Theoretical Computer Science 410 (2009) 1054–1060

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Linear-size log-depth negation-limited inverter for k-tonic binary sequences

Hiroki Morizumi [a], Jun Tarui [b,*]

[a] Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan

[b] Department of Information and Communication Engineering, University of Electro-Communications, Chofu, Tokyo 182-8585, Japan

ARTICLE INFO

Keywords:
Circuit complexity
Negation-limited circuit
Inverter
k-tonic

ABSTRACT

In *negation-limited* complexity, one considers circuits with a limited number of NOT gates, being motivated by the gap in our understanding of monotone versus general circuit complexity. In this context, the study of *inverters*, i.e., circuits with inputs $x_1, \ldots, x_n$ and outputs $\neg x_1, \ldots, \neg x_n$, is fundamental since an inverter with r NOTs can be used to convert a general circuit to one with only r NOTs. Beals, Nishino, and Tanaka [R. Beals, T. Nishino, K. Tanaka, On the complexity of negation-limited Boolean networks, SIAM Journal on Computing 27 (5) (1998) 1334–1347. A preliminary version appears in: Proceedings of STOC95: The 27th Annual ACM Symposium on Theory of Computing, 1995, pp. 585–595] gave a construction of an n-inverter with size $O(n \log n)$, depth $O(\log n)$, and $\lceil \log_2(n+1) \rceil$ NOTs. A zero–one sequence $x_1, \ldots, x_n$ is *k-tonic* if the number of i's such that $x_i \neq x_{i+1}$ is at most k. The notion generalizes well-known *bitonic* sequences. We give a construction of circuits inverting k-tonic sequences with size $O((\log k)\, n)$ and depth $O(\log k \, \log \log n + \log n)$ using $\log_2 n + \log_2 \log_2 \log_2 n + O(1)$ NOTs. In particular, for the case where $k = O(1)$, our k-tonic inverter achieves asymptotically optimal linear size and logarithmic depth. Our construction improves all the parameters of the k-tonic inverter by Sato, Amano, and Maruoka [T. Sato, K. Amano, A. Maruoka, On the negation-limited circuit complexity of sorting and inverting k-tonic sequences, in: Proceedings of COCOON06: The 12th Annual International Computing and Combinatorics Conference, in: Lecture Notes in Computer Science, vol. 4112, 2006, pp. 104–115]. We also give a construction of k-tonic *sorters* achieving linear size and logarithmic depth with $\log_2 \log_2 n + \log_2 \log_2 \log_2 n + O(1)$ NOT gates for the case where $k = O(1)$. The following question by Turán remains open: Is the size of any depth-$O(\log n)$ inverter with $O(\log n)$ NOT gates superlinear?

1. Introduction and summary

Although exponential lower bounds are known for the monotone circuit size [12,8,5], at present we cannot prove a superlinear lower bound for the size of circuits computing an explicit Boolean function. It is natural to ask: What happens if we allow a limited number of NOT gates? The hope is that by the study of *negation-limited* complexity of Boolean functions under various scenarios [6,7,4,3,2,13,9], we obtain a better understanding about the power of NOT gates. As mentioned in the abstract, the study of inverters is fundamental in this context since an inverter with r NOTs can be used to convert a general circuit to one with only r NOTs. In particular, if linear-size log-depth inverter with r NOTs exists, we do not lose generality by only considering circuits with at most r NOTs when we seek superlinear-size lower bounds or superlogarithmic-depth

* Corresponding author.
E-mail addresses: morizumi@kuis.kyoto-u.ac.jp (H. Morizumi), tarui@ice.uec.ac.jp (J. Tarui).

0304-3975/$ – see front matter
doi:10.1016/j.tcs.2008.10.030

Table 1
The parameters of the k-tonic inverters of Sato et al. [13] and this paper.

	Sato et al. [13]	This paper
Size	$O(kn)$	$O((\log k)n)$
Depth	$O(k\log^2 n)$	$O(\log k \log\log n + \log n)$
# of NOTs	$O(k\log n)$	$\log_2 n + \log_2\log_2\log_2 n + O(1)$

lower bounds. Markov [11] showed that the minimum number of NOT gates necessary in an n-inverter is $\lceil \log_2(n+1)\rceil$. We consider circuits consisting of AND/OR/NOT gates, and the *size* of a circuit is the number of gates in it. The best known construction of an n-inverter is due to Beals, Nishino, and Tanaka [4]. Their inverter has size $O(n\log n)$ and depth $O(\log n)$ and uses $\lceil \log_2(n+1)\rceil$ NOT gates.

