On the Negation-Limited Circuit Complexity of Sorting and Inverting k-tonic Sequences

Takayuki Sato[1], Kazuyuki Amano[2], and Akira Maruoka[3]

[1] Dept. of Information Engineering, Sendai National College of Technology
Chuo 4-16-1, Ayashi, Aoba, Sendai 989-3128, Japan
taka@info.sendai-ct.ac.jp
[2] Dept. of Computer Science, Gunma University
Tenjin 1-5-1, Kiryu, Gunma 376-8515, Japan
amano@cs.gunma-u.ac.jp
[3] Graduate School of Information Sciences, Tohoku University
Aoba 6-6-05, Aramaki, Sendai 980-8579, Japan
maruoka@ecei.tohoku.ac.jp

Abstract. A binary sequence $x_1, \ldots, x_n$ is called k-tonic if it contains at most k changes between 0 and 1, i.e., there are at most k indices such that $x_i \neq x_{i+1}$. A sequence $\neg x_1, \ldots, \neg x_n$ is called an *inversion* of $x_1, \ldots, x_n$. In this paper, we investigate the size of a negation-limited circuit, which is a Boolean circuit with a limited number of NOT gates, that sorts or inverts k-tonic input sequences. We show that if $k = O(1)$ and $t = O(\log \log n)$, a k-tonic sequence of length n can be sorted by a circuit with t NOT gates whose size is $O((n \log n)/2^{ct})$ where $c > 0$ is some constant. This generalizes a similar upper bound for merging by Amano, Maruoka and Tarui [4], which corresponds to the case $k = 2$. We also show that a k-tonic sequence of length n can be inverted by a circuit with $O(k \log n)$ NOT gates whose size is $O(kn)$ and depth is $O(k \log^2 n)$. This reduces the size of the negation-limited inverter of size $O(n \log n)$ by Beals, Nishino and Tanaka [6] when $k = o(\log n)$. If $k = O(1)$, our inverter has size $O(n)$ and depth $O(\log^2 n)$ and contains $O(\log n)$ NOT gates. For this case, the size and the number of NOT gates are optimal up to a constant factor.

1 Introduction

To derive a strong lower bound on the size of a Boolean circuit for a function in NP is one of the most challenging open problems in theoretical computer science. But so far, the best known lower bound is only a linear in the number of input variables. This is quite contrast to the case of monotone circuit, which consists only of AND and OR gates, no NOT gates. Exponential lower bounds on the size of monotone circuits for explicit functions have been derived (e.g., [2,5,8,13]).

This motivates us to study the complexity of circuits with a limited number of NOT gates, which are usually called the negation-limited circuits. About a half century ago, Markov [11] proved that $r = \lceil \log(n+1) \rceil$ NOT gates are enough to compute any function on n variables, and that there is a function that requires r NOT gates to compute[1]. Beals, Nishino and Tanaka [6] constructed a circuit with r NOT gates that computes the *inverter* $\mathsf{Inv}_n(x_1, \ldots, x_n) = (\neg x_1, \ldots, \neg x_n)$ whose size is $O(n \log n)$. Thus, for every function f, the size of a smallest circuit with at most r NOT gates that computes f is at most $2\mathrm{Size}(f) + O(n \log n)$, where $\mathrm{Size}(f)$ is the size of a smallest circuit for f. This shows that restricting the number of NOT gates in a circuit to $O(\log n)$ entails only a small blowup in circuit size. Recently, several lower bounds on the size of a negation-limited circuit for an explicit function were obtained [3,4], and the relationship between the number of NOT gates and circuit size was also studied [9,14]. However, it is still unclear the effect on circuit complexity of restricting the number of NOT gates available.

In the first half of the paper (Section 3), we focus on the negation-limited circuit complexity of the *sorting* function, which is a function that sorts n binary inputs. This is motivated by the result of Amano, Maruoka and Tarui [4] showing that for every $t = 0, \ldots, \log\log n$, the size complexity of the *merging* function with t NOT gates is $\Theta((n \log n)/2^t)$. Roughly speaking, the size of a smallest circuit for merging is halved when the number of available NOT gates increases by one. The merging function is a function that takes two presorted binary sequences each of length n as inputs and merges into a sorted sequence of length $2n$. The merging function can be viewed as the special case of the sorting function in which an input is restricted to the form of the concatenation of two sorted sequences. Interestingly, it is known that both merging and sorting have monotone circuit complexity of $\Theta(n \log n)$ and (non-monotone) circuit complexity of $\Theta(n)$. So it is natural to consider the negation-limited circuit complexity of sorting, or an intermediate function between merging and sorting.

In this paper, we parameterize a binary sequence with the number of changes of the values when it is read from left to right. Formally, a binary sequence $x_1, \ldots, x_n$ is called *k-tonic* if there are at most k indices i such that $x_i \neq x_{i+1}$. The k-tonic sorting function is a function that outputs a sorted sequence of $x_1, \ldots, x_n$ if an input is k-tonic, and arbitrarily otherwise. The merging function can be regarded as the 2-tonic sorting function since input sequences $x_1 \geq \cdots \geq x_n$ and $y_1 \geq \cdots \geq y_n$ are 2-tonic if we reorder them to $x_1, \ldots, x_n, y_n, \ldots, y_1$. We show that if k is a constant, the k-tonic sorting function can be computed by a circuit with $t (\leq \log n)$ NOT gates whose size is $O((n \log n)/2^{ct})$ for some constant $1 > c > 0$. This can be viewed as a generalization of a similar upper bound for the merging function in [4], which corresponds to the case $k = 2$.

