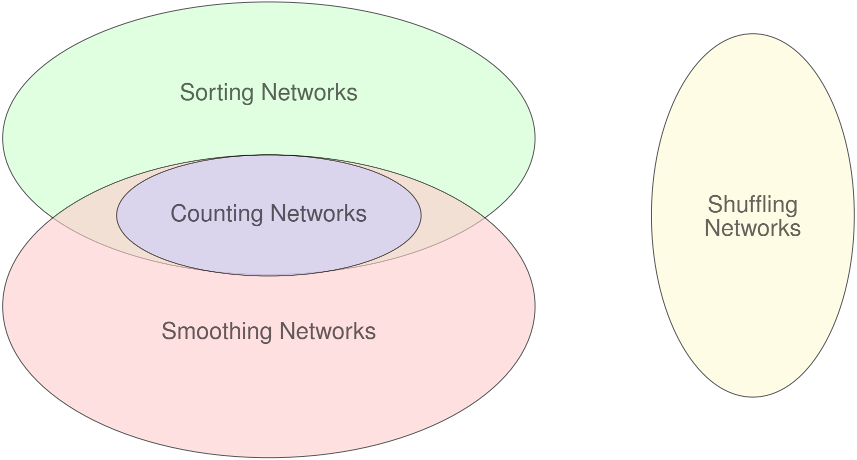
A Survey on Smoothing Networks

Thomas Sauerwald

21 July 2016

UNIVERSITY OF CAMBRIDGE

Landscape

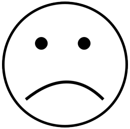

Sorting Networks
Counting Networks
Smoothing Networks
Shuffling Networks

Agenda (and disclaimer...)

Goal: Design smoothing networks for asynchronous load balancing

+ simple and elegant constructions
+ networks will be super-efficient
+ interesting mathematical theory
+ extremely sharp and tight results

- pre-specified static networks
- may not even work for any n



Outline

Sorting Network

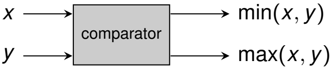
- A sorting network consists solely of wires and comparators:

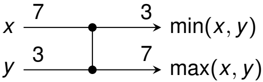
Sorting Networks

Sorting Network

- A sorting network consists solely of wires and comparators:
  - comparator is a device with, on given two inputs, x and y, returns two outputs $x' = \min(x, y)$ and $y' = \max(x, y)$

Sorting Networks

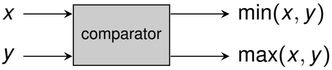


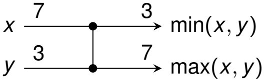


Sorting Network

- A sorting network consists solely of wires and comparators:
  - comparator is a device with, on given two inputs, x and y, returns two outputs $x' = \min(x, y)$ and $y' = \max(x, y)$

Sorting Networks

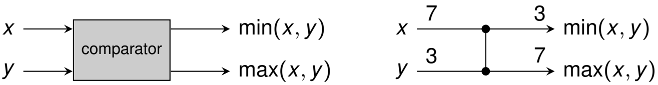




Sorting Network

- A sorting network consists solely of wires and comparators:
  - comparator is a device with, on given two inputs, x and y, returns two outputs $x' = \min(x, y)$ and $y' = \max(x, y)$
  - wires connect output of one comparator to the input of another

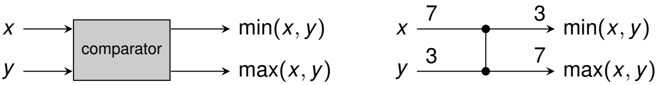
Sorting Networks



Sorting Network

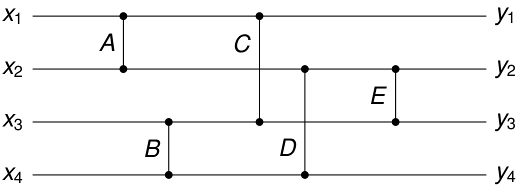
- A sorting network consists solely of wires and comparators:
  - comparator is a device with, on given two inputs, x and y, returns two outputs $x' = \min(x, y)$ and $y' = \max(x, y)$
  - wires connect output of one comparator to the input of another
  - special wires: n input wires $x_1, x_2, \ldots, x_n$ and n output wires $y_1, y_2, \ldots, y_n$

Sorting Networks

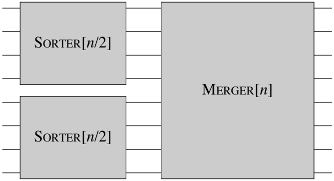


Sorting Network

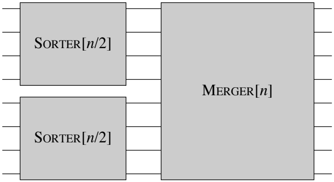
- A sorting network consists solely of wires and comparators:
  - comparator is a device with, on given two inputs, x and y, returns two outputs $x' = \min(x, y)$ and $y' = \max(x, y)$
  - wires connect output of one comparator to the input of another
  - special wires: n input wires $x_1, x_2, \ldots, x_n$ and n output wires $y_1, y_2, \ldots, y_n$

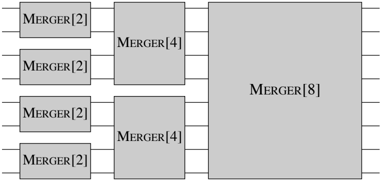


Batcher's Sorting Network

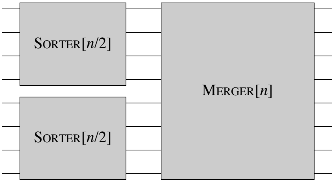


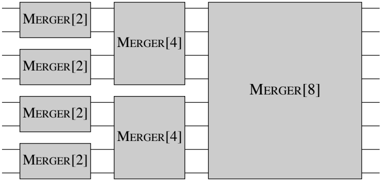
Batcher's Sorting Network

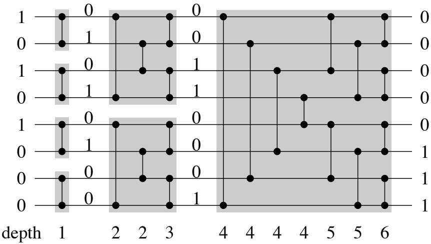




Batcher's Sorting Network





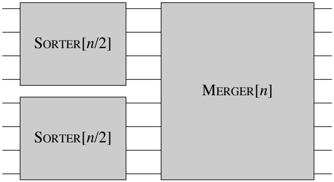


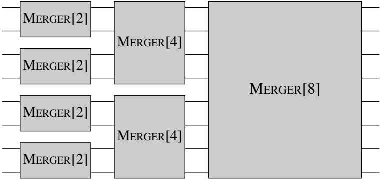
Recursion for $D(n)$:

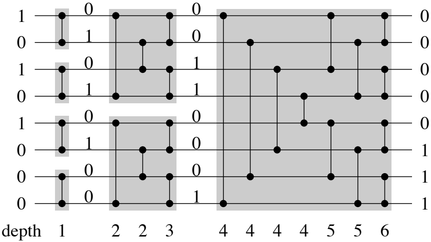
$$D(n) = \begin{cases} 0 & \text{if } n = 1, \\ D(n/2) + \log n & \text{if } n = 2^k. \end{cases}$$

Solution: $D(n) = \Theta(\log^2 n)$.

Batcher's Sorting Network







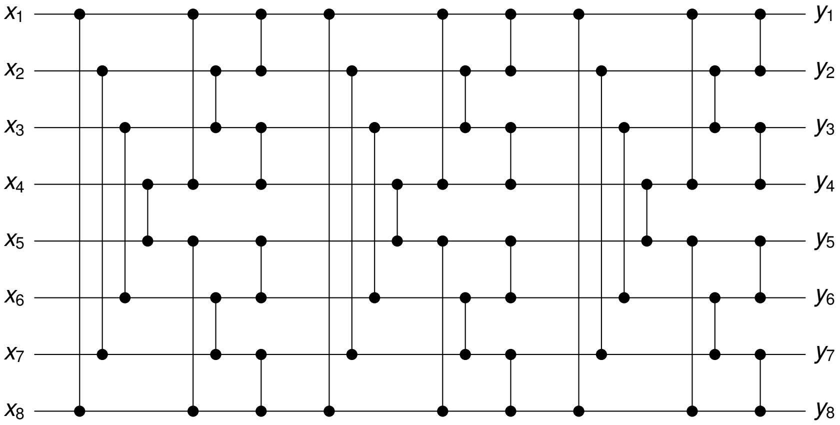
Recursion for $D(n)$:

$$D(n) = \begin{cases} 0 & \text{if } n = 1, \\ D(n/2) + \log n & \text{if } n = 2^k. \end{cases}$$

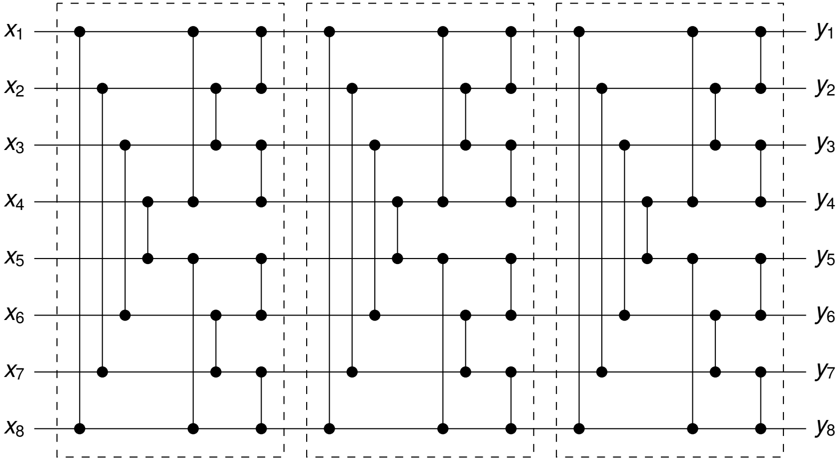
Solution: $D(n) = \Theta(\log^2 n)$.

Batcher's sorting network has depth $\Theta(\log^2 n)$.

Periodic Balanced Sorting Network [Dowd, Perl, Rudolph, Saks'89]



Periodic Balanced Sorting Network [Dowd, Perl, Rudolph, Saks'89]



Consists of $\log_2 n$ BLOCK[n] networks each of which has depth $\log_2 n$

Optimal Sorting Network

$O(\log^2 n)$ sorting networks

- $\frac{1}{2}\log^2 n$ depth: Batcher's Sorting Network
- $\log^2 n$ depth: Periodic Balanced Sorting Network

Optimal Sorting Network

$O(\log^2 n)$ sorting networks

- $\frac{1}{2}\log^2 n$ depth: Batcher's Sorting Network
- $\log^2 n$ depth: Periodic Balanced Sorting Network

Can we construct sorting networks of depth $O(\log n)$?

Optimal Sorting Network

$O(\log^2 n)$ sorting networks

- $\frac{1}{2}\log^2 n$ depth: Batcher's Sorting Network
- $\log^2 n$ depth: Periodic Balanced Sorting Network

Can we construct sorting networks of depth $O(\log n)$?

Ajtai, Komlós, Szemerédi (1983)

There exists a sorting network with depth $O(\log n)$.

Optimal Sorting Network

$O(\log^2 n)$ sorting networks

- $\frac{1}{2}\log^2 n$ depth: Batcher's Sorting Network
- $\log^2 n$ depth: Periodic Balanced Sorting Network

Can we construct sorting networks of depth $O(\log n)$?

Ajtai, Komlós, Szemerédi (1983)

There exists a sorting network with depth $O(\log n)$.

Extremely sophisticated construction that uses expander graphs and involves huges constants.

AKS network vs. Batcher's network

Donald E. Knuth (Stanford)

"Batcher's method is much better, unless n exceeds the total memory capacity of all computers on earth!"

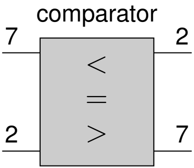
Richard J. Lipton (Georgia Tech)

*"The AKS sorting network is **galactic**: it needs that n be larger than 2^{78} or so to finally be smaller than Batcher's network for n items."*

From Sorting Networks to Counting Networks

Sorting Networks

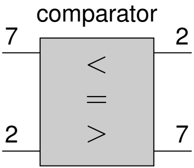
- sorts any input of size n
- special case of Comparison Networks



From Sorting Networks to Counting Networks

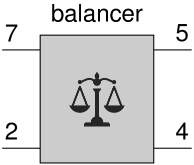
Sorting Networks

- sorts any input of size *n*
- special case of Comparison Networks



Counting Networks

- balances any stream of tokens over *n* wires
- special case of Smoothing Networks



Outline

Introduction

Sorting Networks

Counting Networks

Randomized Smoothing Networks

Stronger Notions of Smoothing Networks

Conclusion

Counting Network

Distributed Counting

Processors collectively assign successive values from a given range.

Counting Network

Distributed Counting

Processors collectively assign successive values from a given range.

Values could represent addresses in memories or destinations on an interconnection network

Counting Network

Distributed Counting

Processors collectively assign successive values from a given range.

Smoothing Networks

- like sorting networks: instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top,. . .)

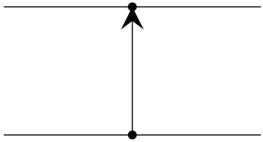
Counting Network

Distributed Counting

Processors collectively assign successive values from a given range.

Smoothing Networks

- like sorting networks: instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top,. . .)



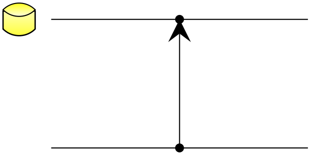
Counting Network

Distributed Counting

Processors collectively assign successive values from a given range.

Smoothing Networks

- like sorting networks: instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top,. . .)



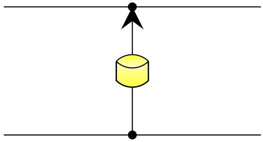
Counting Network

Distributed Counting

Processors collectively assign successive values from a given range.

Smoothing Networks

- like sorting networks: instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top,. . .)



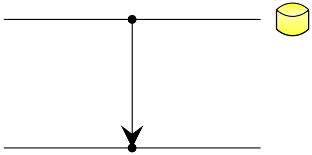
Counting Network

Distributed Counting

Processors collectively assign successive values from a given range.

Smoothing Networks

- like sorting networks: instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top,. . .)



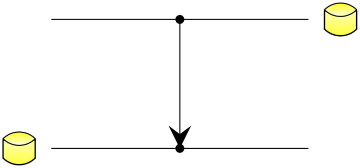
Counting Network

Distributed Counting

Processors collectively assign successive values from a given range.

Smoothing Networks

- like sorting networks: instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top,. . .)



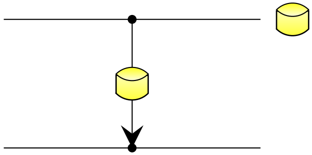
Counting Network

Distributed Counting

Processors collectively assign successive values from a given range.

Smoothing Networks

- like sorting networks: instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top,. . .)



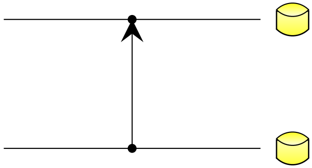
Counting Network

Distributed Counting

Processors collectively assign successive values from a given range.

Smoothing Networks

- like sorting networks: instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top,. . .)



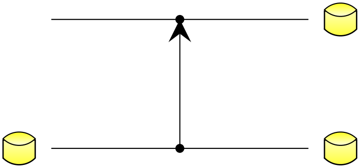
Counting Network

Distributed Counting

Processors collectively assign successive values from a given range.

Smoothing Networks

- like sorting networks: instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top,. . .)



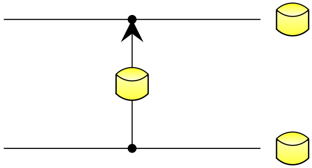
Counting Network

Distributed Counting

Processors collectively assign successive values from a given range.

Smoothing Networks

- like sorting networks: instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top,. . .)



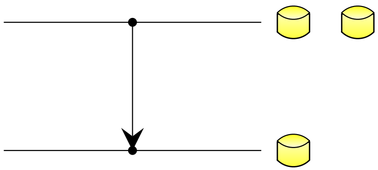
Counting Network

Distributed Counting

Processors collectively assign successive values from a given range.

Smoothing Networks

- like sorting networks: instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top,. . .)



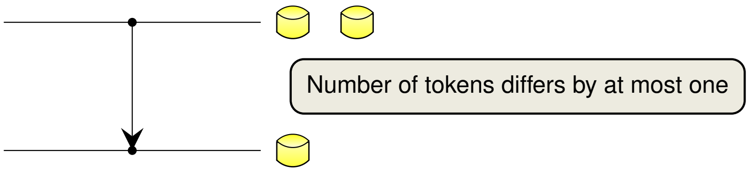
Counting Network

Distributed Counting

Processors collectively assign successive values from a given range.

Smoothing Networks

- like sorting networks: instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top,. . .)



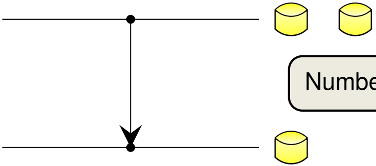
Counting Network

Distributed Counting

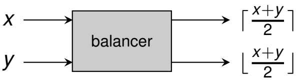
Processors collectively assign successive values from a given range.

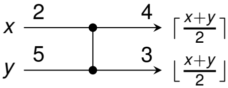
Smoothing Networks

- like sorting networks: instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top,. . .)



Number of tokens differs by at most one





Bitonic Counting Network

Counting Network (Formal Definition)

1. Let $x_1, x_2, \ldots, x_n$ be the number of tokens (ever received) on the designated input wires
2. Let $y_1, y_2, \ldots, y_n$ be the number of tokens (ever received) on the designated output wires

Bitonic Counting Network

Counting Network (Formal Definition)

1. Let $x_1, x_2, \ldots, x_n$ be the number of tokens (ever received) on the designated input wires
2. Let $y_1, y_2, \ldots, y_n$ be the number of tokens (ever received) on the designated output wires
3. In a quiescent state: $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$
4. A counting network is a smoothing network with the **step-property:**

$$0 \leq y_i - y_j \leq 1 \text{ for any } i < j.$$

Bitonic Counting Network

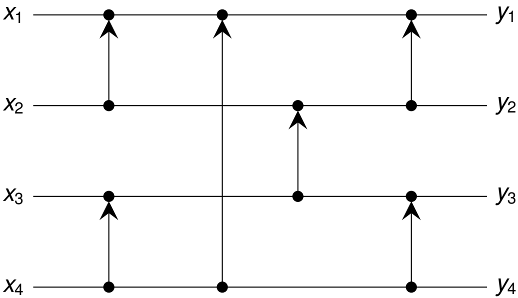
Counting Network (Formal Definition)

1. Let $x_1, x_2, \ldots, x_n$ be the number of tokens (ever received) on the designated input wires
2. Let $y_1, y_2, \ldots, y_n$ be the number of tokens (ever received) on the designated output wires
3. In a quiescent state: $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$
4. A counting network is a smoothing network with the **step-property:**

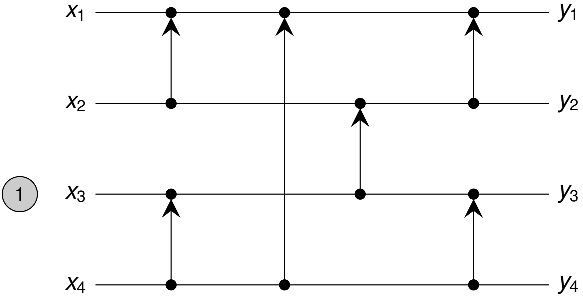
$$0 \leq y_i - y_j \leq 1 \text{ for any } i < j.$$

Bitonic Counting Network: Take Batcher's Sorting Network and replace each comparator by a balancer.