In a recent paper [13], Sato, Amano, and Maruoka considered circuits that is guaranteed to invert a restricted class of inputs, and gave a construction for a *k-tonic inverter*, i.e., a circuit that inverts all k-tonic 0/1 sequences, with size $O(kn)$, depth $O(k\log^2 n)$, and $O(k\log n)$ NOTs. We give an entirely different construction of a k-tonic inverter achieving improvements of all the three parameters; see Table 1. In particular, for $k = O(1)$, we achieve asymptotically optimal linear size and logarithmic depth using only slightly more than $\log_2 n$ NOT gates:

Theorem 1. *There is a k-tonic inverter that has size $O((\log k)n)$ and depth $O(\log k\log\log n + \log n)$, and uses $\log_2 n + \log_2\log_2\log_2 n + O(1)$ NOT gates.*

Amano, Maruoka, and Tarui [3] considered the minimum size of a circuit that merges two 0/1 sequences using t NOT gates, and they showed that it is $\Theta(n\log n/2^t)$, thus demonstrating a smooth trade-off of size versus the number of NOTs from the monotone case of $\Theta(n\log n)$ to the general case of $\Theta(n)$. Their merging circuit actually works for any bitonic sequence. Sato, Amano, Maruoka [13] also considered a generalized scenario in terms of k-tonic sequences and, for $t \le \log_2 n$ and $k = O(\log n)$, they gave a construction of a *k-tonic sorter*, i.e., a circuit that sorts all k-tonic binary sequences, that has size $O(kn + (n\log n)/2^t)$ and uses $O(tk^2)$ NOT gates. The design principle and the analysis of our k-tonic inverter immediately yields an improved k-tonic sorter:

Theorem 2. *There is a k-tonic sorter that uses t NOT gates and has size $O\left((\log k)n + (n\log n)(t/2^t)\right)$ and depth $O((\log k)\,t + \log n)$.*

2. Component circuits/networks

In this section we explain the components that we use in our circuits. The constructions in Sections 2.1 and 2.2 are due to Beals, Nishino, and Tanaka [4]. The reader may choose to skip this section and come back to it after seeing how components are assembled and used in our circuits.

2.1. Inverting the inputs of a comparator network

Let N_1 be a *comparator network* (see, e.g., Knuth [10]) with inputs $v_1, \ldots, v_n$ and outputs $w_1, \ldots, w_n$. Assume that N_1 has depth d and contains s comparators. Consider the case where inputs are Boolean. In the Boolean case, each comparator can be considered as a pair of one AND gate and one OR gate (Fig. 1), and thus N_1 can be considered as a depth-d size-$2s$ monotone circuit.

Assume that the negations of the *outputs* of N_1, i.e., $\neg w_1, \ldots, \neg w_n$ are computed by another circuit and are available. Then, we can construct a circuit N_2 that outputs the negations of the *inputs* $\neg v_1, \ldots, \neg v_n$ as follows. For each comparator c with inputs x_1 and x_2 and outputs y_1 and y_2, we can compute $\neg x_1$ and $\neg x_2$ from $x_1, x_2, \neg y_1, \neg y_2$ as shown in Fig. 1. Repeatedly apply this construction considering comparators one by one from the outputs of N_1 towards the inputs, and obtain the network N_2. The circuit N_2 has depth $2d$ and consists of $2s$ ANDs and $2s$ ORs.

2.2. The Beals–Nishino–Tanaka inverter

The inverter operates as follows. Sort $x_1, \ldots, x_n$ by the AKS n-sorting network [1] with depth $O(\log n)$ and size $O(n)$, and obtain $y_1 \ge \cdots \ge y_n$. Apply Fischer's network M_n [6,7,4], and obtain $\neg y_1, \ldots, \neg y_n$. Finally, apply the network explained in Section 2.1 that outputs $\neg x_1, \ldots, \neg x_n$ using $\neg y_1, \ldots, \neg y_n$. Here M_n is a network that inverts a sorted 0/1-sequence $y_1 \ge \cdots \ge y_n$ with size $O(n)$ and depth $O(\log n)$ using $\lceil \log_2(n+1)\rceil$ NOT gates. (More precisely, for $n = 2^r - 1$, M_n has size $4n - 3r$; this is the minimum size [9] of circuits inverting n sorted inputs with r NOTS.) The inverter uses $\lceil \log_2(n+1)\rceil$ NOT gates and has depth $O(\log n)$ and size $O(n\log n)$.

