Periodic constant depth sorting networks*

Marcin Kik, † Mirosław Kutyłowski, ‡ Grzegorz Stachowiak, §

Abstract

Comparator networks of constant depth can be used for sorting in the following way. The computation consists of a number of rounds, say t, each round being a single run through the comparator network. The output of a round j ($j < t$) is used as the input for the round $j+1$. The output of the round t is the output of the computation. In such a way, we may apply a small comparator network for sorting long input sequences. The main problem, however, is achieving a good computation time.

Odd-Even Transposition Sort gives a periodic sorting network of depth 2, that sorts n numbers in $n/2$ iterations. The network of depth 8 proposed by Schwiegelshohn [8] sorts n numbers in $O(\sqrt{n}\log n)$ iterations. Krammer [4] modified the algorithm and obtained a network of depth 6 sorting in $O(\sqrt{n\log n})$ iterations.

For a fixed but arbitrary $k \in \mathbb{N}$, we present a periodic sorting network of depth $O(k)$ that sorts n input numbers in $O(n^{1/k})$ steps.

1 Introduction

Comparator networks are widely used for sorting sequences of numbers. A comparator network of depth d sorting n input numbers may be viewed as a set of n registers and a set of communication links between the registers divided into d so called levels. Each level can be represented by a directed graph of degree one, i.e. a single register may be connected to at most one other register at a given level. At step $i \leq d$ of the computation, all links of the ith level are used in parallel to perform comparison-exchange operations between the elements stored in different registers. For a given link (R, R'), if numbers x and x' are stored in R and R', respectively, then after the compare-exchange operation R stores $\min(x, x')$ and R' stores $\max(x, x')$.

A large number of sorting comparator networks has been proposed in the literature. The most successful in practice have been the networks proposed by Batcher [2]. They are very elegant in design and have depth $\frac{1}{2}(\log n)^2$. There has been a lot of effort to reduce the depth of sorting comparator networks to $\log n$. This was achieved by the famous AKS network of depth $O(\log n)$ [1]. However, the result is of purely theoretical importance, because of large constants involved.

Except the relatively large depth, the networks of Batcher have yet another disadvantage. Every level is different from another. Therefore, if the sorting network is directly wired in a VLSI chip, then there is a large number of wires put into circuitry, making the chip large and expensive. If the comparator network is emulated by a set of processors, then there is a lot of overhead due to routing along many different paths. One may try to overcome these difficulties by applying periodic sorting algorithms [3]. We may use a single comparator network (presumably of a small depth) repeatedly: after getting output that is still not sorted, we put it as an input to the network again. We stop once the output is sorted. The total computation time is therefore $\delta \cdot T$, where δ is the depth of the network and T is the number of iterations. Dowd *et al* [3] proposed a network of depth $\log n$ that sorts in $\log n$ iterations. All communication links of this network are those of an $\log n$-dimensional hypercube. If the links are restricted to those of a $\sqrt{n} \times \sqrt{n}$-mesh, then we can sort in $\log n$ iterations, but the depth of

*supported by KBN grant 2 1197 91 01 and Volkswagen Foundation, Project "Paralleles Rechnen: Theoretische und experimentelle Untersuchungen zu parallelen Rechnenmodellen und systemnahen Algorithmen", partially this work was done while the first and the second author visited Heinz–Nixdorf-Institut, Universität Paderborn

†kik@plwruw11.bitnet

‡mirekk@uni-paderborn.de

§gst@plwruw11.bitnet,
University of Wrocław, ul. Przesmyckiego 20, PL-51-151 Wrocław, Poland

the network is $2\sqrt{n}$ [7].

To achieve low cost sorting circuits for large inputs it would be desirable to reduce the depth of comparator networks used for periodic sorting to a constant while preserving a small number of iterations. The question whether such networks exist has been raised by Meyer auf der Heide [6]. For example, Odd-Even Transposition Sort leads to a network of depth 2 that sorts in $n/2$ iterations, i.e. n parallel steps [5]. Schwiegelshohn's network of depth 8 sorts n elements in $O(\sqrt{n}\log n)$ iterations [8]. Krammer [4] modified the construction of Schwiegelshohn and obtained a network of depth 6 sorting in $O(\sqrt{n\log n})$ iterations.

In this paper we prove the following theorem.

Theorem 1.1 *Let $k \in \mathbb{N}$ be an arbitrary constant. Let $n \in \mathbb{N}$. There is a comparator network of depth $O(k)$ that sorts n numbers in $O(k \cdot n^{1/k})$ iterations, i.e. in time $O(k^2 \cdot n^{1/k})$.*

The proof of Theorem 1.1 is constructive – we show how to build the required network using ε-halver networks of Ajtai, Komlós and Szemerédi. However, our network is not applicable directly in practice, since the constants are too large and wiring due to expanders is complicated. (The large constants may be due to the present mathematical analysis of the performance of the network.)

2 Preliminaries

Definition 2.1
A comparator network $N=(V,C_1,\dots,C_d)$ *for inputs sequences of n numbers consists of a set V of n registers $R_1,\dots,R_n$ and d directed graphs $C_1,\dots,C_d$, each of degree 1, with vertices in V. For $i \le d$, C_i is called the ith* level *of N; d is called the* depth *of N. Network N works as follows. Initially, the input numbers are stored in the registers, the ith number in the register R_i. At step t ($t \le d$), for each arc $(R_j, R_{j'}) \in C_t$ the numbers stored in R_j and $R_{j'}$ are compared. The minimum of them is put into R_j, the maximum into $R_{j'}$. (Since C_t has degree 1, there is at most one arc incident to a given vertex and the definition is unambiguous.) We say that the registers of N store a sorted sequence if for every $i < i'$ the number stored in the register R_i is not greater than the number stored in the register $R_{i'}$.*

A comparator network N is monotonic *if $j < j'$ for every arc $(R_j, R_{j'}) \in C_t$ ($t \le d$).*

All comparator networks considered in this paper are monotonic.

Definition 2.2 *(a) For a comparator network N, let $N(\vec{x})$ denote the output computed by N on input sequence $\vec{x}$. Let N^j be defined by $N^1(\vec{x}) = N(\vec{x})$ and $N^{j+1}(\vec{x}) = N(N^j(\vec{x}))$ for $i \ge 1$.*
(b) We say that network N sorts a sequence $\vec{x}$ in at most t iterations, if the sequence $N^t(\vec{x})$ is sorted.

To prove that an oblivious algorithm sorts in a stated time one can use well known 0–1 *Principle* (see e.g [5]). (An algorithm is *oblivious*, if it performs only compare-exchange operations, and for each moment of the computation, these operations take place between exactly the same registers for every input sequence.) Clearly, iterating computation of a comparator network leads to an oblivious algorithm.

Lemma 2.3 (0–1 Principle) *An oblivious algorithm correctly sorts every input sequence if and only if it correctly sorts every input sequence consisting of 0's and 1's.*

3 Construction of the network

In our construction we use ε-halver networks introduced in [1]. We recall their definition and basic properties.

Definition 3.1 *Let $\varepsilon > 0$. A comparator network N for inputs consisting of n numbers is an ε-*halver *if the following holds:*

- *The set of registers of N consists of two subsets V_1 and V_2 of cardinality $n/2$, and all comparison arcs point from V_1 to V_2.*
- *Let the input for N contain k ones and $n-k$ zeroes. If $k \le n/2$, then the output of N contains at most $\varepsilon \cdot k$ ones stored in V_1. If $k \ge n/2$, then the output of N contains at most $\varepsilon \cdot (n-k)$ zeroes stored in V_2.*

We shall also say that N is a (V_1, V_2, ε)-halver.

Lemma 3.2 [1] *For each $\varepsilon > 0$ and $n \in \mathbb{N}$, there is an ε-halver for inputs consisting of n numbers which has depth $O(1/\varepsilon \cdot \log(1/\varepsilon))$.*

The construction of ε-halver networks is based on expander graphs. In fact, ε-halver networks are basic components of the AKS sorting network.