In the second half of the paper (Section 4), we investigate the negation-limited complexity of the *inverter* Inv_n. As described before, Beals, Nishino and Tanaka [6] constructed an inverter of size $O(n \log n)$ and depth $O(\log n)$ that contains $\lceil \log(n+1) \rceil$ negation gates. In the same paper [6], they stated the following question as an open problem (which is credited to Turán in [6]) : is the size of any $c \log n$ depth inverter using $c \log n$ NOT gates superlinear?

[1] All logarithms in this paper are base 2.

We give the construction of an inverter for k-tonic sequences whose size is $O(kn)$ and depth is $O(k\log^2 n)$ that contains $O(k\log n)$ NOT gates. If $k = O(1)$, our inverter has size $O(n)$ and depth $O(\log^2 n)$ and contains $O(\log n)$ NOT gates. This shows that the answer of Turán's problem is "no" if we relax the depth requirement from $O(\log n)$ to $O(\log^2 n)$ and restrict the inputs to k-tonic sequence with k being a constant. Both of our results suggest that limiting the number of changes in an input sequence may boost the power of NOT gates in a computation of Boolean functions.

2 Preliminaries and Results

A *circuit* is a combinational circuit that consists of AND gates of fan-in two, OR gates of fan-in two and NOT gates. In particular, a circuit without NOT gates is called *monotone* circuit. The *size* of a circuit C is the number of gates in C.

Let F be a collection of m Boolean functions $f_1, f_2, \ldots, f_m$. The *circuit complexity* of F, denoted by $\mathrm{Size}(F)$, is the size of a smallest circuit that computes F. The *monotone circuit complexity* of F, denoted by $\mathrm{Size}_{mon}(F)$, is the size of a smallest circuit that computes F. Following Beals et al. [6], we call a circuit including at most t NOT gates a *t-circuit.* The *t-negation limited circuit complexity* of F, denoted by $\mathrm{Size}_t(F)$, is the size of a smallest t-circuit that computes F. If F cannot be computed by a t-circuit, then $\mathrm{Size}_t(F)$ is undefined.

For a binary sequence x, the length of x is denoted by $|x|$. For $x = (x_1, \ldots, x_t) \in \{0,1\}^t$, $(x)_2$ denotes the integer whose binary representation is x where x_1 is the most significant bit, i.e., $(x)_2 = \sum_{i=1}^{t} x_i 2^{t-i}$. The number of 1's in a binary sequence x is denoted by $\sharp_1(x)$. For two integers $a < b$, $[a,b]$ denotes the set $\{a, a+1, \ldots, b\}$. The set $[1,n]$ is simply denoted by $[n]$.

Definition 1. *The sorting function on n inputs, denoted by Sort_n, is a collection of Boolean functions that sorts an n-bit binary sequence $x_1, \ldots, x_n$, i.e.,*

$$\mathsf{Sort}_n(x_1, \ldots, x_n) = (z_1, \ldots, z_n),$$

such that $z_1 \geq \cdots \geq z_n$ and $\sum_i x_i = \sum_i z_i$. The merging function Merge_n is a collection of Boolean functions that merges two presorted binary sequences $x_1 \geq \cdots \geq x_n$ and $y_1 \geq \cdots \geq y_n$ into a sequence $z_1 \geq \cdots \geq z_{2n}$, i.e., $z_i = 1$ if and only if the total number of 1's in the input sequences is at least i. $\square$

The following results are known for the complexities of sorting and merging.

Theorem 1. *[12,1] All of the following are true:*

- $\mathit{Size}(\mathsf{Sort}_n) = \Theta(n)$ *and* $\mathit{Size}_{mon}(\mathsf{Sort}_n) = \Theta(n\log n)$.
- $\mathit{Size}(\mathsf{Merge}_n) = \Theta(n)$ *and* $\mathit{Size}_{mon}(\mathsf{Merge}_n) = \Theta(n\log n)$.

For the merging function, there is a clear tradeoff between the size of a circuit and the number of NOT gates.

Theorem 2. *[4] For every $0 \le t \le \log\log n$, $\mathit{Size}_t(\mathsf{Merge}_n) = \Theta((n \log n)/2^t)$.*

So it is interesting to consider whether such a tradeoff exists for more general functions.

Definition 2. *A turning point of a binary sequence $x_1, \dots, x_n$ is an index i such that $x_i \neq x_{i+1}$. A binary sequence is called k-tonic if it has at most k turning points.*

Note that every n-bit binary sequence is $(n-1)$-tonic, and that input sequences to the merging function $x_1 \ge \dots \ge x_n$ and $y_1 \ge \dots \ge y_n$ can be regarded as 2-tonic if we reorder the sequences to $x_1, \dots, x_n, y_n, \dots, y_1$. Thus, we can define an "intermediate" function between merging and sorting based on the notion of k-tonic.

