Fast Periodic Correction Networks

Grzegorz Stachowiak

Institute of Computer Science, University of Wrocław, Przesmyckiego 20, 51-151 Wrocław, Poland (gst@ii.uni.wroc.pl)

Abstract. We consider the problem of sorting N-element inputs differing from already sorted sequences on t entries. To perform this task we construct a comparator network that is applied periodically. The two constructions for this problem made by previous authors required $O(\log n + t)$ iterations of the network. Our construction requires $O(\log n + (\log\log N)^2(\log t)^3)$ iterations which makes it faster for $t \gg \log N$.

Keywords: sorting network, comparator, periodic sorting network.

1 Introduction

Sorting is one of the most fundamental problems of computer science. A classical approach to sort a sequence of keys is to apply a comparator network. Apart from a long tradition, comparator networks are particularly interesting due to hardware implementations. They can be also implemented as sorting algorithms for parallel computers.

In our approach sorted elements are stored in registers $r_1, r_2, \ldots, r_N$. Registers are indexed with integers or elements of other linearly ordered sets. A *comparator* $[i : j]$ is a simple device connecting registers r_i and $r_j (i < j)$. It compares the keys they contain and if the key in r_i is bigger, it swaps the keys. The general problem is the following. At the beginning of the computations the input sequence of keys is placed in the registers. Our task is to sort the sequence of keys according to the linear order of register indices by applying a sequence of comparators. The sequence of comparators is the same for all possible inputs. We assume that comparators connecting disjoint pairs of registers can work in parallel. Thus we arrange the sequence of comparators into a series of *layers* which are sets of comparators connecting disjoint pairs of registers. The total time needed by such a network to sort a sequence is proportional to the number of layers called the network's *depth*.

Much research concerning sorting networks was done in the past. Most famous results are asymptotically optimal AKS [1] sorting network of depth $O(\log N)$ and more 'practical' Batcher [2] network of depth $\sim \frac{1}{2}\log^2 N$ (from now on all the logarithms are binary).

Some research was devoted to problems concerning periodic sorting networks. Such a comparator network is applied not once but many times in a series of iterations. The input of the first iteration is the sequence to be sorted. The input of $(i+1)$st iteration is the output of ith iteration. The output of the last iteration should always be sorted. The total time needed to sort an input sequence is the product of the number of iterations and the depth of the network. Constructing such networks especially of small constant depth gives hope to reduce the amount of hardware needed to build sorting comparator networks. It can be done by applying the same small chip many times to sort an input. We can also view such a network as a building block of a sorting network in which layers are repeated periodically. Main results concerning periodic sorting networks are presented in the table:

	depth	# iterations
DPS [3]	$\log N$	$\log N$
Schwiegelsohn [15]	8	$O(\sqrt{N}\log N)$
KKS [5]	$O(k)$	$O(N^{1/k})$
Loryś et al. [9]	3-5	$O(\log^2 N)$

Last row of this table requires some words of explanation. The paper [9] describes a network of depth 5, but a later paper [10] reduces this value to 3. The number of iterations $O(\log^2 N)$ is achieved by periodification of AKS sorting network for which the constant hidden behind big O is very big. Periodification of Batcher network requires less iterations for practical sizes of the input, though it requires the time $O(\log^3 N)$ asymptotically. It is not difficult to show that 3 is the minimal depth of a periodic sorting network which requires $o(N)$ iterations to sort an arbitrary input.

A sequence obtained from a sorted one by t changes being either swaps between pairs of elements or changes on single positions we call *t-disturbed*. We define *t-correction* network to be a specialized network sorting t-disturbed inputs. Such networks were designed to obtain a sorted sequence from an output produced by a sorting network having t faulty comparators [14, 11, 16]. There are also other potential applications in which we have to deal with sequences that differ not much from a sorted one. Let us consider a large sorted database with N entries. In some period of time we make t modifications of the database and want to have it sorted back. It can be more effective to use a specialized correction unit in such a case, than to apply a sorting network. Results concerning such correction networks are presented in [4, 16].

There was some interest in constructing periodic comparator networks of a constant depth, that sort t-disturbed inputs. The reason is that the fastest known constant depth periodic sorting networks have running time $O(\log^2 N)$. On the other hand in some applications faster correction networks can replace sorting networks. Two periodic correction networks were already constructed by Kik and Piotrów [6, 12]. The first of them has depth 8 and the other has depth 6. Both of them require $O(\log N + t)$ iterations for considered inputs where N is input size and t is the number of modifications. The running time is $O(\log N)$ for $t = O(\log N)$ and the constants hidden behind the big O are small. Unfortunately it is not known how fast these networks complete sorting if $t \gg \log N$.