Definition 3.3 *Let $N = (V, C_1, \dots, C_d)$ and $N' = (V', C'_1, \dots, C'_{d'})$ be comparator networks. If $V \cap V' = \emptyset$ and $d = d'$, then by $N \cup N'$ we mean the network $(V \cup V', C_1 \cup C'_1, \dots, C_d \cup C'_d)$. If $V = V'$, then $N|N'$ denotes the network $(V, C_1, \dots, C_d, C'_1, \dots, C'_{d'})$.*

A key component of our sorting network is an (ε, m)-block that we define below.

Definition 3.4 *An* (ε, m)-block *built on registers* $R_1, \ldots, R_n$ *is defined as follows. Let* $l = \lceil \frac{n}{m} \rceil$. *For* $i < l$, *let* $K_i = \{R_{m\cdot(i-1)+1}, \ldots, R_{m\cdot i}\}$, *and* $K_l = \{R_{m\cdot(l-1)+1}, \ldots, R_n\}$. *Let* E_i *be an* $(K_i, K_{i+1}, \varepsilon)$-*halver. (If* K_l *consists of less than* m *registers, then we add dummy registers to make* K_l *contain* m *registers, we take an* $(K_{l-1}, K_l, \varepsilon)$-*halver and remove all arcs pointing to the dummy registers;* E_{l-1} *consists of the arcs that remain.) Then an* (ε, m)-*block is the network*

$$(E_1 \cup E_3 \cup \cdots \cup E_{l-1})|(E_2 \cup E_4 \cup \cdots \cup E_{l-2})$$

if l *is even; for* l *odd,* (ε, m)-*block is the network*

$$(E_1 \cup E_3 \cup \cdots \cup E_{l-2})|(E_2 \cup E_4 \cup \cdots \cup E_{l-1}).$$

The network $E_1 \cup E_3 \cup \cdots \cup E_{l-1}$ *(or* $E_1 \cup E_3 \cup \cdots \cup E_{l-2}$ *if* l *is odd) is called* the first layer *of the* (ε, m)-*block.*

In particular, a $(0,1)$-*block is a network* $(\{R_1, \ldots, R_n\}, C_1, C_2)$, *where* $C_1 = \{(R_1, R_2), (R_3, R_4), \ldots\}$ *and* $C_2 = \{(R_2, R_3), (R_4, R_5), \ldots\}$

We can imagine the registers of an (ε, m)-block to be arranged in an $m \times l$-matrix, with K_i being the ith column of the matrix. Between each pair of columns we put an ε-halver . During the first part of the computation we make comparisons between the elements of K_1 and K_2, K_3 and K_4, ... During the second part of the computation, we make comparisons between the elements of K_2 and K_3, K_4 and K_5, ... The crucial property of (ε, m)-blocks is that in $O(l)$ iterations each element is moved to a column not far from the column of its final destination in the sorted sequence (to be proved in Section 4).

Now we complete construction of the comparator network that sorts n numbers in $O(n^{1/k})$ iterations:

Definition 3.5 *For* $n, k \in \mathbb{N}$, *let* $I_{n,k}$ *be a comparator network of the form*

$$I_{n,k} = F_1|F_2|\cdots|F_k|F_{k+1}$$

where F_{k+1} *is a* $(0,1)$-*block, and for* $i = 1, \ldots, k$, F_i *is an* $(\varepsilon, n^{(k-i)/k} \cdot (c \cdot \log n)^{i-1})$-*block. The constants* c *and* ε $(0 < \varepsilon < \frac{1}{2})$ *will be determined later.*

It follows from the definition that $I_{n,k}$ has depth $2\delta k + 2$, where $\delta = O(\frac{1}{\varepsilon} \cdot \log(\frac{1}{\varepsilon}))$ denotes the depth of the ε-halvers used in the construction.

4 The time bounds

This section is organized as follows. First we analyze computation of a single (ε, m)-block. In Subsection 4.1 we formulate the main lemma and make some straightforward observations. Subsections 4.2, 4.3, 4.4 are devoted to the proof of the main lemma. Finally, in Subsection 4.5 we use the main lemma to estimate the number of iterations required by the network $I_{n,k}$.

4.1 Main lemma

Let B be an (ε, m)-block consisting of registers $R_1, \ldots, R_n$, $l = \lceil \frac{n}{m} \rceil$. By $K_i = K_i(B, m)$ we denote the ith column of B of height m, i.e., $K_i = \{R_{m(i-1)+1}, \ldots, R_{mi}\}$, for $i < l$, and $K_l = \{R_{m(l-1)+1}, \ldots, R_n\}$. This notation shall be used throughout the whole section.

Definition 4.1 *Let* $C = X_1|B|X_2$ *and let the registers of* C *store* 0*'s and* 1*'s, only. Then we say that network* C *is at most* (p, m)-*dirty, if there is* j *such that the columns* $K_1, \ldots, K_j$ *contain only zeroes, and the columns* $K_{j+p+1}, \ldots, K_l$ *contain only ones.*

Lemma 4.2 (Main Lemma) *Let* B' *be a comparator network such that* $B' = X_1|B|X_2$ *for some monotonic comparator networks* X_1, X_2 *and the* (ε, m)-*block* B. *There are constants* α, β, $\alpha \in \mathbb{N}$, $\beta \in \mathbb{Q}$, *such that if initially* B' *stores* 0*'s and* 1*'s and is at most* (p, m)-*dirty, then after* $\alpha \cdot p$ *iterations,* B' *is at most* $(\beta \cdot \log(p \cdot m), m)$-*dirty.*

Obviously, it suffices to prove Main Lemma for the case when $l = p$. Indeed, if we remove an even number of the leftmost columns filled with 0's or a number of rightmost columns filled with 1's, then the computation inside the remaining columns will run exactly in the same way as before. By the next two observations, it suffices to prove a more restricted version of Main Lemma.

Fact 4.3 *If Main Lemma holds for the case when the last column of* B *consists of* m *registers, then it holds in general.*

Proof. Replace B by an (ε, m)-block C in which there are additional registers in the last column so that the last column contains exactly m registers. Put 1's into the additional registers and use Main Lemma for the input obtained. Computation on C is exactly the same as on B if we restrict ourselves to the original registers. □

Fact 4.4 *If Main Lemma holds for the case when the number of ones equals $m \cdot r$ for an odd r, then it holds in general.*

Proof. If $\vec{x_1}, \vec{x_2} \in \{0,1\}^n$, then we write $\vec{x_1} \subseteq \vec{x_2}$ if $\vec{x_2}$ contains a 1 at each position where $\vec{x_1}$ contains a 1.

Claim. [4] If $\vec{x_1} \subseteq \vec{x_2}$ and F is a comparator network, then $F(\vec{x_1}) \subseteq F(\vec{x_2})$.

We leave an easy proof of the claim to the reader.