Definition 3. *A function $\{0,1\}^n$ to $\{0,1\}^n$ whose output is equal to the output of Sort_n for every k-tonic input is called a k-sorting function and is denoted by Sort_n^k. Note that the output of Sort_n^k is arbitrary if an input is not k-tonic.*

We show in Section 3 that a small number of NOT gates can reduce the size of a circuit for Sort_n^k, which extends the results on the upper bounds in Theorem 2. Precisely, we will show:

Theorem 3. *Suppose that $k = O(\log n)$ and $t \le \log n$. Then there exists a constant c such that $\mathit{Size}_{ctk^2}(\mathsf{Sort}_n^k) = O(kn + (n \log n)/2^t)$. In particular, if $k = O(1)$ and $t = O(\log\log n)$, then $\mathit{Size}_t(\mathsf{Sort}_n^k) = O((n \log n)/2^{c't})$ for some constant $c' > 0$.*

In Section 4, we give the construction of an inverter for k-tonic sequences.

Definition 4. *An inverter with n binary inputs, denoted by Inv_n, is defined by*

$$\mathsf{Inv}_n(x_1, x_2, \dots, x_n) = (\neg x_1, \neg x_2, \dots, \neg x_n).$$

A function $\{0,1\}^n$ to $\{0,1\}^n$ whose output is equal to the output of Inv_n for every k-tonic input is called a k-tonic inverter and is denoted by Inv_n^k. Note that the output of Inv_n^k is arbitrary if an input is not k-tonic.

We will show:

Theorem 4. *The function Inv_n^k can be computed by a circuit of size $O(kn)$ and of depth $O(k \log^2 n)$ that contains $O(k \log n)$ NOT gates. In particular, if $k = O(1)$, then Inv_n^k can be computed by a linear size circuit of depth $O(\log^2 n)$ that contains $O(\log n)$ NOT gates.*

We remark that we need $\Omega(\log n)$ NOT gates to compute Inv_n^k even if $k = 1$. This can be easily proved by the result of Markov [11] (see also [6]). Thus, for the case $k = O(1)$, the size and the number of NOT gates are optimal up to a constant factor.

3 Negation-Limited Sorter for k-tonic Sequences

In this section, we describe the construction of a negation-limited circuit for the k-sorting function to prove Theorem 3.

As for the construction of a linear size sorter by Muller and Preparata [12], the construction is in two stages: the first computes the binary representation of the number of 1's in inputs, and the second generates appropriate outputs from this representation. Throughout this section, we assume that the length n of an input sequence is $n = 2^l$ for some natural number l.

Definition 5. *A counter* Count_n *is a function from* $\{0,1\}^n$ *to* $\{0,1\}^{\log n+1}$ *that outputs the binary representation of the number of 1's in an input sequence. A decoder* Decode_n *is a function from* $\{0,1\}^{\log n+1}$ *to* $\{0,1\}^n$ *such that* $\mathsf{Decode}_n(u) = (x_1, \ldots, x_n)$ *with* $x_1 = \cdots = x_{(u)_2} = 1$ *and* $x_{(u)_2+1} = \cdots = x_n = 0$.

It is obvious that $\mathsf{Sort}_n(x) = \mathsf{Decode}_n(\mathsf{Count}_n(x))$, and thus the size of a circuit for Sort_n is given by the sum of the sizes of circuits for Decode_n and Count_n. The linear sized sorter by Muller and Preparata [12] follows from:

Theorem 5. *[12]* $Size_n(\mathsf{Count}_n) = \Theta(n)$ *and* $Size_{mon}(\mathsf{Decode}_n) = \Theta(n)$.

Since there is a *monotone* circuit for Decode_n whose size is linear, one may think that it is sufficient to focus on the construction of a negation-limited circuit for Count_n. However, the last bit of the output of Count_n is the parity function, and so we need $\log n$ NOT gates to compute it [11]. In order to avoid to use such a large number of NOT gates, we only compute a limited number of significant bits of the number of 1's in inputs at the first stage of the construction.

Definition 6. *A t-counter, denoted by* $\mathsf{Count}_{n,t}$*, is a function from* $\{0,1\}^n$ *to* $\{0,1\}^{t+n/2^t}$ *defined as*

$$\mathsf{Count}_{n,t}(x) = (z, u),$$

where $z \in \{0,1\}^t$ *is the* t *most significant bits of the binary representation of the number of 1's in* x *and* $u = (u_1, \ldots, u_{n/2^t}) \in \{0,1\}^{n/2^t}$ *is a sorted sequence* $u_1 \geq u_2 \geq \cdots \geq u_{n/2^t}$ *such that* $\sharp_1(x) = (z)_2 \cdot n/2^t + \sharp_1(u)$*. A function whose output coincides with* $\mathsf{Count}_{n,t}(x)$ *for every* k*-tonic sequence* x *is denoted by* $\mathsf{Count}^k_{n,t}$.

A t-decoder, denoted by $\mathsf{Decode}_{n,t}$*, is a function from* $\{0,1\}^{t+n/2^t}$ *to* $\{0,1\}^n$ *defined as: For any binary sequence* z *of length* t *and any sorted sequence* u *of length* $n/2^t$,

$$\mathsf{Decode}_{n,t}(z, u) = (y_1, \ldots, y_n)$$

such that $y_1 = \cdots = y_w = 1$ *and* $y_{w+1} = \cdots = y_n = 0$ *where* $w = (z)_2 \cdot n/2^t + \sharp_1(u)$.