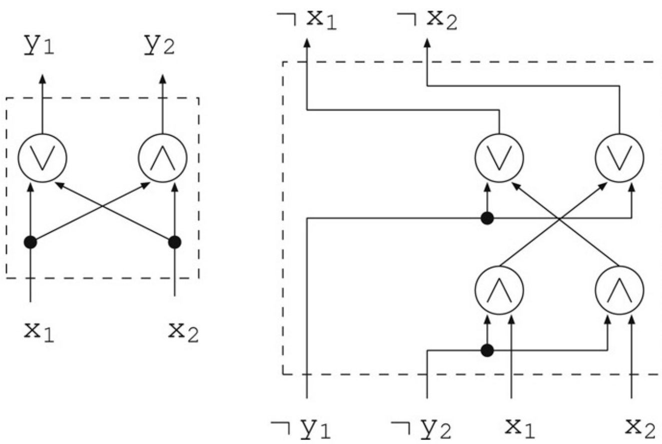

Fig. 1. Computing the negations of inputs using the negations of outputs.

2.3. Conditional shifter

Let $p \in \{0, 1\}$. Assume that $\delta \leq \alpha$ and let $y_1, \ldots, y_{\alpha+\delta}$ be a 0/1-sequence. Suppose that we want to let $z_1, \ldots, z_\alpha$ respectively be $y_1, \ldots, y_\alpha$ if $p = 1$ and $y_{1+\delta}, \ldots, y_{\alpha+\delta}$ if $p = 0$. In other words, we want to either (1) discard the last δ y_j's, or (2) discard the first δ y_j's and then shift by δ. This can be easily be done using p and $\neg p$ as follows: For $j = 1, \ldots, \alpha$, compute

$$z_j = (p \wedge y_j) \vee (\neg p \wedge y_{j+\delta}).$$

Further assume that the following conditions hold:

$$p = 0 \implies y_1 = \cdots = y_\delta = 0;$$
$$p = 1 \implies y_{\alpha+1} = \cdots = y_{\alpha+\delta} = 1.$$

Then, if we can use $\neg z_1, \ldots, \neg z_\alpha$, we can easily compute $\neg y_1, \ldots, \neg y_{\alpha+\delta}$ as follows.

$$\neg y_j = \begin{cases} (p \wedge \neg z_j) \vee \neg p & \text{for } j = 1, \ldots, \delta; \\ (p \wedge \neg z_j) \vee (\neg p \wedge \neg z_{j-\delta}) & \text{for } j = \delta + 1, \ldots, \alpha; \\ \neg p \wedge \neg z_{j-\delta} & \text{for } j = \alpha + 1, \ldots, \alpha + \delta. \end{cases}$$

3. Negation-limited *k*-tonic inverter

Most of this section is devoted to an explanation of our k-tonic inverter claimed in Theorem 1. In Section 3.1 we explain the overall structure of our k-tonic inverter. For the sake of exposition, we first consider computing *pivot bits* in a naive way, and we provide rough analysis for the number of NOT gates needed for reducing the problem size. In Section 3.2 we explain how we actually compute pivot bits in our circuit to achieve the claimed depth. It turns out that most of the work is giving appropriate *definitions* and developing an appropriate *framework* for analysis. In Section 3.3 we explain how we can achieve the number of NOT gates as claimed in Theorem 1 by a simple finer analysis. In Section 3.4 we explain how we can obtain our k-tonic sorter in a similar way.

3.1. Overall structure of the k-tonic inverter

We first explain using a general algorithmic language and then explain in terms of circuits. We consider a k-tonic binary input sequence of length n and assume that $n = k2^r$ for some integer $r \geq 2$. If n is not of this form, we can pad an input sequence $x = \langle x_1, \ldots, x_n \rangle$ with trailing 1's and obtain the sequence $x' = \langle x_1, \ldots, x_n, 1, \ldots, 1 \rangle$ whose length is the minimum $N > n$ of this form, apply the inverter for the $(k+1)$-tonic sequence x', and discard the last $N - n$ outputs.