In this paper we construct a periodic t-correction network to deal with $t : \log N \ll t \ll N$. The reason we assume that t is small in comparison to N is the following. If t is about the same as N, then the periodification scheme gives a practical periodic sorting network of depth 3 requiring $O(\log^3 N) = O(\log^3 t)$ iterations. Actually we do not hope to get better performance in such a case. Our network has depth 3 and running time: $O(\log N + (\log\log N)^2(\log t)^3)$. We should mention that in our construction we do not use AKS sorting network. If this network was used (also in the auxiliary construction of a non periodic t-correction network) we would get the running time: $O(\log N + (\log\log N)(\log t)^2)$. In such case the AKS constant would stand in front of $(\log\log N)(\log t)^2$.

Now we remind of a couple of useful properties of comparator networks. The first of them is a general property of all comparator networks. Let us assume we have two inputs for a fixed comparator network. We say that we have relation $(x_1, x_2, \ldots, x_N) \le (y_1, y_2, \ldots, y_N)$ between these inputs if for all i we have $x_i \le y_i$.

Lemma 1.1. *If we apply the same comparator network to inputs for which we have $(x_1, x_2, \ldots, x_N) \le (y_1, y_2, \ldots, y_N)$ then this relation is preserved for the outputs.*

The analysis of sorting networks is most often based on the following lemma [7]

Lemma 1.2 (zero–one principle). *A comparator network is a sorting network if and only if it can sort any input consisting only of 0s and 1s.*

This lemma is the reason, why from now on we consider inputs consisting only of 0s and 1s. Thus we consider only t-disturbed sequences consisting of 0s and 1s. We note, that 0-1 sequence $x_1, \ldots, x_N$ is t disturbed if for some index b called the *border* at most t entries in $x_1, \ldots, x_b$ are 1s and at most t entries in $x_{b+1}, \ldots, x_N$ are 0s. These 1s (0s) we call *displaced*.

Let us remind the proof of zero–one principle. The input consists of arbitrary elements. We prove that the comparator network sorts it. We consider an arbitrary a from this input and show it gets to the register corresponding to its rank in the sequence. We replace elements bigger than a by 1, and smaller by 0. Indeed the only difference between outputs for sequences where a is replaced by 0 or 1 respectively is the register with the index corresponding to rank(a).

Now we deal with an arbitrary t-disturbed input. We transform it to a t-disturbed 0-1 sequence as in the proof of zero–one principle. This gives us a useful analog of zero-one principle for t-correction networks.

Lemma 1.3. *A comparator network is a t-correction network if it can sort any t-disturbed input consisting of 0s and 1s.*

We define *dirty area* for 0-1 a sequence stored in the registers during computations of a comparator network. Dirty area is the smallest set of subsequent registers such that all registers with lower indices contain 0s and all registers with bigger indices contain 1s. A specialized comparator network that sorts any input having dirty area of a given size we call a *cleaning network*

2 Periodic Sorting Networks

In this section we remind the periodification scheme in [9]. Actually what we present is closer to the version of this scheme described by Oesterdiekhoff [10] which produces a network of depth 3. In comparison to previous authors we change the construction of Schwiegelsohn edges and embed only a single copies of sorting and merging networks. The analysis of the network is almost the same as in abovementioned papers and we do not show it.

The periodification scheme is a method to convert a non periodic sorting network having $T(p)$ layers for input size p into a periodic sorting network of depth 3. This periodic network sorts any input containing $\Theta(pT(p))$ items in $O(T(p)\log p)$ iterations. We take advantage of the fact, that for any sorting network $T(p) = \Omega(\log p)$. The periodification scheme applied to Batcher sorting network gives a periodic sorting network which needs $O(\log^3 N)$ iterations to sort an arbitrary input of size N. If we put AKS sorting network into this scheme, we get a periodic sorting network requiring $O(\log^2 N)$ iterations which is (due to very large constants in AKS) worse solution for practical N.

In the periodification scheme registers are indexed with pairs $(i, j), 1 \le i \le p, 1 \le j \le q$ ordered lexicographically. Thus we view these registers as arranged in rectangular matrix $p \times q$ of p rows and q columns. We have the rows with smallest indices i at the'top' and those with biggest indices at the 'bottom' of the array. We also view columns with smallest indices j to be on the left hand side and those with biggest indices to be on the right hand side. The parameter $q = 10(T(p) + \log p)$ is an even number (for simplicity from now on we write $\log p$ instead of $\lceil \log p \rceil$).

