On the Computational Complexity of Optimal Sorting Network Verification

Ian Parberry[*]
Department of Computer Science
The Pennsylvania State University

Abstract

A *sorting network* is a combinational circuit for sorting, constructed from comparison-swap units. The depth of such a circuit is a measure of its running time. It is reasonable to hypothesize that only the fastest (that is, the shallowest) networks are likely to be fabricated. It is shown that the problem of verifying that a given sorting network actually sorts is Co-$\mathcal{NP}$ complete even for sorting networks of depth only $4\lceil\log n\rceil + O(1)$ greater than optimal. This is shallower than previous depth bounds by a factor of two.

1 Introduction

A *comparator network* is a combinational circuit constructed from comparison-swap units called *comparators*. A *sorting network* is a comparator network which sorts. The *size* of a comparator network is the number of comparators used. The *depth* is the number of layers of comparators, where each layer receives input only from the layers above it. Comparator networks can be fabricated relatively easily using VLSI techniques. It would be useful to be able to verify whether a given sorting network actually works. It is well known that in order to test whether a given n-input comparator network is a sorting network, it is sufficient to check that it sorts the $2^n - n - 1$ nonsorted zero-one inputs (which we will call bit-strings). This observation is called the *zero-one principle*.

Comparator networks which sort all but a single nonsorted bit-string are known. That is, for all nonsorted sequences of n bits x, there exists an n-input comparator network which sorts all bit-strings except x. These are called *single exception sorting networks*. Chung and Ravikumar [5] give a recursive construction of an n-input single exception sorting network of polynomial size and depth. They further deduce in [6] that the sorting network verification problem is Co-$\mathcal{NP}$ complete. Parberry [16] gave a non-recursive construction for a single exception sorting network of depth $D(n-1) + 2\lceil\log(n-1)\rceil + 2$, where $D(n)$ is the minimum depth of an n-input sorting network, and deduced, using the construction of Chung and Ravikumar [6], that the problem of verifying sorting networks of depth $2D(n) + 6\lceil\log n\rceil + O(1)$ is Co-$\mathcal{NP}$ complete. We will show that the sorting network verification problem remains Co-$\mathcal{NP}$ complete even for sorting networks of depth $D(n) + 4\lceil\log n\rceil + O(1)$.

The remainder of this paper is divided into six sections. The first section contains a more formal definition of a sorting network, and briefly describes some standard results. The second section contains a proof that a modified version of the satisfiability problem is $\mathcal{NP}$ complete. The third section contains a sketch of the reduction from that problem to the sorting network verification problem. The fourth section contains the details of the construction of an important component used in that reduction — a comparator network that sorts all except a specific set of inputs. The construction of this component uses the single exception sorting network of Parberry [16]. A slightly improved single exception sorting network is given in the fifth section of this paper. The sixth section contains details on how to reduce the depth of the construction to give the required result.

Let $\mathbf{N}$ denote the natural numbers, and $\mathbf{B}$ denote the Boolean set $\{0, 1\}$. Members of $\mathbf{B}^n$ (the set of n-tuples of bits) will be called *bit-strings*. We will use the standard regular-expression notation to describe

[*]Research supported by NSF Grant CCR-8801659. Author's current address: Department of Computer Sciences, P.O. Box 13886, University of North Texas, Denton, TX. 76203-3886, U.S.A. Electronic mail: `ian@dept.csci.unt.edu`.

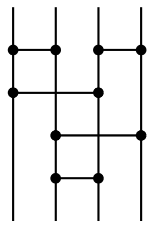

Figure 1: A 4-input sorting network of depth 3 and size 5.

certain sets of bit-strings, for example, $0^n 1^m$ denotes a single bit-string consisting of n ones followed by m zeros, and $(00 \cup 11)^n$ denotes the set of n pairs of bits, where each pair is either 00 or 11, that is,

$$\{x_1 y_1 \cdots x_n y_n \mid x_i = y_i \in \mathbf{B} \text{ for } 1 \leq i \leq n\}.$$

If A and B are sets, $A \backslash B$ denotes $\{x \mid x \in A, \text{ but } x \notin B\}$.

2 Sorting Networks

One of the early investigations into parallel sorting concerned the *Bose-Nelson sorting problem*, named by Floyd and Knuth [9], after Bose and Nelson [4]. The problem involves sorting n values by using a sequence of *oblivious* in-situ comparison-and-swap operations; that is, a sequence of comparisons between the ith and jth value, where i and j are independent of the values being sorted. The obliviousness property allows the following elegant hardware interpretation of the problem. Suppose that we are given a basic unit of hardware called a *comparator*. A comparator takes two values as input and outputs them in ascending order. A *comparator network* consists of n parallel *channels*, which can be thought of as wires carrying values, to which comparators are attached. The network is divided into a finite number of *levels*. Each level consists of one or more comparators. Each comparator is attached to two channels. At most one comparator is placed on any channel at each level. Channels, in our diagrams, will be drawn as vertical lines, and comparators as horizontal lines with heavy dots emphasizing the connection-points. Levels are numbered vertically from top-to-bottom, and channels are numbered horizontally from left-to-right. Level 0 will be used to denote the inputs.

Let C be a comparator network. We define the value carried by channel i of C at level j on input $x = (x_1, \ldots, x_n)$, written $V(C, x, i, j)$, as follows. $V(C, x, i, 0) = x_i$ and for $j > 0$:

- If there is a comparator between channels i and $k > i$ at level j, then

$$V(C, x, i, j) = \min(V(C, x, i, j-1), V(C, x, k, j-1)).$$

- If there is a comparator between channels i and $k < i$ at level j, then

$$V(C, x, i, j) = \max(V(C, x, i, j-1), V(C, x, k, j-1)).$$

- Otherwise, $V(C, x, i, j) = V(C, x, i, j-1)$.

