire containing a tokens, thus averaging both wires so that each contains $a+1$ tokens. In the latter case, if the bottom wire contains $a+1$ or $a+2$ tokens then again the block has been smoothed. Otherwise, the bottom wire contains a tokens. The second p-balancer pairs up this wire with a wire containing $a+2$ tokens again averaging the two so they each contains $a+1$ tokens. Finally, the small block is smoothed with a single p-balancer. Suppose the full-sized blocks produce a or $a+1$ tokens per wire. Then the average number of wires in the small block are between a and $a+1$ tokens per wire, as well. Thus including wires which have a or $a+1$ tokens per wire in the p-balancer will not effect the smoothing of the small block. ■

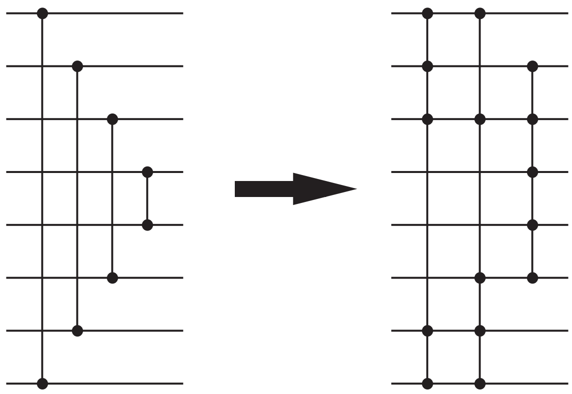

Figure 4-9: Converting ladder with 2-balancers to pladder with 5-balancers.

One way to select m and l (though it may not be optimal) above is to consider the total exponent $\alpha = \sum \alpha_i$. Choose $m = p_1^{\beta_1} p_2^{\beta_2} \cdots p_k^{\beta_k}$ and $l = p_1^{\gamma_1} p_2^{\gamma_2} \cdots p_k^{\gamma_k}$ such that $\alpha_i = \beta_i + \gamma_i$ and $\sum \beta_i = \lceil \alpha/2 \rceil$ and $\sum \gamma_i = \lfloor \alpha/2 \rfloor$. This leads to the recurrence:

$$\begin{aligned} D(\alpha) &= D(\lceil \alpha/2 \rceil) + D(\lfloor \alpha/2 \rfloor) + O(\lg n) \\ &= O(\lg n \lg \lg n) \end{aligned}$$

when $\alpha < \lg n$ which it always is.

■

Chapter 5

An optimal depth counting network

This chapter contains our main result. Namely, the uniform polynomial-time construction of an $O(\lg n)$-depth counting network. There are a number of steps involved in this construction. Tools such as the ladder of Section 4.2 and the 2-smoother in-Section 4.3 are used. We begin by describing and analyzing some other useful tools.

5.1 Building blocks

5.1.1 The butterfly balancing network

Definition 5.1.1 *A 2^k-input* butterfly balancing network*, where k is a nonnegative integer, may be defined recursively as follows. If $k = 0$, then it is a single wire. If $k > 0$, then it is constructed from two 2^{k-1}-input butterfly balancing networks and an additional level of 2^{k-1} balancers as indicated in Figure 5-1.*

Note that a 2^k-input butterfly balancing network has depth k.

Lemma 5.1.1 *Consider a 2^k-input butterfly balancing network. Let i and j denote two k-bit integers that differ in a single bit position, with $i < j$. Then for any input sequence, either $y_i = y_j$ or $y_i = y_j + 1$.*

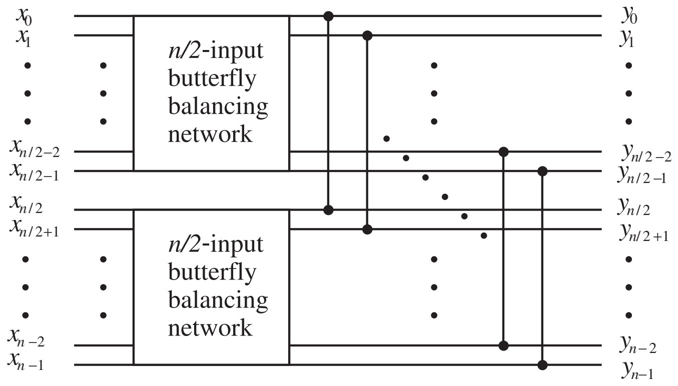

Figure 5-1: An n-input butterfly balancing network where $n = 2^k$.

Proof: We prove the claim by induction on k. If $k = 0$, there is nothing to prove. Now assume that the claim holds for $k < d$, $d \geq 1$, and consider the case $k = d$. If i and j differ in bit position d, then the result is immediate since outputs i and j are connected to the same depth d balancer. Thus, we may assume that i and j differ in some bit position a, $0 \leq a < d$. Let i' (resp., j') denote the integer having the same binary representation as i (resp., j), except in bit position d. Note that outputs y_i and $y_{i'}$ (resp., y_j and $y_{j'}$) represent the two outputs of some balancer b_0 (resp., b_1) at depth d. Let a total of r_0 (resp., r_1) tokens be received by balancer b_0 (resp., b_1). By the induction hypothesis, $r_1 \leq r_0 \leq r_1+2$. If $i < i'$, then $y_i = \lceil r_0/2 \rceil$ and $y_j = \lceil r_1/2 \rceil$, so that either $y_i = y_j$ or $y_i = y_j + 1$. The case $i > i'$ is similar. ∎

Lemma 5.1.1 shows that the output sequence from a butterfly balancing network has a hypercube-like structure.

We use the expression $\text{bin}(i, k)$ to denote the k-bit binary representation of the integer i, for $0 \leq i < 2^k$. When y_i is mapped to node $\text{bin}(i, k)$ of a 2^k-node hypercube, the output wire containing the most tokens corresponds to 0^k (i.e., y_0) while the output wire with the fewest tokens corresponds to 1^k (i.e., y_{2^k-1}). Furthermore, if one considers a sequence of outputs $\{y_{j_0=2^k-1}, y_{j_1}, \ldots, y_{j_{k-1}}, y_{j_k=0}\}$ corresponding to a chain in the hypercube (when viewed as a Boolean lattice) beginning with the node 1^k and ending with the node 0^k, then $0 \leq y_{j_{i+1}} - y_{j_i} \leq 1$, $0 \leq i < k$.

Corollary 5.1.1.1 *For any input sequence to the 2^k-input butterfly balancing network, and any pair of output wires i and j, we have $|y_i - y_j| \leq k$.*

Proof: An immediate consequence of Lemma 5.1.1, along with the fact that k bits are used to describe the address of each wire. ∎

In our "randomized" construction of Section 5.2, we will prove that after the tokens pass through a particular stage of the network containing a butterfly balancing network, with high probability all of the output wires corresponding to nodes in the "middle" levels of the hypercube (i.e., those with a large number of both 0's and 1's in the binary representation of their addresses), will contain very close to μ tokens where μ is the average number of tokens per input wire. Let G denote the set of output wires of a 2^k-input butterfly balancing network whose addresses have binary representations containing at least αk 0's and αk 1's, $0 \leq \alpha \leq 1/2$. We say that G is the set of *α-good* wires while those not in G are *α-bad*. The α-good wires are considered the "middle" levels of the hypercube.

5.1.2 Pairing networks

The pairing network is useful both in the proof of the existence of an $O(\lg n)$-depth counting network and in the construction of a k-smoother of Section 5.4 with an odd number of input wires.

Definition 5.1.2 *An (m, n, k)-*pairing network*, for nonnegative integers m, n, and k satisfying $n \geq m2^k$, is an n-input, depth-k balancing network constructed as follows. Upon entry to the network, m wires have been designated as* bad *inputs while the remaining $n - m$ wires have been designated as* good inputs. *Similarly, the outputs of the network will be partitioned into a set of $m2^k$* bad outputs, *and $n - m2^k$* good outputs. *If $k = 0$, the network consists of n wires, and the bad (resp., good) outputs simply correspond to the bad (resp., good) inputs. If $k > 0$, the desired network $\mathcal{N}$ consists of an $(m, n, k-1)$-pairing network $\mathcal{N}'$ (note that $n \geq m2^k$ implies $n \geq m2^{k-1}$) followed by an additional level of $m2^{k-1}$ balancers pairing (in an arbitrary fashion)* *each of the bad outputs of $\mathcal{N}'$ with a good output of $\mathcal{N}'$. The bad outputs of $\mathcal{N}$ are exactly the $m2^k$ outputs of this set of $m2^{k-1}$ balancers (see also Figure 5-2).*

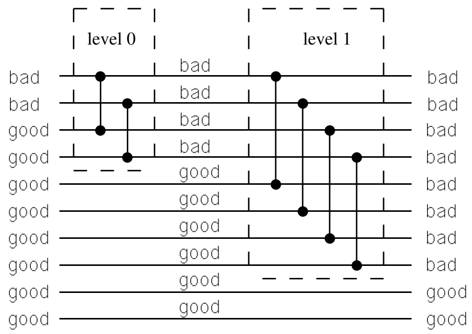

Figure 5-2: A $(2, 10, 2)$-pairing network.

The following lemma shows that a pairing network can be used to smooth the bad inputs using the good inputs:

Lemma 5.1.2 *If the input sequence to an (m, n, d)-pairing network $\mathcal{N}$ is such that for some pair of integers $a \leq b$, every good (resp., bad) input receives between a and b (resp., $a - 2^k$ and $b + 2^k$) tokens (for some $k \geq d$), then every good (resp., bad) output receives between a and b (resp., $a - 2^{k-d}$ and $b + 2^{k-d}$) tokens.*

Proof: The good output wires are not connected to balancers so they will clearly output between a and b tokens. With respect to the bad output wires, we will now argue that no bad output receives more than $b + 2^{k-d}$ tokens; a symmetric argument may be used show that no bad output receives fewer than $a - 2^{k-d}$ tokens. In fact, we will prove the following stronger claim: No wire at depth i receives more than $b + 2^{k-i}$ tokens, $0 \leq i \leq k$. This claim may be proven by induction on i. The base case, $i = 0$, is immediate. For the induction step, note that every wire at depth $i+1$, $0 \leq i < k$, is an output of a balancer that received at most $b + 2^{k-i}$ tokens (by the induction hypothesis) on one input and at most b tokens on the other input (since one of the inputs must be good). Each of the outputs of such a balancer will receive at most $\lceil (2b + 2^{k-i})/2 \rceil = b + 2^{k-i-1}$ tokens, as required. ■

Corollary 5.1.2.1 *If the input sequence to an (m, n, d)-pairing network $\mathcal{N}$ is such that for some pair of integers $a \leq b$, every good (resp., bad) input receives between a and b (resp., $a - 2^d$ and $b + 2^d$) tokens, then every good (resp., bad) output receives between a and b (resp., $a - 1$ and $b + 1$) tokens.*

5.2 A random construction

We now present the first major construction leading to our main result. We make use of the networks defined in the previous section to construct a "random" network.

Definition 5.2.1 *Given a set of balancing networks and an associated probability distribution, the random network obtained by sampling from the set is referred to as a* random balancing network*.*

We will be particularly interested in random balancing networks that count any input sequence with high probability. Such a network will be referred to as a random counting network, and a lower bound on the associated probability of success (the minimum over all input sequences of the probability that the random network counts the sequence) will be explicitly stated.

In this section, we present an $O(d)$-depth family of 2^d-input random counting networks that count with probability at least $1 - 2^{-2^{\alpha d}}$, where α is any constant such that $0 < \alpha < \frac{1}{2}$. Since the depth of the network is independent of α, we will choose α close to $\frac{1}{2}$. Throughout this section, fix a choice of $\alpha < \frac{1}{2}$, let $\alpha_0 = (\alpha + \frac{1}{2})/2$, let d denote an arbitrary nonnegative integer, let $d' = \left\lceil \sqrt{d} \right\rceil$, and let $\mathcal{N}^*$ denote the 2^d-input random counting network to be constructed.

In order to define $\mathcal{N}^*$, we need to provide a set of 2^d-input balancing networks and an associated probability distribution. This set of networks will consist of $(2^d)!$ networks that are identical in every respect except for a permutation of the wires that is applied at one point in the construction. The probability distribution will be uniform over this set of networks. Thus, letting S_d denote the set of $(2^d)!$ permutations on 2^d objects, the networks of the set $\mathcal{N}^*$ are in one-to-one correspondence with the elements of S_d. In particular, for each permutation π in S_d, we will construct the balancing network $\mathcal{N}_\pi$ of $\mathcal{N}^*$ by applying the following procedure (see also Figure 5-3).

Phase 1: Apply a butterfly balancing network to all 2^d input wires.