The periodic sorting network consists of three subsequent layers A, B and C. The layers A and B which are layers of odd-even transposition sort network are called horizontal steps. They are sets of comparators:

$$
\begin{aligned}
A &= \{[(i, 2j-1) : (i, 2j)] | i, j = 1, 2, \ldots\} \\
B &= \{[(i, 2j) : (i, 2j+1)] | i, j = 1, 2, \ldots\} \cup \{[(i, q) : (i+1, 1)] | i = 1, 2, \ldots\}
\end{aligned}
$$

The edges of A and B connecting registers of the same row we call *horizontal*. The layers A, B alone sort any input but in general the time to do it is very long.

Defining layer C called vertical step is much more complicated. We first divide the columns of registers into six subsequent areas: S, M_L, X_L, Y, X_R, M_R. Each of the areas contains an even number of columns. First two columns form an area S where so called 'Schwiegelsohn' edges are located. So the columns with numbers $3, 4, \ldots, 2\log p + 2$ are in the area M_L. Next $2T(p)$ columns form area X_L. Last $2\log p$ columns are contained in area M_R. Area X_R consists of $2T(p)$ columns directly preceding M_R. And the area Y contains all the columns between X_L and X_R. We now say where the comparators of layer C are in each area.

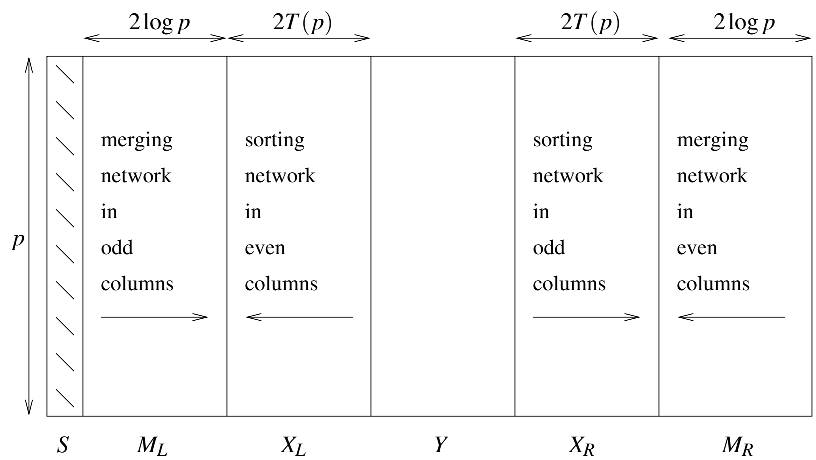

Fig. 1. Areas defined to embed C-layer. Arrows indicate the order of layers of embeded networks.

In area S the comparators form the set

$$\{[(2i-1, 1) : (2i, 2)] | i = 1, 2, \ldots\}$$

Note that this way of defining "Schwiegelsohn" edges differs from one described in previous papers on this subject. Comparators of C in all other areas unlike those in S connect always registers in the same column. There are no comparators in area Y on layer C.

In each area M_L and M_R we embed a single copy of a network which merges two sorted sequences of length $p/2$. In this network's input of length p even indexed entries are one sequence and odd indexed entries are the other. We also assume, that the sequence in odd indexed entries does not have more 1s than one contained in even indexed entries. A comparator network merging two such sequences is the series of layers $L_1, L_2, \ldots, L_{\log p-1}$ where

$$L_i = \{[2j : 2j + 2^{\log p - i} - 1] | j = 1,2,\ldots\}.$$

Thus the set of comparators in M_L is equal to

$$\{[(k,2j+1) : (l,2j+1)] | [k : l] \in L_j, j = 1,2,\ldots\}.$$

The set of comparators in M_R is equal to

$$\{[(k-1, q-2j+2)) : (l-1, q-2j+2)] | [k : l] \in L_j, j = 1,2,\ldots\}.$$

For technical reasons the network embedded in M_R is moved one row up.

Finally we define comparators in X_L and X_R. These comparators are embedding of a single sorting network in each area. Let this sorting network be the series of layers $L'_1, L'_2, \ldots, L'_{T(p)}$. Let $j_L = 2 + 2\log p + 2T(p)$ be the last column of X_L. The set of comparators in X_L is equal

$$\{[(k, j_L - 2(j-1)) : (l, j_L - 2(j-1))] | [k : l] \in L'_j, j = 1,2,\ldots\}.$$

Analogously if $j_R = q - 2\log p - 2T(p) + 1$ is the first column of X_R, then the set of comparators in X_R is equal

$$\{[(k, j_R + 2(j-1)) : (l, j_R + 2(j-1))] | [k : l] \in L'_j, j = 1,2,\ldots\}.$$

The edges connecting registers in the same column we call *vertical*. Almost all the edges of step C are vertical. Only the slanted edges in S are not vertical.

