Lower Bounds for (Non-Monotone) Comparator Circuits

Anna Gál
University of Texas at Austin, Austin TX, United States of America
panni@cs.utexas.edu

Robert Robere
Institute for Advanced Study, Princeton NJ, United States of America
rrobere@ias.edu

Abstract

Comparator circuits are a natural circuit model for studying the concept of *bounded fan-out* computations, which intuitively corresponds to whether or not a computational model can make "copies" of intermediate computational steps. Comparator circuits are believed to be weaker than general Boolean circuits, but they can simulate Branching Programs and Boolean formulas. In this paper we prove the first superlinear lower bounds in the general (non-monotone) version of this model for an explicitly defined function. More precisely, we prove that the n-bit Element Distinctness function requires $\Omega((n/\log n)^{3/2})$ size comparator circuits.

2012 ACM Subject Classification Theory of computation → Circuit complexity

Keywords and phrases comparator circuits, circuit complexity, Nechiporuk, lower bounds

Digital Object Identifier 10.4230/LIPIcs.ITCS.2020.58

Funding *Robert Robere*: Funding provided by an NSERC Postdoctoral Fellowship and the Charles Simonyi Endowment.

Acknowledgements Some of this work was done while the authors were visiting the Simons Institute for the Theory of Computing in Berkeley, and while R.R. was at DIMACS.

1 Introduction

One of the central open problems in circuit complexity is to prove superlinear lower bounds on the size of general Boolean circuits for explicitly defined functions. While theorists have been outstandingly successful at analyzing some restricted circuit models – for instance, exponential lower bounds are known when the circuit is monotone [22], or bounded-depth [11] – the progress remains modest for less restricted classes of circuits:

1. The best lower bounds for the *Boolean formula size* of explicit functions over n variables are of the form $\Omega(n^{3-o(1)})$ [12, 18, 26, 5, 7, 19, 27, 4, 8]. The first such bound is due to Håstad [12] and the current largest bound (improving lower order terms) is due to Tal [27].
2. The best lower bounds for *branching programs*, which capture small-space computation and can simulate formulas, are $\Omega((n/\log n)^2)$ for the deterministic model by Nechiporuk [21] and $\Omega(n^{3/2}/\log n)$ for the non-deterministic- and parity- models by Pudlák[1] [15], and Karchmer and Wigderson [15], respectively.
3. The best lower bounds for *span programs* over $GF(2)$, which can simulate all of the above models, is $\Omega(n^{3/2}/\log n)$ by Beimel, Gál, and Paterson [3], following the initial $\Omega(n \log\log\log^* n)$ lower bounds by Karchmer and Wigderson [15].
4. Finally (and, perhaps, most notoriously) the best lower bounds for *general* Boolean circuits are $5n - o(n)$, due to Iwama and Morizumi [13].

[1] Pudlák's result is unpublished, and referenced by Karchmer and Wigderson in their paper [15].

Improving any of these lower bounds is an outstanding open problem; however, the problem of proving *any* superlinear lower bound on general Boolean circuits is perhaps the most interesting. Span programs form a remarkable model here, since they can simulate all models above for which we have superlinear lower bounds – this means that improving the lower bounds for span programs is one of the outstanding open question at the "frontier" of current techniques for proving lower bounds in general non-monotone circuit models.

In this paper we identify a new problem at the frontier: the complexity of non-monotone *comparator circuits*. A comparator circuit is a circuit composed purely of *comparator gates*, each of which maps a pair of bits (x, y) to $(x \wedge y, x \vee y)$. It is known that, just like span programs, comparator circuits can efficiently simulate non-deterministic branching programs [6], and thus lower bounds for comparator circuits imply lower bounds on branching programs and formulas. However, like branching programs, comparator circuits are known to be stronger than boolean formulas, as it is easy to construct comparator circuits computing Parity in $O(n)$ size (unlike formulas, which require $\Omega(n^2)$ size [16]). Similarly, like span programs, comparator circuits are believed to be weaker than general non-monotone circuits [6, 24]. However, it appears that comparator circuits may be *incomparable* to, or possibly even stronger than span programs: the class of functions computable by polynomial-size comparator circuits is conjectured to be incomparable to NC [25], and this conjecture is supported by oracle separations [6]. In contrast, the class of functions computable by polynomial-size span programs over finite fields is contained in NC^2. Furthermore, while we already had superlinear lower bounds for span programs [3, 15], there have been *no* superlinear lower bounds known for the comparator circuit size of any explicit function.

Continuing the line of work proving superlinear lower bounds for formulas, branching programs, and span programs, we prove the *first* superlinear lower bounds on the size of comparator circuits computing an explicit Boolean function. Let $n = 2m \log m$, and let $ED_n : \{0, 1\}^n \to \{0, 1\}$ be the n-bit *Element Distinctness* function, which takes m integers in the range $\{1, 2, \ldots, m^2\}$ as input (each encoded as $2 \log m$ bits) and outputs 1 iff all of the integers are distinct. Our main result is the following:

▶ **Theorem 1.** *The size of any comparator circuit computing the n-bit Element Distinctness function ED_n is at least $\Omega((n/\log n)^{3/2})$.*

In fact, we prove the stronger statement, that the number of *wires* of any comparator circuit computing ED_n is at least $\Omega((n/\log n)^{3/2})$. (See Section 2 for definitions.)