Let $\vec{x}$ be an arbitrary input to B'; let $\vec{x}$ contain s ones. Consider $\vec{x_1}, \vec{x_2}$ such that: $\vec{x_1} \subseteq \vec{x} \subseteq \vec{x_2}$; for some odd r, $\vec{x_1}$ contains $r \cdot m$ ones and $\vec{x_2}$ contains $(r+2) \cdot m$ ones (if $s < m$, then let $\vec{x_1}$ contain only zeroes, if $(r+2)m > n \geq s > r \cdot m$ for some odd number r, then let $\vec{x_2}$ contain only ones). By the claim, $(B')^{\alpha l}(\vec{x_1}) \subseteq (B')^{\alpha l}(\vec{x}) \subseteq (B')^{\alpha l}(\vec{x_2})$. Let K_{d_i} be the last column of 0's and K_{u_i} be the first column of 1's in $(B')^{\alpha l}(\vec{x_i})$, for $i = 1, 2$. By the assumption, Main Lemma applies to $\vec{x_1}, \vec{x_2}$, so $u_i - d_i \leq \beta \log(lm) + 1$, for $i = 1, 2$. The number of 1's in $\vec{x_1}$ and $\vec{x_2}$ differ by (at most) $2m$, so $u_2 \geq d_1 - 1$. It follows that $u_1 - d_2 \leq 2\beta \log(lm) + 3$. Since $(B')^{\alpha l}(\vec{x_1}) \subseteq (B')^{\alpha l}(\vec{x})$, every column of ones in $(B')^{\alpha l}(\vec{x_1})$ remains a column of ones in $(B')^{\alpha l}(\vec{x})$. Hence the columns K_{u_1} through K_l of $(B')^{\alpha l}(\vec{x})$ contain only 1's. Similarly we show that the columns K_1 through K_{d_2} of $(B')^{\alpha l}(\vec{x})$ contain only 0's. It follows that $(B')^{\alpha l}(\vec{x})$ is at most $(u_1 - d_2 - 1, m)$-dirty, hence also at most $(2\beta \log(lm) + 2, m)$-dirty. $\square$

4.2 Dominance relation

We consider the number of 1's in each column K_i during the computation on inputs consisting of 0's and 1's. This motivates the definitions that follow.

Definition 4.5 *Let $\vec{a} = (a_1, \ldots, a_l), \vec{b} = (b_1, \ldots, b_l)$, where $a_i, b_i \in [0, m]$ ($a_i, b_i \in \mathbb{Q}$), for $i \leq l$. For $i \leq l$, let* $\mathrm{head}_i(\vec{a}) = \sum_{j=1}^{i} a_j$. *We say that $\vec{a}$* dominates *$\vec{b}$ (denoted by $\vec{a} \succeq \vec{b}$), if* $\mathrm{head}_l(\vec{a}) = \mathrm{head}_l(\vec{b})$ *and* $\mathrm{head}_k(\vec{a}) \leq \mathrm{head}_k(\vec{b})$ *for every $k \leq l$.*

Intuitively, a_i, b_i denote the number of ones in K_i. If $\vec{a}$ dominates $\vec{b}$, then the sequence of 0's and 1's corresponding to $\vec{a}$ is closer to the sorted sequence than the sequence corresponding to $\vec{b}$. Equivalently, we could define the relation $\succeq$ as follows: $\vec{a} \succeq \vec{b}$ if $\vec{a}$ and $\vec{b}$ contain the same number of 1's, and for every i, the ith one in $\vec{b}$ occurs no later than the ith one in $\vec{a}$ (cf. [4]). The following properties follow directly from the definition:

Fact 4.6 *(a) The relation $\succeq$ is transitive.*
(b) Let X be an arbitrary monotonic network on the registers $R_1, \ldots, R_n$. Consider an input consisting of 0's and 1's; let e_i denote the number of 1's in K_i and $\vec{e} = (e_1, \ldots, e_l)$. Let x_i denote the number of 1's in K_i after applying X to the input and $\vec{x} = (x_1, \ldots, x_l)$. Then $\vec{x} \succeq \vec{e}$.

Definition 4.7 *Let B' be the network defined in Main Lemma. Let the input of the network B' consist of 0's and 1's. Then $a_{2t,i}$ denotes the number of ones in the column K_i of (ε, m)-block B immediately after iteration t of B', and $a_{2t+1,i}$ denotes the number of ones in K_i after applying the first layer of B at iteration $t+1$ of B'. Let $\vec{a_t} = (a_{t,1}, \ldots, a_{t,l})$.*

The vectors $\vec{a_t}$ might be very difficult to determine. Even if we fix ε-halvers used to build B, there are many different distributions of 0's and 1's corresponding to the same vector $\vec{a_0}$, and each of them influences $\vec{a_t}$ in some way. Therefore, we look for simpler vectors $\vec{b_t}$ such that $\vec{a_t}$ dominates $\vec{b_t}$ for each t. The key point is to define $\vec{b}_{t+1}$ from $\vec{b_t}$ without loosing the property that $\vec{a}_{t+1}$ dominates $\vec{b}_{t+1}$. For this reason we define *exchange relations* which formalize the relationship between $\vec{a_t}$ and $\vec{a}_{t+1}$.

Definition 4.8 *$W \subseteq [0, 2m] \times [0, m] \times [0, m]$ is an* exchange relation, *if $c_1 + c_2 = c$ for every $(c, c_1, c_2) \in W$. If W_1, W_2 are exchange relations, then we say that $W_1 \prec W_2$ if and only if*

$$\forall (c, c_1, c_2) \in W_1 \;\; \forall (c, d_1, d_2) \in W_2 \quad c_2 \leq d_2.$$

Intuitively, an exchange relation W describes the flow of ones while applying an ε-halver to neighboring columns of B: If there are c ones in $K_i \cup K_{i+1}$, and after applying an ε-halver to K_i and K_{i+1} there are c_i ones in K_i and c_{i+1} ones in K_{i+1}, then we require that $(c, c_i, c_{i+1}) \in W$. Note that for a given c, there might be many different triples $(c, c_1, c_2) \in W$. $W_1 \prec W_2$ means that exchange relation W_1 cannot cause more ones to be shifted than W_2 does.

Definition 4.9 *Let $\vec{e} = (e_1, \ldots, e_l), \vec{g} = (g_1, \ldots, g_l)$, $e_i, g_i \in [0, m]$ for $i \leq l$, and W be an exchange relation. We say that $(\vec{e}, \vec{g})$* agrees with W on odd levels *if*

$$\forall i < l, i \text{ odd} \quad (e_i + e_{i+1}, g_i, g_{i+1}) \in W$$

and $e_l = g_l$ if l is odd. We say that $(\vec{e}, \vec{g})$ agrees with W on even levels *if $e_1 = g_1$ and*

$$\forall i < l, i \text{ even} \quad (e_i + e_{i+1}, g_i, g_{i+1}) \in W$$

and $e_l = g_l$ if l is even.
We say that a sequence $\{\vec{e_t}\}_{t \leq T}$, where $\vec{e_t} \in [0, m]^l$, agrees with exchange relation W, *if for each even (odd) $t < T$, the pair $(\vec{e_t}, \vec{e}_{t+1})$ agrees with W on even (odd) levels.*

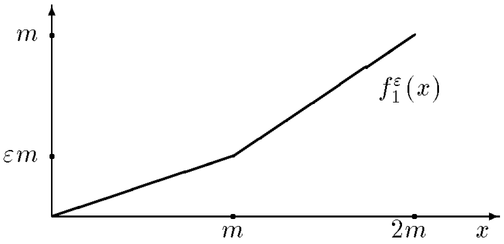

Figure 1: The function f_1^ε

Lemma 4.10 *Suppose that f_1, f_2 are nondecreasing functions from $[0, 2m]$ to $[0, m]$ such that $f_1(x) + f_2(x) = x$ for every x. Let $V = \{(x, f_1(x), f_2(x)) : x \in [0, 2m]\}$ and W be an exchange relation such that $V \prec W$. Assume that $(\vec{e}, \vec{e'})$ agrees with W on even (odd) levels, $(\vec{g}, \vec{g'})$ agrees with V on even (odd) levels, and $\vec{e} \succeq \vec{g}$. Then $\vec{e'} \succeq \vec{g'}$.*

Proof. We have to show that $\mathrm{head}_h(\vec{e'}) \leq \mathrm{head}_h(\vec{g'})$ for every h. If i is even, then $e_i + e_{i+1} = e'_i + e'_{i+1}$ by the assumption that $(\vec{e}, \vec{e'})$ agrees with W on even levels. Hence $\mathrm{head}_h(\vec{e}) = \mathrm{head}_h(\vec{e'})$ for each odd h. Similarly, $\mathrm{head}_h(\vec{g}) = \mathrm{head}_h(\vec{g'})$ for each odd h. Since $\mathrm{head}_h(\vec{e}) \leq \mathrm{head}_h(\vec{g})$, we get $\mathrm{head}_h(\vec{e'}) \leq \mathrm{head}_h(\vec{g'})$ for each odd h. Now we assume that h is even.