Our aim in the analysis of the network obtained in periodification scheme is to prove that it sorts any input in $O(T(p)\log p)$ steps. The proof easily follows from the key lemma

Lemma 2.1 (key lemma). *There exist constants c and d such that after $d \cdot q$ steps*

- *the bottom $c \cdot p$ rows contain only 1s if there are more 1s than 0s in the registers;*
- *the top $c \cdot p$ rows contain only 0s if there are more 0s then 1s in the registers.*

Indeed if we consider only the rows containing dirty area in the key lemma, then this area is guaranteed to be reduced by a constant factor within $O(q)$ steps. Thus applying the key lemma $O(\log p)$ times we reduce this area within $O(q \log p)$ steps to a constant number of rows. Next $O(q)$ steps sort such a reduced dirty area.

We do not describe the proof of key lemma, but define some notions from it to use them further in the paper. In this proof it is assumed, that two 1s or 0s compared by a horizontal edge are exchanged. In a given moment of computations we call an item (i.e. 0 or 1) right-running (left-running) if it is placed in the right (left) register by a horizontal edge of the recently executed horizontal step. We can extend this definition on wrap-around edges of layer B in a natural way saying that they put right-running items in the first column and left-running items in the last. A column containing right-running (left-running) items is called R-column (L-column). Analyzing the network we can observe 'movement' of R-columns of 1s to the right and L-columns of 0s to the left. Thus any column is alternately L-column and R-column and the change occurs during every horizontal step. The only exception are two columns of S. From the proof of key lemma it also follows, that we have the following property

Fact 2.2 *Assume we add any vertical edges to the layer C in area Y. For such a new network the key lemma still holds.*

Now we modify periodification scheme step by step to obtain at the end periodic t-correction network. First we introduce a construction of a *periodic cleaning network* sorting any N-element input with the dirty area of size $qt, q \geq 10(T(2t) + 2\log t)$. In this construction registers are arranged into q columns and dirty area is contained in t subsequent rows. This network needs $O(q \log t)$ iterations to do its job. The construction of periodic correction network is based on this cleaning network. We first build a simple non periodic cleaning network

Lemma 2.3. *Assume we have a sorting network of depth $T(t)$ for input size t. We can construct a comparator network of depth $T'(t) = T(2t) + \log t$ which sorts any input with dirty area of size t.*

Proof. We divide the set of all registers $r_1, r_2, \ldots, r_N$ into N/t disjoint parts each consisting of t subsequent registers. Thus we obtain part P_1 containing registers $r_1, \ldots, r_t$, P_2 containing registers $r_{t+1}, \ldots, r_{2t}$, P_3 containing registers $r_{2t+1}, \ldots, r_{3t}$, and so on. The cleaning network consists of two steps. First we have networks sorting keys in $P_{2i} \cup P_{2i+1}$ for each i. It requires $T(2t)$ layers. Then we have networks merging elements in P_{2i-1} with those in P_{2i} for each i. It requires $\log t$ layers of the network. □

Now we can build a periodic cleaning network. We do it substituting sorting network in the periodification scheme with the cleaning network described above. This way we can reduce X_L and X_R to $2T'(t)$ columns. We also reduce M_L and M_L to $2\log t$ columns, by embedding only $\log t$ last layers of merging network instead of the whole merging network applied in periodification scheme. These layers are (after relabeling) $L_1, L_2, \ldots, L_{\log t}$ where

$$L_i = \{[2j+1 : 2j + 2^{\log t - i + 1}] | j = 1, 2, \ldots\}.$$

They merge any two sequences that do not differ by more than $t/2$ 1s. So instead of a sorting network we use a cleaning one and we reduce the merging network. Such reduced sorting and merging networks are not distinguishable from original merging and sorting networks if we deal only with inputs having dirty areas of size at most qt. The analysis of such a periodification scheme for cleaning networks is the same as the original one for sorting networks and gives us the following fact

Lemma 2.4. *The periodic cleaning network described above has depth 3 and sorts any input with dirty area having t rows in $O(q \log t)$ iterations.*

One can notice that there are no edges of layer C in Y in this construction. If we add any vertical edges in Y or any other edges with the difference between row numbers of end registers bigger than t to layer C, then the network remains a cleaning network. Roughly speaking by adding such edges we are going to transform the periodic cleaning network into a periodic t-correction network.

3 Main Construction

In this section we define our periodic t-correction network. To do it we need another non periodic comparator network. We call it (t,Δ,δ)-*semi-correction* network. If a t-disturbed input with dirty area of size Δ is processed by such a network, then the dirty area size is guaranteed to be reduced to δ. Now we present quite unoptimal construction of (t,Δ,δ)-semi-correction network.