Phase 2: Let A_i denote the set of $2^{d'}$ wires $X_{\pi(j)}$, $i2^{d'} \leq j < (i+1)2^{d'}$, $0 \leq i < 2^{d-d'}$. Apply a $2^{d'}$-input bitonic counting network [5] to each A_i.

Phase 3: Apply a butterfly balancing network to all 2^d wires.

Phase 4: Let B denote the set of α_0-bad outputs of Phase 3. Apply a $(|B|, 2^d, \lg d)$-pairing network, mapping the wires of B to the bad inputs of the pairing network.

Phase 5: Apply a butterfly balancing network to all 2^d wires.

Phase 6: For the remainder of the construction, we refer to B (resp., G) as the set of α_0-bad (resp., α_0-good) outputs of Phase 5. Apply a 2-counter to G.

Phase 7: Partition the outputs of the 2-counter of Phase 6 into two equal-sized sets G_0 and G_1, placing the top $|G|/2$ outputs in G_0 and the bottom $|G|/2$ outputs in G_1. Apply a $(|B|, |B|+|G_0|, 2)$-pairing network to $B \cup G_0$, mapping the wires of B to the bad inputs of the pairing network.

Phase 8: Let B' denote the set of $4|B|$ bad outputs of the pairing network of Phase 7. Apply a $(|B'|, |B'| + |G_1|, 2)$-pairing network to $B' \cup G_1$, mapping the wires of B' to the bad inputs of the pairing network.

Phase 9: Apply a 2-counter to all 2^d wires.

Lemma 5.2.1 *The output of Phase 1 is d-smoothed.*

Proof: Immediate from Corollary 5.1.1.1. ∎

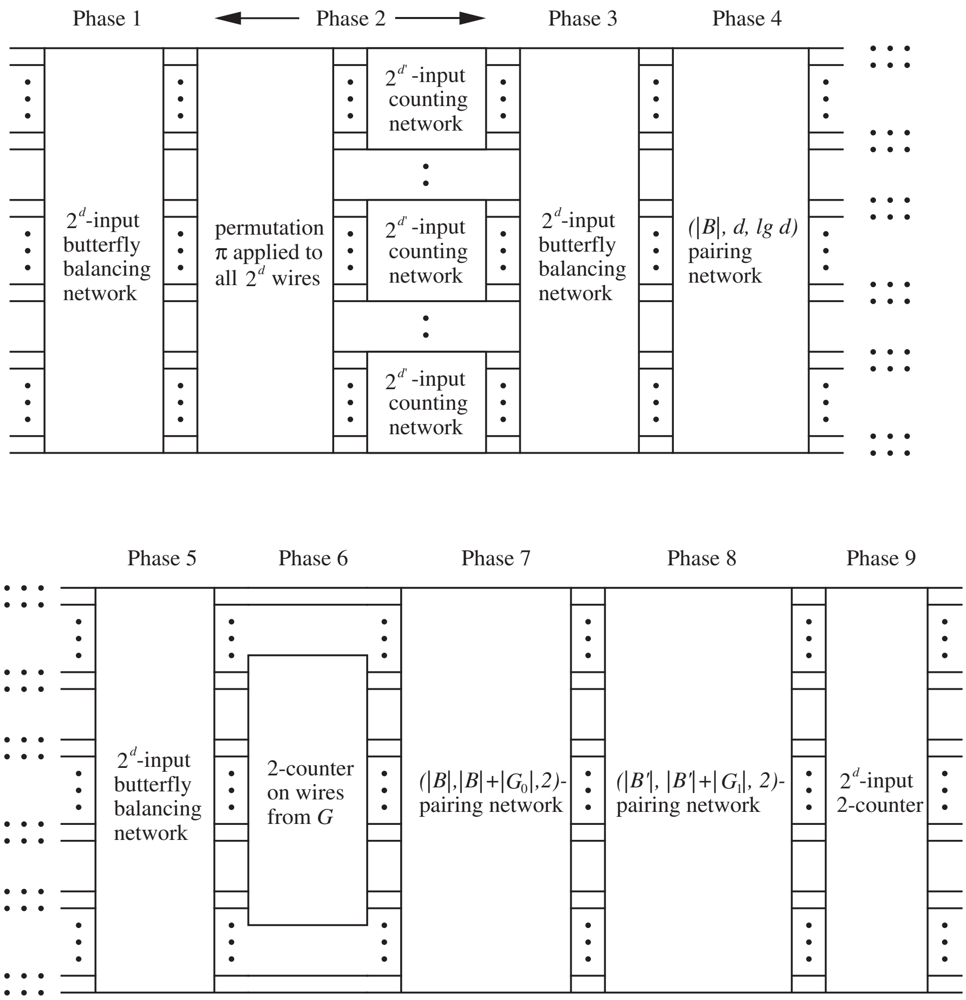

Figure 5-3: The balancing network $\mathcal{N}_\pi$.

We are left to prove that the remainder of the network is a d-counter with high probability. By Lemma 4.3.3, we can assume that the number of tokens per wire is between 0 and d after Phase 1.

Definition 5.2.2 *Let I denote an input sequence containing a total of $\mu 2^d$ tokens for some real value μ, let $\mu' = \left\lfloor \mu + \frac{1}{2} \right\rfloor$, and focus on the 2^d outputs of Phase 2 when input sequence I is applied to network $\mathcal{N}_\pi$. Let a (resp., b) denote the number of these wires receiving strictly more (resp., fewer) than $\mu' + 1$ (resp., $\mu' - 1$) tokens. The pair $(I, \mathcal{N}_\pi)$ is defined to be α_0-nice if and only if $\max\{a, b\} \leq 2^{\alpha_0 d}$.*

In the next portion of our proof we obtain a Chernoff-like bound for sampling *without* replacement. We use martingales for this purpose.

Definition 5.2.3 *A martingale is a sequence of random variables $X_0, \ldots, X_m$ such that*

$$E\left[X_{i+1} \mid X_i\right] = X_i,$$

$0 \leq i < m$.

We will make use of the following variant of Azuma's Inequality [4] in the analysis of Phase 2:

Theorem 5.2.1 *Let $X_0 = c$ and let $X_0, \ldots, X_m$ be a martingale with*

$$|X_{i+1} - X_i| \leq U,$$

$0 \leq i < m$. *Then*

$$\Pr(X_m - c > \lambda U \sqrt{m}) < e^{-\lambda^2/2}$$

for all $\lambda > 0$. ∎

The following lemma represents a straightforward application of Azuma's Inequality.

Lemma 5.2.2 *Let Q be an arbitrary set of N numbers with mean μ drawn from the real interval $[0, U]$. Let the random variable S denote the sum of m elements $\{q_1, \ldots, q_m\}$ chosen uniformly without replacement from Q. Then for all $\lambda > 0$,*

$$\Pr(S > \lambda U\sqrt{m} + m\mu) < e^{-\lambda^2/2}.$$

Proof: Let $X_0 = E[S]$ and for $1 \leq i \leq m$ let

$$X_i = E[S \mid q_1, \ldots, q_i].$$

The X_i's form a martingale. Furthermore,

$$\begin{aligned}
&|X_{i+1} - X_i| \\
&\qquad = \left| E[X \mid q_1, \ldots, q_{i+1}] - E[X \mid q_1, \ldots, q_i] \right| \\
&\qquad = \left| \frac{N-m}{N-i-1} \left(q_{i+1} - \frac{N\mu - \sum_{1 \leq k \leq i} q_k}{N-i} \right) \right| \\
&\qquad \leq U
\end{aligned}$$

since $0 \leq (N-m)/(N-i-1) \leq 1$, $0 \leq q_{i+1} \leq U$, and $0 \leq (N\mu - \sum_{1 \leq k \leq i} q_k)/(N-i) \leq U$. In addition, $X_0 = m\mu$ and $S = X_m$. By Theorem 5.2.1,

$$\Pr(S - m\mu > \lambda U\sqrt{m}) < e^{-\lambda^2/2} \tag{5.1}$$

for all $\lambda > 0$. ■

The following lemma represents the crux of the random construction. It shows that a great deal of "global" smoothing can be accomplished by smoothing small sets of wires in parallel.

Lemma 5.2.3 *Let I be any input sequence to $\mathcal{N}_\pi$ where π is chosen uniformly at random from S_d. Then with probability at least $1 - 2^{-2^{\alpha d}}$, the pair $(I, \mathcal{N}_\pi)$ is α_0-nice.*

Proof: Suppose that input sequence I contains $\mu 2^d$ tokens and let $\mu' = \left\lfloor \mu + \frac{1}{2} \right\rfloor$. The set A_i (refer to Phase 2) is defined to be *high-bad* (resp., *low-bad*) if the total number of tokens received by A_i exceeds $2^{d'}(\mu'+1)$ (resp., is less than $2^{d'}(\mu'-1)$), for $0 \le i < 2^{d-d'}$. We will show that with probability at least $1 - 2^{-2^{\alpha d}}$, at most $2^{\alpha d}$ of the A_i's are high-bad. A symmetric result holds for the number of low-bad A_i's. Thus, at the end of Phase 2, $2^{\alpha d} 2^{\lceil d' \rceil} = o(n^{\alpha_0})$ wires will contain more (resp., fewer) than $\mu'+1$ (resp., $\mu'-1$) tokens, proving the lemma. Let X denote the event that a particular set of $r \stackrel{\text{def}}{=} \left\lfloor 2^{\alpha d} \right\rfloor$ of the A_i's are all high-bad. We have

$$\begin{aligned} &\Pr(\text{at least } r \text{ of the } A_i\text{'s are high-bad}) \\ &\qquad \le \binom{2^{d-d'}}{r} \Pr(X) \\ &\qquad \le 2^{dr} \Pr(X). \end{aligned}$$

Next we show that $\Pr(X) = 2^{-\omega(d) r}$, yielding the result. Let S be the total number of tokens on a particular set of $r2^{d'}$ wires. Then $\Pr(S \ge r2^{d'}(\mu'+1)) \ge \Pr(X)$. S is the sum of values sampled without replacement, so Lemma 5.2.2 can be applied. In this case, $\lambda = \frac{\sqrt{m}}{2U}$, $U = d$, $m = r2^{d'}$, and $N = 2^d$. So the probability bound in Lemma 5.2.2 becomes

$$\begin{aligned} \Pr(S \ge r2^{d'}(\mu'+1)) &\le \Pr(S \ge r2^{d'}(\mu + \frac{1}{2})) \\ &< e^{-\frac{m}{8U^2}} \\ &= e^{-\frac{r2^{d'}}{8d^2}} \\ &= 2^{-\omega(d) r}, \end{aligned}$$

as desired. ∎

For the remainder of this section, we assume that I is a particular input sequence with $\mu 2^d$ tokens and that $\mathcal{N}_\pi$ is a particular network in $\mathcal{N}^*$ such that the pair $(I, \mathcal{N}_\pi)$ is α_0-nice. We also set $\mu' = \left\lfloor \mu + \frac{1}{2} \right\rfloor$.