Let $x = \langle x_1, \ldots, x_n \rangle$ be a k-tonic 0/1 sequence of length $n = 4km$. Think of x_i's as entries of a $4k \times m$ matrix M as follows. (We will not be doing any linear algebra; we can equally speak in terms of a rectangular array or a two-dimensional grid.)

$$M = \begin{pmatrix} x_1 & x_2 & \cdots & x_{m-1} & x_m \\ x_{m+1} & x_{m+2} & \cdots & x_{2m-1} & x_{2m} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{(4k-1)m+1} & x_{(4k-1)m+2} & \cdots & x_{4km-1} & x_{4km} \end{pmatrix}.$$

A row is *dirty* if it contains both 0 and 1; otherwise it is *clean*; an all-0 row is 0-*clean* and an all-1 row is 1-*clean*.

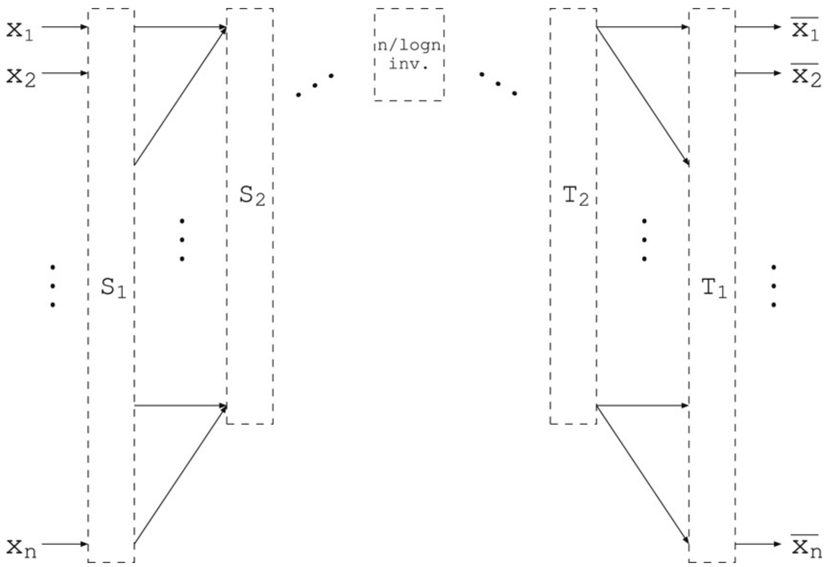

Fig. 2. S_1 contains multiple AKS sorting networks. A conditional shifter feeds the outputs of S_1 into S_2 after discarding the top half or the bottom half. The outputs of T_2 are the negations of the inputs of S_2. The inputs of T_1 are the negations of the outputs of S_1. T_1 unwinds S_1 as explained in Section 2.1.

Table 2
The parameters of subcircuits 1, 2, 3.

	Subcircuit 1	Subcircuit 2	Subcircuit 3
Size	$O((\log k)n)$	$O(n)$	$O((\log k)n)$
Depth	$O(\log n \log\log n)$	$O(\log n)$	$O(\log k \log\log n)$
# of NOTs	$2\log_2\log_2 n + O(1)$	$\log_2 n - \log_2\log_2 n + O(1)$	0

Since the sequence x is k-tonic, among the $4k$ rows of M, at most k rows are dirty. Sort each *column* of M with smaller entries up, and obtain the matrix $\overline{M_0}$. The matrix $\overline{M_0}$ has at most k dirty rows, and all of them as middle rows. Thus either the bottom $(4k - k)/2 = 3k/2$ rows are all 1-clean, or else the top $3k/2$ rows are all 0-clean. For now, we use the following weaker form: Either (1) all the bottom k rows are 1-clean or (2) all the top k rows are 0-clean.

Define *pivot bit* p as $p =$ AND of the km entries in the bottom k rows of $\overline{M_0}$. Use one NOT gate and obtain $\neg p$. Using p and $\neg p$ discard either the bottom k 1-clean rows or the top k 0-clean rows according to whether $p = 1$ or 0, i.e., whether (1) holds or not. The remaining $3k$-row matrix $\overline{M_1}$ has at most k dirty rows, and all of them as middle rows. Again discard the bottom $(3k - k)/2 = k$ rows or the top k rows using the pivot bit for the bottom k rows together with one NOT gate.