The output of an n-input, d-level comparator network C on input x is

$$V(C, x) = (V(C, x, 1, d), V(C, x, 2, d), \ldots, V(C, x, n, d)).$$

If for all inputs x, $V(C, x)$ is in nondecreasing order from left to right, then C is called a *sorting network*.

For example, Figure 1 shows a 4-input sorting network. The comparators on the first level compare the values in pairs. The second layer of comparators determines the maximum and minimum values: the minimum of the two minima is the minimum overall, and the maximum of the two maxima is the maximum overall. The third layer puts the remaining two values into the correct order.

Each level of a comparator network consists of a set of independent comparisons which may be performed in parallel. The number of levels is thus a reasonable measure of parallel time. This is called the *depth* of the network. Another resource of interest is its *size*, which is defined to be the number of comparators used. We will call an n-input sorting network *optimal* if it is $\mathcal{P}$-uniform (that is, there is an algorithm which outputs a description of the sorting network, on input n, in time polynomial in n), and has depth $O(\log n)$. Optimal sorting networks have size $O(n \log n)$.

There have been a number of recent results on optimal sorting networks. For a survey of some of these results, see Parberry [13, 14]. Sorting networks of optimal depth are known for $n \leq 10$ (Parberry [15]) and with optimal size for $n \leq 8$ (Knuth [11]). For all practical purposes, the best sorting networks are constructed using the recursive technique of Batcher [3], which gives depth $O(\log^2 n)$ and size $O(n \log^2 n)$, although some small improvement in the lower-order terms of the size have been made by Drysdale [7] and Van Voorhis [19] in exchange for a large increase in depth. Ajtai, Komlós and Szemerédi [1, 2] have demonstrated that asymptotically optimal (logarithmic depth and log-linear size) sorting networks can be constructed; however their method remains impractical for reasonable values of n despite the efforts of Paterson [17], since the constant multiples involved are extremely large.

We will make use of two standard results. Firstly, if we allow channels to be permuted obliviously between each layer, then the model is the same, in the sense that we can remove the permutations in polynomial time in a manner that affects neither the size of the network, the depth of the network, nor its ability to sort. This result is due to Floyd and Knuth [8]. Therefore, the sorting network verification problems with and without permutations allowed are polynomial-time equivalent. The inclusion of such permutations will simplify the presentation of our results since it will allow us to physically group together logically related channels. The second standard result is the *zero-one* principle, which states that a comparator network is a sorting network *iff* it sorts all bit-strings. In the light of this result, we will throughout this paper consider only zero-one inputs. These results are discussed at great length in Knuth [11], and Parberry [13, 16, 14]. We assume that the reader is familiar with the techniques and terminology of the theory of $\mathcal{NP}$ completeness. The reader who is not should consult a standard reference, such as Garey and Johnson [10].

More formally, the sorting network verification problem can be stated as a decision problem as follows:

Sorting Network Verification (NONSORT)
INSTANCE: A comparator network C.
QUESTION: Is there an input which C does not sort?

The number of comparators in C is a reasonable measure of input size for any sorting network verification program. Furthermore, the number of inputs to C is a valid measure of input size for any program which verifies optimal sorting networks. Clearly NONSORT $\in \mathcal{NP}$, since if we are given a comparator network C and an input x, the output of C on x can be determined in time polynomial in the number of comparators in C. It remains to show that NONSORT is $\mathcal{NP}$ hard. This will imply that the original sorting network verification problem is Co-$\mathcal{NP}$ complete.

3 One-in-Three 3SAT

The following set-theoretic problem was shown to be $\mathcal{NP}$ complete by Schaefer [18].

One-in-Three 3SAT (T3SAT)
INSTANCE: Sets $S_1, \ldots, S_n$ with $|S_i| \leq 3$ for $1 \leq i \leq n$.
QUESTION: Does there exist a set S such that $|S \cap S_i| = 1$ for $1 \leq i \leq n$?

Garey and Johnson [10] list Schaefer's result in a slightly different form which more clearly illustrates that it is a restricted case of the Satisfiability Problem. Define a *clause* over a set V to be a subset of $V \times V \times V$. A set $S \subseteq V$ is said to *satisfy* a clause (v_1, v_2, v_3) iff $|\{i \mid v_i \in S\}| = 1$. If C is a list of clauses over V, then $S \subseteq V$ is said to be a *satisfying assignment* for C iff S satisfies all clauses in C. Intuitively, the elements of V are variables, and S is a set of variables to be assigned the value `true`. A clause represents a ternary Boolean function which is `true` iff exactly one of its inputs is `true`. A list of clauses represents the conjunction of a list of these functions.

Modified One-in-Three 3SAT (M3SAT)
INSTANCE: A list of clauses C over a set V.
QUESTION: Is there a satisfying assignment for C?

Lemma 3.1 *M3SAT is $\mathcal{NP}$ complete.*

PROOF: Obviously, M3SAT $\in \mathcal{NP}$. It is easy to show that T3SAT $\leq_m^p$ M3SAT. $\square$

We will find it useful to consider a restricted version of M3SAT.

Balanced One-in-Three 3SAT (B3SAT)
INSTANCE: A set of variables V and a list of clauses C over V in which every variable appears exactly three times.
QUESTION: Is there a satisfying assignment for C?

Note that each instance of B3SAT with n clauses must have n variables. (Since there are n clauses, there are $3n$ instances of variables. Since every variable has exactly 3 instances, there must be n variables.) Also, every *satisfiable* instance of B3SAT with n clauses must have n divisible by 3. (Let V be a variable-set, and $C = (C_1, \ldots, C_n)$ be a list of clauses over V. Suppose $S \subseteq V$, where $|S| = m$. Then C contains $3m$ instances of variables that are members of S. But if S is a satisfying assignment, since there are n clauses, each of which must contain exactly one instance of a variable in S, there must be n instances of variables that are members of S. Therefore $n = 3m$.)