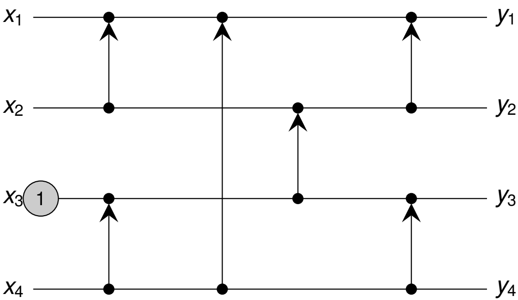
Asynchronous Execution on the Bitonic Counting Network



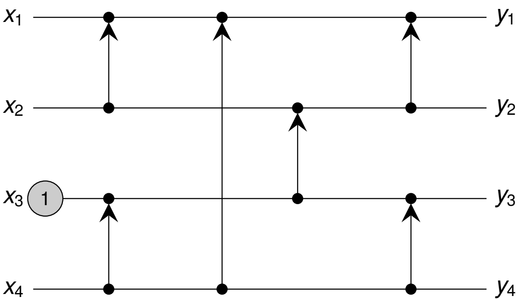
Asynchronous Execution on the Bitonic Counting Network



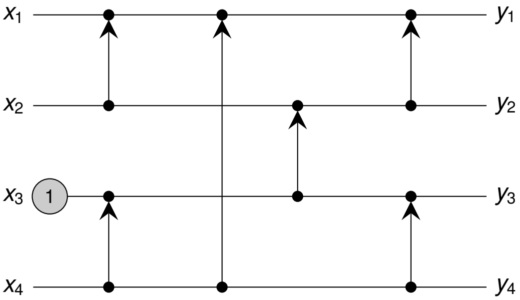
Asynchronous Execution on the Bitonic Counting Network



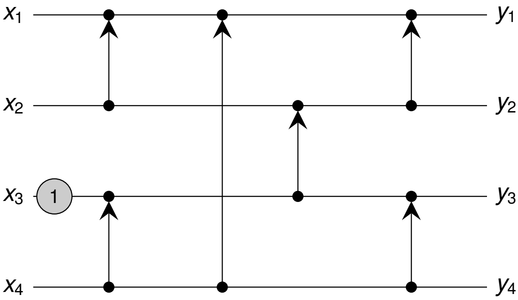
Asynchronous Execution on the Bitonic Counting Network



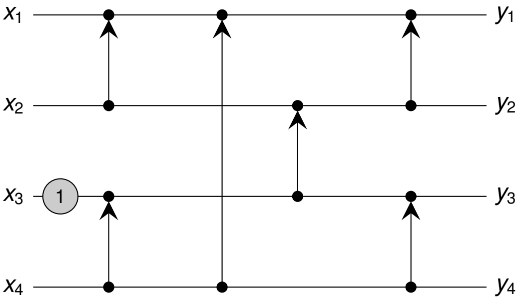
Asynchronous Execution on the Bitonic Counting Network



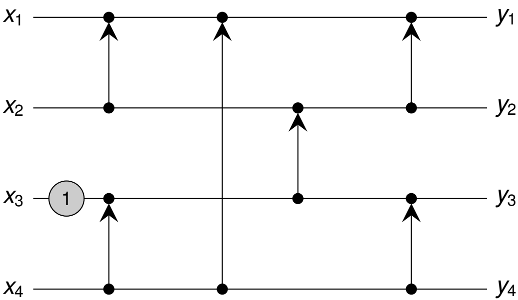
Asynchronous Execution on the Bitonic Counting Network



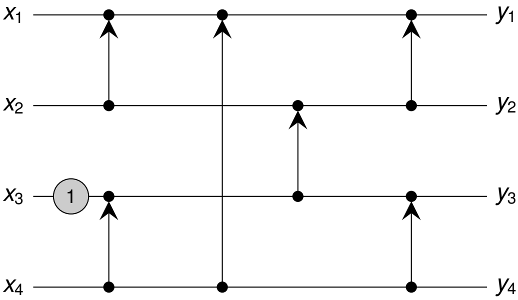
Asynchronous Execution on the Bitonic Counting Network



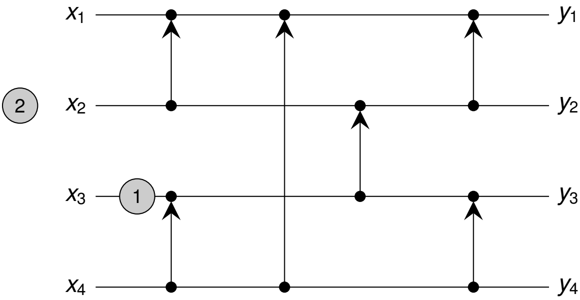
Asynchronous Execution on the Bitonic Counting Network



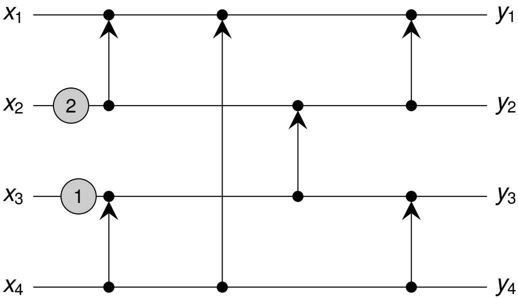
Asynchronous Execution on the Bitonic Counting Network



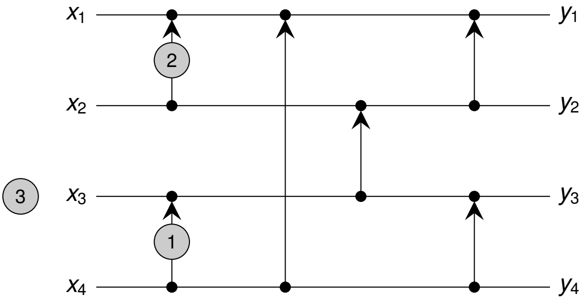
Asynchronous Execution on the Bitonic Counting Network



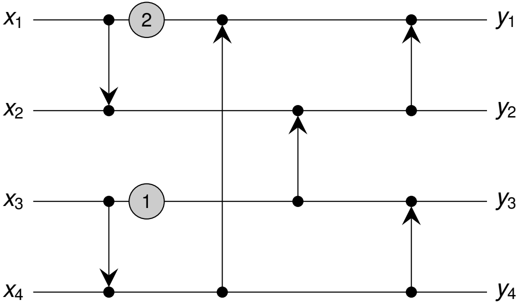
Asynchronous Execution on the Bitonic Counting Network



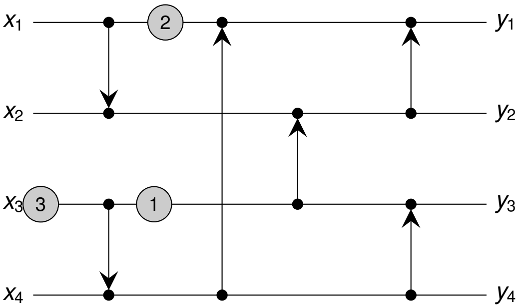
Asynchronous Execution on the Bitonic Counting Network



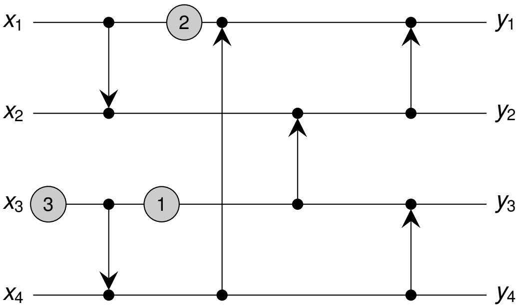
Asynchronous Execution on the Bitonic Counting Network



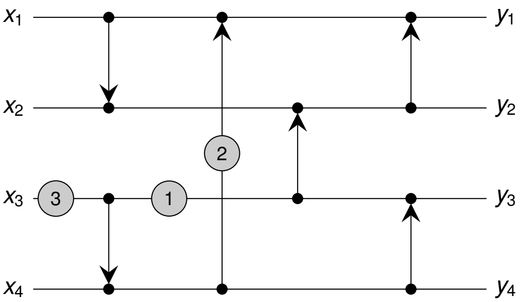
Asynchronous Execution on the Bitonic Counting Network



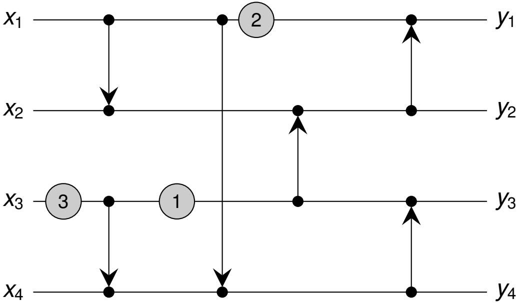
Asynchronous Execution on the Bitonic Counting Network



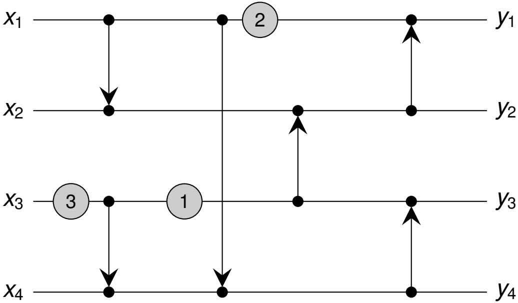
Asynchronous Execution on the Bitonic Counting Network



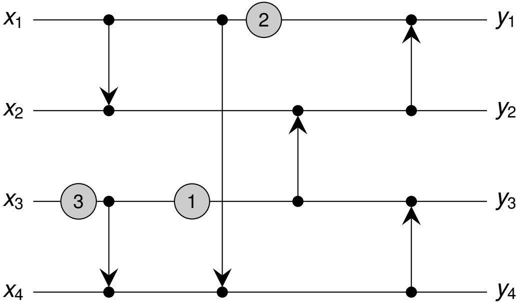
Asynchronous Execution on the Bitonic Counting Network



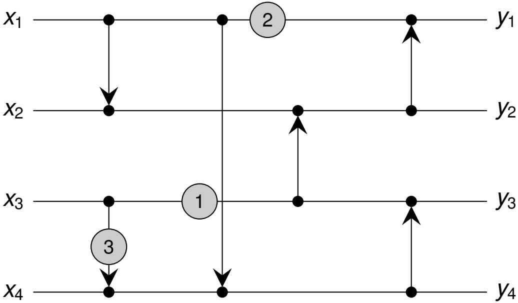
Asynchronous Execution on the Bitonic Counting Network



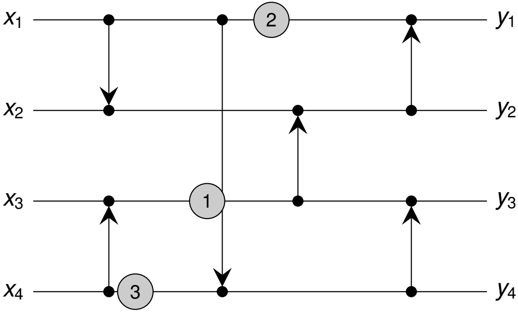
Asynchronous Execution on the Bitonic Counting Network



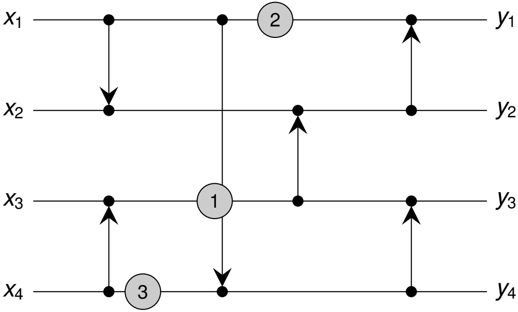
Asynchronous Execution on the Bitonic Counting Network



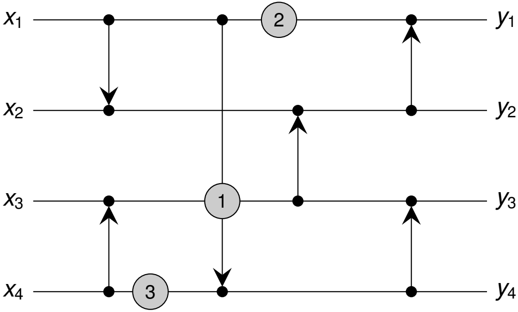
Asynchronous Execution on the Bitonic Counting Network



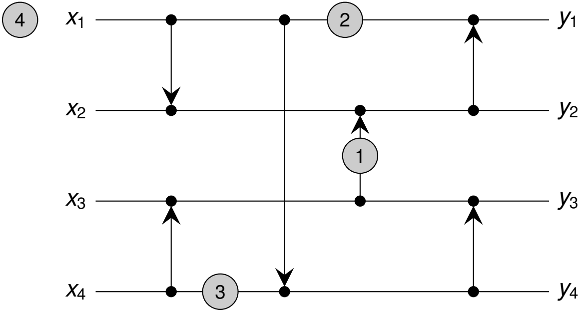
Asynchronous Execution on the Bitonic Counting Network



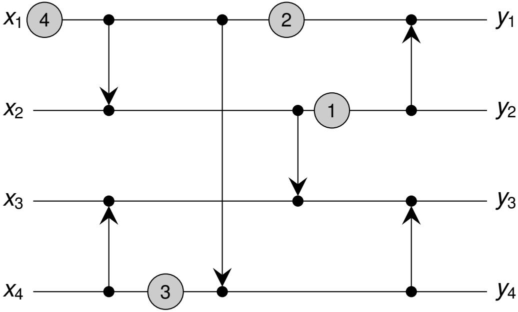
Asynchronous Execution on the Bitonic Counting Network



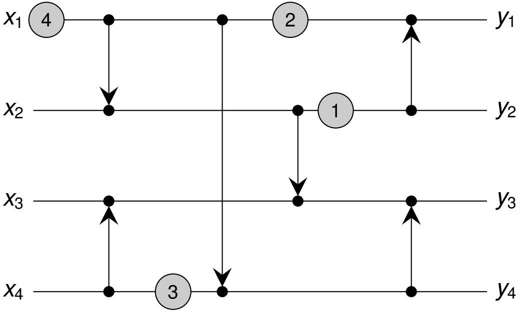
Asynchronous Execution on the Bitonic Counting Network



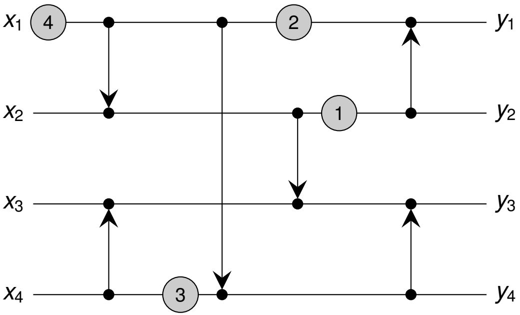
Asynchronous Execution on the Bitonic Counting Network



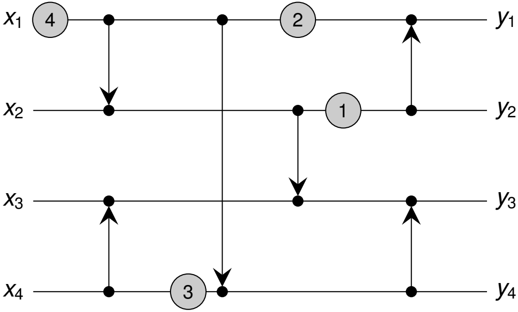
Asynchronous Execution on the Bitonic Counting Network



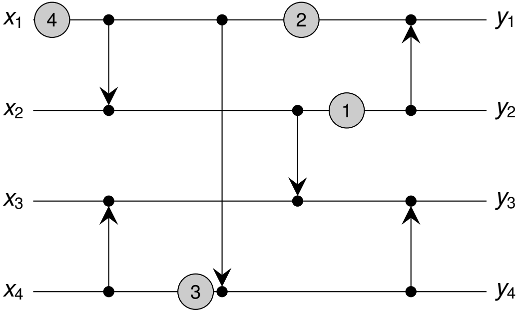
Asynchronous Execution on the Bitonic Counting Network



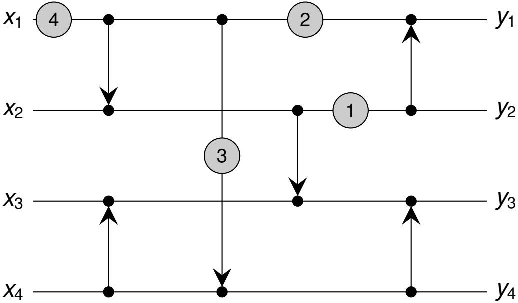
Asynchronous Execution on the Bitonic Counting Network



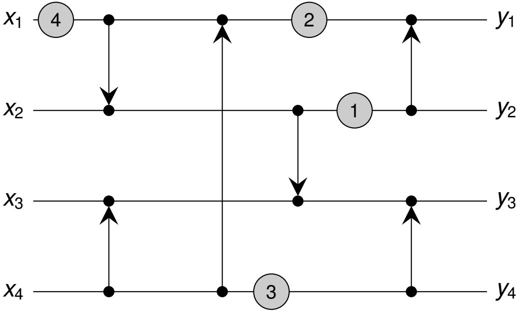
Asynchronous Execution on the Bitonic Counting Network



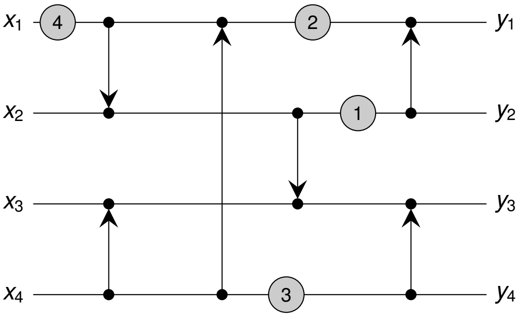
Asynchronous Execution on the Bitonic Counting Network



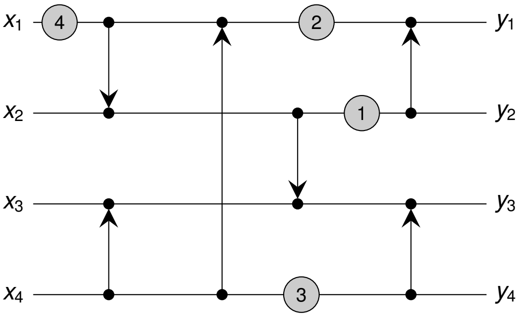
Asynchronous Execution on the Bitonic Counting Network



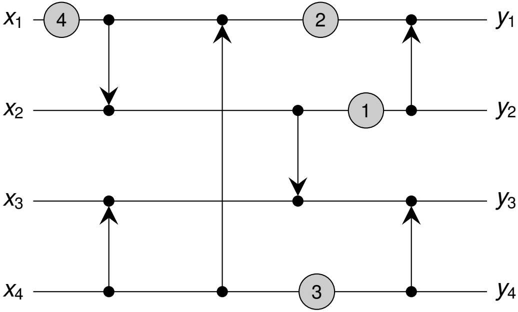
Asynchronous Execution on the Bitonic Counting Network



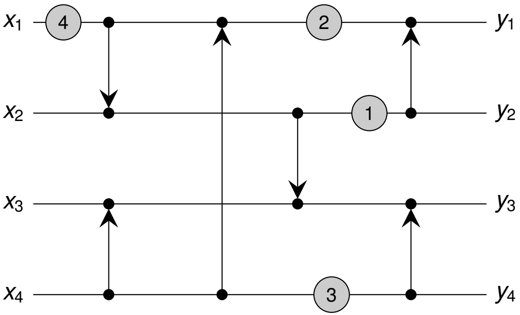
Asynchronous Execution on the Bitonic Counting Network



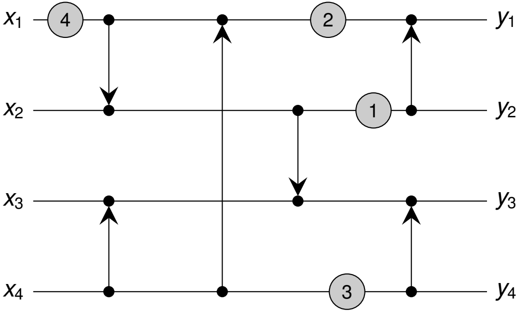
Asynchronous Execution on the Bitonic Counting Network



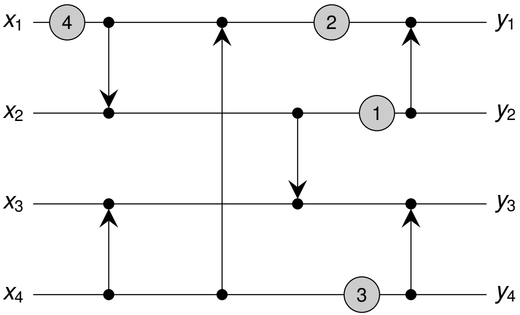
Asynchronous Execution on the Bitonic Counting Network