Case 1. $e_h + e_{h+1} \geq g_h + g_{h+1}$.

Then $\mathrm{head}_h(\vec{e'}) = \mathrm{head}_{h+1}(\vec{e'}) - e'_{h+1} \leq \mathrm{head}_{h+1}(\vec{e'}) - f_2(e_h + e_{h+1}) \leq \mathrm{head}_{h+1}(\vec{e}) - f_2(g_h + g_{h+1}) \leq \mathrm{head}_{h+1}(\vec{g}) - f_2(g_h + g_{h+1}) = \mathrm{head}_{h+1}(\vec{g'}) - g'_{h+1} = \mathrm{head}_h(\vec{g'})$.

Case 2. $e_h + e_{h+1} \leq g_h + g_{h+1}$.

Then $\mathrm{head}_h(\vec{e'}) = \mathrm{head}_{h-1}(\vec{e'}) + e'_h \leq \mathrm{head}_{h-1}(\vec{e'}) + f_1(e_h + e_{h+1}) \leq \mathrm{head}_{h-1}(\vec{e}) + f_1(g_h + g_{h+1}) \leq \mathrm{head}_{h-1}(\vec{g}) + f_1(g_h + g_{h+1}) = \mathrm{head}_h(\vec{g'})$.

For the case when $(\vec{e}, \vec{e'})$ and $(\vec{g}, \vec{g'})$ agree with W and V on odd levels, the proof is analogous. □

Definition 4.11 *Let $\varepsilon \in [0, \frac{1}{2})$. We define*

$$f_1^\varepsilon(x) = \begin{cases} \varepsilon x & \text{for } x \leq m, \\ m - (1-\varepsilon)(2m - x) & \text{for } x > m, \end{cases}$$

$$f_2^\varepsilon(x) = \begin{cases} (1-\varepsilon)x & \text{for } x \leq m, \\ m - \varepsilon(2m - x) & \text{for } x > m, \end{cases}$$

(see Fig. 1).

The next lemma follows immediately from properties of ε-halvers (Definition 3.1) and the definition of f_1^ε and f_2^ε.

Lemma 4.12 *If initially K_i and K_{i+1} contain together x ones and $2m - x$ zeroes, then after a single run of a $(K_i, K_{i+1}, \varepsilon)$-halver there are at most $f_1^\varepsilon(x)$ ones in K_i and at least $f_2^\varepsilon(x)$ ones in K_{i+1}.*

Definition 4.13 *Let $V^\varepsilon = \{(x, f_1^\varepsilon(x), f_2^\varepsilon(x)) : x \leq 2m\}$, and $W^\varepsilon = \{(x, x_1, x_2) : x \leq 2m,\ x_1 \leq f_1^\varepsilon(x),\ x_2 = x - x_1\}$.*

V^ε and W^ε are exchange relations that describe flow of ones between columns of an (ε, m)-block. V^ε describes the "minimal flow" guaranteed by ε-halvers. W^ε contains all triples describing real exchanges that occur during the computation. By definition of V^ε and W^ε, we get easily the following lemma.

Lemma 4.14 *$V^\varepsilon \prec W^\varepsilon$ and $V^\varepsilon \prec V^\varepsilon$.*

The functions f_1^ε, f_2^ε are nondecreasing. Hence by Fact 4.6, Lemma 4.10 and 4.14, we get the following corollary.

Corollary 4.15 *Let $V^\varepsilon \prec W$ (for instance, we could take $W = W^\varepsilon$ or $W = V^\varepsilon$). Let $\{\vec{e}_t\}_{t\geq 0}$ be a sequence such that for every $t \geq 0$ there are vectors $\vec{d}_{2t}, \vec{d}_{2t+1}$ such that $\vec{e}_{2t} \preceq \vec{d}_{2t}$, $(\vec{d}_{2t}, \vec{e}_{2t+1})$ agrees with W on odd levels, $(\vec{e}_{2t+1}, \vec{d}_{2t+1})$ agrees with W on even levels and $\vec{d}_{2t+1} \preceq \vec{e}_{2t+2}$. Assume that $\{\vec{g}_t\}_{t\geq 0}$ agrees with V^ε and $\vec{g}_0 \preceq \vec{e}_0$. Then $\vec{g}_t \preceq \vec{e}_t$ for every t.*

For a given $\vec{g}_0$, there is exactly one sequence $\{\vec{g}_t\}_{t\geq 0}$ that agrees with V^ε. Moreover, $\{\vec{g}_t\}_{t\geq 0}$ can be calculated without detailed knowledge about the structure of an (ε, m)-block. In order to analyze vectors $\{\vec{a}_t\}_{t\geq 0}$, we shall consider a simpler sequence $\{\vec{b}_t\}_{t\geq 0}$ such that $\vec{b}_0 \preceq \vec{a}_0$, and $\{\vec{b}_t\}_{t\geq 0}$ agrees with V_ε. Then, by Corollary 4.15, $\vec{b}_t \preceq \vec{a}_t$ for each t. Below, we formally define such a vector $\vec{b}_0$.

Definition 4.16 *Let $m|s$. Then $\vec{b} = (b_1, \ldots, b_l)$ is the* worst vector for s ones, *if $b_i = m$ for $i \leq s/m$, and $b_i = 0$, otherwise.*

In other words, the worst vector for s ones corresponds to the input in which all s ones are grouped in the columns as far from their final destination as possible. Clearly, if $\vec{a}_0$ represents a sequence that contains s ones and $\vec{b}_0$ is the worst vector for s ones, then $\vec{b}_0 \preceq \vec{a}_0$.

Before we proceed any further with the proof of Main Lemma, we show it for $m = 1$. In fact, this special case is a slight generalization of the well known result on Odd-Even Transposition Sort and has been already proved in [4].

Lemma 4.17 *Suppose that C is a comparator network on n registers of the form $X|F$, where F is a $(0,1)$-block and X is monotonic. Assume that the contents of C is at most $(s,1)$-dirty. Then after at most $\lceil s/2 \rceil$ iterations the contents of C is sorted.*

Proof. It suffices to prove the lemma for $s = n$. Let $\vec{g}$ be an input sequence. Let $\vec{b}$ be the worst vector for the same number of 1's as contained by the sequence represented by $\vec{g}$.

Note that $\vec{e} \preceq \vec{d}$ implies $F(\vec{e}) \preceq F(\vec{d})$ for each $\vec{e}$ and $\vec{d}$. Indeed, by definition, there are $\vec{e'}$ and $\vec{d'}$ such that $(\vec{e}, \vec{e'})$, $(\vec{d}, \vec{d'})$ agree with V^0 on odd levels, and $(\vec{e'}, F(\vec{e}))$, $(\vec{d'}, F(\vec{d}))$ agree with V^0 on even levels. Then by Lemma 4.10, $F(\vec{e}) \preceq F(\vec{d})$. Using the mentioned property and Fact 4.6, by a straightforward induction we may prove that $F^i(\vec{b}) \preceq C^i(\vec{g})$. On the other hand, it may be easily checked by examining the computation that the sequence $F^{\lceil s/2 \rceil}(\vec{b})$ is sorted. Now the lemma follows from the property that for every $\vec{e}, \vec{d} \in \{0,1\}^l$, if $\vec{e} \preceq \vec{d}$ and $\vec{e}$ is sorted, then $\vec{d}$ is sorted, too. □

4.3 Convex estimates

From now on, we consider only inputs to network B' that contain 0's and 1's, only, and contain exactly $r \cdot m$ ones for some fixed but arbitrary odd number r.