We divide the set of all registers $r_1, r_2, \ldots, r_N$ into N/Δ disjoint parts each consisting of Δ subsequent registers. Thus we obtain part P_1 containing registers $r_1, \ldots, r_\Delta$, P_2 containing registers $r_{\Delta+1}, \ldots, r_{2\Delta}$, P_3 containing registers $r_{2\Delta+1}, \ldots, r_{3\Delta}$, and so on. The construction consists of two steps. In step 1 we give new indices to the registers of each sum $P_{2k} \cup P_{2k+1}, k = 1, 2, \ldots$. These indices are lexicographically ordered pairs $(i, j), 1 \le i \le 2t\Delta/\delta, 1 \le j \le \delta/(2t)$. The ordering of new indices is the same as the main ordering of indices. We apply a t-correction network to each column j of each sum separately. This way we obtain dirty area of size at most δ in each sum. In step 2 we repeat the construction from step 1 for sums $P_{2k-1} \cup P_{2k}$ Because any dirty area of size Δ is contained in one of the sums $P_l \cup P_{l+1}$ from step 1 or 2, this dirty area is reduced to size δ. Thus we get the following lemma

Lemma 3.1. *Let $t \ll \delta$ and $t \ll \Delta/\delta$. There exists a (t,Δ,δ)-semi-correction network of depth $O\left(\log x + (\log t \log\log x)^2\right)$, where $x = \Delta/\delta$.*

Proof. Description of t-correction networks of depth $O\left(\log N + d(\log t \log\log N)^2\right)$ (N is the input size) can be found in [4, 16]. We apply such a network in the construction presented above and obtain a semi-correction network with desired depth. Simple calculations are left to the reader. □

Now at last we get to the main construction of this paper. We assume, that $\log N \ll t \ll N$ and want to construct an efficient periodic t-correction network. Without loss of generality we assume that t is even. Let $S(N,t) = O\left(\log N/\log t + (\log\log N)^2(\log t)^2\right)$ be the maximum depth of a (t,Δ,δ)-semi-correction network for $x = \Delta/\delta = N^{1/\log t}$. As before $T(t)$ is the depth of a sorting network. In our construction the registers are arranged into an array of q columns and N/q rows, where

$$q = \max\{10(T(4t+4) + 2\log t), 4(T(4t+4) + 2\log t) + 2S(N,t)\}$$

The rows of this array are divided into N/pq *floors* which are sets of $p = 4t+4$ subsequent rows. So the floor 1 consists of rows $1, 2, \ldots, p$, floor 2 of rows $p+1, p+2, \ldots, 2p$ and so on. We use the notions of 'bottom' and 'top' registers from the proof of key lemma. Thus we divide each floor into two halves: top and bottom. They consist of $p/2 = 2t+2$ top and bottom rows of each floor respectively. We define a *family of floors* to be a the set of all floors whose indices differ by $i \cdot \log t$ for some integer i. Altogether we have $\log t$ families of floors. To each family of floors we assign the index of its first floor.

From now on we all the time deal only with t-disturbed 0-1 input sequences. Any such a sequence has a border index b. The b-th register we call the border register. Its row we call the border row. Its floor we call the border floor. In the analysis we take into account only behavior of displaced 1s. Due to symmetry of the network the analysis for displaced 0s is the same and can be omitted.

We begin with defining a particular kind of periodic cleaning network, which the whole construction is based on. By adding comparators to this network we finally obtain a periodic t-correction network. The periodic cleaning network is constructed in the similar way as one in the previous chapter.

Above all we want to have some relation between vertical edges in areas X_L and X_R and the division of rows into floors. These comparators are embeddings of a cleaning network for dirty area $p/2 = 2t+2$ in each area. Note that such a network also sorts any input with dirty area of size t, so can be used in the construction of periodic cleaning network for t dirty rows. The cleaning network consists of three subsequent parts. The first part are sorting networks – each sorting a group of p subsequent registers corresponding to a single floor. This part has depth $T(p)$. The second part consists of merging networks which merge neighboring upper and lower halves of each pair of subsequent groups from the first part. It has depth $\log p$. The third part is the last layer which we can add arbitrarily, because any layer of comparators does nothing to a sorted sequence. This layer is defined a bit later in the paper.