Note that $\mathsf{Sort}^k_n(x) = \mathsf{Decode}_{n,t}(\mathsf{Count}^k_{n,t}(x))$. We first construct a negation-limited circuit for $\mathsf{Count}^k_{n,t}$.

Theorem 6. *Suppose that $k = O(\log n)$ and $t \leq \log n$. Then there exists a constant c such that $Size_{ctk^2}(\mathsf{Count}^k_{n,t}) = O(kn + (n \log n)/2^t)$.*

Proof. We first show an algorithm for computing $\mathsf{Count}^k_{n,t}$, and then we will describe a construction of a circuit which follows the algorithm.

A binary sequence is called *clean* if it consists of 0's only or 1's only, otherwise it is called *dirty.* Let x be an input sequence for $\mathsf{Count}^k_{n,t}$. The key observation to the algorithm is the fact that if we divide a k-tonic sequence x into $2k$ blocks, then at least half of them are clean. For simplicity, we suppose that $|x| = n = 2^l$ and $k = 2^a$ for some natural numbers l and a with $l > a$.

Algorithm C. This algorithm takes a binary sequence x of length n as an input and outputs (z, u) satisfying $\mathsf{Count}^k_{n,t}(x) = (z, u)$.

```
C1. For each $i = 1, 2, \ldots, t$, do the following:
    1. Divide $x$ into $2k$ blocks of equal length: $B_1, B_2, \ldots, B_{2k}$.
    2. Let $p_1, p_2, \ldots, p_k \in [2k]$ be the first $k$ indices of clean blocks.
    3. For each $j \in [k]$, let $c_{i,j} = 1$ if $B_{p_j}$ is all 1's and $c_{i,j} = 0$ if $B_{p_j}$ is all 0's.
    4. Let $\tilde{x}$ be a sequence of length $|x|/2$ obtained from $x$ by removing $B_{p_1}, B_{p_2}$,
       $\ldots, B_{p_k}$ (i.e., the first $k$ clean blocks).
    5. Substitute $x$ by $\tilde{x}$.
C2. Let $z_H$ and $z_L$ be two binary sequences of length $t$ and of length $a$ such
    that
        $(z_H)_2 = \sum_{j=1}^{k} (c_{1,j} c_{2,j} \cdots c_{t,j})_2 \text{ div } 2^a, \qquad (z_L)_2 = \sum_{j=1}^{k} (c_{1,j} c_{2,j} \cdots c_{t,j})_2 \bmod 2^a,$
    where "div" and "mod" denote the quotient and remainder of two integers.
C3. Let $u^1$ be a sorted sequence of length $n/2^t$ that contains $(z_L)_2 \cdot 2^{l-(a+t)}$ 1's,
    and $u^2$ be a sorted sequence of length $n/2^t$ obtained by sorting $x$.
C4. If $\sharp_1(u^1) + \sharp_1(u^2) \geq n/2^t$, then let $z$ be a sequence of length $t$ such that
    $(z)_2 = (z_H)_2 + 1$ and let $u$ be a sorted sequence of length $n/2^t$ that contains
    $\{\sharp_1(u^1) + \sharp_1(u^2) - n/2^t\}$ 1's. Otherwise, let $z = z_H$ and let $u$ be a sorted
    sequence of length $n/2^t$ that contains $\{\sharp_1(u^1) + \sharp_1(u^2)\}$ 1's.
C5. Output $(z, u)$.
```

In the following we show the correctness of the above algorithm. Consider the i-th iteration of the for loop at step C1 of the algorithm. Let x and $\tilde{x}$ be binary sequences before and after the i-th iteration. Suppose that a sequence x is k-tonic. This means that there are at most k dirty blocks, or equivalently, at least k clean blocks in $B_1, \ldots, B_{2k}$. So we can always choose k indices $p_1, \ldots, p_k$. It is easy to check that the sequence $\tilde{x}$ is also k-tonic. Since each block has length $n/(k \cdot 2^i) = 2^{l-(a+i)}$, it is obvious that

$$\sharp_1(x) = \sum_{j=1}^{k} c_{i,j} \cdot 2^{l-(a+i)} + \sharp_1(\tilde{x}).$$

By summing the above equation over $i = 1, \dots, t$, the number of 1's in an initial input sequence is given by

$$\begin{aligned}\sum_{j=1}^{k}(c_{1,j}c_{2,j}\cdots c_{t,j})_2 \cdot 2^{l-(a+t)} + \sharp_1(u^2) &= (z_H)_2 \cdot 2^{l-t} + (z_L)_2 \cdot 2^{l-(a+t)} + \sharp_1(u^2)\\ &= (z_H)_2 \cdot 2^{l-t} + \sharp_1(u^1) + \sharp_1(u^2)\\ &= (z)_2 \cdot 2^{l-t} + \sharp_1(u).\end{aligned}$$

This completes the proof of the correctness of the algorithm.

Now we describe the construction of a circuit along the algorithm C starting from step C1, which is the most complex part of the construction. As for the above discussion, we first concentrate on the i-th iteration of the for loop, and so describe the construction of a circuit that takes sequences $B_1, \dots, B_{2k}$ each of length $n/(k \cdot 2^i)$ as an input and outputs $c_{i,j}$ for $j \in [k]$ and $B_{s_1}, \dots, B_{s_k}$ where $(s_1, \dots, s_k) = [2k] \backslash (p_1, \dots, p_k)$.