Lemma 5.2.4 *If I is input to $\mathcal{N}_\pi$ then at any level after Phase 3, at most $d2^{\alpha d+d'+1}$ wires will receive more than $\mu'+1$ (resp., fewer than $\mu'-1$) tokens.*

Proof: Before Phase 3, we know that at most $2^{\alpha d+d'+1}$ wires received more than $\mu'+1$ tokens. Since each of these wires receives at most d tokens, they can contribute tokens to at most $d2^{\alpha d+d'+1}$ wires at any given level of the network. Thus, at any level after Phase 3 there will be at most $d2^{\alpha d+d'+1}$ wires with more than $\mu'+1$ tokens. An analogous argument yields the upper bound for the number of wires with fewer than $\mu'-1$ tokens. ■

Lemma 5.2.5 *If I is input to $\mathcal{N}_\pi$ then every α_0-good output of Phase 3 will receive $\mu'-1$, μ', or $\mu'+1$ tokens.*

Proof: By Lemma 5.2.4, at most $d2^{\alpha d+d'+1}$ outputs of Phase 3 contain more than $\mu'+1$ tokens. Now consider the hypercube-like structure of the wires after Phase 3 (Lemma 5.1.1). Recall that $\mathrm{bin}(i,k)$ denotes the k-bit binary representation of the integer i, for $0 \le i < 2^k$. A wire X_i where $\mathrm{bin}(i,d)$ has at least $\alpha_0 d$ 1's cannot receive more than $\mu'+1$ tokens. Suppose one such wire did. Then all wires corresponding to the subcube with dimension at least $\alpha_0 d$ defined by fixing all the 0 bits in $\mathrm{bin}(i,d)$ and allowing the others to vary would have more than $\mu'+1$ tokens. But there are more than $d2^{\alpha d+d'+1}$ such wires, a contradiction. The argument above can be repeated to show that a wire X_i where $\mathrm{bin}(i,d)$ has at least $\alpha_0 d$ 0's cannot have fewer than $\mu'-1$ tokens. This proves the lemma. ■

We must ensure that enough good wires are input to the pairing networks in $\mathcal{N}_\pi$ so that all of the bad wires can be matched. Since no pairing network in $\mathcal{N}_\pi$ has depth greater than $\lg d$, the following lemma provides the necessary lower bound on the number of good wires by providing an upper bound on the number of bad wires:

Lemma 5.2.6 *The number of α_0-bad wires in $\mathcal{N}_\pi$ is $o(\frac{2^d}{d})$.*

Proof: We have

$$\begin{aligned}
|\alpha_0\text{-bad wires}| &= 2 \sum_{0 \le i < \alpha_0 d} \binom{d}{i} \\
&= 2^{d(H(\alpha_0)+o(1))} \\
&= O(2^{cd}),
\end{aligned}$$

where H denotes the entropy function, and c is a constant with $c < 1$ since $\alpha_0 < \frac{1}{2}$. ∎

Lemma 5.2.7 *If I is input to $\mathcal{N}_\pi$ then every output wire of Phase 4 will receive between $\mu' - 2$ and $\mu' + 2$ tokens. Furthermore, $o(2^{\alpha_0 d})$ of these outputs will receive exactly $\mu' - 2$ (resp., $\mu' + 2$) tokens.*

Proof: The claim that all wires will contain between $\mu' - 2$ and $\mu' + 2$ tokens follows immediately from Corollary 5.1.2.1 with $a = \mu' - 1$, $b = \mu' + 1$, and $k = \lg d$. By Lemma 5.2.4, the number of outputs of Phase 4 that receive exactly $\mu' - 2$ (resp., $\mu' + 2$) tokens is at most $d2^{\alpha d + d' + 1} = o(2^{\alpha_0 d})$. ∎

Lemma 5.2.8 *If I is input to $\mathcal{N}_\pi$ then after Phase 5, each α_0-good wire will contain $\mu' - 1$, μ', or $\mu' + 1$ tokens and each α_0-bad wire will have between $\mu' - 2$ and $\mu' + 2$ tokens.*

Proof: Before Phase 5, the maximum number of tokens per wire is $\mu' + 2$ while the minimum number is $\mu' - 2$. As a result, for the remainder of the network, the number of wires containing $\mu' + 2$ (similarly $\mu' - 2$) tokens cannot increase. The butterfly in Phase 5 restructures the wires in the form of a hypercube (Lemma 5.1.1). After this phase, there will be $o(2^{\alpha_0 d})$ wires with either $\mu' - 2$ or $\mu' + 2$ tokens. Arguing as in the proof of Lemma 5.2.5, each α_0-good wire will receive $\mu' - 1$, μ', or $\mu' + 1$ tokens. ∎

Lemma 5.2.9 *If I is input to $\mathcal{N}_\pi$ then after Phase 6, the good wires will have a counted shape in which each wire contains either $\mu' - 1$ or μ' tokens, or contains μ' or $\mu' + 1$ tokens.*

Proof: Immediate from Lemma 5.2.8 and the definition of a 2-counter. ∎

Lemma 5.2.10 *If I is input to $\mathcal{N}_\pi$ then the output of Phase 8 is 2-smoothed.*

Proof: Since the good wires have a counted shape, either G_0 or G_1 is homogeneous (i.e., all wires in the set receive the same number of tokens). Assume without loss of generality that the good wires from Phase 6 contain either μ' or $\mu' + 1$ tokens. We consider two cases:

G_0 is homogeneous: If each wire in G_0 receives $\mu' + 1$ tokens, then by Corollary 5.1.2.1, every output of Phase 7 will receive between μ' and $\mu' + 2$. If each wire in G_0 receives μ' tokens, then by the same reasoning every output of Phase 7 will receive between $\mu' - 1$ and $\mu' + 1$ tokens.

G_1 is homogeneous: After Phase 7, only the wires in B' can receive either $\mu' - 2$ or $\mu' + 2$ tokens. Arguing as in the case where G_0 is homogeneous, the output of Phase 8 will be 2-smoothed.

∎

Lemma 5.2.11 *If I is input to $\mathcal{N}_\pi$ then the output of Phase 9 will have a counted shape.* ∎

Theorem 5.2.2 *For any input sequence I, the $O(d)$-depth random counting network $\mathcal{N}^*$ will count I with probability at least $1 - 2^{-2^{\alpha d}}$.* ∎

5.3 An optimal existence result

In this section, we establish the existence of $O(\lg n)$-depth counting networks. Our networks are non-uniform in that we do not know of any polynomial-time procedure for generating the network with n inputs. However, our results can easily be extended to provide a randomized algorithm that, given n, produces an $O(\lg n)$-depth counting network in polynomial-time with extremely high probability.

Our deterministic network is constructed recursively using the random counting network of Section 5.2 as a building block. Our approach is to recursively construct deterministic counting networks over small sets of input wires and then to "merge" the outputs of these networks via a larger random counting network. In order to achieve a small depth of recursion, it is desirable to partition the n inputs into a large number of small sets. The granularity of the recursive partition will be determined in such a way that the total number of possible inputs to the larger network will be small relative to the probability with which the network succeeds (i.e., small relative to the reciprocal of the probability of failure). As a result, we will be able to argue that some fixed choice for the larger network will be guaranteed to produce a counted output.

Lemma 5.3.1 *If the number of tokens input to each wire of a 2^d-input counting network is no more than l, then the number of possible output shapes is at most $l2^d + 1$.* ∎

Lemma 5.3.2 *Consider a 2^d-input balancing network $\mathcal{N}$ made up of $2^{d-\lfloor \beta d \rfloor}$ disjoint $2^{\lfloor \beta d \rfloor}$-input counting networks, $0 \leq \beta \leq 1$. If the number of tokens received by each input wire of $\mathcal{N}$ is no more than d, then the number of possible output shapes is at most $(d2^d + 1)^{2^{d-\lfloor \beta d \rfloor}} < 2^{d2^{d-\lfloor \beta d \rfloor}(1+o(1))}$.*

Proof: By Lemma 5.3.1, each $2^{\lfloor \beta d \rfloor}$-input network can produce at most $d2^{\lfloor \beta d \rfloor} + 1 \leq d2^d + 1$ possible output shapes. Since $\mathcal{N}$ contains $2^{d-\lfloor \beta d \rfloor}$ such networks, the result follows. ∎

Lemma 5.3.3 *Let $\frac{1}{2} < \beta < 1$, and let S denote any fixed set of at most $2^{d2^{d-\lfloor \beta d \rfloor}(1+o(1))}$ possible input sequences of length 2^d. Then there exists a 2^d-input balancing network of depth $O(d)$ that counts every shape in the set S.*

Proof: In Section 5.2, we constructed a set $\mathcal{N}^*$ of 2^d-input balancing networks with the property that any fixed input sequence is counted by at least a $1 - 2^{-2^{\alpha d}}$ fraction of the networks in $\mathcal{N}^*$, where α is any constant such that $0 < \alpha < \frac{1}{2}$. If

$2^{d2^{d-\lfloor \beta d \rfloor}(1+o(1))} 2^{-2^{\alpha d}} < 1$, then at least one of the networks in $\mathcal{N}^*$ must count every input sequence in the set S. The desired inequality is satisfied for $\beta > 1 - \alpha$ (e.g., $\beta = 1 - \alpha/2$), and d sufficiently large. ∎

Theorem 5.3.1 *There exists a 2^d-input counting network of depth $O(d)$.*

Proof: Consider the following recursive construction. First, apply a 2^d-input butterfly balancing network. This yields a d-smoothed shape, and we can assume without loss of generality that each output wire receives a number of tokens between 0 and d, inclusive. Second, partition the butterfly outputs into $2^{d-\lfloor \beta d \rfloor}$ sets of size $2^{\lfloor \beta d \rfloor}$, $\frac{1}{2} < \beta < 1$, and count each of these sets recursively. Finally, feed the outputs of these networks into a 2^d-input network $\mathcal{N}$ of depth $O(d)$ that counts every input sequence that it could possibly receive from the smaller counting networks. The existence of network $\mathcal{N}$ is guaranteed by Lemmas 5.3.2 and 5.3.3. Note that it is not necessary to include Phase 1 of the construction of $\mathcal{N}$. This butterfly network is not needed because the butterfly applied at the beginning of this existential construction performs the same function function, namely, making sure all wires are within d of one another. Let $D(d)$ denote the depth of such a 2^d-input network. Then

$$\begin{aligned} D(d) &= D(\lfloor \beta d \rfloor) + O(d) \\ &= O(d). \end{aligned}$$

∎

5.4 A deterministic k-smoother

In this section, we continue the analysis of Section 4.3 by presenting an explicit construction of a n-input k-smoother, where n is any positive integer and k is a positive integer with $2^{\lceil \lg k \rceil} \leq n - 1$. When n is a power of 2, our construction works for all k. When k is a constant, our construction yields an $O(\lg n)$-depth network.

The network is recursive and consists of the following five phases (see also Figure 5-4): If the network has $2n + 1$ input wires (an odd number) then we apply these first

5 phases to all but one of these wires and include an additional two phases discussed at the end of this section. If the network has $2n$ input wires then these first 5 phases are sufficient to smooth.

Phase 1: Apply a $2n$ input sorting network to the $2n$-input wires.

Phase 2: Recursively apply an n-input $k-1$-smoother to the top n wires and another n-input $k-1$-smoother to the bottom n wires.

Phase 3: Apply an n-input sorting network to the top n wires and another n-input sorting network to the bottom n wires.

Phase 4: Apply a ladder to all $2n$ wires.

Phase 5: Apply a n-input $\left\lceil \frac{k+1}{2} \right\rceil$-smoother to all $2n$ wires.

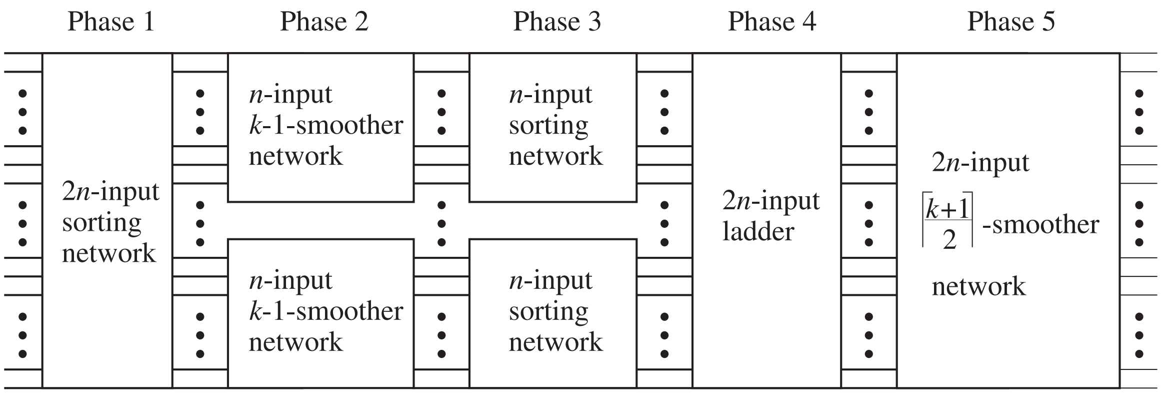

Figure 5-4: A $2n$-input k-smoother.

The base case in the recursion is the 2-smoother constructed in Section 4.3.

If $2n+1$ input wires are involved then we perform 2 additional phases which are described later in this section.

We begin by proving that our construction produces a $2n$-input k-smoother.

By lemma 4.3.3, it is sufficient to prove that our network counts when each x_i is drawn from $[0, k]$, for $0 \le i < n$. Fixing a particular input sequence of this type, let n_0 and n_k denote the number of wires receiving 0 and k tokens, respectively.