We are left with a $2k \times m$ matrix $\overline{M_2}$. Split each row of $\overline{M_2}$ into the first half and the last half. Let L be a $4k \times \frac{m}{2}$ matrix whose $4k$ rows are the $4k$ halves of the rows of $\overline{M_2}$. At most k rows of L are dirty. Thus using two NOT gates we have halved the problem size: We can apply the same operation and arguments for L, i.e., sort each column and discard $2k$ clean rows using two NOT gates.

We now explain in terms of circuits. We start over with the $4k \times m$ matrix M above. To sort each column of M, apply AKS sorting networks each sorting $4k$ elements; use m separate networks for m columns in parallel. Now consider the column-sorted matrix $\overline{M_0}$, and let $y_1, \ldots, y_n$ be the entries in its first row through its last row.

Consider, for now, computing the pivot bit p naively as $p = \wedge_{j=3km+1}^{4km} y_j$. Using one NOT gate compute $\neg p$. Discard the top k rows or the bottom k rows by the conditional shifter in Section 2.3: For $j = 1, \ldots, 3km$, compute $z_j = (p \wedge y_j) \vee (\neg p \wedge y_{j+k})$. The z_j's are the entries of the $3k \times m$ matrix $\overline{M_1}$. Assume that $\neg z_1, \ldots, \neg z_{3km}$ are the outputs of our subcircuit inverting $z_1, \ldots, z_{3km}$. We can use the conditional shifter for negations in Section 2.3 and obtain $\neg y_j$'s.

Continue halving the problem size $\nu = \log_2\log_2 n$ times so that the size is $n' = n/\log_2 n$, and then use the Beals–Nishino–Tanaka inverter for n' inputs.

Our circuit consists of three parts (Fig. 2):

Subcircuit 1: computing pivot bits and reducing the problem size from n to n'.

Subcircuit 2: the Beals–Nishino–Tanaka inverter for $n' = n/\log_2 n$ inputs.

Subcircuit 3: shifting the outputs using the pivot bits and obtaining the negations by "unwinding" the AKS soring networks as explained in Section 2.1.

The inverter explained so far has the parameters shown in Table 2. We provide some explanation for the parameters of subcircuit 1.

For $n = 4km$, consider the first parallel application of m separate AKS sorting networks each sorting $4k$ elements. This first part has size $O((\log k)n)$. Since the value of n geometrically decreases by a constant factor, the size of subcircuit 1 is dominated by the size of this first part. If we compute each pivot bit naively as taking the AND of some y_j's as above, this takes depth $O(\log n)$; we repeat this ν times; thus the depth of subcircuit 1 will be $O(\log n \cdot \nu) = O(\log n \log\log n)$. In the consideration above we halve the problem size by two NOT gates; thus a total of $2\nu = 2\log_2\log_2 n$ NOTs are used for the problem size reduction.

In Sections 3.2 and 3.3 we explain how to reduce the depth of subcircuit 1 to $O(\log k \log\log n)$ and the number of NOTs in subcircuit 1 to $\log_2\log_2 n + \log_2\log_2\log_2 n + O(1)$, and thus obtain a k-tonic inverter with the parameters claimed in Theorem 1.

3.2. Reducing the depth

Let M be an $l \times m$ 0/1-matrix. Let t be a nonnegative integer such that 2^t divides m, i.e., we can divide each row into 2^t consecutive parts of equal size. The parameter t represents the number of pivot bits in our circuit, which equals the number of times that we shrink the problem size by a constant factor, which also equals the number of NOT gates that we use.

For $s = 0, 1, \ldots, t$, we define an *s-block* of M as follows. Each row itself forms a 0-block; there are l 0-blocks. Split each 0-block, i.e., split each row into the first half and the last half. These halves are the $2l$ 1-blocks. Similarly, splitting each $(s-1)$-block yields two s-blocks; there are $2^s l$ s-blocks. Thus each row forms 2^s s-blocks for $s = 0, \ldots, t$, and hence forms a total of $u = \sum_{s=0}^{t} 2^s = 2^{t+1} - 1$ blocks. For each row, order these u blocks as follows. The first block is the 0-block, i.e., the whole row. Then comes the two 1-blocks, i.e., the first half and the last half, in this order. Then comes the four 2-blocks, i.e, the first quarter up to the last quarter; and so on.