Given $(B_1, \dots, B_{2k})$, we put $B^{(0)} = (B_1^{(0)}, \dots, B_{2k}^{(0)}) = (B_1, \dots, B_{2k})$. For each $p \in [k]$, define $B^{(p)} = (B_1^{(p)}, \dots, B_{2k-p}^{(p)})$ as

$$B_q^{(p)} = \begin{cases} B_q^{(p-1)} & \text{if all of } B_1^{(p-1)}, \dots, B_q^{(p-1)} \text{ are dirty,}\\ B_{q+1}^{(p-1)} & \text{there exists a clean block in } B_1^{(p-1)}, \dots, B_q^{(p-1)}.\end{cases} \tag{1}$$

In other words, $B^{(p)}$ is a sequence obtained from $B^{(p-1)}$ by removing the first clean block in it. Then $B^{(p)}$ is equal to a sequence obtained from $B^{(0)}$ by removing the first p clean blocks. Hence $B^{(k)} = (B_1^{(k)}, \dots, B_k^{(k)})$ is a desired sequence.

Now we introduce two types of auxiliary Boolean functions. For $p \in [0, k]$ and for $q \in [2k - p]$, let $\mathsf{Is\_Clean}_q^{(p)}$ be a function that outputs 1 if and only if $B_q^{(p)}$ is clean and let $\mathsf{Exist\_Clean}_q^{(p)}$ be a function that outputs 1 if and only if there exists a clean block in $B_1^{(p)}, \dots, B_{q-1}^{(p)}$. These functions can be easily computed in a following way:

$$\begin{aligned}\mathsf{Is\_Clean}_q^{(0)} &= (\bigwedge_{v \in B_q^{(0)}} v) \vee \overline{(\bigvee_{v \in B_q^{(0)}} v)}, \quad (\text{for } q \in [2k]),\\ \mathsf{Exist\_Clean}_0^{(p)} &= 0, \quad (\text{for } p \in [0, k]),\\ \mathsf{Exist\_Clean}_q^{(p)} &= \mathsf{Exist\_Clean}_{q-1}^{(p)} \vee \mathsf{Is\_Clean}_q^{(p)}, \quad (\text{for } p \in [0, k], q \in [2k - p]),\\ \mathsf{Is\_Clean}_q^{(p)} &= (\mathsf{Is\_Clean}_{q+1}^{(p-1)} \wedge \mathsf{Exist\_Clean}_q^{(p-1)})\\ &\quad \vee (\mathsf{Is\_Clean}_q^{(p-1)} \wedge \overline{\mathsf{Exist\_Clean}_q^{(p-1)}}), (\text{for } p \in [k], q \in [2k - p]).\end{aligned}$$

Thus by Eq. (1), for each $l = 1, 2, \dots$, the l-th bit of $B_q^{(p)}$ is given by

$$(B_{q+1}^{(p-1)}[l] \wedge \mathsf{Exist\_Clean}_q^{(p-1)}) \vee (B_q^{(p-1)}[l] \wedge \overline{\mathsf{Exist\_Clean}_q^{(p-1)}}),$$

where $B[l]$ denotes the l-th bit of the block B. We also have

$$c_{i,j} = \bigvee_{q=1}^{k+1} \left(\mathsf{Is\_Clean}_q^{(j-1)} \wedge \overline{\mathsf{Exist\_Clean}_{q-1}^{(j-1)}} \wedge B_q^{(j-1)}[1] \right),$$

where $B_q^{(j-1)}[1]$ denotes the first bit of the block $B_q^{(j-1)}$.

Now we estimate the number of gates needed to compute these functions. Let $n_i = n/(2^{i-1})$, which is the length of an input sequence at the beginning of the i-th iteration of step C1. We use NOT gates at the computation of $\mathsf{Is\_Clean}_q^{(0)}$ for each $q \in [2k]$ and $\overline{\mathsf{Exist\_Clean}_q^{(p)}}$ for each $p \in [0, k]$ and $q \in [2k - p]$. So the number of NOT gates we need is at most $2k + 2k^2 \leq 3k^2$. The total number of gates is easily shown to be $O(k^2 + kn_i)$. Summing these over $i = 1, \ldots, t$, we need at most $3tk^2$ NOT gates and $\sum_{i=1}^{t} O(k^2 + kn_i) = O(tk^2 + kn)$ gates in total to simulate step C1 of the algorithm.

In step C2, all we have to do is to compute the addition of k integers of t bits. Since it is well known that the addition of two t-bit integers can be computed by a circuit of linear size (see e.g., [16, Chapter 3]), the number of gates needed to compute the addition of k integers of t bits is $O(k(t + \log k))$. The term $t + \log k$ here comes from the fact that the summand has at most $t + \log k$ digits. Here we use kt NOT gates, which is equal to the number of total input variables.

In step C3, we obtain u^1 as $\mathsf{Decode}_{n/2^t}(0z_L0^{l-(a+t)})$, which can be computed by a monotone circuit of size $O(n)$ by Theorem 5, and obtain u^2 as $\mathsf{Sort}_{n/2^t}(x)$, which can be computed by a monotone circuit of size $O((n/2^t)\log(n/2^t)) = O((n \log n)/2^t)$ by using the AKS-sorting network [1].