Lemma 5.4.1 *After applying Phase 1 of the construction (i.e., a $2n$-input sorting network), all wires containing k tokens are located in the top n_k wires. Similarly, all wires containing* 0 *tokens are located in the bottom n_0 wires.*

Proof: Immediate from lemma 4.3.2. ∎

Lemma 5.4.2 *After Phase 2 either the wires from the top $k-1$-smoother are smoothed or those from the bottom $k-1$-smoother are smoothed.*

Proof: If $n_k \geq n$ then none of the wires entering the top $k-1$-smoother will contain 0 tokens (Lemma 5.4.1). As a result, wires entering the top $k-1$-smoother will be $k-1$-smoothed. Similarly, if $n_k < n$ then none of the wires entering the bottom $k-1$-smoother will contain k tokens, so the input to the $k-1$-smoother will be $k-1$ smoothed. ∎

Lemma 5.4.3 *After Phase 3, either*

1. *The wires from the top sorting network are counted and there is a number n_p, with $0 \leq n_p \leq n$ such that the top n_p output-wires from the bottom sorting network contain no* 0*'s and the bottom $n - n_p$ output-wires contain no k's or*

2. *The wires from the bottom sorting network are counted and there is a number n_p, with $0 \leq n_p \leq n$ such that the top n_p output-wires from the top sorting network contain no* 0*'s and the bottom $n - n_p$ output-wires contain no k's.*

Proof: If the top $k-1$-smoother smooths its input in Phase 2, then the top sorting network will count its input by Lemma 2.2.1. The bottom sorting network will separate the wires containing k tokens from those containing 0 tokens as described above due to Lemma 4.3.2. We have shown that when the top $k-1$-smoother smooths, case 1 in the lemma holds. By a symmetric argument, when the bottom $k-1$-smoother smooths, case 2 in the lemma holds. ∎

Lemma 5.4.4 *After Phase 4, all $2n$ wires are $\left\lceil \frac{k+1}{2} \right\rceil$-smoothed.*

Proof: Assume without loss of generality that the upper sorting network from Phase 3 is counted. Suppose these wires contain either a or $a+1$ tokens each. Let n_a be the number of wires in this portion of the network containing exactly a tokens. Recall the integer n_p from Lemma 5.4.3. There are two cases to consider (see also Figure 5-5).

$n_a \geq n_p$: In this case no wires containing k tokens will be paired with wires containing $a+1$ tokens by the ladder. So at most $\left\lceil \frac{k+a}{2} \right\rceil$ tokens and at least $\left\lfloor \frac{a+0}{2} \right\rfloor$ tokens are output on any wire from Phase 4. As a result, all wires are smoothed to within $\left\lceil \frac{k+a}{2} \right\rceil - \left\lfloor \frac{a}{2} \right\rfloor \leq \left\lceil \frac{k+1}{2} \right\rceil$.

$n_a < n_p$: In this case no wires containing 0 tokens will be paired with wires containing a tokens by the ladder. So at most $\left\lceil \frac{k+a+1}{2} \right\rceil$ tokens and at least $\left\lfloor \frac{a+1}{2} \right\rfloor$ tokens are output on any wire from Phase 4. As a result, all wires are smoothed to within $\left\lceil \frac{k+a+1}{2} \right\rceil - \left\lfloor \frac{a+1}{2} \right\rfloor \leq \left\lceil \frac{k+1}{2} \right\rceil$.

■

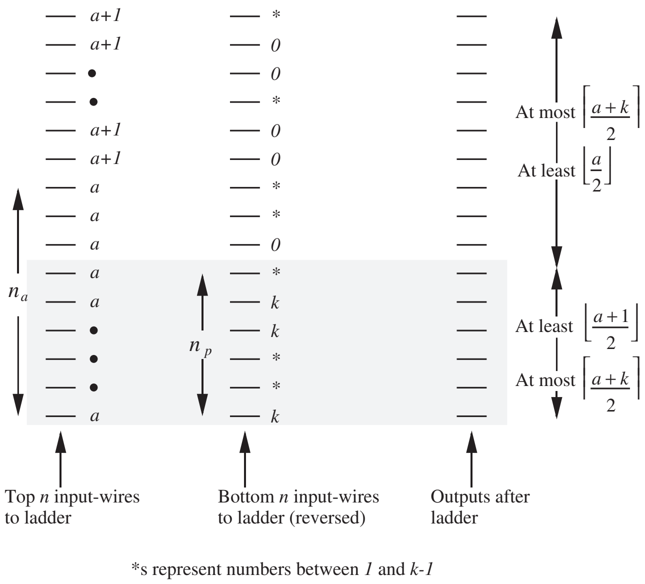

Figure 5-5: Case $n_a \geq n_p$

Lemma 5.4.5 *The output of Phase 5 is smoothed.*

Proof: Immediate from the definition of a $\left\lceil \frac{k+1}{2} \right\rceil$-smoother. ∎

We now consider k-smoothers with $2n+1$ wires. In this case, we first apply a $2n$-input k-smoother to the top $2n$-input wires and then add the following 2 phases (see also Figure 5-6):

Phase 6: Apply a $(1, 2^{\lceil \lg k \rceil}, \lceil \lg k \rceil)$-pairing network. The bad wire is the wire which has not yet been smoothed. The good wires are any subset of the remaining $2n$ wires.

Phase 7: Apply a 2-smoother to all $2n+1$ wires.

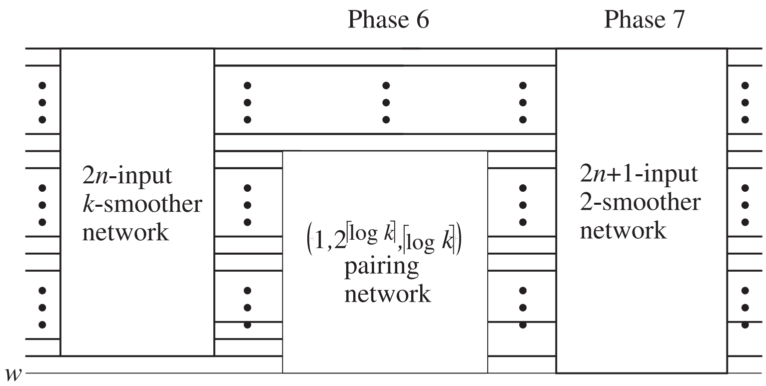

Figure 5-6: A $2n+1$-input k-smoother.

Lemma 5.4.6 *After Phase 6 the input is 2-smoothed.*

Proof: Immediate from definition of a pairing network. ∎

Lemma 5.4.7 *After Phase 7 all $2n+1$ wires are smoothed.*

Proof: Immediate from definition of a 2-smoother. ∎

Theorem 5.4.1 *There exists an explicitly constructible family of $O(k^{\lg k} \lg n)$-depth, n-input k-smoothers for $2^{\lceil \lg k \rceil} \leq n-1$. For k constant, this depth is $O(\lg n)$.*

Proof: Our construction is recursive and has depth

$$\begin{aligned} depth(k,n) &= depth(k-1,\lfloor n/2 \rfloor) + depth\left(\left\lceil \frac{k+1}{2} \right\rceil, n\right) + O(\lg k) + O(\lg n) \\ &= O(k^{\lg k} \lg n) \end{aligned}$$

■

The construction above is sufficient for our needs in the construction of our $O(\lg n)$-depth counting network. We next show that a k-smoother can be built for any n. Though the following construction has large depth, it answers the open question asked in [1]: Can a counting network be constructed for n-inputs when the input is k-smoothed. Note that the depth of a network has to be at least $\lg k$, independently of n. This is because by Theorem 3.1.2 a 2^d-smoother is a smoothing network where d is the depth of the network. But by Theorem 3.2.1, there is no such network if n is not a power of 2. Thus k must be at least 2^d yielding this result.

Theorem 5.4.2 *There exists an explicitly constructible family of n-input k-smoothers.*

Proof: The construction uses the following idea: we construct a balancing network such that if the input is not smoothed, two wires which differ by at least 2 will be connected by a balancer. We repeat this balancing network enough times to ensure that the input has been smoothed. To see that a limited number of balancing networks are required to ensure that the input is smoothed, we consider the variance of the input after each phase. Let $\overline{x}$ be the average number of tokens per wire for a given input I. The variance is then $\sum(x_i - \overline{x})^2 = \sum x_i^2 - n\overline{x}^2$. Since the input is k-smoothed this function can be no more than nk^2. We now show that if the input is not smoothed, then a single phase will reduce the variance by at least 2. Thus, at most $nk^2/2$ phases are required to smooth. Suppose x_i and x_j tokens pass through a balancer. Let us consider how the variance is affected. The total change in variance is $\left\lfloor \frac{x_i+x_j}{2} \right\rfloor^2 + \left\lceil \frac{x_i+x_j}{2} \right\rceil^2 - x_i^2 - x_j^2$. When $x_i + x_j$ is even,

$$\left\lfloor \frac{x_i+x_j}{2} \right\rfloor^2 + \left\lceil \frac{x_i+x_j}{2} \right\rceil^2 - x_i^2 - x_j^2 = 2\left(\frac{x_i+x_j}{2}\right)^2 - x_i^2 - x_j^2$$

$$= -\frac{(x_i - x_j)^2}{2}$$

So for $x_i + x_j$ even, the variance is nonincreasing. Also, if x_i differs from x_j, they will differ by at least 2, so the variance will decrease by at least this amount. We now consider the case when $x_i + x_j$ is odd.

$$\begin{aligned} \left\lfloor \frac{x_i + x_j}{2} \right\rfloor^2 + \left\lceil \frac{x_i + x_j}{2} \right\rceil^2 - x_i^2 - x_j^2 &= \left(\frac{x_i + x_j + 1}{2}\right)^2 + \left(\frac{x_i + x_j - 1}{2}\right)^2 - x_i^2 - x_j^2 \\ &= -\frac{(x_i - x_j)^2}{2} + \frac{1}{2} \end{aligned}$$

Because $x_i + x_j$ is odd, x_i and x_j differ by at least one. So the variance again is nonincreasing. When x_i differs from x_j by at least 2, then they differ by at least 3. In this case, the variance is reduced by at least 4.

A simple example of a phase which ensures that wires differing by at least two will be connected by a balancer (if there are such wires) is the following: apply a sorting network to all n wires with the comparators replaced with balancers. Then add apply a balancer which connects the top wire to the bottom wire. If the sorting network balances two wires which differ by at least two, then the phase has done its job. If not the wire containing the greatest number of packets will be on top and the wire with the fewest will be on the bottom. This is because the balancers will mimick comparators in this case. Thus the output of the sorting network will be sorted. The final balancer is then guaranteed to balance wires which differ by at least two unless the input is already smoothed. ■

5.5 A polynomial-time construction

In this section, we present a uniform polynomial-time construction of an $O(\lg n)$-depth counting network. This construction is derived from the existential proof in the previous section. In the last section, we showed the existence of a permutation mapping the output wires from the $2^{\lfloor (1-\alpha)d \rfloor}$-input counting networks to the input wires of the $2^{d'}$-input bitonic networks which causes the average number of tokens output from “most” of the bitonic counters to be extremely close to the overall average number of tokens per wire. In this section, we compute, in uniform polynomial-time, a permutation Π which has nearly this desired property and then correct for its shortcomings.

We construct a permutation Π which has the desired properties when we make a simplifying assumption about the outputs of the $2^{\lfloor (1-\alpha)d \rfloor}$-input counting networks. Namely, we assume that rather than outputting a counted shape, the networks output a *uniform* shape (i.e., each network outputs the same number of tokens on each of its wires). This assumption is not correct but we show that the output is not affected significantly when the assumption is removed. We then show that with a slight modification to the network in the existential proof, the simplifying assumption is not necessary.

The existential proof is a recursive construction which makes use of the randomized construction in Section 5.2. The construction of the counting network in this section is recursive, as well, and makes use of nearly the same construction as in Section 5.2. As a result, we will reuse analysis from Section 5.2 to prove our results. As in the existential proof, we do not use Phase 1 in Section 5.2. Instead, we make use of Phases 2 through 8 and we use our deterministic permutation Π which is described later in this section in place of the random permutation.

Because we apply a butterfly network followed by the recursive counting networks, we may assume that the input I to the “random” part of the network contains between 0 and d tokens per wire and under our simplifying assumption we may assume that each set of $2^{\lfloor (1-\alpha)d \rfloor}$ wires contains the same number of tokens. Under the properties of Π described below, Lemma 5.2.3 holds not only with high probability, but with certainty. Since Lemma 5.2.3 directly implies Lemma 5.2.10, we can conclude that the shape output from Phase 8 is 2-smoothed.

We now show that if we eliminate the simplifying assumption and allow the recursive counters to output counted shapes, rather than uniform shapes, then at the end of Phase 8 the output is 3-smoothed. Here we make use of Lemma 2.1.1. Suppose the i^{th} recursive counting network inputs x_i or $x_i + 1$ tokens per wire into Phase 2. Then we first pass x_i tokens per wire through Phase 2 until they leave Phase 8. By our analysis, the shape output will be 2-smoothed. We now have at most 1 token per wire left which must pass from Phase 2 through Phase 8. When these tokens leave Phase 8 they can increase the final output shape by at most 1 token per wire. Thus, after all tokens have left Phase 8, the shape will be 3-smoothed. In Section 5.4 we have constructed an $O(\lg n)$-depth 3-smoother. Thus, rather than applying a 2-smoother in Phase 9 as we did in the randomized construction, we apply a 3-smoother.