The above result provides a new and natural class of circuits for which we can obtain superlinear size lower bounds, and thus represents progress towards the goal of proving superlinear lower bounds for general non-monotone circuits. We believe that the techniques we employ are also interesting. The current best $\Omega(n^{3-o(1)})$ lower bounds for formulas follow by restriction based techniques, which we do *not* use for our lower bounds. In fact, as we discuss in Section 5, there seem to be obstacles to obtaining superlinear lower bounds for comparator circuits this way. Instead, we use a generalized version of the classic *Nechiporuk method* [21], which was used for obtaining the current best lower bounds for branching programs and span programs stated above. However, it is not possible to apply Nechiporuk's method directly to estimate the number of gates of comparator circuits, because the partial computations corresponding to subcircuits over disjoint subsets of inputs may significantly overlap (see Remark 8 and Section 1.2 for more details.) Instead, we need to exploit some interesting structure of comparator circuits to enable a more general version of the method. For these reasons we believe that our above lower bound (and comparator circuits more generally) are particularly interesting.

1.1 Comparator Circuits and Bounded Fanout Computation

Let us first describe comparator circuits more formally. As stated above, a *comparator gate* computes the map $(x, y) \mapsto (x \wedge y, x \vee y)$ where $x, y \in \{0, 1\}$; that is, the gate outputs both the AND and the OR of its input bits. A *comparator circuit* is a Boolean circuit whose inputs are labeled by input literals (i.e. either variables x_i or their negations $\overline{x}_i$), and is composed entirely of comparator gates; the comparator circuit is said to be *monotone* if no input literal is negated. We emphasize here that every comparator gate in the circuit has exactly two inputs and two outputs, and the (Boolean) output of the circuit is obtained by fixing some designated output of a comparator gate in the circuit.

The definition of a comparator gate suggests a convenient method of visualizing such circuits, depicted in Figure 1. We draw m lines from left-to-right – we call these lines *wires*[2] – that are initially labeled with input literals or the Boolean constants $0, 1$. We connect wires together with *comparator gates*: each gate is drawn connecting a pair of wires; one of the inputs is drawn with a *circle* (representing the output of the $\wedge$ gate) and the other with an *arrow* (representing the output of the $\vee$ gate).

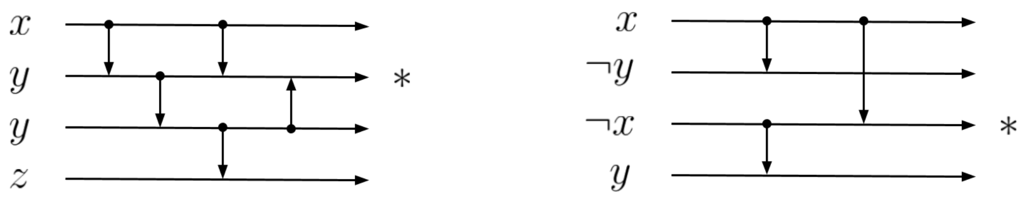

Figure 1 Examples of comparator circuits, with outputs designated. The left is a simple monotone comparator circuit, the right is a non-monotone comparator circuit for parity.

The most studied examples of comparator circuits in the literature are *sorting networks*, which are typically studied in a more general (i.e. non-Boolean) setting. A sorting network is a mathematical model of a sorting algorithm which is *input-oblivious*: the algorithm performs the same sequence of compare-and-swap operations on its input sequence of integers for all possible inputs. There are n wires traveling from left to right, each labeled with a distinct input variable, and a sequence of *comparator gates* connecting the wires. (Note that a comparator gate, intuitively, just *sorts* its inputs in order, with the larger input sent to the $\vee$ output and the smaller to the $\wedge$ output.) The goal is to construct the *fastest* network – where fastest can mean "shallowest", with the smallest depth connecting inputs to outputs, or with the fewest possible comparator gates – that sorts all n inputs. Shallow sorting networks have many applications in theory and practice, and thus have been extensively studied in theoretical computer science, with much effort spent on finding the shallowest possible networks (see [1, 2, 17, 9]). A restricted version of comparator circuits appeared implicitly in a paper by Graham [10]. His model is equivalent to *read once* comparator circuits, where each literal is used on at most one wire. Graham [10] showed that there exists a Boolean function on 11 variables that cannot be computed by this model, and it was proved by Robere [23] that the s, t- connectivity problem cannot be solved by read once monotone comparator circuits.

[2] We note that this differs from the standard usage of the word "wires" in circuit complexity, which is usually used to mean the edges in the underlying directed graph of the circuit.

Studying comparator circuits as a circuit model for general computation arguably began with the work of Subramanian [25], who showed that comparator circuits are a natural model for extending the study of *bounded fan-out* circuits beyond Boolean formulas. Subramanian considered different classes of circuits depending on whether or not the gates in the circuit could simulate a COPY gate: that is, the gate that takes some input x and outputs two copies of x. For example, when a gate can "fan-out" its output then we are allowing the gate to implicitly copy its output many times over. On the other hand, it is easy to see that a circuit constructed of comparator gates can *not* simulate a COPY gate, as the Hamming weight of the *output* of a comparator gate is always the same as the Hamming weight of the *input* of a comparator gate. Thus comparator circuits form an interesting intermediate class between Boolean formulas and Boolean circuits: they cannot create copies of intermediate computations, and so are apparently weaker than Boolean circuits; but, the ability to compute AND and OR simultaneously makes them apparently more powerful than Boolean formulas.