Parts S, M_L, M_R are defined exactly the same way as earlier for a periodic cleaning network. So as we previously proved the periodic network we now defined is a cleaning network for dirty areas consisting of at most t rows and the following key lemma describing its running time holds

Lemma 3.2 (key lemma). *We consider $t', t' \leq t$ subsequent rows of above defined network, such that above (below) these rows there are only 0s (1s). Let we have majority of 0s (1s) in these rows. There exist constants c and d such that after $d \cdot q$ steps the top (bottom) $c \cdot t'$ of these t' rows contain only 0s (1s).*

Note that if we add to C any edges in Y or connecting rows whose difference is bigger than t, then the key lemma still holds and so all its consequences hold too. We prove the following lemma

Lemma 3.3. *The periodic cleaning network described above sorts considered inputs with dirty area having $a \cdot t$ rows in $O(qa + q\log t)$ iterations.*

Proof. If the number of rows in dirty area is smaller than t then a standard reasoning for periodic sorting networks works. We need only to consider what happens if the number of rows in dirty area is bigger than t. If there are at least highest $t/2$ rows of dirty area above the border row, then we can apply key lemma to these rows. Since the input is t-disturbed we have majority of 0s in these rows. So we obtain $ct/2$ top rows of 0s in time dq. Thus the dirty area is reduced by $ct/2$ rows. In the opposite case an analogous reasoning can be applied to $t/2$ lowest rows where we have majority of 1s. ◻

Now we add some comparators to layer C so that our network gains new properties. First we add in area S comparators

$$\{[(2i, 1) : (2i+t+1, 2)] | i\}.$$

To formulate the fact which follows from the presence of these comparators we must specify what exactly we mean by right-running items. In the proof of key lemma right-running items were those 0s and 1s which were on the right of a horizontal edge after step

A or B. We redefine it saying that in area S right-running items go right in step C instead of step A that is just after this step. Analogously we can redefine left-running items. We assume that two diplaced 1s or two 0s are swapped by an edge if this is an edge of step B or a slanted edge of step C or an edge of step A not belonging to area S. Displaced 1s are not swapped with non-displaced 1s. We can now formulate a simple property of our network that is preserved when we add edges

Fact 3.4 *In the network defined above right-running displaced 1s remain right-running as long as they are more than* $t+1$ *rows above border row.*

Now for a while we assume, that we deal only with displaced 1s that are more that one floor above the border. We remind, that R-columns and L-columns after a given step are columns containing right-running and left-running items respectively. We can note that R-column which gets to the column j_R while moving through X_R is first sorted separately on each floor by the first part of the cleaning network. Next the displaced 1s from each floor go half a floor down by the second part. An analogous process is also performed for left-running 1s in X_L as long as they remain left-running.

Thus after the second part of their way through X_R right-running displaced 1s are located at the bottom of the top half of each floor above the border floor. Analogously left-running displaced 1s are also moved just before the last layer embedded in X_L to the bottom of the top half of each floor.

We now should specify what the additional layer in the third part of X_R does. Formally speaking this layer is the set of comparators

$$\{[(kp+p/2+2i):(kp+p/2-2i+1)]|, 0<i\leq t/2\}.$$

It moves right-running displaced 1s that went through X_R to odd indexed rows in the middle of each floor. Analogously the last layer embedded in X_L is

$$\{[(kp+p/2+2i-1):(kp+p/2-2i)]|, 0<i\leq \lceil t/2\rceil\}.$$

It moves left-running displaced 1s to even indexed rows in the middle of each floor.

Let us call these rows for a while starting rows of these 1s. We can see that these all right-running displaced 1s then pass M_R, S, M_L and X_L without being moved by vertical edges in M_R, M_L, X_L. Note, that they encounter vertical edges only in M_L and they are at the bottom of these edges. The same happens to left-running 1s when they pass M_L, S, M_R and X_R. After passing X_L each right-running 1 is $t+2$ rows below its starting (odd) row. After passing X_R each left-running 1 is 2 rows above its starting (even) row. These 1s are still on the same floors as their starting rows. Similar facts can be proved for displaced 0s below the border which are also moved by last layers of X_L and X_R described above.

Now we define the vertical edges added in area Y of layer C. These comparators are embeddings of four semi-correction networks in each family of floors. Now we describe the comparators embedded in r-th family of floors. Let $a_1, a_2, \ldots, a_{2N/(q\log t)}$ be the indices of odd rows in this family of floors. We can build a $(t, N^{1-(r-1)/\log t}, N^{1-r/\log t})$-semi-correction network on registers with these indices. The depth of this network is not bigger (from the assumption about q) than the number of odd indexed columns in Y. Let this network be the sequence of layers $L_1, L_2, L_3, \ldots$. The first set of comparators is

$$\{[(j_L+2j-1,k):(j_L+2j-1,l):[k,l]\in L_j]\}.$$

We assumed that after passing X_L right-running 1s are in odd rows. Assume that they can be present only in $N^{1-(r-1)/\log t}$ odd rows of rth family directly above the border. When they pass Y they can be present only in $N^{1-r/\log t}$ odd rows of rth family directly above the border. Passing Y in family $r = \log t$ finally causes these 1s get to some of t odd rows of this family directly above the border. We formulate this assertion as a fact later because we need some additional assumptions. Analogously we can embed the same network once again to deal with left-running 1s that are in even rows. Formally speaking we add to C the following set of comparators