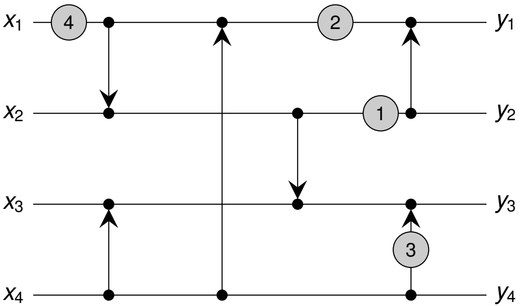
Asynchronous Execution on the Bitonic Counting Network



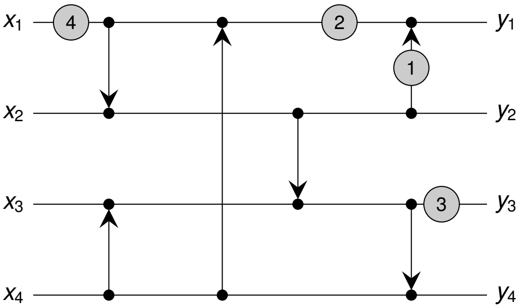
Asynchronous Execution on the Bitonic Counting Network



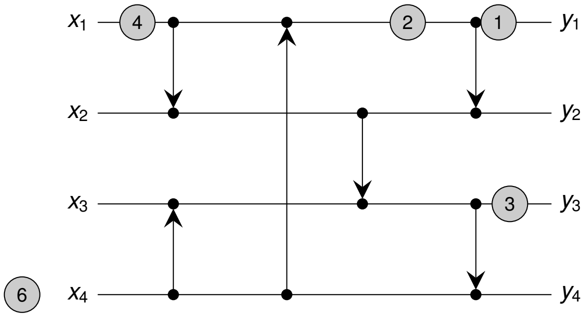
Asynchronous Execution on the Bitonic Counting Network



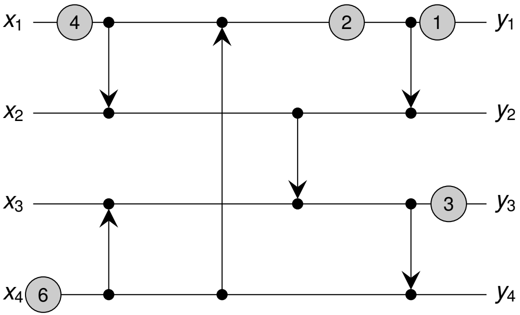
Asynchronous Execution on the Bitonic Counting Network



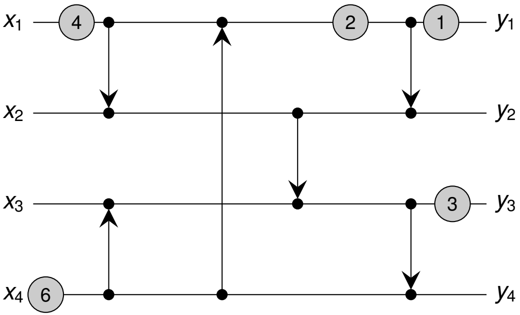
Asynchronous Execution on the Bitonic Counting Network



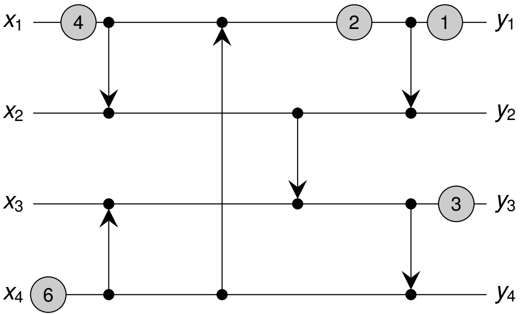
Asynchronous Execution on the Bitonic Counting Network



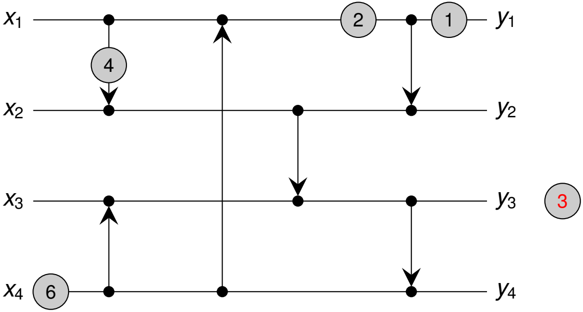
Asynchronous Execution on the Bitonic Counting Network



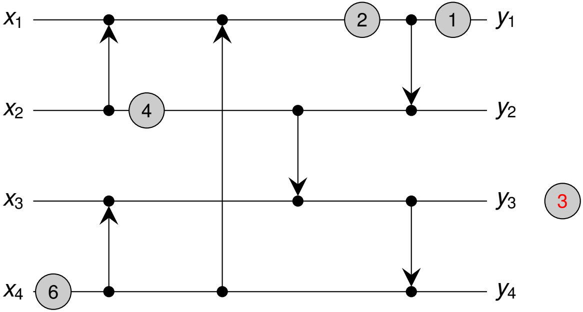
Asynchronous Execution on the Bitonic Counting Network



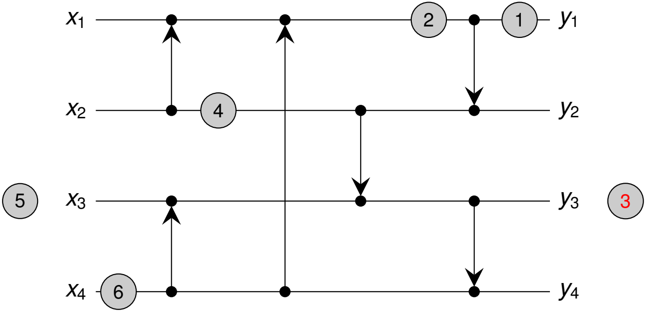
Asynchronous Execution on the Bitonic Counting Network



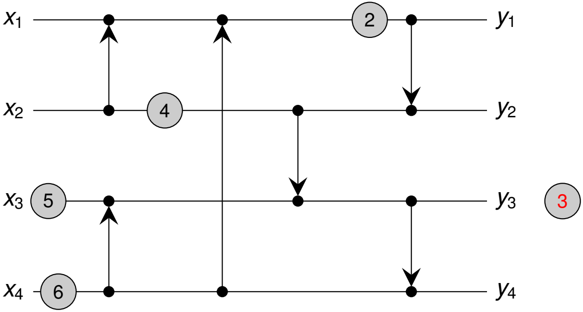
Asynchronous Execution on the Bitonic Counting Network



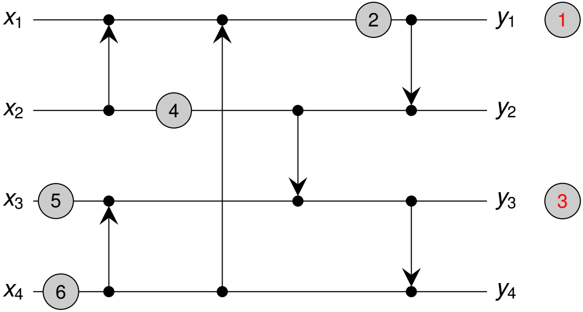
Asynchronous Execution on the Bitonic Counting Network



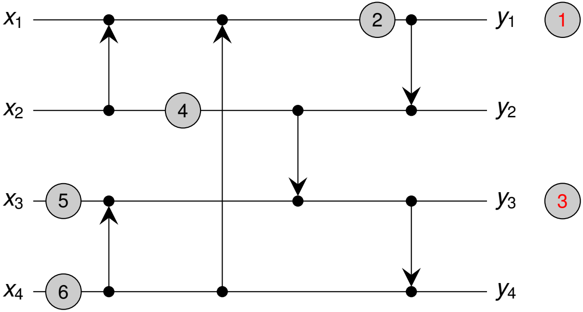
Asynchronous Execution on the Bitonic Counting Network



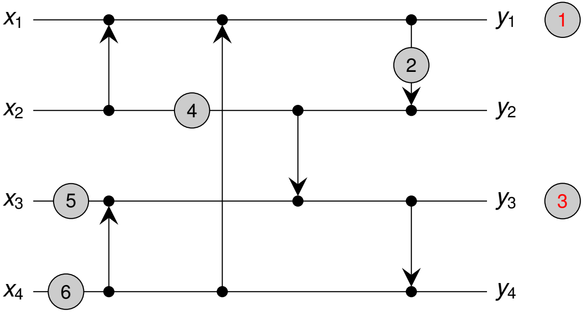
Asynchronous Execution on the Bitonic Counting Network



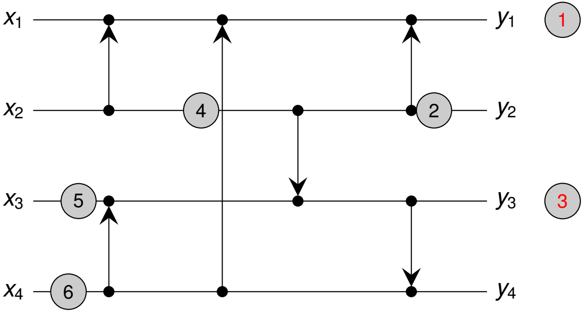
Asynchronous Execution on the Bitonic Counting Network



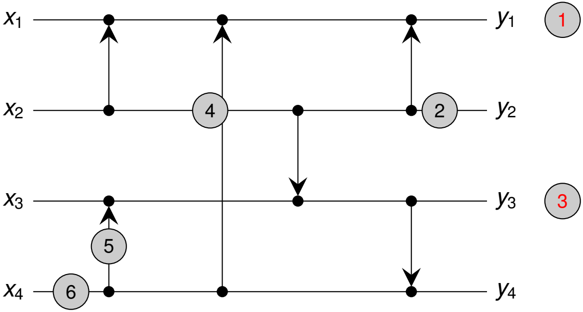
Asynchronous Execution on the Bitonic Counting Network



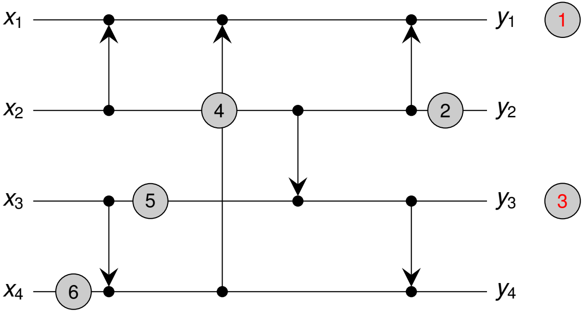
Asynchronous Execution on the Bitonic Counting Network



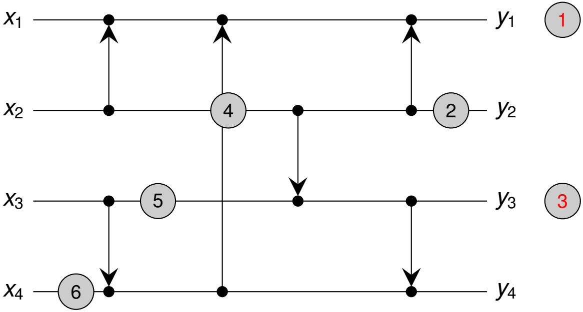
Asynchronous Execution on the Bitonic Counting Network



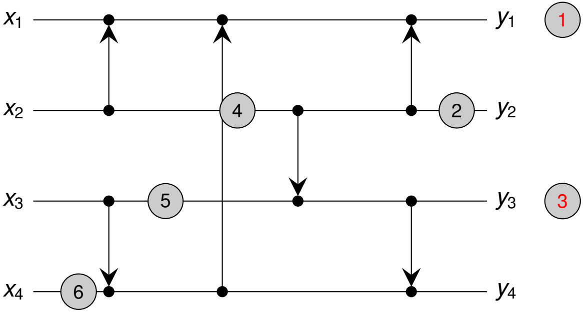
Asynchronous Execution on the Bitonic Counting Network



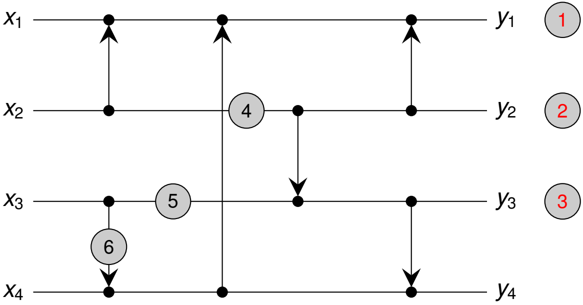
Asynchronous Execution on the Bitonic Counting Network



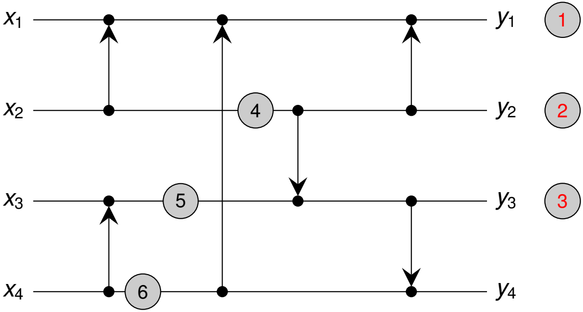
Asynchronous Execution on the Bitonic Counting Network



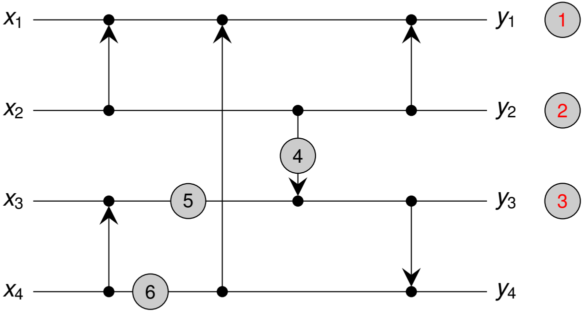
Asynchronous Execution on the Bitonic Counting Network



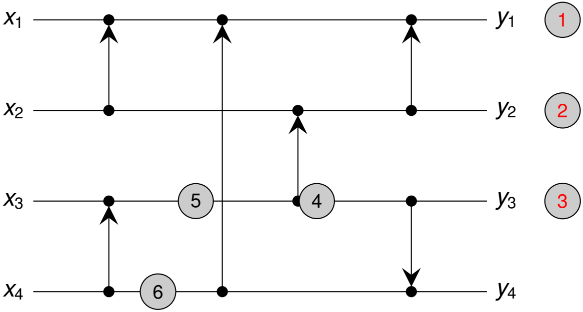
Asynchronous Execution on the Bitonic Counting Network



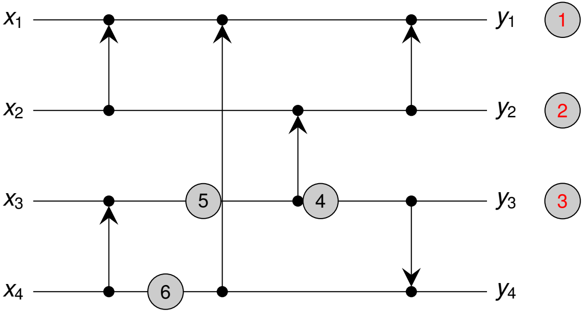
Asynchronous Execution on the Bitonic Counting Network



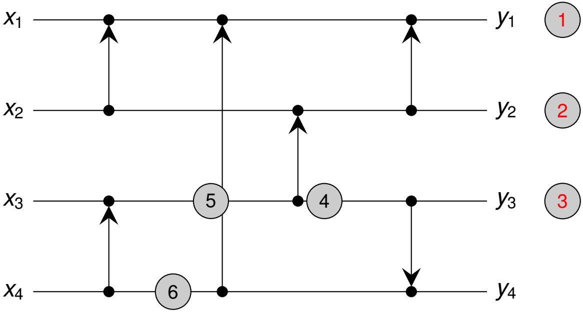
Asynchronous Execution on the Bitonic Counting Network



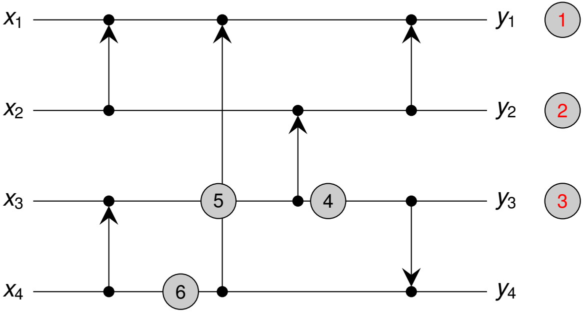
Asynchronous Execution on the Bitonic Counting Network



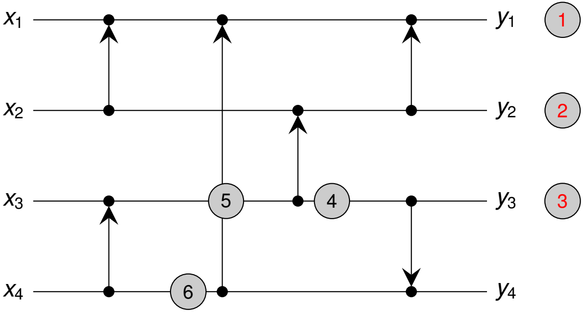
Asynchronous Execution on the Bitonic Counting Network



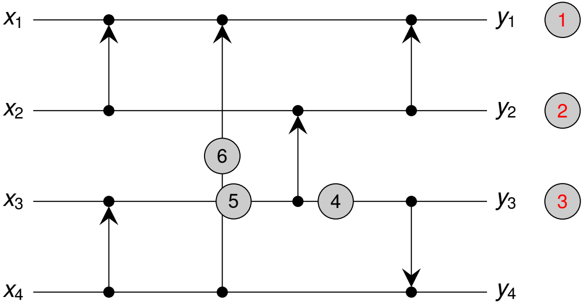
Asynchronous Execution on the Bitonic Counting Network



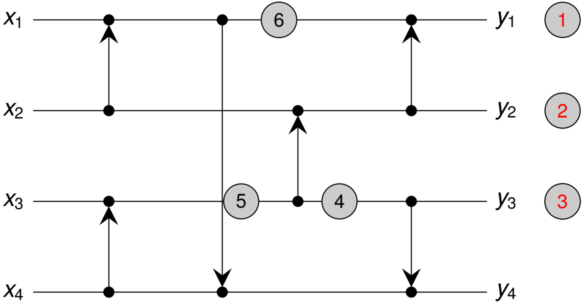
Asynchronous Execution on the Bitonic Counting Network



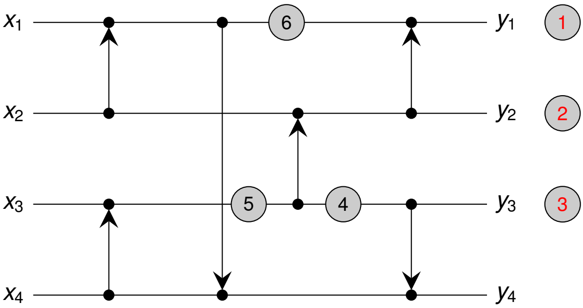
Asynchronous Execution on the Bitonic Counting Network



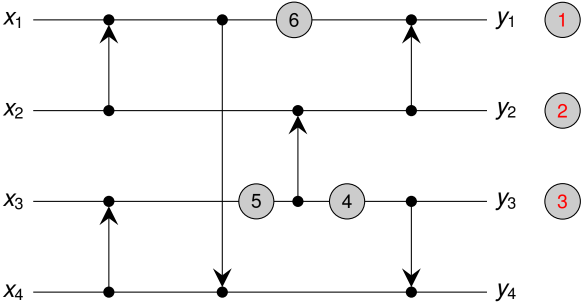
Asynchronous Execution on the Bitonic Counting Network



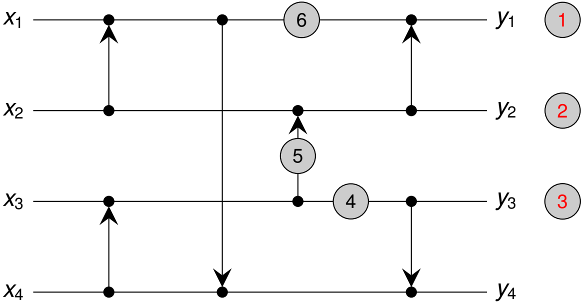
Asynchronous Execution on the Bitonic Counting Network