Theorem 3.2 *B3SAT is $\mathcal{NP}$ complete.*

PROOF: Clearly B3SAT $\in \mathcal{NP}$. By Lemma 3.1, it suffices to show that M3SAT $\leq_m^p$ B3SAT.

Suppose $C_1, \ldots, C_n$ is an instance of M3SAT over variable-set V. The corresponding instance of B3SAT is constructed as follows. For every clause C_i, there are three new clauses $(a_{i,1}, a_{i,2}, a_{i,3})$, $(b_{i,1}, b_{i,2}, b_{i,3})$, $(c_{i,1}, c_{i,2}, c_{i,3})$ called *structural enforcers.* For each variable $v \in V$, let

$$X_v = \{a_{i,j}, b_{i,j}, c_{i,j} \mid v \text{ is the } j\text{th variable in } C_i\}.$$

Suppose

$$X_v = \{a_{i_1,j_1}, b_{i_1,j_1}, c_{i_1,j_1}, \ldots, a_{i_m,j_m}, b_{i_m,j_m}, c_{i_m,j_m}\}$$

for some $m \leq n$. Define

$$I_v = \{(a_{i_k,j_k}, b_{i_k,j_k}, c_{i_k,j_k}) \mid 1 \leq k \leq m\}.$$

If $k = 1$, define $E_v = I_v$, otherwise define

$$\begin{aligned} E_v \quad = \quad & \{(b_{i_1,j_1}, c_{i_1,j_1}, a_{i_2,j_2}), (b_{i_2,j_2}, c_{i_2,j_2}, a_{i_3,j_3}), \\ & \ldots, (a_{i_{m-1},j_{m-1}}, b_{i_{m-1},j_{m-1}}, c_{i_m,j_m}), (a_{i_m,j_m}, b_{i_m,j_m}, a_{i_1,j_1})\} \end{aligned}$$

For each $(p, q, r) \in I_v$ and each $(p, q, r) \in E_v$ there are three clauses (p, x, y), (q, x, y), and (r, x, y), called *equality enforcers*, where x and y are new variables not previously used. It is clear that this transformation can be computed in time polynomial in n.

The new instance of M3SAT consisting of the structural enforcers and the equality enforcers is actually an instance of B3SAT, that is, that every variable appears in exactly three clauses. Each of the $a_{i,j}$, $b_{i,j}$, and $c_{i,j}$ variables occurs in exactly one structural enforcer, and two equality enforcers (one corresponding to an element of I_v, and one corresponding to an element of E_v, for some variable v). The extra variables in the equality enforcers also appear in exactly three clauses, by construction.

We claim that $C_1, \ldots, C_n$ is satisfiable iff there is a satisfying assignment for the structural enforcers and the equality enforcers. Clearly if S is a satisfying assignment for $C_1, \ldots, C_n$, then $\cup_{v \in S} X_v$ satisfies every structural enforcer and every equality enforcer. Conversely, suppose that S satisfies the structural enforcers and equality enforcers. Since the equality enforcers corresponding to members of I_v are satisfied by S, then for all variables $v \in V$, either $X_v \cap S = \{\}$ or $X_v \subseteq S$. Thus if we set $T = \{v \mid X_v \subseteq S\}$, then T satisfies $C_1, \ldots, C_n$. $\square$

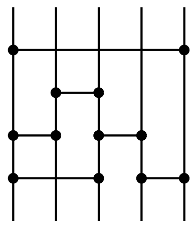

Figure 2: A variable component.

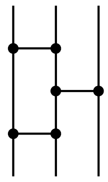

Figure 3: A clause component.

4 The Reduction

In order to show that NONSORT is $\mathcal{NP}$ complete, it is sufficient to show that B3SAT $\leq_m^p$ NONSORT. Suppose we are given an instance of B3SAT, that is, a list of clauses $C = (C_1, \ldots, C_n)$ over a set of variables $V = \{v_1, \ldots, v_n\}$ such that every variable in V appears exactly three times in C. We will construct a comparator network with $5n$ inputs. An input $x = (x_1, \ldots, x_{5n})$ to the comparator network is said to *correspond to assignment* S for C iff for all $1 \leq i \leq n$, $x \in (0^5 \cup 1^5)^n$, and $v_i \in S$ iff $x_{5i-4} = 1$. Our comparator network will sort only inputs that do not correspond to satisfying assignments for C, that is, inputs that do not correspond to any assignment, and inputs that correspond to nonsatisfying assignments. Therefore, it will be a sorting network iff C is not satisfiable.

For each variable $v \in V$, we have a *variable component*, consisting of five channels and six comparators, of depth three (see Figure 2). For each clause C_i we have a *clause component*, consisting of three channels and three comparators, of depth three (see Figure 3). These components are connected as follows.

The $5n$ inputs are divided into quintuples and put into n variable components, one for each variable. The center three outputs of each variable component are put into the inputs of the appropriate clause components. Specifically, suppose $C_i = (v_{i,1}, v_{i,2}, v_{i,3})$, for $1 \leq i \leq n$ where $v_{i,1}, v_{i,2}, v_{i,3} \in V$. Let $c(i, j)$ denote the number of previous occurrences of $v_{i,j}$ in $C_1, \ldots, C_i$, that is, $c(i, j) = |\{v_{k,l} \mid k < i \text{ or } (k = i \text{ and } l < j)\}| + 1$, for $1 \leq i \leq n$, $1 \leq j \leq 3$. Note that $0 \leq c(i, j) \leq 2$. Then for each clause $C_i = (v_{i,1}, v_{i,2}, v_{i,3})$, put the $(c(i, k) + 2)$th output of the variable component corresponding to $v_{i,k}$ into the kth input of the clause component corresponding to C_i, for $1 \leq k \leq 3$. The first two outputs of the clause components are routed to the far right, and the last output of the clause components is routed to the far left. All of the channels are then put into a special component called a *selector*, which will sort every input except those which correspond to satisfying assignments.