We are now left to define the deterministic permutation Π and show that it has the desired properties. To construct Π we construct a regular bipartite graph $G = U \times V$ representing the permutation (see Figure 5-7). Each node in U corresponds to one of the recursive counting networks while each node in V corresponds to a bitonic counter. Each edge in the graph represents a wire connecting an output-wire from a particular recursive counter to an input-wire of a particular bitonic counter. Note that the bipartite graph does not specify which output-wire of the recursive counter corresponds to which edge connected to the appropriate node of U. But the number of tokens on each of these wires is the same and so this decision may be made arbitrarily under the simplifying assumption because this decision does not affect the number of tokens entering a bitonic network. In other words, no matter what connections are made, the network will still perform its function.

The bipartite graph that we construct is based directly on work done by Noam Nisan and David Zuckerman in [25]. We begin the discussion of the properties of Π by defining the notion of *quasi-randomness*, an (ϵ, γ)-extractor, and the (ϵ, γ)-extractor property for a bipartite graph:

Definition 5.5.1 *[25] A probability distribution D on a set S is quasi-random within ϵ if for all $X \subseteq S$, $|D(X) - |X|/|S|| \leq \epsilon$. Here $D(X)$ denotes the probability of the set X according to distribution D.*

Definition 5.5.2 *[25] Let $E : \{0,1\}^r \times \{0,1\}^t \to \{0,1\}^s$. E is called a (γ, ϵ)-extractor if for every $A \subset \{0,1\}^r$, such that $|A| \geq 2^{\gamma r}$, the distribution of $E(x,y) \circ y$ induced by choosing x uniformly in A and y uniformly in $\{0,1\}^t$ is quasi- random (on $\{0,1\}^s \times$*

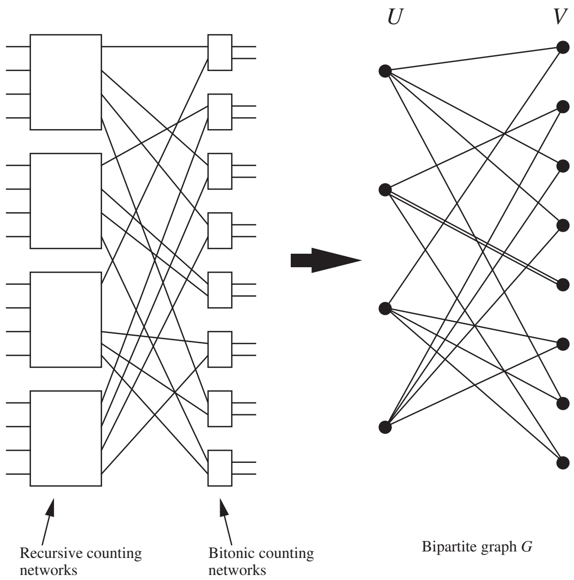

Figure 5-7: Correspondence between permutation Π and bipartite graph.

$\{0,1\}^t$) within ϵ.

Definition 5.5.3 *A bipartite graph G on $U \times V$ has the (ϵ, γ)-extractor property if for all $A \subseteq V$ with $|A| \geq \gamma$, the distribution of U induced by choosing the edges emanating from A uniformly is quasi-random on U within ϵ (see Figure 5-8).*

We construct a graph G corresponding to the permutation which has the (ϵ, γ)-extractor property for appropriate ϵ and γ. We begin by introducing some notation to simplify the analysis. Recall that μ is the average number of tokens per wire. The objective of the remainder of this section is to show that if a bipartite graph with the (ϵ, γ)-extractor property is used to define Π for appropriate values of ϵ and γ, then almost all the bitonic networks receive an average of close to μ tokens per wire under our simplifying assumption. The first several lemmas address the distribution of edges leading from a set $A \subseteq V$ to a set U.

For $A \subseteq V$ and $u \in U$ we let $d_A(u)$ denote the number of edges connecting u to the set A. We let $\overline{d_A}$ denote the average number of edges connecting a node in U to the set A. For $B \subseteq U$ we let $\overline{d_B}$ denote the average number of edges connecting a node in V to the set B.

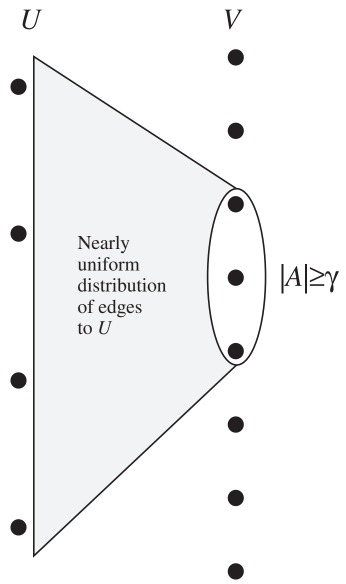

Figure 5-8: Bipartite graph with extractor property

Lemma 5.5.1 *Consider a bipartite graph G on $U \times V$ with the (ϵ, γ)-extractor property. Choose $A \subseteq V$ with $|A| \geq \gamma$. Let $f \in (0, 1]$. Let $X \subseteq U$ be the set of nodes in U such that for $x \in X$, $d_A(x) \geq (1 + \epsilon/f)\overline{d_A}$. Let $Y \subseteq U$ be the set of nodes in U such that for $y \in Y$, $d_A(y) \leq (1 - \epsilon/f)\overline{d_A}$. Then $|X| \leq f|U|$ and $|Y| \leq f|U|$.*

Proof: We prove the lemma for the set X. The proof for the set Y is symmetric. Suppose $|X| > f|U|$. Choose an edge e uniformly among those emanating from A.

$$\begin{aligned} Pr[e \text{ has an edge in } X] &= \frac{\sum_{x \in X} d_A(x)}{\sum_{u \in U} d_A(u)} \\ &\geq \frac{|X|\overline{d_A}(1 + \epsilon/f)}{|U|\overline{d_A}} \\ &= \frac{|X|}{|U|} + \frac{\epsilon}{f}\frac{|X|}{|U|} \\ &> \frac{|X|}{|U|} + \epsilon \end{aligned}$$

However, this means that G cannot have the (ϵ, γ)-extractor property, a contradiction.

■

Corollary 5.5.1.1 *Consider a bipartite graph G on $U \times V$ with the (ϵ, γ)-extractor property. Choose $A \subseteq V$ with $|A| \geq \gamma$. Let $X \subseteq U$ be the set of nodes in U such that* *for $x \in X$, $d_A(x) \geq (1+\sqrt{\epsilon})\overline{d_A}$. Let $Y \subseteq U$ be the set of nodes in U such that for $y \in Y$, $d_A(y) \leq (1-\sqrt{\epsilon})\overline{d_A}$. Then $|X| \leq \sqrt{\epsilon}|U|$ and $|Y| \leq \sqrt{\epsilon}|U|$.*

Proof: Immediate from Lemma 5.5.1 with $f = \sqrt{\epsilon}$. ∎

Lemma 5.5.2 *If $A \subseteq V$ and $|A| \geq \gamma$ then no more than a $3\sqrt{\epsilon} - 2\epsilon$ fraction of edges emanating from A have endpoints in X or Y where X and Y are defined in the previous corollary.*

Proof: The number of edges emanating from A is $|U|\overline{d_A}$. At least $(|U| - |X| - |Y|)\overline{d_A}(1-\sqrt{\epsilon})$ of these edges do not have endpoints in X or Y by Corollary 5.5.1.1. So the total number of edges leading from A to X or Y is at most

$$\begin{aligned} |U|\overline{d_A} - (|U| - |X| - |Y|)\overline{d_A}(1-\sqrt{\epsilon}) &\leq |U|\overline{d_A}(1-(1-2\sqrt{\epsilon})(1-\sqrt{\epsilon})) \\ &= |U|\overline{d_A}(3\sqrt{\epsilon}-2\epsilon) \\ &= (\#\text{ of edges connected to } A)(3\sqrt{\epsilon}-2\epsilon) \end{aligned}$$

∎

We now consider the number of tokens entering each of the bitonic counting networks when the permutation Π is defined by a bipartite graph with the (ϵ,γ)-extractor property.

Lemma 5.5.3 *Let $G = U \times V$ be a bipartite graph with the (ϵ,γ)-extractor property. Furthermore, suppose G is used to define the permutation Π. So $|V| = 2^{d-d'}$. Consider $A \subseteq V$ with $|A| \geq \gamma$. Let μ be the average number of tokens per wire in the network and let μ' be the average number of tokens per wire entering the bitonic counting networks represented by A. Then $\mu - 2\sqrt{\epsilon}d \leq \mu' \leq \mu + 4\sqrt{\epsilon}d$.*

Proof: Let t_u denote the number of tokens output on each wire of the recursive counting network represented by $u \in U$ in the graph under the simplifying assumption. Let X and Y be defined as in Corollary 5.5.1.1. The number of tokens received by the bitonic counting network represented by A is

$$\begin{aligned}\sum_{u\in U} d_A(u)t_u &= \overline{d_A}\sum_{u\in U} t_u + \sum_{u\in U}(d_A(u)-\overline{d_A})t_u \\ &= \overline{d_A}\sum_{u\in U} t_u + \sum_{u\in X\cup Y}(d_A(u)-\overline{d_A})t_u + \sum_{u\in U\setminus\{X\cup Y\}}(d_A(u)-\overline{d_A})t_u\end{aligned}$$

If this total were $\overline{d_A}\sum_{u\in U} t_u$, then the average number of tokens per wire entering A would be μ as desired. So the remaining terms in the sum are the "error" terms. We bound each of these terms. By Lemma 5.5.2, $\sum_{u\in X\cup Y}(d_A(u)-\overline{d_A}) \le (3\sqrt{\epsilon}-2\epsilon)\overline{d_A}|U|$ and since $t_u \le d$,

$$\sum_{u\in X\cup Y}(d_A(u)-\overline{d_A})t_u \le (3\sqrt{\epsilon})\overline{d_A}|U|d$$

Also, by using Corollary 5.5.1.1 and the fact that $d_A(u)-\overline{d_A} \ge -\overline{d_A}$ we obtain the lower bound

$$\begin{aligned}\sum_{u\in X\cup Y}(d_A(u)-\overline{d_A})t_u &\ge \sum_{u\in Y}(d_A(u)-\overline{d_A})t_u \\ &\ge -\overline{d_A}|U|\sqrt{\epsilon}d\end{aligned}$$

To bound the last sum in the "error" term, we note that by the definition of X and Y, $|d_A(u)-\overline{d_A}| \le \overline{d_A}\sqrt{\epsilon}$ for $u \in U\setminus\{X\cup Y\}$. So, $|\sum_{U\setminus\{X\cup Y\}}(d_A(u)-\overline{d_A})t_u| \le \overline{d_A}|U|\sqrt{\epsilon}d$.

Thus,

$$\overline{d_A}\sum_{u\in U} t_u - 2\overline{d_A}|U|\sqrt{\epsilon}d \le \overline{d_A}\sum_{u\in U} t_u + \text{ error term } \le \overline{d_A}\sum_{u\in U} t_u + 4\overline{d_A}|U|\sqrt{\epsilon}d$$

Since $\overline{d_A}|U|$ is the total number of edges entering A, our bounds are met. ∎

We now select parameters for the regular bipartite graph $G = U \times V$. $|U| = 2^{\Omega(d)}$ and $|V| = 2^{d-d'}$ for some $d' = \lceil\sqrt{d}\rceil$. What is required for the size of $|U|$ is that it be sufficiently small to make construction of the permutation possible in polynomial-time. A more precise specification for $|U|$ is given in the next section where the construction of the permutation is described. These values correspond to those used in the section containing the existential proof of the $O(\lg n)$-depth counting network. The values for γ and ϵ so that Π has the desired properties are as follows: $|X|+|Y| \leq 2\gamma$ is the number of bitonic counting networks that may receive far too many or far to few tokens where X and Y are defined in Corollary 5.5.1.1. So we choose $\gamma = 2^{\lfloor \alpha d \rfloor - 1}$ where $0 < \alpha < 1/2$. By choosing $\epsilon = \frac{1}{64d^2}$, we are assured that the average number of tokens per wire in the remaining $2^{d-d'} - 2\gamma$ bitonic counting networks is at most $1/2$ from the overall average since the average will be off by at most $4\sqrt{\epsilon}d = 1/2$. Thus, the output of the bitonic networks is α_0-nice as defined in Section 5.2.