Definition 4.18 *Let $N^\varepsilon(\vec{x})$ ($P^\varepsilon(\vec{x})$) denote the vector $\vec{y}$ such that $(\vec{x}, \vec{y})$ agrees with V^ε on odd (even) levels.*

Definition 4.19 *For $i \in \mathbb{N}$, let $\vec{b_i^\varepsilon}$ be defined as follows: $\vec{b_0^\varepsilon} = \vec{b_0}$, $\vec{b_{i+1}^\varepsilon} = N^\varepsilon(\vec{b_i^\varepsilon})$, if $i+1$ is odd, and $\vec{b_{i+1}^\varepsilon} = P^\varepsilon(\vec{b_i^\varepsilon})$, if $i+1$ is even.*

Lemma 4.20 *$\vec{b_t^\varepsilon} \preceq \vec{a_t}$, for every t.*

Proof. The lemma follows from Corollary 4.15: we take $W = W^\varepsilon$, the vectors $\vec{d_{2t}}$ and $\vec{d_{2t+1}}$ correspond to the output of X_1 and the output of B, respectively, during iteration $t+1$. Since X_1 and X_2 are monotonic, $\vec{a_{2t}} \preceq \vec{d_{2t}}$ and $\vec{d_{2t+1}} \preceq \vec{a_{2t+2}}$, as required. □

For $\varepsilon = 0$, we get vectors $\vec{b_i^0}$ that are closely related to the sequences of 0's and 1's occurring during Odd-Even Transposition Sort. Namely, let $\tilde{b}_i^0$ be the sequence obtained from $\vec{b_i^0}$ by replacing every m by 1. Then $\tilde{b}_i^0$ is the sequence obtained after i steps of Odd-Even Transposition Sort, when started with $\tilde{b}_0$. So we can reformulate well known properties of Odd-Even Transposition Sort in the following way.

Lemma 4.21
For $t \geq l-1$, $\vec{b_t^0} = (0, \ldots, 0, m, \ldots, m)$. For $t < l-1$, $\vec{b_t^0}$ is a sequence containing only 0's in positions 1 through $(l-r)-(l-1-t)$ and containing only m's in positions $(l-r+1)+(l-1-t)$ through l.

Proof. The lemma follows from the fact that Odd-Even Transposition Sort sorts $\tilde{b}_0^0$ in exactly $l-1$ steps and that at time step t every element must be at a distance not larger than $l-1-t$ from its final destination. □

Recall that for $\alpha_1, \ldots, \alpha_k \geq 0$ such that $\sum_{i=1}^k \alpha_i = 1$, the vector $\alpha_1 \vec{e_1} + \cdots + \alpha_k \vec{e_k}$ is a *convex combination of* $\vec{e_1}, \ldots, \vec{e_k}$. First, we prove the following property of the operators N^ε and P^ε on convex combinations:

Lemma 4.22 *Let $\alpha_1, \ldots, \alpha_k \geq 0$, $\sum_{i=1}^k \alpha_i = 1$ and $\vec{e_1}, \ldots, \vec{e_k} \in [0,m]^l$. Then*

$$\sum_{i=1}^{k} \alpha_i N^\varepsilon(\vec{e_i}) \preceq N^\varepsilon(\sum_{i=1}^{k} \alpha_i \vec{e_i})$$

and

$$\sum_{i=1}^{k} \alpha_i P^\varepsilon(\vec{e_i}) \preceq P^\varepsilon(\sum_{i=1}^{k} \alpha_i \vec{e_i})$$

Proof. We prove the lemma for $k = 2$. Generalizing this for any k is standard.
Claim. *Let $\alpha, \beta \geq 0, \alpha + \beta = 1$, and $x, y \in [0, 2m]$. Then $f_1^\varepsilon(\alpha x + \beta y) \leq \alpha f_1^\varepsilon(x) + \beta f_1^\varepsilon(y)$.*

Since verification of the claim is elementary and based on the definition of the function f_1^ε, we skip it.

For any exchange relation W and $(\vec{z_1}, \vec{z_2})$ that agrees with W on odd levels, $\text{head}_h(\vec{z_1}) = \text{head}_h(\vec{z_2})$ for every even h. Also, $\text{head}_h N^\varepsilon(\alpha\vec{z}) = \text{head}_h \alpha N^\varepsilon(\vec{z})$ for every $\vec{z}$, α and even h. It follows easily that for every even h

$$\text{head}_h(\alpha N^\varepsilon(\vec{x})) + \text{head}_h(\beta N^\varepsilon(\vec{y})) = \\ \text{head}_h(N^\varepsilon(\alpha\vec{x} + \beta\vec{x})).$$

If h is odd, then

$$\begin{aligned} &\alpha \text{head}_h(N^\varepsilon(\vec{x})) + \beta \text{head}_h(N^\varepsilon(\vec{y})) = \\ &\quad \alpha\left(f_1^\varepsilon(x_h + x_{h+1}) + \text{head}_{h-1}(N^\varepsilon(\vec{x}))\right) + \\ &\quad \beta\left(f_1^\varepsilon(y_h + y_{h+1}) + \text{head}_{h-1}(N^\varepsilon(\vec{y}))\right) \geq \\ &\quad f_1^\varepsilon(\alpha(x_h + x_{h+1}) + \beta(y_h + y_{h+1})) + \\ &\quad \alpha \text{head}_{h-1}(\vec{x}) + \beta\, \text{head}_{h-1}(\vec{y}) = \\ &\quad \text{head}_h(\alpha\vec{x} + \beta\vec{y}). \end{aligned}$$

For P^ε the proof is analogous. □

Lemma 4.23 *Let $0 \leq i \leq l-2$. If i is even, then*

$$N^\varepsilon(\vec{b_i^0}) = N^\varepsilon(\vec{b_{i+1}^0}) = \varepsilon \vec{b_i^0} + (1-\varepsilon)\vec{b_{i+1}^0}.$$

If i is odd, then

$$P^\varepsilon(\vec{b_i^0}) = P^\varepsilon(\vec{b_{i+1}^0}) = \varepsilon \vec{b_i^0} + (1-\varepsilon)\vec{b_{i+1}^0}.$$

Moreover, $P^\varepsilon(\vec{b_0^0}) = \vec{b_0^0}$; $P^\varepsilon(\vec{b_{l-1}^0}) = \vec{b_{l-1}^0}$, if l is even; $N^\varepsilon(\vec{b_{l-1}^0}) = \vec{b_{l-1}^0}$, if l is odd.

The proof of Lemma 4.23 is easy and we leave it to the reader.

We may apply Lemma 4.23 as follows. Assume that $\vec{b} \succeq \alpha_0 \vec{b_0^0} + \cdots + \alpha_{l-1}\vec{b_{l-1}^0}$. Then we get:

$$\begin{aligned} N^\varepsilon(\vec{b}) &\succeq N^\varepsilon(\alpha_0\vec{b_0^0} + \cdots + \alpha_{l-1}\vec{b_{l-1}^0}) \succeq \\ &\alpha_0 N^\varepsilon(\vec{b_0^0}) + \cdots + \alpha_{l-1}N^\varepsilon(\vec{b_{l-1}^0}) = \\ &\varepsilon(\alpha_0+\alpha_1)\vec{b_0^0} + (1-\varepsilon)(\alpha_0+\alpha_1)\vec{b_1^0} + \\ &\varepsilon(\alpha_2+\alpha_3)\vec{b_2^0} + (1-\varepsilon)(\alpha_2+\alpha_3)\vec{b_3^0} + \cdots. \end{aligned}$$

Similarly,

$$\begin{aligned} P^\varepsilon(\vec{b}) \succeq \alpha_0\vec{b_0^0} &+ \varepsilon(\alpha_1+\alpha_2)\vec{b_1^0} + (1-\varepsilon)(\alpha_1+\alpha_2)\vec{b_2^0} \\ &+\varepsilon(\alpha_3+\alpha_4)\vec{b_3^0} + (1-\varepsilon)(\alpha_3+\alpha_4)\vec{b_4^0} + \cdots. \end{aligned}$$