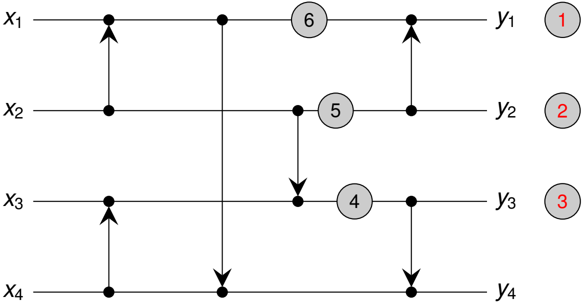
Asynchronous Execution on the Bitonic Counting Network



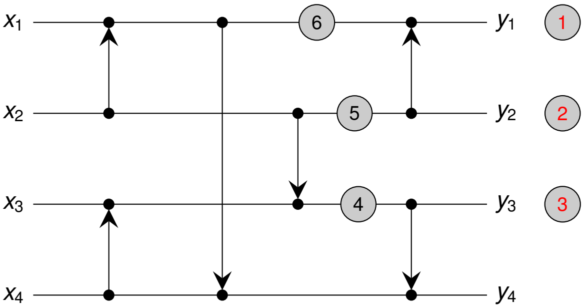
Asynchronous Execution on the Bitonic Counting Network



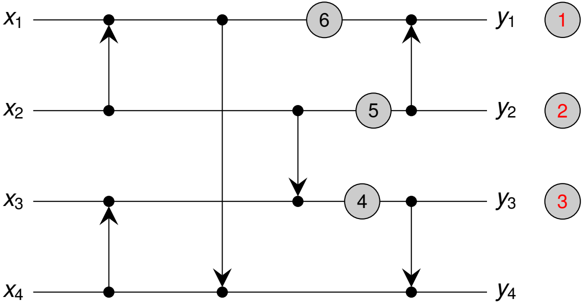
Asynchronous Execution on the Bitonic Counting Network



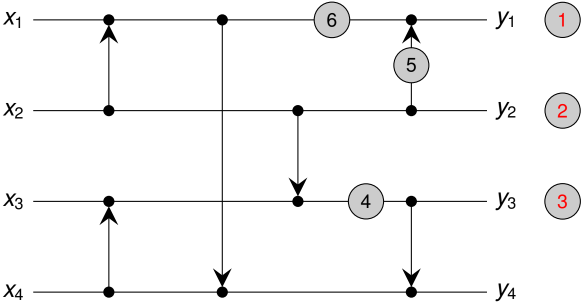
Asynchronous Execution on the Bitonic Counting Network



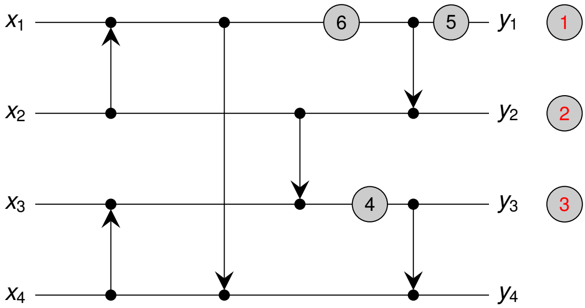
Asynchronous Execution on the Bitonic Counting Network



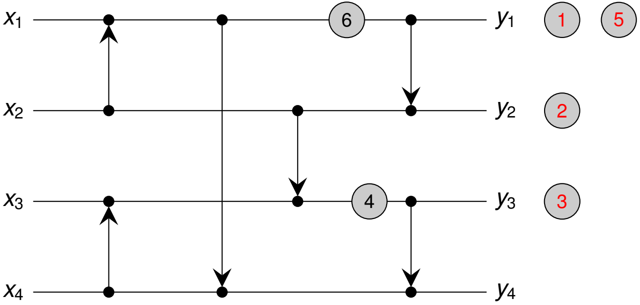
Asynchronous Execution on the Bitonic Counting Network



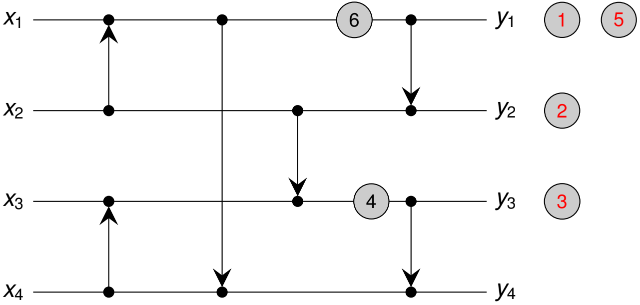
Asynchronous Execution on the Bitonic Counting Network



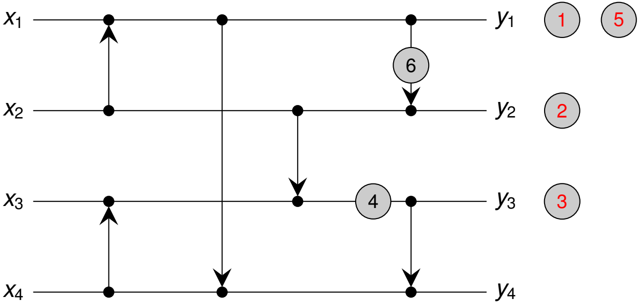
Asynchronous Execution on the Bitonic Counting Network



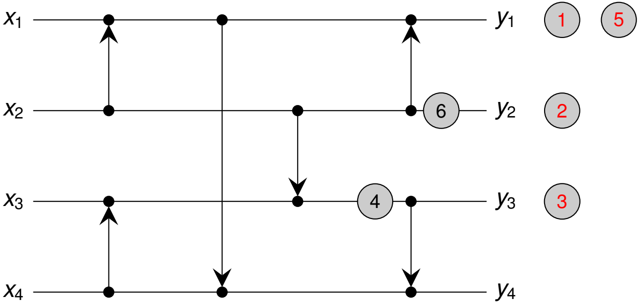
Asynchronous Execution on the Bitonic Counting Network



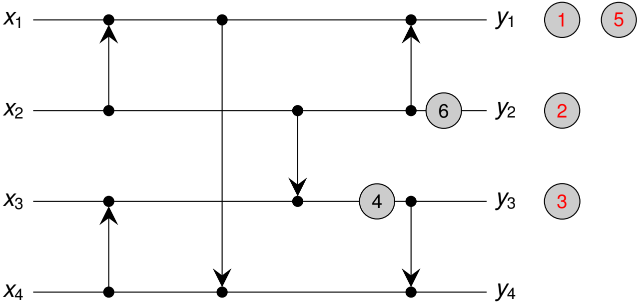
Asynchronous Execution on the Bitonic Counting Network



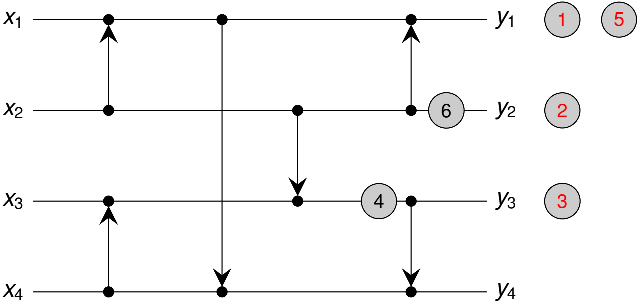
Asynchronous Execution on the Bitonic Counting Network



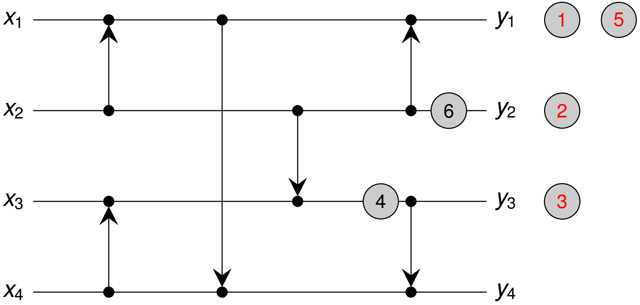
Asynchronous Execution on the Bitonic Counting Network



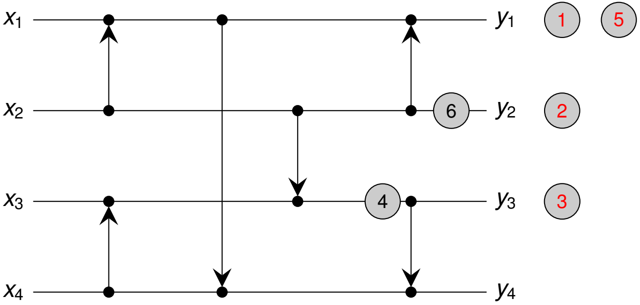
Asynchronous Execution on the Bitonic Counting Network



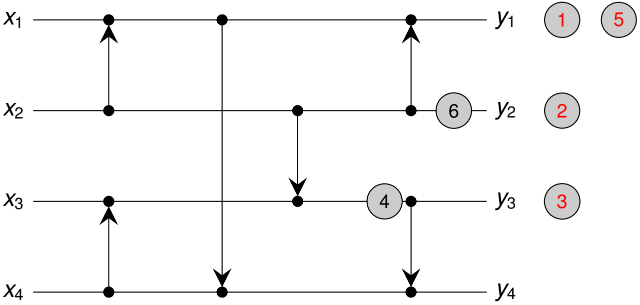
Asynchronous Execution on the Bitonic Counting Network



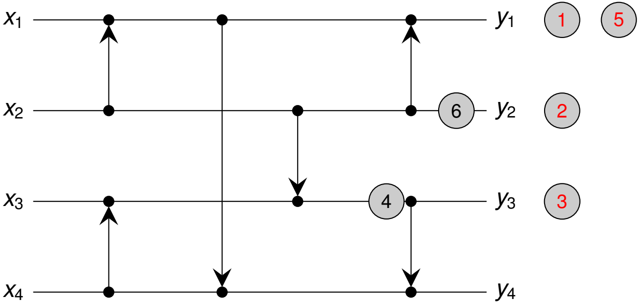
Asynchronous Execution on the Bitonic Counting Network



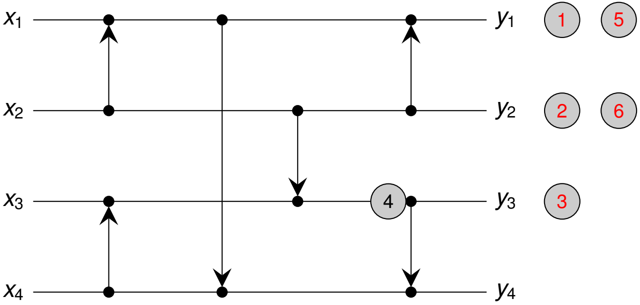
Asynchronous Execution on the Bitonic Counting Network



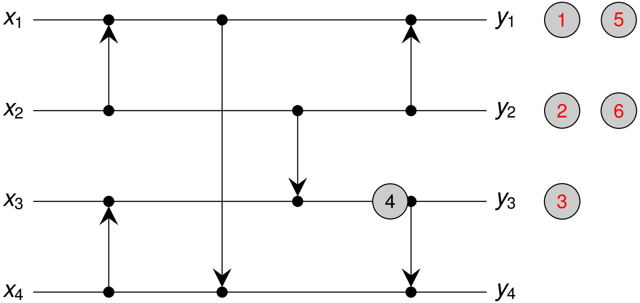
Asynchronous Execution on the Bitonic Counting Network



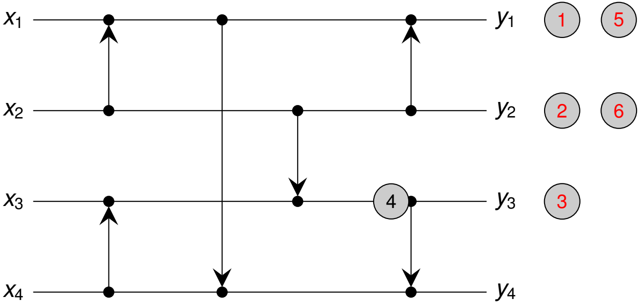
Asynchronous Execution on the Bitonic Counting Network



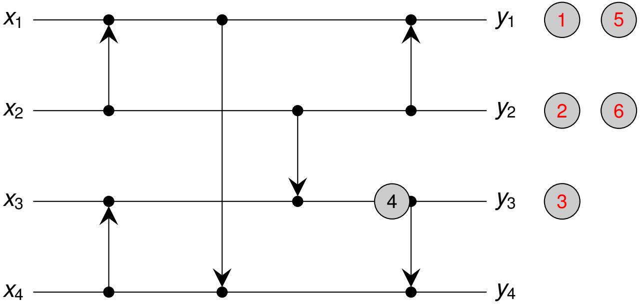
Asynchronous Execution on the Bitonic Counting Network



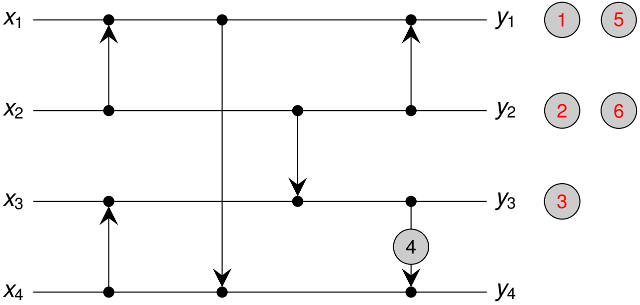
Asynchronous Execution on the Bitonic Counting Network



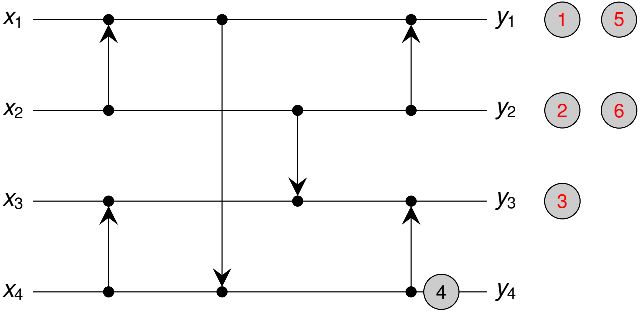
Asynchronous Execution on the Bitonic Counting Network



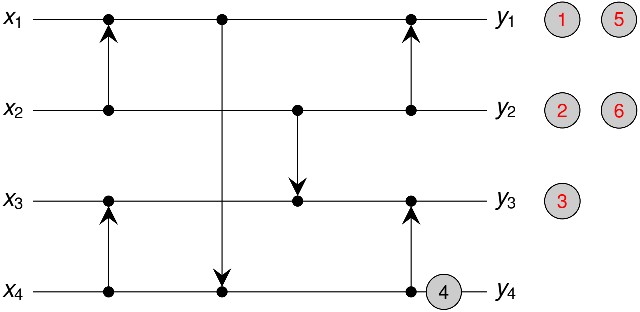
Asynchronous Execution on the Bitonic Counting Network



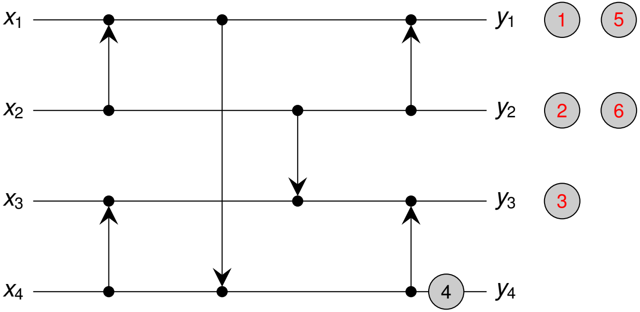
Asynchronous Execution on the Bitonic Counting Network



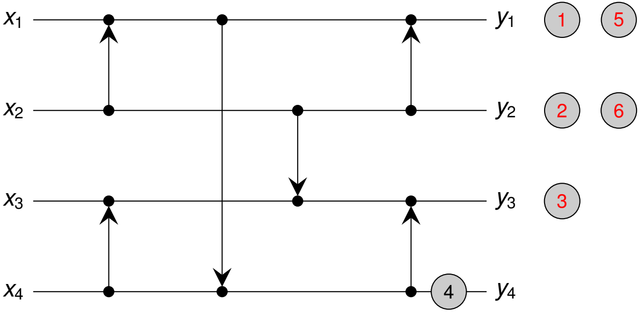
Asynchronous Execution on the Bitonic Counting Network



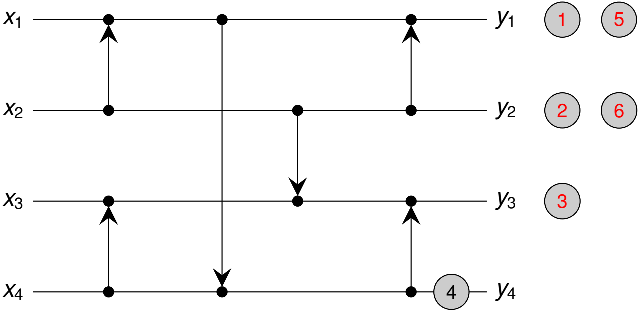
Asynchronous Execution on the Bitonic Counting Network



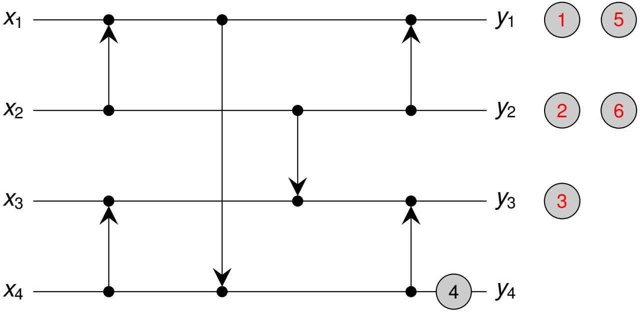
Asynchronous Execution on the Bitonic Counting Network



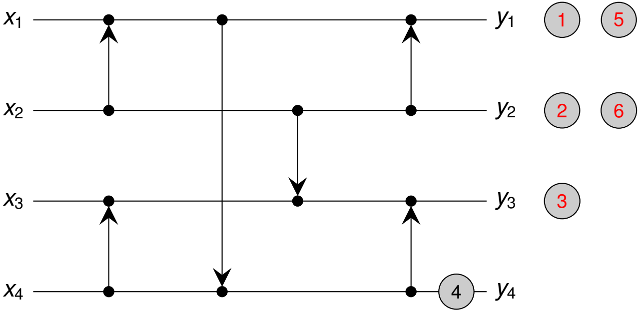
Asynchronous Execution on the Bitonic Counting Network



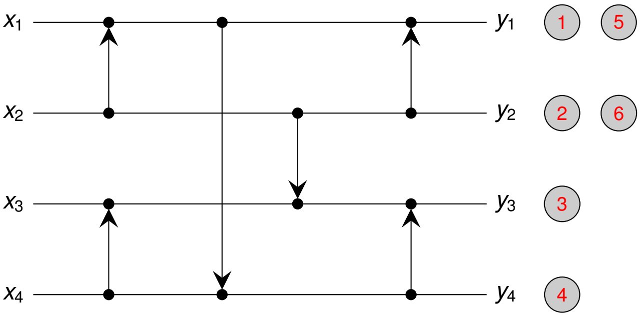
Asynchronous Execution on the Bitonic Counting Network



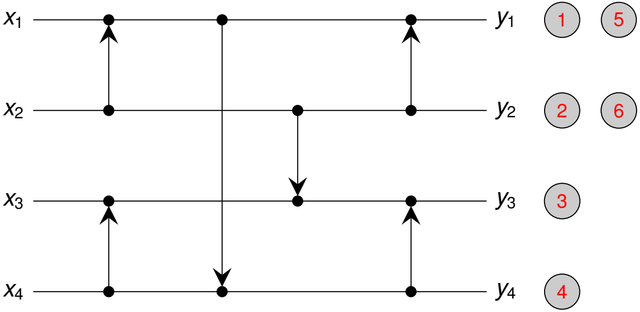
Asynchronous Execution on the Bitonic Counting Network



Asynchronous Execution on the Bitonic Counting Network

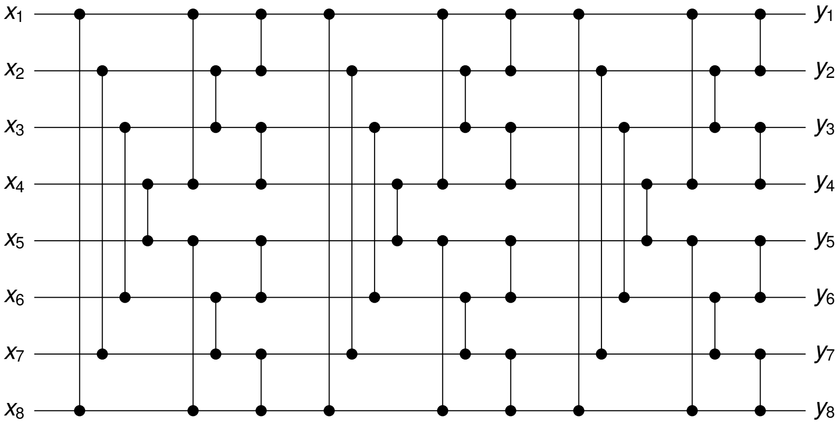


Asynchronous Execution on the Bitonic Counting Network

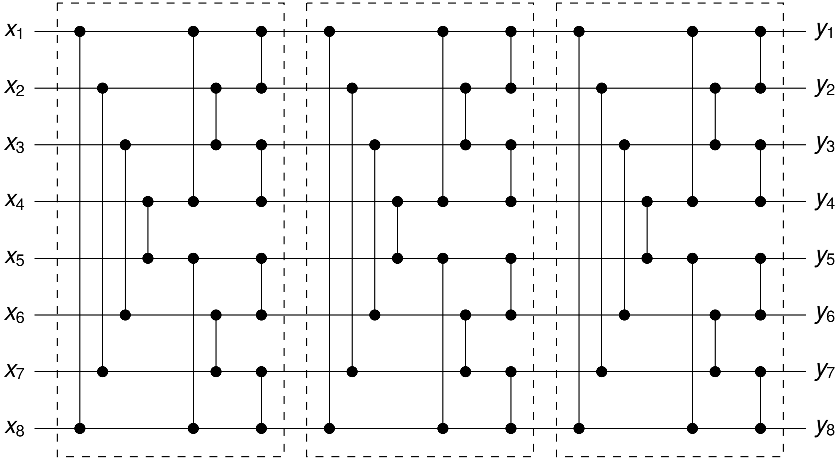


Counting can be done as follows:
Add **local counter** to each output wire i, to assign consecutive numbers $i, i + n, i + 2 \cdot n, \ldots$

A Periodic Counting Network [Aspnes, Herlihy, Shavit '94]



A Periodic Counting Network [Aspnes, Herlihy, Shavit '94]



Consists of $\log n$ BLOCK[n] networks each of which has depth $\log n$

From Counting to Sorting

Counting vs. Sorting

If a network is a counting network, then it is also a sorting network.