5.5.1 Construction of the bipartite graph with the extractor property

We are now left to show that we can construct a regular bipartite graph $G = U \times V$ with the appropriate extractor property in polynomial-time. We draw upon the work of [25] and [29]. Recall definitions 5.5.1, 5.5.2, and 5.5.3.

The authors show that in polynomial-time it is possible to construct an extractor with the following properties:

Lemma 5.5.4 *For any parameter $\gamma = \gamma(r)$ and $\epsilon = \epsilon(r)$ with $1/r \leq \gamma \leq 1/2$ and $2^{-\gamma r} \leq \epsilon \leq 1/r$, there exists an easily computable (and explicitly given) (ϵ, γ)-extractor $E : \{0,1\}^r \times \{0,1\}^t \to \{0,1\}^s$, where $t = O(\lg \epsilon^{-1} \lg^2 r/\gamma^2)$ and $s = \Omega(\gamma^2 r/\lg \gamma^{-1})$.*

In [29], the authors use the extractor above to generate a bipartite graph. From this bipartite graph they then construct a graph with high degree and extremely strong expansion properties. The initial bipartite graph they construct is the first step in our construction, as well.

The extractor defines a very natural mapping to a bipartite graph $U \times V$. Given an (ϵ, γ)-extractor, we create 2^r nodes we call V labeled with each element of $\{0,1\}^r$, and 2^s nodes we call U labeled with each element of $\{0,1\}^s$. The function E defines the edge set in the graph: there is an edge between a node $v \in V$ and a node $u \in U$ iff $\exists e \in \{0,1\}^t$ such that $E(v, e) = u$. As the authors note in [29] as t becomes larger or s becomes smaller, the construction of the extractor becomes easier. One can increase t by increasing the multiplicity of edges in the graph. One can decrease s by partitioning the nodes in U into blocks of the same size and treating each block as a single node. In both of the operations, the graph retains the same extractor properties.

This construction produces a bipartite graph G with the $(\epsilon, 2^{r\gamma})$-extractor property. For the appropriate choices of r, s, t, ϵ, and γ, this construction nearly meets the criteria for our bipartite extractor. However, it is lacking in one important respect. Though the nodes of V are regular (each node has degree 2^t), the nodes of U are not. We describe the algorithm for modifying the graph to correct this problem and then provide the analysis to prove correctness. Our construction is described for arbitrary ϵ and γ. At the end of the section we describe the particular values of these variables which allow the permutation Π to have the necessary properties.

We define $\overline{d(B)}$ to be the average degree of nodes in the set B. In the step by step process we generate graphs $G_i = U_i \times V_i$ in Step i.

Step 1: Construct a bipartite graph G_1 from the extractor function as described above. G_1 has the (ϵ, γ)-extractor property for some ϵ and γ. The number of nodes in U_1 and V_1 will be even for the purposes of our construction and the degrees of nodes in V_1 will be powers of 2 as well.

Step 2: Remove nodes from U_1 with degrees significantly greater (smaller) than the average degree. Specifically, remove nodes with degree more than $(1+4\epsilon)\overline{d(U_1)}$ or less than $(1-4\epsilon)\overline{d(U_1)}$. If fewer than half the nodes have been removed, arbitrarily remove nodes until half remain. The resulting graph is G_2.

Step 3: Make V_3 as close to regular as possible (up to divisibility) by moving endpoints of edges from nodes of V_2 with too high a degree to those with too low a degree until degrees of all nodes of V_2 are within 1 of one another. This can be done in an arbitrary manner. The resulting graph is G_3.

Step 4: Let $d_{max}(U_3)$ be the maximum degree of any node in U_3. So $d_{max}(U_3) \leq \left\lfloor (1+4\epsilon)\overline{d(U_1)} \right\rfloor$. Add edges to nodes in U_3 until all nodes have degree $\left\lfloor (1+4\epsilon)\overline{d(U_1)} \right\rfloor$.

The endpoints of these edges in V_3 are constructed to keep the nodes of V_3 as regular as possible. The resulting graph is G_4.

Step 5: Remove edges maintaining regularity of U_4 and ensuring that all nodes of V_4 have degrees within 2 of one another until degree per node of U_5 is $\overline{d(U_1)}$. Degrees of nodes of V_5 will be $\overline{d(V_1)}/2-1, \overline{d(V_1)}/2$, or $\overline{d(V_1)}/2+1$. The resulting graph is G_5.

Step 6: Make V_5 regular. The desired degree of the nodes of V_6 is $\overline{d(V_1)}/2$. Some nodes will have degree $\overline{d(V_1)}-1$ and the same number will have degree $\overline{d(V_1)}+1$. Move the endpoint of an edge connected to the node of high degree to one of low degree. Repeat until V_6 is regular.

We now analyze the algorithm and show that G_6 has the desired properties maintains strong extractor properties and is regular.

Lemma 5.5.5 $|U_2| = |U_1|/2$.

Proof: We need only show that not too many nodes have degrees far from the average. This is immediate from Lemma 5.5.1 with $f = 1/4$. ∎

For the remainder of this section we assume ϵ a constant nonzero amount less than 1/4. This will be the case for the construction of the permutation.

Lemma 5.5.6 *G_2 has the $(O(\epsilon), \gamma)$-extractor property.*

Proof: Consider graph G_1. Choose a set $A \subseteq V_1$ with $|A| \geq \gamma$. Choose $X \subseteq U_2$. Let E_X be the number of edges entering X from A. Since G_1 has the (ϵ, γ)-extractor property, we know that $\frac{|X|}{|U_1|} - \epsilon \leq \frac{E_X}{|U_1|\overline{d(U_1)}} \leq \frac{|X|}{|U_1|} + \epsilon$. Now consider the graph G_2. There are still E_X edges leading from A to X but the total number of edges leading into U_2 denoted by E_{U_2} is bounded by $\frac{|U_1|}{2}(1-4\epsilon)\overline{d(U_1)} \leq E_{U_2} \leq \frac{|U_1|}{2}(1+4\epsilon)\overline{d(U_1)}$. Thus the probability an edge chosen uniformly out of A strikes X in G_2 is bounded above by

$$\frac{E_X}{|U_1|\overline{d(U_1)}(1-4\epsilon)/2} \leq \frac{2\left(\frac{|X|}{|U_1|}+\epsilon\right)}{1-4\epsilon}$$

$$
\begin{aligned}
&= \frac{|X|}{|U_2|} + \frac{4\epsilon\frac{|X|}{|U_2|}}{1-4\epsilon} + \frac{2\epsilon}{1-4\epsilon} \\
&= \frac{|X|}{|U_2|} + O(\epsilon).
\end{aligned}
$$

The probability is bounded below by

$$
\begin{aligned}
\frac{E_X}{|U_1|\overline{d(U_1)}(1+4\epsilon)/2} &\geq \frac{2\left(\frac{|X|}{|U_1|}-\epsilon\right)}{1+4\epsilon} \\
&= \frac{|X|}{|U_2|} - \frac{4\epsilon\frac{|X|}{|U_2|}}{1+4\epsilon} - \frac{2\epsilon}{1+4\epsilon} \\
&= \frac{|X|}{|U_2|} - O(\epsilon).
\end{aligned}
$$

■

We now show that removing nodes from U_1 does not have a detrimental effect on the degree of nodes in V_2.

Lemma 5.5.7 *Let M be the nodes of V_2 with degree greater than $(1/2+\epsilon)\overline{d(V_1)}$. Let m be the nodes of V_2 with degree less than $(1/2-\epsilon)\overline{d(V_1)}$. Then $|M| < \gamma$ and $|m| < \gamma$ (see Figure 5-9).*

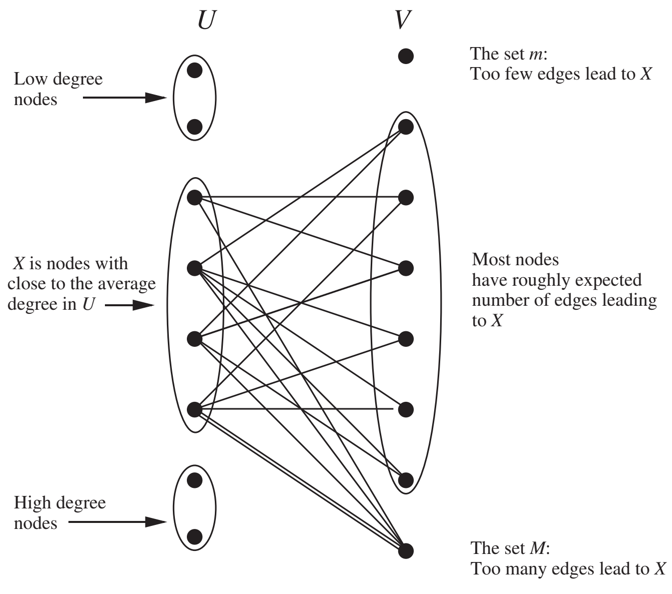

Figure 5-9: Most nodes in V have roughly half original degree after Step 2.

Proof: Let $A \subseteq V_1$ be the set with $|A| = \gamma$ and the largest number of edges going to $X = U_2$ in G_1. In G_1, the probability that an edge leading from A strikes X is no more than $1/2 + \epsilon$. Since all nodes of V_1 have degree $\overline{d(V_1)}$, at most $\gamma(1/2 + \epsilon)\overline{d(V_1)}$ edges lead from A to X. Therefore there cannot be γ nodes in M. By a symmetric argument the same holds for m. ■

For the remainder of this section we assume that $(1/2 - \epsilon)\frac{\overline{d(V_1)}\gamma}{|V_1|} \leq 1$. Again in our construction of our permutation this bound will hold.

Lemma 5.5.8 $(1/2 - \epsilon)\overline{d(V_1)} - 1 \leq \overline{d(V_2)} \leq (1/2 + \epsilon)\overline{d(V_1)} + 1$

Proof: We prove the upper bound. The lower bound is analogous. At most γ nodes of V_2 have degree greater than $(1/2 + \epsilon)\overline{d(V_1)}$. In addition, these γ nodes can have degree at most $\overline{d(V_1)}$. Thus the average number of edges per node in V_2 is at most $\frac{(1/2+\epsilon)\overline{d(V_1)}(|V_1|-\gamma)+\overline{d(V_1)}\gamma}{|V_1|} \leq (1/2 + \epsilon)\overline{d(V_1)} + 1$ ■

At this point in the construction, the graph still has the desired extractor property. In addition, the nodes of U_2 and V_2 are nearly regular. To make V_2 regular, we perform the following algorithm:

Let $|E|$ be the number of edges in G_2. When there is a node $x \in V_2$ with degree greater than $\lceil |E|/|V_2| \rceil$, there is another node $y \in V_2$ with degree less than $\lfloor |E|/|V_2| \rfloor$. Move an edge with endpoint equal to x so that this endpoint is now y. After repeating this process a polynomial number of times, the new graph G_3 will be regular (up to divisibility). We now examine how this algorithm affects the extractor property.

Lemma 5.5.9 *G_3 has the $(O(\epsilon) + O(1/k), k\gamma)$-extractor property for any k with $k >$ 3.*

Proof: Consider a set $A \subseteq V_3$ with $|A| \geq k\gamma$.

In the worst case, the 2γ nodes from M and m defined in Lemma 5.5.7 are contained in A and they either direct all their edges to X or direct none of their edges to X. These edges account for at most a $2/(k-2)$ fraction of the edges leading from A so they affect the probability that an edge from A strikes X by at most $O(1/k)$. Each of the remaining $(k-2)\gamma$ nodes in A have their edge sets changed only slightly in the the transformation from G_2 to G_3. In G_2, these nodes begin with between $(1/2-\epsilon)\overline{d(V_1)}-1$ edges and $(1/2+\epsilon)\overline{d(V_1)}+1$ edges. In G_3 they are in this range as well. Thus the edges removed from or added to a node change its edge set by a factor of at most $\frac{(1/2+\epsilon)\overline{d(V_1)}+1}{(1/2-\epsilon)\overline{d(V_1)}-1} = 1 + O(\epsilon)$. As a result, in the transformation from G_2 to G_3, these $(k-2)\gamma$ nodes can change (in terms of G_2 and G_3) the probability that an edge from A strikes X by an additive factor of at most $O(\epsilon)$. This yields the result. ∎

We now consider the extractor property of G_4.