Definition 4.24
Convex estimate $\alpha_0^t\vec{b_0^0} + \cdots + \alpha_{l-1}^t\vec{b_{l-1}^0}$ *of* $\vec{b_t^\varepsilon}$ *is defined as follows:*

- *for* $t=0$, *the convex estimate equals* $\vec{b_0^0}$ *(i.e.* $\alpha_0^0 = 1$ *and* $\alpha_i^0 = 0$ *for* $i \neq 0$*),*
- *if* t *is even, then the convex estimate of* $\vec{b_{t+1}^\varepsilon}$ *equals* $\alpha_0^t N^\varepsilon(\vec{b_0^0}) + \cdots + \alpha_{l-1}^t N^\varepsilon(\vec{b_{l-1}^0})$, *that is,*

$$\alpha_i^{t+1} = \begin{cases} \varepsilon(\alpha_i^t + \alpha_{i+1}^t) & \text{if } i \text{ is even, } 0 \le i < l-1, \\ (1-\varepsilon)(\alpha_{i-1}^t + \alpha_i^t) & \text{if } i \text{ is odd, } 0 \le i \le l-1, \\ \alpha_{l-1}^t & \text{if } i = l-1,\ l-1 \text{ is even,} \end{cases}$$

- *if* t *is odd, then the convex estimate of* $\vec{b_{t+1}^\varepsilon}$ *equals* $\alpha_0^t P^\varepsilon(\vec{b_0^0}) + \cdots + \alpha_{l-1}^t P^\varepsilon(\vec{b_{l-1}^0})$, *that is,*

$$\alpha_i^{t+1} = \begin{cases} \alpha_0^t & \text{if } i = 0, \\ \varepsilon(\alpha_i^t + \alpha_{i+1}^t) & \text{if } i \text{ is odd, } 0 \le i < l-1, \\ (1-\varepsilon)(\alpha_{i-1}^t + \alpha_i^t) & \text{if } i \text{ is even, } 0 \le i \le l-1, \\ \alpha_{l-1}^t & \text{if } i = l-1,\ l-1 \text{ is odd.} \end{cases}$$

By Lemma 4.22, for every t, $\vec{b_t^\varepsilon}$ dominates its convex estimate. The coefficients α_j^t are relatively easy to compute. Moreover, one can see that the coefficients of $\vec{b_i^0}$, for a small i, quite fast begin to decrease exponentially when t grows. If they become very small, then we shall see that $\vec{b_t^\varepsilon}$ is "almost sorted".

4.4 Flow of coefficients

It would be tiresome to estimate the coefficients α_i^t directly. Instead, we use a method based on the concept of *flow of coefficients*:

Definition 4.25 *For* $i, t \in \mathbb{N}$, $i \le l-1$, *let* $\text{flow}_t(i) = \alpha_i^t - \alpha_i^{t-1}$, *if* t *and* i *have the same parity, and* $\text{flow}_t(i) = 0$, *otherwise.*

Intuitively, $\text{flow}_t(i)$ describes increase of α_i at expense of α_{i-1} at step t. Indeed, if t and i have the same parity, then $\alpha_i^t = (1-\varepsilon)\cdot(\alpha_i^{t-1} + \alpha_{i-1}^{t-1})$, $\alpha_{i-1}^t = \varepsilon\cdot(\alpha_i^{t-1} + \alpha_{i-1}^{t-1})$, hence $\alpha_i^t + \alpha_{i-1}^t = \alpha_i^{t-1} + \alpha_{i-1}^{t-1}$. So $\alpha_{i-1}^t = \alpha_{i-1}^{t-1} - \text{flow}_t(i)$.

Lemma 4.26 *Let* $i < l$ *and let* $t+1$ *and* i *have the same parity. Then for* $1 < i < l-1$,

$$\text{flow}_{t+1}(i) = (1-\varepsilon)\cdot\text{flow}_t(i-1) + \varepsilon\cdot\text{flow}_t(i+1).$$

For $i = 1$,

$$\text{flow}_{t+1}(1) = \varepsilon\cdot\text{flow}_t(2).$$

For $i = l-1$,

$$\text{flow}_{t+1}(l-1) = (1-\varepsilon)\cdot\text{flow}_t(l-2).$$

Proof. By the construction of convex estimates, $\alpha_i^{t-1}/\alpha_{i-1}^{t-1} = (1-\varepsilon)/\varepsilon$. Hence

$$(1-\varepsilon)\alpha_{i-1}^{t-1} = \varepsilon\alpha_i^{t-1}. \tag{1}$$

Let $\text{flow}_t(i-1) = x$ and $\text{flow}_t(i+1) = y$. So $\alpha_{i-1}^t = \alpha_{i-1}^{t-1} + x$ and $\alpha_i^t = \alpha_i^{t-1} - y$. Hence, $\alpha_i^{t+1} = (1-\varepsilon)(\alpha_{i-1}^{t-1} + x + \alpha_i^{t-1} - y) = \alpha_i^{t-1} + (1-\varepsilon)x - (1-\varepsilon)y$ (the second equality follows from (1)). So $\text{flow}_{t+1}(i) = \alpha_i^{t+1} - (\alpha_i^{t-1} - y) = (1-\varepsilon)x + \varepsilon y$.

The formulas for $\text{flow}_{t+1}(1)$ and $\text{flow}_{t+1}(l-1)$ can be obtained in a similar way. □

Lemma 4.27 $\text{flow}_t(i) \ge 0$ *for every* t, i.

Proof. The proof is by induction on t. Since $\alpha_0^0 = 1$, $\alpha_1^0 = 0$, $\alpha_0^1 = \varepsilon$, and $\alpha_1^1 = 1-\varepsilon$, we get $\text{flow}_1(1) = 1-\varepsilon$. Obviously, $\text{flow}_1(i) = 0$ for $i \neq 1$. Hence the lemma holds for $t = 1$.

Now let us assume that the lemma holds for t. If i and $t+1$ have the same parity, then $\text{flow}_{t+1}(i) \ge 0$ by Lemma 4.26 and the induction hypothesis. If i and $t+1$ have different parities, then $\text{flow}_{t+1}(i) = 0$ by the definition. □

To get an upper bound on flows we define $\text{upflow}_t(i)$ for $i \in \mathbb{Z}$. Namely, for $t = 1$ we put $\text{upflow}_1(1) = 1$, and $\text{upflow}_1(i) = 0$ for $i \neq 1$. (We

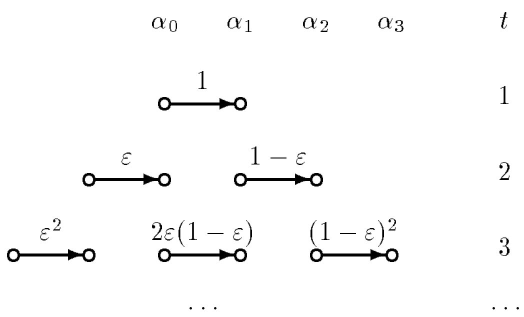

Figure 2: Upper bounds for the flows

put $\mathrm{upflow}_1(1) = 1$ instead of $\mathrm{upflow}_1(1) = 1-\varepsilon$ in order to simplify some expressions.) To define upflow_{t+1} from upflow_t, we use a formula based on the formula for flows given in Lemma 4.26:

$$\mathrm{upflow}_{t+1}(i) = (1-\varepsilon)\cdot \mathrm{upflow}_t(i-1) + \varepsilon \cdot \mathrm{upflow}_t(i+1),$$

if $t+1$ and i have the same parity, and $\mathrm{upflow}_{t+1}(i) = 0$ otherwise. It easily follows from the definition that $\mathrm{upflow}_t(i) \geq 0$ for every t, i, and $\sum_{i\in\mathbb{Z}} \mathrm{upflow}_t(i) = 1$.