From Counting to Sorting

Counting vs. Sorting

If a network is a counting network, then it is also a sorting network.

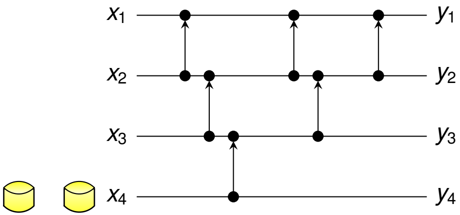
the converse is not true!

From Counting to Sorting

Counting vs. Sorting

If a network is a counting network, then it is also a sorting network.

the converse is not true!

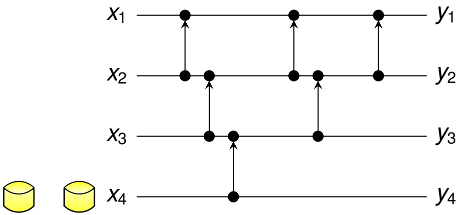


From Counting to Sorting

Counting vs. Sorting

If a network is a counting network, then it is also a sorting network.

the converse is not true!



Observation

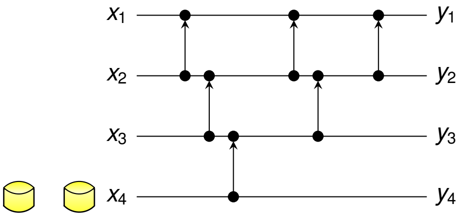
Any sorting network of depth d and n wires yields a sorting network of depth d and $n-1$ wires.

From Counting to Sorting

Counting vs. Sorting

If a network is a counting network, then it is also a sorting network.

the converse is not true!



Observation

Any sorting network of depth d and n wires yields a sorting network of depth d and $n-1$ wires.

Simply delete the bottom wire!

Necessary Conditions for Counting Networks

Observation

Any counting network must have $n = 2^k$ wires.

Necessary Conditions for Counting Networks

Observation

Any counting network must have $n = 2^k$ wires.

Proof:

Necessary Conditions for Counting Networks

Observation

Any counting network must have $n = 2^k$ wires.

Proof:

- Any output wire y_i can be expressed as:

$$y_i = \sum_{j=1}^{n} \frac{a_j}{2^\ell} x_j + \boxed{\text{“Rounding Error”}},$$

Necessary Conditions for Counting Networks

Observation

Any counting network must have $n = 2^k$ wires.

Proof:

- Any output wire y_i can be expressed as:

$$y_i = \sum_{j=1}^{n} \frac{a_j}{2^\ell} x_j + \boxed{\text{“Rounding Error”}},$$

- where:
  - $a_1, a_2, \ldots, a_n$ and ℓ are integers,
  - $\boxed{\text{“Rounding Error”}}$ is at most the depth of the network

Necessary Conditions for Counting Networks

Observation

Any counting network must have $n = 2^k$ wires.

Proof:

- Any output wire y_i can be expressed as:

$$y_i = \sum_{j=1}^{n} \frac{a_j}{2^\ell} x_j + \boxed{\text{“Rounding Error”}},$$

- where:
  - $a_1, a_2, \ldots, a_n$ and ℓ are integers,
  - $\boxed{\text{“Rounding Error”}}$ is at most the depth of the network

$\Rightarrow$ necessary condition is that $\frac{a_j}{2^j} = \frac{1}{n}$ $\square$

An Optimal Counting Network

Klugerman, Plaxton (1992)

There exists a $O(\log n)$-depth counting network.

An Optimal Counting Network

Klugerman, Plaxton (1992)

There exists a $O(\log n)$-depth counting network.

uses AKS network as a building block!

An Optimal Counting Network

Requires specific initialization of all balancers!

Klugerman, Plaxton (1992)

There exists a $O(\log n)$-depth counting network.

uses AKS network as a building block!

An Optimal Counting Network

Requires specific initialization of all balancers!

Klugerman, Plaxton (1992)

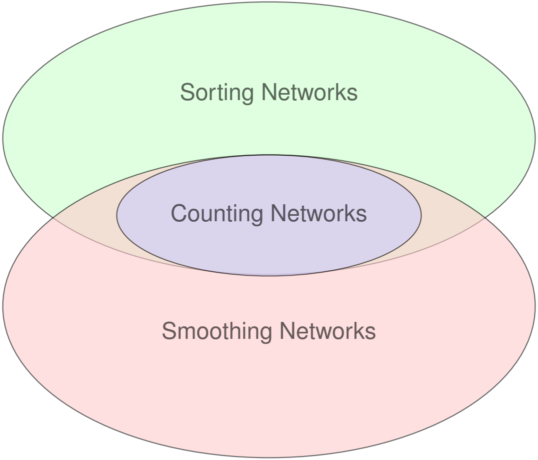
There exists a $O(\log n)$-depth counting network.

uses AKS network as a building block!

Can we trade-off simplicity of network and initialization against smoothness?

Outline

Landscape



How to Initialize The Balancers?

- Deterministic:
  Each balancer must be oriented to a certain state

How to Initialize The Balancers?

- Deterministic:
  Each balancer must be oriented to a certain state
- Randomized:
  Each balancer is oriented top or bottom uniformly at random

How to Initialize The Balancers?

- Deterministic:
  Each balancer must be oriented to a certain state
- Randomized:
  Each balancer is oriented top or bottom uniformly at random
- Arbitrary:
  Each balancer is oriented arbitrarily

How to Initialize The Balancers?

- Deterministic:
  Each balancer must be oriented to a certain state
- Randomized:
  Each balancer is oriented top or bottom uniformly at random
- Arbitrary:
  Each balancer is oriented arbitrarily

strong assumption
small discrepancy

↕

weak assumption
large discrepancy

How to Initialize The Balancers?

strong assumption
small discrepancy

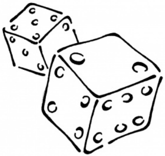
↕

weak assumption
large discrepancy

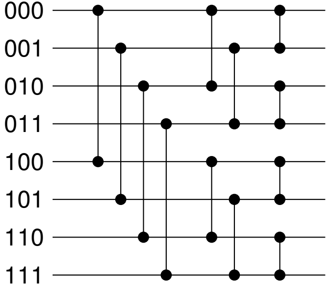
- Deterministic:
  Each balancer must be oriented to a certain state
- Randomized:
  Each balancer is oriented top or bottom uniformly at random
- Arbitrary:
  Each balancer is oriented arbitrarily

Randomized Initialization is a promising compromise:

- avoids need of global coordination
- achieves good discrepancy (=difference between maxload and minload)



The CCC (Cube-Connected-Cycles) Smoothing Network

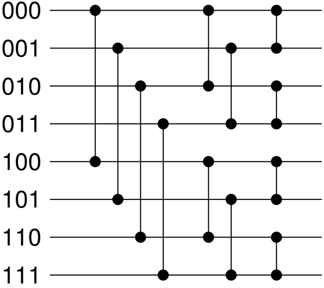


The CCC (Cube-Connected-Cycles) Smoothing Network

Motivation

Why this network?

- very simple recursive structure
- connects all inputs and outputs using minimum depth $\log_2 n$
- corresponds to dimension exchange on hypercubes

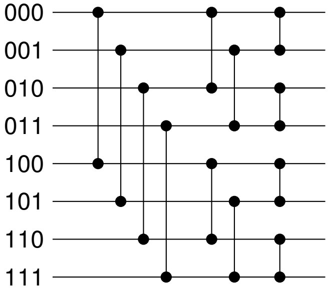


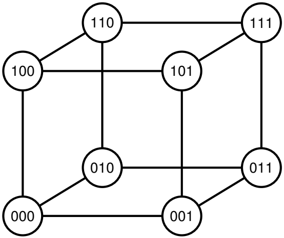
The CCC (Cube-Connected-Cycles) Smoothing Network

Motivation

Why this network?

- very simple recursive structure
- connects all inputs and outputs using minimum depth $\log_2 n$
- corresponds to dimension exchange on hypercubes



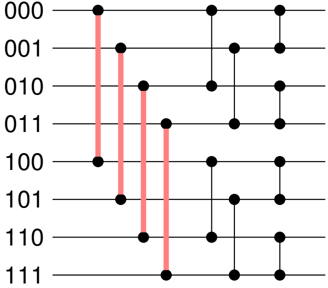


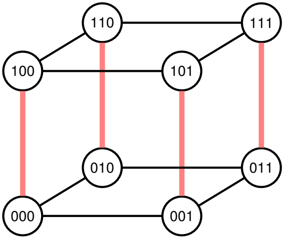
The CCC (Cube-Connected-Cycles) Smoothing Network

Motivation

Why this network?

- very simple recursive structure
- connects all inputs and outputs using minimum depth $\log_2 n$
- corresponds to dimension exchange on hypercubes



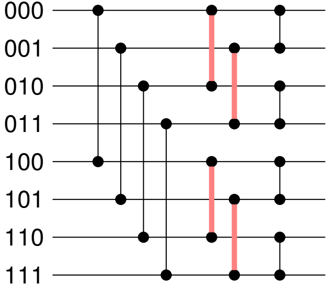


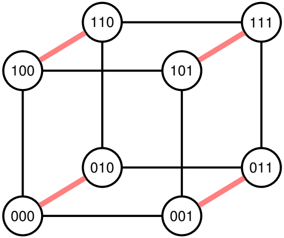
The CCC (Cube-Connected-Cycles) Smoothing Network

Motivation

Why this network?

- very simple recursive structure
- connects all inputs and outputs using minimum depth $\log_2 n$
- corresponds to dimension exchange on hypercubes



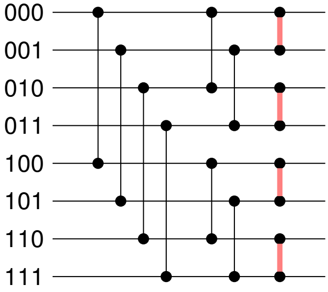


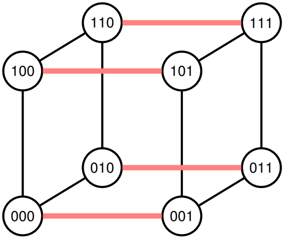
The CCC (Cube-Connected-Cycles) Smoothing Network

Motivation

Why this network?

- very simple recursive structure
- connects all inputs and outputs using minimum depth $\log_2 n$
- corresponds to dimension exchange on hypercubes



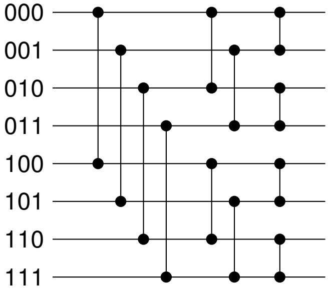


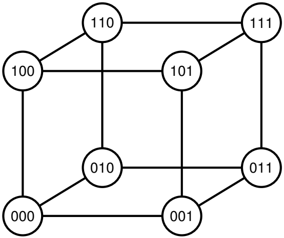
The CCC (Cube-Connected-Cycles) Smoothing Network

Motivation

Why this network?

- very simple recursive structure
- connects all inputs and outputs using minimum depth $\log_2 n$
- corresponds to dimension exchange on hypercubes

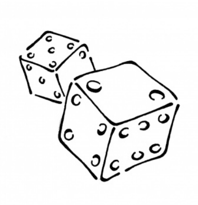




Upper Bounds on the discrepancy for a single CCC

Randomized Initialization is a promising compromise:

- avoids need of global coordination
- achieves good discrepancy (=difference between maxload and minload)



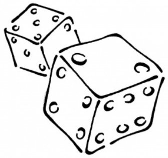
Herlihy, Tirthapura, 2006

For any input the discrepancy is at most $\mathcal{O}(\sqrt{\log n})$ w.p. $1 - n^{-1}$.

Upper Bounds on the discrepancy for a single CCC

Randomized Initialization is a promising compromise:

- avoids need of global coordination
- achieves good discrepancy (=difference between maxload and minload)



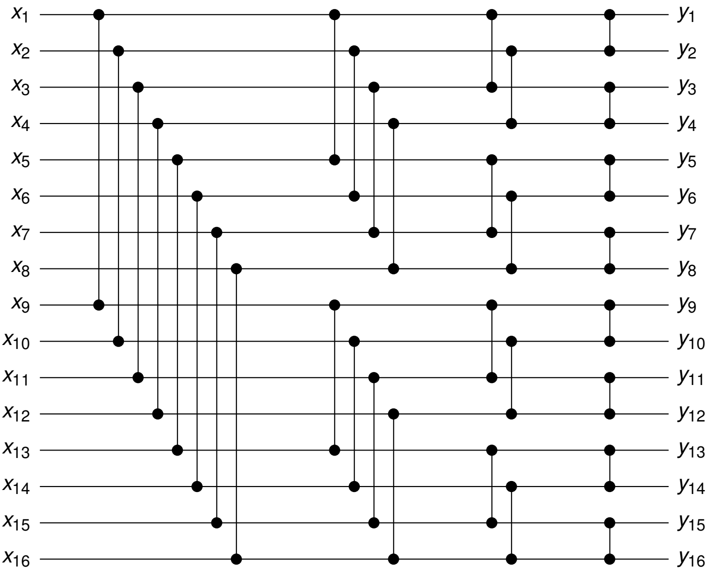
Herlihy, Tirthapura, 2006

For any input the discrepancy is at most $\mathcal{O}(\sqrt{\log n})$ w.p. $1 - n^{-1}$.

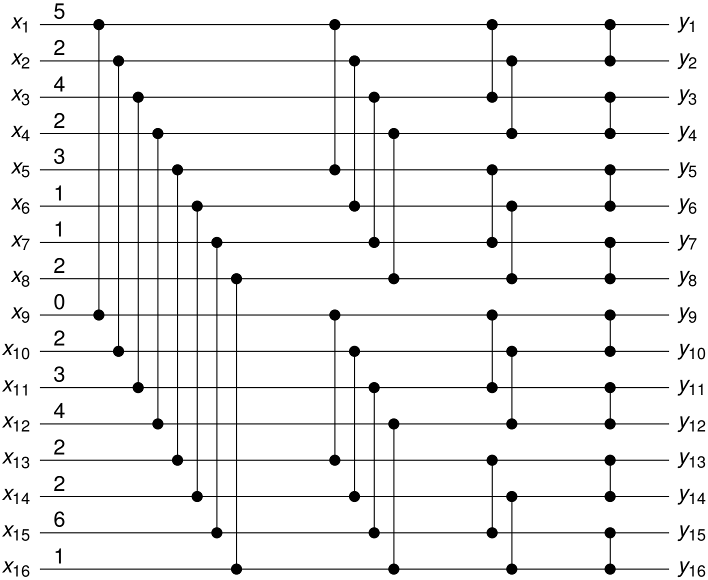
Mavronicolas, S., 2010

For any input the discrepancy is at most $\log_2 \log_2 n + 4$ w.p. $1 - n^{-1}$.

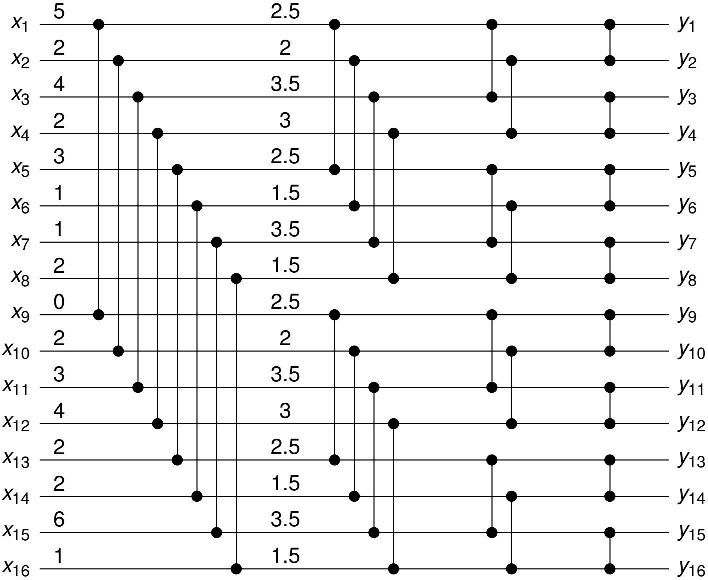
Step 1: The Continuous Case (Expected Case)



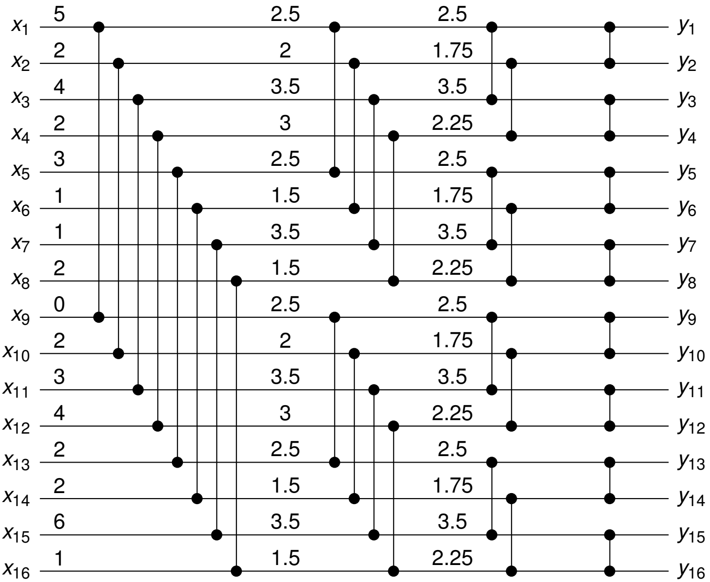
Step 1: The Continuous Case (Expected Case)