Let CC denote the class of languages computable by AC^0-uniform polynomial-size comparator circuits. This complexity class has an interesting characterization as the class of problems reducible to the *stable marriage* problem [25]. Closer to our purposes, Subramanian showed that if a circuit composed from gates chosen from a set S has "bounded fanout", in the sense that the gates from S cannot simulate a COPY gate, and all the gates in S compute monotone functions, then the corresponding circuit value problem for circuits with gates from S is either in NC or is CC-hard [25]. Despite their inability to copy, polynomial-size comparator circuits can compute everything in NL [20, 6] (which is conjectured to be strictly more powerful than polynomial-size Boolean formulas), and appear to be incomparable with NC [25]. Therefore, CC is a natural class to continue the study of bounded fan-out computation past NC^1, along with being a candidate for a class C satisfying $\mathsf{NL} \subset C \subset \mathsf{P}$ and $C \neq \mathsf{NC}$. The computational complexity of the class CC was recently analyzed by Cook, Filmus and Le [6] where, among other things, the question of uniformity and oracle separations with other classes was addressed.

However, a natural question left open by [6] is that of *lower bounds.* In the monotone case, since monotone comparator circuits are simulated by monotone circuits, exponential lower bounds are implied by the known lower bounds for monotone circuits (for instance, by Razborov [22]), and exponential separations between monotone comparator circuits and monotone circuits have been proved in [24]. Lower bounds for general Boolean circuits imply lower bounds for comparator circuits, but other than such implications, we are not aware of any previous nontrivial lower bounds for general (non-monotone) comparator circuits. In particular, before our work, no superlinear lower bounds have been obtained for comparator circuits computing an explicit Boolean function.

1.2 Techniques

To prove Theorem 1 we recruit a classic tool from non-monotone circuit lower bounds: the *Nechiporuk method* [21]. This method was invented by Nechiporuk to prove lower bounds on the size of deterministic branching programs and formulas, and is currently one of the few techniques capable of proving superlinear lower bounds against non-monotone computation. The idea of the method is as follows, in the setting of branching programs (the argument for formulas is identical, see e.g. [29]). Consider a branching program B computing a Boolean function f on n variables. First, observe that fixing the values outside of a subset $S \subseteq [n]$ of the variables to constants results in a branching program that computes a subfunction of the original function on the variables in the subset S. It follows that as we range over all substitutions to $[n] \setminus S$, the number of different branching programs obtained must be at least as large as the number of different subfunctions of f on S. Thus, to obtain a lower bound, take any partition of the input variables $[n] = Y_1 \cup Y_2 \cup \cdots \cup Y_n$ and let h_i be the number of nodes of the branching program B labeled with variables from the set Y_i. Recall that the size of a deterministic branching program is the number of its nodes, and each node is labeled by exactly one input variable. Thus, the size of the branching program B is $|B| = \sum_i h_i$ and, in turn, we can lower-bound each h_i by relating it to the number of subfunctions of f on the variables Y_i obtained by restricting $[n] \setminus Y_i$ in all possible ways. This yields lower bounds for functions with large number of different subfunctions.

The Nechiporuk method works well for branching programs as the sets of nodes of the program associated with disjoint subsets of inputs are disjoint. The situation is similar for Boolean formulas: each leaf of the formula is labeled by a variable (or negated variable), and again the sets of leaves associated with disjoint subsets of inputs are disjoint. In fact, this property (that the subcircuits corresponding to disjoint subsets of inputs are disjoint) is true for *all* prior applications of the method. For example, in the application to non-deterministic and parity branching programs [15], the *edges* of the program are now labeled by variables, and thus the sets of edges corresponding to disjoint subsets of variables are disjoint. Similarly, in span programs over $GF(2)$ [3], the rows are labeled by variables and so once again, subsets of rows corresponding to disjoint subset of variables are disjoint.

In principle, one could imagine generalizing this method to other circuit models, as long as the subcomputations corresponding to disjoint subsets of input variables do not overlap much. On the negative side, the method can provably *not* be applied to general Boolean circuits [28], which intuitively makes sense as circuits can copy intermediate computations as many times as they like.

So: what about comparator circuits? At first glance it seems like the method should not be applicable to comparator circuits either, since the circuits contain gates with fanout *two*, and this implies that the computation of the circuits on disjoint sets of variables can badly overlap, just like general Boolean circuits and unlike formulas, branching programs or span programs. Indeed, as we discuss in Remark 8, the set of gates of a comparator circuit contributing to the computation on a given subset of variables can potentially involve *all* the gates of the original circuit.

We can, however, overcome this problem. Our key observation is the following: the idea of Nechiporuk's method may be still applicable in situations where the subcomputations badly overlap, as long as we are able to bound the size of the computation by some (not too fast growing) function of the number of occurrences of input literals. This is simply because the *occurrences of input literals* (or in other words the *input queries*) made by the computation will always be disjoint over disjoint subsets of variables.

Let us elaborate further in the case of comparator circuits. In comparator circuits, the number of *wires* – which, we recall, refers to the number of distinct left-to-right lines in Figure 1 – corresponds exactly to the number of occurrences of input literals in the circuit. The fact that comparator circuits cannot simulate COPY gates suggests the following question. If F is a Boolean formula with ℓ leaves then it is easy to see that the number of internal gates s of F is *exactly* $s = \ell - 1$. In other words, the size of the formula F is *linearly* related to the number of its leaves, and this is a result of F having bounded fanout. Does a similar relation hold for comparator circuits between *gates* and *wires*?