Let $F = (f_{ij})$ be an $l \times u$ 0/1-matrix, where u is as above. Our intention will roughly be to let the equality $f_{ij} = 1$ represent the fact that the jth block in the ith row of M is 1-clean, i.e., all-1.

In our circuit, we call f_{ij} a *flag bit*. We compute flag bits f_{ij}'s *just once* as follows. For each t-block b, which is a smallest block, compute the flag bit f_b for block b as $f_b = \wedge_{x_i \in b} x_i$. For $s = t-1, \ldots, 0$, each s-block b contains two $(s+1)$-blocks b_1 and b_2; compute f_b as $f_b = f_{b_1} \wedge f_{b_2}$. Thus all flag bits are initially computed using only ANDs in depth $\lceil \log_2 m \rceil$. After initial computation, we *sort* flag bits column-wise using AKS sorting networks and discard bottom or top rows of flag bits as we discard top or bottom rows of input bits.

Right after the initial computation, the flag bit f_b for a block b is 1 iff the block b is 1-clean. After sorting f_b's, this may not hold: it is possible that a block b is 1-clean but $f_b = 0$. But sorting maintains the property that if $f_b = 1$, then b is 1-clean (we later call this property 1-*conservative*), and we show how this suffices for our purposes. We discard the bottom k rows of input bits if the kth largest flag bit is 1. In other words, the first pivot is computed in depth $O(\log n)$, but thereafter we use one output of AKS sorting network as pivot bit. This is how we obtain the claimed depth.

We proceed to show the correctness of the method above. We give definitions of key properties; Lemma 1 says that the properties hold after the initial computation of flag bits; Lemma 2 says that the properties are maintained by the operations above.

For two matrices M and F as above, the pair (M, F) is 1-*conservative* if the following holds: For $1 \le i \le l$ and $1 \le j \le u$, if $f_{ij} = 1$ then the jth block in the ith row of M is 1-clean. The jth block b in the ith row of M is *good* if either (1) b is 1-clean *and* $f_{ij} = 1$ or (2) b is 0-clean; otherwise b is a *bad* block. Say that (M, F) is *k-mixed* if there are at most k bad s-blocks for each $s = 0, \ldots, t$. Note that the definitions of 1-conservative and k-mixed are with respect to the parameter t. When appropriate, we make this dependence explicit by saying, e.g., *k-mixed with respect to t subdivisions*.

Let (M, F) be as above: M is an $l \times m$ matrix and F is an $l \times u$ matrix, where $u = \sum_{s=0}^{t} 2^s = 2^{t+1} - 1$ for a parameter t. *Stacking* (M, F) yields the pair of matrices $(\widehat{M}, \widehat{F})$, where $\widehat{M}$ is an $2l \times (m/2)$ matrix and $\widehat{F}$ is an $2l \times ((u-1)/2)$ matrix obtained as follows. Split each row of M into the first half and the last half. Stack the $2l$ halves thus obtained and obtain a $2l \times m/2$ matrix $\widehat{M}$. In other words, the first half and the last half of the ith row of M is respectively the ith row and the $(l+i)$th row of $\widehat{M}$. As for $\widehat{F}$: Throw away the first column of F, which corresponds to 0-blocks of M; the 0-blocks have been thrown away by stacking M into $\widehat{M}$. Put the second column of F on top of the third column; put the 4th and 5th columns on top of the 6th and 7th columns respectively; in general put the $(2^s + r)$th column on top of the $(2^s + 2^{s-1} + r)$th column $(0 \le r < 2^{s-1}, 1 \le s \le t)$, and obtain $\widehat{F}$.

Lemma 1. *Let $x_1, \ldots, x_n$ be a k-tonic 0/1-sequence of length $n = lm$. Let M be the $l \times m$ matrix having x_i's in a row-major form: e.g., the first row is $x_1, \ldots, x_m$. Consider s-blocks of M for $0 \le s \le t$. Let $F = (f_{ij})$ be the $l \times m$ 0/1-matrix such that f_{ij} = AND of all x_r's in the jth block of the ith row. Then, (M, F) is 1-conservative and k-mixed with respect to t subdivisions.*

Proof. By definition of f_{ij}'s, the pair (M, F) is 1-conservative; furthermore, clearly there is no 1-clean block with the corresponding flag bit f_{ij} being 0: In the setting above a block b is bad iff it is dirty. A 0–1 change in the sequence $x_1, \ldots, x_n$ produces at most one dirty s-block for each $s = 0, \ldots, t$. The lemma follows. □