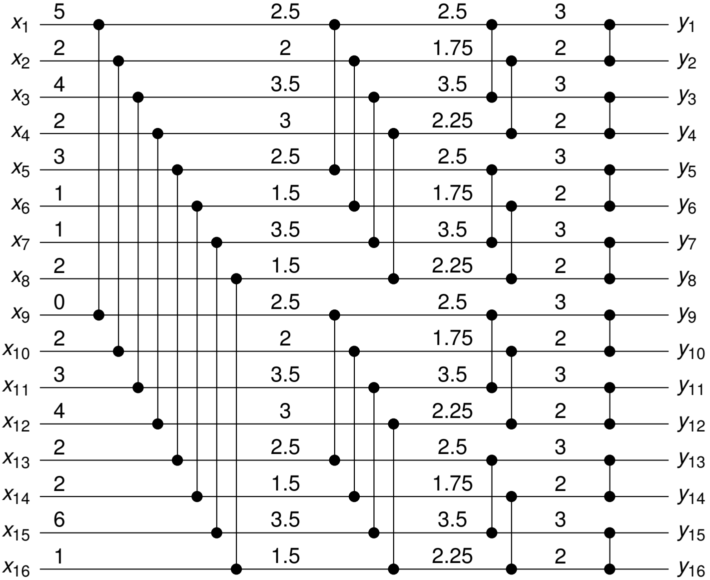
Step 1: The Continuous Case (Expected Case)



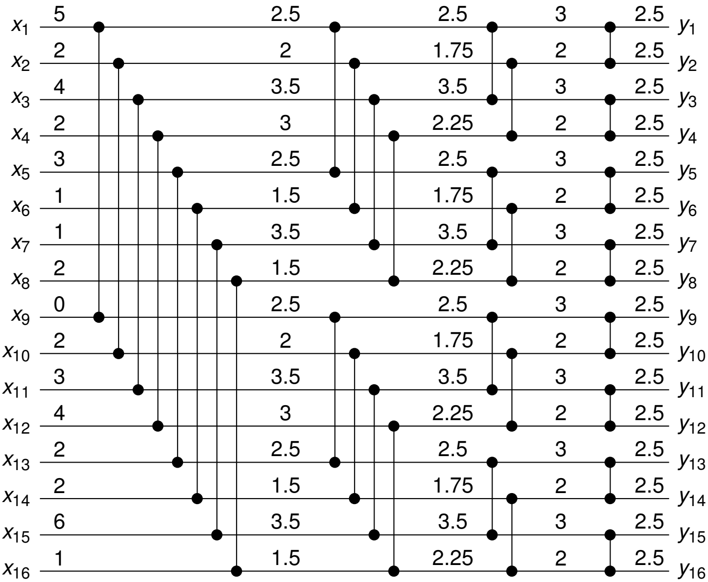
Step 1: The Continuous Case (Expected Case)



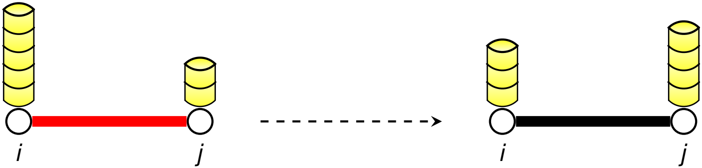
Step 1: The Continuous Case (Expected Case)



Step 1: The Continuous Case (Expected Case)



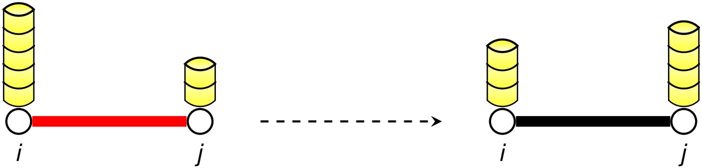
Step 2: Expressing the Rounding Error



$$x_i^t = \frac{x_i^{t-1} + x_i^{t-1}}{2}$$

$$x_j^t = \frac{x_i^{t-1} + x_j^{t-1}}{2}$$

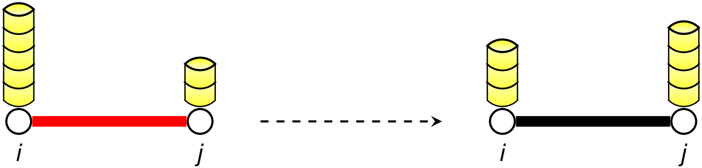
Step 2: Expressing the Rounding Error



$$x_i^t = \frac{x_i^{t-1} + x_i^{t-1}}{2} + e_i^t$$
$$x_j^t = \frac{x_i^{t-1} + x_j^{t-1}}{2} - e_i^t,$$

with e_i^t being the rounding error,

Step 2: Expressing the Rounding Error



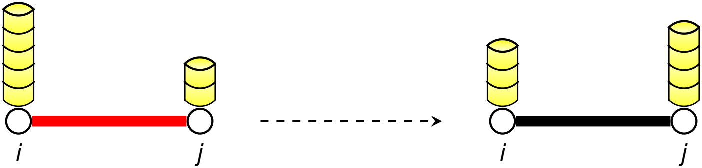
$$x_i^t = \frac{x_i^{t-1} + x_i^{t-1}}{2} + e_i^t$$
$$x_j^t = \frac{x_i^{t-1} + x_j^{t-1}}{2} - e_i^t,$$

with e_i^t being the rounding error,

$$e_i^t = \text{Odd}(x_i^{t-1} + x_j^{t-1}) \cdot \Phi_i^t,$$

where the $\Phi_i^t \in \{-1/2, +1/2\}$ is the (random) orientation.

Step 2: Expressing the Rounding Error



$$x_i^t = \frac{x_i^{t-1} + x_i^{t-1}}{2} + e_i^t$$

$$x_j^t = \frac{x_i^{t-1} + x_j^{t-1}}{2} - e_i^t,$$

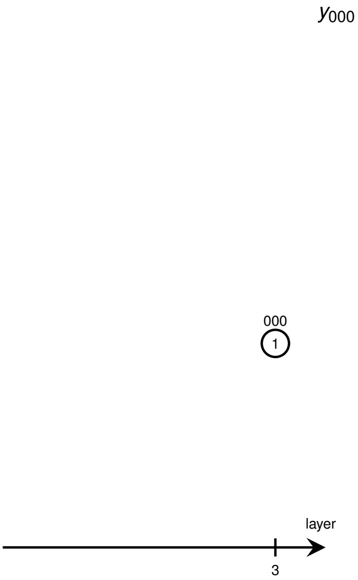
with e_i^t being the rounding error,

$$e_i^t = \text{Odd}(x_i^{t-1} + x_j^{t-1}) \cdot \Phi_i^t,$$

where the $\Phi_i^t \in \{-1/2, +1/2\}$ is the (random) orientation.

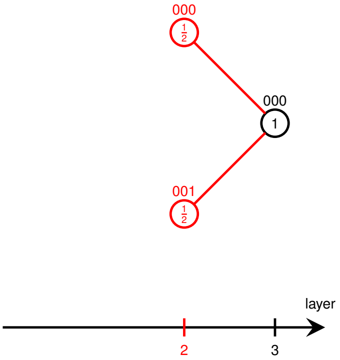
$e_i^t \in \{-1/2, 0, 1/2\}$ and $\mathbf{E}\left[e_i^t \right] = 0.$

Step 3: Solving the Recursion



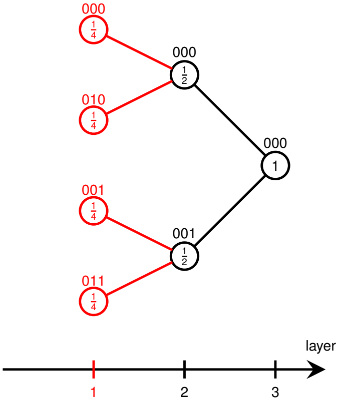
Step 3: Solving the Recursion

$$y_{000} = \frac{1}{2} y^2_{000} + \frac{1}{2} y^2_{001} + e^3_{000}$$



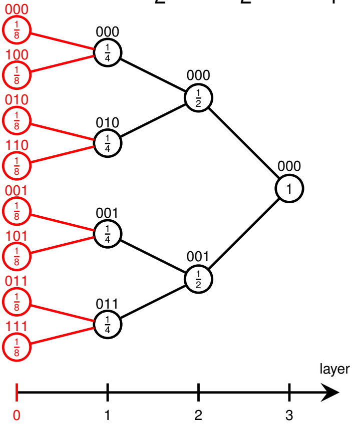
Step 3: Solving the Recursion

$$
\begin{aligned}
y_{000} &= \frac{1}{2}x^2_{000} + \frac{1}{2}x^2_{001} + e^3_{000} \\
&= \frac{1}{4}y^1_{000} + \frac{1}{4}y^1_{001} + \frac{1}{4}y^1_{001} + \frac{1}{4}y^1_{011} + e^3_{000} + \frac{1}{2}e^2_{000} + \frac{1}{2}e^2_{001}
\end{aligned}
$$



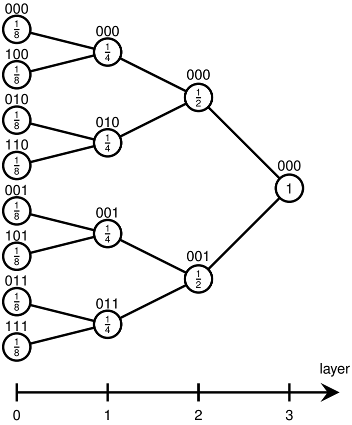
Step 3: Solving the Recursion

$$
\begin{aligned}
y_{000} &= \frac{1}{4} y^1_{000} + \frac{1}{4} y^1_{001} + \frac{1}{4} y^1_{001} + \frac{1}{4} y^1_{011} + e^3_{000} + \frac{1}{2} e^2_{000} + \frac{1}{2} e^2_{001} \\
&= \frac{1}{8} y^0_{000} + \frac{1}{8} y^0_{100} + \frac{1}{8} y^0_{010} + \frac{1}{8} y^0_{110} + \frac{1}{8} y^0_{001} + \frac{1}{8} y^0_{101} + \frac{1}{8} y^0_{011} + \frac{1}{8} y^0_{111} \\
&\quad + e^3_{000} + \frac{1}{2} e^2_{000} + \frac{1}{2} e^2_{001} + \frac{1}{4} e^1_{000} + \frac{1}{4} e^1_{010} + \frac{1}{4} e^1_{001} + \frac{1}{4} e^1_{011}
\end{aligned}
$$



Step 3: Solving the Recursion

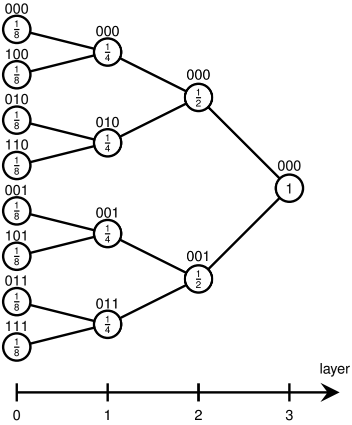
$$y_{000} = \frac{1}{8}x_{000} + \frac{1}{8}x_{100} + \frac{1}{8}x_{010} + \frac{1}{8}x_{110} + \frac{1}{8}x_{001} + \frac{1}{8}x_{101} + \frac{1}{8}x_{011} + \frac{1}{8}x_{111}$$
$$+ e^3_{000} + \frac{1}{2}e^2_{000} + \frac{1}{2}e^2_{001} + \frac{1}{4}e^1_{000} + \frac{1}{4}e^1_{010} + \frac{1}{4}e^1_{001} + \frac{1}{4}e^1_{011}$$



- continuous part and discrete part

Step 3: Solving the Recursion

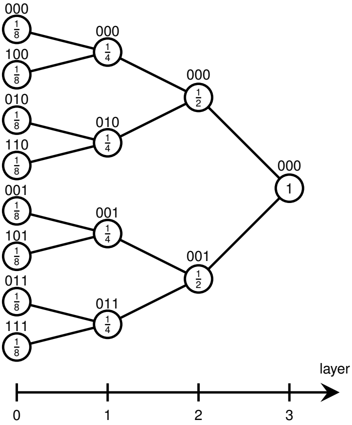
$$y_{000} = \frac{1}{8}x_{000} + \frac{1}{8}x_{100} + \frac{1}{8}x_{010} + \frac{1}{8}x_{110} + \frac{1}{8}x_{001} + \frac{1}{8}x_{101} + \frac{1}{8}x_{011} + \frac{1}{8}x_{111}$$
$$+ e^3_{000} + \frac{1}{2}e^2_{000} + \frac{1}{2}e^2_{001} + \frac{1}{4}e^1_{000} + \frac{1}{4}e^1_{010} + \frac{1}{4}e^1_{001} + \frac{1}{4}e^1_{011}$$



- continuous part and discrete part
- continuous part equals the average load

Step 3: Solving the Recursion

$$y_{000} = \frac{1}{8}x_{000} + \frac{1}{8}x_{100} + \frac{1}{8}x_{010} + \frac{1}{8}x_{110} + \frac{1}{8}x_{001} + \frac{1}{8}x_{101} + \frac{1}{8}x_{011} + \frac{1}{8}x_{111}$$
$$+ e^3_{000} + \frac{1}{2}e^2_{000} + \frac{1}{2}e^2_{001} + \frac{1}{4}e^1_{000} + \frac{1}{4}e^1_{010} + \frac{1}{4}e^1_{001} + \frac{1}{4}e^1_{011}$$



- continuous part and discrete part
- continuous part equals the average load

⇒ loads are divisible, then perfectly balanced

Step 4: Handling the Deviation

$$y_{000} = \frac{1}{8}x_{000} + \frac{1}{8}x_{100} + \frac{1}{8}x_{010} + \frac{1}{8}x_{110} + \frac{1}{8}x_{001} + \frac{1}{8}x_{101} + \frac{1}{8}x_{011} + \frac{1}{8}x_{111}$$
$$+ e^{3}_{000} + \frac{1}{2}e^{2}_{000} + \frac{1}{2}e^{2}_{001} + \frac{1}{4}e^{1}_{000} + \frac{1}{4}e^{1}_{010} + \frac{1}{4}e^{1}_{001} + \frac{1}{4}e^{1}_{011}$$

Step 4: Handling the Deviation

$$y_{000} = \frac{1}{8}x_{000} + \frac{1}{8}x_{100} + \frac{1}{8}x_{010} + \frac{1}{8}x_{110} + \frac{1}{8}x_{001} + \frac{1}{8}x_{101} + \frac{1}{8}x_{011} + \frac{1}{8}x_{111}$$
$$+ e^3_{000} + \frac{1}{2}e^2_{000} + \frac{1}{2}e^2_{001} + \frac{1}{4}e^1_{000} + \frac{1}{4}e^1_{010} + \frac{1}{4}e^1_{001} + \frac{1}{4}e^1_{011}$$

- Divide rounding errors into two groups:

Step 4: Handling the Deviation

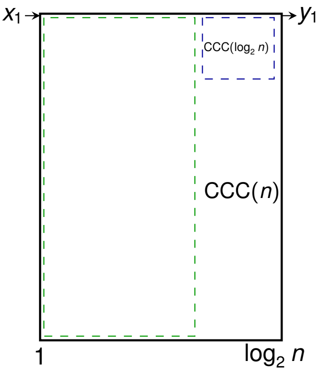
$$y_{000} = \frac{1}{8}x_{000} + \frac{1}{8}x_{100} + \frac{1}{8}x_{010} + \frac{1}{8}x_{110} + \frac{1}{8}x_{001} + \frac{1}{8}x_{101} + \frac{1}{8}x_{011} + \frac{1}{8}x_{111}$$
$$+ e^3_{000} + \frac{1}{2}e^2_{000} + \frac{1}{2}e^2_{001} + \frac{1}{4}e^1_{000} + \frac{1}{4}e^1_{010} + \frac{1}{4}e^1_{001} + \frac{1}{4}e^1_{011}$$

- Divide rounding errors into two groups:
  1. Layers $1, \ldots, \log_2 n - \log_2 \log_2 n$
  2. Layers $\log_2 n - \log_2 \log_2 n + 1, \ldots, \log_2 n$

Step 4: Handling the Deviation

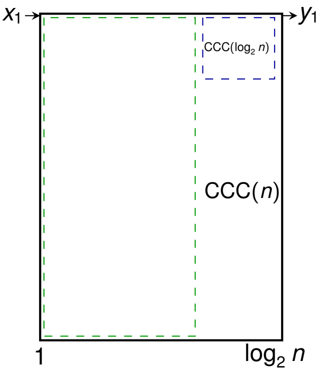
$$y_{000} = \frac{1}{8}x_{000} + \frac{1}{8}x_{100} + \frac{1}{8}x_{010} + \frac{1}{8}x_{110} + \frac{1}{8}x_{001} + \frac{1}{8}x_{101} + \frac{1}{8}x_{011} + \frac{1}{8}x_{111}$$
$$+ e^{3}_{000} + \frac{1}{2}e^{2}_{000} + \frac{1}{2}e^{2}_{001} + \frac{1}{4}e^{1}_{000} + \frac{1}{4}e^{1}_{010} + \frac{1}{4}e^{1}_{001} + \frac{1}{4}e^{1}_{011}$$

- Divide rounding errors into two groups:
  1. Layers $1, \ldots, \log_2 n - \log_2 \log_2 n$
  2. Layers $\log_2 n - \log_2 \log_2 n + 1, \ldots, \log_2 n$



Step 4: Handling the Deviation

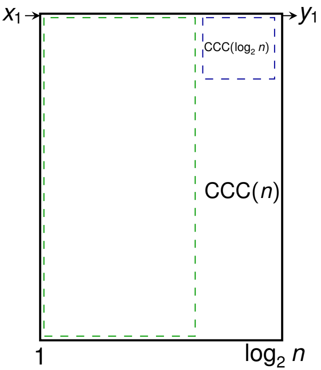
$$y_{000} = \frac{1}{8}x_{000} + \frac{1}{8}x_{100} + \frac{1}{8}x_{010} + \frac{1}{8}x_{110} + \frac{1}{8}x_{001} + \frac{1}{8}x_{101} + \frac{1}{8}x_{011} + \frac{1}{8}x_{111}$$
$$+ e^3_{000} + \frac{1}{2}e^2_{000} + \frac{1}{2}e^2_{001} + \frac{1}{4}e^1_{000} + \frac{1}{4}e^1_{010} + \frac{1}{4}e^1_{001} + \frac{1}{4}e^1_{011}$$



- Divide rounding errors into two groups:
  1. Layers $1, \ldots, \log_2 n - \log_2 \log_2 n$
  2. Layers $\log_2 n - \log_2 \log_2 n + 1, \ldots, \log_2 n$
- Chernoff $\Rightarrow$ first group contributes ≤ 2

Step 4: Handling the Deviation

$$y_{000} = \frac{1}{8}x_{000} + \frac{1}{8}x_{100} + \frac{1}{8}x_{010} + \frac{1}{8}x_{110} + \frac{1}{8}x_{001} + \frac{1}{8}x_{101} + \frac{1}{8}x_{011} + \frac{1}{8}x_{111}$$
$$+ e^{3}_{000} + \frac{1}{2}e^{2}_{000} + \frac{1}{2}e^{2}_{001} + \frac{1}{4}e^{1}_{000} + \frac{1}{4}e^{1}_{010} + \frac{1}{4}e^{1}_{001} + \frac{1}{4}e^{1}_{011}$$



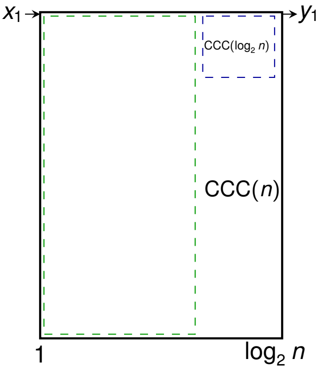
- Divide rounding errors into two groups:
  1. Layers $1, \ldots, \log_2 n - \log_2 \log_2 n$
  2. Layers $\log_2 n - \log_2 \log_2 n + 1, \ldots, \log_2 n$
- Chernoff $\Rightarrow$ first group contributes ≤ 2
- trivial bound $\Rightarrow$ second group contributes $\frac{1}{2} \log_2 \log_2 n$

Step 4: Handling the Deviation

$$y_{000} = \frac{1}{8}x_{000} + \frac{1}{8}x_{100} + \frac{1}{8}x_{010} + \frac{1}{8}x_{110} + \frac{1}{8}x_{001} + \frac{1}{8}x_{101} + \frac{1}{8}x_{011} + \frac{1}{8}x_{111}$$
$$+ e^3_{000} + \frac{1}{2}e^2_{000} + \frac{1}{2}e^2_{001} + \frac{1}{4}e^1_{000} + \frac{1}{4}e^1_{010} + \frac{1}{4}e^1_{001} + \frac{1}{4}e^1_{011}$$

- Divide rounding errors into two groups:
  1. Layers $1, \ldots, \log_2 n - \log_2 \log_2 n$
  2. Layers $\log_2 n - \log_2 \log_2 n + 1, \ldots, \log_2 n$
- Chernoff $\Rightarrow$ first group contributes ≤ 2
- trivial bound $\Rightarrow$ second group contributes $\frac{1}{2} \log_2 \log_2 n$

$\Rightarrow$ discrepancy is at most $\log_2 \log_2 n + 4$ $\square$



Lower Bound for a single CCC

Upper Bound (Mavronicolas, S., 2010)

For any input the discrepancy is at most $\log_2 \log_2 n + 4$ w.p. $1 - n^{-1}$.