We can now proceed to step C4. Let $\tilde{u}$ be a sorted sequence of the concatenation of u^1 and u^2, which can be computed by a monotone circuit of size $O((2n/2^t)\log(2n/2^t)) = O((n \log n)/2^t)$ by using the AKS-sorting network [1]. Let $w \in \{0, 1\}$ be the $n/2^t$-th bit of $\tilde{u}$. Then $w = 1$ if and only if $\sharp_1(u^1) + \sharp_1(u^2) \geq n/2^t$. Thus, the desired sequence u is obtained by taking the first half of $\tilde{u}$ if $w = 0$ and the last half of $\tilde{u}$ if $w = 1$, which can be computed as $u_i = \overline{w}\tilde{u}_i \vee w\tilde{u}_{i+n/2^t}$ where u_i and $\tilde{u}_i$ denote the i-th bit of u and $\tilde{u}$, respectively. Clearly, z is given by the binary representation of $(z_H)_2 + w$ which can be computed by a t-bit adder. All these can be computed by a circuit of size $O((n \log n)/2^t)$ with $O(t)$ NOT gates.

The following table summarizes the number of gates used in each step.

Step	NOT gates	Total Size
C1	$O(tk^2)$	$O(tk^2 + kn)$
C2	kt	$O(k(t + \log k))$
C3	0	$O((n \log n)/2^t)$
C4	$O(t)$	$O((n \log n)/2^t)$

By summing these numbers, we conclude that the number of NOT gates in our circuit is $O(tk^2)$, and the total size is

$$O(tk^2 + kn + k(t + \log k) + (n \log n)/2^t) = O(kn + (n \log n)/2^t).$$

Here we use the assumption that $k = O(\log n)$ and $t \leq \log n$. This completes the proof of the theorem. □

We can now proceed to the construction of a circuit for $\mathsf{Decode}_{n,t}$.

Theorem 7. *Suppose that $t \leq \log n$. Then $Size_{mon}(\mathsf{Decode}_{n,t}) = O(n)$.*

Proof. Let $z \in \{0,1\}^t$ and $u \in \{0,1\}^{n/2^t}$ be inputs to $\mathsf{Decode}_{n,t}$. For such inputs, the output of $\mathsf{Decode}_{n,t}$ should be

$$\overbrace{11\cdots11}^{(z)_2\cdot n/2^t}u_1\cdots u_{n/2^t}00\cdots00.$$

For a binary sequence S, $S[i]$ denotes the i-th bit of S. Let A be an n-bit binary sequence given by 2^t copies of u. Put $B = \mathsf{Decode}_n(0z0^{l-t})$ and $C = \mathsf{Decode}_n(0z1^{l-t})$ Recall that $n = 2^l$. Let D be an n-bit binary sequence given by $D[i] = (A[i] \vee B[i]) \wedge C[i]$ for $i \in [n]$. Then the sequence D is

$$\overbrace{11\cdots11}^{(z)_2\cdot n/2^t}u_1\cdots u_{n/2^t-1}00\cdots00,$$

which is very close to the desired sequence, i.e., it misses the last bit of u. This discrepancy is fixed by putting $D[in/2^t] = (D[in/2^t-1] \wedge u_{n/2^t}) \vee D[in/2^t]$ for each $i \in [2^t]$. Since Decode_n has a linear size monotone circuit (Theorem 5), the sequence D can also be computed by a monotone circuit of linear size. □

Theorem 3 follows immediately from Theorems 6 and 7.

4 Negation-Limited Inverter for *k*-tonic Sequences

In this section, we describe the construction of a negation-limited circuit for the k-tonic inverter to prove Theorem 4. Throughout this section, we suppose that the length n of an input is $2^a - 1$ for some natural number a. We first introduce several auxiliary functions.

Definition 7. *Let $b \in \{0,1\}$. Let $\mathsf{Left}^b_n : \{0,1\}^n \to \{0,1\}^a$ be the collection of Boolean functions defined as $\mathsf{Left}^b_n(x) = p$ if $x_1 = \cdots = x_{(p)_2-1} = 1-b$ and $x_{(p)_2} = b$, i.e., p is the binary representation of the smallest index i with $x_i = b$. If there are no b's in x, then the output of Left^b_n is unspecified. Let $\mathsf{Decode}^b_n : \{0,1\}^a \to \{0,1\}^n$ be the collection of Boolean functions defined as $\mathsf{Decode}^b_n(p) = b^{(p)_2-1}(1-b)^{n-(p)_2+1}$. If p is all 0's then the output of Decode^b_n is unspecified. Let Or_n and And_n denote the functions that output bitwise OR and AND of two input sequences, respectively.*

Lemma 8. *For each $b \in \{0,1\}$, Left^b_n can be computed by a circuit of size $O(n)$ and of depth $O(\log^2 n)$ that contains $O(\log n)$ NOT gates.*

Proof. We first give an algorithm to compute Left_n^1.