First, observe that since each gate in the comparator circuit connects two wires, in order for each wire to be connected to the output gate we must have $\ell - 1 \leq s$, where s is the number of gates, just like in formulas. However, is it the case that $s = O(\ell)$? Or even $s = O(\text{poly}(\ell))$? Surprisingly, we are able to show that $s \leq \binom{\ell}{2}$ assuming that none of the gates in the circuit are “useless” in a specific technical sense and can be removed (cf. Section 3); furthermore, we show that this relationship is sharp. For the special case of read once comparator circuits, essentially this statement was proved implicitly by Graham [10].

▶ **Theorem 2.** *Let C be any comparator circuit with ℓ wires and s gates such that every gate in C is useful. Then $s \leq \ell(\ell-1)/2$.*

It is precisely this relationship between wires and gates which allows us to overcome the “overlapping subcomputations” issue in applying Nechiporuk’s method, and it seems to be a remarkable property of this model. As far as we know, our results are the first generalization of Nechiporuk’s method to such a scenario.

2 Definitions

For any positive integer m let $[m] := \{1, 2, \ldots, m\}$, and if n is an integer satisfying $m \leq n$ let $[m, n] := \{m, m+1, \ldots, n\}$. If S is a set then $\wp(S)$ denotes the power set of S.

Throughout we will be interested in Boolean functions $f : \{0,1\}^n \to \{0,1\}$. If $x \in \{0,1\}^n$ then x_i is the value of the ith bit in x. If $x, y \in \{0,1\}^n$ are binary strings then $x \leq y$ if $x_i \leq y_i$ for all $i \in [n]$. If $x \in \{0,1\}^n$ then $|x|$ is the Hamming weight of x.

A *comparator gate* is the function $C : \{0,1\}^2 \to \{0,1\}^2$ computing the map $(x, y) \mapsto (x \wedge y, x \vee y)$. It is natural to think of a comparator gate C as “sorting” the input (x, y), as the smaller input goes to the first coordinate and the larger input goes to the second. A *comparator circuit* is a circuit composed of comparator gates with arbitrary Boolean literals as input. For later convenience, we use the following, alternative definition of a comparator circuit. A comparator circuit C is defined by a tuple (W, G, I, o), where

- W is a set of elements called *wires*,
- $I : W \to \{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n, 0, 1\}$ is an initial labeling of each wire with an input literal or Boolean constant,
- $G \subseteq W^2$ is an ordered list of *comparator gates*,
- $o \in W$ is the designated output wire.

Each comparator gate $g = (i, j) \in G$ is specified by the pair of wires the gate connects: the AND output of the gate is interpreted as the first coordinate and the OR output of the gate is interpreted as the second coordinate. A comparator circuit computes a Boolean function in the natural way: given an assignment to the input literals, we evaluate the comparator gates one by one in order according to G, and then output the Boolean value labeled on the output wire, o. We define the *size* of the comparator circuit to be the number of gates in the circuit, and note that the number of wires in the circuit will also be an interesting complexity measure.

3 Wires vs. Gates in Comparator Circuits

▶ **Definition 3.** *Let C be a comparator circuit with m wires, let $g = (i, j)$ be a gate in C, and let $v_i(x), v_j(x)$ be the values of the wires i, j, respectively, on input x just before applying the gate g. Then g is* useful *if there is a pair of inputs x, y to C such that $(v_i(x), v_j(x)) = (1, 0)$ and $(v_i(y), v_j(y)) = (0, 1)$. If g is not useful then we say it is* useless.

Equivalently, if a gate g is useless then for every pair of inputs (x, y) either $(v_i(x), v_j(x)) = (v_i(y), v_j(y))$ or $v_i = v_j$ for one of x, y. The next proposition states that gates that are not useful can be removed without loss of generality.

▶ **Proposition 4.** *Let C be a comparator circuit with ℓ wires and s gates computing some Boolean function f. Then there exists a comparator circuit C' with (at most) ℓ wires and $s' \leq s$ gates computing f such that every gate in C' is useful. Moreover, the labels of the wires that are not removed remain the same in C' as in C.*

Proof. Let $g = (i, j)$ be any gate in C such that g is not useful. Let $v_i(x), v_j(x)$ be the values of the wires i, j before applying g on input x. Since g is not useful, then it follows that for every pair of inputs x, y to the circuit either $(v_i(x), v_j(x)) = (v_i(y), v_j(y))$ or one of x, y satisfies $v_i = v_j$.

Let $O \subseteq \{0,1\}^n$ be the set of inputs such that for all $x \in O$, exactly one of $v_i(x), v_j(x)$ is 1. Since g is useless, it follows that either $(v_i(x), v_j(x)) = (0, 1)$ for every x in O or $(v_i(x), v_j(x)) = (1, 0)$ for every x in O.

Assume that the outputs of g are ordered so that $g(0, 1) = (0, 1)$ and $g(1, 0) = (0, 1)$. The proof for the other type of gates is symmetric. Consider the following two cases.

Case 1: For all $x \in O$, $(v_i(x), v_j(x)) = (0, 1)$.
We claim that for all $x \in \{0,1\}^n$, applying g does not change the value of any wire, and so g can be removed without affecting the computation C. This is because for every $x \in \{0,1\}^n$, either $x \in O$ and so applying g does not change any value (since $g(0,1) = (0,1)$), or $x \notin O$, and by the definition of the set O we must have $v_i(x) = v_j(x)$, and so applying g will not change the values of the wires.