Lower Bound for a single CCC

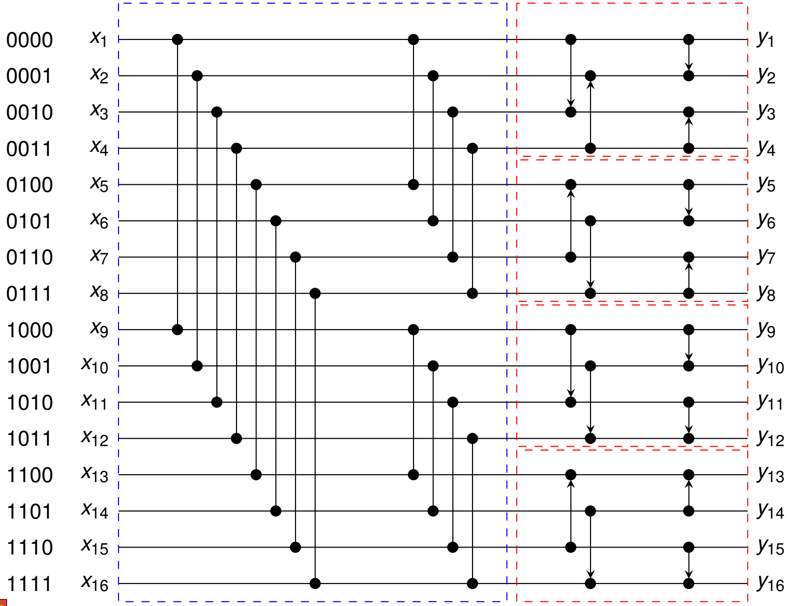
Upper Bound (Mavronicolas, S., 2010)

For any input the discrepancy is at most $\log_2 \log_2 n + 4$ w.p. $1 - n^{-1}$.

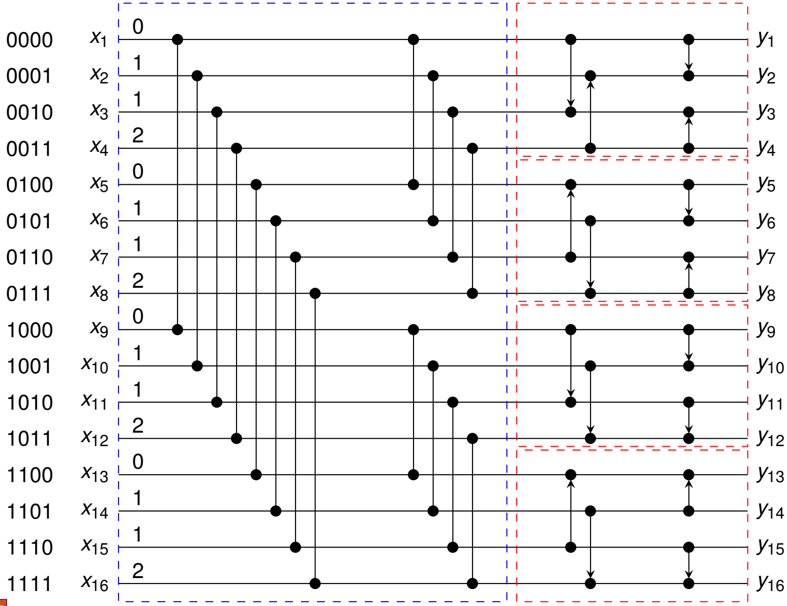
Lower Bound (Mavronicolas, S., 2010)

For some inputs the discrepancy is at least $\log_2 \log_2 n - 2$ w.p. $1 - n^{-1}$.

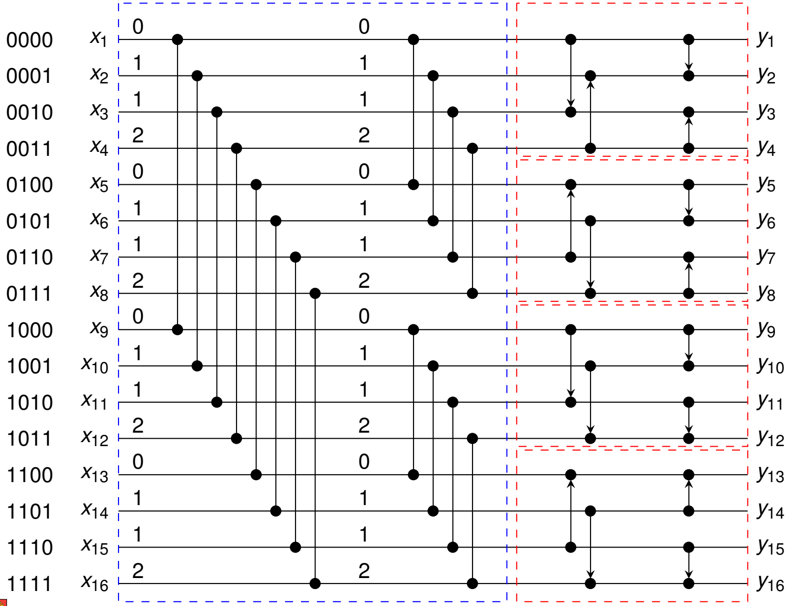
Lower Bound (Proof Idea)



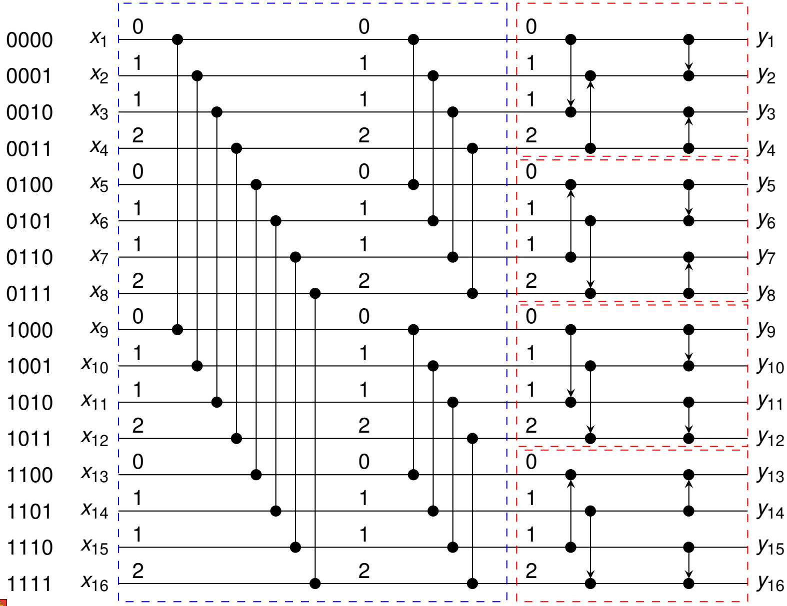
Lower Bound (Proof Idea)



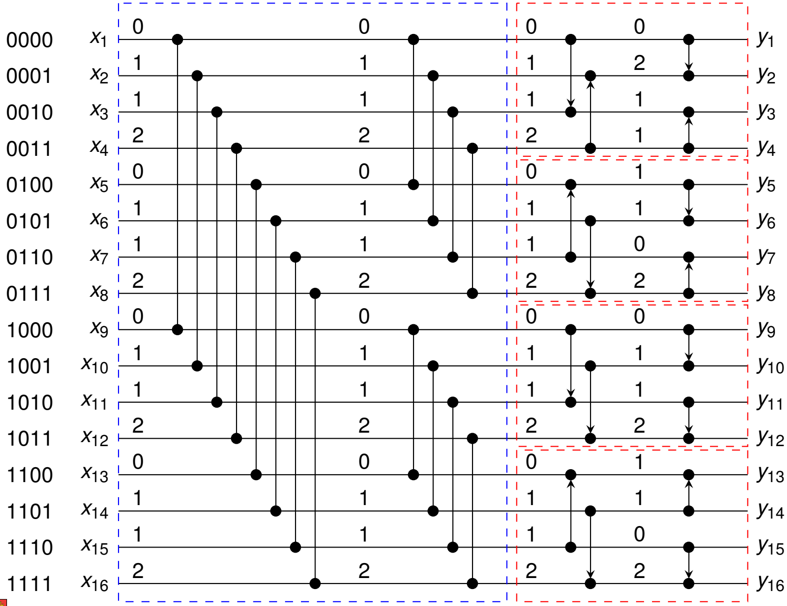
Lower Bound (Proof Idea)



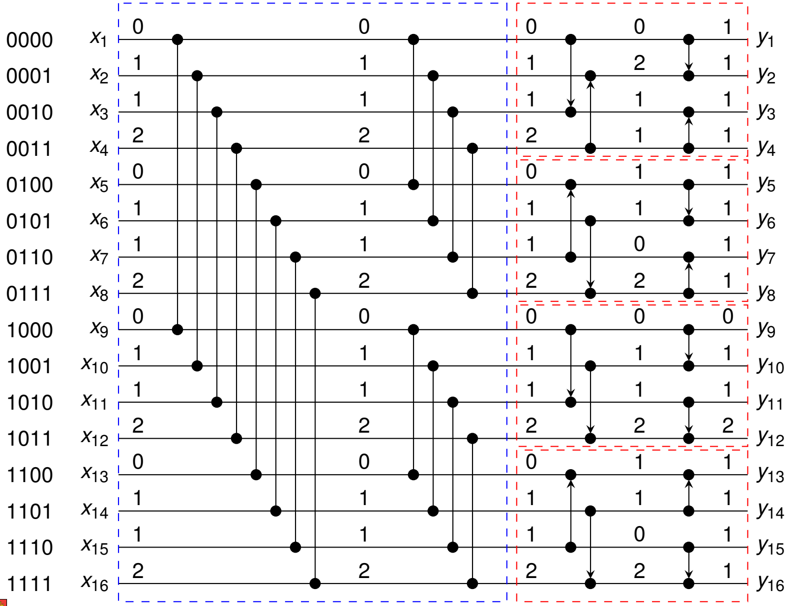
Lower Bound (Proof Idea)



Lower Bound (Proof Idea)



Lower Bound (Proof Idea)



Further Thoughts on the Lower Bound

Input for lower bound look contrived, so what about **random inputs**?

Further Thoughts on the Lower Bound

Input for lower bound look contrived, so what about **random inputs**?

Average-Case Model

Assume number of tokens at each input wire is $\sim Uni[0, \log_2 n - 1]$.

Further Thoughts on the Lower Bound

Input for lower bound look contrived, so what about **random inputs**?

One can prove that the range $[0, \log_2 n - 1]$ is canonical.

Average-Case Mod

Assume number of tokens at each input wire is $\sim \mathit{Uni}[0, \log_2 n - 1]$.

Further Thoughts on the Lower Bound

Input for lower bound look contrived, so what about **random inputs**?

One can prove that the range $[0, \log_2 n - 1]$ is canonical.

Average-Case Mod

Assume number of tokens at each input wire is $\sim Uni[0, \log_2 n - 1]$.

Friedrich, Vilenchik, S.'11

For this input, discrepancy is at least $(1/2 - o(1)) \log_2 \log_2 n$ w.p. $1 - o(1)$.

Further Thoughts on the Lower Bound

Input for lower bound look contrived, so what about **random inputs**?

One can prove that the range $[0, \log_2 n - 1]$ is canonical.

Average-Case Mod

Assume number of tokens at each input wire is $\sim \mathit{Uni}[0, \log_2 n - 1]$.

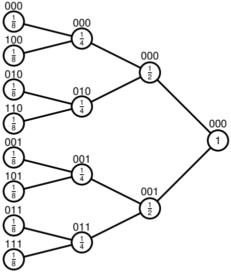
Friedrich, Vilenchik, S.'11

For this input, discrepancy is at least $(1/2 - o(1)) \log_2 \log_2 n$ w.p. $1 - o(1)$.

"Magic Property":

- All rounding errors become **independent**(!) random variables

$$\begin{cases} -1/2 & \text{with probability } 1/4 \\ 0 & \text{with probability } 1/2 \\ +1/2 & \text{with probability } 1/4 \end{cases}$$



Cascading more CCC's

Mavronicolas, S., 2010

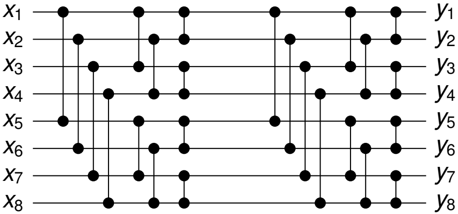
For any input to the CCC, the discrepancy is $\log_2 \log_2 n \pm O(1)$.

Cascading more CCC's

Mavronicolas, S., 2010

For any input to the CCC, the discrepancy is $\log_2 \log_2 n \pm O(1)$.

What happens if we take the cascade of two or more CCC's?

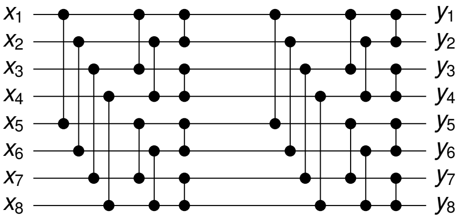


Cascading more CCC's

Mavronicolas, S., 2010

For any input to the CCC, the discrepancy is $\log_2 \log_2 n \pm O(1)$.

What happens if we take the cascade of two or more CCC's?



Mavronicolas, S., 2010

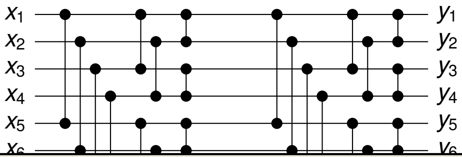
For any input the discrepancy is at most ~~16~~ 3 w.p. $1 - n^{-1}$.

Cascading more CCC's

Mavronicolas, S., 2010

For any input to the CCC, the discrepancy is $\log_2 \log_2 n \pm O(1)$.

What happens if we take the cascade of two or more CCC's?



Proof more involved than the analysis of one CCC.
Relies heavily on the symmetry and recursive structure.

Mavronicolas, S., 2010

For any input the discrepancy is at most ~~16~~ 3 w.p. $1 - n^{-1}$.

Outline

- Introduction
- Sorting Networks
- Counting Networks
- Randomized Smoothing Networks
- Stronger Notions of Smoothing Networks
- Conclusion

A (More) "Universal" Model

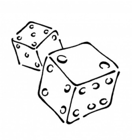
(Standard) Smoothing Network

- deterministic initialization: input arbitrary
- random initialization: input arbitrary, but without knowing initialization

A (More) "Universal" Model

(Standard) Smoothing Network

- deterministic initialization: input arbitrary
- random initialization: input arbitrary, but without knowing initialization



$\forall \vec{x}$: $\mathbf{Pr}\left[\ CCC(\vec{x}) \text{ is } \lambda\text{-smooth}\right] \geq 1 - n^{-1}$

A (More) "Universal" Model

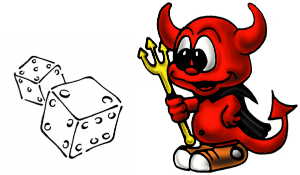
(Standard) Smoothing Network

- deterministic initialization: input arbitrary
- random initialization: input arbitrary, but without knowing initialization

$\forall \vec{x}$: **Pr** $\left[\ CCC(\vec{x}) \text{ is } \lambda\text{-smooth} \right] \geq 1 - n^{-1}$

Universal Randomized Smoothing Network

- random initialization: input arbitrary with knowledge of initialization



A (More) "Universal" Model

(Standard) Smoothing Network

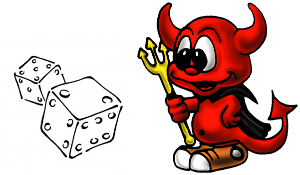
- deterministic initialization: input arbitrary
- random initialization: input arbitrary, but without knowing initialization

$\forall \vec{x}$: **Pr** $\left[\ CCC(\vec{x}) \text{ is } \lambda\text{-smooth} \right] \geq 1 - n^{-1}$

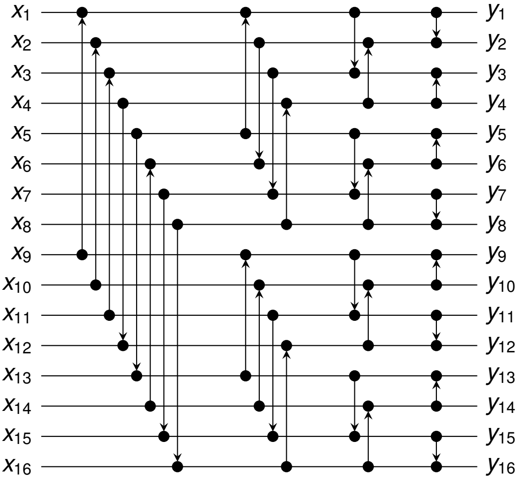
Universal Randomized Smoothing Network

- random initialization: input arbitrary with knowledge of initialization

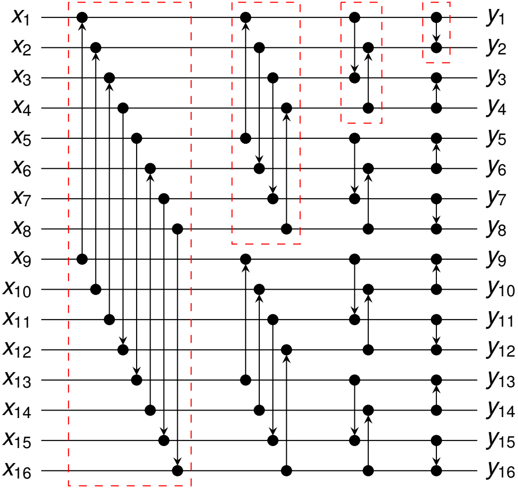
Pr $\left[\forall \vec{x}:\ CCC(\vec{x}) \text{ is } \lambda\text{-smooth} \right] \geq 1 - n^{-1}$



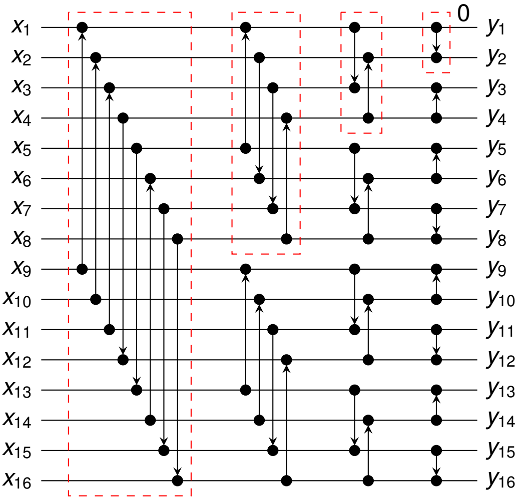
One CCC as a Universal Randomized Smoothing Network?



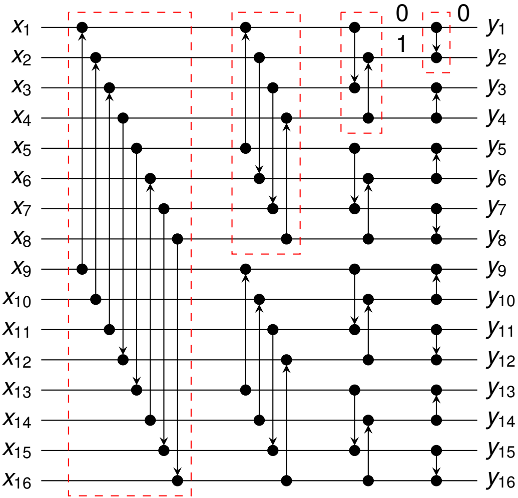
One CCC as a Universal Randomized Smoothing Network?