Lemma 5.5.10 *G_4 has the $(O(\epsilon)+O(1/k), k\gamma)$-extractor property.*

Proof: The minimum degree of any node in U_3 is $(1-4\epsilon)\overline{d(U_1)}$. So adding edges increases the number of edges by a factor of at most $\frac{\lfloor (1+4\epsilon)\overline{d(U_1)} \rfloor}{(1-4\epsilon)\overline{d(U_1)}} = 1 + O(\epsilon)$. Thus, the degrees of the nodes of V_3 increase by at most this factor, as well. This means that for any set $A \subseteq V_4$, the fraction of edges leading to a set $X \subseteq U_4$ increases or decreases by an additive amount of at most $O(\epsilon)$. ∎

We now go into more detail in describing and analyzing Step 5. To remove the appropriate edges, we solve a max-flow problem using the Edmonds-Karp polynomial-time algorithm [12]. We begin with graph G_4 and form a directed graph from it. From the construction all the nodes of U_4 have degree $d_{start}(U_4) = \left\lfloor (1+4\epsilon)\overline{d(U_1)} \right\rfloor$ and all the nodes of V_4 have degree either $\lfloor d_{start}(V_4) \rfloor$ or $\lceil d_{start}(V_4) \rceil$ where $d_{start}(V_4) = \frac{|V_4|}{|U_4|} d_{start}(U_4)$. At the end of Step 5 the graph will have nodes U_5 with degrees equal to $d_{end}(U_4) = \overline{d(U_1)}$ and V_5 with degrees equal to $d_{end}(V_4)-1$, $d_{end}(V_4)$, or $d_{end}(V_4)+1$, where

$$\begin{aligned} d_{end}(V_4) &= \frac{|U_4|}{|V_4|}\overline{d(U_1)} \\ &= \frac{|U_1|/2}{|V_1|}\overline{d(U_1)} \\ &= \overline{d(V_1)}/2. \end{aligned}$$

We define the following max-flow problem: Direct all edges in G_4 toward V_4. Set the lower capacity to 0 and the upper capacity to 1. Add a source node s. Direct an edge from s to every node in U_4. Each of these edges will have capacity exactly $d_{end}(U_4)$. Add a sink node t. Direct an edge from each node in V_4 to t. Each of these edges will have lower capacity $d_{end}(V_4) - 1$ and upper capacity $d_{end}(V_4) + 1$. (see Figure 5-10).

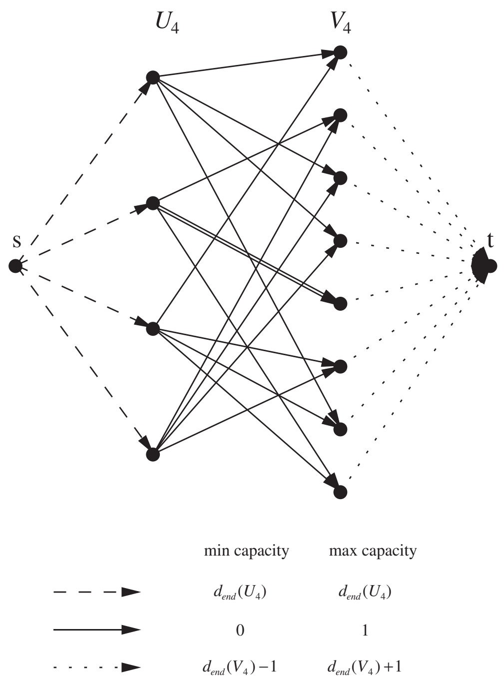

Figure 5-10: Flow problem defined in Step 5.

We first note that we can find a noninteger solution to this problem. We show the existence of a feasible solution by allowing flow $d_{end}(V_4)/d_{start}(V_4)$ along each edge in the nodes leading from U_4 to V_4. This clearly satisfies the constraints of the edges in G_4. In addition, the degree of nodes in U_4 is $d_{start}(U_4)$, so the flow along edges from s to nodes in U_4 will be

$$\begin{aligned} d_{start}(U_4)\frac{d_{end}(V_4)}{d_{start}(V_4)} &= \frac{|U_4|}{|V_4|}d_{end}(V_4) \\ &= \overline{d(U_1)} \\ &= d_{end}(U_4) \end{aligned}$$

as desired. Lastly, we consider the flow along edges leading from V_4 to t. All nodes in

V_4 have degree either $\lfloor d_{start}(V_4) \rfloor$ or $\lceil d_{start}(V_4) \rceil$. Thus flow along each edge to t will be in the range between $\frac{d_{end}(V_4)}{d_{start}(V_4)} \lfloor d_{start}(V_4) \rfloor$ and $\frac{d_{end}(V_4)}{d_{start}(V_4)} \lceil d_{start}(V_4) \rceil$. But these values are between $d_{end}(V_4) - 1$ and$d_{end}(V_4) + 1$, as required. By the Integrality Theorem in [14], because the problem has a fractional solution, we know the problem has an integral solution and, in fact, it can be found in polynomial-time. When the edges with positive flow in the integer solution are kept, G_5 is formed.

Lemma 5.5.11 *G_5 has the $(O(\epsilon) + O(1/k), k\gamma)$-extractor property.*

Proof: This proof is similar to the proof of Lemma 5.5.10. However, instead of adding edges we are removing them. All nodes of U_4 have degree $\left\lfloor (1 + 4\epsilon)\overline{d(U_1)} \right\rfloor$. In this step their degrees are reduced to $\overline{d(U_1)}$. So the degrees are reduced by a factor of $1 + O(\epsilon)$. Similarly, the degrees of the nodes in V_4 are also reduced by a factor of $1 + O(\epsilon)$. In the worst case, the edges eliminated from some set $A \subseteq V_4$ were all directed to some set $X \subseteq U_4$ or they were all directed away from X. This changes the probability of an edge from A hitting X by at most an additive factor of $O(\epsilon)$. ■

Lastly we show that Step 6 does not destroy the extractor property.

Lemma 5.5.12 *G_6 has the $(O(1/\overline{d(V_1)}) + O(\epsilon) + O(1/k), k\gamma)$-extractor property.*

Proof: G_5 has the $(O(\epsilon) + O(1/k), k\gamma)$-extractor property. Consider a set $A \subseteq V_6$ with $|A| \geq k\gamma$ and a set $X \subseteq U_6$. In transforming G_5 to G_6, each node of A changes by at most one edge, which is at most a $1/(\overline{d(V_1)} - 1)$ fraction of its edges. ■

We now construct the required regular bipartite graph corresponding to the permutation Π. Recall that $|U| = 2^{\Omega(d)}$, $|V| = 2^{d-d'}$, and G_6 must have the $(\frac{1}{64d^2}, 2^{\lfloor \alpha d \rfloor - 1})$-extractor property.

Given these parameters, we will construct an irregular bipartite graph G_1 with $|U_1| = 2|U|$, $|V_1| = |V|$, and it will have the $(1/d^3, 2^{\lfloor \alpha d \rfloor - 1}/d^3)$- extractor property. After converting this graph to a regular graph G_6, the graph will have the desired number of nodes and edges and have the $(O(1/d^3), 2^{\alpha d - 1})$-extractor property which is slightly stronger than necessary. Note that to achieve this result, we use $k = d^3$ during the analysis (Lemma 5.5.9).

In terms of the Nisan/Zuckerman extractor, we therefore have $r = d - d'$, $s = \Omega(d)$, $t = 1 + d'$, $\epsilon = 1/d^3$, $\gamma = \alpha + o(1)$.

These values meet the criteria of Lemma 5.5.4. Namely, since $\alpha < 1/2$, $\gamma < 1/2$. Also, $\epsilon \approx 1/r^{3/2} < 1/r$. To see that t is sufficiently large we note that $\lg \epsilon^{-1} \lg^2 r/\gamma^2 = O(\lg^3 d) = o(t)$ And finally, we set $s = \Omega(d)$ sufficiently small so that $s < O(\gamma^2 r/\lg \gamma^{-1})$ as defined in Lemma 5.5.4.

5.6 Smoothing to within $O(\lg \lg n)$

The construction in the previous section smooths the input and is an $O(\lg n)$-depth network. However, the constant involved in the Big-Oh notation is large due to the presence of the AKS balancing network. In this section we show that when the AKS balancing network is removed from the construction above, what is left is a network which $O(\lg \lg n)$-smooths. The butterfly balancing network constructed in Section 5.1.1 is $\lg n$-depth and $\lg n$-smooths. The hope is that this new construction will shed some light on possible approaches to constructing an $O(\lg n)$-depth network which $O(1)$-smooths but does not depend on sorting so heavily that the use of the AKS balancing network is required.

The construction above makes use of the AKS balancing network in one portion of the network. Namely, in the section which requires a 3-smoother to finish the smoothing. We now examine the construction above without the use of the 3-smoother.

Theorem 5.6.1 *When the construction in Section 5.5 is made without the 3-smoother phase, the network $O(\lg \lg n)$-smooths.*

Proof: This is a proof by induction. The construction in Section 5.5 is recursive. We show that at the i^{th} level of recursion, the network $3i$-smooths. Since there are $O(\lg \lg n)$ levels of recursion, this completes the proof.

Base Case: This is the first level of recursion. We know that the input is smoothed to within 3 when the 3-smoother in the construction of the previous section is reached.

Inductive Step: By the inductive hypothesis after the i^{th} level of recursion, the network $3i$-smooths. We examine the $i+1^{st}$ level. This consists of level i networks input to the final stage network. Let each of the level i networks be labelled from 1 to j. Let the m^{th} such network output between x_m and $x_m + 3i$ tokens. We invoke Theorem 2.1.1 which allows us to pass the tokens through the network in any desired order. First we allow x_m tokens on each wire from the m^{th} level i network to pass through the final stage. By our analysis in the previous section, upon reaching the 3-smoother in that construction, this input is 3-smoothed. There are now at most $3i$ tokens per wire left to enter the final stage. As a result, these tokens can increase the smoothed property by at most $3i$. Thus, the output is $3+3i$-smoothed as desired.

■

Chapter 6

Other directions

6.1 Other modifications and analysis of the counting network model

The emphasis of our work on counting networks involved finding a network of minimum depth for the given number of input/output wires. This result is of practical significance because the latency before a request is fulfilled is lower bounded by the depth (maximum number of balancers a token may have to pass through to get from an input wire to an output wire). However, contention in the network also plays a major role in determining delays for a request to be fulfilled. If several tokens enter a balancer at the same time, they are required to leave one at a time. This produces a sequential bottleneck. In [11], Dwork and Herlihy broach this subject by introducing a model for measuring contention in distributed systems to help evaluate the latency expected in various algorithms including counting network implementations. In their model, the contention of an algorithm is measured by counting the worst case number of *stalls* the algorithm may suffer in a given run. A *stall* occurs whenever a processor is delayed because it attempted to access the same memory location as another processor. For example, if k processors attempt to access the same memory location simultaneously, the algorithm will suffer $k-1$ stalls during that step because only one of the processors will be successful in accessing the location.

In the case of data structures, such as counting networks, the authors define the *amortized contention* to be the worst case contention divided by the number of requests made to the object. Thus, this value measures the average contention per request. Because this is a worst case measure, an adversary is involved. With respect to counting networks, this adversary has control over how many tokens enter each input wire and the order with which the tokens pass through the network.

The authors show that the amortized contention of a bitonic counting network with n-inputs and n-outputs which is implemented using p processors is $\Theta(\frac{p}{n}\lg^2 n)$ as the number of tokens input to the network approaches infinity. They do this by showing that balancers at depth i for any i can have amortized contention at most $O(p/n)$. Because there are $\lg^2 n$ levels, the result follows. They also note that if the same number of tokens are input to each wire of the network then this amortized contention bound can be met. For the case $p = n$, the amortized contention is equivalent to the depth of the network (up to constant factors) and so the depth plays as significant a role in latency as the amortized contention.

In [2], Aiello, Venkatesan, and Yung continue this contention analysis, obtaining bounds on other counting networks and on the bitonic network. Namely, they show bounds on the number of tokens which must enter the bitonic network before the $\Theta(\frac{p}{n}\lg^2 n)$ bound in [11] holds. They also examine the butterfly balancing network described in Section 5.1. They show that when a sufficient number of tokens have entered such an n-input n-output butterfly network which is implemented on a p processor machine, its amortized contention is $\lg n(4\frac{p}{n} + \lg n + 1)$. They then use the property of the butterfly, namely that it $\lg n$-smooths, to show that any network which begins with the butterfly balancing network has amortized contention $O((d + \lg n)(\frac{p}{n} + \lg n))$ where d is the depth of the network, when sufficiently many tokens enter the network. Because our $O(\lg n)$-depth counting network is preceded by a butterfly balancing network, for $p = \Omega(n \lg n)$ our network has optimal amortized contention.