$$\{[(j_R - 2j + 1, k + 1) : (j_R - 2j + 1, l + 1) : [k,l] \in L_j]\}.$$

This set of edges again causes left running 1s which are in $N^{1-(r-1)/\log t}$ even rows of rth family directly above the border reduce the number of these rows between these 1s and the border to at most $N^{1-r/\log t}$. Analogously we also embed two copies of $(t, N^{r/\log t}, N^{(r-1)/\log t})$-semi-correction network to deal with displaced 0s below the border row.

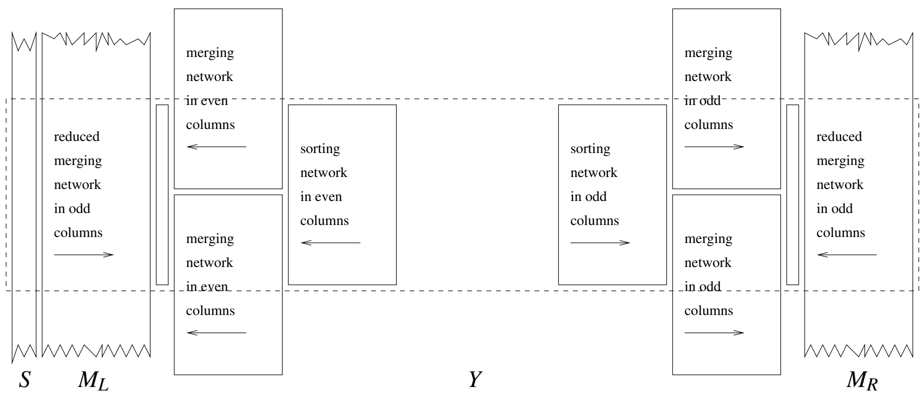

Fig. 2. Comparator networks embeded on a single floor.

We described the whole network and the way it works informally. To make this analysis more formal we assign colors to displaced 1s. We use five colors: blue, black, red, yellow and green. Let β be the index of the border floor. We assume the following rules of coloring displaced 1s:

- At the beginning the color of all displaced 1s is blue.
- If a blue 1 is compared with a non-blue 1 by a vertical edge, then the blue 1 behaves like a 0.
- When any 1 gets to the floor with the index not smaller than $\beta - 1$, it changes its color to green.
- When a right-running non-blue 1 gets to the floor $\beta - 2$, it changes its color to green.
- When a non-green left-running 1 changes to be right-running it becomes blue.
- When a blue 1 gets from Y to outside of Y it changes its color to black.

– When a black 1 enters Y from outside of Y on the floor belonging to the family 1, then it changes its color to red.
– When a red 1 leaves Y on the floor in the last family of floors (family $\log t$), then it becomes yellow.

First we prove, that all green 1s stay close to the border. They prove to be all the time at the floors with indices not smaller than $\beta - 2$, so they are not more than $13t$ rows above the border row. We notice that right-running 1s can go only to the lower rows. Left-running 1s can go to the higher rows only in area S of layer C and by wrap-around edges of layer B. So only left-running can go up from the floor $\beta - 2$. Each q horizontal steps a left-running 1 can go up by maximum $t + 2$ rows. But on the other hand each q horizontal steps it passes X_L once. Passing X_L it goes to the row t-th or lower counting from the bottom of floor $\beta - 2$. Thus it cannot leave floor $\beta + 2$ going up not more than $t + 2$ rows. Moreover because our network is periodic correction network for dirty area of t rows we have the following fact

Fact 3.5 *If all displaced 1s are green, then the time to sort all the items above the border is $O(q \log t)$.*

Now we consider a right-running blue 1 or a left-running blue 1 assuming it stays left-running. From what we said before a right-running 1 stops to be right-running only when it is green. We want to see how quickly it becomes green. After $O(q)$ steps this 1 stops to be blue. The worst case is that it becomes black. The following fact can be viewed as a summary of what we said defining comparators of last column of X_L and X_R. We take advantage of the fact that right-running 1s that just changed from being left-running above floor $\beta - 1$ are blue. We also take advantage of the fact, that right-running 1s which are more than t rows above the border do not become left-running. Such 1s is the only factor that could disturb the 1s we are interested in to go one floor down.