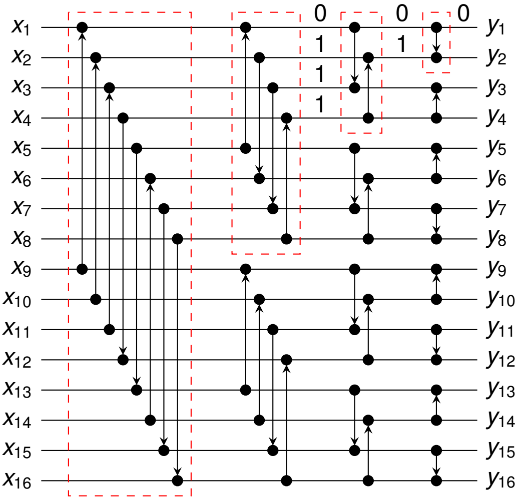
One CCC as a Universal Randomized Smoothing Network?



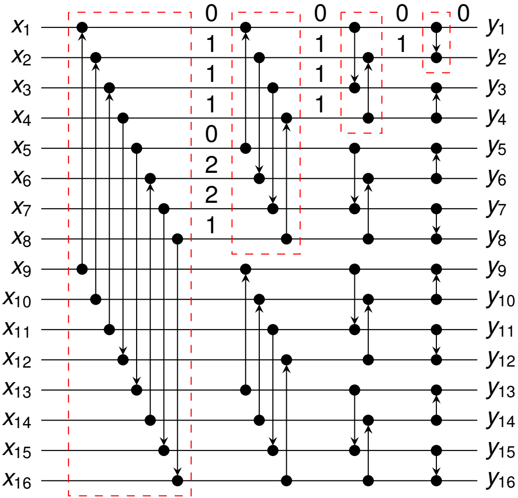
One CCC as a Universal Randomized Smoothing Network?



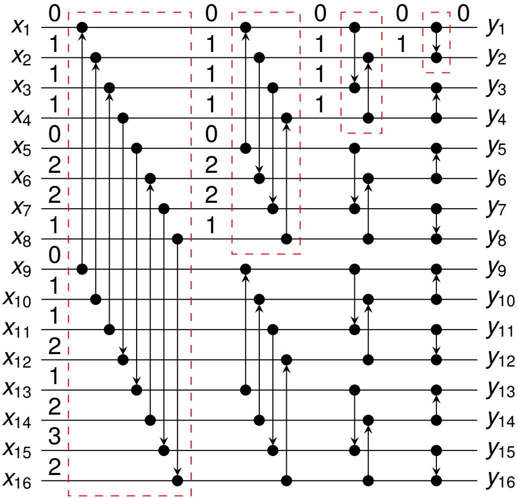
One CCC as a Universal Randomized Smoothing Network?



One CCC as a Universal Randomized Smoothing Network?



One CCC as a Universal Randomized Smoothing Network?



Results (and Conjectures)

Lower Bound (Mavronicolas, S.'10)

For any initialisation of the CCC, there exists an input so that the discrepancy is at least $(1/4)\log_2 n$.

Results (and Conjectures)

Lower Bound (Mavronicolas, S.'10)

For any initialisation of the CCC, there exists an input so that the discrepancy is at least $(1/4)\log_2 n$.

Is the cascade of two (or more) CCC's a universal smoothing network?

Results (and Conjectures)

Lower Bound (Mavronicolas, S.'10)

For any initialisation of the CCC, there exists an input so that the discrepancy is at least $(1/4)\log_2 n$.

Is the cascade of two (or more) CCC's a universal smoothing network?

- Basic lower bound analysis does not work...
- Proving positive result is also challenging because:
  - number of possible inputs is $(\log n)^n$
  - probabilistic analysis gives error bounds like $n^{-\Theta(\text{polylog}(n))}$ at best

Results (and Conjectures)

Lower Bound (Mavronicolas, S.'10)

For any initialisation of the CCC, there exists an input so that the discrepancy is at least $(1/4)\log_2 n$.

Is the cascade of two (or more) CCC's a universal smoothing network?

- Basic lower bound analysis does not work...
- Proving positive result is also challenging because:
  - number of possible inputs is $(\log n)^n$
  - probabilistic analysis gives error bounds like $n^{-\Theta(\text{polylog}(n))}$ at best

Conjecture (Kosowski, S.)

There is a constant $\epsilon > 0$, so that cascading $\Theta(\text{polylog}(n))$ CCC's achieves a discrepancy of at most $O((\log n)^{1-\epsilon})$.

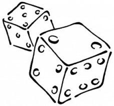
???

An Even More “Universal” Model

Universal Randomized Smoothing Network

- random initialization: input arbitrary with knowledge of initialization

$\mathbf{Pr}\left[\forall\vec{x}:\ \ CCC(\vec{x}) \text{ is } \lambda\text{-smooth}\right] \geq 1 - n^{-1}$

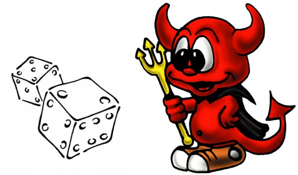


An Even More “Universal” Model

Universal Randomized Smoothing Network

- random initialization: input arbitrary with knowledge of initialization

$\textbf{Pr}\left[\forall \vec{x}: \ CCC(\vec{x}) \text{ is } \lambda\text{-smooth}\right] \geq 1 - n^{-1}$



“Doubly” Adversarial Model

- input and initialisation controlled by an adversary

Some First Results on the Adversarial Model

Observation

One CCC achieves discrepancy of $\log_2 n$ for any input and initialisation.

Some First Results on the Adversarial Model

Observation

One CCC achieves discrepancy of $\log_2 n$ for any input and initialisation.

Conjecture (Kosowski, S., 2016)

Take the cascade of $O(\text{polylog}(n))$ random perfect matchings. Then the discrepancy is at most $O(\log n / \log \log n)$.

Some First Results on the Adversarial Model

Observation

One CCC achieves discrepancy of $\log_2 n$ for any input and initialisation.

Conjecture (Kosowski, S., 2016)

Take the cascade of $O(\text{polylog}(n))$ random perfect matchings. Then the discrepancy is at most $O(\log n / \log \log n)$.

Lower Bound

For any universal smoothing network of depth d, there is an input so that the discrepancy is at least $\frac{\log n}{\log d}$.

Some First Results on the Adversarial Model

Observation

One CCC achieves discrepancy of $\log_2 n$ for any input and initialisation.

Conjecture (Kosowski, S., 2016)

Take the cascade of $O(\text{polylog}(n))$ random perfect matchings. Then the discrepancy is at most $O(\log n / \log\log n)$.

Lower Bound

For any universal smoothing network of depth d, there is an input so that the discrepancy is at least $\frac{\log n}{\log d}$.

↑

Lemma

For any graph with maxdegree Δ, $\text{diam}(G) \geq \log n/(\log \Delta)$.

Some First Results on the Adversarial Model

Observation

One CCC achieves discrepancy of $\log_2 n$ for any input and initialisation.

Conjecture (Kosowski, S., 2016)

Take the cascade of $O(\text{polylog}(n))$ random perfect matchings. Then the discrepancy is at most $O(\log n / \log\log n)$.

Lower Bound

For any universal smoothing network of depth d, there is an input so that the discrepancy is at least $\frac{\log n}{\log d}$.

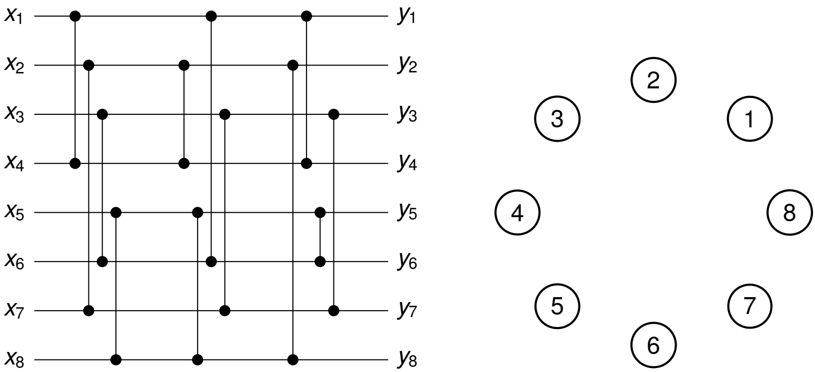
↑

Lemma

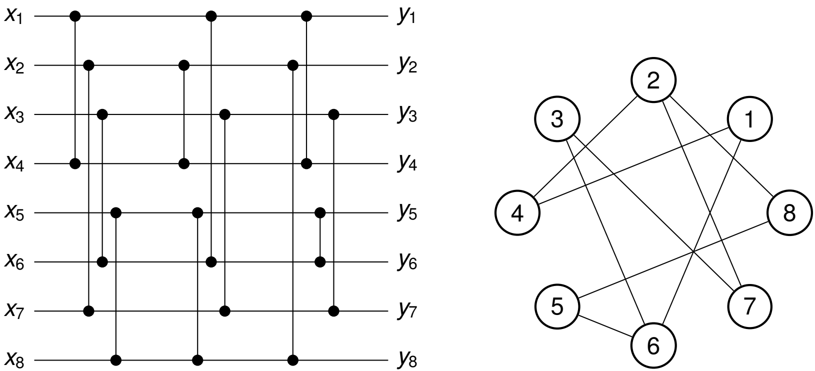
For any graph with maxdegree Δ, $\text{diam}(G) \geq \log n/(\log \Delta)$.

One can reach at most Δ^k vertices from any node in k hops.

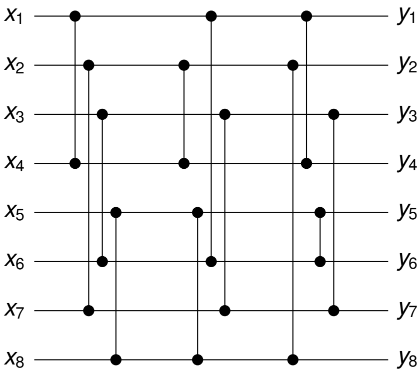
Proof Sketch of Lower Bound

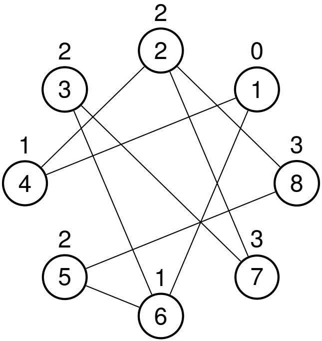


Proof Sketch of Lower Bound

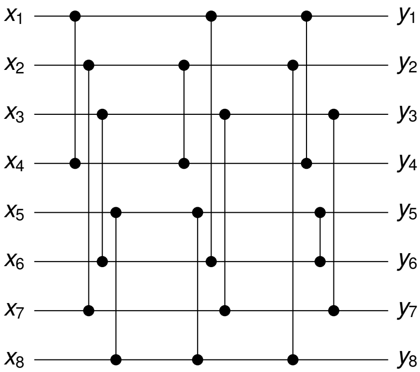


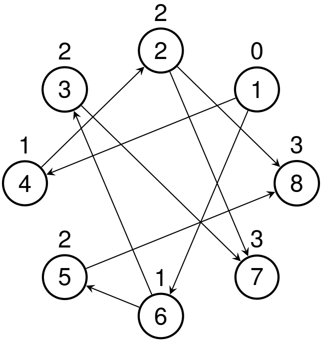
Proof Sketch of Lower Bound



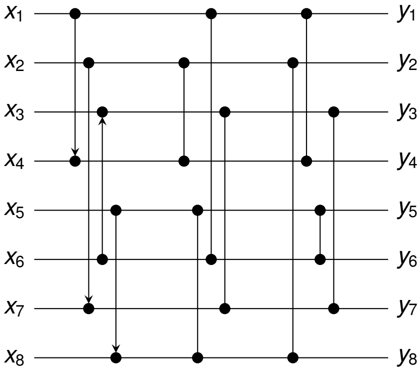


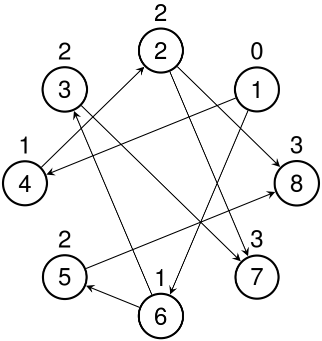
Proof Sketch of Lower Bound



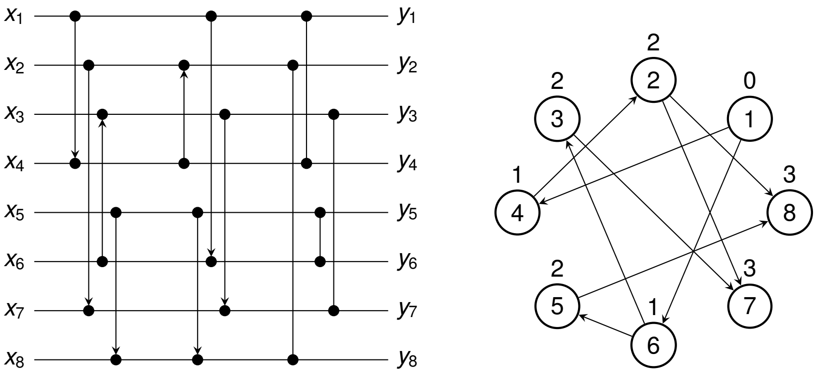


Proof Sketch of Lower Bound

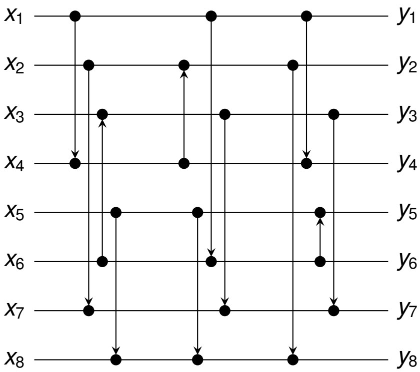


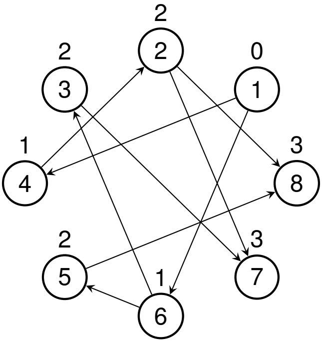


Proof Sketch of Lower Bound

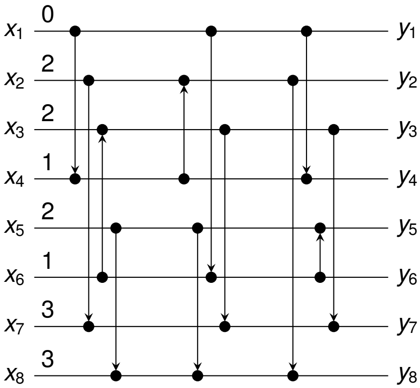


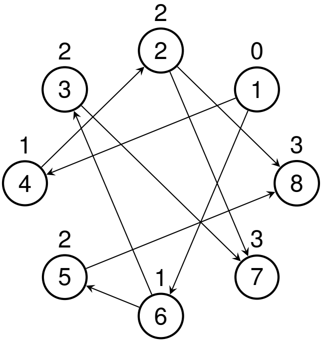
Proof Sketch of Lower Bound



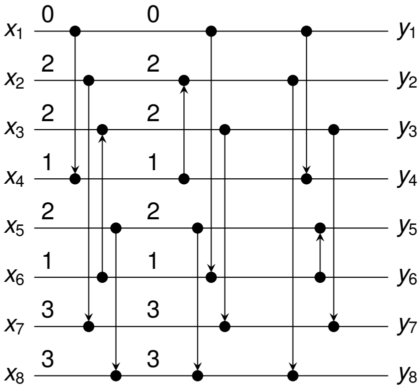


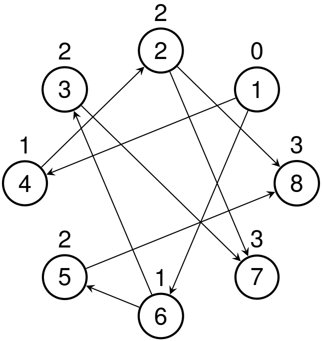
Proof Sketch of Lower Bound



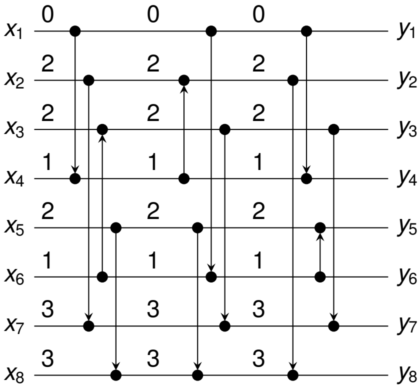


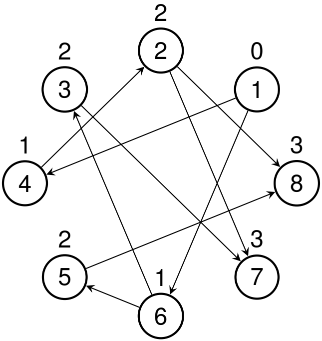
Proof Sketch of Lower Bound



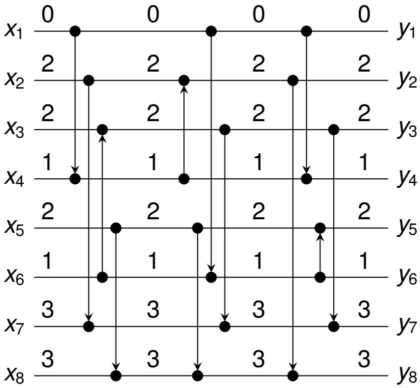


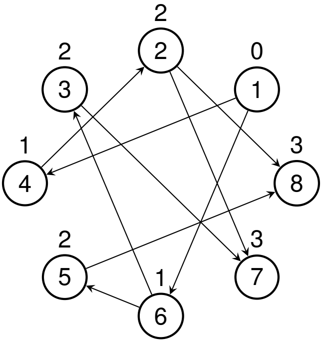
Proof Sketch of Lower Bound





Proof Sketch of Lower Bound





Outline

Summary

Model / Discrepancy	1	2	$(\log n)^{1-\epsilon}$	$\frac{\log n}{\log\log n}$	$\log n$
Counting	$\Theta(\log n)$	✓	✓	✓	✓
Random Smoothing	$\Omega(n)$	$O(\log n)$	✓	✓	✓
Random Universal	??	??	$O(\mathrm{polylog}(n))$	✓	$\log n$
Adversarial	∞	$\Omega(n)$	$\Omega(2^{(\log n)^{1-\epsilon}})$	$O(\mathrm{polylog}(n))$	$\log n$

Summary

Model / Discrepancy	1	2	$(\log n)^{1-\epsilon}$	$\frac{\log n}{\log\log n}$	$\log n$
Counting	$\Theta(\log n)$	✓	✓	✓	✓
Random Smoothing	$\Omega(n)$	$O(\log n)$	✓	✓	✓
Random Universal	??	??	$O(\mathrm{polylog}(n))$	✓	$\log n$
Adversarial	∞	$\Omega(n)$	$\Omega(2^{(\log n)^{1-\epsilon}})$	$O(\mathrm{polylog}(n))$	$\log n$

Random Smoothing on Arbitrary Graphs (Sun, S.'12)

For arbitrary graphs, one can achieve a constant discrepancy in $\Theta(\log(Kn)/(1-\lambda_2))$ rounds, where K is the initial discrepancy and $1-\lambda_2$ is the spectral gap.

Summary

Model / Discrepancy	1	2	$(\log n)^{1-\epsilon}$	$\frac{\log n}{\log \log n}$	$\log n$
Counting	$\Theta(\log n)$	✓	✓	✓	✓
Random Smoothing	$\Omega(n)$	$O(\log n)$	✓	✓	✓
Random Universal	??	??	$O(\text{polylog}(n))$	✓	$\log n$
Adversarial	∞	$\Omega(n)$	$\Omega(2^{(\log n)^{1-\epsilon}})$	$O(\text{polylog}(n))$	$\log n$

Random Smoothing on Arbitrary Graphs (Sun, S.'12)

For arbitrary graphs, one can achieve a constant discrepancy in $\Theta(\log(Kn)/(1-\lambda_2))$ rounds, where K is the initial discrepancy and $1-\lambda_2$ is the spectral gap.

- random smoothing networks appear to be relatively well-understood
- first results in the universal/adversarial model are sketchy, but hint at an interesting landscape