The centre three outputs of a variable component corresponding to variable $v \in V$ are copies of variable v which are to be used in the clauses in which v appears. It can be demonstrated (by case analysis of the 32 different input strings) that if the first and last outputs of a variable component are zero, then all outputs are zero, and if the first and last outputs of a variable component are one, then all outputs are one. Thus the centre three outputs of a variable component carry a valid truth assignment for the variable v if the first and last output of the variable component are identical. The output of a clause component is 001 iff exactly one of its input channels carries a one (since each clause component is actually a three-input sorting network).

Suppose the input to the comparator network corresponds to a satisfying assignment S. Since there is exactly one variable from each clause in S, the first n inputs to the selector will be one, and the last $2n$ inputs to the selector will be zero. The remaining $2n$ inputs will consist of n pairs, each of which is either

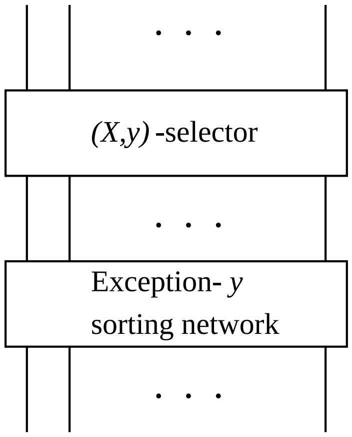

Figure 4: An exception-X sorting network.

00 or 11. Since variable appears in exactly three clauses, S must contain $n/3$ variables. Therefore $2n/3$ pairs will be 00, and $n/3$ pairs will be 11. That is, the input to the selector will be a member of the set $1^n(\mathbf{B}^{2n}_{2n/3} \cap (00 \cup 11)^n)0^{2n}$, where $\mathbf{B}^n_k = \{x \in \mathbf{B}^n \mid x \text{ has exactly } k \text{ ones}\}$. Conversely, if the input to the comparator network does not correspond to satisfying assignment, then it is clear that the input to the selector will not be of this form. The above reduction can be carried out in time polynomial in n provided there exist $\mathcal{P}$-uniform selectors. The construction of the selector is the subject of the next section.

5 Selective Sorting Networks

The *selector* used in the previous section was a comparator network which selectively sorts all bit-strings except for members of a certain, slightly obscure set. We use the generic term *selective sorting network* for such a comparator network. If the comparator network sorts all members of $\mathbf{B}^n \backslash X$, than it is called an *exception-X sorting network*.

Comparator networks which sort all but a single bit-string are known. That is, for all nonsorted $x \in \mathbf{B}^n$, there exists an n-input comparator network which sorts all members of $\mathbf{B}^n$ except x. These are called *single exception sorting networks*. Chung and Ravikumar [5] give a recursive construction of an n-input single exception sorting network of polynomial size and depth. Parberry [16] gives a non-recursive construction for a single exception sorting network of depth $D(n-1) + 2\lceil\log(n-1)\rceil + 2$, where $D(n)$ is the minimum depth of an optimal n-input sorting network. We wish to find a particular selective sorting network with $5n$ inputs, where n is a multiple of three, and an exception set of size

$$\binom{n}{n/3}.$$

If $X \subseteq \mathbf{B}^n$, and $y \in \mathbf{B}^n$ is nonsorted, a comparator network C is an (X, y)-*selector* iff for all $x \in \mathbf{B}^n$, the output of C on input x is y iff $x \in X$.

Theorem 5.1 *If $X \subseteq \mathbf{B}^n$, and there exists an (X, y)-selector of depth $D_X(n)$, then there exists an exception-X sorting network of depth $D_X(n) + D(n) + 2\lceil\log n\rceil + 2$.*

PROOF: Suppose $X \subseteq \mathbf{B}^n$, and there exists a comparator network C of depth $D_X(n)$ and nonsorted $y \in \mathbf{B}^n$ such that for all $x \in \mathbf{B}^n$, the output of C on input x is y iff $x \in X$. Then an exception-X sorting network is constructed by composing C and an exception-y sorting network as in Figure 4. The total depth is bounded above by the depth of C plus the depth of the exception-y sorting network. Parberry [16] gives a construction for exception-y sorting networks for all nonsorted y of depth $D(n) + 2\lceil\log n\rceil + 2$. □

The exception-set from the previous section is the set of bit-strings of the following form: n ones, followed by n pairs of bits, $2n/3$ of which are 00 and $n/3$ of which are 11, followed by $2n$ zeros. That is,

$$X = 1^n(\mathbf{B}^{2n}_{2n/3} \cap (00 \cup 11)^n)0^{2n}.$$

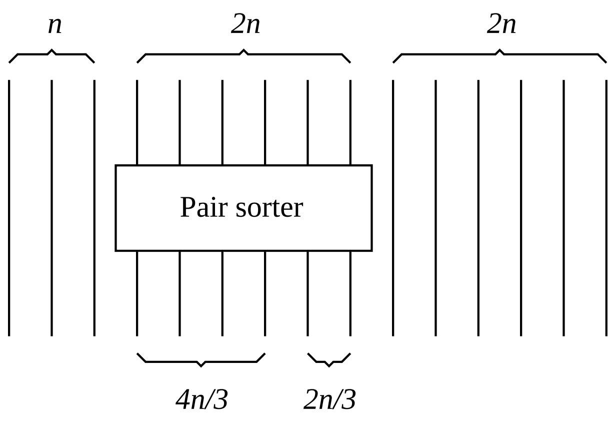

Figure 5: An X-selector.