Case 2: For all $x \in O$, $(v_i(x), v_j(x)) = (1, 0)$.
By definition $g(1, 0) = (0, 1)$, and so we perform the following operation: remove g, and for each wire i, j find the *first* gate h using that wire as an input. If it is the same gate for both wires, then h is redundant, so remove it and continue searching forward along each wire. Otherwise, let $h_i = (k, \ell)$ where exactly one of k, ℓ is i and let $h_j = (k', \ell')$ where exactly one of k', ℓ' is j. Let the wire i feed into the gate h_j and the wire j feed into the gate h_i. Switching the wires this way will simulate the action of the gate g that we just eliminated.

Once you have found a non-redundant gate h for each of the two wires $\{i, j\}$ (or if no such gate exists), then stop.

To see that this operation does not affect the output of the circuit, consider the following two cases:

1. $v_i(x) = v_j(x)$. In this case, performing this operation will not affect the computation of the circuit as either of the h gates will receive the same input, regardless of which wire they are connected to.
2. $v_i(x) \neq v_j(x)$. In this case, $(v_i(x), v_j(x)) = (1, 0)$, and $g(1, 0) = (0, 1)$, so by removing g and switching the inputs of the h gates we ensure that in the new circuit the computation will be the same.

By proceeding from the inputs of the circuit to the outputs we can apply this operation to remove every useless gate in the circuit. In each case, the output of the circuit will not be affected.

Finally, notice that if a wire is not connected to any remaining gate in C', we may remove it, except if the wire is designated as the output of the circuit. The wires that are not removed, retain the same label as in the original circuit C, and the proof of the proposition is complete. ◀

Now we can prove the main theorem of this section, which we restate from the Introduction.

▶ **Theorem 2.** *Let C be any comparator circuit with ℓ wires and s gates such that every gate in C is useful. Then $s \leq \ell(\ell-1)/2$.*

Proof. For $j = 1, 2, \dots, s$ let g_j denote the jth gate of C, and let C_j be the subcircuit of C consisting of the first j gates of C. Let C_0 be the circuit C with all of its gates removed. We say that two inputs $x, y \in \{0,1\}^n$ are *useful with respect to the pair* (i,j) if $w_i(x) = 1, w_i(y) = 0$ and $w_j(x) = 0, w_j(y) = 1$, where w_i, w_j are the outputs of wires i and j of the comparator circuit C. If there exist two inputs $x, y \in \{0,1\}^n$ that are useful with respect to (i,j) then we say that (i,j) is a *useful pair.*

Let U_j denote the collection of all useful pairs in the circuit C_j. We show that the size of U_j decreases by at least one after applying each gate: that is $|U_j| \leq |U_{j-1}| - 1$ for all j. Since $|U_0| \leq \binom{ell}{2}$, it follows that the number of gates in C is at most $\binom{\ell}{2}$.

Let $1 \leq j \leq s$ be an integer, and suppose the gate g_j connects two wires (a,b). Since g_j is useful we have that (a,b) is a useful pair in U_{j-1}. It is also clear that $(a,b) \notin U_j$, as applying the gate g_j removes all useful inputs with respect to (a,b). However, we are not finished, as applying the gate g_j could have introduced a new useful pair (c,d): so, $(c,d) \in U_j$ and $(c,d) \notin U_{j-1}$. The proof will be complete once we prove the following claim, which states that if such a new useful pair is introduced, then there must exist another distinct useful pair that was removed.

▷ Claim 5. Suppose there exists a useful pair $(c,d) \in U_j$ such that $(c,d) \notin U_{j-1}$. Then there is another useful pair (α,β), uniquely associated with (c,d), such that $(\alpha,\beta) \neq (a,b)$, $(\alpha,\beta) \in U_{j-1}$, and $(\alpha,\beta) \notin U_j$.

Proof of Claim. It is clear that while $(c,d) \neq (a,b)$, it cannot be the case that $c, d \notin \{a,b\}$: that is, at least one of c or d must be equal to a or b. This is because applying the gate g_j only modifies the outputs of wires a and b, and so any new useful pair must include one of these modified outputs.

First assume that $c = a$ (and we note that the other cases ($c = b, d = a, d = b$) follow by nearly identical proofs). It follows that $(a,d) \in U_j$ and $(a,d) \notin U_{j-1}$. We show that this implies that $(b,d) \in U_{j-1}$ and $(b,d) \notin U_j$. For any wire t let v_t denote the output of the tth wire *before* applying the gate g_j, and let w_t denote the output of the tth wire *after* applying the gate g_j. Since (a,d) is a useful pair it follows that there exists a pair of inputs $x, y \in \{0,1\}^n$ such that $w_a(x) = 1, w_d(x) = 0$ and $w_a(y) = 0, w_d(y) = 1$. Now, since $(a,d) \notin U_{j-1}$ we can deduce the values of v_a, v_b, and v_d on inputs x and y. It is obvious that $v_d(x) = w_d(x)$ and $v_d(y) = w_d(y)$ as the gate g_j is not connected to the wire d. We can also conclude that $v_a(x) = v_b(x) = 1$, as the gate g_j connects a to b and $w_a(x) = 1$ (for if $v_b(x) = 0$, then $w_a(x) \neq 1$, as the 1 would have been moved to the $\vee$ output of g_j). Finally, since $(a,d) \notin U_{j-1}$ we know that $v_a(y) = 1$ and $v_b(y) = 0$. For if $v_a(y) = 0$, it would follow that (a,d) is a useful pair in U_{j-1} (a contradiction). Thus $v_a(y) = 1$, and since $w_a(y) = 0$, the only possibility is that the gate g_j moved a 1 from wire a to wire b on input y. We now record the values of v_a, v_b, v_d and w_a, w_b, w_d on inputs x and y:

	x	y
v_a	1	1
v_b	1	0
v_d	0	1
w_a	1	0
w_b	1	1
w_d	0	1

By examining the table it is easy to see that (b,d) is a useful pair in U_{j-1}, as $v_b(x) = 1, v_d(x) = 0$ and $v_b(y) = 0, v_d(y) = 1$. We show that $(b,d) \notin U_j$, proving the claim in this case.