Lemma 4.28 *For every $t, i \in \mathbb{N}$, $i < l-1$,*

$$\mathrm{flow}_t(i) \leq \mathrm{upflow}_t(i).$$

Proof. The proof is by induction on t. For $t = 1$ the lemma is obviously true. Now let us assume that the lemma holds for t. Let $t+1$ and i have the same parity. For $i = 1$, we have

$$\begin{aligned}\mathrm{flow}_{t+1}(1) &= \varepsilon\cdot\mathrm{flow}_t(2) \leq \varepsilon\cdot \mathrm{upflow}_t(2) \leq \\ &(1-\varepsilon)\cdot \mathrm{upflow}_t(0) + \varepsilon\cdot\mathrm{upflow}_t(2) = \\ &\mathrm{upflow}_{t+1}(1).\end{aligned}$$

The cases of $i = l-1$ and $1 < i < l-1$ can be checked similarly. □

Lemma 4.29 *If t and i have the same parity, $-t+2 \leq i \leq t$, then*

$$\mathrm{upflow}_t(i) = \binom{t-1}{\frac{t+i}{2}-1}\varepsilon^{\frac{t-i}{2}}(1-\varepsilon)^{\frac{t+i}{2}-1}. \qquad (2)$$

Lemma 4.29 follows directly from the formulas defining upflows (see also Fig. 2).

For the rest of the proof we fix constants $d, c_0 \in \mathbb{Q}$ such that

- $d > \frac{1}{1-2\varepsilon}$ and dl is an even natural number,
- $c_0 = \frac{(d+1)\varepsilon}{(d-1)(1-\varepsilon)}$.

Corollary 4.30 *If $0 < i < l$ and i is even, then*

$$\frac{\mathrm{upflow}_{dl}(i)}{\mathrm{upflow}_{dl}(i+2)} \leq c_0 < 1. \qquad (3)$$

Proof. The first inequality may be easily obtained using equality (2). The second one follows easily from the assumption that $d > \frac{1}{1-2\varepsilon}$. We leave the details to the reader. □

Corollary 4.31 *If i is even, $0 < i < l$, then $\mathrm{upflow}_{dl}(i) \leq c_0^{\lceil (l-i)/2\rceil}$. Hence, $\mathrm{flow}_{dl}(i) \leq c_0^{\lceil (l-i)/2\rceil}$.*

Proof. By inequality (3), we get $\mathrm{upflow}_{dl}(i) \leq c_0^j \cdot \mathrm{upflow}_{dl}(i+2j)$, for $i+2j \leq l+1$. Let $s = \lceil (l-i)/2 \rceil$. Then $i+2s \leq l+1$. The sum of all upflows is 1, so $\mathrm{upflow}_{dl}(i+2s) \leq 1$. Hence $\mathrm{upflow}_{dl}(i) \leq c_0^s$. The second part of the lemma follows from Lemma 4.28. □

Lemma 4.32 *There is a constant c_1 such that for every odd i, $0 < i < l$,*

$$\alpha_{i-1}^{dl-2} + \alpha_i^{dl-2} < c_1 \cdot c_0^{(l-i)/2}.$$

Proof. Let $i < l$ be odd. We define $x_j = \alpha_{j-1}^{dl-2} + \alpha_j^{dl-2}$, for $j < l$. Then $\alpha_{i-1}^{dl-1} = \varepsilon x_i$ and $\alpha_i^{dl-1} = (1-\varepsilon)x_i$. For $j < l$, let y_j denote $\mathrm{flow}_{dl}(j)$. Then $\alpha_i^{dl} = \alpha_i^{dl-1} - y_{i+1}$ and $\alpha_{i+1}^{dl} = \alpha_{i+1}^{dl-1} + y_{i+1}$. On the other hand, $\alpha_i^{dl}/\alpha_{i+1}^{dl} = \varepsilon/(1-\varepsilon)$. Applying all these equalities we get

$$\frac{(1-\varepsilon)x_i - y_{i+1}}{\varepsilon x_{i+2} + y_{i+1}} = \frac{\varepsilon}{1-\varepsilon},$$

or equivalently,

$$x_i = \tfrac{\varepsilon^2}{(1-\varepsilon)^2}\cdot x_{i+2} + \tfrac{1}{(1-\varepsilon)^2}\cdot y_{i+1}. \qquad (4)$$

(Note that $\frac{\varepsilon^2}{(1-\varepsilon)^2} < 1$, since $\varepsilon < \frac{1}{2}$.) By Corollary 4.31, $y_{i+1} \leq c_0^{(l-i-1)/2}$, so we get easily

$$x_i \leq R + \tfrac{1}{(1-\varepsilon)^2}\cdot \sum_{j=0}^{\lfloor (l-i)/2\rfloor} \left(\tfrac{\varepsilon}{1-\varepsilon}\right)^{2j} c_0^{(l-i-1)/2-j}$$

where $R = \left(\frac{\varepsilon}{1-\varepsilon}\right)^{2\lfloor (l-i)/2\rfloor}$. Hence

$$x_i \leq R + \tfrac{1}{(1-\varepsilon)^2}\cdot c_0^{(l-i-1)/2}\cdot \sum_{j=0}^{\infty}\left(\left(\tfrac{\varepsilon}{1-\varepsilon}\right)^2\cdot \tfrac{1}{c_0}\right)^j.$$

Note that $\left(\frac{\varepsilon}{1-\varepsilon}\right)^2\cdot\frac{1}{c_0} \leq \frac{\varepsilon}{1-\varepsilon} < 1$. Hence

$$\begin{aligned}x_i \leq\ & R + \tfrac{1}{(1-\varepsilon)^2}\cdot c_0^{(l-i-1)/2}\cdot \sum_{j=0}^{\infty}\left(\tfrac{\varepsilon}{1-\varepsilon}\right)^j = \\ & \left(\tfrac{\varepsilon}{1-\varepsilon}\right)^{2\lfloor (l-i)/2\rfloor} + \tfrac{1}{(1-\varepsilon)(1-2\varepsilon)}\cdot c_0^{(l-i-1)/2}. \qquad (5)\end{aligned}$$

Note that $\frac{\varepsilon}{1-\varepsilon} < c_0 < 1$, so $(\frac{\varepsilon}{1-\varepsilon})^2 < c_0$. Hence the last term in (5) dominates the first one and therefore $x_i \leq c_1 \cdot c_0^{(l-i)/2}$ for some constant c_1. □

We are now looking for an η such that $\sum_{i=0}^{l-\eta} \alpha_i^{dl-2} < \frac{1}{ml}$, i.e., up to $\alpha_{l-\eta}^{dl-2}$, the coefficients α_i^{dl-2} are very small (which means that the vectors $\vec{b_i^0}$ contribute very little to the convex estimation of $\vec{b_{dl-2}^\varepsilon}$).