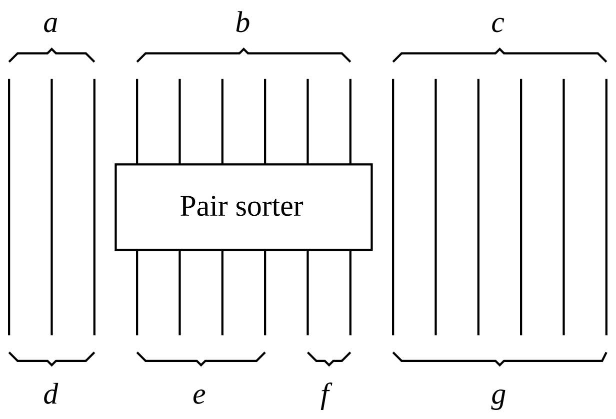

Figure 6: An X-selector labelled for the proof of Theorem 5.2.

Theorem 5.2 *Suppose $n \in \mathbf{N}$, and $X = 1^n(\mathbf{B}^{2n}_{2n/3} \cap (00 \cup 11)^n)0^{2n}$. There exists an exception-X sorting network of depth $D(n) + D(5n) + 2\lceil \log(5n) \rceil + 3$.*

PROOF: Let $X = 1^n(\mathbf{B}^{2n}_{2n/3} \cap (00 \cup 11)^n)0^{2n} \subseteq \mathbf{B}^{5n}_{5n/3}$. By Theorem 5.1, we can build an exception-X sorting network from an (X, y)-selector. An (X, y)-selector with $y = 1^n 0^{4n/3} 1^{2n/3} 0^{2n}$ can be constructed as in Figure 5, by leaving the first n and the last $2n$ channels alone, and placing a $2n$-input *pair sorter* on the remaining $2n$ channels. A *pair sorter* is a comparator network which has an even number of inputs. The inputs are treated as bit-pairs. Each bit-pair is sorted, and the sorted bit-pairs are then sorted into lexicographic order, that is, the output of the pair sorter is a member of $(00)^*(01)^*(11)^*$. We will return to the construction of the pair sorter later in this proof.

Suppose the input to the network shown in Figure 5 is $abc \in \mathbf{B}^{5n}_{5n/3}$, where $a \in \mathbf{B}^n$, $b, c \in \mathbf{B}^{2n}$, and its output is $defg \in \mathbf{B}^{5n}_{5n/3}$, where $d \in \mathbf{B}^n$, $e \in \mathbf{B}^{4n/3}$, $f \in \mathbf{B}^{2n/3}$, and $g \in \mathbf{B}^{2n}$ (see Figure 6). We claim that $defg = 1^n 0^{4n/3} 1^{2n/3} 0^{2n}$ iff $abc \in X$. Suppose $abc \in X$. then $b \in \mathbf{B}^{2n}_{2n/3} \cap (00 \cup 11)^n$. Since $b \in (00 \cup 11)^n$, the output of the pair sorter, ef, is a sorted sequence of bits. Since $b \in \mathbf{B}^{3n}_{2n/3}$, $ef = 0^{4n/3} 1^{2n/3}$. Therefore, since $d = a$ and $g = c$, $defg = 1^n 0^{4n/3} 1^{2n/3} 0^{2n}$, as claimed. Conversely, suppose that $abc \notin X$. We claim that $defg \neq 1^n 0^{4n/3} 1^{2n/3} 0^{2n}$, that is, either $d \neq 1^n$, $e \neq 0^{4n/3}$, $f \neq 1^{2n/3}$, or $g \neq 0^{2n}$. Since $abc \notin X$, either $a \neq 1^n$, (in which case there is a zero in d), $c \neq 0^{2n}$, (in which case there is a one in g), or $b \notin \mathbf{B}^{2n}_{2n/3} \cap (00 \cup 11)^n$. In the latter case, suppose $b \in \mathbf{B}^{2n}_m$. If $m = 2n/3$, then $b \notin (00 \cup 11)^n$, and so there is a one in e. If $m < 2n/3$, then there is a zero in f. If $m > 2n/3$, then there is a one in e. In all cases, $defg \neq 1^n 0^{4n/3} 1^{2n/3} 0^{2n}$, as claimed.

It is clear that the depth of our $5n$-input (X, y)-selector is equal to the depth of a $2n$-input pair sorter.

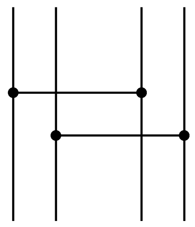

Figure 7: A pair comparator.

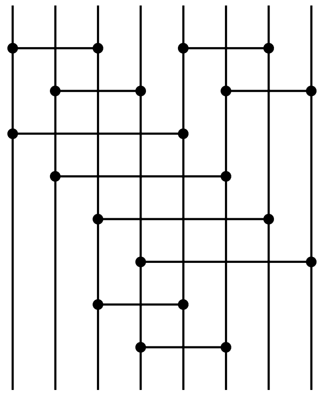

Figure 8: A 4-pair sorter constructed from Figure 1.

This pair-sorter is constructed as follows. The pairs are sorted with a single layer of comparators, one per pair. A pair of pairs can be sorted into lexicographic order by comparing the first element of the first pair with the first element of the second pair, and simultaneously comparing the second element of the first pair with the second element of the second pair (see Figure 7). Therefore n pairs can be sorted into lexicographic using a comparator network obtained from an n-input sorting network by doubling all the channels, and replacing every comparator with a pair-comparator (for example, see Figure 8). Thus the depth of the pair sorter is $D(n) + 1$. Therefore by Theorem 5.1, the depth of our $15n$-input exception-X sorting network is $D(n) + D(5n) + 2\lceil\log(5n)\rceil + 3$. $\square$

Theorem 5.3 *NONSORT is $\mathcal{NP}$ complete even for n-input sorting networks of depth* $2D(n)+2\lceil\log n\rceil+9$.