To see this, suppose that $(b,d) \in U_j$. Then there must exist an input z such that $w_b(z) = 0, w_d(z) = 1$, on which we can similarly deduce the values of v_a, v_b, v_d. Clearly $v_d(z) = 1$ since the gate g_j connects the two wires (a,b). Since $w_b(z) = 0$, it must be that $v_a(z) = v_b(z) = 0$. However, this means that $v_a(z) = 0, v_d(z) = 1$, and from the table we can see that $v_a(x) = 1, v_d(x) = 0$. This means that (a,d) is a useful pair in U_{j-1}, a contradiction! ◁

We make a final remark on the uniqueness property. We technically have four cases to prove here, as the new useful pair must be in one of the following forms:

1. (a,d)
2. (b,d)
3. (c,a)
4. (c,b)

In the proof above we showed that if the new useful pair is of the form (a,d), then this implies that $(b,d) \in U_{j-1}$ and $(b,d) \notin U_j$. In the other three cases (which proceed by identical proofs), other uniquely associated pairs are introduced. In general, if $\alpha \in \{a,b\}$ and $\beta \in \{a,b\}$, $\beta \neq \alpha$, and the new useful pair is of the form (α,d) then the same proof shows that (β,d) is in U_{j-1} and not in U_j. Similarly, if the new useful pair is of the form (c,α) then a similar proof shows that (c,β) is in U_{j-1} and not in U_j. These facts together prove uniqueness. ◀

To see that the upper bound given in the previous theorem is sharp, consider the following comparator circuit C_n which sorts its n inputs via the "bubblesort" method: the circuit incrementally bubbles the largest value from the top wire down as far as it can go, and then the second wire, and so on. This circuit has n wires and $\binom{n}{2}$ gates, and it is not hard to see that every gate is useful.

4 Lower Bound for Element Distinctness

Let $n = 2m \log m$, and recall the definition of the Element Distinctness function ED_n: it takes $n = 2m \log m$ input bits divided into m blocks of $2 \log m$ bits each, interpreted as m integers in the range $\{1, \ldots, m^2\}$, and decides whether all m numbers are distinct.

In this section we prove our main lower bound, restated here for convenience:

▶ **Theorem 1.** *The size of any comparator circuit computing the n-bit Element Distinctness function ED_n is at least $\Omega((n/\log n)^{3/2})$.*

We note that the lower bound holds for any function with a similar number of subfunctions; for instance, the *Indirect Storage Access function* [29].

We will need the following Lemma. Recall that in a comparator circuit, each of the wires is labeled with either a constant $0, 1$, some input variable or its negation.

▶ **Lemma 6.** *Let $\ell \geq 1$ be an integer. For any fixed labeling of ℓ wires, the number of different Boolean functions that can be computed by comparator circuits with ℓ wires with the given labeling is at most ℓ^{ℓ^2}.*

Proof. By Proposition 4, every comparator circuit C with ℓ wires is equivalent to another comparator circuit C' with (at most) ℓ wires that has no useless gates. By Theorem 2 the number of gates of C' is at most $\binom{\ell}{2}$. Recall also that C' keeps the same labeling of the wires (that are not removed) as C; and if a wire is not connected to any remaining gate in C', we may remove it, except if the wire is designated as the output of the circuit.

To prove the lemma it is enough to estimate the number of different comparator circuits with at most ℓ wires of a given labeling and at most $\binom{\ell}{2}$ gates. For each gate, there are at most $\binom{\ell}{2}$ choices for the pair of wires it takes as inputs, and two choices for the ordering of the $\wedge$ output and $\vee$ output of the gate. In addition, we have at most ℓ choices to designate one of the wires as the output. Thus the number of possible such comparator circuits is at most $\ell \cdot (2\binom{\ell}{2})^{\binom{\ell}{2}} \leq (\ell)^{(\ell^2)}$. ◀

▶ Remark 7. We can also bound the number of different Boolean functions on n variables that are computed by comparator circuits with s gates by $(2(n+1)s)^{2s}$ using a similar counting argument. Note that here we do not assume a fixed labeling, and we use that the number of wires is at most the number of gates, assuming that each wire is connected to the output gate.

If the subsets of gates used in subcircuits over disjoint subsets of inputs would not overlap much, then a straightforward application of Nechiporuk's method seemingly would yield nearly quadratic lower bounds using this counting argument. However, as we discuss in some more details in Remark 8, the subsets of gates can badly overlap. Thus, even though a counting argument in terms of gates is available, Nechiporuk's argument is not applicable directly to the gates of comparator circuits.

Note that we could have also stated a similar bound for comparator circuits with ℓ wires and n input variables, without considering a fixed labeling of the wires. We find the current version of Lemma 6 more convenient for our purposes. We are now ready to prove Theorem 1.