Algorithm L. This algorithm takes an n-bit binary sequence $x = (x_1, \dots, x_n)$ as an input and outputs $p = (p_1, \dots, p_a)$ which satisfies $x_1 = \cdots = x_{(p)_2 - 1} = 0$ and $x_{(p)_2} = 1$.

```
L1. Let $x^0 = x$ and $flag_0 = 1$.
L2. For $i = 1, \dots, a-1$ do the following:
    1. $p_i = \neg(\bigvee_{j=1}^{2^{a-i}-1} x_j^{i-1}) \wedge flag_{i-1}$,
    2. $x_j^i = \neg p_i x_j^{i-1} \vee p_i x_{j+2^{a-i}}^{i-1}$ (for $j = 1, \dots, 2^{a-i}-1$),
    3. $flag_i = \neg(p_i \wedge x_{2^{a-i}}^{i-1}) \wedge flag_{i-1}$,
L3. $p_a = x_1^{a-1} \wedge flag_{a-1}$.
L4. Outputs $(p_1, \dots, p_a)$.
```

We now consider the correctness of algorithm L. We focus on the i-th iteration of step L2. In the case $p_i = 1$, since the number of 0-bits before the leftmost 1 in x^{i-1} is at least $2^{a-i} - 1$, we can show $(p_1 \cdots p_{i-1} 1 0^{a-i})_2 \le (p)_2 \le (p_1 \cdots p_{i-1} 1 1^{a-i})_2$. In particular, if $p_i = 1$ and $x_{2^{a-i}}^{i-1} = 1$ (which implies $flag_i = 0$), then the number of 0-bits before the leftmost 1 is equal to $2^{a-i} - 1$. Hence $(p)_2 = (p_1 \cdots p_{i-1} 1 0^{a-i})_2$, i.e., $p_{i+1}, \dots, p_a$ should be all 0's. This will be satisfied since $flag_i = 0$. In the case $p_i = 0$, the number of 0-bits before the leftmost 1 in x^{i-1} is at most $2^{a-i} - 2$. Thus $(p_1 \cdots p_{i-1} 0 0^{a-i})_2 \le (p)_2 \le (p_1 \cdots p_{i-1} 0 1^{a-i})_2$. Therefore algorithm L outputs p_i correctly.

We now estimate the size of a circuit. For the i-th iteration of the for loop at step L2, p_i can be computed by a circuit of size 2^{a-i} and depth $a - i + 2$ with one NOT gate. A sequence x_j^i can be obtained by a circuit of size $3 \cdot (2^{a-i} - 1) = 3 \cdot 2^{a-i} - 3$ with one NOT gate, and $flag_i$ can be computed by using three gates including one NOT gate. For each i, step L2 can be done by a circuit of size $4 \cdot 2^{a-i} + 1$ with three NOT gates and depth $a - i + 4$. We only need one AND gate at step L3. Therefore, algorithm L can be simulated by a circuit of size $\sum_{i=1}^{a-1}(4 \cdot 2^{a-i} + 1) + 1 = 4(2^{a-1} - 1) + (a-1) + 1 = O(n)$ with $3(a-1) = O(\log n)$ NOT gates and of depth $\sum_{i=1}^{a-1}(a - i + 4) + 1 = O(a^2) = O(\log^2 n)$. The construction of a circuit for Left_n^0 is similar to that for Left_n^1 and is omitted. □

Lemma 9. *For each $b \in \{0,1\}$, Decode_n^b can be computed by a circuit of size $O(n)$ and of depth $O(\log n)$ that contains $O(\log n)$ NOT gates.*

Proof. It is obvious that $\mathsf{Decode}_n^1(p) = \mathsf{Decode}_n(q)$ with $(q)_2 = (p)_2 - 1$, and $\mathsf{Decode}_n^0(p)$ is equal to the reverse of $\mathsf{Decode}_n(q')$ with $(q')_2 = n - (p)_2 + 1$. We can easily see that each of q and q' can be computed by a circuit of size $O(\log n)$ and of depth $O(\log n)$ with $O(\log n)$ NOT gates. Since it is well known that Decode_n has a linear size $O(\log n)$ depth monotone circuit [12], we can obtain a desired circuit for Decode_n^b. □

Proof. (of Theorem 4) As for the proof of Theorem 6, we first show an algorithm to compute Inv_n^k. Suppose that x is a k-tonic sequence starting with "0".

Algorithm I. This algorithm takes an n-bit binary sequence $x = (x_1, \ldots, x_n)$ as an input and outputs $z = (z_1, \ldots, z_n)$ where $z_i = \neg x_i$ if x is k-tonic.

```
I1. Let $x^0 = x$ and $z^0 = 1^n$.
I2. For $i = 1, \ldots, k$ do the following:
    If $i$ is odd then
      1. $s^i = \mathsf{Decode}_n^1(\mathsf{Left}_n^1(x^{i-1}))$,
      2. $x^i = \mathsf{Or}_n(x^{i-1}, s^i)$,
      3. $z^i = \mathsf{And}_n(z^{i-1}, s^i)$,
    else
      1. $s^i = \mathsf{Decode}_n^0(\mathsf{Left}_n^0(x^{i-1}))$,
      2. $x^i = \mathsf{And}_n(x^{i-1}, s^i)$,
      3. $z^i = \mathsf{Or}_n(z^{i-1}, s^i)$,
I3. Outputs $z = z^k$.
```