Proof: The reduction is as described in Section 4, using the selector from Theorem 5.2. The depth of an n-input comparator network constructed using this technique is bounded above by 3 for the variable components, plus 3 for the clause components, plus $2D(n) + 2\lceil\log n\rceil + 3$ for the selector, a total of $2D(n) + 2\lceil\log n\rceil + 9$. $\square$

6 Improved Single Exception Sorting Networks

In the construction of the selector in the previous section, we used the single exception sorting network from Parberry [16], which has depth $D(n-1) + 2\lceil\log(n-1)\rceil + 2$. It is possible to improve that result by a small constant. Suppose $n \in \mathbf{N}$, and $1 \leq k < n$. A better single exception sorting network with exception 1^k0^{n-k} is constructed as follows (see Figure 9). In [16], a single exception sorting network with this exception is called a *canonical k-ones single exception sorting network.* The first k inputs are put into a *min network.* The leftmost output of this network is the minimum of its inputs. The last $k-1$ outputs of this network, and the remaining $n-k$ channels are sorted together. The leftmost channel, and the leftmost $n-k-1$ outputs of the sorting network are put into an *insertion network.* This network takes

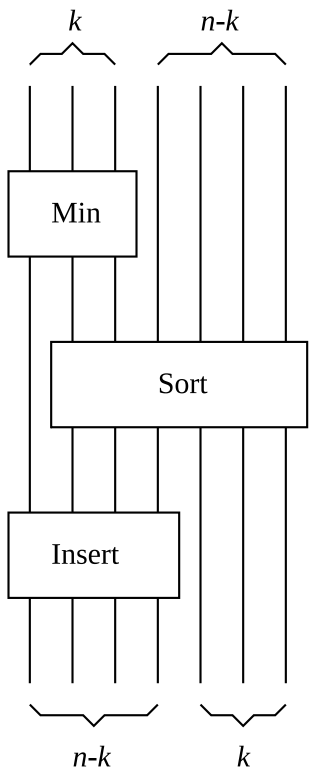

Figure 9: A single exception sorting network.

as input a single value followed by a sorted sequence, and it inserts the new value into the correct place in the sequence. It is straightforward to recursively construct n-input min and insertion networks of depth $\lceil \log n \rceil$.

It is easy to see that this construction gives a single exception sorting network. Suppose the input to the network is $1^k 0^{n-k}$. Then the leftmost output of the min network is 1, and the output of the sorting network is $0^{n-k}1^{k-1}$, and hence the output of the insertion network is $0^{n-k-1}101^{k-1}$, which is not sorted. Now suppose the input to the network is not $1^k 0^{n-k}$. In particular, suppose it is $ab \neq 1^k 0^{n-k}$, where $a \in \mathbf{B}^k$, $b \in \mathbf{B}^{n-k}$. Then either $a \neq 1^k$ or $b \neq 0^{n-k}$. In the former case, the leftmost output of the min network is 0, hence the values on the channels immediately after the sorting network are sorted, and they remain sorted through the rest of the network. If $a = 1^k$ and $b \neq 0^{n-k}$, then b contains at least one 1, and so the insertion network carries the 1 from the leftmost channel into the correct place.

Theorem 6.1 *For all $n > 1$ and all nonsorted bit-strings x, there exists an n-input comparator network of depth $D(n-1) + 2\lceil \log n \rceil - 1$. which sorts all bit-strings except x.*

PROOF: If $D(n)$ is the depth of the optimal n-input sorting network, then the depth of the canonical k-ones single exception sorting network shown in Figure 9 is

$$D(n-1) + \lceil \log \lceil n/2 \rceil \rceil + \lceil \log \lfloor n/2 \rfloor \rceil \leq D(n-1) + 2\lceil \log n \rceil - 1.$$

If x is an arbitrary nonsorted bit-string with k ones, then a comparator network can be constructed from the canonical k-ones single exception sorting network using the technique of Theorem 7 of [16]. □

This new construction can be used to improve slightly on the results in [16], and to improve slightly on the bound in Theorem 5.3.

Theorem 6.2 *NONSORT is $\mathcal{NP}$ complete even for n-input sorting networks of depth $2D(n)+2\lceil \log n \rceil + 6$.*