Proof of Theorem 1. We prove the stronger statement, that the number of wires of any comparator circuit computing ED_n is at least $\Omega((n/\log n)^{3/2})$. Recall that the size of a comparator circuit is the number of its gates, and the number of gates in a comparator circuit with w wires, where each wire is connected to the output gate, is at least $w-1$. Thus, a lower bound on the number of wires implies lower bounds on the size of the comparator circuit.

Partition the $n = 2m\log m$ input variables into m groups of $2\log m$ variables each, such that the variables in the i-th group represent the i-th integer in the input. For $i = 1, \ldots, m$ let S_i be the set of variables in the i-th group. Let N_i denote the number of different subfunctions over the variables in S_i that can be obtained by fixing all variables outside S_i to constants. It is known (see e.g. [14]) that $N_i = 2^{\Omega(n)}$ for each $i = 1, \ldots, m$.

Let C be a comparator circuit computing ED_n with w wires. Let w_i denote the number of wires of C labeled by a variable from the i-th group. Then $w = \sum_{i=1}^{m} w_i$.

For a given $i \in [m]$, consider a fixed assignment α of constants 0 or 1 to all variables outside of S_i. Consider the resulting comparator circuit $C_{i,\alpha}$ over the variables in S_i. Applying Proposition 4 to $C_{i,\alpha}$ we can obtain a comparator circuit $C'_{i,\alpha}$ over variables from S_i with no useless gates. Notice that if a wire is labeled by constant 0 or 1 then any gate directly using this wire must be useless (in the formal sense of Definition 3). Note also that after removing all useless gates, wires with constant label are not connected to any remaining gates. Thus, they can be removed, except when designated as the output wire. Note however, that if a wire with constant label that is not connected to any gate is designated as the output wire, then the function computed is constant 1 or 0. Thus, unless the function computed by $C'_{i,\alpha}$ is constant, all wires in $C'_{i,\alpha}$ are labeled by variables or their negation from S_i, regardless of the particular assignment α.

This fact has two important consequences for us. First, it means that the number of wires of $C'_{i,\alpha}$ is at most w_i. Second, it means that for given i, unless the function computed by $C'_{i,\alpha}$ is constant, the wires of $C'_{i,\alpha}$ have the same labels (by variables or negated variables from S_i) regardless of the particular assignment α. (To see this, recall that in Proposition 4 the labels of the wires that are not removed remain the same as in the original circuit.)

This allows us to conclude using Lemma 6 that $N_i \leq 2 + w_i^{(w_i^2)}$ for $i = 1, \ldots, m$. Thus, we have for $i = 1, \ldots, m$ that

$$w_i^2 \geq \frac{\log(N_i - 2)}{\log w_i} \geq \frac{\Omega(n)}{\log w_i} .$$

Note that if $\log w_i > \frac{1}{2} \log n$ then $w_i \geq \sqrt{n}$. On the other hand, if $\log w_i \leq \frac{1}{2} \log n$ then we get $w_i^2 \geq \frac{\Omega(n)}{\log n}$. Thus, for $i = 1, \ldots, m$ we have $w_i \geq \Omega(\frac{\sqrt{n}}{\sqrt{\log n}})$, which yields $w \geq \Omega((n/\log n)^{3/2})$. ◀

▶ Remark 8. It is crucial in the above argument that the set of *wires* used by the subcircuits $C'_{i,\alpha}$ is the same for fixed i regardless of the assignment α, and that these sets do not overlap for different values of i. One could try to consider a similar argument directly for gates instead of wires. For instance, one could define $G_{i,\alpha}$ as the set of gates participating in the circuit $C'_{i,\alpha}$, and consider $G_i = \cup G_{i,\alpha}$. But for different assignments α, the circuits $C'_{i,\alpha}$ may retain different gates of the original circuit, and the sets G_i may badly overlap. In particular, for some values of i, G_i may contain all gates of the original circuit.

5 Conclusion and Future Work

In this paper we have proved the first superlinear lower bound on the size of comparator circuits computing an explicit Boolean function. As we have remarked above, we actually prove a superlinear lower bound on the number of *wires*, or equivalently, *input queries*, of any comparator circuit for ED_n, which is stronger than a lower bound on the number of gates. Furthermore, by our Theorem 2, there is at most a quadratic separation between the number of wires and the number of gates in any minimal comparator circuit. A natural problem is to try and prove a lower bound on the number of gates directly. However, as we discuss above in Remark 8 this would require a different technique.

We remark that it seems quite difficult to apply restriction techniques to obtain *wire* lower bounds for comparator circuits. This is for a simple reason: observe that restricting one input to a single comparator will restrict exactly one output of the gate and re-wire the other input to the other output. This implies that if we have a comparator circuit with m wires, and we restrict values to t of them, then we are left with a new comparator circuit with exactly $m - t$ unrestricted wires after propagating this rewiring process. Note that some wires could possibly be removed if they are "separated" from the output gate, but, if the topology of the circuit is highly connected (e.g. is an expander) then we should expect this to be very unlikely.

Finally, we remark on a second natural open problem. As we discuss in the introduction, the key structural property of comparator circuits that enabled us to apply Nechiporuk's method to comparator circuits is Theorem 2, relating the number of wires to the number of gates. The crucial intuition in the proof of this Theorem is that comparator circuits *cannot* copy intermediate computations. There is a rich structure of circuit classes extending comparator circuits which cannot copy intermediate computations, as explored by Subramanian [25]. Can one extend any of our results to these more general classes?

References

1 Miklós Ajtai, János Komlós, and Endre Szemerédi. An O(n log n) Sorting Network. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 1–9, 1983. doi:10.1145/800061.808726.