Fact 3.6 *Any black, red or yellow right-running 1 on the floor higher than $\beta - 1$ passing areas X_R, M_R, S, M_L, X_L goes one floor down and ends up in an odd indexed row. Any black, red or yellow left-running 1 on the floor higher than β passing areas X_L, M_L, S, M_R and X_R goes one floor down and ends up in an even indexed row.*

The comparators in Y connect only the rows belonging to the same family of floors. So passing Y a displaced 1 does not change its family of floors. Thus we have the next fact.

Fact 3.7 *Every q horizontal steps a black or red 1 gets from a family r to the family $r + 1$. The exception is family $r = \log t$ from which it gets to family 1.*

So after at most $q \log t$ horizontal steps a black 1 becomes red, unless it starts to be green. We measure the *distance* of a red 1 that is in family r to the border as the number of rows that belong to the family r and are between this 1 and the floor $\beta - 2$. Passing Y in family 1 a red 1 reduces this distance from at most N to at most $2N^{1-1/\log t}$. Then it gets to families $2, 3, \ldots, \log t - 1$. Passing Y in family r a red 1 reduces this distance from $2N^{1-(r-1)/\log t}$ to $2N^{1-r/\log t}$. Then after passing Y in family $\log t$ a red 1 is in the distance at most $2t$. This way a red 1 becomes yellow after $q \log t$ horizontal steps. Now it is at most $\log t + 2$ floors above the border. A yellow right-running 1 goes at least 1 floor down each q horizontal steps, till it becomes green after at most $q \log t$ horizontal steps.

This whole process of color change from blue to green takes altogether $3q\log t$ horizontal steps. It always succeeds for right-running 1s. Left-running 1s can switch to be right-running before they become green. They have to do it before $3q\log t$ horizontal steps in which they have to become green if they are all the time left-running. In such a case they become inevitably green after next $3q\log t$ iterations as right-running 1s. Thus we have the following fact.

Fact 3.8 *All disturbed 1s start to be green after at most* $6q\log t$ *horizontal steps.*

Because inputs having only green 1s are quickly sorted we get the main result of the paper

Theorem 3.9. *The periodic* t*-correction network we defined in this paper sorts any* t *disturbed input in* $O(q\log t)$ *iterations, which is equal to*

$$O(\log N + (\log\log N)^2(\log t)^3)$$

Acknowledgments

The author wishes to thank Marek Piotrów and other coworkers from algorithms and complexity group of his institute for helpful discussions.

References

1. M. Ajtai, J. Komolós, 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 (1968), 307-314.
3. M. Dowd, Y. Perl, M. Saks, L. Rudolph. The Periodic Balanced Sorting Network. *Journal of the ACM* 36 (1989), 738-757.
4. M. Kik, M. Kutyłowski, M. Piotrów, Correction Networks, in *Proc. of 1999 ICPP*, 40-47.
5. M. Kik, M. Kutyłowski, G. Stachowiak, Periodic constant depth sorting networks, *Proc. of the 11th STACS*, 1994, 201-212.
6. M. Kik, Periodic Correction Networks, *EUROPAR 2000 Proceedings*, LNCS 1900, 471-478
7. D. E. Knuth, *The Art of Computer Programming*, Vol 3, 2nd edition, Addison Wesley, Reading, MA, 1975.
8. J. Krammer, Lösung von Datentransportproblemen in integrierten Schaltungen. Dissertation, TU München 1991.
9. K. Loryś, M. Kutyłowski, B. Oesterdiekoff, R. Wanka, Fast and Feasible Periodic Sorting Networks of Constant Depth, *Proc of 35 IEEE-FOCS*, 1994, 369-380.
10. B. Oesterdiekoff, On the Minimal Period of Fast Periodic Sorting Networks, *Technical Report TR-RI-95-167*, University of Paderborn,1995.
11. M. Piotrów, Depth Optimal Sorting Networks Resistant to k Passive Faults in *Proc. 7th SIAM Symposium on Discrete Algorithms* (1996), 242-251 (also accepted for *SIAM J. Comput.*).
12. M. Piotrów, Periodic Random-Fault-Tolerant Correction Networks, *Proceedings of 13th SPAA*, ACM 2001, 298-305.
13. L. Rudolph, A Robust Sorting Network, *IEEE Transactions on Computers* 34(1985), 326-336.
14. M. Schimmler, C. Starke, A Correction Network for N-Sorters, *SIAM J. Comput.* 18 (1989), 1179-1197.
15. U. Schwiegelsohn. A shortperiodic two-dimensional systolic sorting algorithm. In International Conference on Systolic Arrays, Computer Society Press, Baltimore 1988, 257-264.
16. G. Stachowiak, Fibonacci Correction Networks, in *Algorithm Theory – SWAT 2000* , M Halldórsson (Ed.) , LNCS 1851, Springer 2000, 535-548.