Lemma 4.33 *There is a constant $c_2 > 0$ such that $\sum_{i=0}^{l-\eta} \alpha_i^{dl-2} < \frac{1}{ml}$ for $\eta = c_2 \cdot \log(m \cdot l)$.*

Proof. We consider η such that $l - \eta$ is odd. Then

$$\begin{aligned} \sum_{i=0}^{l-\eta} \alpha_i^{dl-2} &= \sum_{j=0}^{(l-\eta-1)/2} x_{2j+1}^{dl-2} \leq c_1 \cdot c_0^{\eta/2} \cdot \sum_{s=0}^{\infty} c_0^s \\ &= \frac{c_1}{1-c_0} \cdot c_0^{\eta/2}. \end{aligned}$$

For an $\eta = O(\log(ml))$ appropriately chosen (namely, for $\eta > \frac{2}{-\log c_0}(\log(ml) + \log \frac{c_1}{1-c_0})$), the last expression is smaller than $\frac{1}{ml}$. □

Proof of Main Lemma (Lemma 4.2). Recall that we consider an input to network B' containing rm ones and $n-rm$ zeroes, that $a_{2t,i}$ denotes the number of ones in K_i immediately after iteration t, $a_{2t+1,i}$ denotes the number of ones in K_i after applying the first layer of B at iteration $t+1$ and $\vec{a_j} = (a_{j,1}, \ldots, a_{j,l})$, that $\vec{b_0^\varepsilon}$ is the worst vector for $r \cdot m$ ones, that $\{\vec{b_t^\varepsilon}\}_{t \geq 0}$ is the sequence that agrees with V^ε, and that

$$\vec{a_t} \succeq \vec{b_t^\varepsilon} \succeq \sum_{i=0}^{l-1} \alpha_i^t \cdot \vec{b_i^0}.$$

It follows that for every $h \leq l$,

$$\text{head}_h(\vec{a_t}) \leq \text{head}_h(\sum_{i=0}^{l-1} \alpha_i^t \cdot \vec{b_i^0}).$$

Let $\eta = c_2 \cdot \log(m \cdot l)$ and $h = l - r - \eta - 1$. Then

$$\text{head}_h(\vec{a}_{dl-2}) \leq \sum_{i=0}^{l-1} \alpha_i^{dl-2} \cdot \text{head}_h(\vec{b_i^0}).$$

By Lemma 4.21, $\vec{b_i^0}$ contains only zeroes up to position $(l-r)-(l-1-i)$. For $i > l-\eta$, $(l-r)-(l-1-i) > h$, hence $\text{head}_h(\vec{b_i^0}) = 0$. So

$$\text{head}_h(\vec{a}_{dl-2}) \leq \sum_{i=0}^{l-\eta} \alpha_i^{dl-2} \cdot \text{head}_h(\vec{b_i^0}) \leq ml \cdot \sum_{i=0}^{l-\eta} \alpha_i^{dl-2}.$$

Hence by Lemma 4.33, $\text{head}_h(\vec{a}_{dl-2}) < 1$. Since $a_{t,i} \in \mathbb{N}$ for every t and i, it follows that $a_{dl-2,1}, \ldots, a_{dl-2,l-r-\eta-1} = 0$.

The proof that $a_{dl-2,h}, \ldots, a_{dl-2,l} = m$, for $h = (l - r + 1) + \eta + 1$, is similar. (If we change the directions of all comparators, we will obtain an (ε, m)-block that works in opposite direction. Ones in this (ε, m)-block behave exactly like zeroes in the original (ε, m)-block.) It follows that after iteration $(dl-2)/2$, the contents of the network is at most $(2\eta + 2, m)$-dirty. Since $\eta = O(\log(m \cdot l))$, Main Lemma follows. □

4.5 Analysis of the algorithm

We shall show that the network $I_{n,k}$ sorts in $\alpha k n^{1/k} + (c \log n)^k$ iterations, where the constants α, β are from Lemma 4.2, while $c = \beta + 1$ is a parameter from the definition of $I_{n,k}$. By 0-1 Principle, it suffices to consider inputs consisting of 0's and 1's. We divide the iterations into $k+1$ phases. Each of the first k phases consists of $\alpha n^{1/k}$ iterations; the last phase consists of the last $(c \log n)^k$ iterations. In order to analyze phase i, for $i \leq k$, we use the fact that $I_{n,k}$ is of the form $X_1 | F_i | X_2$, where X_1, X_2 are some monotonic networks. Let m_i denote the size of the columns of F_i, that is, $m_i = n^{(k-i)/k} \cdot (c \log n)^{i-1}$ and $m_{k+1} = 1$.

Claim 4.34 *For $i \leq k$, at the beginning of phase i, the network $I_{n,k}$ is at most $(n^{1/k}, m_i)$-dirty. At the beginning of phase $k+1$, the network $I_{n,k}$ is at most $((c \log n)^k, 1)$-dirty.*

Proof. The proof is by induction on i. Every input sequence is by definition at most $(n^{1/k}, n^{(k-1)/k})$-dirty. Hence the lemma holds for $i = 1$. By Main Lemma (Lemma 4.2), if the input of phase i is at most $(n^{1/k}, m_i)$-dirty, then the output of phase i is at most $(\beta \log(n^{1/k} \cdot m_i), m_i)$-dirty. Since $n^{1/k} \cdot m_i \leq n$, the output is at most $(\beta \log n, m_i)$-dirty, as well. This means that the number of registers from the first register containing a 1 to the last register containing a 0 is at most $m_i \cdot \beta \log n$. For $i < k$, these registers are contained in at most $\lceil \frac{m_i \cdot \beta \log n}{m_{i+1}} \rceil + 1 = \lceil n^{1/k} \frac{\beta}{\beta+1} \rceil + 1$ adjacent columns of (ε, m_{i+1})-block F_{i+1}. We may assume that n is sufficiently large so that $\lceil n^{1/k} \frac{\beta}{\beta+1} \rceil + 1 \leq n^{1/k}$. Hence the input of phase $i+1$ is at most $(n^{1/k}, m_{i+1})$-dirty. If $i = k$, then $m_i \cdot \beta \log n \leq (c \log n)^k$, hence the output of stage k is at most $((c \log n)^k, 1)$-dirty. □

The output of phase k is at most $((c \log n)^k, 1)$-dirty. $I_{n,k} = X | F_{k+1}$, where X is some monotonic network, so by Lemma 4.17, after $\lceil (c \log n)^k / 2 \rceil$ iterations at stage $k+1$ the sequence stored by $I_{n,k}$ is sorted.

The total time for sorting a sequence of n elements on $I_{n,k}$ is the depth of the network multiplied by the number of necessary iterations, that is, at most $(2\delta k+2)(k\alpha n^{1/k}+(c\log n)^k) = O(k^2\cdot n^{1/k})$, where $\delta = O(\frac{1}{\varepsilon}\cdot\log\frac{1}{\varepsilon})$ is the depth of an ε-halver. This completes the proof of Theorem 1.1. $\square$

Remark. If during phase i $(1\le i\le k+1)$ we use F_i instead of the whole $I_{n,k}$, then the total time will be $2\delta k\alpha n^{1/k}+(c\log n)^k$. In this case, we can also use pipelining for sorting $k+1$ sequences simultaneously.

Acknowledgment

We thank Friedhelm Meyer auf der Heide for presenting us the problem of periodic sorting networks of constant depth. Many ideas leading to the solution presented in the paper have been contributed by Friedhelm Meyer auf der Heide, Juraj Hromkovič, Krzysztof Loryś and Rolf Wanka.

References

[1] M. Ajtai, J. Komlós and E. Szemerédi. Sorting in $c\log n$ parallel steps. *Combinatorica* **3** (1983) 1-19.

[2] K. E. Batcher. Sorting networks and their applications. In *AFIPS Conf. Proc. 32*, pp. 307–314, 1968.

[3] M. Dowd, Y. Perl, M. Saks, and L. Rudolph. The periodic balanced sorting network. *J. ACM* **36** (1989) 738–757.

[4] J. G. Krammer. *Lösung von Datentransportproblemen in integrierten Schaltungen.* Ph.D. Dissertation, Technical University Munich, 1991.

[5] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes* (Morgan Kaufmann, San Mateo, 1992).

[6] F. Meyer auf der Heide. *Personal communication*, 1991.

[7] I. D. Scherson, S. Sen, and A. Shamir. Shearsort: A true two-dimensional sorting technique for VLSI networks. In *Proc. of the IEEE International Conference on Parallel Processing*, pp. 903–908, 1986.

[8] U. Schwiegelshohn. A shortperiodic two-dimensional systolic sorting algorithm. In *IEEE International Conference on Systolic Arrays*, pp. 257-264, 1988.