2 Kenneth E. Batcher. Sorting Networks and Their Applications. In *American Federation of Information Processing Societies: AFIPS Conference Proceedings: 1968 Spring Joint Computer Conference, Atlantic City, NJ, USA, 30 April - 2 May 1968*, pages 307–314, 1968. doi:10.1145/1468075.1468121.

3 Amos Beimel, Anna Gál, and Mike Paterson. Lower Bounds for Monotone Span Programs. *Computational Complexity*, 6(1):29–45, 1997. doi:10.1007/BF01202040.

4 Andrej Bogdanov. Small Bias Requires Large Formulas. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 22:1–22:12, 2018. doi:10.4230/LIPIcs.ICALP.2018.22.

5 Ruiwen Chen, Valentine Kabanets, Antonina Kolokolova, Ronen Shaltiel, and David Zuckerman. Mining Circuit Lower Bound Proofs for Meta-Algorithms. *Computational Complexity*, 24(2):333–392, 2015. doi:10.1007/s00037-015-0100-0.

6 Stephen A. Cook, Yuval Filmus, and Dai Tri Man Le. The complexity of the comparator circuit value problem. *TOCT*, 6(4):15:1–15:44, 2014. doi:10.1145/2635822.

7 Irit Dinur and Or Meir. Toward the KRW Composition Conjecture: Cubic Formula Lower Bounds via Communication Complexity. *Computational Complexity*, 27(3):375–462, 2018. doi:10.1007/s00037-017-0159-x.

8 Anna Gál, Avishay Tal, and Adrian Trejo Nuñez. Cubic Formula Size Lower Bounds Based on Compositions with Majority. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, pages 35:1–35:13, 2019. doi:10.4230/LIPIcs.ITCS.2019.35.

9 Michael T. Goodrich. Zig-zag sort: a simple deterministic data-oblivious sorting algorithm running in O(n log n) time. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 684–693, 2014. doi:10.1145/2591796.2591830.

10 RL Graham. A mathematical study of a model of magnetic domain interactions. *Bell System Technical Journal*, 49(8):1627–1644, 1970.

11 Johan Håstad. Almost Optimal Lower Bounds for Small Depth Circuits. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 6–20, 1986. doi:10.1145/12130.12132.

12 Johan Håstad. The Shrinkage Exponent of de Morgan Formulas is 2. *SIAM J. Comput.*, 27(1):48–64, 1998. doi:10.1137/S0097539794261556.

13 Kazuo Iwama and Hiroki Morizumi. An Explicit Lower Bound of 5n - o(n) for Boolean Circuits. In *Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002, Warsaw, Poland, August 26-30, 2002, Proceedings*, pages 353–364, 2002. doi:10.1007/3-540-45687-2_29.

14 Stasys Jukna. *Boolean Function Complexity - Advances and Frontiers*, volume 27 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-24508-4.

15 Mauricio Karchmer and Avi Wigderson. On Span Programs. In *Proceedings of the Eigth Annual Structure in Complexity Theory Conference, San Diego, CA, USA, May 18-21, 1993*, pages 102–111, 1993. doi:10.1109/SCT.1993.336536.

16 Valeriy M Khrapchenko. Method of determining lower bounds for the complexity of P-schemes. *Mathematical Notes*, 10(1):474–479, 1971.

17 Donald Ervin Knuth. *The art of computer programming, , Volume III, 2nd Edition.* Addison-Wesley, 1998. URL: http://www.worldcat.org/oclc/312994415.

18 Ilan Komargodski and Ran Raz. Average-case lower bounds for formula size. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 171–180, 2013. doi:10.1145/2488608.2488630.

19 Ilan Komargodski, Ran Raz, and Avishay Tal. Improved Average-Case Lower Bounds for DeMorgan Formula Size. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 588–597, 2013. doi:10.1109/FOCS.2013.69.

20 Ernst W. Mayr and Ashok Subramanian. The Complexity of Circuit Value and Network Stability. *J. Comput. Syst. Sci.*, 44(2):302–323, 1992. doi:10.1016/0022-0000(92)90024-D.

21 Edward I Nechiporuk. A Boolean function. *Engl. transl. in Sov. Phys. Dokl.*, 10:591–593, 1966.

22 Alexander A Razborov. Lower bounds for the monotone complexity of some Boolean functions. In *Soviet Math. Dokl.*, volume 31, pages 354–357, 1985.

23 Robert Robere. Notes on Comparator Circuits. Unpublished manuscript, 2014.

24 Robert Robere, Toniann Pitassi, Benjamin Rossman, and Stephen A. Cook. Exponential Lower Bounds for Monotone Span Programs. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 406–415, 2016. doi:10.1109/FOCS.2016.51.

25 Ashok Subramanian. *The computational complexity of the circuit value and network stability problems*. PhD thesis, Stanford University, 1990.

26 Avishay Tal. Shrinkage of De Morgan Formulae by Spectral Techniques. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 551–560, 2014. doi:10.1109/FOCS.2014.65.

27 Avishay Tal. Formula lower bounds via the quantum method. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1256–1268, 2017. doi:10.1145/3055399.3055472.

28 Dietmar Uhlig. Boolean functions with a large number of subfunctions and small complexity and depth. In *International Symposium on Fundamentals of Computation Theory*, pages 395–404. Springer, 1991.

29 Ingo Wegener. *The complexity of Boolean functions*. Wiley-Teubner, 1987. URL: http://ls2-www.cs.uni-dortmund.de/monographs/bluebook/.