PROOF: Use Theorem 6.1 in the construction of Theorem 5.3. □

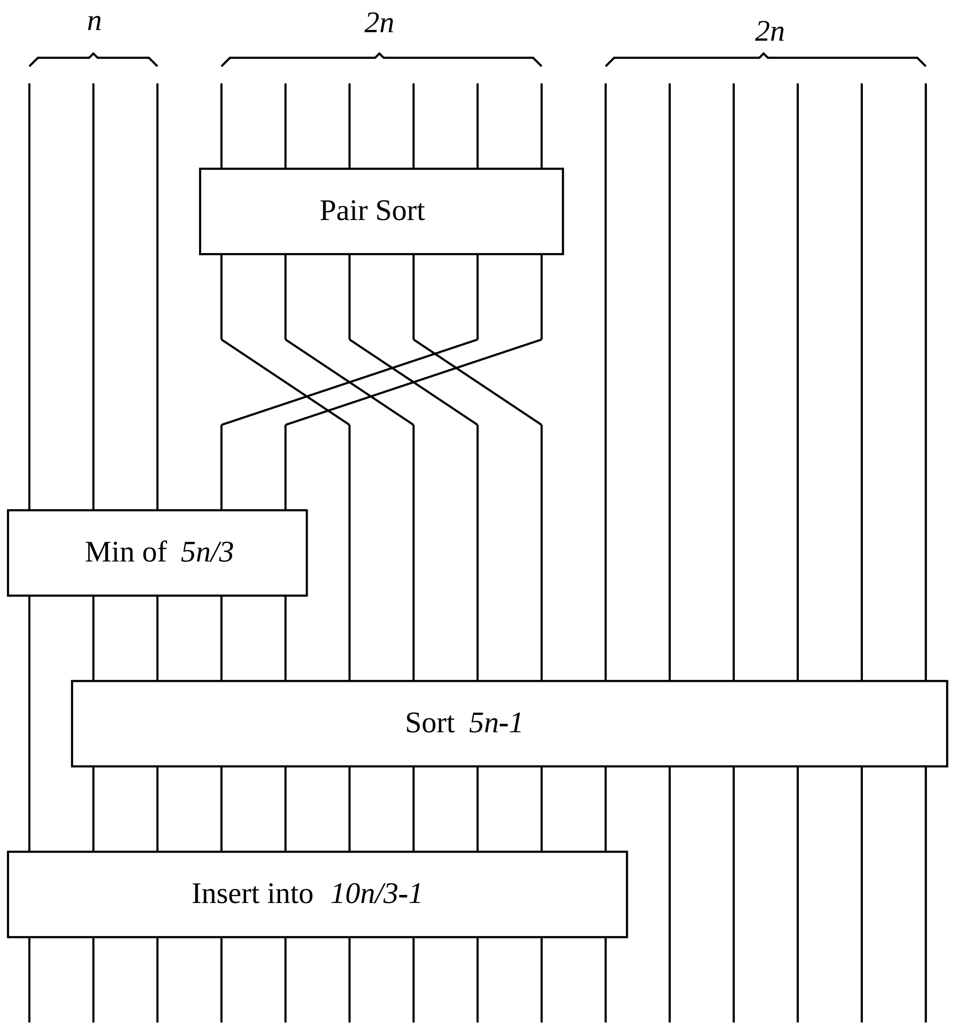

Figure 10: Details of the selector construction.

7 Improved Selectors

The selective sorting network described in Section 5 was developed using general techniques which will work for any exception set of the appropriate form. However, the exception set that appears in the reduction of Section 4 has additional special properties which allow a reduction in the depth bound.

Let us re-draw the selector using the canonical single exception sorting network from Section 6 (see Figure 10). The selector consists of the pair sorter of depth $D(n)$, a sorting network of depth $D(5n)$, and a min network and an insertion network of combined depth $2\lceil\log(5n)\rceil - 1$. Thus the total depth of the $5n$-input selector is $2D(5n) + 2\lceil\log(5n)\rceil - 1$. We route the last $2n/3$ outputs from the pair sorter to the right of the first $4n/3$ outputs, so that the exception becomes $1^{5n/3}0^{10n/3}$ instead of $1^n0^{4n/3}1^{2n/3}0^{2n}$.

However, we have not used the fact that the output of the pair sorter is sorted in pairs. Thus there is no need for the sorting network in the part of the selector corresponding to the single exception sorting network. If we sort the first n and last $2n$ values in parallel with the pair sorter, then all we need to do is merge the sorted pairs in two groups after the pair sorter, and merge these with the outputs of the sorters (see Figure 11).

Parberry [12] gives a construction for an n-input pair merger (which is called an *alternating merging network* in that reference), of depth $\lceil\log n\rceil$. Batcher [3] gives a construction for an n-input merging network of depth $\lceil\log n\rceil + 1$. Therefore the new $5n$-input selector has depth bounded above by $D(2n)$ for the sorters, plus $\lceil\log(4n/3)\rceil \leq \lceil\log(5n)\rceil - 1$ for the pair mergers, plus $\lceil\log(5n)\rceil + 1$ for each of two mergers, plus $\lceil\log(5n)\rceil$ for the inserter, giving a total of $D(2n) + 4\lceil\log(5n)\rceil + 1$.

Therefore, we obtain the main result of this paper:

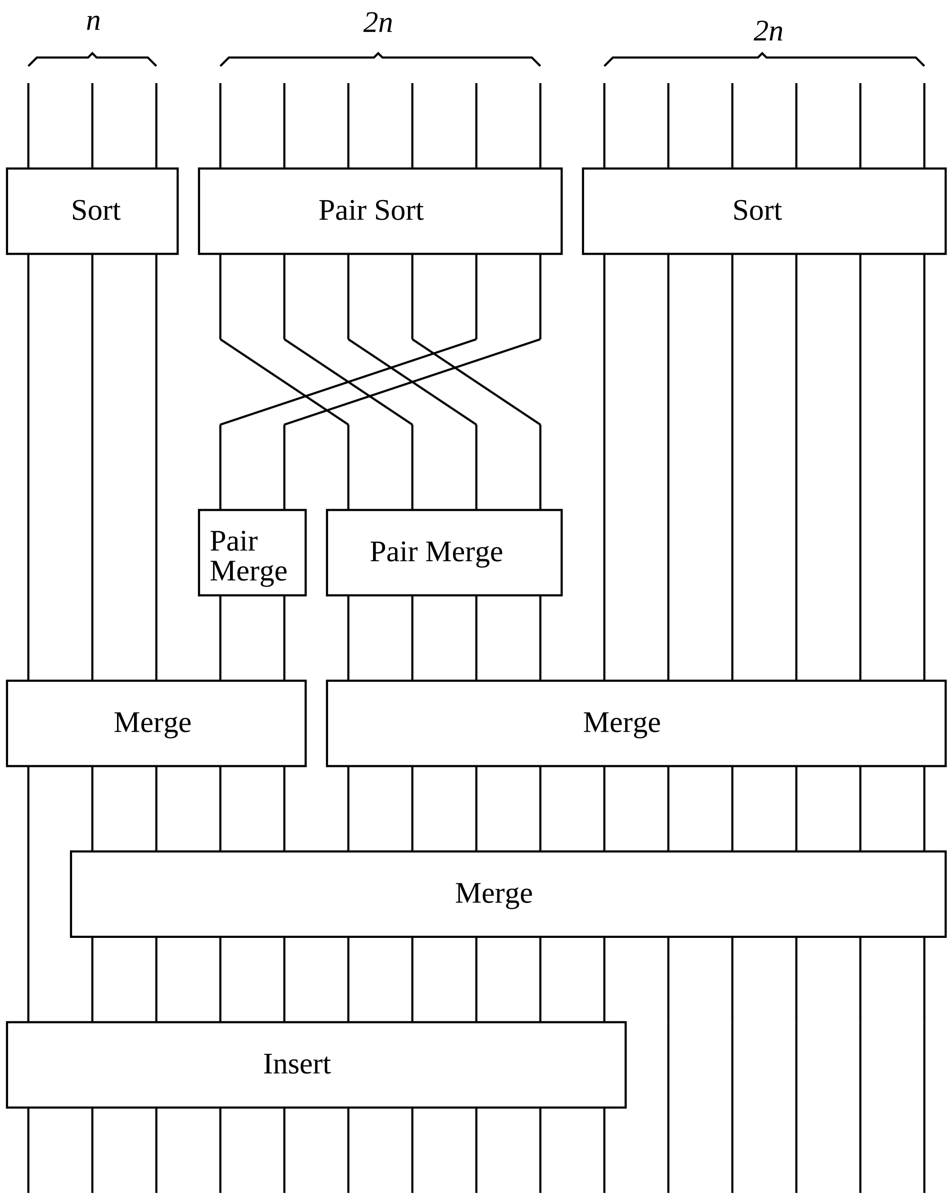

Figure 11: Details of the modified selector construction.

Theorem 7.1 *NONSORT is $\mathcal{NP}$ complete even for n-input sorting networks of depth $D(n)+4\lceil \log n \rceil + 7$.*

Proof: The proof is similar to that of Theorem 5.3, substituting the more efficient selector described in this section. The depth in this case is bounded above by 3 for the variable components, 3 for the clause components, plus $D(2n/5) + 4\lceil \log n \rceil + 1$ for the selector. □

8 Conclusion and Open Problems

We have shown that sorting network verification is intractable even for sorting networks of depth $D(n) + 4\lceil \log n \rceil + 7$, where $D(n)$ is the depth of an optimal n-input sorting network. This is smaller by a factor of two than previous results. Our result is fairly strong, given the current state of knowledge about $D(n)$, which is that $\lceil \log n \rceil \leq D(n) < 6100\lceil \log n \rceil$ (the lower bound is obvious, and the upper bound is from Paterson [17]). We conjecture that sorting network verification remains intractable even for the shallowest sorting networks, that is, sorting networks of depth $D(n)$.

It is also interesting to consider the depth of single exception sorting networks, since their existence implies an exponential time lower bound for deterministic and probabilistic verification algorithms based on the zero-one principle (Parberry [16]). If $S(n)$ is the minimum depth of an n-input single exception sorting network, we know that $D(n) - 1 \leq S(n) \leq D(n) + 2\lceil \log n \rceil - 1$, where $D(n)$ is the minimum depth of an n-input sorting network. We conjecture that $S(n) = D(n)$.

It should be noted that it is an open problem as to whether the result of Theorem 7.1 is better than that of Theorem 6.2. The former is better than the latter iff $D(n) \geq 2\lceil \log n \rceil - 1$, which is the case for large enough n (Yao [20]).

References

[1] M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n. \log n)$ sorting network. *Proc. 15th Ann. ACM Symp. on Theory of Computing*, pages 1–9, April 1983.

[2] M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3:1–48, 1983.

[3] K. E. Batcher. Sorting networks and their applications. In *Proc. AFIPS Spring Joint Computer Conference*, volume 32, pages 307–314, April 1968.

[4] R. C. Bose and R. J. Nelson. A sorting problem. *J. Assoc. Comput. Mach.*, 9:282–296, 1962.

[5] M. Chung and B. Ravikumar. On the size of test sets for sorting and related problems. In *Proc. 1987 International Conference on Parallel Processing.* Penn State Press, August 1987.

[6] M. J. Chung and B. Ravikumar. Strong nondeterministic Turing reduction — a technique for proving intractability. *J. Comput. System Sci.*, 39(1):2–20, 1989.

[7] R. L. Drysdale. Sorting networks which generalize batcher's odd-even merge. Honors Paper, Knox College, May 1973.

[8] R. W. Floyd and D. E. Knuth. Improved constructions for the Bose-Nelson sorting problem (preliminary report). *Notices of the AMS*, 14:283, 1967.

[9] R. W. Floyd and D. E. Knuth. The Bose-Nelson sorting problem. In J. N. Srivastava, editor, *A Survey of Combinatorial Theory.* North-Holland, 1973.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[11] D. E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming.* Addison-Wesley, 1973.

[12] I. Parberry. The alternating sorting network. Technical Report CS-87-26, Dept. of Computer Science, Penn. State Univ., September 1987.

[13] I. Parberry. *Parallel Complexity Theory.* Research Notes in Theoretical Computer Science. Pitman Publishing, London, 1987.

[14] I. Parberry. Sorting networks. Technical Report CS-88-08, Dept. of Computer Science, Penn. State Univ., March 1988.

[15] I. Parberry. A computer-assisted optimal depth lower bound for sorting networks with nine inputs. In *Proceedings of Supercomputing '89*, pages 152–161, 1989.

[16] I. Parberry. Single-exception sorting networks and the computational complexity of optimal sorting network verification. *Mathematical Systems Theory*, 23:81–93, 1990.

[17] M. S. Paterson. Improved sorting networks with $O(\log n)$ depth. *Algorithmica*, 5(4):75–92, 1990.

[18] T. J. Schaefer. The complexity of satisfiability problems. In *Proc. 10th Annual ACM Symposium on Theory of Computing*, pages 216–226. Association for Computing Machinery, 1978.

[19] D. C. Van Voorhis. An economical construction for sorting networks. In *Proc. AFIPS National Computer Conference*, volume 43, pages 921–926, 1974.

[20] A. Yao. Bounds on selection networks. *SIAM Journal on Computing*, 9, 1980.