The correctness of the algorithm I can be verified as follows: For some non-negative integers $p_0, \ldots, p_k \geq 0$, we can write x^0 as

$$x^0 = 0^{p_0}1^{p_1}0^{p_2}1^{p_3}0^{p_4}\cdots 0^{p_k}.$$

Then we have

$$\begin{aligned}
s^1 &= 1^{p_0}0^{p_1}0^{p_2}0^{p_3}0^{p_4}\cdots 0^{p_k},\\
x^1 &= 1^{p_0}1^{p_1}0^{p_2}1^{p_3}0^{p_4}\cdots 0^{p_k},\\
z^1 &= 1^{p_0}0^{p_1}0^{p_2}0^{p_3}0^{p_4}\cdots 0^{p_k},\\
s^2 &= 0^{p_0}0^{p_1}1^{p_2}1^{p_3}1^{p_4}\cdots 1^{p_k},\\
x^2 &= 0^{p_0}0^{p_1}0^{p_2}1^{p_3}0^{p_4}\cdots 0^{p_k},\\
z^2 &= 1^{p_0}0^{p_1}1^{p_2}1^{p_3}1^{p_4}\cdots 1^{p_k}.
\end{aligned}$$

Note that x^1 is a $k-1$ tonic sequence starting with $1^{p_0+p_1}$ and x^2 is a $k-2$ tonic sequence starting with $0^{p_0+p_1+p_2}$. Similarly, we can show that x^i is a $k-i$ tonic sequence and that

$$z^i = 1^{p_0}0^{p_1}\cdots b^{p_i}b^{p_{i+1}}\cdots b^{p_k}.$$

Hence

$$z^k = 1^{p_0}0^{p_1}1^{p_2}0^{p_3}1^{p_4}\cdots 0^{p_k},$$

which is a desired output.

We now estimate the size of a circuit. For each iteration of the for loop at step I2, a sequence s^i can be computed by a circuit of size $O(n)$ and depth $O(\log^2 n)$ with $O(\log n)$ NOT gates, and sequences x^i and z^i can be computed by n gates and depth 1 without NOT gates. Hence the size and depth of an entire circuit are $O(kn)$ and $O(k\log^2 n)$, respectively. The total number of NOT gates is clearly $O(k\log n)$. □

We finally remark that if we can improve the depth of our circuit for Left_n^b to $O(\log n)$, then we will have a negation-limited k-tonic inverter of depth $O(k\log n)$ which gives a negative answer to Turán's problem for the case $k = O(1)$.

Acknowledgment

The authors would like to thank Eiji Takimoto for helpful discussions and encouragement. This work was supported in part by Grant-in-Aid for Scientific Research on Priority Areas "New Horizons in Computing" from MEXT of Japan.

References

1. M. Ajtai, J. Komós and E. Szemerédi, An $O(n \log n)$ Sorting Network, *Proc. 15th STOC*, pp. 1–9, 1983.
2. N. Alon and R.B. Boppana, The Monotone Circuit Complexity of Boolean Functions, *Combinatorica*, 7(1), pp. 1–22, 1987.
3. K. Amano and A. Maruoka, A Superpolynomial Lower Bound for a Circuit Computing the Clique Function with At Most $(1/6) \log \log n$ Negation Gates, *SIAM J. Comput.*, 35(1), pp. 201–216, 2005.
4. K. Amano, A. Maruoka and J. Tarui, On the Negation-Limited Circuit Complexity of Merging, *Discrete Applied Mathematics*, 126(1), pp. 3–8, 2003.
5. A.E. Andreev, On a Method for Obtaining Lower Bounds for the Complexity of Individual Monotone Functions, *Sov. Math. Dokl.*, 31(3), pp. 530–534, 1985.
6. R. Beals, T. Nishino and K. Tanaka, More on the Complexity of Negation-Limited Circuits, *Proc. 27th STOC*, pp. 585–595, 1995.
7. M.J. Fischer, The Complexity of Negation-Limited Network–A Brief Survey, *LNCS*, 33, pp. 71–82, 1974.
8. D. Harnik, R. Raz, Higher Lower Bounds on Monotone Size, *Proc. 32nd STOC*, pp. 378–387, 2000.
9. S. Jukna, On the Minimum Number of Negations Leading to Super-Polynomial Savings, *Inf. Process. Lett.*, 89(2), pp. 71–74, 2004.
10. E. A. Lamagna, The Complexity of Monotone Networks for Certain Bilinear Forms, Routing Problems, Sorting and Merging, *IEEE Trans. of Comput.*, 28(10), pp. 773–782, 1979.
11. A.A. Markov, On the Inversion Complexity of a System of Functions, *J. ACM*, 5, pp. 331–334, 1958.
12. D. E. Muller and F. P. Preparata, Bounds to Complexities of Networks for Sorting and Switching, *J. ACM*, 22, pp. 195–201, 1975.
13. A.A. Razborov, Lower Bounds on the Monotone Complexity of Some Boolean Functions, *Soviet Math. Dokl.*, 281, pp. 798–801, 1985.
14. S. C. Sung and K. Tanaka, An Exponential Gap with the Removal of One Negation Gates, *Inf. Process. Let.*, 82(3), pp. 155–157, 2002.
15. K. Tanaka and T. Nishino, On the Complexity of Negation-Limited Boolean Networks, *SIAM J. Comput.*, 27(5), pp. 1334–1347, 1998.
16. I. Wegener, *The Complexity of Boolean Functions*, Wiley-Teubner, 1987.