Lemma 2. *Assume that (M, F) is 1-conservative and is k-mixed with respect to t subdivisions. Let $\overline{M}$ and $\overline{F}$ respectively denote the matrix obtained by sorting columns of M and F. Furthermore, let $\widehat{\overline{M}}$ and $\widehat{\overline{F}}$ respectively denote the matrix obtained by stacking $\overline{M}$ and $\overline{F}$. Then, $(\overline{M}, \overline{F})$ is 1-conservative and k-mixed with respect to t subdivisions. Further, $(\widehat{\overline{M}}, \widehat{\overline{F}})$ is 1-conservative and k-mixed with respect to $t-1$ subdivisions.*

Proof. For simplicity, first consider 0-blocks, i.e, rows of M and the corresponding first column $c = (c_{i1})$ of F. Assume that c contains α 1's, and that $c_{i_1 1} = c_{i_2 1} = \cdots = c_{i_\alpha 1} = 1$, where $i_1 < i_2 < \cdots < i_\alpha$.

Since (M, F) is 1-conservative, the α rows of M, rows $i_1, i_2, \ldots, i_\alpha$, are 1-clean. After column-sorting, the first column $\overline{c_{i1}}$ of $\overline{F}$ contains α 1's at the bottom, and the bottom α rows of $\overline{M}$ are 1-clean. Thus the condition of being 1-conservative holds with respect to the first column of F and the corresponding blocks, i.e., all the 0-blocks. Exactly the same argument applies for any column and the corresponding blocks. Hence $(\overline{M}, \overline{F})$ is 1-conservative.

Assume that M and F have l rows and that (M, F) is k-mixed. To see that $(\overline{M}, \overline{F})$ is k-mixed, again first consider 0-blocks, i.e., rows of M and the corresponding first column $c = (c_{i1})$ of F. Assume that M has α 1-clean good rows and β 0-clean good rows. By definition of goodness, the column c contains α 1's. After column-sorting, the bottom α rows are 1-clean and the bottom α entries of $\overline{c}$ are 1's; thus there are α 1-clean good blocks. The top β rows are 0-clean, and hence they are good. Thus there are as many good 0-blocks in $(\overline{M}, \overline{F})$ as in (M, F).

Now consider the 1-blocks of M, i.e., the first halves and the last halves of the rows. Assume that there are k_1 bad 1-blocks among the first halves and k_2 bad 1-blocks among the last halves, where $k_1 + k_2 \le k$. We can apply the above argument for 0-blocks separately for the first halves and for the last halves. In each of the two cases, after column-sorting there are as many good 1-blocks as before. Hence the assertion holds for 1-blocks. For the s-blocks, we can argue similarly separately considering 2^s groups of s-blocks. This completes our proof that $(\overline{M}, \overline{F})$ is k-mixed.

Finally, stacking does not destroy being 1-conservative nor does it introduce any new bad block. □

In our circuit stacking simply corresponds to rearranging the ordering of the intermediate gates; it does not need any gate. This completes our proof of the claimed depth reduction.

3.3. Reducing the number of NOTs

Consider, as in Section 3.1, a $4k$-row matrix M consisting of a k-tonic 0/1-sequence, and the matrix $\overline{M_0}$ obtained from M by sorting each column. Discard k rows using one NOT gate. Now, instead of again discarding as explained in Section 3.1, consider processing the remaining $6k$ 1-blocks; i.e., halve the $3k$ rows, stack the $6k$ halves, and consider the resulting $6k$-row matrix. At most k rows are bad. Discard $2k$ rows. (Actually we can discard $(6k - k)/2 = (5/2)k$ rows; this does not yield an asymptotic improvement.)