In addition to their analysis of amortized contention in counting networks, the authors in [2] examine a number of variants of counting networks. In one variant, the authors allow *dispersers* and *combiners* in addition to balancers to be used to construct the network. A *disperser* is a 1-input 2-output balancer, while a *combiner* is a 2-input 1-output objects which outputs the tokens entering it on its single output wire (see Figure 6-1). From these objects, the authors construct a simple $O(\lg n)$-depth network using tools presented earlier in this thesis. They begin with an n-input butterfly balancing network. They then use $\lg \lg n$ levels of dispersers to distribute each output wire of the butterfly balancing network among $\lg n$ wires. Note that this increases the width of the network from n to $n \lg n$. These dispersers have the affect of 2-smoothing the $n \lg n$ wires. The 2-smoother from Section 5.4 is used to smooth this input. Next, the AKS balancing network is applied to count the input. And finally, $\lg \lg n$ levels of combiners are used to bring the width of the network back to n (wires which are the same height, modulo n, are combined to form a single output wire).

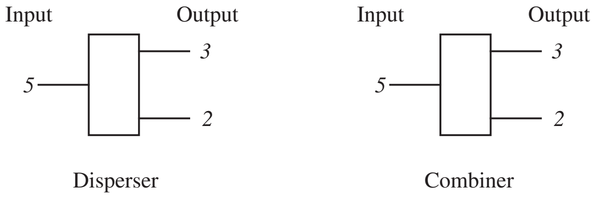

Figure 6-1: A disperser and a combiner.

In addition to deterministic constructions, the authors also consider balancers which begin in a random state. Known as *r-balancers*, these objects act as regular balancers, but output their first token along the top wire with probability 1/2 and along the bottom wire with probability 1/2. Using r-balancers, the objective is to construct a small-depth balancing network which counts any input with high probability. The authors construct a $\lg n + O(\lg^2 \lg n)$-depth network which 2-smooths any input with $1 - \frac{1}{superpoly(n)}$ probability. What is so nice about this result is the small constant in the $O(\lg n)$-depth network. The means by which they achieve this result is to begin with the first $\lg n - c \lg \lg n$ levels of the butterfly balancing network with r-balancers rather than balancers where c is some constant. By using only the first

$\lg n - c\lg\lg n$ levels of the butterfly, $2n/\lg^c n$ $\lg^c n/2$-input butterflies have been cut off from the end of the complete butterfly. Instead, each of these sub-butterflies is replaced with a $\lg^c n/2$-input bitonic counter where the balancers used are deterministic. The use of the randomized balancers in the initial levels of the butterfly ensure that the number of tokens enter each bitonic counter is roughly the same, with high probability. This is shown using induction on depth and Hoeffding's Inequality [20]. The bitonic counters then count their respective shapes resulting in a nearly smoothed shape.

In addition to discussions of randomized balancers and issues of contention, in [2] the authors discuss constructions involving expanders and models involving tokens with weights. We will not delve into these subjects in this thesis, however.

In [1], Aharonson and Attiya present impossibility results which we discussed in Section 3.2. Because their result shows that a counting network with n inputs cannot be constructed from 2-balancers if n is not a power of 2, the authors address the issue of constructing a network with n output wires where n is not a power of 2 by designing a slightly modified model of counting networks. The new model allows output wires to feed back into the input wires. They construct the desired network by beginning with a counting network with $2^{\lceil \lg n \rceil}$ input and output wires and then feeding $2^{\lceil \lg n \rceil} - n$ wires back into the original input wires (see Figure 6-2). The authors argue that if a finite number of tokens enter such a network that they exit the network within a finite period of time. This is apparent from the fact that a token recycled through the system is no different than a newly entered token and the fact that $2^{\lceil \lg n \rceil} = O(n)$.

In [19], Herlihy, Shavit and Waarts introduce the notion of a *linearizable counting network*. Such a network ensures that if a token has left the counting network and has been assigned a value i and at some later time another token enters the network and then exits with the value j. Then $i < j$ always. With the standard notion of a counting network, one cannot hope to create a linearizable counting network. So again, with some modifications to the model, the authors are able to construct such a network.

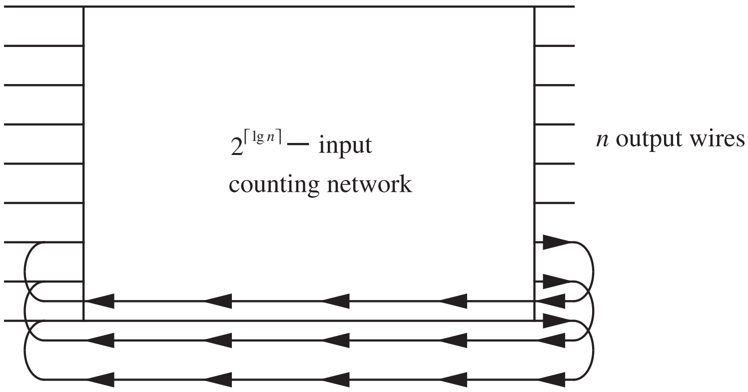

Figure 6-2: n output counting network where n is not a power of 2.

6.2 Other approaches to counting

In addition to the single variable Fetch&Add and counting network solutions to shared counting in a distributed system, other promising software approaches have been offered to solve this problem. Two of the more notable are *diffracting trees* and *software combining trees*.

In [27], Shavit and Zemach recently introduced a model called the *diffracting tree* model. This model uses objects very much like the dispersers defined above. However the initial state of the object is set to output the first token on the top wire with probability 1/2 and output it on the bottom wire with probability 1/2. We will call these objects r-dispersers. A big gain in their model with respect to reducing latency is the particular implementation they use in software. They note that if two tokens enter a r-disperser then the toggle bit need not be flipped but may remain in its initial state. The two tokens may simply exit along different wires. Their approach is to reduce the number of times toggle bits need to be flipped. For each balancer, an array is kept. When a token enters a r-disperser it chooses an element of this array uniformly at random and hopes that there is another token waiting in the same element, as well. If so, the two tokens can continue without having to change the toggle bit by leaving along each output wire. Otherwise, a token waits at the element of the array for a designated period of time before the processor responsible for the token attempts to flip the toggle bit. The authors show, in [27], that experimentally the diffracting tree performs quite well.

In [16] Goodman, Vernon, Woest use a binary tree data structure to implement a shared counter. The data structure is what they call a *software combining tree.* The processors reside at the leaves of the binary tree and at any time a processor can perform a fetch and add request by sending the request up the tree. The values requested are added up the tree until the root is reached where the current value of the counter is stored. If a requests were made in the left subtree under the root, b were made in the right subtree and c is the initial value of the counter, then the root, which holds the current counter value, updates its value to $c + a + b$, it passes the values c to the left subtree and $c + a$ to the right subtree. The values continue down in a recursive fashion until the leaves receive their updated values. In [18], the authors show that software combining tree and counting networks are similar in their efficiency, though when contention becomes high the counting network tends to outperform the software combining tree.

Chapter 7

Conclusions

This thesis has made substantial progress towards improving our understanding of the depth complexity of counting networks. In Chapter 5, we established the existence of an optimal $O(\lg n)$-depth counting networks and presented an uniform polynomial-time construction for such a network.

The Big-Oh depth bound for our optimal-depth counting network construction hides an unrealistically large multiplicative constant, largely due to the fact that the AKS sorting network is used as a subroutine. Of course, we cannot hope to improve this constant without also improving the sorting constant, since every counting network corresponds to a sorting network with the same depth/topology (by replacing each balancer with a comparator). With regard to the construction of smoothing networks, however, the situation is not so clear. In view of the fact that every smoothing network produced thus far has incorporated a "sorting" network as a primary component, it would be interesting to either bound the depth complexity of sorting by a small constant times the depth complexity of smoothing, or to construct a small-depth smoothing network that makes no use of sorting networks. In Section 5.6 we constructed an $O(\lg n)$-depth balancing network which $O(\lg \lg n)$-smooths. This network does not incorporate AKS and so has more reasonable constants. This construction may be a step toward constructing a smoothing network without the use of AKS.

We have shown impossibility results showing restrictions on the possible widths of a counting network. Namely, that a counting network must have 2^k input and output wires for some positive integer k. We also considered more general networks introduced in [1] and reviewed similar restrictions found in that paper. In Section 4.5 we constructed networks of width n from balancers of size $\{p_1, p_2, \cdots, p_k\}$ where $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$. This in some sense establishes the "tightness" of the impossibility result in [1]. The analysis of these impossibility results also leads to a method of testing balancing networks to determine if they are, in fact, smoothing or counting networks. This test is made in the same spirit as the 0-1 sorting lemma test for sorting networks, though the balancing network version is far weaker.

Bibliography

[1] E. Aharonson and H. Attiya. Counting networks with arbitrary fan-out. In *Proceedings of the 3rd ACM-SIAM Symposium on Discrete Algorithms*, pages 104–113, January 1992.

[2] W. Aiello, R. Venkatesan, and M. Yung. Asynchronous low latency counting and smoothing networks. Unpublished manuscript, 1993.

[3] M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. *Combinatorica*, 3:1–19, 1983.

[4] N. Alon and J. H. Spencer. *The Probabilistic Method.* Wiley-Interscience, New York, NY, 1992.

[5] J. Aspnes, M.P. Herlihy, and N. Shavit. Counting networks and multi-processor coordination. In *Proceedings of the 23rd Annual Symposium on Theory of Computing*, pages 348–358, May 1991.

[6] K. E. Batcher. Sorting networks and their applications. In *Proceedings of the AFIPS Spring Joint Computer Conference,* vol. 32, pages 307–314, 1968.

[7] C. Busch and M. Mavronicolas. A combinatorial treatment of balancing networks. In *Proceedings of the 13th ACM Symposium on Principles of Distributed Computation*, 1994. To appear.

[8] V. Chvátal. Lecture notes on the new AKS sorting network. Technical Report 92-29, DIMACS Center for Discrete Mathematics and Theoretical Computer Science, June 1992.

[9] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

[10] M. Dowd, Y. Perl, M. Saks, and L. Rudolph. The balanced sorting network. Technical Report DCS–TR–127, Department of Computer Science, Rutgers University, June 1983.

[11] C. Dwork and M. Herlihy. Contention in shared memory algorithms. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 174–183, May 1993.

[12] J. Edmonds and R. Karp. Theoretical improvements in the algorithmic efficiency for network flow problems. *Journal of the ACM*, 19:248–264, 1972.

[13] C.S. Ellis and T.J. Olson. Algorithms for parallel memory allocation. *Journal of Parallel Programming*, 17:303–345, 1988.

[14] L.R. Jr. Ford and D. R. Fulkerson. *Flows in Networks.* Princeton University Press, 1962.

[15] E. Freudenthal and A. Gottlieb. Process coordination with fetch-and-increment. In *Proceedings of the 4th International Conference on Architecture Support for Programming Languages and Operating Systems*, April 1991.

[16] J.R. Goodman, M.K. Vernon, and P.J. Woest. Efficient synchronization primitives for large-scale cache-coherent multiprocessors. In *Proceedings of the 3rd ASPLOS*, pages 64–75, April 1989.

[17] A. Gottlieb, B.D. Lubachevsky, and L. Rudolph. Basic techniques for the efficient coordination of very large numbers of cooperating sequential processors. *ACM Transactions on Programming Languages and Systems*, 5:164–189, 1983.

[18] M.P. Herlihy, B. Lim, and N. Shavit. Low contention load balancing on large-scale multiprocessors. In *Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures*, July 1992.

[19] M.P. Herlihy, N. Shavit, and O. Waarts. Linearizable counting networks. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, pages 526–535, October 1991.

[20] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Am. Statist. Assoc. J.*, 58:13–30, 1963.

[21] M. Klugerman and C. G. Plaxton. Small-depth counting networks. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 417–428, May 1992.

[22] M.R. Klugerman. Lecture 17: Counting networks. In F.T. Leighton, C.E. Leiserson, and N. Kahale, editors, *Research Seminar Series 15: Advanced Parallel and VLSI Computation*, pages 153–161. MIT Press, 1991.

[23] D. E. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, Reading, MA, 1973.

[24] F. T. Leighton and C. G. Plaxton. A (fairly) simple circuit that (usually) sorts. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 264–274, October 1990.

[25] N. Nisan and D. Zuckerman. More deterministic simulation in logspace. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 235–244, May 1993.

[26] C. G. Plaxton. A hypercubic sorting network with nearly logarithmic depth. May 1992.

[27] N. Shavit and A. Zemach. Diffracting trees. In *Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures*, July 1994. To appear.

[28] H.S. Stone. Database applications of the fetch-and-add instruction. *IEEE Transactions on Computers*, C–33:604–612, 1984.

[29] A. Wigderson and D. Zuckerman. Expanders that beat the eigenvalue bound: Explicit construction and applications. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 245–251, May 1993.