Further halve and stack to obtain $2(6k - 2k) = 8k$ rows, discard $3k$ rows, obtain $2(8k - 3k) = 10k$ rows, discard $4k$ rows, and so on. In general, at iteration s, discard sk rows out of $(2s + 2)k$ rows; halve and stack to obtain $((2s + 2)k - sk) \times 2 = 2sk + 4k = (2(s + 1) + 2)\, k$ rows. Thus with t NOT gates we can reduce the problem by the following factor. We assume that $t \ge 2$.

$$\frac{3}{4} \cdot \frac{4}{6} \cdot \frac{5}{8} \cdot \frac{6}{10} \cdot \cdots \cdot \frac{t+2}{2t+2} = \prod_{s=1}^{t} \frac{s+2}{2s+2} = (1/2)^{t-1}\,\frac{t+2}{4} \le (1/2)^t \cdot t,$$

where we have the second equality since each denominator is twice the previous numerator. Thus we can reduce the size to $1/\log_2 n$ using t NOT gates with t satisfying $2^t/t \ge \log_2 n$, and hence with $t = \log_2 \log_2 n + \log_2 \log_2 \log_2 n + O(1)$.

In the scheme above, the column size in column-sorting increases, and we use AKS k'-sorting networks for increasing k'. This increases the depth of subnetwork 1, but we can easily see that the asymptotic depth does not change. This completes the description of our k-tonic inverter as claimed in Theorem 1, and thus the proof of Theorem 1.

3.4. Negation-limited k-tonic sorter

We can obtain a k-tonic sorter in Theorem 2 as follows. Use exactly the same design as above to reduce the problem size to $n' = n\ (t/2^t)$ with t NOT gates. Then, instead of the Beals–Nishino–Tanaka inverter use the AKS n'-sorting network. Use shifters to obtain the output.

3.5. Open problems

The following question by Turán remains open: Is the size of any depth-$O(\log n)$ inverter with $O(\log n)$ NOT gates superlinear?

Acknowledgements

We thank the referees for TAMC07 and TCS for their valuable comments and suggestions.

References

[1] M. Ajtai, J. Komlós, E. Szemerédi, An $O(n \log n)$ sorting network, in: Proceedings of STOC83: The 15th Annual ACM Symposium on Theory of Computing, 1983, pp. 1–9.

[2] K. Amano, A. Maruoka, A superpolynomial lower bound for a circuit computing the clique function with at most $(1/6) \log \log n$ negation gates, SIAM Journal on Computing 35 (1) (2005) 201–216.

[3] K. Amano, A. Maruoka, J. Tarui, On the negation-limited circuit complexity of merging, Discrete Applied Mathematics 126 (1) (2003) 3–8.
[4] R. Beals, T. Nishino, K. Tanaka, On the complexity of negation-limited Boolean networks, SIAM Journal on Computing 27 (5) (1998) 1334–1347. A preliminary version appears in: Proceedings of STOC95: The 27th Annual ACM Symposium on Theory of Computing, 1995, pp. 585–595.
[5] R. Boppana, M. Sipser, The complexity of finite functions, in: J.V. Leeuwen (Ed.), Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity, Elsevier/MIT Press, 1990, pp. 757–804.
[6] M. Fischer, The Complexity of Negation-Limited Networks — A Brief Survey, in: Lecture Notes in Computer Science, vol. 33, 1975, pp. 71–82.
[7] M. Fischer, Lectures on network complexity, Technical Report 1104, (1974, revised 1996) CS Department, Yale University (available on the web).
[8] D. Harnik, R. Raz, Higher lower bounds on monotone size, in: Proceedings of STOC00: The 32nd Annual ACM Symposium on Theory of Computing, 2000, pp. 378–387.
[9] K. Iwama, H. Morizumi, J. Tarui, Negation-limited complexity of parity and inverters, in: Proceedings of ISAAC06: The 17th International Symposium on Algorithms and Computation, in: Lecture Notes in Computer Science, vol. 4280, 2006, pp. 223–232. a journal version is to appear in Algorithmica.
[10] D. Knuth, The Art of Computer Programming vol. 3: Sorting and Searching, 2nd edition, Addison-Wesley, 1998.
[11] A. Markov, On the inversion complexity of a system of functions, Journal of the ACM 5 (4) (1958) 331–334.
[12] A. Razborov, Lower bounds on the monotone complexity of some Boolean functions, Doklady Akademiya Nauk SSSR 281 (4) (1985) 798–801 (in Russian); English translation in: Soviet Math. Dokl. 31 (1985) 354–357.
[13] T. Sato, K. Amano, A. Maruoka, On the negation-limited circuit complexity of sorting and inverting *k*-tonic sequences, in: Proceedings of COCOON06: The 12th Annual International Computing and Combinatorics Conference, in: Lecture Notes in Computer Science, vol. 4112, 2006, pp. 104–115.